A Framework for Sharing Voluminous Content in P2P Systems

Dimitrios Tsoumakos
Department of Computer Science
University of Maryland
College Park, MD 20742, U.S.A.
dtsouma@cs.umd.edu

Nick Roussopoulos
Department of Computer Science
University of Maryland
College Park, MD 20742, U.S.A.
nick@cs.umd.edu

***Abstract*— File-sharing applications remain today the most representative and popular realization of the Peer-to-Peer paradigm. Large objects receive an increasing amount of interest in such systems. In this paper, we identify several challenges related to sharing voluminous content such as movies, OS distributions, games, etc, in unstructured Peer-to-Peer networks. We then describe our scheme which adaptively *expands* or *contracts* system resources in order to improve the sharing process and achieve a fair load distribution among the providers.**

Keywords: Peer-to-Peer, Distributed Algorithms, Internet and Cooperative Applications, Load-Balancing

I. Introduction

Peer-to-Peer (hence P2P) computing represents the notion of sharing resources available at the edges of the Internet [1]. Its success can still be largely attributed to file-sharing applications (e.g., [2]–[4]), which enable users worldwide to exchange popular content. While the scientific community has embraced this paradigm with a variety of applications, it is still file-sharing that provides P2P with its majority of users.

We are currently in the middle of a large debate (with legal implications as well) concerning the lawfulness of sharing copyrighted material through P2P applications. While music undeniably receives the largest share of user interest, a recent study [5] reports that there has been an increase in the demand for large multimedia files (movies) among P2P users. There also exist other types of large objects for which communities would be greatly interested, for example new OS distributions, educational/commercial software, games, high quality satellite images and maps, etc.

Obviously, sharing large files is very different from sharing objects with considerably smaller sizes (less than a few megabytes). The inherent difficulties arise mainly from the size of the content, its replication inside the network and the dynamic nature of the environment. This is the case for most of the popular P2P applications today which operate on *unstructured* networks. In these systems, document location and replication is user-controlled, peers connect in an ad-hoc fashion and obtain only local knowledge. Moreover, little or no guarantees regarding the basic operations are offered.

In this work, we first describe this problem and the specific difficulties entailed. We then present our framework to achieve efficient voluminous content sharing. The core of our protocol is the “Expand-Contract” scheme for adaptively increasing or decreasing the object replication ratio and achieving an equal load distribution in a completely decentralized manner. During this process, we also identify several of the emerging research challenges we currently pursue.

II. The Problem of Sharing Large Objects

A. Research Challenges and Related Work

In this section we discuss the inherent difficulties that arise in a distributed content-sharing environment, when efficient sharing of very large objects is required. Typical network node configurations now include more than a GByte of main memory, 2+ GHz of processor speed and 200+ GBytes of secondary storage, which strongly imply that local space and processing are no longer the primary concerns. Experience shows that users face serious problems when trying to retrieve large files. In some cases, very few relevant nodes are willing to share, and those that do are often overwhelmed by the number of requests they receive, making them reluctant to keep sharing.

First, it is the size of the objects itself. Given the highly dynamic nature of file-sharing networks, it is obvious that the waste in network and node resources resulting from failing transactions (due to peer departures or network failures) is enormous. Not only have the two communicating peers wasted bandwidth and time, but they have also possibly denied their services to other peers in the mean time, used up link capacity, etc.

File fragmentation provides a reliable solution for this problem. Nevertheless, it also shifts our attention to how we should fragment each file, what to retrieve and from where. In general, the following two approaches exist:

1) Files comprise of equal size chunks and are individually indexed
2) Peers dynamically decide the portion that is retrieved from each source peer

The first approach is utilized by Overnet [6], BitTorrent [7] and Slurpie [8]. Each file is divided into a number of standard-size fragments (9500KB, 256KB, 256KB for those systems respectively). A peer may then download different fragments from various sources. Upon completion, each fragment becomes available for sharing with other nodes. This approach has some plausible characteristics, mainly the fact that peers can immediately serve[1] fragments they have retrieved. Of course, this means that each chunk must be identified as an individual entity now. The interesting questions here relate to the choice of the chunk sizes, as well as the order of their retrieval. Obviously, the smaller the size, the less bandwidth will be wasted in case of failure, but more transactions will be needed and more objects to be indexed.

The second approach [9] (or modifications of it [10]) is currently used by other P2P applications (e.g., Morpheus [11]). A requester contacts many source peers and retrieves small portions of the file from each of them. When each small chunk is retrieved, more is asked from that specific source. This has the effect that the number of the small portions retrieved is always proportional to the quality of each connection, even if it varies with time. This approach is beneficial only when the size of the file is in the order of at least a few hundred kilobytes. Given very large files (which is the case here), it actually behaves similarly to the previous approach, but has the disadvantage of requiring all contacted peers to obtain the whole document.

There also exist several schemes (e.g. [12], [13]) which allow for increased robustness in reconstructing a file by receiving a few extra parts of it. These methods are most beneficial for medium–small size files.

[1]The words serve and server for object *i* are used in this work to indicate peers that maintain *i* and are willing to share it with others (depending on their individual capacities)

An equally important issue is the availability of resources inside the system. We generally expect voluminous content to be more sparsely replicated than audio or image files for example. Combined with their large size (and thus increased amount of download time), this has the effect of overloading most of the servers. In turn, this reduces the availability of the shared files, as servers refuse connections, while download performance degrades for all involved peers. In general, it is interesting to investigate different means of system response to increased workload and overloaded servers. In Slurpie and BitTorrent, interested peers have to contact a central service in order to learn about other interested peers and then collaborate with them, forming a mesh and communicating through it. The Overnet protocol specifies that peers cooperate with four other downloaders of the same file (no details as to how they are found and maintained) that have high upload speeds.

Our approach mostly focuses on providing an adaptive solution to the problem of availability together with minimizing the instances of server overloads or serious service degradation. We intend for our system to “expand” and “contract” its resources according to the workload as perceived locally and balance content sharing in overloaded areas. Moreover, this will be done in a completely decentralized manner, with minimal communication overhead and using absolutely affordable memory space per node.

B. Our Framework

We assume a *pure* Peer-to-Peer model, with no imposed hierarchy over the set of participating peers. All of them may serve and request various objects. Each peer locally maintains its own collection of objects, as well as a local view of the system. Ignoring physical connectivity and topology from our talk, we generally expect peers to be aware of their one-hop neighbors in the overlay, while maintaining any other protocol-specific information (e.g., search indices, routing tables, etc). The system is expected to exhibit a dynamic behavior, with peers entering and leaving at will and also inserting/removing objects from their repositories. The overlay structure will also be affected, since nodes are not guaranteed to connect to the same neighbors each time.

This is a general model for resource-sharing in a completely decentralized, infra-structureless P2P network. We now extend it with details specific to the problem

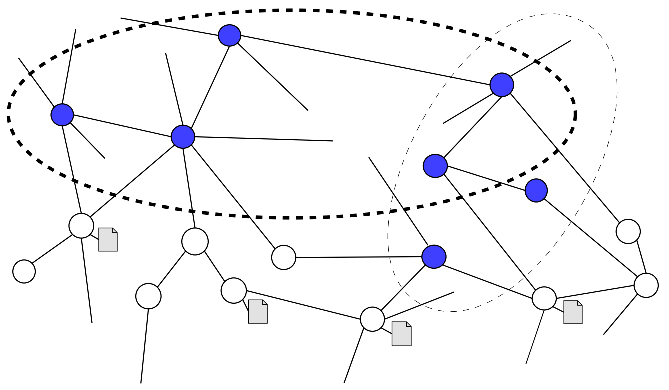

Fig. 1. Part of the overlay network of our model. Shaded peers inside the bold dotted line represent $\mathcal{M}_i$, while those inside the thin dotted eclipse represent $\mathcal{M}_j$. Other peers (those with a file connected to them) also serve objects i or j

in hand (see Figure 1). For each object i, there exists a set of peers called the *server set* $\mathcal{S}_i = \{s_{i_1}, s_{i_2}, \ldots, s_{i_k}\}$ that serve the specific object. These are the nodes that, at a given time, are online, store object i and are willing to share it. A subset of $\mathcal{S}_i$, the *mirror set* $\mathcal{M}_i \subseteq \mathcal{S}_i$ (the shaded peers) represents the set of peers that, if online, *always* serve i. This does not imply that all peers in $\mathcal{M}_i$ will always be online, their connectivity in the overlay will remain the same, or that they will never refuse connections. But we can assume, without loss of generality, that these nodes will be mostly available. Our assumption is not unrealistic: Imagine that these servers can represent mirror sites/authority nodes that provide with up-to-date content. Nevertheless, they are not guaranteed to be always on-line, nor do they provide similar services. Apart from the mirror set, other peers that already host or have recently retrieved an object can serve requests for it (nodes with files attached to them in Figure 1). A server set comprises of these nodes plus the corresponding mirror set.

Naturally, peers may belong to server or mirror sets for multiple objects. Moreover, they can make requests and retrieve other objects of interest from the network. While this is a symmetric environment, it is clear that nodes will exhibit different sharing abilities. This is simply the effect that is produced by a number of different conditions:

- Varying storage and CPU capabilities,
- Popularity of stored objects,
- System workload,
- Overlay connectivity for each peer,
- Connection speeds, etc

Some of these factors remain more or less static over time (e.g., processing power or the maximum available bandwidth of a host), while others change dynamically. Whichever the case, it is safe to assume that each peer in such a system will impose a limit on the services it can provide to other peers. This is something that is already utilized by several file-sharing applications (e.g., Kazaa [4]), FTP servers, etc. There exist a variety of metrics that can be used to realize those limits, for example the maximum number of concurrent connections, upload bandwidth, number of shared files, rate of received requests, etc.

Finally, we can assume (something that can be verified by experience) that some peers prefer to keep already downloaded files off the system, or even choose to disconnect than continuously share them. We expect that a certain level of user cooperation is necessary in order to achieve application-specific goals. For example, many current applications require that online peers serve all previously downloaded content, while others increase peers' priorities the more they share. In our approach, we assume a similar level of cooperation, except that we do not impose it until necessary.

Given this general framework, our goal is to design and implement a system that will exhibit the following characteristics:

1) Efficient sharing of large objects in terms of response times, load distribution at peers and adaptation to dynamic workloads and network conditions
2) Bandwidth-efficiency and scalability

III. Our "Expand-Contract" Technique

Our system design provides with two basic operations: *Expansion* and *Contraction*. These are shown pictorially in Figures 2 and 3 respectively. Our main goal is to provide a completely decentralized mechanism through which the system will adaptively expand its replica size when demand is increased and will shrink when demand will fall.

Let us now discuss why the system would benefit from those two operations. When parts of the server set $\mathcal{S}_i$ receive too many requests, the following may occur: Clients' connections get refused, while servers receive an increasing amount of requests and their performance deteriorates. Both groups would benefit from an increase in the number of replicas available, especially if those replicas were placed inside the areas of high demand in the overlay. Conversely, consider that one or more subsets of $\mathcal{S}_i$ have recently received very few requests for object i. This practically means that a potentially large amount of their storage space is under-utilized.

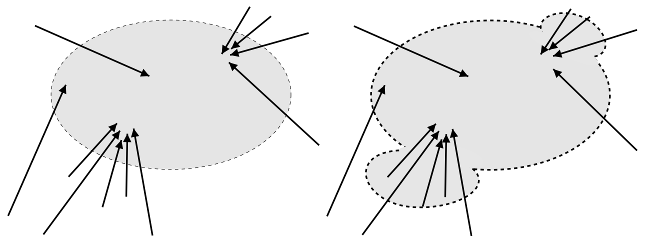

Fig. 2. Example of *Expansion*

Algorithm 1 *Expand*

```
1: if Server-Node x reaches its limit then
2:     $i \leftarrow ChooseObject()$;
3:     $P \leftarrow FindPossibleServers(i)$; $\{P \cap \mathcal{S}_i = \emptyset\}$
4:     send to $Y \subseteq P$: $Activate(Y, i)$;
5: end if
```

They could remove the document to free up space or replace it with another document of high demand. We have to stress here the point that the system will not force any peer to store or serve an object until this becomes necessary; Only peers that have already retrieved the document, or have available storage (depending on what replication strategy we use) can play that role.

Figure 2 shows an example of our system expanding in response to increased demand for a specific object. On the left we see some initial server set (gray oval) and the demand for *i* (arrows from various parts of the network). Servers in two areas are overloaded with requests, thus forcing extra replicas in those two areas to be activated for sharing. $\mathcal{S}_i$ expands, as we see on the right part of the picture, in response to the demand and replication status for object *i*. Similarly, Figure 3 shows that the two white areas of the server set (the area inside the dotted line) do not receive requests for object *i*. This leads to the contraction of $\mathcal{S}_i$ which is now the gray oval on the right part of the figure. Algorithms 1 and 2 present a high-level description of how these two methods would work.

Note here that we do not differentiate between a document chunk and a whole document. Lookups and downloads can be issued for either type. Assuming a set chunk size, requesters are responsible for assembling all the pieces. The order by which chunks will be requested is a problem of its own as described in [7]. Therefore, our method can be also applied for sharing objects of smaller sizes.

Vital to the success of our scheme are the following points:

1) Minimization of communication inside $\mathcal{S}_i$

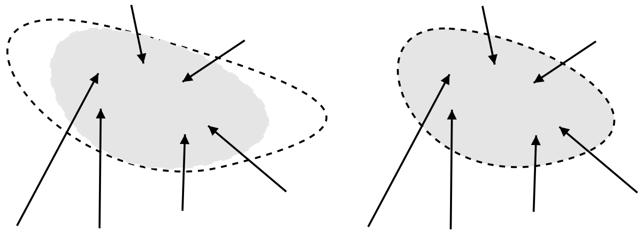

Fig. 3. Example of *Contraction*

Algorithm 2 *Contract*

```
1: if (Server-Node x is under-utilized) or
   (x receives $Activate(x, j)$) then
2:     $i \leftarrow ChooseObject()$;
3:     $DeactivateReplica(i)$;
4:     if (x received an $Activate(x, j)$) then
5:         $ActivateReplica(j)$;
6:     end if
7: end if
```

2) Mechanism to identify when an object is “hot” (many requests/pending downloads) or “cold”
3) Mechanism to locate and activate replicas inside high-demand areas
4) Mechanism to decide on the object replacement policy in peer storage space

For the first point, we believe that each peer can independently choose when to initiate an expansion or when to deactivate replica(s) of an object. Therefore, there is no need for any message exchange between peers inside $\mathcal{S}_i$.

The conditions of line 1 in Algorithms 1 and 2 describe when *Expand* or *Contract* will be initiated. We present the high-level behavior of our system using Figure 4: In normal mode, nodes can adequately serve requests and also retrieve objects. As load increases, some reach their self-imposed limits and invoke the *Expand* process (either in conjunction with *Contract* or independently). *Expand* aims at bringing the node status back to normal and/or balance the demand for a specific object inside an area. Normal operation and load balancing for an object will not be necessarily achieved simultaneously. Consider, for example, that a peer initiating *Expand* may receive requests for multiple objects. Expanding in respect to one of them will probably lower its load, but will not necessarily bring its level back to normal.

Similarly, when peers are under-utilized, they invoke *Contract* which frees idle resources from nodes and

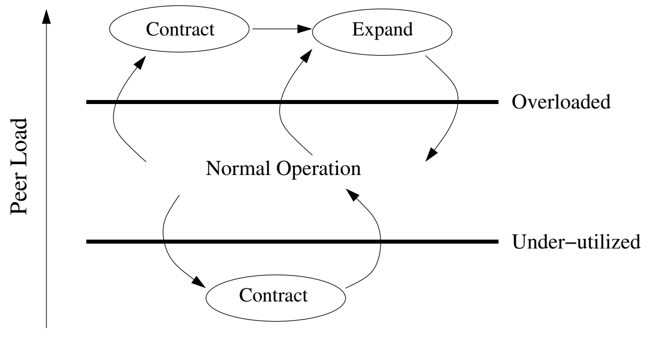

Fig. 4. State transitions in our system

increases the utilization of the servers. *Contract* will also be invoked when a peer is called to join $\mathcal{S}_i$ but cannot do so without exceeding its limits (e.g., maximum sharing capacity). Note that peers can still choose to reject a certain action, for example refuse to deactivate an object in order to serve a new one.

For the second point, there are various metrics that can be used to achieve that categorization, for example the number (Req_i) or rate of requests (λ_i) for object i observed in a time window. Function *ChooseObject* decides at each point which stored or requested object is the most beneficial in order to be replicated or removed. Therefore, the second and fourth points above should be jointly considered. One of our goals is to achieve a system behavior that resembles the buffer management techniques in databases: Viewing the P2P network as a large buffer, we want to decide (in a distributed manner) which object(s) should be kept in the buffer (become share-enabled) and which should be replaced (or put to secondary/tertiary storage), given a dynamic workload.

We assume that peers locate various servers by sending out a query each time. This is done mainly because peers want to get the most up-to-date information about replica locations in their area. The *APS* [14] algorithm has proved to be an adaptive, bandwidth-efficient scheme which also provides for robust behavior in dynamic environments [15]. It can be used to locate multiple copies (or fragments) of a requested object by deploying probabilistically directed walkers. *APS* utilizes an adaptive indexing scheme through which peers keep a relative probability for each of their neighbors. These index values are refined as more searches take place, enabling the network to build a useful soft-state. In Figure 5, we present part of a 4,000-node power-law graph created by the Inet-3.0 topology generator [16]. The circle represents a server node and arcs represent links to or from the server and 200 randomly selected nodes

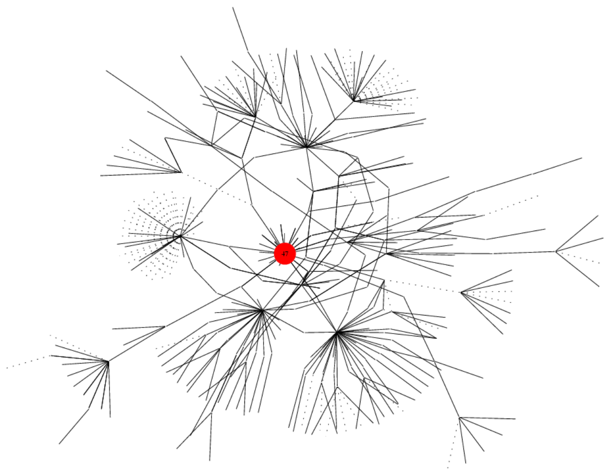

Fig. 5. Visual representation of a sample power-law graph, after several searches for a single object using the APS method. Solid line arcs show high index value links between nodes

that search for the object. Links drawn with solid lines represent index values stored by *APS* above a certain threshold (i.e. many successful searches through them), while dotted links show paths with low probability of success. After only a small number of requests, most paths that connect the server to the requesters are drawn with solid lines.

Augmenting this technique, we have designed a lightweight scheme [17] that enables peers to efficiently identify nodes that have shown interest in an object (point 3). Our method utilizes the state created during the search process to efficiently route messages from server nodes to peers that have searched for and retrieved an object. It requires no message exchange, nor any membership state to be kept at the nodes, having very reasonable space requirements. The same scheme can simultaneously detect a decrease in demand for a specific object. Therefore, it can also be used as a mechanism (other than the obvious request rate statistic) for peers to decide when to perform contraction, again with zero message exchange.

Figure 6 shows some preliminary results from this method. Using the same 4,000-node power-law graph, one server node and a varying number of nodes with request rates $\lambda = \{0.1, 1\}\frac{queries}{sec}$, we plot the success rate (= number of contacted requesters over total number of requesters) and stress (= number of messages sent over total number of requesters) of our method. We notice that around 90% of the requesters can be discovered with a very low message overhead (given that at least n messages are needed to contact n peers). Our method

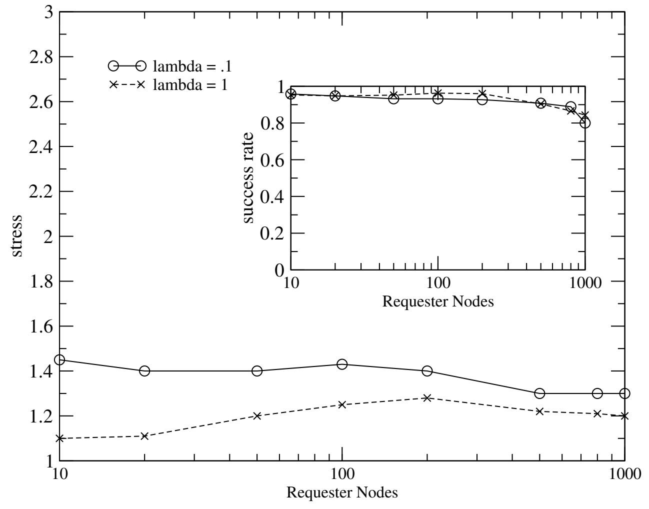

Fig. 6. The success rate and stress of our group notification scheme under variable number of requesters and request rates

can also easily adjust its routing scheme so that, given a number of requesters $\mathcal{R}$, at most $m \ll \mathcal{R}$ of them can be contacted with high probability.

Some other interesting questions emerge from our proposed solution. First, we intend to compare two different activation strategies, where either some intermediate peers or the affected server(s) initiate the expansion phase. Another question is how aggressive should the expansion or contraction be, under a certain workload and/or network dynamics. In other words, what is a good number of new replicas/servers to be activated given various demands for different objects. A valid concern is to operate in a proactive manner but avoid network oscillation; very aggressive behavior can result in recurrent transitions between over- and under-utilization conditions. We plan on implementing our system and measure its effectiveness with varying workloads and peer behavior, as well as investigate many of these interesting directions.

IV. Conclusions

In this position paper we presented the problem of sharing voluminous content using a Peer-to-Peer system. We believe this to be a particularly attractive and suitable application, based on a completely decentralized and unstructured system model. Our design aims at providing an adaptive response to workload changes, by creating extra service points in needy areas or releasing redundant servers from areas of low demand. There exist a lot of interesting questions which we intend to pursue, mostly in the direction of minimizing communication overhead while maintaining the quality of content sharing under varying conditions.

V. Acknowledgments

This material is based upon work supported by, or in part by, the U.S. Army Research Laboratory and the U.S. Army Research Office under contract/grant number DAAD19-01-1-0494

References

[1] Clay Shirky, "What Is P2P ... And What Isn't," *OpenP2P.com*, 2000.

[2] "http://www.napster.com.," Napster website.

[3] "http://www.gnutella.com," Gnutella website.

[4] "http://www.kazaa.com," Kazaa website.

[5] Sandvine Inc., "Regional Characteristics of P2P: File sharing as a multi-application, multi-national phenomenon," *An Industry White Paper*, 2003.

[6] "http://www.overnet.com/," Overnet website.

[7] Bram Cohen, "Incentives build robustness in bittorrent," 2003.

[8] Rob Sherwood, Ryan Braud, and Bobby Bhattacharjee, "Slurpie: A cooperative bulk data transfer protocol," in *IEEE Infocom*, 2004.

[9] Pablo Rodriguez and Ernst W. Biersack, "Dynamic parallel access to replicated content in the Internet," *IEEE/ACM Transactions on Networking*, vol. 10, no. 4, Aug. 2002.

[10] John W. Byers, Michael Luby, and Michael Mitzenmacher, "Accessing multiple mirror sites in parallel: Using tornado codes to speed up downloads," in *INFOCOM*, 1999.

[11] "http://www.morpheus.com," Morpheus website.

[12] Michael G. Luby, Michael Mitzenmacher, M. Amin Shokrollahi, Daniel A. Spielman, and Volker Stemann, "Practical loss-resilient codes," in *STOC*, 1997.

[13] Michael O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *JACM*, vol. 36, no. 2, 1989.

[14] D. Tsoumakos and N. Roussopoulos, "Adaptive Probabilistic Search for Peer-to-Peer Networks," in *3rd IEEE Intl Conference on P2P Computing*, 2003.

[15] D.Tsoumakos and N. Roussopoulos, "A Comparison of Peer-to-Peer Search Methods," in *WebDB*, 2003.

[16] C. Jin, Q. Chen, and S. Jamin, "Inet: Internet Topology Generator. Technical Report CSE-TR443-00, Department of EECS, University of Michigan," 2000.

[17] D.Tsoumakos and N. Roussopoulos, "Efficient Group Notification for Unstructured P2P Networks," *under submission*, 2004.