A Survey of Peer-to-Peer Storage Techniques for Distributed File Systems

Ragib Hasan[‡†] Zahid Anwar[‡†] William Yurcik[‡] Larry Brumbaugh[‡] Roy Campbell[†]

[‡]*National Center for Supercomputing Applications*
[†]*Department of Computer Science*
University of Illinois at Urbana Champaign
{rhasan,anwar, byurcik, ljbrumb}@ncsa.uiuc.edu, roy@cs.uiuc.edu

Abstract

The popularity of distributed file systems continues to grow. Reasons they are preferred over traditional centralized file systems include fault tolerance, availability, scalability and performance. In addition, Peer-to-Peer (P2P) system concepts and scalable functions are being incorporated into the domain of file systems. This survey paper explores the design paradigms and important issues that relate to such systems and discusses the various research activities in the field of Distributed Peer- to-Peer file systems.

1. Introduction

In the recent years, Peer-to-Peer system research has grown significantly. Using a large scale distributed network of machines has become an important element of distributed computing due to the phenomenal popularity of Peer-to-Peer (P2P) services like Napster [19], Gnutella [10], Kazaa [14] and Morpheus [17]. Though these systems are more famous for file-sharing, and related legal problems, P2P systems are becoming a very promising and exciting area of research. P2P systems offer a decentralized, self-sustained, scalable, fault tolerant and symmetric network of machines providing an effective balancing of storage and bandwidth resources.

File Systems have been a basic element of Systems research. Efforts have focused on providing a stable, reliable, efficient central storage system with certain performance constraints. Experience has shown that a distributed approach is better for achieving these goals. Early efforts included SUN NFS, CODA, Plan 9, XFS and SFS. Initial efforts emphasized sharing data in a secure and reliable way. Important features of these systems included a client-server architecture that was fundamental to their design caching, replication and availability.

Internet growth resulted in a new approach, the building of distributed file system. As the host nodes storing the shared objects became more geographically distributed and diverse, new criteria and performance constraints like availability, fault tolerance, security, robustness and location mechanisms became important issues in designing distributed file systems.

In recent years, P2P systems have emerged as a viable architecture for implementing distributed file systems. In a P2P network, end users share resources via direct exchange between computers. Information is distributed among the member nodes instead of concentrated at a single server. A pure peer-to-peer system is a distributed system without centralized control, where the software running at each node is equivalent in functionality. A P2P system should be highly scalable, available, distributed and more or less symmetric. The attractive properties of a Peer-to-Peer architecture have generated many research efforts in building distributed P2P file systems. Because of the success in this area, P2P systems are almost certain to become a major part of current and future research activities in file systems. This survey paper attempts to explore the inherent properties of such systems and analyze the characteristics of some major distributed P2P file systems. Also, the comparative advantage and disadvantages of each system are discussed in detail.

The rest of the paper is organized as follows: Section 2 discusses the benefits of using P2P systems over other distributed storage mechanisms. Section 3 explores design issues and desired properties of distributed P2P file systems. Section 4 identifies major research distributed P2P file systems, analyzing their comparative suitability depending upon selected criteria. Section 5 presents an analysis of the open problems. A summary and conclusions follow in Section 6.

Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05)
0-7695-2315-3/05 $ 20.00 **IEEE**

2. Justification of Using P2P Architecture for File System Design

The term Peer-to-Peer refers to a class of systems and applications that employ distributed resources to perform a function in a decentralized manner. Each node potentially has the same responsibility. Shared items can include CPU cycles (SETI@Home) or Storage space (Napster [19], Gnutella [10], OceanStore [15]).

Basic P2P system goals are decentralization, ad hoc connectivity, reduced cost of ownership and anonymity. P2P has more decentralized control and data compared to alternatives. P2P is potentially more scalable than centralized and client-server solutions. The basic defining feature of a P2P system is that it is characterized by direct access between peer computers, not through a centralized server.

Androutsellis-Theotokis et al. [2] defined P2P as "applications that take advantage of resources (storage, cycles, content, human presence) available at the edges of the Internet." According to [2], "the "litmus test"' for P2P is:

- Does it treat variable connectivity and temporal network addresses as the norm?
- Does it give the nodes at the edges of the network significant autonomy?"

In lieu of the above definition, a noticeable characteristic of P2P systems is that they have interesting self-organizing capacity, in that the topology of a P2P network can often change as nodes enter or leave the system. Important issues in a P2P system are considerably different than in traditional distributed systems, However, P2P systems provide certain advantages over conventional file systems that justify their usage in building distributed file system. For example, compared to the client-server model, P2P systems provide inherent scalability and availability of resources. They take advantage of the redundancy of the resources and construct a coherent view of the system using decentralized, independent components. The diverse nature of P2P systems and the large-scale distributed structure ensures the fault tolerance and resolute nature of P2P systems as compared with client server systems. The sheer number of nodes participating in P2P architecture contributes to advantage such as being adaptable, scalable and self-organizing. These essential features contrast distinctly with traditional client-server approaches that are limited by their lack of scalability and robustness in cases of component failures.

To make ubiquitous computing become a reality, the computing devices must become reliable, resilient and have distributed access to data. With this view in mind, the P2P system architecture appears to be most suitable to ensure the changing storage requirements of next-generation computing. The P2P architecture can help reduce storage system costs and allow cost sharing by using existing infrastructures and bundling resources from different sites. Resource aggregation adds value beyond the mere accumulation of resources and provides a rich, robust platform on which to build persistent storage systems. Considering all these factors, the P2P model should be very useful in designing the future generation distributed file systems.

3. Design Issues in P2P File Systems

Peer-to-Peer systems have basic properties that separate them from conventional distributed systems. Inherently, P2P systems are loosely coupled, and no performance guarantee can be provided; but the system as a whole contains common characteristics that affect its behavior in varying circumstances. This section discusses different design issues of a P2P file system and the potential effect of the issues on performance.

3.1 Symmetry

P2P systems are characterized by symmetry among the roles of the participating nodes. It assumes no special capability of certain nodes that would mark them separate from the rest of the nodes. Conventional client-server systems are asymmetric and the servers are often more powerful than the clients. However, in P2P systems, all peers are symmetric. They have the ability to function both as a client and a server.

3.2 Decentralization

P2P systems are decentralized by nature. Hence, P2P systems have mechanisms supporting distributed storage, processing, information sharing, etc. This allows increased extensibility, resilience to faults and higher system availability [16]. However, getting a global view of the system state is difficult. Also, system behavior no longer remains deterministic. Another problem is the issue of joining a group and discovering the peers belonging to that group.

3.3 Operation with Unmanaged Volunteer Participants

An important P2P design issue is that the participation of a given element can neither be expected nor enforced. System elements and storage nodes are not managed by any centralized authority. They are assumed to be prone to failure and removed from the system at any time. The system should be robust enough to survive the removal or failure of nodes at any moment.

3.4 Fast Resource Location

One of the most important P2P design issues is the method used for resource location. As resources are distributed in diverse peers, an efficient mechanism for object location becomes the deciding factor in the performance of such systems. The mechanism should be capable of adapting to a changing network topology.

Contrary to the P2P concept, Napster uses a centralized directory of object locations that proves to be a bottleneck. Gnutella [10] incorporates object location using non-scalable flooding. Elaborate schemes have been developed to solve this problem efficiently. Notable currently used object location and routing systems include Chord [26], Pastry [24], Tapestry [27] and CAN [23]. Pastry and Tapestry uses Plaxton [22] trees, basing their routing on address prefixes. This approach is a generalization of hypercube routing. However, Pastry and Tapestry add robustness, dynamism and self-organizing properties to the Plaxton scheme. Chord [26] uses the numerical difference with the destination address to route messages. This is unlike Pastry [24] or Tapestry [27] that use successively longer address prefixes with the destination. The Content Addressable Network or CAN [23] uses a d-dimensional space to route messages; with each node maintaining a O(d) sized routing table and any node within O(dN1/d) hops and the routing table does not grow with network size.

An important location strategy used in several systems is Distributed Hash Table (DHT). It uses hashing of file or resource names to locate the object. Kelips [12] is a DHT based system, which has the advantage of being efficient and scalable as well as using O(n1/2) space per node and O(1) lookup times.

3.5 Load Balancing

Load balancing is an important issue in building robust P2P file systems. The system should be able to make optimal distribution of resources based on capability and availability of node resources. The system should have mechanisms for preventing the build up of hotspots, locations where the load is disproportionately high. Also, it should be possible to rebalance the system based on usage patterns.

3.6 Churn Protection

Churn describes the fast oscillations in the P2P system caused by the rapid joining and leaving of nodes. It occurs when there is a node failure and corresponding joining of new nodes at the same time. Churn causes reduced performance in any distributed system. One form of a denial of service attack is to introduce churn in a system. Hence, a P2P distributed file system should be able to resist the churn effect.

3.7 Anonymity

In a distributed storage system, anonymity is an important issue to ensure resistance to censorship. There is need for resistance to attempts by third parties to deny access to information and provide anonymity for both the producers and consumers of information.

3.8 Scalability

Scalability implies the ability of the system to support millions of peers into a peer-to-peer system. Traditional distributed systems usually are not scalable beyond a few hundreds or thousands of nodes.

3.9 Persistence of Information

A P2P system should be able to provide persistent access to data. Methods should be present to ensure that even in the case of untrusted peers, the data stored in the system is safe, protected against destruction, and highly available in a transparent manner.

3.10 Security

Security from attacks and system failure are design goals for every system. P2P systems are built on unmanaged, geographically distributed hosts and data security is the systems responsibility. Encryption, different coding schemes, etc can help achieve this.

4. Some Existing Systems

Designing a P2P file system that can implement all the properties described in Section 3 is exceedingly difficult. Recently, a number of efforts have been made to achieve most of the goals. However, most of these systems utilize specific properties or mechanisms and

specialize in particular fields. This section discusses some existing P2P based distributed file systems.

4.1 FreeNet

Freenet [3,7] is an adaptive peer-to-peer file system that enables the publication, replication and retrieval of data while protecting the anonymity of the authors, data location and the readers. It uses probabilistic routing to preserve the anonymity of its users, data publishers, and data hosts. Basically, Freenet operates as a location-independent distributed file system across many individual computers that allow files to be inserted, stored and requested anonymously. The design goals of Freenet are: anonymity; deniability for the storers of information; resistance to 3rd party access; dynamic storage and routing; and decentralized policy.

4.1.2 Location and Access Mechanisms Freenet identifies files by keys obtained through a hash function, Current implementations of Freenet use 160 bit SHA1 cryptographic function as the hashing method. The key may be keyword signed key (KSK), Signed Subspace key (SSK) or Content Hash Key (CHK). Using any of the hash mappings, the source of the search sends queries. The query may be locally processed, or on failure may be routed to the lexicographically closest matching node according to the routing table. Communications by Freenet nodes are encrypted and are routed through other nodes to make it extremely difficult to determine who is requesting the information and what its content is.

On receipt of an insert request, a node first checks its own storage to see whether the key is already taken. In case of collisions, the user tries again using a different key. If the key is not found, the node looks up the nearest key in its routing table and forwards the insert request to the corresponding node that propagates through the nodes until the hops-to-live limit is reached. If there is no key collision, a success message is propagated back to the original sender. The data follows along the path established and is stored in nodes along the way. Data is stored in an LRU fashion and older unused information gradually fades away.

Table 1. Freenet Tradeoffs

Advantages	Disadvantages
Freenet attempts to provide anonymity both for producers and consumers of information.	Anonymity requirements limit reliability and performance, since the probabilistic routing mechanism stops forming of any coherent topology among servers.
Performance analysis shows: as the network converges, median request path length drops.	An unpopular file might disappear from the network if all nodes decide to drop its copies.
Network is scalable up to a million nodes with a median path length of just 30.	Dictionary attacks to modify of requested files en route is possible for files stored under keyword-signed keys.
Replicate popular data items transparently near requesting node. With time, the network routing learns and remembers requests for better performance.	Denial-of-Service attack through insertion of a large number of junk files.
The network is robust against quite large failures.	The flat namespace produces globally unique identifiers and versioning might become a problem as the system grows
The popularity of each site's material causes the system to actually alter its topology	Suffers from problems of establishing initial network connection.
Hashing renders Freenet unusable for random searches	No search mechanism. A standard search allows attacks to take out specific content holders
Rewards popular material and allows unpopular material to disappear quietly.	Scalability, resilience testing in a real world scenario is lacking.

4.2 CFS

Cooperative File System (CFS) [5] is a peer-to-peer read only storage system developed at MIT with the following design goals: provable guarantee for efficiency, robustness, load balancing and scalability.

4.2.1 Mechanism. CFS uses Distributed Hash table (Dhash) for storage of blocks. The file system is designed as a set of blocks distributed over the CFS servers. A file is divided into constituent blocks that are stored among different nodes. CFS has 3 layers: FS which interprets blocks as files and presents a file system interface to applications, DHash, distributed hash table that stores unstructured data blocks reliably and Chord [26] which maintains routing tables for lookup and query management

CFS is a read only system from the perspective of the users. However, the publishers can update their work. Key based authentication is used.

Table 2. CFS Tradeoffs

Advantages	Disadvantages
Quota on publishers provides a security advantage	Maintaining a single file in many blocks introduce overhead of fetching the blocks
Dividing a large file into chunks removes the problem that one node may not have the capacity to store the whole file	To enhance performance, CFS sacrifices anonymity. So, it does not provide the same censorship-resistance as Freenet
Caching and replications decreases response time	
Usage of Chord allows logarithmic lookup times	
Distributed storage of a file allows parallel block retrieval	

4.3 PAST

PAST [25] is a large scale P2P persistent storage management system. It is comprised of self-organizing, Internet Based overlay network of storage nodes which route file queries in a cooperative manner, perform replica storage and caching.

4.3.1 Mechanism. PAST is built on top of the Pastry [24] lookup system. The nodes form an overlay network. A 128-bit node identifier that is assigned quasi-randomly uniquely identifies each node. This uniformly chosen random identifier ensures load balancing. Files also have a file id that is a SHA-1 hash of the file name and the public key of the client. The Pastry layer handles the lookup requests. Replications enable fast lookup and transmission. To retrieve a file, a client uses its fileID, and in some cases, the decryption key. For the client, PAST provides three main sets of operations.

- Insert: store a file replicated k times, k being a user specified number,
- Lookup: reliably retrieve a copy of the file identified by fileId if it exists in the PAST
- Reclaim: reclaim the storage occupied by k copies of the file.

4.3.2 Security using Smart-Cards. The system uses smart-card key based techniques for security, load balancing and free storage re-allocation by replica diversion.

Table 3. PAST Tradeoffs

Advantages	Disadvantages
There is no restriction that Pastry must to be used. Due to modular design, Chord, CAN or others can also be used.	PAST stores a single large file without breaking it into smaller chunks (as in CFS). This is not efficient or fault tolerant.
Files in PAST are immutable, so multiple files cannot have the same fileId.	PAST is an archive and storage system, rather than a general-purpose file system utility.
Smart cards are not used in other systems.	

4.4 IVY

IVY [18] is a read/write peer-to-peer file systems that is distributed and decentralized and able to support multiple users concurrently. The system is based on a set of logs and the DHash distributed hash. It provides an NFS-like file system view to the users, while at the same time; it can detect conflicting modifications and recover from network failure.

4.4.1 Mechanism. The IVY file system is based on a set of logs that each participant keeps to record the changes made to the system. Each user scans and synchronizes the logs. Snapshot mechanisms prevent scanning of all but the most recent log. The logs are themselves stored in DHash. IVY overcomes the overhead of multiple accesses and locking. It also uses version vectors for synchronization. Integrity of each block is ensured by either content hash key or public key. Since logs are stored indefinitely, recovery is always possible in case of network partitions. The total system state is a composite of all the individual logs. Periodically, each participant takes snapshots to avoid future scanning of the entire log.

Table 4. IVY Tradeoffs

Advantages	Disadvantages
It enables writing with reading. Other systems discussed so far seem to be read only systems.	Slow. Ivy is 2 to 3 times slower than NFS [18]
No need for explicit trust between the hosts	Conflicting log records generated. Explicit conflict resolution tools have to be used.

4.5 OceanStore

OceanStore [15] is a proposed system to provide distributed access to persistent nomadic data in a uniform global scenario. It is designed using a cooperative utility model in which consumers pay the service providers certain fees to ensure access to persistent storage. The service providers in turn use utility model to form agreement and resource sharing. Data stored in OceanStore

4.5.1 Mechanism. Using mainly untrusted servers, OceanStore caches data anywhere in the network, with encryption. This provides high availability and prevention of denial-of-service type of attacks. Persistent objects are uniquely identified by a Global ID (GUID) and are located by either a non-deterministic but fast algorithm (Attenuated Bloom Filters) or a slower deterministic algorithm (Modified Plaxton Trees [22]). OceanStore uses ACL for restricting write access to data, while read access is available with the key. Updates are achieved using the Byzantine agreement protocol between the primary replica and the secondaries. For high performance, OceanStore also provides self-monitoring introspection mechanisms for data migration based on access patterns. This is also used to detect clusters and improve routing performance.

Table 5. OceanStore Tradeoffs

Advantages	Disadvantages
It is suitable for ubiquitous	The system is still in the

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.