Unlearning the Traditional File Systems Model, Applying Peer to Peer Techniques for Information Management

Chaitanya Vinay Hazarey
CSCI 555 Term Research Paper
Department of Computer Science,
University of Southern California, Los Angeles

Abstract- Distributed File Systems since the very early days have been thought of and developed on the lines of a "strict"[1] Client–Server architecture. This is the thing Sturgis et al.[1] refer to in their paper as an "independent file facility". This facility was always visualized as an extension to the traditional (non-distributed or local) filesystem and consequently was applied the same design model. This paper tries to explore the possibilities of augmenting the distributed file systems design model, with the features of the Peer to Peer file sharing system.

An analogy can be drawn to the method of solving a maze by starting at the desired exit and tracing the path towards the subject. We take a look at the various Peer to Peer[2] systems in use now, their structure (or the lack of one), their strong and weak points. Next we compare various P2P architectures, and some implementations based on each of them. Although we will try and explore the weak and strong points of each relevant in this case an in depth discussion is avoided and the interested reader can find the respective papers in the references.

There are some examples of P2P based file systems which have been proposed and implemented, quite a few of them based on the Chord distributed lookup system[2] (using the Distributed Hash Tables). We explain the system in brief and judge the suitability of the proposed mechanism for P2P systems. Finally we explore the Prospero Resource Manager interface, for its suitability to this case.

I. INTRODUCTION

We have always perceived Distributed File Systems as an extension to the local file system of the machine. This perception is natural and true, but when we apply the design model used for the local file systems to the design of distributed file systems, we encounter many problems and shortcomings of the model. These shortcomings in the model manifest themselves due to the fact that the traditional file systems were designed with a localized and specialized system in mind. This system was constrained by the location, physical constraints, capacity, operating specifications, slow and costly I/O interconnects. It was bound to a single device which was considered as the only secondary memory of the system. This seminal paper on the development of the Unix OS [3] nicely summarizes the challenge that was the design of a file system. This excellent UNIX file system design has been the basis of many file systems since then.

Due to the recent advances in the Hardware and the Software used to build and interconnect computer systems, there is room for expansion of the secondary storage outside the boundaries of the local machine (or the local network for that matter). This expansion has brought with it a need for the efficient and seamless management of the vast amount of information present. Now the data can span not only several devices, machines, networks but also different authoritative domains. Different distributed file systems were deployed to exploit the recent advances in the technology, each one stretching the basic traditional file system model in a different direction. The main concern till now has been the efficient storage of data and the simple on demand data retrieval mechanism for the user.

P2P file sharing system is on the other end of the spectrum, though not a new concept and dates back to 1979 (the Usenet system), recently it has been in the news as "Pirate to Pirate" network. This (negative) publicity of the concept is owing to some of the immensely famous P2P file sharing applications such as KaZaA [18], Gnutella [6], Napster [15] [16], etc. Although efficient tools to distribute and organize data, there capabilities were exploited for some illegal purposes. We in this paper will not dwell into the moral of the legal issues of such tools and mechanisms, but keep an extremely objective view point. We will try and highlight the various elements in the design which are responsible for the efficient working of the file sharing system. A P2P system, if we consider an ideal system with all the various parts perfectly designed and working together, without any freeloaders, is a unique model of content distribution in which the Producers increase in direct proportion to Consumers.

For the rest of the paper we will start with the study of various P2P mechanisms in place, and their respective implementations. Then we will move on to the inferences gathered from the study of these systems and will present the reader with the strong points and the weak points of such these systems. We will then go ahead and have a look at the various P2P based file systems in implementation and will see if these distributed file systems fare and better than the traditional ones. We will conclude the paper with a proposed mechanism for the management of information using P2P systems, which build up on some existing systems and

[1] We view Peer to Peer to be a kind of lenient Client Server Architecture in which each node participating is the client and the server.

[2] Hereafter we refer to these systems as P2P systems, as they are widely know.

mechanisms and will try to extend the models to include some aspects of P2P systems.

II. Peer to Peer File Sharing Architectures

The reader can visualize a P2P system as a network formed (we base this on the assumption and the requirement of similar P2P software installed, functional nodes and the ability to share data) between certain nodes, which as the situation demands assume the role of a client or a server. There is one thing which separates this sort of network from a Distributed File System, which is the obvious lack of central control and coordination. This and the fact that the Requestor will make direct contact with the Provider and access the data. Also as in the P2P network there is an absence of some controlling authority a fair amount of Control-Data is generated in an effort to keep the current "knowledge" of the system coherent. But this type of generic description will not aid us in the developing the central topic of the paper, which require a better understanding of the various P2P system architectures. So we will look into some classifications of the P2P systems and their implementations.

We can generally separate the existing P2P mechanisms in 2 groups, Structured and Unstructured. But this classification as given by Lua et al.[4] does not cover all the various aspects of the P2P architectures. A better classification is proposed by Backx et al.[5]. They suggest that the P2P systems can be separated into various categories based on the distribution of data and the management of control. So we proceed to elaborate the classification along these lines. The paper proposes three classes of architectures for the P2P systems, i.e. Mediated, Pure P2P and Hybrid. Now we will highlight the structure of each of the system and proceed to give the example implementations.

The pure P2P architecture is characterized by total absence of a central controlling authority. The peers of the network may join or leave the network arbitrarily, but based on some non-enforced rules. This type of architecture is quite easy to bootstrap and does not require accumulation of any central knowledge for functioning. Data is usually stored in the form of individual and uncontrolled file structures. These are maintained on each peer and constitute its data together with a list of files the peer is willing to share. Queries are usually flooded amongst the nearby peers (proximity can be based on TTL, hop-count, in/out degree of the node or any other factor). Although this design contains potential to implement anonymity and also resilience to dynamic hosts, but this comes at a cost. The so called pure P2P networks are costly to maintain due to the overhead of control-data which is required to keep the "knowledge" of the system coherent. An excellent example of this architecture would be the Gnutella system [6] (old version, with no ultra-peers). The popularity of the application was partly due to the open publication of the Gnutella Protocol and the availability of various stable tools to implement it *** please see if you have to say some thing more here about the popularity. I am forgetting some words from a paper ***.

The presence of a central authority which contains, controls or mediates certain amount of "control-data", for the knowledge of the location of data on the peers. This can be implemented as a simple central server database which will aggregate the different data inventories provided by the peers contributing to the network. Each search which is made to the central server to query the database, can be handled there or can be distributed to various peers, and the results subsequently aggregated. These type of P2P systems although have the excellent ability to handle sudden increase in peers and therefore the resultant queries, but there are some problems introduced because of a central knowledge repository. This central authority not only becomes a central point of failure, but is very harmful to anonymity of the network and also makes an easy target for a lawsuit to be filed against. Napster***[?]*** give a source here.*** is an excellent example of such a system.

The last category of the systems is called the hybrid architecture based systems. These are supposed to be a sort of one of the "win-win" situations. The motivation behind these is to draw on the best of both the worlds, capitalize on the features that made them successful and popular. But there are some concerns about these as introduction of all the good things about the previous approaches do not necessarily constitute a good system. The basic idea here is to form a sort of an "super-peer" network [7]. They capitalize on the good points of the pure P2P systems by forming a completely distributed network; this exploits the virtues of the pure systems such as, autonomy, load balancing and resilience to attacks. This is done by giving recognition to the heterogeneity of peers, it basically means taking away some functionality or duties away from the weak or normal peers and giving these to more computationally and better networked efficient peers. This enables the super peers to form their own small P2P networks with a small number of peers. These normal peers are dependant on these super peers for the facilitation of search procedures, and other control and house keeping information. The communication and coordination between the super peers is pure P2P. The system also defines a specific mechanism for the transition of peers between the two different status *** does not sound good here just check this out ***.

An excellent example of such a system would be the newer version of Gnutella protocol which select some peers in the network to be "ultrapeers", and these in turn maintain the metada about the data stored on their member normal peers. This facilitates a sort of central search algorithm. There are studies underway [7] to validate this claim and provide some more answers about the exact organization of the normal and super peers.

So, P2P systems can belong to any of the above mentioned architectures or could combine together some features of each approach. But these classifications give us the tools to properly study the various P2P implementations and present a platform for comparing there various features

and facilities. But these systems in themselves are "file-sharing" systems and although have some features of a file system, but are not full fledged file systems. So these can only serve as a source for the inspirations of certain mechanisms on which full fledged distributed file systems can be based upon.

Next we study some P2P based file systems, their respective mechanisms, strong and weak points. This information would give us better arguments to be made in the favor of deploying distributed file systems which draw inspiration from P2P systems.

III. Peer to Peer based Distributed File Systems Implementations

The CFS file system, described by Dabek [8] [9] as a P2P read only storage system is a good example of an entire file system based on the P2P systems. The CFS manages the constituent files as blocks, these can be the root blocks, data blocks, directory blocks, inode blocks, etc. One of the important things that CFS assumes that the data that is to be shared is supposed to be read only. The data is distributed and indexed amongst the contributing nodes using the Chord system [10]. There is a layer called the Distributed Hash System[3] which is implemented on the top of the Chord system. This is responsible for the storage of unstructured data blocks reliably. Chord assigns the data blocks and the nodes a unique ID, which is from a 160 bit namespace. It then implements a hash function which is responsible for mapping the ID from the block identifiers to the servers where the data is stored. The DHash layer uses Chord [10] for the lookup of this information. DHash layer implements the load balancing, caching, block storage and also lenient quota allocation to deter data pollution. The Chord layer is responsible for the decentralized search, scalability and availability. Chord provides the hash functions and also tables which maintain the data routing information on each node. These tables are structured in such a way that, the magnitude of dynamicity in the number of nodes in the system does not require a lot of changes in the tables.

CFS is an implemented file system which adopts the naming, authorization and file system structure ideas from the SFSRO [11]. Because if this it also suffers from the same problems as the SFSRO. One of these is that every modification of a file (it's a read only system, so this will be rare but present), will require the owner to reinsert the blocks from that file to the root block, which intrun would again be resigned and reinserted. The speed of CFS is a little slower than that of the FTP protocol and still has got some room for improvement.

Another example of a file system which is based on the P2P systems is Ivy [12]. Ivy on the other hand is a read/write P2P file system. The interfaces provided by Ivy are standard file system interfaces. There is a lack of centralized control and dedicated components. The implementation of this file system is similar to the pervious file system; Ivy is implemented on the top of DHash which is layered on the top of Chord. DHash provides the interfaces to accept the key values and providing the relevant data. And Chord provides the lookup system. The thing that makes Ivy different from the previous file system apart from the fact that it is a read/write system is that it is a Log-based file system. This file system also provides for close-to-open consistency of files. Its mechanism is similar to the Sprite File system [13] (Log based file system). This file system is implemented as a set of logs, one per participant. The owner appends own log only but can read all others. A log-head points to the recent log-record. These log records are in turn stored in the DHash. Periodically Ivy server consults all logs to find relevant information, and combines them to maintain state. The file system supports concurrent file operations by ordering the file system operations and executing them. The full traversal of the log files by each participant is prevented by allowing the participants to maintain a private snapshot of the system.

The idea of using multiple logs for this purpose is debated and under question. The performance of Ivy is some what comparable to NFS and is therefore acceptable.

All the above systems have used Chord in some way to facilitate the data search. It is a P2P lookup service. The purpose of Chord is to facilitate data lookup in a collection of distributed nodes, taking care of the frequent additions and deletions to the set. Chord also provides for the data routing capabilities. In each Chord node there is some amount of routing information maintained and updated every time a node joins or leaves the system. For lookup of data items between Chord nodes a single node requires information about few other neighboring nodes.

But Chawathe et al. [14] provide an alternative approach based on Gnutalla called "Gia" and also cite certain problems in Chord system.

They cite the following problems in the Chord system:

1) **P2P clients are extremely transient:** The measured churn rate for a system the size of about 100,000 nodes is over 1600 nodes per minute. This does not severely affect the Gnutella based systems, in the worst case will warrant a bootstrap of the system with new neighbors. But in a DHT (Chord) system it is required to perform O(log n) operations (repair) to maintain consistency after such a failure. This overhead is quite substantial in case of high churn rate and may require the slow peers to go offline.
2) **Keyword searches are more prevalent, and more important, than exact-match queries:** DHTs are very efficient when they are required to search for an exact match. But in a real world environment, the need is basically for a keyword based search or a broad search. This is not a trivial extension of the facilities provided by the DHT. But it requires significant modifications and will take quite an extensive effort to maintain. The overhead generated would defeat the purpose of maintaining such a search.
3) **Most queries are for hay, not needles:** DHT is very apt at finding a data item even if there exists a

[3] Called DHash hereafter.

single copy of the item in the system. Gnutella is very poor at performing searches for such items, which are termed as “needles”. But in the real world the searches made are usually for the “Hay” or the well replicated items. Gnutella excels at this thing.

They also propose a design of a system called “Gia”, drawing inspiration from Gnutella, KaZaA and various other advances in the field of P2P computing. The various enhancements are replacing of the Gnutella flooding algorithm with random walks. These random walks will be enhanced taking into consideration of the topology of the network and will be optimized according to that. It also removes the weakness in the previous Gnutella algorithm by removing the absence of flow control and introduces a token based flow control. The discussion of various enhancements in detail is beyond the scope of this paper, but the cited paper provides the details of the implementation and the resulting performance enhancements.

IV. Lessons that can be Learnt from the P2P systems and P2P based File Systems

We have upto now explored the various architectures of P2P systems, their respective implementations and also the file systems based on P2P. There are various unique points in the design model of these systems and there are some lessons to be learnt from them. I have enumerated the specific design aspects of P2P systems that can be used and used efficiently in the Distributed File Systems design.

Following are the lessons that can be learnt from the various P2P systems and P2P file systems implemented:

1) P2P search: Most of the distributed file systems in implementation currently lack this facility in the design model. This would be a great advantage for the distributed file systems if it is implemented in the file system model itself instead of layering on top. We have a choice to implement this file system, it can be done using the Gnutella/Gia approach or the Chord/DHT approach.
2) P2P data caching: In a distributed file system, we may find and exploit any locality of reference present in the data access pattern. If we implement the data caching based on a P2P network and limit the P2P network to the maintenance of coherence of data, it would save a lot of communication overhead.
3) P2P data routing: A feature that is common to all the Peer to Peer system is that, the data transfer is always from a Peer to Peer, there is no intermediate node. This can be included in the distributed file system model and the need for server intervention all the time could be avoided. Also the facility to route the data through different peers is not implemented in distributed file systems, if it is, we may be able to exploit the topological factors in the network.
4) Recognize the heterogeneity of nodes: The distributed file system assigns similar roles to all the participating nodes. This is not fair to the weaker nodes which have fewer resources at their command. And also the resources of the nodes which have capability to handle more load is underutilized. Instead we should design the file system such as the heterogeneity of the peers is recognized and exploited and the bottlenecks caused because of a small number of weak peers is avoided.
5) Multisource downloading and Swarming: This may be taken as a sort of “side-effect” of P2P caching. As this is a P2P system the nodes will be aware of their neighbors and will attempt to retrieve the required data from various sources. This may lead to a situation where a single file is stored on different machines and the requestor and download it from all of these nodes. If the data is divided into blocks as it is done in CFS, and the P2P system is aware of the constituent blocks of a file, then different parts of the file can be downloaded from different peers and then combined to form the original file. *** Telel them that it would require additional security measures to be taken but it should be good. ***
6) Incentive model: A kind of P2P system which has not been mentioned earlier is the BitTorrent system [17]. This system is trying to implement an incentive scheme in the design of the system so as to “reward”, the peers who are actively sharing files from their repository. Basically the real advantage of this scheme is to reduce the number of free loaders on the network. Freeloaders are the peers who download the data but refuse to share it with others and thus harm the efficiency of P2P ecology. Because of a large number of freeloaders, the P2P system may suffer adverse effects due to the overloading of the peers who are allowing downloads from their repositories.
7) Harness idle resources: P2P systems due to their nature and design are particularly good at harnessing idle computing and networking resources. This increases the overall performance of the system and also is important to increase the efficiency of the system.

V. Conclusions

In this paper we have explored the various architectures of the P2P systems and also the respective implementations. The Hybrid architecture of the P2P systems seems to be the best solution in context of current systems. It seems to be the healthiest tradeoff between the anonymity of sources and efficiency of data provision. In the context of the P2P based file systems I think that the DHT/Chord type file system is still the most viable solution if we look at the problem from a design perspective. There may be some enhancements which can be made to the DHT/Chord model to further enhance the good points about that design. Such additions may include but not limit themselves to implementation of keyword search and the ability to implement query and data caching at various places in the design.

In the end we saw the lessons we could learn from the design model of the P2P file systems. In essence the P2P file systems are distributed systems in themselves so the research done in distributed systems is applicable to them. This also means that, the aspects of P2P systems found interesting have the potential to be integrated seamlessly in the current design. There are two aspects that matter the most while this integration is to be done, that is Performance and Security. All the file systems which were based upon the P2P systems and I have elaborated are either lagging in performance or security. And there seems to be a tradeoff between the two. I hope that the future distributed file systems will be designed taking into considerations the lessons learnt from the P2P systems and will find a "sweet spot", where the tradeoffs between the various factors in the different implementations will give the maximum efficiency of data storage and retrieval.

REFERENCES

[1] H. Sturgis, J. Mitchell, J. Israel, "Issues in the design and use of a distributed file system" *ACM SIGOPS Operating Systems Review archive*, Volume 14, Issue 3, pp 55-69, July 1980.

[2] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, Hari Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications", *IEEE/ACM Transactions on Networking (TON) archive*,Volume 11, Issue 1, February 2003.

[3] Dennis M. Ritchie, "The Evolution of the Unix Time-sharing System", *AT&T Bell Laboratories Technical Journal* 63 No. 6 Part 2, October 1984, pp. 1577-93.

[4] Lua K, Crowcroft J, Pias M, Sharma R, Lim S, "A survey and comparison of peer-to-peer overlay network schemes", *Communications Surveys & Tutorials, IEEE (2005)*, pp. 72-93.

[5] Peter Backx, Tim Wauters, Bart Dhoedt, Piet Demeester, "A comparison of peer-to-peer architectures", *Eurescom Summit 2002, Heidelberg, Germany*.

[6] Gnutella Development Team, Gnutella development forum, the gnutella v0.6 protocol, [*Online*], Available: http://groups.yahoo.com/group/the gdf/files/ (2001).

[7] Beverly Yang, Hector Garcia-Molina, "Designing a Super-Peer Network", 19th International Conference on Data Engineering (ICDE'03) pp. 49.

[8] Frank Dabek, "A Cooperative File System", Master's Thesis, *Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology*, September, 2001.

[9] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative , storage with CFS", *In Proceedings of the 18th ACM Symposium on Operating Systems 18th Principles (SOSP '01)*, Chateau Lake Louise, Banff, Canada, Oct. 2001.

[10] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, Hari Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications", *IEEE/ACM Transactions on Networking (TON) archive*, Volume 11, Issue 1 February 2003 pp. 17 - 32.

[11] Kevin Fu, M. Frans Kaashoek, and David Mazières, "Fast and secure distributed read-only file system", *In the Proceedings of the 4th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2000)*, San Diego, California, October 2000.

[12] Athicha Muthitacharoen, Robert Morris, Thomer M. Gil, Bengie Chen, "Ivy: A Read/Write Peer-to-Peer File System", *Proceedings of 5th Symposium on Operating Systems Design and Implementation*, 2002.

[13] Michael N. Nelson, Brent B. Welch, and John K. Ousterhout, "Caching in the Sprite network file system", *ACM Transactions on Computer Systems*, 6(1):134-154, February 1988.

[14] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, Scott Shenker, "Making gnutella-like P2P systems scalable", *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, Karlsruhe, Germany, pp. 407-418.

[15] OpenNap: Open Source Napster Server, Napster protocol specification Current documentation for known client-server protocol, [Online] (last updated on March 12, 2001), Available: http://opennap.sourceforge.net/napster.txt.

[16] Napster, From Wikipedia, the free encyclopedia, [Online], Available: http://en.wikipedia.org/wiki/Napster

[17] J.A. Pouwelse, P. Garbacki, D.H.J. Epema, H.J. Sips, "A Measurement Study of the BitTorrent Peer-to-Peer File Sharing System", Technical Report Delft University of Technology, The Netherlands, Available: http://www.isa.its.tudelft.nl/~pouwelse/bittorrent_measurements.pdf

[18] Nathaniel Leibowitz, Matei Ripeanu, Adam Wierzbicki, "Deconstructing the Kazaa Network", Proceedings of the The Third IEEE Workshop on Internet Applications, 2003, pp 112.

Add all the references you have touched till now !!!!