A Distributed File System Model for Shared, Dynamic and Aggregated Data

Bardac M.*, Milescu G., Rughiniş R.*****

Computer Science Department, Politehnica University of Bucharest,
Splaiul Independentei 313, Bucharest, Romania
**mircea.bardac@cs.pub.ro **george.milescu@cs.pub.ro ***razvan.rughinis@cs.pub.ro*

Abstract: This paper presents a modern design based on peer-to-peer networks for storing and retrieving data in a distributed loosely-coupled environment. Using a conventional file-system interface, the designed distributed file system provides access to shared, dynamic and aggregated data. A flexible interface design allows local or global querying and aggregating of the dynamic data in the distributed system, setting up an environment for future developments on information discovery.

1 INTRODUCTION

Distributed file systems are typically used to provide access to a large amount of resources connected in a network. These systems are usually managed and data is spread across the network in order to provide high throughput and availability. Data sharing in peer-to-peer networks provides an alternative storage possibility across a dynamic network.

Data processing in distributed environments has been mainly used on static data. The Map-Reduce programming model described in (Dean et. al., 2008) allows distributed data processing on a large cluster of commodity machines. This processing environment is suited for static data. Having an environment capable of distributing dynamic data as presented in this paper, processing of this type of data becomes a real possibility.

The paper presents a design model that uses a peer-to-peer network as transport environment for the distributed file system. This brings in advantages such as decentralization and reduced management of the file system infrastructure. The added advantage using the peer-to-peer network provides also the ability to offload data processing to peers serving the requests, an essential feature in the presented file system design model.

The most common and simple application for the proposed file system is file sharing between users. Given the nature of the underlying peer-to-peer network, shared files can easily be accessed by participating peers.

The peer-to-peer infrastructure also allows gathering node-specific (non-shared) data. This creates new usage possibilities for the distributed file system, such as collecting data from a network of sensors. Considering such a network (for example temperature, humidity or proximity sensors), the data collected by them can be the integrated in the designed ad-hoc file system, thus making it available under a single, conventional interface for user applications. Moreover, sensors could easily exchange data between them, allowing better data correction and interpretation.

Moreover, the gathered data can be processed and returned in a summarized form to the requesting entity, thus offloading the processing to the peer-to-peer network entities.

2 FILE SYSTEM DESIGN MODEL

2.1 Components

The designed distributed file system model has two major components: storage entities and client entities. **Storage Entities** are nodes in the system, which make their data available through the distributed file system. **Client Entities** are nodes in the system where data in the distributed file system is being made accessible through normal Virtual File System (VFS) entries.

VFS entries are being created through File-system in userspace (FUSE) kernel module (Szeredi, 2005) available on Linux, FreeBSD, OpenSolaris and Mac OS X. The designed distributed file system (DFS) model provides access to several types of files: shared files, dynamic files and aggregated data in the form of files as described in section 2.3.

2.2 Overlay network

All the entities are connected through an overlay network (Andersen, et. al., 2001,) built over the actual peer-to-peer network. This minimizes the number of connections each entity uses to maintain its position in the distributed file system. The overlay network is used to pass control messages between entities and to send and receive aggregated data. Static or dynamic data (file) transfer between entities is being done using direct connections between the requesting Client entities and the serving Storage entities.

The overlay network is a spanning tree covering all DFS nodes. When an entity enters the system, it determines the "closest neighbor" available and connects to it. In the proposed design, a "close neighbor" is a node which can be used for transfers using the fastest speed of all neighboring

nodes. The "closest neighbor" becomes a parent for the new node in the spanning tree. When the new node connects with its parent, the parent adds it to its neighbors list. Updates in the network and aggregation requests will be passed to the newly connected node by the parent.

2.3 Data representation

There are three main types of data which can be served through the distributed file system: persistent shared data, dynamic local data and aggregated data.

Persistent shared data

Most distributed file systems to date are designed to serve persistent data. This data does not usually have frequent updates and can be easily cached. This is the case for large media files such as video and audio files which are being exchanged through peer-to-peer networks (Androutsellis et. al., 2004). Maximizing throughput in the current design is realized by increasing the number of replicas for a given file. This also leads to increased availability for the replicated files but decreases the available space on the sharing peers. These problems can be overcome by including in the design model different data caching policies (Burns et. al., 2000) that govern how much data should be cached, where caching should occur and how caching should be triggered. For design simplicity, the file system model presented in this work uses a very simple caching policy for shared data: an entity will cache a file (create a replica) only when opening the file.

Dynamic local data

The DFS model presented in this paper is also designed to serve "dynamic data" in the form of "dynamic files". This data can be produced by sensors or by processes whose state is continuously changing. The dynamic data is node-local data, accessible only by querying the generating node and is not shared across the DFS. The high rate of changes prevents caching for this type of data. In order to simplify data processing, the distributed file system model also supports built-in aggregation of the dynamic data.

Aggregated data

The dynamic data can be aggregated in the network, offloading processing to all the nodes involved in the aggregation. This decreases load on the requesting entity and minimizes the number of messages exchanged in the network.

Aggregation can be done using a process similar with Map-Reduce (Dean et. al., 2008). The Mapper and Reducer functions are available on each node. An aggregation request triggers all mappers on the nodes involved in the query. The Mapper functions map requested local dynamic data to keys, forming (key, value) pairs. Results are sent back through the parents. Leaf nodes only run the Mapper function. All non-leaf nodes except the request node first run the Mapper function. All generated (key, value) pairs, including the pairs received from the child nodes, go through the Reduce function which generates a new set of (key, value) pairs, summarizing for example the input data. This is similar to a typical Map-Reduce functionality (Dean et. al., 2008) but adds the complexity of having incremental reductions. The requesting Client entity receives the aggregated results from the neighboring nodes and applies the last Reduce function, obtaining the data that is being delivered as contents of a file.

Typical uses of aggregation include knowledge gathering in multi-agent systems, data mining, content search (in available files), merging data from sensor-networks and determining the state of a particular area in a distributed environment.

Aggregation operations can be localized by providing a range. The functions can be applied by using a simple VFS path to a particular area of the distributed system (described in sections 2.4 and 3.3). For example, an overlay network with a 5-nodes radius is a sub-tree of the overlay network where, starting from a node, all nodes at a distance not higher than 5 edges are included. The entire distributed system can also be used as a range. A sample usage for aggregation on a 5-node radius would be getting the average temperature in an "area" of the distributed system.

2.4 File system hierarchy

The data in the DFS model is exported for simple manipulation through FUSE (Szeredi, 2005). Files associated with each data type described in section 2.3 are available under different folders in the VFS hierarchy.

Starting a Client Entity with the default options and a mount point in the `/mountpoint` directory will make all file types available within paths as presented in Table 1.

Table 1. DFS Mount Paths

Path	Contents
`/mountpoint/share/ <file-path>`	shared files, available on one or more nodes
`/mountpoint/node/ <node-name>/ share/<file-path>`	local cache of shared files on node <node-name>
`/mountpoint/node/ <node-name>/ dynamic/ <file-path>`	dynamic files served by the node <node-name>
`/mountpoint/range/ <range-description>/ <function-name>/ <file-path>`	aggregated result for function <function-name> applied on nodes within range <range-decription> on the dynamic files accessible on each node at <file-path>

The `<range-description>` is used for limiting the Map-Reduce functions to a specific area of the overlay network.

3 SYSTEM ARCHITECTURE

3.1 Entities

There are two types of entities in the DFS design model: Storage and Client entities. **Storage Entites** (Figure 1) are run on each node where data for the DFS is placed. **Client Entities** are run on systems where the DFS needs to be exported through FUSE (Szeredi, 2005). Each Client Entity is designed to run together with a Storage Entity, sharing the Multicast Discovery Component and the Peer-to-Peer Connection Component. The Client Entities use the Permanent Storage Components on the Storage Entities for data caching (only for shared files).

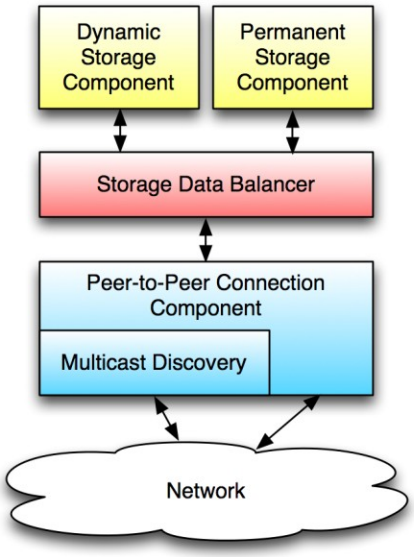

Fig. 1. Storage Entity

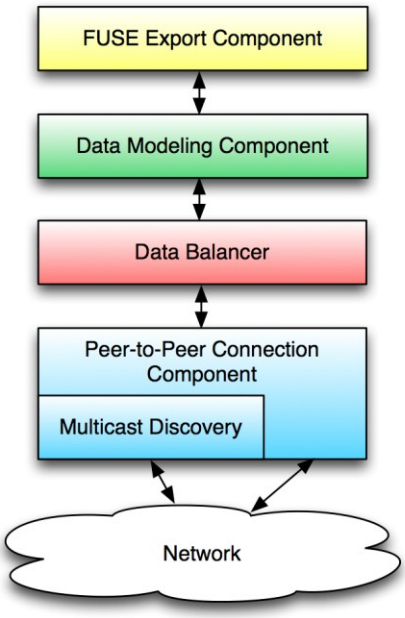

Fig. 2. Client Entity

3.2 Entity Components

Entities in the DFS model have been designed in a multi-layer architecture. Some layers are shared between Storage and Client entities, making this architecture easily maintainable. Components on each layer are designed as service providers to higher-layer components, providing extensibility in the case of future improvements.

Common components used in both types of entities are the Multicast Discovery Component and the Peer-to-Peer Connection component.

The **Multicast Discovery Component** provides local-network entry-point peer discovery, allowing the Storage or the Client entity to connect to the DFS using the Peer-to-peer connection component. The discovery process is facilitated by each entity through multicast presence advertising in the local network. (Grönvall et all., 1999)

The **Peer-to-Peer Connection Component** has a double purpose in the system.

First, it ensures host discovery outside the local network. From this point of view, it is similar to the Multicast Discovery Component. In order for a host to enter the peer-to-peer swarm, an entry point is defined (section 3.3, Initialization).

Second, it provides file transfer services, being responsible for the data transfer between peers, inside and outside of the local network.

The Peer-to-Peer component is being governed by a set of rules, defining the communication directives between peers and offering data availability for Client Entity.

The Storage Data Balancer (Storage Entity) and the Data Balancer (Client Entity) are similar entities.

The **Storage Data Balancer** (Storage Entity) manages accesses to the shared and dynamic storage repositories, avoiding concurrency problems for the dynamic data and providing availability for the shared data. Besides managing accesses to the other Serving Peers, the Storage Data Balancer is also responsible for executing and replying to data aggregation requests.

The **Data Balancer** (Client Entity) manages accesses to the Serving Peers available in the DFS.

The **Dynamic Storage Component** and the **Permanent Storage Component** on the Storage Entity are designed to manage access to the dynamic and shared file repositories. Multiple such components can be loaded on top of a Storage Data Balancer. Data in the dynamic storage can be changed using external processes.

The **Data Modeling Component** on the Client Entity translates the file system requests to requests in the DFS for shared, dynamic or aggregated data.

The **FUSE Export Component** exports the data in a file hierarchy, making it accessible for the end user.

The main FUSE advantage is providing easy integration with other user file systems. From the user's point of view, there are no differences in the access mode between offline, local files or files located on the distributed system.

Also, the FUSE integration provides compatibility with any GNU/Linux, Free BSD and Open Solaris operating system, as well as with Mac OS X. Thus, the file system can be used by a wide range of users and devices that use the above specified operating systems.

3.3. File system operations

Initialization

The DFS model infrastructure is provided by the cooperation and self-organization of the Storage Entities. As mentioned in section 2.2, even though peer-to-peer connections are possible between each entity, the overlaying network approach is preferred for maintaining the DFS infrastructure. This reduces the number of connections and simplifies propagation of file system updates and client requests.

The model defines what minimal parameters need to be passed on the initialization of an entity in order to obtain the desired entity behavior.

A **Storage Entity** is started by specifying the root of the Persistent Storage and the root of the Dynamic Storage.

```
./dfs-storage <path-to-persistent-storage>
<path-to-dynamic-storage> [peer-entry-point]
```

The storage entity enters the DFS network by using the Multicast Discovery component or by using a peer already in the network. The Multicast Discovery Component makes it easy for the Storage Entity to find another Storage Entity in the same network.

Once a peer is connected, the Storage Entity can get more information about the entire distributed system. A list of neighboring hosts is received from the remote entity through the Peer-to-Peer protocol - a protocol used to interact with these hosts and to discover new ones based on an entry point in the swarm. In order for the Storage Entity to get added to the overlay network used for system management and update notifications, the entity must determine its "parent node" from the list of neighboring hosts.

Client Entities provide access to the resources in the DFS through FUSE. The DFS model provides several methods of mounting the DFS, depending on what type of files need to be accessed. The Client Entity start command syntax is the following:

```
./dfs-client --mountpoint=<directory>
[--type=<file-type>]
[--param=<aggregation-path>]
```

The `<file-type>` can be `"shared"`, `"dynamic"`, `"aggregated"` or any combination of them. If no `--type` parameter is being passed to the mount command, the default option will be `"shared,dynamic,aggregated"`.

If `"aggregated"` is passed, an optional `"--param"` parameter can be used to indicate the subset of aggregation paths to be exported through the mount point.

Using the default value for the `--type` parameter will provide access to the files as described in sections 2.4 and 3.3.

If `"--type=shared"`, the shared files will be available directly in `/mountpoint/<file-path>`, with no access to dynamic files or aggregated data.

If `"--type=dynamic"`, the dynamic files can be accessed directly in `"/mountpoint/<node-name>/<file-path>"`, with no access to shared files or aggregated data.

If `"--type=aggregated"` is used with no `"--param"` option, the aggregated data can be accessed through `"/mountpoint/<range-description>/ <function-name>/<file-path>"`.

If `"--param"` is being used, an aggregation path can be automatically prepended to all paths. For example, if `"--param=<range-description>/<function-name>"` then the function defined by `"<function-name>"` within range `"<range-description>"` will be applied implicitly to all file-paths used as `"/mountpoint/<file-path>"`. This provides a simple method of direct data aggregation without any other requirement or step needed.

Peers

There are two types of peers in the DFS model network, considering the source and the destination of data: Serving and Remote Peers. Both types of peers are usually represented by Storage Entities. **Serving Peers** are entities delivering data from the files in the shared storage or in the dynamic storage. **Remote Peers** are entities that request data from the Serving Peers available in the DFS.

The communication between the peers is based on a simple protocol that uses the messages types described in Table 2.

Table 2. P2P Message Types

Message type	Explanation
ping	Tests basic connectivity and protocol functionality test. The reply message contains information about the peer such as its ID, name and IP address.
error	Signals an error in the protocol. A text describing the error is sent together with an error code.
get_peer_list	Informs the receiver to send back its list of known working peers.
get_file_list	Informs the receiver to send back its list of shared files.

get_file	Informs the receiver to transmit back the content of the specified file.
send_peer_list	Sends the receiver the current list of known working peers.
send_file_list	Sends the receiver the list of shared files
send_file	Sends the receiver the content of the specified file.
find	Returns a list of files that contain the searched string in their name.
delete	Removes the specified file from the local cache. If the file is requested by the user again, it will be copied from the network.
broadcast	This is a meta message. It contains another message in it, message that is being sent to all the nodes in the distributed file system.
hello	Registers to the peer. The reply message contains information about the peer, such as ID, name and IP address.

After building the list of local neighbors, a node gets from each of them a list of peers, and continues discovering the network. When there are no local neighbors or if the number of discovered nodes is too small, the node can enter the peer-to-peer swarm via an entry point.

The entry point can be associated with any node in the network or with some particular nodes that present more powerful hardware and network connection.

The connection management messages (used for example to fetch the list of files from a neighbor or to delete the copy of a file from all the nodes) are sent using the overlay network.

Actual file transfers are done using directly the peer-to-peer swarm. The result of a completed search includes the address of the peers that have the specified file. The current node starts copying the file from one of the peers and, if any problem occurs, another peer is chosen and the file is downloaded again. This represents a known limitation of the model.

Update Notification System

The update messages sent between peers (mostly for management purpose) are managed by the Update Notification System (UNS).

The UNS uses the overlay network to send data to each host in the network and run as part of the Peer-to-Peer Connection Component.

Basics

All operations for a file are executed on a file descriptor (FD). The same FD is used both by the Serving and the Remote Peer. This FD is used in all the file system operations just like in a local file system (Bishop, 1996). The main difference between the FD in the DFS and the FD in a local file system is the mapping procedure used for associating file descriptors with file resources. This procedure ensures fault tolerance and availability depending on the type of storage being used and the type of access. The Table 3 displays the FD mappings depending on access type and storage type.

Table 3. FD Mappings

Access \ Storage	Shared Storage	Dynamic Storage
read	FD points to a local file copy	FD points to a local file copy
write	FD points to a local file copy	FD points to a remote file replica
read & write	FD points to a local file copy	FD points to a remote file replica

For the Shared Storage, the Remote Peer creates a local file copy from the possible Serving Peers (entities serving the same file) – this copy is also part of the cache. The FD returned by the open function points to the local file copy. After the local file copy is realized, the Serving Peer becomes the local entity.

For the Dynamic Storage, the Remote Peer receives a FD from the Serving Peer. No copies or caches are being created for the dynamic data. If the dynamic file is being accessed for writing, the Serving Peer creates a replica of the file and provides the Remote Peer a FD pointing to that replica. On closing, the replica will be served instead on the local file. After all the file descriptors pointing to the old file are closed, the old dynamic data file gets deleted. Multiple replicas might exist at the same time on the Serving Peer. The last closed replica will be used for future read accesses and will be used as a base for new replicas (in the case of future write accesses).

Shared data files operations

create. When a new file is created, it is first created in the local repository. Using the Update Notification System all the other peers are notified of the new entry in the file system and add it to their file list.

open. When a file is opened, the file is first searched in the list of locally available files. If it is found, it is served from the local cache. Otherwise, it is copied from another host in the network. Then the local copy of the file is being opened.

write. Changes to files are made to their local copy. Afterwards, the changes are announced using the Update Notification System to all other peers and peers start updating their copies.

read. The read operation is performed on the local copy of the file.

close. The operation is performed on the local copy of the file

seek. The operation is performed, as well, on the local copy of the file

unlink. When a file if being unlinked, the local copy of the file is removed. Afterwards, all entities will be announced about the unlink using UNS.

Dynamic data files operations

create. Creating a dynamic file in the DFS model must be done for a specific node - therefore, a node path must be used.

open. When opening a dynamic file, the Remote Peer asks for read / write / read&write access to the file. The Serving Peer checks for the existence of the Dynamic File in its Dynamic Storage. If the file is opened for writing, the Serving Peer creates a second replica of the file to be used for this particular access, otherwise the existing file is used and its reference count is increased.

write. When writing a dynamic file, all writes performed by the Remote Entity are being passed to the Serving Peer. The Serving Peer performs all accesses on the replica associated on opening the dynamic file.

read. Reading a dynamic file is performed by the Serving Peer. The Remote Peer receives the read data from the Serving Peer.

close. The close operation is being sent to the Serving Peer. If there is a new replica of the dynamic file and there are no longer references for the closed dynamic file, the dynamic file is being removed from the system.

seek. Seeking a dynamic file is performed on the Remote Peer.

unlink. Unlinking a dynamic file is performed by the Serving Peer directly in the Dynamic Storage.

4 DESIGN LIMITATIONS

The current design of the distributed file system model presents some limitations, addressing availability and data access.

The peer-to-peer component does not download data simultaneously from multiple peers. The main focus was not on the peer-to-peer optimizations, but on the component integration.

Security was not one of the issues considered in this stage of the design model. In the current design, the model trusts all the peers and all the users about the data they write and/or delete. Also, access rights among peers are usable only if the same users exist on the communicating nodes (requires having the same user IDs on the nodes).

Data availability is limited by the peer availability. Data is not replicated in order to prevent lack of availability. Heuristics can be used in order to improve availability based on previously recorded file usage patterns.

5 RELATED WORK

The large interest in grid computing has led to the development of multiple Distributed File Systems with the main purpose of efficiently and reliably sharing the data between multiple networked peers. Distributed File Systems usually feature fault-tolerance and parallel striping of for the served data, depending on the purpose of the file system. The file system model presented in this paper provides fault-tolerance by replicating files on usage. In order to provide consistency in the underlying storage file system, data is not being striped but replicated at file-level. Widely used fault-tolerant file systems include Google File System (Ghemawat et. all, 2003), Hadoop (Borthakur, 2007) and GlusterFS. These file systems assume the infrastructure is stable and not loosely coupled, as it is in the peer-to-peer networks.

Several peer-to-peer file systems have been implemented by now, mostly focusing on leveraging the distributed storage under the easy-to-use namespace provided by a file system.

CFS (Dabek et. all, 2001) is a read-only, highly robust and efficient file system based on distributed hash tables for block storage. (Muthitacharoen et al., 2002) presents a multiuser read/write peer-to-peer file system. Modifications made by the users are only saved locally, while data search is always made on all network nodes. This allows the file system to maintain metadata consistency without locking.

Peer-to-peer file systems were also researched for mobile ad-hoc networks by Kleem in (Klemm et al., 2003). Due to host mobility in this particular case, an overlay network is being used on demand by the search algorithm.

IgorFS (Amann et. all, 2008) is a distributed peer-to-peer file system built on top of the Igor overlay network with specific support for files subject to frequent but minor modifications.

The main difference between the model presented in this paper and the above distributed file systems is the transparent support for data aggregation across the peer network and support for both shared and dynamic data.

6 FUTURE WORK

Future development will be focused on evaluating the performance of the proposed design model and removing limitations that impact the overall system capabilities.

Overlay networks are commonly used for organizing loosely coupled environments, such as peer-to-peer networks. Several designs for overlay networks have been researched (Lua et. all, 2005). In order to improve the performance of the designed distributed file system, a structured overlay

network model that leverages the locality attributes of each peer should be evaluated and implemented.

Having this set up, a significant improvement would be to start a file replication process as soon as a file is opened. This would bring the entire contents of the file close to the opening-entity (possibly in multiple geographically closed peers) and would increase fault-tolerance by generating multiple replicas.

Other future work is related to stream-reading - this process would read the file as it is received by the reading entity. Seeking in this situation will require a specific data block caching algorithm that would download the blocks where seeking is being frequently performed.

7 CONCLUSIONS

The distributed file system design model presented by the paper provides new facilities of sharing data across a peer-to-peer network. It has numerous applications such as simple peer-to-peer file exchange, distributed system monitoring and dynamic data publication. Data is made available to high-level applications through the FUSE file system, making it accessible to all types of applications through common file system operations.

The purpose of the design is to provide a solution for storing data in the distributed loosely-coupled environment given by peer-to-peer networks. It also takes advantage of the peer-to-peer infrastructure to provide access to dynamic data published by the peers. Dynamic data can be aggregated in the system and served to the requesting nodes through an overlay network designed to minimize the number of connections and to speed up traffic. The model offers a flexible interface for querying localized or global dynamic data available in the system, making it well suited for environment and network status discovery applications.

REFERENCES

Amann, B. and Elser, B. and Houri, Y. and Fuhrmann, T. and Munich, G. 2008, IgorFs: A Distributed P2P File System. In *Proceedings of the 2008 Eighth International Conference on Peer-to-Peer Computing-Volume 00*, pp. 77-78, IEEE Computer Society Washington, DC, USA

Andersen, D., Balakrishnan, H., Kaashoek, F., and Morris, R. 2001. Resilient overlay networks. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles* (Banff, Alberta, Canada, October 21 - 24, 2001). SOSP '01. ACM, New York, NY, 131-145

Androutsellis-Theotokis, S. and Spinellis, D. (2004). A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.* 36, 4 (Dec. 2004), 335-371.

Bishop, M, Dilger, M, (1996) Checking for race conditions in file accesses, *Computing Systems*, 1996, vol 9, pp. 131-152

Borthakur, D. (2007), The Hadoop distributed file system: Architecture and design, *Retrieved from lucene. apache.org/hadoop*

Burns, Randal C., Rees, Robert M., Long, Darrell D. E. (2000), Safe Caching in a Distributed File System for Network Attached Storage, *Parallel and Distributed Processing Symposium, International*, vol. 0, no. 0, pp. 155, 14th International Parallel and Distributed Processing Symposium (IPDPS'00).

Dabek F., Frans Kaashoek M., Karger D., Morris R., Stoica I. (2001), Wide-area cooperative storage with CFS, *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, October 2001

Dean, J., Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications ACM* 51, 1 (Jan. 2008), 107-113.

Ghemawat, S. and Gobioff, H. and Leung, S.T. (2003), The Google file system, *ACM SIGOPS Operating Systems Review vol. 37, no. 5*, pp. 29-43, ACM New York, NY, USA

Grönvall, B., Westerlund, A., and Pink, S. (1999). The design of a multicast-based distributed file system. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation* (New Orleans, Louisiana, United States). Operating Systems Design and Implementation. USENIX Association, Berkeley, CA, 251-264.

Klemm, A.; Lindemann, C.; Waldhorst, O.P. (2003), "A special-purpose peer-to-peer file sharing system for mobile ad hoc networks," *Vehicular Technology Conference, 2003*. VTC 2003-Fall. 2003 IEEE 58th , vol.4, no., pp. 2758-2763 Vol.4, 6-9 Oct. 2003

Lua, E.K. and Crowcroft, J. and Pias, M. and Sharma, R. and Lim, S. (2005), A survey and comparison of peer-to-peer overlay network schemes, *IEEE Communications Surveys & Tutorials vol. 7, no. 2*, pp. 73-93

Muthitacharoen, A., Morris, R., Gil, T. M., and Chen, B. (2002). Ivy: a read/write peer-to-peer file system. *SIGOPS Oper. Syst. Rev.* 36, SI (Dec. 2002), 31-44

Szeredi, M. (2005). Filesystem in userspace. *fuse.sourceforge.net*