Scalable Computing: Practice and Experience
Volume 11, Number 3, pp. 305–325. http://www.scpe.org

ISSN 1895-1767
© 2010 SCPE

WIDE AREA DISTRIBUTED FILE SYSTEMS—A SCALABILITY AND PERFORMANCE SURVEY

KOVENDHAN PONNAVAIKKO* AND JANAKIRAM DHARANIPRAGADA*

Abstract. Recent decades have witnessed an explosive growth in the amounts of digital data in various fields of arts, science and engineering. Such data is generally of interest to a large number of people spread over wide geographical areas. Over the years, several Distributed File Systems (DFS) have, to varying degrees, addressed this requirement of sharing large amounts of data, stored in the form of files, among several users and applications. Scalability and performance are two important measures that determine the suitability of a file system for the applications executing over them. We perform a detailed comparative analysis of popular distributed file systems in terms of these measures in our survey.

1. Introduction.

In recent decades, we have been witnessing increasingly large rates of data generation and growing numbers of widely spread collaborative applications. For example, data requirements of *High Performance Computing* (HPC) applications have been continuously growing over the past few years and are expected to grow even more rapidly in the years to come [23]. Experimental setups, deployments of sensors, simulators, agents, etc. generate large amounts of data which researchers world over can have use for. Other examples include WikipediaFS [10], and large scale telemedicine [24].

Organizing and sharing raw and processed data files owned by different users and groups calls for the need of large scale *Distributed File Systems* (DFS) [46] [7] [8].

Any file system that allows files to be placed across the network and yet make accesses appear local is a distributed file system. Certain systems are *Client-Server* based (*Asymmetric*) in that dedicated servers exist to provide file services. In *Peer-to-Peer* (P2P) or *Symmetric* file systems, data/metadata management load is distributed among all the nodes. *Clustered* file systems are those in which the data/metadata server is replaced by a cluster of servers to better distribute load and handle failures. A *Parallel* file system enables concurrent reads and writes of the same file and parallel I/O [22]. Some parallel file systems support the striping of a file across multiple storage devices.

There exist several large scale distributed file systems. For our survey, we consider a set of popular production systems and research prototypes (table 1.1)[1]. This set has been chosen so as to cover the major architectural variations of existing systems.

These systems vary in terms of their typical application workloads and the geographical spread of their typical usage. For example, some of them are designed for desktop workloads and some for scientific applications. Some of the analyzed systems are not designed to be wide area file systems, i. e., clients and servers are not designed to be geographically spread across *Wide Area Networks* (WAN). However, other features such as high scalability have prompted researchers to adapt even such systems for use across WANs. Some examples include the usage of Lustre file system in [42] and Parallel Virtual File System 2 in [5].

Keeping in mind the common nature of new generation applications, we analyze the architectures of these systems with respect to the following application requirements. The first requirement is that of scalability with respect to the number of nodes and files. In other words, increasing the number of nodes and/or files must not adversely affect query/access times. The other major requirement is that of maintaining high application performance. For HPC applications, performance can be measured in terms of makespan, computation or I/O throughput, etc. In file systems maintained for home directories and such, performance can be measured in terms of query response latencies, file access/update times, and so on.

Using a few system parameters, we attempt to characterize the effects of increasing query and I/O loads on individual file system servers. We also study the support provided by the different systems for sophisticated data placement and migration strategies, which are critical for high application performance. In section 2, we discuss some of the design considerations in the context of large scale DFSs. Section 3 summarizes the system architectures of the various DFSs analyzed in this survey. The comparative analysis is presented in section 4.

*Distributed and Object Systems Lab, Department of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai, India

[1]An extensive list of computer file systems can be found at [3]. Comparisons of general and technical features of a large number of file systems can be found at [2].

Table 1.1
Set of Analyzed File Systems

Andrew File System
Ceph
Common Internet File System
Edge Node File System
Farsite
Google File System
Ivy
Lustre File System
OceanStore
Panasas Parallel File System
Pangaea
Parallel Virtual File System 2
WheelFS

Table 2.1
Classification of the Analyzed File Systems

Category	**Name**	**Systems**
I	Traditional Distributed File Systems	Andrew File System, Common Internet File System
II	Asymmetric Cluster File Systems	Ceph, Google File System, Lustre File System, Panasas Parallel File System, Parallel Virtual File System 2, WheelFS
III	Self-Organizing P2P File Systems	Edge Node File System, Farsite, Ivy, OceanStore, Pangaea

2. Design Considerations. Traditionally, distributed file system designers have adopted a client-server model. In these asymmetric systems, dedicated servers exist to provide file services and clients only consume the services. Typically, the server exports hierarchical namespaces and clients mount the exported hierarchies in their local namespaces.

A client-server approach has several advantages such as ease of maintenance, efficient management of concurrent reads and writes of the same file, and centralized security control. However, the presence of a centralized server presents significant scalability constraints. File system performance degrades with increasing file sizes, and increasing numbers of files and users.

One of the early approaches to improve file system performance is client side caching. While caching helps in reducing network traffic, it also introduces consistency issues. Cached content can become stale and write collisions can occur, especially in file systems with stateless servers.

In later distributed file system designs, a multitude of strategies have been employed to address issues related to scalability. Individual servers have been replaced by clusters of servers. Analogous to *Sharding* in databases, in such file systems, namespaces are partitioned and distributed among the different servers in the cluster. This helps in the distribution of load and hence better performance.

Another effective strategy is to decouple data management from metadata management. While data refers to the actual content of files, metadata in the context of file systems refers to the data about file contents. Unlike data operations, metadata operations are usually small, random and non-sequential.

Decoupling is achieved by using different sets of servers for data and metadata management. In a typical file system, a large proportion of queries are related to file metadata. On the other hand, responses to data access queries are much more voluminous. Using different sets of servers for managing data and metadata therefore helps improve system performance. Clustering and decoupling data and metadata have enabled other scalability and performance optimizing strategies such as replication and striping a file's content across multiple storage devices.

DFS features such as concurrent access, file striping and replication complicate the task of presenting a consistent view of the file system to all users. Concurrent accesses can be controlled by associating data and

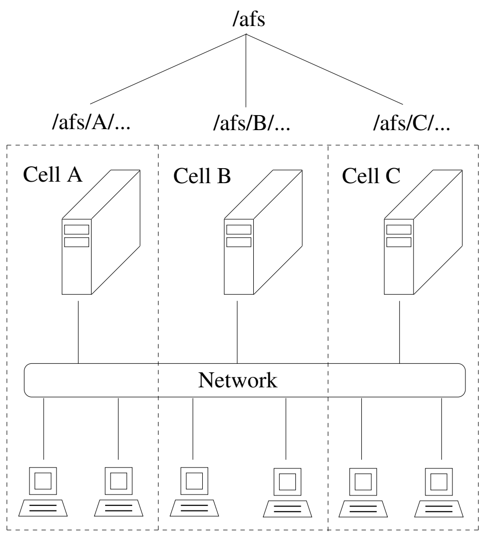

FIG. 3.1. *AFS System Architecture*

metadata with different kinds of locks. In UNIX, the two common locking mechanisms, *fcntl* and *flock*, allow *Exclusive* and *Shared* locks to be applied to files/blocks. All exclusive locks must have been released before shared locks can be obtained by clients and all kinds of locks (shared and exclusive) must be released before an exclusive lock can be obtained.

While pessimistic approaches such as locking allow file systems to support *Strict Consistency Semantics*[2], they also affect application performance by increasing messaging overheads and wait times. Certain file systems support weaker consistency semantics by allowing concurrent accesses in conflicting modes. In such systems, applications either ensure that colliding accesses do not occur, or have appropriate conflict resolution mechanisms in place.

High availability of data and metadata is usually a crucial requirement of distributed file systems. Several approaches exist to improve a file system's availability, each associated with certain overheads. Some of the approaches are replication, caching, versioning, logging, and anticipatory reads. Different systems employ different combinations of these techniques to achieve the required levels of availability.

Though clustered file systems are more scalable than traditional client-server systems, their scalability is limited because of the manually maintained set of server clusters. A central augmentable set of servers has other drawbacks too. Clusters are expensive to set up and maintain. Storage of entire file systems in a limited number of sites makes access from distant locations inefficient as a result of high network latencies. Moreover, such setups create single points of failure, and are prone to physical vulnerabilities.

Increasing rates of data generation and number of collaborations among geographically distributed groups of users have created the need for *Global* and P2P file systems. P2P systems involve minimal or no central coordination. In P2P or symmetric file systems, data and metadata management load is distributed among all the nodes in the system. These systems are generally designed to be self-organizing due to the impracticality of manually administrating huge numbers of storage/compute resources.

Based on the different evolutionary stages of DFS design, we classify the analyzed systems into the categories of *Traditional Distributed File Systems*, *Asymmetric Cluster File Systems* and *Self-Organizing P2P File Systems* (table 2.1).

3. System Architectures.

In this section, we present brief independent reviews of the system architectures of the considered file systems.

3.1. Traditional Distributed File Systems.

Though *Network File System* (NFS) [39] (up to version 3) is one of the most commonly used distributed file system protocols, it is usually used in a local area network or within a single administrative domain. We have therefore not included NFS in this survey. Influenced by

[2]A read returns the most recently written value.

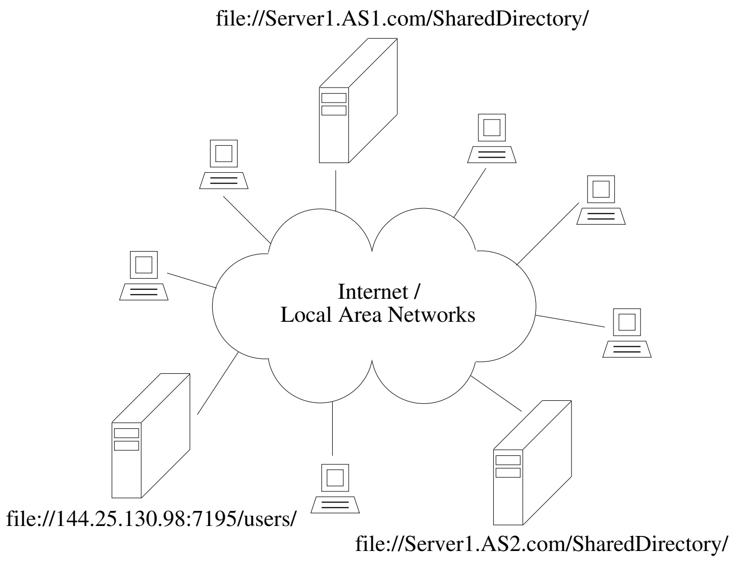
file://Server1.AS1.com/SharedDirectory/
Internet /
Local Area Networks
file://144.25.130.98:7195/users/
file://Server1.AS2.com/SharedDirectory/

Fig. 3.2. *CIFS System Architecture*

Andrew File System [21] and Common Internet File System [28], version 4 of NFS [43] supports stateful servers and locks, includes other performance improvements and can be used in wide area networks.

3.1.1. Andrew File System (AFS). Started at the Carnegie Mellon University, AFS [21] uses a set of trusted servers for sharing a common directory structure among several thousand client machines. AFS relies on data caching to address the issue of scalability. While earlier versions of AFS required clients to fetch whole files, versions since AFS 3 support the transfer of smaller blocks of files.

Servers maintain state about clients which have files open. *Callbacks* are used to maintain the consistency of cache contents. Whenever file contents are altered, servers send invalidation messages to the corresponding clients. A client, on the other hand, informs the server about the changes that it has made only at the time of closing. As a result, AFS only supports *Session Semantics*[3] and not *One-Copy Update Semantics*[4], which is supported by UNIX.

The AFS model (figure 3.1) comprises of a set of cells, each cell usually being a set of hosts with the same Internet domain name. Each cell has servers executing the *Vice* process and clients executing the *Venus* process. AFS provides location independence by performing the mapping between filenames and locations at the servers. The hierarchical directory structure is partitioned into *Volumes*, which act as containers for related files and directories. Volumes can be transparently migrated between servers. Read-only cloned copies of volumes must be created by administrators to enable recovery in the case of failures. The Kerberos [44] protocol is used for the mutual authentication of clients and servers.

3.1.2. Common Internet File System (CIFS). CIFS [28] is Microsoft's version of the *Server Message Block* (SMB) protocol along with certain other protocols. CIFS provides remote file access over the Internet (figure 3.2) with features such as global naming, caching, volume replication, remote sharing and locking. SMB uses flat namespaces to address files and CIFS makes use of the Internet naming system, *Domain Name Service* (DNS). While changes in file addresses are difficult to propagate in SMB, CIFS uses the scalable notification system of DNS to handles such changes. Unlike several other wide area file systems, *Unicode* filenames are supported.

Parallelism is supported at the directory level only and individual files cannot be split among multiple servers. Since each file/directory must be associated with particular servers and servers are manually administered, scalability with respect to installations and query/data transfer loads in CIFS is limited.

[3]Changes made to a file are visible to the other clients only after the writing client closes the file.

[4]In one-copy update semantics, every read sees the effect of all previous writes and a write is immediately visible to clients who have the file open for reading.

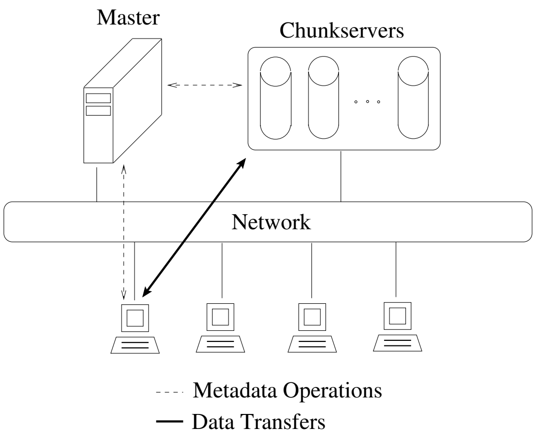

FIG. 3.3. *GoogleFS System Architecture*

3.2. Asymmetric Cluster File Systems. There are different kinds of storage architectures that distributed file systems use. Traditional distributed file systems discussed in section 3.1 such as NFS, AFS and CIFS adopt a *Network-Attached Storage* (NAS) architecture. Servers in these systems provide file-based access to their dedicated storage devices, to clients across networks.

In the *Storage Area Network* (SAN) architecture, large storage devices such as arrays of disks are shared by a cluster of nodes. Unlike NAS, data access is block-based (finer granularity), which results in increased flexibility in storing huge files. SAN based file systems translate file-level operations to block-level operations at the client. Metadata management is either handled by a central server or distributed among the cluster nodes.

IBM's *General Parallel File System* (GPFS) [18] is an example for a clustered file system that adopts the SAN architecture. GPFS uses a distributed token management system to handle concurrent file accesses among cluster nodes. It also supports data sharing among multiple GPFS clusters.

Another storage architecture employed by several clustered file systems such as Lustre [40], Panasas [50] and Ceph [48], uses *Object-based Storage Devices* (OSD). OSDs are evolved disk drives that can directly handle the storage and serving of objects as against normal disk drives which work at the level of bits, tracks, and sectors. In other words, an OSD handles lower level functionalities related to object management within the device and exposes object access interfaces to applications.

In block-based file systems, file metadata, which includes block locations, is managed by the file system. As a result, performance is effected for large files since metadata sizes are also large. On the other hand, OSD based file systems manage objects only. The lower level details about content striping are handled by the storage devices themselves. This results in improved performance and throughput.

Several client applications benefit from moving computation to where the data is, instead of getting the content transferred to the clients [36] [47]. For such applications, performance depends on the intelligence of OSDs [17], in terms of their ability to execute user specified computations, as well as on their processing power.

3.2.1. Google File System (GoogleFS). GoogleFS [19] is a DFS for data intensive applications, custom-built for the application workload and technical environment at Google. A GoogleFS cluster comprises of a single *Master* and several *Chunkservers*, as shown in figure 3.3.

The master manages the metadata and the chunkservers store the data. The master uses *Heartbeat* messages to periodically monitor the chunkservers. A *Shadow* master is maintained in order to handle the failure of the primary master. Files are split into fixed size chunks. A certain number of replicas (three is the default number) of the chunks are stored in the chunkservers. Chunk replicas are spread across racks to maximize availability.

The master maintains information about the location of each chunk and access control information. The master performs periodic re-balancing of data to ensure that the chunkservers are uniformly loaded at all times. Clients obtain file metadata from the master and perform all data related operations at the chunkservers.

The datasets that applications at Google work with are usually huge in size and the workload primarily involves append operations. Hence, GoogleFS supports record append operations only and not random write operations. Servers are stateless and clients do not cache data in GoogleFS. That is because applications at

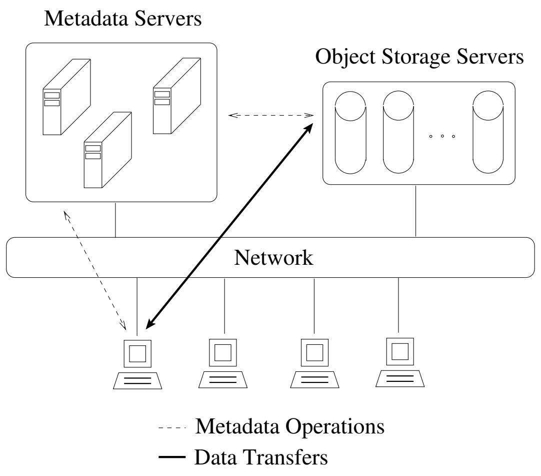

Fig. 3.4. *Lustre System Architecture*

Google usually require certain operations to be performed on file contents and only the result to be returned to them. In fact, the predominant class of application is *MapReduce* [16].

The architecture of GoogleFS makes it suitable for a specialized set of workloads only. Also, its centralized master can become a performance bottleneck, especially for metadata intensive workloads. *Hadoop Distributed File System* (HDFS) [13] is an open source Java product with almost the same architecture as that of GoogleFS.

3.2.2. Lustre File System. Lustre [40] is an object based DFS used primarily for large scale cluster computing. It is a production system used in several HPC clusters. The system architecture of the Lustre file system is shown in figure 3.4. The system comprises of three main components, namely, file system clients, *Object Storage Servers* (OSS) which provide file I/O services, and Metadata servers (MDS).

Typically, the above three components are on independent nodes which communicate over the network. Using an intermediate network abstraction layer, Lustre supports multiple network types such as *Ethernet* and *Infiniband*. Redundancy, in the form of an active/passive pair of MDSs and active/active pairs of OSSs, helps Lustre maintain high availability.

Lustre enforces strict consistency semantics, using locks to enforce serialization. It also uses the *Journaling File System Technology*[5] to prevent data/metadata corruption due to system failures and to enable persistent state recovery.

Since metadata servers as well as object storage servers need to be manually administered, Lustre does not scale transparently.

3.2.3. Panasas Parallel File System. Panasas [50] uses parallel and redundant access to OSDs to provide a high performance DFS. At a high level, the system model of Panasas is similar to that of the Lustre (figure 3.4).

The Panasas object storage nodes have a *Blade* architecture, each blade comprising of disks, a processor, memory, and a network interface. Thus, adding storage capacity includes the addition of the required computing power to efficiently manage the new disks. The storage blades use a specialized file system which implement the object storage primitives. A per-file *RAID* system [32] is used to provide for data integrity and scalable performance.

The storage blades are managed by a set of *Quorum-based* cluster managers. The set of managers maintains the replicated system state using a quorum-based voting protocol. Managers stripe file contents across the OSDs. They also handle multi-user access, consistent metadata management, client cache coherence, and recovery from client and OSD failures. *Transaction Log Replication* protocol is used to tolerate metadata server crashes.

3.2.4. Parallel Virtual File System, Version 2 (PVFS2). PVFS2 [4] is an open source DFS that provides high performance and scalable file system services for large node clusters. Each cluster node can be a

[5]Maintains logs of impending changes before committing them to the file system.

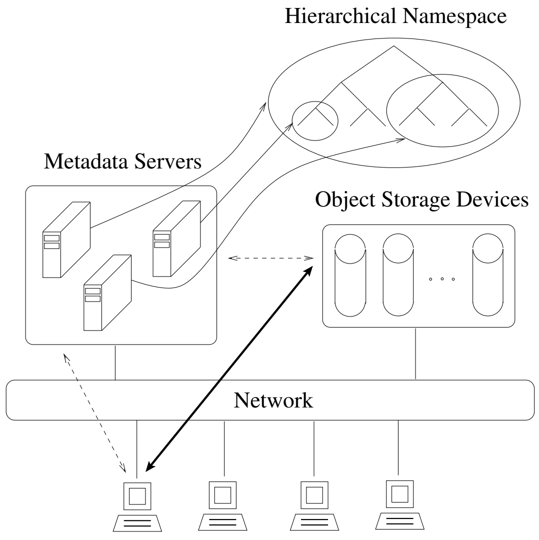

FIG. 3.5. *Ceph System Architecture*

server, a client, or both. Like several other clustered file systems, PVFS2 also supports the striping of a file's data across several storage nodes. PVFS2 allows for a subset of the servers to be configured as metadata servers.

PVFS2 servers are stateless and as a result, locks are not supported. Client failures thereby do not affect the system in anyway. While this lets the system scale to a large number of clients, it results in little support for different kinds of access semantics. While PVFS2 provides atomicity guarantees for updates to non-overlapping portions of a file, simultaneous writes to overlapping regions can result in inconsistent file states.

New file/directory creation is performed by first creating the data object and the corresponding metadata object, and then making the metadata object point to the data object, and finally creating a directory entry pointing to the metadata object. This way, the file system remains in a consistent state always. This mechanism can result in significant amounts of clean up load in case of collisions, i. e., in case of simultaneous updates to the same portions of the namespace.

PVFS2 specializes in supporting flexible data distribution as well as flexible data access patterns. For example, it supports access to non-contiguous portions of a file in a single operation. In that sense, PVFS2 implements *MPI-IO Semantics* closely.

Like Lustre, PVFS2 uses an intermediate layered interface to support multiple network types. Traditional solutions for high availability, such as those used by Lustre, can be used in PVFS2. An experimental comparison of PVFS2 and Lustre for large scale data processing is presented in [41].

3.2.5. Ceph. Ceph [48] is an object-based distributed file system designed to provide high performance, reliability and scalability. *Dynamic Subtree Partitioning* and the distribution of objects using a pseudo random function, are a couple of its unique features. The system (figure 3.5) comprises of clients, OSDs and a metadata servers cluster.

Ceph completely does away with allocation lists and *inode* tables. Instead, a pseudo random function called *CRUSH* [49] is used for the distribution of objects among the OSDs. Clients can therefore calculate the location of file objects instead of performing a look-up.

Some file systems use static subtree partitioning to delegate authority for different subtrees of a hierarchical namespace to different metadata servers. Another approach uses hash functions to distribute metadata. While the first approach cannot handle dynamic loads efficiently, the later approach does away with metadata locality. Ceph uses a dynamic subtree partitioning strategy, in which responsibilities for different subtrees of the namespace are dynamically distributed among the MDSs. The distribution ensures that server loads are kept balanced with changing access patterns. Popular portions of the namespace are also replicated on multiple servers.

Ceph replicates data using a variant of the *Primary-Copy Replication*[6] technique to maintain high availability. The usage of CRUSH rules out the possibility of considering specific node characteristics while making

[6]One of the replicas, which is made the primary copy, serializes transactions and sends updates to the secondary replicas.

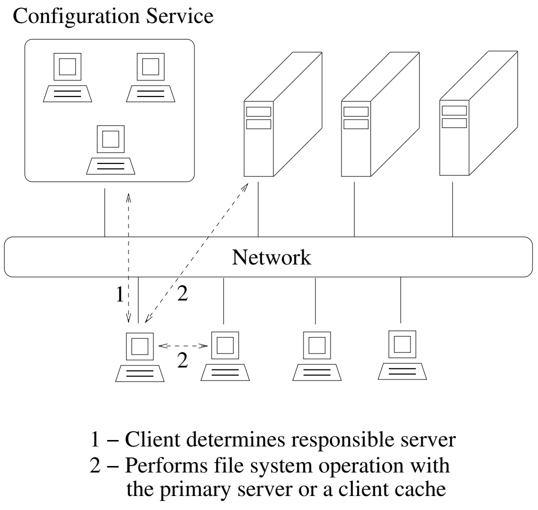
Configuration Service
Network
1
2
2
1 – Client determines responsible server
2 – Performs file system operation with the primary server or a client cache

FIG. 3.6. *WheelFS System Architecture*

object placement decisions. In wide area installations, the average network latency between clients and Ceph's metadata servers can be high, affecting the performance of applications involving large proportions of metadata operations.

3.2.6. WheelFS. WheelFS [46] provides applications control over replica placement, consistency and failure handling mechanisms using *Semantic Cues*. The system allows applications to manage the trade-off between the immediacy of update visibility and the independence of client sites to operate on the data. A set of WheelFS servers (figure 3.6) store file and directory objects. Each file/directory has a primary server which holds its latest content. Clients also maintain local caches of the files accessed. By default, WheelFS uses strict *Close-to-Open Consistency Semantics*[7], with the primary server being responsible for serializing operations.

Semantic cues can be used to specify application policies with respect to placement, durability, consistency and large reads. To reduce the effects of network latency, data can be placed close to clients that are likely to use the data. Files can be clustered together to optimize the performance of operations that access multiple files, and replication levels can be specified.

The system can be adjusted to wait for only a specified number of replicas to be created or updated before acknowledging a client's new file or file update request respectively. This helps in achieving quicker response times even in the presence of slow servers. Consistency related cues allow clients to specify time-out periods for remote communications corresponding to file system operations. Applications can also use the *Eventual Consistency Semantics*[8] to improve availability.

Also, a client can prefer to read stale copies of files when the primary servers are hard to reach. While reading large files, clients can choose to prefetch entire files into its local cache. Cues also enable clients to obtain file contents from multiple cached sources in parallel to reduce the load on the primary server.

A *Configuration Service*, maintained as a replicated state machine at multiple sites, is used by clients to learn about the servers responsible for the different objects. Based on the first S bits of the object identifier, the identifier space is split into 2^S slices. The configuration service maintains a mapping between slices and the primary and replica servers responsible for the slices.

While resource location aware data placement is supported, WheelFS does not provide resource characteristics aware data placement. The configuration service, maintained as a replicated state machine, can be a bottleneck for large system sizes and heavy query loads.

3.3. Self-Organizing P2P File Systems. In P2P systems, every node is both a supplier and consumer of resources. Some of the benefits of such an architecture are distribution of load among all the peers, increased robustness, and lack of a single point of failure. On the other hand, high system dynamics is one of its major

[7]When A opens a file after B has modified and closed it, A is guaranteed to see B's updates.
[8]If no new updates are made, the latest updates will propagate through the system eventually and make all the replicas consistent.

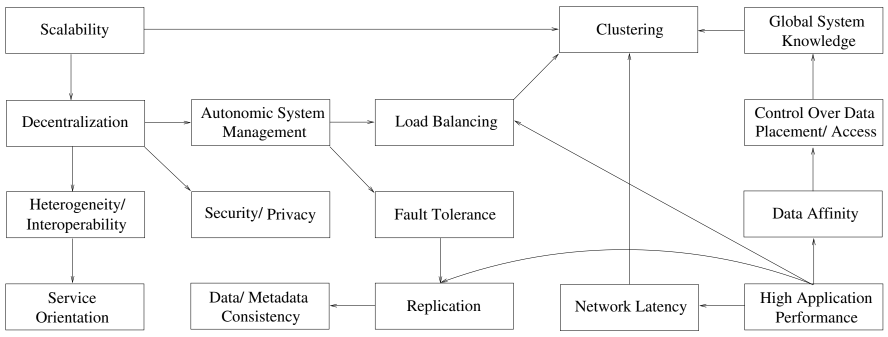

FIG. 3.7. *System Requirements of P2P File Systems (A → B represents B supports A)*

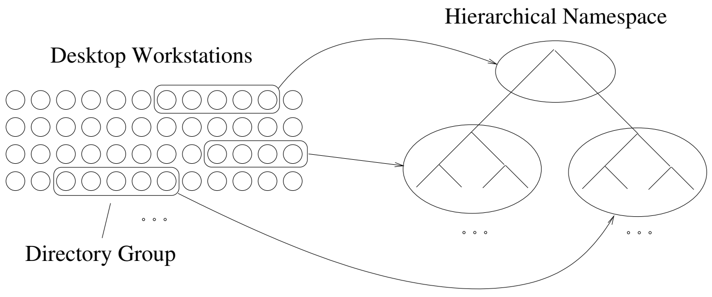

FIG. 3.8. *Farsite System Architecture*

drawbacks. In P2P file systems, peers share the load of file storage and metadata management. Figure 3.7 shows some of the system requirements of P2P file systems. As discussed earlier, scalability and high application performance are the two primary requirements under consideration.

It is well known that decentralization of control and autonomous system management are central to the design of scalable distributed systems. In such systems, load balancing and resource discovery are complex tasks because of the lack of any central entity with knowledge about the entire system.

However, awareness of resource characteristics and locations while placing file replicas is critical for achieving high application performance. That is because network bandwidth and latency concerns dictate that data and metadata be placed in proximity to where they are consumed. Achieving a trade-off between these conflicting requirements of decentralization and system awareness is an important design consideration, especially in the case of P2P file systems. One of the approaches to achieve the trade-off is to design the system as a federation of manageable clusters.

3.3.1. Farsite. Farsite (Federated, Available, and Reliable Storage for an Incompletely Trusted Environment) [6] [12] is a DFS from Microsoft Research built over a network of unstructured desktop workstations. Farsite provides high file availability and security utilizing the unused storage space and processing power of a large number of nodes. Issues of security and trust are addressed using *Public-Key Cryptographic Certificates* such as namespace, user and machine certificates. Users and directory groups authenticate each other before performing file system operations.

File contents are encrypted and replicated and the corresponding metadata are managed by *Byzantine-Replicated* finite state machines [33]. Farsite provides hierarchical directory namespaces, each namespace having its own root. Roots are maintained by a designated group of nodes. Directory groups can split to distribute metadata management load. Splitting can happen by randomly selecting a group of nodes and designating a portion of the namespace to them (figure 3.8).

Content hashes of files are stored in the corresponding directory groups to maintain file integrity. Different kinds of leases are issued on files to clients. Caching is used for improving access times and reducing network load. Updates made to files are not immediately propagated to all the replicas. Instead, a lazy propagation mechanism is employed in order to improve performance.

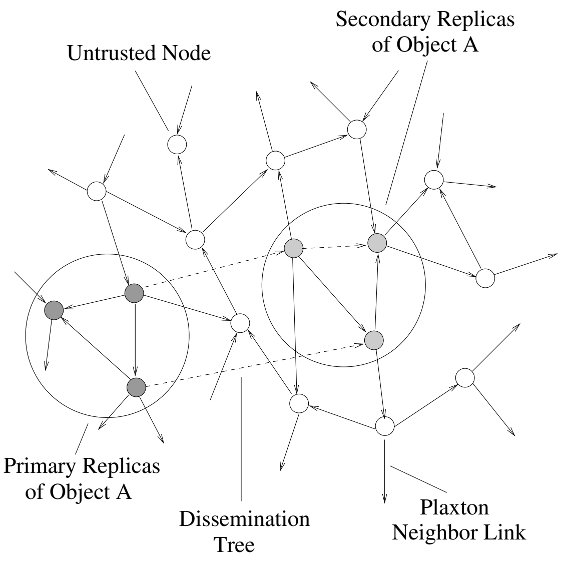

Fig. 3.9. *OceanStore System Architecture*

As with other hierarchy traversal systems, locating the directory group for a file deep in the hierarchy may require several hops, thus making metadata access expensive. In systems with high churn rates, group membership can keep changing, resulting in high group management overheads.

3.3.2. OceanStore. OceanStore [26] is a global scale data storage utility that uses untrusted infrastructure. The primary objective is to provide continuous access to persistent information.

Each object in OceanStore is assigned a unique global identifier and is replicated and stored in a set of servers. A few of the servers in the high connectivity and high bandwidth regions are made primary replicas and the rest are made secondary replicas (figure 3.9). Updates made to the objects are ordered by the primary replicas using a *Byzantine Fault Tolerant* algorithm [14]. Secondary replicas communicate with the primary replicas and among themselves to propagate updates in an epidemic manner. Every update results in the creation of a new version which is archived in the system, making the system inefficient for large sized files.

Each object is associated with a root node in the system which holds information about the object's replica locations. A variation of *Plaxton*'s randomized hierarchical distributed data structure [34] is used by nodes to reach the root of any object in $O(logN)$ hops, where N is the number of nodes in the system. A probabilistic algorithm using attenuated *Bloom Filters* [11] is also used to rapidly locate objects if they are in the local vicinity.

The policy of *Promiscuous Caching* which allows files to be replicated in any node in the system makes OceanStore highly scalable. However, the overheads involved in the maintenance of two tiers of nodes and a dissemination tree for each data object can be high. High churn rates among the primary tier nodes can also result in expensive maintenance overheads. Maintenance of Bloom filters and the Plaxton data structure at each node can result in high network usage.

3.3.3. Ivy. Ivy [31] is a P2P read/write file system based on logs. Each participant maintains a log with information about all the changes made to the files in the system by the participant. The logs of all the participants need to be parsed to be able to get the current state of a file. Updating a file's contents however requires an append to the participant's log only. Ivy uses *DHash* [1] as the *Distributed Hash Table* (DHT) [45] for storing all its logs and, as a result, all its data. The set of all logs in the file system is referred to as *View* (figure 3.10).

A participant's log is a linked list of log records. The log-head points to the most recent entry. Content hashes are used as keys for storing log records in DHash. The public key of a participant is the key for a log-head. The log-head is digitally signed by the participant's private key. The digital signatures and content hashes help ensure the integrity of logs in Ivy.

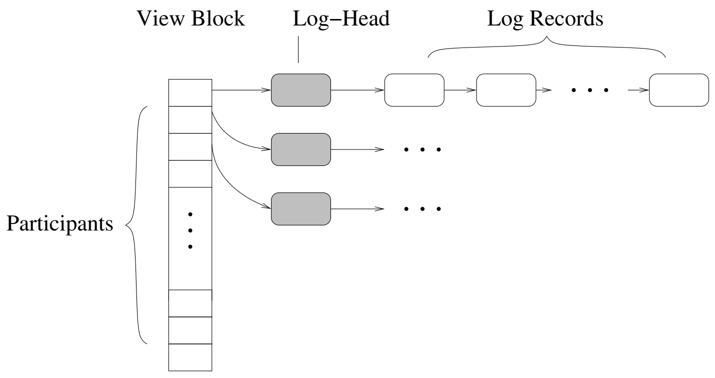

Fig. 3.10. *Ivy: File System View*

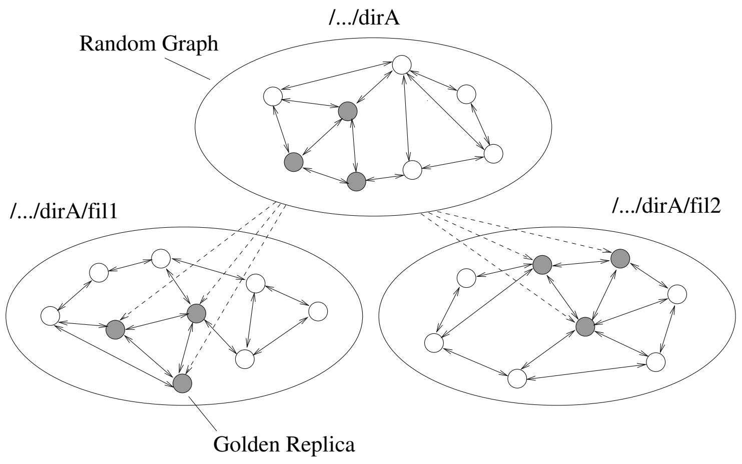

Fig. 3.11. *Pangaea System Architecture*

A private snapshot of the system is maintained by the participants in order not to have to scan all the logs for every read. Only the most recent log records need to be scanned. Since Ivy avoids using shared mutable data structures, locking is not necessary. Ivy logs contain version vectors and timestamps. These can help applications in detecting and resolving conflicts that may arise due to concurrent updates.

This strategy of maintaining per-participant logs makes Ivy suitable only for a small number of cooperating users. Moreover, high possibilities of conflicting concurrent updates result in Ivy providing weak consistency semantics.

3.3.4. Pangaea. The objective of Pangaea [38] is to build a planetary-scale P2P file system used by groups of collaborating users all over the world. The system attempts to achieve low access latency and high availability using *Pervasive Replication* techniques. Whenever and wherever a file is accessed, a replica is created. Popular files therefore get heavily replicated and personal files reside only on the nodes used by the owners.

A random graph of all the replicas is maintained for propagating updates and ensuring availability (figure 3.11). The random graph is created by making each replica maintain links to k other replicas chosen randomly. A few of the replicas are designated as *Golden* replicas. The golden replicas maintain links with each other and ensure that their set always maintains specified membership levels. Replicas perform random walks starting from one of the golden replicas to create random links. This way the graph stays connected.

Links to the golden replicas are recorded in the data object's parent directory (which is also maintained as a file). To access and replicate a file, its parent directory must be accessed and hence replicated. The recursive operation can proceed all the way to the file system's root.

By default, update propagation happens lazily. A strategy involving *Harbinger* messages is used to build a spanning tree which is used for quick update propagation. Strict consistency semantics are also supported by

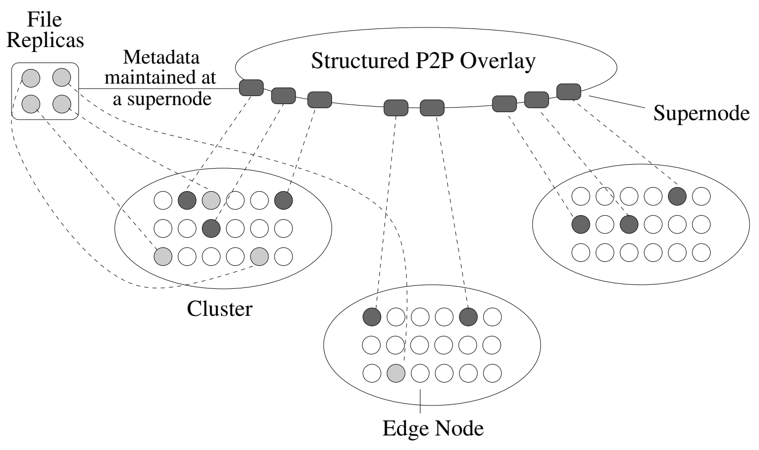

Fig. 3.12. *ENFS System Architecture*

making the updating client wait for acknowledgments from the replicas. A version vector based algorithm [37] is used for resolving conflicting updates.

3.3.5. Edge Node File System (ENFS). ENFS [25] exploits the resources of Internet edge nodes to provide scalable DFS services. Undedicated Internet edge nodes are enabled to function as both data and metadata servers. The presence of a large number of edge nodes results in scalable metadata access and high I/O throughputs.

ENFS uses proximity-based clustering of edge nodes (figure 3.12) for the efficient management of resources, balancing of load (storage, computational, query), and handling latency issues. A few reliable and capable edge nodes from each cluster are chosen to be the metadata servers (*Supernodes*) for that cluster. These supernodes are chosen based on capabilities such as network bandwidth, processor speed, storage space, and memory capacity. Each supernode is associated with a replica set consisting of a fixed number of other supernodes from the same cluster. The replica sets ensure high system availability.

Supernodes from all the clusters form a single system-wide structured P2P overlay network for use as a distributed hash table. By connecting up all the clusters in the system, the overlay enables nodes of a cluster to discover supernodes (of other clusters) which are responsible for specific portions of the file namespace. The structured overlay also helps in the efficient discovery of resources with specific characteristics in the entire system.

Since the sets of data and metadata servers change autonomously and dynamically to suit prevalent workloads, ENFS scales transparently. The architecture of the system allows data placement/access decisions to be based on applications' requirements of resource characteristics and locations. The metadata of each file has a single point of access (one of the cluster supernodes). This allows ENFS to support a large spectrum of access semantics.

4. Comparative Analysis.

In this section, we analyze the above reviewed systems with respect to their scalability and the support they provide for high application performance only. We do not address other aspects of distributed file systems such as user/group management, security and trust, etc. In [30], the authors provide a survey of decentralized access control mechanisms in large scale distributed file systems. An overview of I/O systems (including file systems) dealing with massive data is presented in [22].

The manner in which the load on different file system servers vary with increasing numbers of users, and therefore user files, primarily determines the scalability of a distributed file system. Increase in the number of files results in an increase in the number of queries and in the amount of data I/O.

The system parameters used in the analysis are shown in table 4.1. For the sake of simplicity, we assume uniform server capabilities and that the file system metadata and data are equally distributed among the servers. We also assume that the metadata queries and I/O requests are generated in an independent and completely random manner.

We study the dependence of metadata and data server loads on the query and I/O rates in tables 4.2 and 4.3 respectively. The overheads of overlay network management also add to server loads, especially in the P2P file systems. The overheads are presented in table 4.4.

TABLE 4.1
System Parameters and Metrics

Parameter	Details
N	Number of nodes (servers/clients/peers) in the system
N_M	Number of metadata servers in the system
N_D	Number of storage nodes (data servers) in the system
F	Number of data items (files and directories) in the system
R	Average number of replicas per data item
Q	Number of metadata queries generated per unit of time in the system
D	Data transfer demand to and from the data servers in the system per unit of time
l_C	Network latency between nodes within a cluster/LAN (Intranet)
l_W	Network latency between nodes in different clusters (Internet)
$P(n)$	Cost of achieving consensus (Paxos [27], Byzantine fault tolerant algorithm, quorum-based voting) among n nodes in terms of time and number of messages
L_{MS}	Average query handling load on a metadata server
L_{DS}	Average I/O load on a data server
L_{OM}	Message, time and space overheads of maintaining the different overlays

In GoogleFS, Lustre, Panasas, PVFS2, Ceph, OceanStore and ENFS, support for file striping and parallel I/O helps in distributing data server load at a finer granularity. From table 4.3, we can see that, L_{DS}, the data server load, can be represented as $f(D/N_D)$ for category I and category II file systems and as $f(D/N)$ for category III file systems.

The components that get overloaded in the first category of file systems are clearly the servers. In these systems, the N_M metadata servers are usually the data servers also. The load on each server therefore is $L_{MS} + L_{DS}$. Both increasing query rates and I/O demands affect the same set of servers.

In the second category of file systems, decoupling of data and metadata helps in splitting the load among different sets of servers (L_{MS} for metadata servers and L_{DS} for data servers). However, due to rigid server configurations which require manual administration, the values of N_M and N_D are more or less fixed. This results in these systems supporting only constrained levels of metadata and I/O demands. Additionally, in WheelFS, the configuration service can potentially become a bottleneck with increasing query rates.

Since Farsite, OceanStore, Ivy, Pangaea and ENFS are P2P file systems (category III), the load on each node is $L_{MS} + L_{DS} + L_{OM}$. The number of nodes, N, is however virtually unlimited. Therefore, the loads are well distributed.

However, Ivy is a log-based file system and so performance falls significantly with increasing numbers of participants. Network usage is excessively high in OceanStore and Pangaea due to overlay management messages, pervasive replication and update propagations. Since a considerable number of peers in a wide area installation may possess low bandwidth connections, system performance can be affected by increasing load levels in these two systems.

The performance of applications executing over file systems depends mainly on the speed of metadata access and data I/O throughput. Metadata query and update times experienced by applications depend on several factors such as metadata server load, query routing mechanism, network latency, and consistency management strategy. Table 4.5 analyzes these factors in the various systems.

Data I/O throughput depends on server load and network latency/bandwidth. Server loads are discussed in table 4.3. The support provided by the file systems to reduce the effects of network latency and bandwidth on data transfer/processing speed, and hence on application performance, is discussed in table 4.6.

TABLE 4.2
Metadata Server Load as a Function of Query Rate

System	**Load/Server (L_{MS})**	Comments
AFS	$f(Q/N_M)$	The load is distributed among the N_M servers. Since the number of servers is fixed and can be extended only through administrator intervention, server load keeps increasing with Q.
CIFS	$f(Q/N_M)$	The load is distributed among the N_M servers that are sharing content. Typically, the number of servers in CIFS installations are much larger than in AFS installations. Query loads are therefore better distributed.
GoogleFS	$f(Q)$	The master server handles all the queries. As a result, such an architecture's scalability is limited.
Lustre/ Panasas/ PVFS2	$f(Q/N_M)$	The query load is distributed among the N_M metadata servers. Since the number of MDSs is fixed and can be extended only by manual intervention, load on an MDS keeps increasing with Q.
Ceph	$f(\alpha \cdot Q/N_M)$	The metadata query load is distributed among the servers in the MDS cluster. The dynamic subtree partitioning scheme employed by Ceph distributes the query load among the servers uniformly. Moreover, since clients can calculate object locations themselves, metadata server loads are significantly reduced (represented by α).
WheelFS	$f(Q/N_M)$	The query load is distributed among the N_M WheelFS primary servers.
	$f(Q)$	Clients get information about the primary servers responsible for files from the configuration service. The load on the configuration service therefore increases along with Q.
Farsite	$f(Q/(\kappa \cdot N))$	When query rates increase, directory groups split and distribute the load among more nodes. Since any peer can be a part of a directory group, query loads are shared by a significant fraction (κ) of all the nodes in the system.
OceanStore	$f(Q/N)$	Information about files in OceanStore are obtained using pure P2P algorithms. The metadata query load is therefore distributed among all the peers.
Ivy	$f(Q/N)$	Metadata queries result in getting the recent log records of all participants and scanning the records locally at the querying peer. Thus, the query load is distributed among all the peers.
Pangaea	$f(Q/N)$	Metadata accesses happen using P2P routing protocols and result in replicas getting created at the querying peers. Thus the query load is shared by all the peers.
ENFS	$f(Q/(\kappa \cdot N))$	The number of supernodes increases with increasing query loads (Q). Since any node in the system can be made a supernode, the load is shared by a significant fraction (κ) of N, as in Farsite.

Table 4.3
Data Server Load as a Function of the I/O Demand

System	Comments
AFS	Callback promises and invalidations, and whole file caching help in reducing the load on the AFS servers. This is one of the main reasons for AFS scaling better than NFS.
CIFS	Stateful servers, elaborate locking mechanisms, caching, and read-aheads, help in reducing the load on the servers. A large number of servers sharing files helps distribute the load better than in AFS.
GoogleFS	The data load is distributed among the N_D chunkservers in the GoogleFS cluster. GoogleFS does not support client side caching, especially because the applications usually require computations to be performed at the chunkservers itself.
Lustre	The load is shared among the N_D object storage servers. Server based distributed file locking protocols and client side caching in Lustre help reduce data server loads.
Panasas	The data serving load is shared among the N_D OSDs. File locking services and consistent client caching is supported in Panasas.
PVFS2	PVFS2 does not cache data on the clients and so the entire load is distributed among the N_D I/O servers.
Ceph	Client side caching absorbs some load off the N_D OSDs.
WheelFS	All clients maintain caches of files read. Semantic cues help in satisfying a client's data needs with nearby caches as much as possible. Such *Cooperative Caching* mechanisms help in reducing the loads on WheelFS servers significantly.
Farsite	All the nodes in the system are capable of storing data. As data loads increase, more replicas can be created among the peers. Thus, data transfer loads are shared by a large number of nodes ($O(N)$).
OceanStore	Promiscuous caching and P2P data location algorithms enable data serving loads to be distributed among the peers in the system.
Ivy	All the data objects in Ivy are stored in the DHash DHT, which comprises of all the nodes in the system. Thus data transfer load is shared by the entire set of nodes.
Pangaea	Pervasive caching results in files and directories getting replicated in a large number of peers in the system. I/O load is therefore distributed widely.
ENFS	Supernodes ensure that file contents in ENFS are distributed uniformly across all the storage nodes in the system. Data transfer loads are therefore shared by a large number of nodes ($O(N)$).

Apart from data server loads, application performance largely depends on the network distance between servers and clients. In most file systems of category I and II, server locations are fixed and so in wide area installations, data access usually happens across long distances. Data caching helps in reducing the distance to some extent, especially in AFS and WheelFS.

File systems belonging to category III, however, do not have fixed servers. The peer-to-peer nature of these systems support the creation of new file replicas closer to their users. ENFS goes a step further and pro-actively creates file replicas on nodes which are likely to process the contents, based on user specification or application type.

4.1. Observations. In summary, our analysis of these systems has led to the following observations:

- *Decentralization* Most of the production file systems today use central servers (or clusters of servers). While such an infrastructure can support a large number of users and files, their scalability is limited. Since the digital data generation capabilities of the masses has increased tremendously, the next few years are expected to witness huge rates of data creation. Decentralization is therefore essential to manage the accompanying data management demands. Decentralization also has other benefits such as not having to completely trust one central entity, lack of a single point of failure, robustness, and lack of the need for expensive servers.

Table 4.4
Overlay Maintenance Overheads

System	**Overhead (L_{OM})**	**Comments**
WheelFS	$f(C_{RSM})$	The configuration service is implemented as a replicated state machine with a certain number of nodes. Maintaining the state machine involves operations such as handling membership changes, and electing a new leader. C_{RSM} represents the corresponding message and time overheads for the configuration service nodes.
Farsite	$f(C_{BFT})$	All the nodes in Farsite which are part of a directory group incur the overheads of maintaining a Byzantine fault tolerant group. The overhead associated with Byzantine fault tolerance is represented by C_{BFT}.
OceanStore	$f(logN, C_{BF})$	Every node in OceanStore maintains a routing table associated with the Plaxton scheme for global data location. The size of the table is $O(logN)$. Moreover, changing object contents in a node and its local vicinity, results in changes to its attenuated Bloom filter. The network and computational (multiple hashing) overheads of maintaining the filters is also significant and is represented by C_{BF}.
Ivy	$f(logN)$	Nodes in Ivy are part of the DHash DHT and so maintain routing tables with $O(logN)$ entries.
Pangaea	$f((F \cdot R \cdot k)/N)$	Every replica of a data item must maintain at least k links to other replicas. This results in significant message, time and space overheads.
ENFS	$f(logN_M)$	Supernodes from all the clusters form a structured overlay in ENFS. Each supernode maintains a routing table of size $O(logN_M)$.

- *Autonomic System Management* Since decentralized systems usually exploit the resources of unreliable nodes, mechanisms must be in place to provide notions of reliability and availability to the users/applications. It is impractical for large distributed systems to be manually administered. Essential tasks such as handling node failures, and load balancing must be autonomically managed for better resource utilization and application performance.
- *Pervasive Replication* High levels of replication, especially of read-only files, increases availability and brings data closer to the users, thereby improving application performance. Replication has the added benefit of enabling parallel access to files. Parallel access enables computations on different parts of a file to be performed simultaneously. In a well designed system, the benefits of replication must over-weigh the overheads of additional data transfer and consistency management.
- *Flexible Consistency Semantics* Often, the stronger the consistency semantics supported by a system, the poorer the application performance. The consistency requirements of different applications vary widely. Thus, file systems must be capable of flexing their consistency semantics in accordance to application requirements. This way, users/applications can themselves adjust the required levels of consistency/performance trade-off.
- *Data Affinity* Data affinity refers to the concept of ensuring that files are stored close to the nodes which are most suited and likely to process their contents. For example, in HPC applications, due to large data set sizes, schedulers attempt to schedule computations on resources which contain the required data [36] [47], thus reducing the amount of data movement. Therefore, file systems which support resource characteristics aware data placement are highly useful. Data migration with changing access patterns is also beneficial.
- *Proximity-based Node Clustering* A large system which cannot be managed by a central controller is best managed by being partitioned into proximity-based node clusters of manageable sizes. In dis-

TABLE 4.5
Factors affecting Metadata Query Response Times

System	**Comments**
AFS	$f(L_{MS} + L_{DS}, l_C(or)l_W)$ Servers are usually distributed across wide areas. Servers in every cell possess information about the servers hosting different data volumes across the entire system. Therefore, there are no query routing overheads. The effect of network latency depends on whether queries are made for files served locally or by a server in a different cell. Data volumes are placed close to users/groups owning the corresponding data items and so latency effects are generally low.
CIFS	$f(L_{MS} + L_{DS}, l_C(or)l_W)$ CIFS servers are usually distributed across wide areas. Clients either possess information about servers hosting different data items or can use browsing protocols to search for servers. When a client queries a distant CIFS server, high network latency is likely to affect the response time.
GoogleFS	$f(L_{MS}, l_C, P(2))$ Since GoogleFS installations are usually cluster based, network latency is l_C. All metadata queries are handled by the master server. Metadata updates must be serialized in the master server and its shadow.
Lustre	$f(L_{MS}, l_W, P(2))$ The set of metadata servers are clustered in a single location and so most client queries have to travel across the network in a wide area installation. Metadata updates must be serialized in the active and passive metadata servers associated with a data item.
Panasas	$f(L_{MS}, l_W, P(N_M))$ Panasas uses a quorum-based voting protocol to commit metadata operations in its metadata servers. As in Lustre, network latency is usually l_W since the servers are clustered in one location.
PVFS2	$f(L_{MS}, l_W)$ PVFS2 avoids serialization of independent metadata operations using an explicit state machine, threads (to provide non-blocking access), and a component that monitors completion of operations across devices. Avoiding serialization makes metadata access faster.
Ceph	$f(L_{MS}, l_W, P(k))$ Since the metadata servers are clustered, far-off clients experience high network latencies. Metadata updates must be synchronously journaled to a cluster (of size k) of OSDs for safety.
WheelFS	$f(L_{MS}, l_W)$ Accessing the configuration service to determine the primary may involve a query to a far-off node. Clients can specify location preferences for the primary servers for their files and directories based on expected access patterns and so latency overheads of accessing the primary servers are optimized.
Farsite	$f(L_{MS} + L_{OM}, d \cdot l_W, P(k))$ Metadata access may require traversal from the root to the directory of interest. Each directory may be managed by a different group. d represents the average number of hops between directory groups required to reach a data item. Metadata updates require Byzantine fault tolerant agreement among the k directory group members.
OceanStore	$f(L_{MS} + L_{DS} + L_{OM}, l_W \cdot logN, C_{ARC})$ Locating the root of an object in OceanStore can require $O(logN)$ hops across a wide area network. Some files, especially popular ones, can however be located in the local vicinity of the client. Every update (or group of updates) involves storing the object in an archival form. C_{ARC} represents the corresponding costs of encoding the file using erasure coding and distributing it across hundreds of machines.
Ivy	$f(L_{MS} + L_{OM}, p \cdot (logN) \cdot l_W)$ Accessing the metadata requires the gathering of the most recent log records of all the participants (p). Metadata updates are performed in the local log alone.
Pangaea	$f(L_{MS} + L_{OM}, l_C, C_{ST})$ The pervasive replication strategy results in most data items being available in close proximity. Propagation of updates happens in two phases along the spanning tree for that data item rooted at the source. The corresponding message and time costs are represented by C_{ST}.
ENFS	$f(L_{MS} + L_{OM}, l_C(or)l_W, P(k))$ Metadata of user files are managed by supernodes in the same cluster as that of the user. However, accessing the metadata of files in other clusters requires across network querying. Metadata servers responsible for individual files/directories are identified using index files stored in the system wide DHT and actively cached in the local cluster's supernodes. Discovery can therefore usually happen within a couple of hops. Metadata updates are serialized in the responsible supernode and its replica set. k represents the supernode replica set size.

Table 4.6
Support for Application Performance

System	**Support**
AFS/ CIFS	Servers in these systems only perform file I/O. Any other operation to be performed on the data must be performed at the client site. Client side caching is supported to varying degrees. AFS, especially, improves application performance using whole file caching. However, the benefits of caching come at the expense of consistency management. AFS provides weak consistency semantics. CIFS uses elaborate locking mechanisms to provide strong consistency semantics. I/O throughputs are largely dependent on client-server network distance.
GoogleFS	GoogleFS is optimized for the MapReduce class of applications. GoogleFS's support for appending records to existing datasets in a quick, atomic and race-free manner is critical for MapReduce applications. GoogleFS stores replicas of data chunks on different machines. This increases the chances of MapReduce scheduling mappers on nodes with the data or on nodes close to the data. GoogleFS supports relaxed consistency semantics, which helps speed up data appends.
Lustre/ Panasas/ Ceph	Since object-based storage devices support the storage and serving of objects directly at the hardware level, better I/O throughputs can be achieved compared to normal disc I/O. Application specific processing/computations however cannot be performed at the servers. These systems provide strong consistency semantics. I/O throughputs are largely dependent on client-server network distance.
PVFS2	Client side caching is not supported. Client server distance can therefore be detrimental to application performance. PVFS2 implements *Non-Conflicting Write* semantics, thus allowing clients to update non-conflicting portions of the namespace simultaneously without locks.
WheelFS	Placement semantic cues such as *.Site*, *.KeepTogether* and *.RepSites* allow owners to place their data close to the users most likely to use the data. This helps optimize data throughputs. Cues can also be used to fetch file contents from the cache of other clients in parallel.
Farsite	Farsite does not attempt to reduce latency. It is designed to support typical user home directory I/O instead of the high performance I/O of scientific applications. Byzantine fault tolerant agreement protocols and leases help in providing strong consistency guarantees in Farsite.
OceanStore	Users choose primary and secondary tier storage nodes on which to store their files. Moreover, popular files get widely cached. These measures help in improving data throughputs. Based on application requirements, OceanStore can provide a variety of consistency semantics.
Ivy	Nodes maintain a private snapshot of all the logs and so file reads only require the most recent records to be obtained from the DHash DHT. Ivy provides weak consistency semantics with application assisted conflict resolutions.
Pangaea	In Pangaea, replica locations are determined by user activities. Files can therefore usually be located close to the clients. By default, Pangaea implements weak consistency semantics. However, stronger guarantees can be provided by trading off performance.
ENFS	ENFS focuses on the principle that awareness of the capabilities of storage nodes is critical for a file system to be useful for applications. Cluster supernodes can inexpensively discover resources with specific characteristics across the entire system. File/Replica placement decisions are based on the requirements of the applications expected to operate on the files. This helps applications achieve high performance. Home-based consistency protocols allow a wide variety of access semantics to be supported.

tributed systems, clustering supports the scalable and efficient discovery of data and resources with specific characteristics from the entire system [35]. Clustering also provides for efficient communication mechanisms among proximal and far-off nodes in the system. Co-location of servers and their associated clients, which helps in optimizing network latency, also becomes simpler when the system is partitioned into clusters. A lot of work, done on network distance measurement [15], topology discovery [20] [9] and proximity-based node clustering [51] [29] [35], can be used for autonomous cluster formation and management.

- *Capability-based Role Assignment* Farsite and ENFS are examples of P2P file systems in which peers are assigned different roles based on their current capability levels (CPU load, memory, network). Nodes with relatively high levels of capability are made responsible for file metadata services. This helps in reducing the effects of system dynamics on file availability and access. OceanStore and Pangaea do not perform capability-based role assignment. These systems therefore use up a lot of network bandwidth and space in maintaining per-file overlays.

5. Conclusions. This survey analyzes popular wide area distributed file systems for their scalability and the support they provide for high application performance. Several design decisions affect the way file systems scale and applications perform.

We categorize the systems as *Traditional Distributed File Systems*, *Asymmetric Cluster File Systems* and *Self-Organizing P2P File Systems*, based on the extent of data/metadata distribution across the system.

We perform scalability analysis by characterizing the loads on file system servers as functions of query rates and data I/O demands. Application performance is studied by characterizing query response times as functions of the appropriate system parameters. Data I/O throughputs and support for data affinity are also analyzed. The summarized observations are presented in section 4.1.

It is not possible to design a wide area distributed file system that performs ideally for all kinds of applications. Often, providing support for one feature affects another negatively. For wider acceptance, distributed file systems must allow client applications to conveniently control the different trade-offs amongst file system features.

REFERENCES

[1] *The Chord/DHash project. An implementation of the Chord DHT in C++.* http://pdos.csail.mit.edu/chord/.

[2] *Comparison of file systems.* http://en.wikipedia.org/wiki/Comparison_of_file_systems.

[3] *List of file systems.* http://en.wikipedia.org/wiki/List_of_file_systems#Distributed_file_systems.

[4] *Parallel Virtual File System, version 2.* http://www.pvfs.org/.

[5] *Framework for loosely coupled wide area file system access.* http://www.diceprogram.org/reports/docs/FrameworkforLooselyCoupledWideAreaFileSystemAccess.pdf, 2008.

[6] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer, *Farsite: Federated, available, and reliable storage for an incompletely trusted environment*, SIGOPS Operating Systems Review, 36 (2002), pp. 1–14.

[7] T. Anderson and A. Vahdat, *GENI distributed services.* http://www.geni.net/GDD/GDD-06-24.pdf, 2006.

[8] P. Andrews, P. Kovatch, and C. Jordan, *Massive high-performance global file systems for Grid computing*, in SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing, Washington, DC, USA, 2005, IEEE Computer Society, p. 53.

[9] Y. Bejerano, Y. Breitbart, M. N. Garofalakis, and R. Rastogi, *Physical topology discovery for large multisubnet networks*, in Proceedings IEEE INFOCOM âĂŹ03, 2003, pp. 342–352.

[10] M. Blondel, *WikipediaFS.* http://wikipediafs.sourceforge.net/, 2007.

[11] B. H. Bloom, *Space/time trade-offs in hash coding with allowable errors*, Communications of the ACM, 13 (1970), pp. 422–426.

[12] W. J. Bolosky, J. R. Douceur, and J. Howell, *The Farsite project: a retrospective*, SIGOPS Operating Systems Review, 41 (2007), pp. 17–26.

[13] D. Borthakur, *The Hadoop Distributed File System: Architecture and design.* http://hadoop.apache.org/core/docs/current/hdfs_design.html, 2008.

[14] M. Castro and B. Liskov, *Practical Byzantine Fault Tolerance*, in Operating Systems Design and Implementation, Feb. 1999, pp. 173–186.

[15] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, *Vivaldi: A decentralized network coordinate system*, in Proceedings of the ACM SIGCOMM'04 Conference, 2004, pp. 15–26.

[16] J. Dean and S. Ghemawat, *MapReduce: Simplified data processing on large clusters*, Communications of the ACM, 51 (2008), pp. 107–113.

[17] A. Devulapalli, I. T. Murugandi, D. Xu, and P. Wyckoff, *Design of an intelligent object-based storage device.* http://www.osc.edu/research/network_file/projects/object/papers/istor-tr.pdf.

[18] S. Fadden, *An introduction to GPFS Version 3.3*, IBM White Paper, (2010).

[19] S. Ghemawat, H. Gobioff, and S.-T. Leung, *The Google File System*, SIGOPS Operating Systems Review, 37 (2003), pp. 29–43.

[20] R. Govindan and H. Tangmunarunkit, *Heuristics for Internet map discovery*, in Proceedings IEEE INFOCOM âĂŹ00, 2000, pp. 1371–1380.

[21] J. H. Howard, *An overview of the Andrew File System*, in USENIX Winter, 1988, pp. 23–26.

[22] R. Hubovsky and F. Kunz, *Dealing with massive data: From parallel I/O to Grid I/O*, Master's thesis, University of Vienna, 2004.

[23] T. Kosar and M. Livny, *Stork: Making data placement a first class citizen in the Grid*, in Proceedings of the International Conference on Distributed Computing Systems '04, 2004, pp. 342–349.

[24] P. Kovendhan and D. Janakiram, *Arogyashree: A distributed file system for large scale Internet-based telemedicine*, in Proceedings of MobiHealthInf '09, The First International Workshop on Mobilizing Health Information to Support Healthcare-related Knowledge Work, 2009, pp. 105–114.

[25] P. Kovendhan and D. Janakiram, *The Edge Node File System: A distributed file system for high performance computing*, Scalable Computing: Practice and Experience, 10 (2009), pp. 115–130.

[26] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, and B. Zhao, *Oceanstore: An architecture for global-scale persistent storage*, SIGARCH Computer Architecture News, 28 (2000), pp. 190–201.

[27] L. Lamport, *The part-time parliament*, Transactions on Computer Systems, 16 (1998), pp. 133–169.

[28] P. Leach and D. Naik, *A Common Internet File System Protocol (CIFS)*. Internet draft, Internet Engineering Task Force (IETF), 1997.

[29] E. K. Lua, J. Crowcroft, and M. Pias, *Highways: Proximity clustering for scalable Peer-to-Peer networks*, in Proceedings 4th International Conference on P2P Computing, 2004, pp. 266–267.

[30] S. Miltchev, J. M. Smith, V. Prevelakis, A. Keromytis, and S. Ioannidis, *Decentralized access control in distributed file systems*, ACM Computing Surveys, 40 (2008), pp. 1–30.

[31] A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen, *Ivy: A read/write Peer-to-Peer file system*, SIGOPS Operating Systems Review, 36 (2002), pp. 31–44.

[32] D. A. Patterson, G. A. Gibson, and R. H. Katz, *A case for Redundant Arrays of Inexpensive Disks (RAID)*, in SIGMOD Conference, 1988, pp. 109–116.

[33] M. Pease, R. Shostak, and L. Lamport, *Reaching agreement in the presence of faults*, Journal of the ACM, 27 (1980), pp. 228–234.

[34] G. Plaxton, R. Rajaraman, and A. W. Richa, *Accessing nearby copies of replicated objects in a distributed environment*, in SPAA '97: Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures, ACM, 1997, pp. 311–320.

[35] L. Ramaswamy, B. Gedik, and L. Liu, *A distributed approach to node clustering in decentralized Peer-to-Peer networks*, IEEE Transactions on Parallel and Distributed Systems, 16 (2005), pp. 814–829.

[36] K. Ranganathan and I. Foster, *Simulation studies of computation and data scheduling algorithms for data Grids*, Journal of Grid Computing, 1 (2003), pp. 53–62.

[37] D. H. Ratner, *Roam: A Scalable Replication System for Mobile and Distributed Computing*, PhD thesis, University of California Los Angeles, 1998.

[38] Y. Saito and C. Karamanolis, *Pangaea: a symbiotic wide-area file system*, in EW10: Proceedings of the 10th workshop on ACM SIGOPS European workshop, New York, NY, USA, 2002, ACM, pp. 231–234.

[39] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, *Design and implementation of the Sun Network Filesystem*, in Proceedings Summer 1985 USENIX Conference, 1985, pp. 119–130.

[40] P. Schwan, *Lustre: Building a file system for 1000-node clusters*, in Proceedings of Linux Symposium, 2003, pp. 380–386.

[41] Z. Sebepou, K. Magoutis, M. Marazakis, and A. Bilas, *A comparative experimental study of parallel file systems for large-scale data processing*, in LASCO'08: First USENIX Workshop on Large-Scale Computing, Berkeley, CA, USA, 2008, USENIX Association, pp. 1–10.

[42] S. C. Simms, G. G. Pike, and D. Balog, *Wide area filesystem performance using Lustre on the TeraGrid*, in in Proceedings of the TeraGrid 2007 Conference, 2007.

[43] B. P. Spencer, D. Noveck, D. Robinson, and R. Thurlow, *The NFS version 4 protocol*, in Proceedings of International System Administration and Networking (SANE) Conference '00, 2000, pp. 94–113.

[44] J. Steiner and J. I. Schiller, *An authentication service for open network systems*, in USENIX Conference Proceedings, 1988, pp. 191–202.

[45] I. Stoica, R. Morris, D. Karger, F. M. Kaashoek, and H. Balakrishnan, *Chord: A scalable Peer-to-Peer lookup service for Internet applications*, in Proceedings of ACM SIGCOMM '01, vol. 31, October 2001, pp. 149–160.

[46] J. Stribling, Y. Sovran, I. Zhang, X. Pretzer, J. Li, M. F. Kaashoek, and R. Morris, *Flexible, wide-area storage for distributed systems with WheelFS*, in Networked Systems Design and Implementation, Boston, MA, April 2009, pp. 43–58.

[47] A. Takefusa, O. Tatebe, S. Matsuoka, and Y. Morita, *Performance analysis of scheduling and replication algorithms on Grid Datafarm architecture for high-energy physics applications*, in Proceedings of 12th IEEE High Performance Distributed Computing, 2003, pp. 34–43.

[48] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, *Ceph: A scalable, high-performance distributed file system*, in Operating Systems Design and Implementation, 2006, pp. 307–320.

[49] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, *CRUSH: Controlled, scalable, decentralized placement of replicated data*, in Proceedings of Supercomputing '06, 2006, pp. 31–31.

[50] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou, *Scalable performance of the Panasas parallel file system*, in Proceedings of the 6th USENIX Conference on File and Storage Technologies, 2008, pp. 1–17.

[51] Q. Xu and J. Subhlok, *Automatic clustering of Grid nodes*, in Proceedings of 6th IEEE/ACM International Workshop on Grid Computing, IEEE Computer Society, 2005, pp. 227–233.

Edited by: Thomas Ludwig
Received: June 6-th, 2010
Accepted: September 22-nd, 2010