Pangaea: a symbiotic wide-area file system

Yasushi Saito and Christos Karamanolis
HP Labs, Storage Systems Department
1501 Page Mill Rd, Palo Alto, CA, USA.
{ysaito,christos}@hpl.hp.com

1 Introduction

Pangaea is a planetary-scale file system designed for large, multi-national corporations or groups of collaborating users spread over the world. Its goal is to handle people's daily storage needs—e.g., document sharing, software development, and data crunching—that can be write intensive. Pangaea uses *pervasive replication* to achieve low access latency and high availability. It creates replicas dynamically whenever and wherever requested, and builds a random graph of replicas for each file to propagate updates efficiently. It uses an optimistic consistency semantics by default, but it also offers a manual mechanism for enforcing consistency. This paper overviews Pangaea's philosophy and architecture for accommodating such environments and describes randomized protocols for managing large numbers of replicas efficiently.

Pangaea is built by federating a large number of unreliable, widely distributed commodity computers, provided by the system's users. Thus, the system faces continuous reconfiguration, with users moving, companies restructuring, and computers being added and removed. In this context, the design of Pangaea must meet three key goals:

Performance: Hide the wide-area networking latency; file access latency should resemble that of a local file system.

Availability: Not depend on the availability of any specific (central) servers; Pangaea must *adapt* automatically to server additions and failures without disturbing users.

Autonomy: Avoid centralized management; each node in the system should be able to perform its own resource management, e.g., the amount of disk space to offer and when to reclaim it.

To achieve these goals, we advocate a *symbiotic* approach to the system design. Each server should be able to function autonomously, and serve to its users most of their files even when disconnected. However, as more computers become available, they should dynamically adapt and collaborate with each other in a way that enhances the overall performance and availability of the system. Pangaea is an example of this approach.

1.1 Pangaea Overview

Pangaea builds a unified file system over thousands of servers (nodes) distributed over the world. We currently assume that these servers are trusted. Pangaea addresses the aforementioned goals via *pervasive replication*; it creates replicas of a file or a directory dynamically whenever and wherever requested and distributes updates to these replicas efficiently. Thus, there may be billions of files, each replicated on hundreds on nodes. Files shared by many people (e.g., the root directory) are replicated on virtually every node, whereas users' personal files reside on their local nodes and only a few remote nodes for availability. Pangaea demands no synchronous coordination among nodes to update file contents or to add or remove replicas. All changes to the system are propagated epidemically in the background.

Pervasive replication offers three main advantages: 1) provides fault tolerance, stronger for popular files; 2) hides network latency; 3) supports disconnected operations by containing a user's working set in a single server. These are key features for realizing the symbiotic approach in Pangaea's design.

1.2 Related work

The idea of pervasive replication is similar to persistent caching used in systems such as AFS, Coda [4] and LBFS [5]. However, Pangaea offers several additional advantages. First, it provides better availability. When a node crashes, there are always other nodes providing access to the files it hosted. Secondly, the decentralized nature of Pangaea also allows for better site autonomy; it lets any node be removed or replaced at any time transparently to the user. Finally, Pangaea achieves better performance by creating new replicas from a nearby existing replica and by propagating updates along fast network links. In this sense, Pangaea provides a generalization of the idea of Fluid replication [2] that utilizes surrogate Coda servers placed in strategic (but fixed) locations to improve the performance and availability of the system.

Mobile data sharing services, such as Lotus Notes [3], Roam [7], and Bayou [6], allow mobile users to replicate data dynamically and work disconnected. However, they lack a replica location service. Humans are responsible for synchronizing devices to keep replicas consistent. In contrast, Pangaea keeps track of the location of replicas and distributes updates proactively and transparently to the users.

Farsite [1] is similar to Pangaea in that it provides a unified file system over pervasive number of nodes. The two systems, however, have different focuses. Farsite builds a reliable service on top of untrusted desktop nodes by using Byzantine consensus protocols, but it is not concerned with reducing latency. On the other hand, Pangaea assumes trusted servers, but it dynamically replicates files closer to their point of accesses to minimize the use of wide-area networks.

Recent peer-to-peer data sharing systems, such as CFS, Oceanstore, and PAST, build flat distributed tables using randomization techniques. Although Pangaea shares many of their goals—decentralization, availability and autonomy—its applications are different. In Pangaea, replicas are placed by user activity, not by randomization; files encounter frequent updates and are structured hierarchically. These differences force Pangaea to maintain a graph of replicas explicitly.

1.3 Challenges in Pangaea

While offering many benefits, pervasive replication introduces fundamental challenges as well. The first is the computational and storage overhead of meta-data management. While this problem is genuine, servers in cooperative-work environments, which we target initially, are known to waste much of their resources anyway [1].

The second challenge is to design algorithms for keeping track of a large number of files and replicas in a decentralized and highly available way. To address this problem, Pangaea builds a random graph over the set of replicas of each file and uses the graph to locate replicas and distribute updates. We present simple randomized algorithms for maintaining these graphs in Section 2.

The third challenge is the difficulty of propagating updates reliably yet efficiently and providing strong consistency guarantees. We solve this challenge by the combination of two techniques: 1) dynamic overlaying of a spanning tree over the random graph, and 2) a mechanism to explicitly enforce replica consistency for demanding applications. We discuss these mechanisms in Section 3.

2 Managing replica membership

Pangaea experiences very frequent replica additions and removals, because it manages billions of files, each of which is replicated independently on many servers. Thus, it calls for available and cheap mechanisms to manage the replica membership efficiently for each file and distribute updates reliably among replicas. Pangaea decentralizes both the tasks to

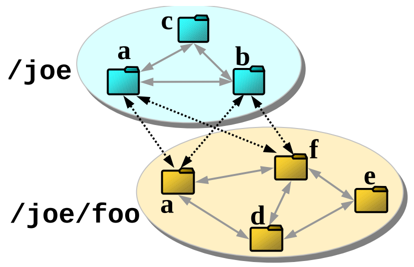

Figure 1: *A replica graph. Each file (an oval) consists of its own set of replicas. Labels show the names of the servers that store replicas. Edges between replicas show acquaintance relationships. Updates to the file are propagated along these edges. Edges are also spanned between some of the file's replicas and those of its parent directory to allow for path traversal.*

achieve this goal. There is no entity that centrally manages the replicas for a file. Instead, Pangaea lets each replica maintain links to k ($k = 3$ in our implementation) other random replicas of the same file. An update can be issued at any replica, and it is flooded along these edges (Section 3.1 describes a protocol for minimizing the flooding overhead.) Links to some of these replicas are also recorded in the entry of the parent directory, and they act as starting points file lookup (Section 2.3). Figure 1 shows an example. This design addresses our goals as follows:

Available update distribution: Pangaea can always distribute updates to all live replicas of a file even after k simultaneous server failures. With timely graph repairing (Section 2.2), it can tolerate arbitrary number of failures over time.

Available membership management: Pangaea lets a replica be added by connecting to any k live replicas, no matter how many other replicas are unavailable. This property essentially allows popular files to remain always available.

Available replica location: Each directory entry links to multiple replicas of the file. Directories themselves are replicated just like any regular file. Thus, Pangaea allows path name traversal even when some servers are unavailable.

Inexpensive membership management: Adding or removing a replica involves only a constant cost, regardless of the total number of replicas.

The following sections introduce randomized protocols for dynamically reconstructing a graph in response to replica addition or removal.

2.1 Adding a replica

When a file is first created, the user's local server selects a few (two in the current implementation) servers in addition to itself. We currently choose them randomly using a gossip-based membership service [9], but we plan to investigate more intelligent placement in the future. Pangaea then creates replicas of the new file on the chosen servers and spans graph edges among them. These replicas, called *golden replicas*, are registered in the parent directory and used for path traversals (Section 2.3).

A replica is added when a user tries to look up a file missing from her local server. First, the server retrieves the file's contents from a nearby replica (found in the parent directory) and serves them to the user immediately. In the background, the server adds k ($k = 3$ in our implementation) bi-directional edges to existing replicas. Parameter k trades off availability (tolerating a node or edge loss) and replica-management overhead. Lacking a central replica-management authority, Pangaea chooses peer replicas using *random walks*, starting from a golden replica and performing a series of remote procedure calls along graph edges. Our simulation shows that a random-walk distance of three is enough to keep the graph diameter at $O(\log N)$, or 5 for a 1000-replica graph.

Directories are files with special contents and are replicated using the same random-walk-based protocol. Thus, to replicate a file, its parent directory must also be replicated on the same server beforehand. This recursive step potentially continues all the way to the root directory. The locations of root-directory replicas are maintained using the gossip-based membership service.

2.2 Removing a replica

A replica is removed from a file's graph either involuntarily or voluntarily. It is removed involuntarily when its host server remains unavailable for a certain period (e.g., a week). When a replica detects the death of a graph neighbor, it autonomously initiates a random walk and spans an edge with another live replica. Starting the walk from a live golden replica ensures that the graph remains strongly connected.

A replica is removed voluntarily when the hosting server runs out of disk space or decides that the replica is not worth keeping. The protocol is the same as above, except that the retiring replica proactively sends notices to its graph neighbors, so that they can replace edges immediately to minimize the period of low graph connectivity.

2.3 Maintaining hierarchical name space using golden replicas

Pangaea's decentralized replica maintenance approach has some downsides. In particular, it is not trivial to ensure a minimum replication factor for a file, or to ensure that a file's replicas are reachable from the parent directory even after repeated replica additions and removals.

We solve these problems by marking some replicas "golden" (currently, replicas created initially are marked golden.) The golden replicas form a complete subgraph in the replica graph to let them monitor each other and maintain a minimum replication factor. Non-golden replicas store one-way edges to golden replicas and use them as starting points for random walks. The file's entry in the parent directory also points to the golden replicas. Otherwise, golden replicas act exactly like other replicas. This design allows non-golden replicas of a file to be added or removed without affecting its parent directory, and non-golden replicas of the directory to be added or removed without affecting children files.

Golden replicas must be given a high priority to stay on disk, since adding or removing a golden replica is a relatively expensive operation. For example, removing a golden replica involves picking one non-golden replica, "engolden" it, and telling all replicas of the file and the parent directory to update their pointers to the golden-replica set.

3 Propagating updates

A wide-area file system faces two inherently contradicting goals: providing high end-to-end availability and maintaining strong data consistency. We decided to favor availability when a trade-off is inevitable, based on a recent study that reveals that most instances of write sharing can be predicted easily, and that they demand consistency only within a window of minutes [8]. Thus, Pangaea maintains both replica membership and file contents completely *optimistically* without any synchronous coordination—any user can read or update any replica any time.

We discuss our solutions to three particular challenges that arise from our approach: efficient update propagation, providing strong consistency, and conflicting updates.

3.1 Efficient propagation using harbingers

Unlike the Web or p2p data sharing systems, file systems must handle updates efficiently since they are relatively common and synchronous, i.e., the user often waits for a write to complete before issuing subsequent operations. A naive (but safe) approach for distributing updates to replicas would be to "flood" the change along graph edges hop by hop. However, that would consume network bandwidth k times as much as optimal and increase propagation delay, especially for large updates. Thus, Pangaea uses a two-phase strategy to propagate updates that exceed a certain size ($>$ 1K bytes in the current implementation). In the first phase, a small message that only contains the timestamp of the update, called a *harbinger*, is pushed along the graph edges. When a node receives a new harbinger, it asks the sender to push the update body. When a node receives a duplicate harbinger without having received the actual update,

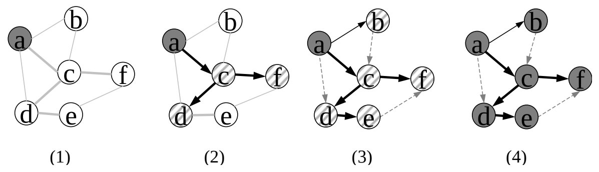

Figure 2: *An example of update propagation in Pangaea. Thick edges represent fast links. (1) An update is issued at A. (2) A sends a harbinger to the fat edge, C. C forwards the harbinger to D and F quickly. (3) D forwards the harbinger to E. After some wait, A sends harbinger to B, and a spanning tree is formed. Links not in the tree are used as backups when some of the tree links fail. (4) The update's body is pushed along the tree edges. In practice, steps 3 and 4 proceed in parallel.*

it asks the sender to retry later.[1] Moreover, before pushing (or forwarding) a harbinger to a graph edge, a node adds a slight delay inversely proportional to the estimated bandwidth of the edge. This way, for each update, Pangaea dynamically constructs a spanning tree whose shape closely matches the physical network topology. Figure 2 shows an example.

This harbinger algorithm yields two important benefits. First, it shrinks the effective window of replica inconsistency. When a user tries to read a file for which only a harbinger is received, she waits until the update body arrives. Because harbinger-propagation delay is independent of the actual update size, the chance of a user seeing stale file contents is greatly reduced. Second, it saves wide-area network usage, because it propagates updates along the fast edges. Our evaluation shows that this algorithm consumes only 5% more network bandwidth than an idealized optimal algorithm.

For users with strong consistency requirements, Pangaea also provides a mechanism to enforce consistency by synchronously circulating harbingers of recent updates. The user waits until all replicas of the files send back acknowledgments (or times out when remote nodes are unavailable). This mechanism may be initiated either manually by the user or automatically by Pangaea. The latter will be based on hints such as file names or application names, since write-sharing situations can often be predicted [8].

3.2 Conflict resolution

Because Pangaea requires no global coordination during any change, two nodes may issue different updates to the same object. Furthermore, some file operations—e.g., `mkdir` and `rename`—change two files (a file itself and its parent directory) that may encounter conflicts independently. We developed a proven-correct distributed protocol for this purpose. It is a variant of the version-vector-based algorithms used by Ficus and Roam [7], but we omit the details due to space constraints.

[1]The sender must retry the propagation later because the node that send the harbinger the earliest may crash before sending the update body.

4 Current status and future work

We are implementing Pangaea on Linux as a user-space loopback NFS server. Preliminary evaluations indicate that, in a uniformly configured network, its performance is comparable to that of Coda. In a geographically distributed setting, Pangaea outperforms Coda by being able to transfer data from closer nodes, not just from a statically configured central server.

We identify two key areas of future research. First is the study of intelligent file placement heuristics. We plan to consider information such as storage capacity, network bandwidth, and content type to optimize both performance and availability. Second is security. We plan to study end-to-end data and metadata encryption and protocols for tolerating Byzantine servers.

References

[1] William J. Bolosky, John R. Douceur, David Ely, and Marvin Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In *ACM SIGMETRICS*, pages 34–43, June 2000.

[2] Minkyong Kim, Landon P. Cox, and Brian D. Noble. Safety, visibility, and performance in a wide-area file system. In *FAST*. Usenix, January 2002.

[3] C. Mohan. A database perspective on Lotus Domino/Notes. In *ACM SIGMOD*, page 507, May 1999.

[4] Lily B. Mummert, Maria R. Ebling, and M. Satyanarayanan. Exploiting weak connectivity for mobile file access. In *SOSP*, pages 143–155, December 1995.

[5] Athicha Muthitacharoen, Benjie Chen, and David Mazieres. A low-bandwidth network file system. In *SOSP*, pages 174–187, October 2001.

[6] Karin Petersen, Mike J. Spreitzer, Douglas B. Terry, Marvin M. Theimer, and Alan J. Demers. Flexible update propagation for weakly consistent replication. In *SOSP*, pages 288–301, October 1997.

[7] David H. Ratner. *Roam: A Scalable Replication System for Mobile and Distributed Computing*. PhD thesis, UC Los Angeles, 1998.

[8] Susan Spence, Erik Riedel, and Magnus Karlsson. Adaptive consistency–patterns of sharing in a networked world. Technical report, HP Labs, 2002.

[9] R. van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. In *IFIP Int. Conf. on Dist. Sys. Platforms and Open Dist. Proc. (Middleware)*, 1998.