Appeared in the Proceedings of the
Second IEEE Workshop on Experimental Distributed Systems,
Huntsville, AL, October 1990, pages 20-25

Name Transparency in Very Large Scale Distributed File Systems[*]

Richard G. Guy   Thomas W. Page, Jr.   John S. Heidemann
Gerald J. Popek[†]

Department of Computer Science
University of California Los Angeles

Abstract

Previous distributed file systems have relied on either convention or obtaining dynamic global agreement to provide network transparent file naming. This paper argues that neither approach can succeed as systems scale to the kind of size that is anticipated in the current decade. We propose instead a novel name mapping scheme which relies on a fragmented, selectively replicated name translation database. Updates to the naming database are coordinated by an optimistic concurrency control strategy with automatic propagation and reconciliation. A prototype of the name mapping mechanism has been implemented and is in use in the Ficus replicated file system.

1 Introduction

During this decade, very large scale wide area distributed computing environments (DCEs) will emerge. We believe that the principle of *network transparency* will be even more important to large scale DCEs than it has been in current small scale DCEs. A key challenge of providing network transparency is found in file systems, which must provide *location transparent* and *name transparent* naming services.

This paper argues that name transparency is essential to the success of a very large scale wide area distributed computing environment. It describes the difficulties of scaling current small scale name transparency techniques, and proposes a new mechanism suitable for use in a very large scale DCE.

Section 2 describes salient characteristics of a large scale DCE. File system aspects of network transparency are outlined in Section 3, followed by a discussion of name transparency in Section4. The naming mechanisms of several distributed file systems are studied in Section 5. Section 6 then presents a new mechanism which relies on *optimistic replication* to support name transparency. We present our conclusions in Section 7.

2 Large scale

Our model of large scale distributed systems is based on our perspective of the directions we expect computer networking to take in the next decade. We anticipate wide area internetworks of a million or more hosts, much larger than the current DARPA Internet. Large scale systems will display great diversity of host and communications capability and availability. Hosts will range from pocket computers to supercomputers; mobile computers will often have unreliable communications links, while other hosts will be highly redundantly con-

[*]This work was sponsored by DARPA under contract number F29601-87-C-0072.

[†]This author is also affiliated with Locus Computing Corporation.

Appeared in the Proceedings of the
Second IEEE Workshop on Experimental Distributed Systems,
Huntsville, AL, October 1990, pages 20-25

nected. Many communications links will routinely be periodically unavailable, in response to varying tariffs, security concerns, scheduled maintenance, and so on. The potential for unintentional service denial will result not only from broken hardware and software, but also from overburdened gateways and limited bandwidth channels. Some components will be permanently taken out of service, with no intended replacement.

The resulting critical characteristic of a very large system is *continuous partial operation*: the entire system will *never* be completely operational simultaneously. Some portions of the system may seldom be able to communicate directly with each other.

A wide area, large scale distributed system of this magnitude will inevitably span many administrative and organizational boundaries. These bodies will have overlapping, but often conflicting interests; they will be mutually suspicious of each other.

The foundation for large scale distributed computing environments has been laid by networks such as the DARPA Internet and Bitnet. Corporate environments already provide partially transparent solutions. Digital's DECNET, for example, provides remote file access to tens of thousands of users. Further impetus will come from the proposed National Research Network[4], intended to provide high bandwidth communications between thousands of universities, colleges, and research laboratories.

3 Network transparency

The goal of network transparency is to enable clients to work effectively without knowledge of myriad system configuration details, especially host boundaries and network characteristics. Network transparency has become widely accepted in medium scale local area network distributed file systems; we argue that it is even more critical in very large scale environments. In the file system arena, we wish to discuss three aspects of network transparency: location transparency, name transparency, and replica transparency.

A *location transparent* name contains no information about the named object's physical location. Location transparent naming is critical for flexibility in managing storage resources; it is essential to enable movement of resources from one site to another, and to support such high availability services as multisite replication.

A naming system is described as *name transparent* when the result of mapping a name to an object is independent of the host from which the name is uttered. This property is essential for very large scale distributed computing environments: clients cannot be expected to learn the naming peculiarities of each host they use; often, they may not even be aware from which host a name is uttered or that multiple hosts may be involved.

A file system that supports file replication is *replica transparent* when a client is normally unaware that more than a single copy of a file exists; replica management is largely automatic.

4 Name transparency

Complete name transparency implies that all components of a distributed system support a common global name space, and no other name space. In practice, complete name transparency is too limiting: for example, system configuration files often contain host-specific contents, and the result of name mapping should be determined by the context (i.e., the host) in which a name is uttered. The LOCUS [6] file system and the cellular Andrew File System [9] contain mechanisms which specifically address this exception to complete name transparency.

We believe that complete name transparency (with the exception noted above) and the implied single, common global name is feasible for a very large scale DCE. The LOCUS file system has demonstrated the feasibility of transparency for small scale systems; Sun Microsystems' Network File System (NFS) [7] has demonstrated trans-

Appeared in the Proceedings of the
Second IEEE Workshop on Experimental Distributed Systems,
Huntsville, AL, October 1990, pages 20-25

parency for small and medium scale systems, yet NFS also reveals the problems of supporting name transparency without also providing a single, common global name space.

The feasibility of a large scale common global name space assumes that solutions to serious technical and administrative issues can be found. In the remainder of this section we consider administrative difficulties; the following sections address technical concerns.

The foremost administrative problem is obtaining agreement upon the structure and management of the global name space. Several large communities have demonstrated that such agreement is possible: the host naming conventions adopted by the DARPA Internet community is a good example[3, 8].

The Internet host name space is a hierarchy. The uppermost level contains relatively few entries; substantial agreement within the community is required to add entries at this level, and modification or deletion of entries is extremely rare (due to the enormous impact of a name change). The result is that this level is an essentially static portion of the name space.

The second level of the Internet host name space consists of organization names. Each organization has limited freedom to choose its own name: any name may be proposed, but the community sharing the specific portion of the name space may exercise veto power over the choice if it seems misleading. The third (and any number of subsequent) levels of naming are completely under the control of each organization.

A useful balance is found in the Internet name space between organizational autonomy in selecting names and community interest in limiting the ability of individuals to change the upper levels of the name space. The relatively static nature of that portion of the name space which is most critical in ensuring name transparency enables consistent, yet efficient mechanisms to support a large scale, name transparent name space.

5 Global name space maintenance

In this discussion of name space support, terminology from UNIX will be used. However, the reader should be easily able to translate into his own context. For example, the problems addressed here occur in a similar form in the X.500 approach, and appear amenable to similar solutions.

Hierarchical name spaces, such as those used by file systems, are commonly composed of disjoint sub-hierarchies of names. We will call these sub-hierarchies *volumes*. Volumes are typically “glued” together by associating the root node of one volume with a leaf node in another volume. One volume is usually designated as the root volume of the name space.

In the UNIX world, the activity of “gluing” name hierarchies together is commonly known as *mounting* a hierarchy. The definitions of the associations are stored in a *mount table*. In a single host system, only one mount table exists, but in a distributed file system the equivalent of the mount table must be a distributed data structure.

Providing a name transparent, distributed global name space requires a mechanism to ensure coherence of the distributed name translation database containing the mount information. This can be achieved either through convention (as in NFS), or automatically (as in LOCUS).

In NFS, each host independently maintains its own mount table. A mount table must contain a definition for each volume (local or remote) which is nameable from the host. Thus, each host in a distributed system maintains a number of redundant mount table entries to support name transparency. No explicit mechanism exists to coordinate changes to the mount tables.

LOCUS incorporates a replicated mount table: each host works closely within a set of communicating hosts to maintain agreement among the mount table replicas. Each mount table replica lists all extant mount definitions.

Appeared in the Proceedings of the
Second IEEE Workshop on Experimental Distributed Systems,
Huntsville, AL, October 1990, pages 20-25

Neither of these two approaches to maintaining a common view of the name space is amenable to scaling up to large numbers of hosts. Convention and manual agreement (as in NFS) is unworkable in such a large scale. The tightly coupled global agreement mechanism in LOCUS does not scale well either, as it relies upon a common network partition-wide view of partition membership. Reaching consensus is fairly expensive, and so partition stability is important.

In a very large DCE, partition status will be continually changing, as opposed to occasional change in a small scale system.

Both approaches also suffer from monolithic mount tables. Small scale DCEs seldom have more than hundreds of mount table entries, but large scale systems can be expected to contain large numbers of volumes. The burden of supporting frequent changes[1] to a large, globally replicated table is especially undesirable when only a tiny (although unpredictable) fraction of the volumes will ever be examined by clients from any given host.

6 Fragmented, selectively replicated mount tables

The key to realizing name transparency in a very large scale environment is to recognize that monolithic, globally replicated mount tables are not necessary for providing a transparent name space. Rather, mount table information should be fragmented and selectively replicated, and an optimistic approach to name translation and automatic database consistency employed.

6.1 Mount table fragments

The mount table is a partial mapping between nodes in one volume and root nodes of another. The semantics of the mapping are that when a node marked as a *mount point* is encountered during name translation (path name expansion), the mount table is consulted and the corresponding root node is substituted for the mount point. A specific mount table entry is then required only when the corresponding mount point is encountered. The mount table information can therefore be fragmented and placed exactly (and only) at mount points, where it is always available precisely when it is needed.

In standard UNIX, each mount point contains the name of the storage device which houses the volume to be mounted. In a distributed file system such as Ficus[1] or NFS, the network location of the volume to be mounted is also required.[2]

Any given host will only ever access a tiny percentage of the millions of volumes on the network. A lazy evaluation mount approach allows only those volumes in use to consume local resources. When a mount point is traversed a volume is automatically mounted, transparently to the user; volumes which remain unused for a set period of time are automatically unmounted.

6.2 Fragment replication

A replicated volume can contain mount points, which are replicated just as is any other volume object. Each volume replica that contains a name for a mount point also contains a complete mount point replica, so the resulting availability of the mounted volume is equivalent to that achieved by a globally replicated monolithic mount table.

As with any replicated object, mount point update consistency and availability are primary concerns. Mount point updates occur when volume replicas listed in the mount point are deleted, moved to another host, or new volume replicas created and must be listed.

[1] While change to any single table item is infrequent, the aggregate update rate to the table as a whole would be much higher.

[2] The storage device name is replaced by a "device location transparent" identifier for the volume: NFS uses a host context dependent name; Ficus uses a separate "volume" identifier.

Appeared in the Proceedings of the
Second IEEE Workshop on Experimental Distributed Systems,
Huntsville, AL, October 1990, pages 20-25

6.3 Update consistency

Mount point updates are rare, but typically occur during periods of network instability and unreliability. For example, one of the greatest motivations to add more volume replicas is a situation in which very few, (perhaps only one) volume replica is accessible. Standard serializable techniques (primary copy, quorum consensus, majority voting and its variants) are not appropriate for this case. Instead, we propose the use of optimistic consistency methods.

In [5, 2], we argue that optimistic approaches to replicated file management are essential in very large scale distributed filing environments. The optimistic philosophy provides equivalent read and update availability by allowing either activity so long as at least one replica is available. The actual occurrence of conflicting, unsynchronized updates is rare, favoring the a posteriori detection and repair of inconsistencies over the more restrictive limitations of pessimistic serializable algorithms.

Interestingly, the very same algorithms utilized in [2] for detecting and automatically reconciling directory updates are directly applicable to the fragmented, replicated mount table structure. The technique is robust with respect to delayed update propagation: data is self-validating when used, in a manner analogous to lazy evaluation.

A prototype of several key aspects of the solution has been built (see [1]) and will serve as a basis for further evaluation of the impact of large scale in practice.

7 Conclusion

This paper has outlined an example of the crucial role an optimistic philosophy plays in the design of a very large scale distributed system. The dynamic mount mechanism described above has been successfully implemented in the Ficus replicated file system, though it has yet to be tested in a large scale environment.

Acknowledgements

The authors wish to acknowledge the contributions of Dieter Rothmeier and Wai Mak to the implementation of the Ficus replicated file system.

References

[1] Richard G. Guy, John S. Heidemann, Wai Mak, Thomas W. Page, Jr., Gerald J. Popek, and Dieter Rothmeier. Implementation of the Ficus replicated file system. In *USENIX Conference Proceedings*, pages 63–71. USENIX, June 1990.

[2] Richard G. Guy and Gerald J. Popek. Reconciling partially replicated name spaces. Technical Report CSD-900010, University of California, Los Angeles, April 1990.

[3] P. Mockapetris. Domain names: Concepts and facilities. Network Working Group Request for Comments: 1034, November 1987.

[4] National Research Council National Research Network Review Committee. Toward a national research network. National Academy Press, 1988.

[5] Thomas W. Page, Jr., Gerald J. Popek, Richard G. Guy, and John S. Heidemann. The Ficus distributed file system: Replication via stackable layers. Technical Report CSD-900009, University of California, Los Angeles, April 1990.

[6] Gerald J. Popek and Bruce J. Walker. *The LOCUS Distributed System Architecture*. The MIT Press, 1985.

[7] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon. Design and implementation of the Sun Network File System. In *USENIX Conference Proceedings*, pages 119–130. USENIX, June 1985.

[8] M. Stahl. Domain administrators guide. Network Working Group Request for Comments: 1032, November 1987.

Appeared in the Proceedings of the
Second IEEE Workshop on Experimental Distributed Systems,
Huntsville, AL, October 1990, pages 20-25

[9] Edward R. Zayas and Craig F. Everhart. Design and specification of the cellular Andrew environment. Technical Report CMU-ITC-070, Carnegie-Mellon University, August 1988.