A Highly Available Network File Server

Anupam Bhide
IBM T.J. Watson Research Center

Elmootazbellah N. Elnozahy *
Department of Computer Science, Rice University

Stephen P. Morgan
IBM T.J. Watson Research Center

Abstract

This paper presents the design and implementation of a Highly Available Network File Server (HA-NFS). We separate the problem of network file server reliability into three different subproblems: server reliability, disk reliability, and network reliability. HA-NFS offers a different solution for each: dual-ported disks and impersonation are used to provide server reliability, disk mirroring can be used to provide disk reliability, and optional network replication can be used to provide network reliability. The implementation shows that HA-NFS provides high availability without the excessive resource overhead or the performance degradation that characterize traditional replication methods. Ongoing operations are not aborted during fail-over and recovery is completely transparent to applications. HA-NFS adheres to the NFS protocol standard and can be used by existing NFS clients *without* modification.

1 Introduction

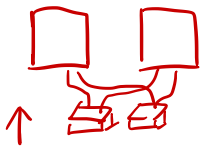
Traditional approaches for providing reliability in network file systems by server replication suffer from excessive resource overheads, performance degradation, and increased complexity. Replicated servers use expensive protocols to maintain consistency and coherence, leading to performance degradation during failure-free operation. They also use complex protocols to update the state of a stale replica when it is repaired after failure. Further, handling network partition requires quorum management, increasing system complexity.

This paper describes the design and implementation of a Highly Available Network File Server (**HA-NFS**) that adheres to the semantics of SUN's Network File System[1] (NFS) [1]. HA-NFS differs from traditional approaches in that it considers the problem of providing a reliable network file system as three separate subproblems, namely: recovering from server failures, recovering from disk failures, and recovering from network failures. HA-NFS offers a different solution to each of these subproblems.

Server failures are tolerated by using dual-ported disks that are accessible to two servers, each acting as a backup for the other. The disks are divided into two sets, each served by one server during normal operation. Each server maintains

*Supported by an IBM fellowship.

[1]Network File System and NFS are trademarks of Sun Microsystems, Inc.



on *its* disks enough information to reconstruct its current volatile state. The two servers periodically exchange liveness-checking messages. If one server fails, its disks will be taken over by the other server, which will reconstruct the lost volatile state using the information on disk. Then, it *impersonates* the failed server, and operation continues with a potential reduction in performance due to the increased load. The machines on the network are oblivious to the failure, and continue to access the file system using the same address. During normal operation, the servers communicate only for periodic liveness-checking. The servers do not maintain any information about each other's volatile state or attempt to access each other's disks.

Fast recovery from disk failures is achieved by mirroring files on different disks. However, all copies of the same file are on disks that are controlled by the same file server, eliminating the overhead of ensuring consistency and coherence between the two servers that would otherwise occur. Since disk failures are not frequent, mirroring is only used for applications that require continuous availability. Otherwise, archival backups could be used to recover from disk failures.

Network failures are tolerated by optional replication of the network components, including the transmission medium. However, packets are not replicated over the two networks. Instead, the network load is distributed over the networks.

HA-NFS servers conform with the server protocol of SUN's NFS. NFS has gained wide acceptance as a general purpose network file system. By adhering to a standard file system, our results can have direct application in practical environments. In addition to adherence to standards, our design has several important goals:

- Failure and recovery must be completely transparent to applications running on the file-server's clients. A failure must not force operations in progress to terminate.
- Failure-free performance must not be penalized to provide high availability.
- NFS client protocol implementations should not require modification to use HA-NFS servers.

We have implemented a prototype of HA-NFS on a network of workstations and two file servers from the IBM RISC System/6000 family[2] of computing systems running the AIX Version 3 (AIXv3) operating system, and connected by both a 10 Mbit/s Ethernet network and a 4 Mbit/s token ring network. We construct dual-ported disks from off-the-shelf SCSI disks attached to a SCSI bus that is shared by the two servers. The prototype is operational and has satisfied the design goals.

In section 2, we present background information on NFS and AIXv3. We present the design of an HA-NFS server in section 3. We discuss the performance in section 4. We compare our design to related work in section 5. Finally, we draw conclusions and outline future work in section 6.

2 Background

HA-NFS is implemented on top of the AIXv3 journaled file system. The AIXv3 file system provides serializable and atomic modification of file system meta-data by using transactional locking and logging techniques. File system meta-data are composed of directories, inodes, and indirect blocks.

[2]RISC System/6000 and AIX are trademarks of International Business Machines Corporation.

Every AIXv3 system call that modifies the meta-data does so as a transaction, locking meta-data as they are referenced, and recording the changes in a disk log before allowing the meta-data to be written to their "home" locations on disk. In the case of system failure, the meta-data are restored to a consistent state by applying the changes contained in the log. The reliability of ordinary files is ensured by NFS semantics, which require forcing the data to disk before sending an acknowledgement to the client.

AIXv3 supports *logical volumes*, which provide the abstraction of logical disks. Logical volumes can be mirrored to provide disk reliability. Each logical volume can have up to three copies, each on a different physical disk.

Although NFS is defined as a stateless file server protocol, most NFS implementations maintain a small amount of state information. Some NFS operations, such as erasing a file, cannot be implemented by idempotent remote procedure calls (RPC). An NFS server maintains a *reply cache*[3] to remember successful, non-idempotent RPC's. If an RPC is unsuccessful, the server uses the reply cache to tell whether the RPC is a retry of a previously successful, non-idempotent RPC. If it is, the server responds to the client that the RPC completed successfully. HA-NFS records changes to this volatile state in the AIXv3 disk log, so that the reply cache can be reconstructed in the case of failure.

3 HA-NFS Architecture

An HA-NFS node consists of two NFS servers sharing a number of SCSI buses. Each shared SCSI bus and the disks connected to it have one of the servers designated as their *primary server*. During normal operation, the disks are served only by their corresponding primary server, the other server does not access the bus. The primary server for each bus is selected such that the total load is balanced (statically) over the two servers. Both servers act as backups for each other. NFS clients perceive an HA-NFS node as two independent NFS servers, each serving a distinct set of file systems.

Each server has two network interfaces and IP addresses. The server uses its *primary interface* for normal operation, and its *secondary interface* when *impersonating* the other server after its failure. The server also uses its secondary interface when re-integrating with the system after repair or maintenance.

Figure 1 shows a single HA-NFS node consisting of two servers on a single network.

3.1 Normal Operation

During normal operation, a server performs the operation described in each NFS RPC it receives. If the operation is successful, the server will record the meta-data changes in the AIXv3 file system log and enough information to identify the RPC if it is non-idempotent. (This information is identical to that in the volatile reply cache and will be used in the case of failure to reconstruct the volatile state.) If the operation completes successfully and is non-idempotent, the server will add an entry in its reply cache for the RPC.

If the operation did not complete successfully, the server will determine whether there is an entry in the reply cache corresponding to the RPC. If an entry is found, then the RPC is a retry of a non-idempotent operation that succeeded before. The server will reply to the client that the RPC completed successfully; otherwise, the server replies to

[3] Also called a duplicate cache.

the client with an appropriate error code, indicating the failure of the requested operation.

Both servers in the node exchange NFS RFS_NULL RPC's to monitor the liveness of each other. An RFS_NULL is a "no operation" RPC that is echoed back from the server if it is running. If a server does not receive an acknowledgement for the RFS_NULL after a specified number of such RPC's, it will start failure detection. First, it checks if it can "ping" the suspect server by sending an ICMP echo packet. Second, it attempts to communicate with the suspect server via the shared SCSI bus in "target mode". This is analogous to pinging except that the requests are sent over the SCSI bus rather than the network, and the response is sent by a device driver which must respond within a certain period of time. Both the ICMP communication and the target mode communication on the SCSI bus are performed conceptually at the interrupt handler level. Thus, a response is generated even though the server may be so overloaded that it cannot respond to NFS RPC's.

If a response is obtained from either of these two tests, it is likely that the suspect server is undergoing a period of slow response. The failure detection tests are conservative, so it is possible that the tests indicate that a server is alive while it is "brain-dead", i.e., able to respond correctly to the tests, but incapable of processing NFS RPC's. In such unlikely cases, HA-NFS refrains from continuing the take-over and relies on operator intervention. In a network, it is impossible to determine with absolute certainty whether a certain machine has failed [2]. However, the failure detection tests never declare a server dead while it is operational. This prevents a race condition where both servers attempt to access all the disks in the node at the same time, which can lead to corruption of the file systems.

3.2 Take-over

If a server fails, its disks will be taken over by the other server. The live server brings the failed server's volume groups on-line by running their logs and restoring the file systems to a consistent state. The server also uses the log to retrieve the reply cache entries of the failed server and inserts them in its own cache. Then, it starts impersonating the failed server by changing the IP address of its secondary network interface to the primary address of the failed server. The live server also changes the hardware address of its secondary interface to that of the primary interface of the failed server. Thus, packets that were intended for the failed server can now be received by the live server on its secondary interface.

If network interfaces that can change their hardware address are not available, an alternate scheme may be used to allow the live server to receive the packets intended for the failed one. The scheme consists of using the ARP [3] protocol to update the mapping between the failed server's IP address and the hardware address to reflect the change. HA-NFS updates stale mappings in the clients' ARP caches by sending an ARP request which queries for the hardware address corresponding to some machine's IP address. The query appears to have been sent from the failed server's IP address, but with the live server's secondary interface as the source hardware address. On receiving this ARP broadcast, each machine on the local network updates its mappings to reflect the change. The update is automatically performed by the ARP protocol layer on the clients, so no modification in the network software is necessary. The broadcast is repeated several times to ensure that virtually all

clients on the local network will eventually receive it.

We have decided to use the hardware address change approach in our final prototype since the ARP approach relies on the correct implementation of the ARP protocol on all types of clients. However, the take-over time reported in section 4 was measured using the ARP approach since at that time we did not have the special type of network interfaces that allowed changing hardware addresses.

During take-over, clients of the failed server continue to retransmit their requests. When the live server starts to impersonate the failed one, it receives the clients' requests and begins to serve them. Clients are oblivious to the change, all they can detect is that the server has gone through a period of slow response.

3.3 Re-Integration

When a server comes up, either normally or after repair or maintenance, it cannot immediately configure its primary network interface to its primary IP address, since it may be impersonated by its backup.

Instead, the server comes up with its primary network interface turned off, and uses its secondary interface to send a re-integration request to the backup.

If the backup is running, it will acknowledge the receipt of the re-integration request. After unmounting the corresponding file systems, the backup switches the IP and hardware addresses of its secondary network interface back to their normal settings, thereby stopping the impersonation of the other server. Finally, the backup sends a message to the re-integrating server allowing it to proceed. The re-integrating server reclaims its SCSI buses and disks, runs the log and reconstructs the reply cache, switches its own primary interface on, and starts serving NFS requests.

Care is taken to recover from failures of either server during re-integration. The servers periodically exchange liveness messages until the backup relinquishes the buses. Communication through the SCSI buses will also be used if either server suspects the failure of the other. If the re-integrating server fails, the backup will reclaim the disks as in take-over. If the backup fails, the re-integrating server will start normal operation on its own. Later, it will start impersonating the failed backup.

3.4 Network Failure

To tolerate network failures, HA-NFS relies on replicating the network. Figure 2 shows an HA-NFS node in a two-network configuration.

Recovery from server failures does not require any changes to clients. Recovery from network failure, however, requires a daemon to run on the client to observe the status of each network and reroute requests to the operational network if a failure occurs. Since the daemon is run as a user process, no change to the kernel or to the NFS protocol is necessary.

When an HA-NFS node is connected to two networks, each server has its network interfaces connected to different networks. Further, the two servers have their primary interfaces on different networks. Thus, the servers receive their requests on different networks, which provides a degree of network (static) load balancing. Clients are also configured to (statically) balance the load on both networks.

In addition to its roles during take-over and re-integration, the secondary network interface now

serves as an alternate path to the server, should its primary interface becomes unreachable because of a network failure. Each server broadcasts a "heartbeat" message from its primary interface. The daemon on every client detects the heartbeats of the servers on both networks. When the daemon detects the loss of the heartbeat of one server, the daemon concludes after a timeout period that the path to the server's primary network interface is broken. In this case, the daemon updates the client's routing table to use the alternate path to the server. The daemon also sends a request to the daemon on the server to update the routes for RPC acknowledgements to that client to the operational network, if necessary[4].

Once the daemon detects the return of the server heartbeat on a broken path, it restores routing to its normal setting and requests that the server reroute the path for the RPC acknowledgements, if necessary.

When a server takes over the role of its counterpart in the HA-NFS node, it needs to broadcast a heartbeat on behalf of the failed server, so that clients continue to believe that the server is still reachable across its default path. The server will set its secondary interface's IP address to that of the failed server's primary interface (which is on the same network). Combinations of network and server failures are tolerated. For example, a server taking over the role of a failed server may face a failure on the network on which the failed server's primary interface resides. In this case, the daemons on the clients should route requests for the failed server to the primary interface of the live server, since the secondary interface used for impersonation is now unreachable.

[4]If the client's main address is on the operational network, then the server need not reroute the acknowledgements.

4 Performance

HA-NFS provides high availability without incurring excessive performance penalty. We measured the performance of HA-NFS by running a set of experiments on a number of RISC System/6000 family workstations (25 MHz), connected by a 10 Mbit/sec Ethernet. All measurements were obtained by directly calling the SUN RPC layer, bypassing the NFS client cache. The underlying system uses 4 KByte disk blocks.

4.1 The Effect of Disk Logging

Table 1 shows a comparison between HA-NFS, and a traditional implementation of NFS that does not use disk logging. The traditional implementation of NFS forces the data and the meta-data to their home locations on disk before responding to the RPC. In contrast, HA-NFS records meta-data modifications as a log record, requiring no disk arm movement. The meta-data are written asynchronously to their home locations. Because the reply cache entries are piggybacked on the normal disk log information, saving the volatile state on disk does not incur appreciable overhead beyond the cost of basic disk logging. As expected, disk logging improves the response time of all RPC's that modify the file system structure, such as SETATTR, CREATE, REMOVE, MKDIR, and RMDIR RPC's. The table shows that disk logging improvement ranges from 33% for SETATTR and WRITE RPC's, up to 75% for MKDIR RPC.

In the above measurements, the log was placed on a separate disk. Placing the disk log on the same disk with the data reduces the performance

gain due to the extra disk arm movement. For example, the improvement in CREATE RPC drops from 58% to 20%.

	HA-NFS (ms)	NFS (ms)	Improvement %
NULL	5.26	5.26	0
GETATTR	6.04	6.04	0
SETATTR	32.08	48.32	33
LOOKUP	6.96	6.96	0
READ	12.13	12.13	0
WRITE	72.28	108.80	33
CREATE	38.07	91.81	58
REMOVE	35.22	87.37	60
RENAME	35.58	75.96	53
MKDIR	37.20	151.47	75
RMDIR	34.73	118.40	70
READDIR	11.08	11.08	0
STATFS	6.05	6.05	0

Table 1: Traditional NFS vs. HA-NFS.

4.2 The Effect of Mirroring

The only overhead introduced by mirroring is a 17% slow-down for the WRITE RPC. This overhead is attributed to the variation in the disk arm position among the mirrors at the time of writing to disk and to the performance overhead of the mirroring software. When client caching is turned on, the overhead drops to 2% at the application-program level.

Because of disk logging, mirroring does not introduce any overhead to the RPC's that maintain the file system structure (eg. CREATE).

4.3 Take-over and Re-integration

A second set of measurements shows how long it takes a backup to take over the failed server's role, and how long it takes a recovering server to re-integrate with the rest of the system.

Failure detection consists of an empirically chosen timeout period of 10 seconds and a number of precautionary tests for liveness which take 5 seconds (these parameters are configurable.). We measured a total time of 15 seconds for the backup to perform all take-over operations, excluding failure detection. Thus, the backup takes about 30 seconds before starting to serve the disks of the failed server. During that time, the disks are not available.

We measured a total time of 60 seconds for a server to re-integrate into the system after repair. Re-integration takes longer than fail-over because the backup must wait for ongoing NFS RPC's to terminate in addition to the overhead of unmounting the corresponding file systems.

5 Related Work

HA-NFS is unique in that it considers the problem of providing reliability to file servers as three separate subproblems, namely: server reliability, disk reliability, and network reliability. Each subproblem is handled differently. Logging the server's volatile state to a disk accessible to the backup tolerates server failures, optional replication of disks tolerates disk failures, and optional replication of the network components tolerates network failures.

HA-NFS does not suffer from the problems associated with the traditional approaches based on replicating the file server as a unit [4, 5, 6, 7, 8, 9]. Replicating the file server as a unit introduces additional overhead during failure-free operation due to the need to enforce consistency among the replicas. Re-integrating a recovering server into the system can be expensive since it requires updating

the server's stale view of the replicated file system. To tolerate network partition, a replication-based system must support read and write quorums, incurring a substantial performance penalty. This penalty has led some systems [4, 9] to abandon quorums, allowing divergence in replicas during network partition. While this solution may be acceptable in many practical environments, it cannot be relied on in general and it exposes failures to the users. In HA-NFS, we direct our effort to making the network more reliable independent of the solution we employ to provide server reliability. After all, the effects of a network partition are not limited to file servers. Also, the availability of a replicated file server is greatly compromised for the clients that are in a partition without a replica.

On the other hand, replicated file servers can distribute the file-read load on the different replicas to achieve load balancing. Replicated file servers are also better suited for wide area networks where a client can access its files from the nearest replica, reducing network load during file read. A recent comparison study with the Deceit file server [10] shows that the performance benefits of HA-NFS and its relative simplicity come at the expense of a lack of flexibility. Both servers of an HA-NFS node must be physically close to each other because of the restriction on the SCSI bus length. HA-NFS cannot resist site disasters or total site failures. Installing an HA-NFS node is complicated by the provisions taken to ensure independent failure modes of the servers and the disks in the node.

Like HA-NFS, several reliable file servers attempt to provide reliability to NFS without changing the client implementation of the protocol [5, 6, 8]. HA-NFS is unique in that it uses impersonation to mask fail-over from the clients. In the other systems [5, 6], the clients continue to attempt to access the files from the failed server, therefore "hanging" until the user intervenes and remounts the file systems from an alternate source. The reliable file system of MIT [8] suggests the use of IP multicast addressing to solve this problem, but no implementation has been reported. When compared to impersonation, IP multicast increases the load on the replicas and introduces complexity, since all replicas must process the multicasts in the same order.

Using dual-ported disks is also the basis of the reliable file system of Tolerant [11] and the Echo [12] reliable file system. Tolerant and HA-NFS are similar in that they rely on a non-dedicated backup to provide reliability against server failure. However, Tolerant relies on transaction semantics at the application level, and ongoing transactions are aborted during fail-over. In contrast, HA-NFS does not rely on transaction semantics at the application level, and ongoing operations are not affected during fail-over. HA-NFS differs from Echo in that an HA-NFS backup does not maintain information about the current volatile state of the main server, and that HA-NFS clients are oblivious to the backup take-over. In Echo, the primary informs the backup about its state, and each client has a "clerk" layer that isolates the application programs from failures and recoveries.

6 Conclusions and Future Work

As modern computer hardware becomes intrinsically more reliable, traditional solutions to provide reliability in network file systems by server replication become less attractive because of the performance penalty and the complexity they incur. We have presented HA-NFS, a reliable file server

based on offering different treatment to the reliability of each component of the file system. Our approach offers server reliability by using dual-ported disks and impersonation, disk reliability by using mirroring, and network reliability by replication. HA-NFS is one of the few designs that recognize that network failures require an independent solution from that used to provide reliability to the file server.

Comparing the performance of HA-NFS with traditional implementation of NFS shows that disk logging improves the performance of many NFS RPC's. Mirroring, when used, adds only 17 % overhead to WRITE RPC. Saving the reply cache entries is piggybacked on the normal disk logging operation, thus adding no more overhead beyond that of the basic disk logging.

Recovery in HA-NFS is completely transparent to applications and does not involve aborting ongoing operations. Impersonation prevents the client from "hanging" during fail-over. User programs and NFS client protocol implementation need no modification to use HA-NFS. HA-NFS shows that it is possible to add transparent fault-tolerance to existing systems without adding significant overhead.

The high performance and relative simplicity of HA-NFS come at the expense of a loss in flexibility. HA-NFS cannot tolerate more than one server failure. During fail-over, the disks are not available for a period of 30 seconds. The servers must be physically close because of the restriction on the length of the SCSI bus. HA-NFS cannot tolerate total site failures or site disasters.

We are currently addressing the shortcomings of HA-NFS. We are considering the use of optical links instead of the shared SCSI bus to simulate dual-ported disks. Because of their high performance and capability of extending over relatively long distances, optical links can remove the restrictions of installing servers and their disks in close proximity. This will also facilitate the realization of independent failure conditions and allow more than two servers to share the same disk. We are considering adding stable semiconductor memory on the disk controller to remove all the overhead of disk logging. We are also considering adding extensions to HA-NFS operations to support consistency of concurrent file access in the presence of client caching. Finally, we plan to use the HA-NFS methodology to provide higher availability for stateful server protocols such as Andrew [13].

References

[1] Sun Microsystems, Inc. NFS: Network file system protocol specification. RFC 1094, Network Information Center, SRI International, March, 1989.

[2] M.J. Fisher, N.A. Lynch, and M.S. Paterson. Impossibility of distributed consensus with one faulty process. *JACM*, 32(2):374–382, April 1985.

[3] D. C. Plummer. Ethernet address resolution protocol; RFC 826. In *ARPANET Working Group Requests for Comments, no. 826.* SRI International, Menlo Park, California, November 1982.

[4] Mahadev Satyanarayanan, James J. Kistler, Puneet Kumar, Maria E. Okasaki, Ellen H. Seigel, and David C. Steere. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4):447–459, April 1990.

[5] Keith Marzullo and Frank Schmuck. Supplying high availability with a standard network file system. In *Proceedings of the 8th International Conference on Distributed Computing Systems*, pages 447–453, May 1988.

[6] Alex Siegel, Kenneth Birman, and Keith Marzullo. Deceit: A flexible distributed file system. Techni-

cal Report TR 89-1042, Cornell University, November 1989.

[7] Uppaluru Premkumar, W. Kevin Wilkinson, and Hikyu Lee. Reliable servers in the JASMIN distributed system. In *Proceedings of the 7th International Conference on Distributed Computing Systems*, pages 105–112, September 1987.

[8] Barbara Liskov, R. Gruber, P. Johnson, and L. Shrira. A replicated Unix file system. In *Proceedings of the First IEEE Workshop on Management of Replicated Data*, pages 11–14, November 1990.

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

[10] Anupam Bhide, Elmootazbellah N. Elnozahy, Stephen Morgan, and Alex Siegel. A comparison between two reliable file servers. In preparation.

[11] Dale L. Shipley, Joan D. Arnett, William A. Arnett, Steven D. Baumel, Anil Bhavnani, Chuenpu J. Chou, David L. Nelson, Maty Soha, and David H. Yamada. Distributed multiprocess transaction processing system and method. U.S. Patent No. 4819159, April 1989.

[12] Andy Hisgen, Andrew Birrell, Timothy Mann, Michael Schroeder, and Garret Swart. Availability and consistency tradeoffs in the ECHO distributed file system. In *Proceedings of the 2nd Workshop on Workstation Operating Systems*, pages 49–54, 1989.

[13] John H. Howard, Micheal L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyaranayanan, and Robert N. Sidebotham. Scale and performance in a distributed file system. In *ACM Transactions on Computer Systems*, pages 51–81, February 1988.