Warm Backup using Snooping *

Danny Dolev Dalia Malki Yuval Yarom

The Hebrew University of Jerusalem
Jerusalem 91904, Israel
E-mail: {dolev,dalia,yval}@cs.huji.ac.il

Abstract

Local Area Networks use a broadcast media to transfer messages between hosts. This allows for network snooping *by unlisted parties. This paper proposes a novel way for cheaply replicating services in a local area network via snooping. We present a tool for Warm Backup of files that employs network snooping for data dissemination. The tool allows for a selective replication of files in the system. The use of snooping significantly reduces the overhead of file replication. Operations on non-replicated files suffer only a slight overhead.*

1 Introduction

Providing highly available services through replication of servers is a well known method. This paper proposes a novel way for cheaply replicating services in a Local Area Network (LAN) via *snooping*. The paradigm we develop is a general replication method, and we study this approach by an example tool for *warm backup* (WB) replication of files in the Sun's Network File System (NFS) environment. This method can also be extended for off-line backups through a Wide Area Network (WAN).

In an NFS environment, applications access files throughout the network in an automatic, transparent way. We can view the entire network as providing a global file system service that is distributed among different machines. While being convenient in all ways, this distribution leads to a reliability problem: the failure of any of the machines that provide file system services can block an application from running. The need for increased file system availability was, therefore, apparent as soon as NFS systems became available.

In these environments, local-area broadcast networks such as the Ethernet and token-rings are a *de facto* standard. These broadcast media carry the point-to-point NFS messages and enable snooping by unlisted parties.

Since faults are not common, our primary design goal is minimizing the overhead incurred by replication on the primary server, during fail-free operation. The optimal way to provide cheap replication of a file server in a broadcast network would be to place another machine, the backup machine, that puts the network interface in promiscuous mode, and *snoops* to the network and updates its state whenever an update-request to the file server is intercepted. This scheme is depicted in Figure 1(a). This scheme is, indeed, attractive, since it does not incur any communication overhead for replicating the service. The problem with this naive approach is that snooping is not foolproof, and some update-requests may be missed by the backup. In this case, the states of the primary file server and the backup will diverge.

In order to guarantee the consistency of the primary and the backup, we have to introduce an *acknowledgment* message from the backup to the primary. This results in the paradigm depicted in Figure 1(b). This paradigm is similar to other primary–backup mechanisms, such as [4, 1], with several important differences:

1. A novel feature of the WB scheme is the utilization of broadcast hardware (and snooping) for the efficient dissemination of data to the primary and its backup. Note that in the paradigm of Figure 1(b), the (potentially large) update requests are transmitted only once over the network. The only communication overhead is the short acknowledgment message. (Thus, the solid arrows in the figure represent "real" cost, whereas the dashed arrows are achieved by the cost-free communication snooping.)

2. The WB scheme does not require modification of the underlying file system service. The *snooping* program can be implemented as a user-level tool, and be arbitrarily placed on any machine in the network. Our prototype implementation has exactly this property, and it provides the full NFS interface and semantics.

3. Another consequence of Item 2 above is that the WB tool can make a *selective* replication of files, be it the files accessed by a particular applications, or those residing in certain designated directory subtrees. The tool can dynamically accept commands for replication of different directories at different times. We expect this to be a most useful feature of this tool.

*This paper appears in the first intl. workshop on Services in Distributed and Networked Environments (SDNE), June 1994, Prague, Czech Republic, pp. 60-65.

4. One warm backup machine can provide (selective) replication for several file servers.

5. This scheme is scalable: additional backup machines intercept the update data by snooping, and only the acknowledgment mechanism has to be extended to incorporate the extra backups. For example, Figure 2 demonstrates a possible paradigm for one primary and two backups. For simplicity, the rest of this paper deals with the single backup case. All the mechanisms can be easily extended to the multiple backups case.

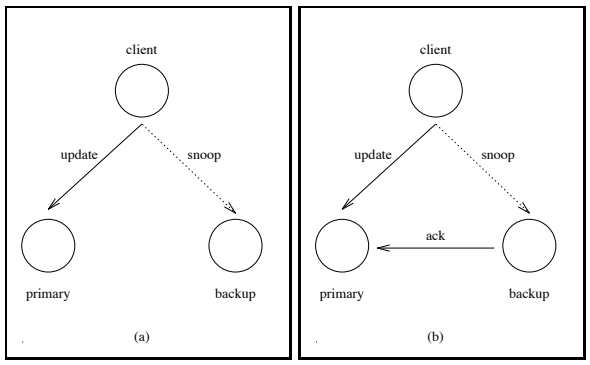

Figure 1: The Primary-back Paradigm using Snooping

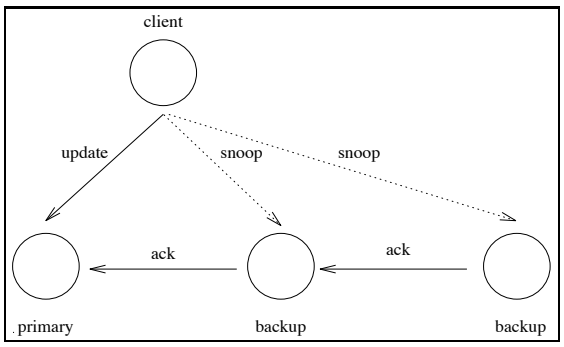

Figure 2: Multiple backups using Snooping

The WB tool has been fully implemented on a network of Sun SparcStations. The tool is implemented as a user level program. It allows the system administrator to specify on the fly the directories to be replicated. The performance section below provides experimental results, showing an overhead of less than 8% (on average) for non-replicated files, and up to 56% for replicated files.

1.1 Related Work

To the best of our knowledge Powell and Presotto were the first to describe a high availability mechanism based on snooping in a LAN [10]. The paper introduces a general scheme to recover from partial failures in a distributed systems, based on checkpoint and replay. Their approach uses a special node, a *recording process*, which collects all the interactions between nodes in the system for later replay. They assume that the recording process is fail proof, and can intercept all communication in the network. Today we know that it is impossible to implement this approach without further interaction with the recording process.

Our case study of snooping involves replicating a file system service. The issue of file system replication is an active area of research. This section does not attempt to cover the vast number of works in this area. It relates to a chosen few, that represent corresponding leading approaches to this problem.

One approach for replicating services is the *State Machine approach* [12]. In this scheme, the client requests are delivered to all the servers in the same order, to act upon. The servers are symmetrical and in case any of them fails, all the other replica have consistent states of the service. The advantage of this approach is that faults are completely masked from the client. However, the requirement of reliable and ordered delivery of client requests to all the replica presents a noticeable burden on the system. The RNFS system [9] and the Deceit system [14], which is based on it, are examples of replicated file systems implemented using this approach. They provide NFS clients with a completely compatible interface, while underneath they replicate each file operation on a set of disks. Both projects report that the reliable communication between the replication *stubs* incurs a considerable slowdown for non-replicated files [9, 13] (*e.g.* in RNFS write operations are 2–3 times slower than regular NFS). Most of the slowdown is incurred by the additional communication layer and the ordering of the updates. As a result, additional replica incur only a small overhead. The State Machine approach requires that changes to the file-service are made at each client machine. As there may be many more clients than servers, this may pose a considerable burden. In contrast, in the WB tool no modifications at the clients are required; Only the file servers need be modified.

A non-symmetrical (*primary-backup*) solution to replication is to designate one server as primary, and let the other replica act as its backups. This scheme has been used extensively in practice, because it is cheaper during normal operation. Its drawback is in the handling of failures and recoveries, which involves re-electing a primary and is more expensive than the State-Machine approach. The work of Budhiraja et al. [3, 5] studies this approach formally. The primary-backup paradigm of the WB tool is similar to [1, 4]. The difference is that the WB tool exploits snooping to optimize the propagation of updates ¿from the client to the backups. In addition, implementing the paradigms in [1, 4] would require changes to the NFS protocol, since the response to the client is delivered by the backup rather than the primary server. One of our goals was to produce a paradigm that is feasible to implement at the user level, without any visible change to the client.

Both HA-NFS [2] and Harp [8] use the primary

backup approach for replication, but utilize special hardware. The HA-NFS scheme relies on dual-port disks, and uses mirrored-updates for achieving atomicity of updates. Harp avoids physical writing to disk while preserving the safety of update semantics by logging updates into a memory unit which is supported by a UPS. In this way, updates are never lost, but the actual write to the disk may lag behind.

There are other replicated file systems like LOCUS [16] and Coda [11], that do not maintain the Unix file system structure or semantics. These systems have their own kernel implementations of both client and server. Their replication scheme employs version time-stamping, and allows different replica to diverge. In contrast, the WB tool is built on top of the Unix file system, and does not modify its structure or semantics. Its implementation is entirely at the user level, and is added at the server machine only. It is provably impossible to provide primary-backup replication with one backup, in the case of network partitions [3]. In case of network partitions, the WB tool allows the backup to diverge from the primary copy.

2 The WB Tool over NFS

This section describes the structure of the WB tool within the NFS environment. For this section, we assume a network consisting of three machines: the client machine, the NFS server machine, and the backup machine. The client machine executes user applications that access files on the NFS server machine. The backup machine snoops update-requests on the network. The WB paradigm in this environment works as follows:

1. The NFS and the WB are notified by the system administrator about the directories she wishes to be backed-up.

2. When the NFS server receives an update-request from an application to a replicated file, it must wait for an acknowledgment-message from its warm-backup. Update-requests are those that modify the contents or the state of the file, *e.g.* *write* requests.

3. The WB server *snoops* for all NFS messages. When it intercepts an NFS update-request to a replicated file, it issues the appropriate operation on the local replica, and sends an acknowledgment message via a reliable channel, *e.g.* a TCP connection, to the NFS server in the site of the accessed file. This message contains a sequence number and an identification of the request (containing the Sun-RPC ID of the request, and the address of the client).

4. The NFS server serves the update-requests it receives in the order of the acknowledgment-messages it receives from the WB server. It returns the results to the application.

5. If either the NFS server or the WB server loses a message from the application, the application will time-out and retransmit the request. (This is the standard fault handling of the NFS/RPC protocol.)

The modification of the NFS server can be done either internally, by modifying the NFS server (Figure 3(a)), or externally, by placing a mediator, on the NFS server's machine, between the application and the NFS server (Figure 3(b)). We chose to implement the external modifications approach in order to avoid modifications to existing software.

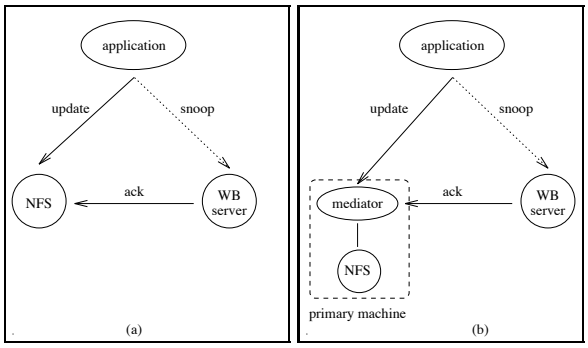

Figure 3: The WB tool in the NFS Environment

Failure Recovery

The modified NFS server and the WB server have to dynamically detect each other's failures and recoveries. If the WB server fails, the NFS server continues normal operation. It should, however, detect the WB recovery, and update it to a consistent state. In case the primary NFS server fails, there are up-to-date copies of all of the backed-up files on the WB machine. There are two possible approaches to handle this case. First, the application can be modified to access the replica of the files upon detecting an NFS server failure. This modification can be done in the libraries level, and is transparent to the application program. The second approach is to implement a *takeover* mechanism in the WB server, which detects the NFS server's failure, and responds to the application request in a transparent way. Both these approaches are standard in such a system, and their details are beyond the scope of this compact presentation. The topic of re-integrating an NFS server upon recovery is detailed elsewhere [7].

3 Implementation Details

The WB is feasible to implement, although we have learned that its implementation involves a fair bit of hacking system code. The WB tool has been implemented in two parts: the backup server and the NFS mediator.

The WB server

The backup process utilizes the Sun *Network Interface Tap* (nit) device for putting the network interface at the backup machine in promiscuous mode, and for intercepting all the network traffic. The nit device provides access to the raw network packets. These have to

	1Mbytes	2Mbytes	4Mbytes	slowdown
Standard NFS	12.57 (12.46-12.70)	26.87 (26.27-27.45)	54.06 (53.22-54.83)	
Modified NFS	13.37 (12.96-14.28)	28.00 (27.04-29.58)	57.88 (55.16-66.12)	6%
Replicated	19.73 (18.40-21.52)	39.25 (34.29-40.79)	80.10 (74.45-82.84)	49%

Table 1: NFS, modified-NFS and WB-replication response times (in seconds).

be reassembled by the WB server into the RPC/UDP datagrams of the NFS protocol. The server then filters the update-requests for the replicated files.

In order to do this filtering, the server has to be able to identify the files on the primary server machine. The only identification sent over the network by the NFS protocol is an opaque *file handle*, obtained by the client when the file is *opened* (see [15]). The WB server obtains a copy of the relevant file handles at startup, and compares these handles against the file handles in the intercepted datagrams.[1]

The mediator

The mediator process resides on the primary NFS server machine, and simply "hides" the NFS server. It intercepts all incoming requests, and performs fast filtering of the relevant datagrams, in order to detect update-requests to replicated files. In case of an update-request for a replicated file, it buffers the request until it is acknowledged by the backup. In all other cases, the request is immediately forwarded unaltered to the NFS server.

In all cases, when a reply from the local NFS server is ready, it is immediately forwarded back to the client.

4 Performance

The issue of performance has always been an important consideration when building fault tolerant programs. This has an additional facet in the WB tool, since the tool provides a selective replication service. Therefore, the slowdown of the file-server for *non-replicated* files should also be taken into account. This cost is incurred by placing the *mediator* at the file server machine, and having to filter all of the update requests that modify replicated-files (see Figure 3(b)).

The WB architecture is designed for incurring a minimal overhead on the message traffic in the system. The dashed-arrows in Figure 3 are almost cost free, and are done by network snooping. The reason for saying that this is *almost* cost free is that in case the WB server loses a message, it will be retransmitted. Still, for write-operations on files, the written data is essentially sent only once over the network. Hence, the main source of delay is the need to wait for an acknowledgment message from the WB server.

We have measured the response time of the file service for both replicated and non-replicated files. The overhead for the non-replicated files includes only the mediator overhead, while the overhead for the replicated files also includes the acknowledgment message latency, and the cost of the potential retransmission due to WB omissions. All the test were taken when the load on the network and the machines was low. The experimental results are summarized in Table 1.

The first row shows the measured response time of vanilla NFS for various file writing scenarios. Row 2 lists the modified NFS service response times for non-replicated files (both the average and and the range of observed times are shown). The last column compares the average response time with standard NFS. We performed the same tests while loading the file server concurrently from another machine with various I/O activities. Similarly, we compared the response time for more natural activities, such as compiling the WB tool and LaTeX'ing this paper.[2] In all of these tests, the overhead incurred by the mediator process and the filtering of packets was less than 8%.

Row 3 lists the response times and slowdown for replicated files with a single backup. In these tests, the percentage of packets that were dropped by the WB server was 0.5%. These tests show that the slowdown of a single backup replication is between 45–56%.

The slowdown of replication is less than twice the standard NFS service. This demonstrates clearly the advantage of the primary-backup paradigm in the NFS environment, and of utilizing the broadcast capability. Note that the WB paradigm preserves the NFS semantics in writing updates directly to the disk before responding to the client.

The WB on the backup machine snoops for messages addressed at multiple NFS servers. It therefore must put the network-interface of the backup machine in promiscuous mode, and filter the relevant messages from among the multitude of messages transferred in the system. This means that the WB server message loss-rate is influenced by the local load on the machine. Consequently, for best results, the WB server should run on a dedicated machine—the *backup machine*. We argue that this price is acceptable; the alternatives are either supporting a fully replicated file system, which burdens all the file servers in the system, or using a special hardware, as in [2].

A Formal Analysis

We proceed with a more formal analysis of complexity of the protocol. A precise characterization of primary-backup protocols is provided in [5]. They present several formal performance parameters concerned with building a fault tolerant service using the primary backup approach: (1) The number of replica,

[1] This scheme relies on the fact that identical copies of the file handle are handed to all clients accessing the same file. In the Sun NFS implementation this is guaranteed to hold.

[2] In the final version of this paper, we intend to provide performance measurements for the Andrew benchmark. However, the main slowdown incurred by replication is due to write operations, and we focused our tests on them.

(2) the time it takes the system to recover from a failure and resume normal service, and (3) the response time of the service during a no-failure period.

The first and second issues are concerned with the failure-detection and the recovery take-over mechanism. Note that these issues are separate from the provision of a *single server equivalent* service during normal operation. We chose to leave these issues out of this paper, and solutions for them can be found in the literature, *e.g.* in [1, 2, 5, 14]. Also, since concurrent updates to files are rare in the Unix environment, we are not concerned with the possibility of network partitions. Our focus is on optimizing operation during normal behavior, *i.e.* the third issue.

In the terminology of [5], the service *blocking time* is the longest period between the time a client request is received at the primary, and the time a response is sent. Let δ be an upper bound on the communication latency between two machines. The *blocking time* of the WB tool with a single backup is δ, which matches the lower bounds achieved in [5] (making similar assumptions about failure types, and ignoring processing time).[3]

Their analysis can be refined by noting that the latency of short messages, *e.g.* acknowledgments, is much smaller than the latency of large data messages. Let ϵ be the latency of the acknowledgment messages. Usually $\epsilon \ll \delta$. For example, in our environment, which consists of SparcStations over an Ethernet, the latency of a 4 bytes datagram using UDP/IP is about 0.4 milliseconds, so $\epsilon = 0.4ms$. The latency of the 8K-Bytes NFS write requests is about 3 milliseconds, thus $\delta = 3ms$.

Using this refinement, the *blocking time* of our protocol in the case of a single backup is only ϵ; the time it takes to send the *acknowledgment* message from the backup to the primary. This demonstrates the advantage of utilizing broadcast (and snooping) over point-to-point protocols, where the lower bound ([5]) is δ.

5 Other Uses

Our approach offers the potential for a geographically distributed backup of file systems without burdening the local file server with remote communication. This can be done by having a local node snoop on the local network, and asynchronously off-load the backup to the remote site. The advantage of asynchronous update is that the remote backup is not a part of the update loop, and the response to local updates is not delayed by backing up. In addition, in our approach, the primary server is not disrupted by the back up process.

The scheme of [10] for fault-tolerance in a distributed environment requires a reliable *Publishing* mechanism, which is not achievable without software protocols. Snooping can provide a tool for achieving the reliability of this scheme. To achieve this, we replace their *recording process*, which relies on reliable publishing, with our snooping paradigm, which is based on software protocols for reliability.

[3] For multiple backups, [5] makes the assumption that a designated backup can intercept messages from all other backups in a single δ delay time. This assumption does not hold in the environments we have in mind. These results are, therefore, incomparable.

6 Limitations

We are aware of several limitations of the approach presented here:

1. The WB tool relies on network snooping, which works only on broadcast media. In networks, such as ATM or FDDI, that do not operate with broadcast media, snooping will not work. A possible workaround is to modify the client to use the multicast feature of these networks, and to send updates to a multicast group, in which both the primary server and the replicas are members.

2. The current design does not have congestion control, and in highly loaded networks will incur additional load by inducing retransmissions.

3. The view of the file at the WB site will diverge from the primary replica in the following cases:

   - Non-deterministic updates to the state of the file, such as the recording of the *last modify time*, will not be consistent at the primary and backup sites.
   - Rejected updates (*e.g.* when the file system is full) are not detected by the WB server in the current implementation. In order to handle this case, the mediator should inform the WB server about the rejection before returning the rejection-response to the client. We plan to incorporate this error handling in the next version of the WB tool.
   - Client crashes, when the last update is received only by the WB server but not by the NFS server. The WB server will update its copy of the file assuming that the NFS server received the update, or will receive a retransmission. But, if the client crashes, there is no guarantee that the NFS server will receive the update.

   Every replication system is provably either prone to blocking, or to inconsistency *e.g.* in case of network partition. Practical tools should provide the user with means to overcome the inconsistency, or at least to detect it. As these problems are common to all replication schemes, we have chosen not to discuss them in the context of this paper.

7 Conclusions

Computer networks are using broadcast media. This offers an efficient way to disseminate messages to multiple destinations. Future networks such as the high-speed FDDI ring and wireless networks also possess the broadcast capability. Understanding the potential in broadcast communication is, therefore, important (see also [6]).

The ability to snoop on a broadcast network has led us to devise a completely new scheme for a cheap replication of files on a LAN. The Warm Backup (WB) tool *snoops* on the network, and updates the state of the backup by applying the updates it intercepts. This tool is attractive because it can be plugged into any system without changing the structure of the file service. Our prototype implementation within the NFS environment operates entirely at the user level, is easy to use, and can be programmed to selectively backup certain directories. This flexibility makes it an attractive tool.

The experimental results presented here show that the regular file system service during no-failure periods suffers on average less than 8% slowdown. Similarly, the replicated service costs around 50% slowdown on average, over the regular NFS service. These performance results compare favorably with existing replicated file systems.

Acknowledgments

We would like to thank Gil Shwed for initiating this work, and for many fruitful discussions. Yuval Harari implemented an initial package that tested several implementation ideas, and helped formulate the current paradigm. We are thankful to the Computer Science dept. of the Cornell University and to the ISIS lab, for allowing us to use their machines for implementing and experimenting with the WB tool, and for enduring the numerous 'reboot's we performed. We benefitted from discussions with Bradford Glade and Keith Marzullo.

References

[1] P. Alsberg and J. Day. A Principle for Resilient Sharing of Distributed Resources. In *Proceedings of the Second International Conference on Software Engineering*, pages 627–644, Oct. 1976.

[2] A. Bhide and S. P. Morgan. A Highly Available Network File Server. RC 16161, IBM Research, May 1990.

[3] N. Budhiraja. *The Primary-Backup Approach: Lower and Upper Bounds.* PhD thesis, dept. of Computer Science, Cornell University, June 1993. (TR 93-1353).

[4] N. Budhiraja, K. Marzullo, F. B. Schneider, and S. Toueg. Optimal Primary-Backup Protocols. In *6th Intl. Workshop on Distributed Algorithms proceedings (WDAG-6), (LCNS, 647)*, pages 362–378, November 1992.

[5] N. Budhiraja, K. Marzullo, F. B. Schneider, and S. Toueg. The Primary-Backup Approach. In S. Mullender, editor, *Distributed Systems.* ACM Press, 1993.

[6] D. Dolev and D. Malki. On Distributed Algorithms in a Broadcast Domain. In *Intl. Conference on Automata, Languages and Programming*, pages 371–387, July 1993.

[7] Y. Harari. Warm Backup Tool for Unix Network File System. internal manuscript, 1992.

[8] B. Liskov, S. Ghemawat, R. Gruber, P. Johnson, L. Shrira, and M. Williams. Replication in the Harp File System. In *Proceedings of the 13th Symposium on Operating Systems Principles*, pages 226–238, Oct. 1991.

[9] K. Marzullo and F. Schmuck. Supplying High Availability with a Standard Network File System. In *4th Intl. Conf. Distributed Computing Systems*, pages 447–453. IEEE, June 1988.

[10] M. Powell and D. Presotto. Publishing: a Reliable Broadcast Communication Mechanism. In *Symposium on Operating Systems Principles*, number 9, pages 100–109, October 1983.

[11] M. Satyanarayanan, J. Kistler, P. Kumar, M. Okasaki, E. Siegel, and D. Steere. Coda: A Highly Available File System for a Distributed Workstation Environment. *IEEE trans. on Computers*, 39(4):447–459, April 1990.

[12] F. Schneider. Implementing Fault Tolerant Services Using the State Machine Approach: A Tutorial. *Computing Surveys*, 22(4):299–319, December 1990.

[13] A. Siegel. *Perfomance in Flexible Distributed File Systems.* PhD thesis, dept. of Computer Science, Cornell University, Feb 1992. (TR 92-1266).

[14] A. Siegel, K. Birman, and K. Marzullo. Deceit: A Flexible Distributed File System. TR 89-1042, dept. of computer science, Cornell University, Ithaca, NY, Nov 89.

[15] Sun Micronsystems Inc. NFS: Network File System Protocol Specification. RFC 1094, SRI Network Information Center, March 1989.

[16] B. Walker, G. Popek, R. English, C. Kline, and G. Thiel. The LOCUS Distributed Operating System. In *9th Symp. on Operating Systems Principles*, pages 49–70, 1983.