

Transparent TCP/IP based Replication

Christof Fetzer
Department of Computer Science & Engineering
University of California, San Diego
La Jolla, CA 92093-0114, USA.
<http://www.cs.ucsd.edu/~cfetzer>

Shivakant Mishra
Department of Computer Science
University of Wyoming, P.O. Box 3682
Laramie, WY 82071-3682, USA.
<http://www.cs.uwyo.edu/~mishra>

1 Introduction

Replication is a fundamental technique to increase the availability, dependability, and performance of services provided in a distributed system. The key idea is that the service is implemented by a set of servers running on different machines with independent failure modes. These servers replicate the service state and continue to provide the service despite communication or processor failures. While replication is certainly a useful technique, there are two crucial problems associated with it in its deployment in practice: (1) maintaining a consistent replicated service state is complicated, and (2) replication is expensive in terms of resource consumption and performance.

In our current work we address these two problems associated with replication. In particular, we propose a mechanism to replicate a service that is efficient, and transparent to the servers and service clients. By *efficient replication* we mean that the replication has very little performance overhead at the primary server during normal operation. By *transparent replication* we mean that (1) the clients are not aware of the existence of replicated servers, and (2) the servers are not aware of the replication. We assume that clients communicate with a service via TCP/IP. We implement the primary-backup replication scheme by modifying the server-side TCP/IP code and introducing service specific wrappers.

2 Design

A TCP connection between a client and a server is typically routed through one or more routers. The key idea behind our approach is that we use a router or, alternatively, a link between two routers to *tap* the byte streams flowing between a client and the primary server (see Figure 1). The tapped byte streams are forwarded to a backup server. Of course, one could

tap a TCP connection multiple times to provide for multiple backup servers.

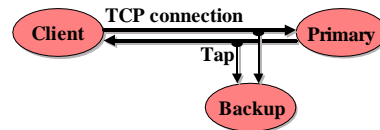


Figure 1: The backup taps all TCP connections between the primary server and its clients to keep its state consistent with the primary.

The actual server that processes the client's requests is the primary server. Since every byte exchanged between the client and the primary server is routed through the intermediate router, the backup server learns the complete communication state of the primary. In particular, we can guarantee that a non-crashed backup server receives all data that the primary server receives and it also receives all data that a client receives.

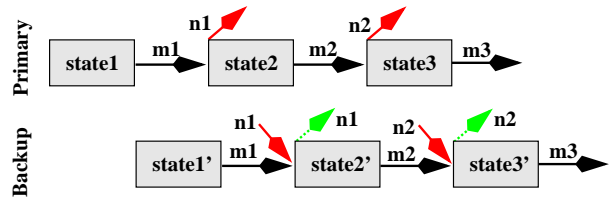


Figure 2: A backup server uses a leader/follower consistency protocol to keep its state consistent with the primary.

In case a server is completely deterministic, a backup server can keep its state consistent with the primary server by executing the same sequence of requests as the primary does. Most servers might however not be completely deterministic, e.g. the backup server might use different time-stamps or ids than the primary. In our approach, we want to support the

replication of non-deterministic servers by applying a *leader/follower* consistency protocol (see Figure 2). The primary (leader) executes a request (say, m_1), and then sends its reply n_1 to the client. Consider that the primary is in state 2 after sending n_1 . The backup server receives this reply n_1 and uses it together with request m_1 to transit to a state 2' that is consistent with state 2. By *consistent* we mean that 1) the backup would send the same reply n_1 during the execution of m_1 (note, all backup replies are suppressed), and 2) the backup's state is consistent with the service specification. Note that the backup can detect when its state becomes inconsistent or its (suppressed) replies differ from that of the primary. This allows the backup to detect failures of the primary other than crash or performance failures.

Because the backup server is in a state consistent with the state of the primary, the backup server can take over the functions of the primary in case the primary crashes (or, is suspected to have crashed). The backup can request to take over all TCP connections of the primary. This guarantees consistent behavior even when the primary is incorrectly suspected to have crashed: the primary cannot reply to clients anymore since all its TCP connections have been taken over by the backup.

The replication of a server should be transparent to the clients and the servers. Achieving server-side transparency is not easy. We address this issue by designing service specific wrappers (see Figure 3). The wrapper of the backup takes care of keeping the backup server state consistent with the primary. The primary runs also in a wrapper which detects failures of the backup and replaces the backup if needed.

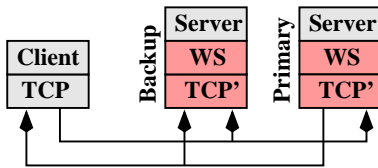


Figure 3: A server runs in a wrapper and on top of a modified TCP/IP protocol stack. Clients are not modified.

3 Tapping

We are currently investigating different alternatives of streaming the traffic through the backup system to find the most efficient way of keeping the backup server up-to-date. These alternatives involve using a

router to tap the TCP connections to the primary server (see Figure 4), a proxy-ARP server that provides a tap (see Figure 5), or using an Ethernet sniffer (see Figure 6) that captures the traffic between a server and its clients.

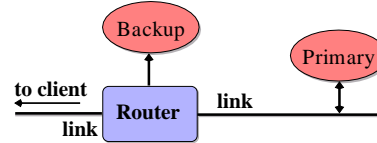


Figure 4: *Alternative 1: The router forwards all messages sent between the primary and a client to the backup.*

We also address the issue of how one can switch automatically between the primary and the backup whenever there is a failure. In particular, we have to make sure that at most one of the two servers believes that it is the leader, i.e. only one server should be able to reply to client requests. We address this issue using a leader election protocol.

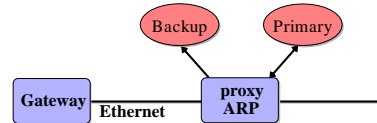


Figure 5: *Alternative 2: The traffic between a client and the primary is forwarded to the backup by a proxy ARP server.*

We are currently modifying the TCP/IP code of the Linux operating system to facilitate tapping and a take-over of TCP connections. Recall that only servers have to run a modified TCP/IP stack while clients can use a standard TCP/IP stack. This modification will 1) allow one to transfer the state of a TCP connection to a new backup site, 2) enable a backup to keep a connection alive when the primary is slow, disconnected, or crashed, and 3) permit a backup to take over a connection from the current primary if the primary is suspected to have crashed.

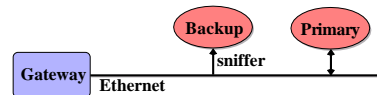


Figure 6: *Alternative 3: The backup sniffs the Ethernet. One has to make sure that in case the backup misses a packet, it can ask the primary (or, some other node) to retransmit the packet.*