FreEBS - A Virtual Block Storage System

Igor Canadi, Rebecca Lam, James Paton
University of Wisconsin-Madison
{canadi,rjlam,jpaton}@cs.wisc.edu

1. INTRODUCTION

FreEBS is a distributed virtual block device that utilizes replication and log-based storage to provide reliability and durability. The major goal of FreEBS is to create a free and open version of Amazon Elastic Block Store (EBS) [1], a virtual mountable block device used by EC2 instances whose implementation details have not been publicized.

There are two main features that are publicly known about Amazon EBS — replication and snapshots. EBS mirrors volume data across different servers within the same Availability Zone. This provides some increased reliability over a single disk, but EBS further adds durability using snapshots. Snapshots are customizable incremental backups stored on Amazon S3. Each snapshot only saves the blocks that have been modified since the last snapshot. The frequency at which they are performed determines the durability of the volume in the case that the EBS volume fails. [1]

Because of the lack of concrete, detailed information about Amazon EBS, we draw many of our ideas about EBS from Distributed Replicated Block Device® (DRBD) [2], an open source virtual RAID-1 block device. We discuss DRBD® in more detail in Section 2.

Given what we know of EBS and DRBD, FreEBS seeks to provide the below:

Availability The system should continue to operate with a certain number of replica failures.

Durability In the case of a failure, the most recent data should still exist on the volume.

Snapshots Our backing file should support snapshots.

Reliability The system should operate as expected and behave deterministically.

We address these goals by using replica servers that coordinate using a userspace process on each machine. These replicas propagate writes and perform synchronization. They also interact with a log-structured virtual disk that provides checkpointing and the framework for snapshotting. *Talk briefly about results here?*

The rest of the paper is organized as follows. In Section 2 we discuss related work, and in Section 3 we explain the overall design of the system. We go into more detail about our system in Section 4. Finally, we evaluate our design and draw appropriate conclusions in Section 5 and Section 6, respectively.

2. RELATED WORK

2.1 DRBD

DRBD® is a virtual RAID-1 block storage device consisting of multiple shared-nothing nodes. The DRBD® interface is implemented as a kernel module that can be mounted like a normal block device. In this section we focus on replication and synchronization.

Replication.

DRBD® uses a RAID-1 method of replication, meaning there are two nodes in a basic DRBD® cluster — one *primary* and one *secondary* — that each contain typical Linux kernel components [2, 3]. The primary node services all reads and writes, and the secondary node fully mirrors the primary node by propagation of writes across the network. If the primary node fails, then service migrates to the secondary node. There are two main modes for replication — fully synchronous and asynchronous. The former means that the primary node reports a completed write only after the write has been committed to both nodes in the cluster. The latter means that each node reports a successful write as soon as the data is written to its local disk.

FreEBS uses a similar technique to the asynchronous method of replication; we utilize a primary replica that services reads and propagates write requests. However, FreEBS waits for a majority of replicas to respond with a successful write completion. Our system also offers much more flexibility in terms of the number of configurable replicas. For instance, DRBD® must use stacked DRBD volumes in order to create more than two replicas, which can get cumbersome as we add more replicas.

Synchronization.

DRBD® offers variable-rate and fixed-rate synchronization [3]. For the first method, DRBD® selects a synchronization rate based on the network bandwidth. For the fixed-rate case, synchronization is performed periodically at some constant time interval. Synchronization can be made more efficient by using checksums to identify blocks that have changed since the last synchronization. This eliminates the need to synchronize blocks that were overwritten with identical contents. Changes are tracked using the activity log (AL) which records *hot extents*, the blocks that have been modified between synchronization points. The activity log uses a quick-sync bitmap to keep track of modified blocks.

FreEBS uses a fixed-rate synchronization method that allows nodes to request all writes since an arbitrary version. *Describe how LSVD fetches writes here???*

should we include the DRBD figure here?

2.2 Distributed File Systems

AFS NFS Cassandra Openstack Eucalyptus

3. DESIGN

FreEBS is a virtual block device backed by replication and log-structured virtual disks for high reliability and availability. For all intents and purposes, the FreEBS storage system appears to the kernel as a normal block device and thus reads and writes are issued to FreEBS without knowledge of its internals. The architecture of FreEBS is depicted in Figure 1. As we observe, the most important component in the system is the *driver*, which is a mountable kernel module that interfaces between the OS and the rest of the system. It accepts normal block device commands and forwards the requests appropriately to the *replicas*. Our current configuration uses three replicas, but our design is flexible enough that we can arbitrarily add more if necessary. We discuss replication in more detail in Section 4.2.

Replicas are essentially server machines that communicate with each other and the driver across the network using a custom TCP protocol. Each replica runs a userspace process called `sdaemon` that is in charge of propagating write requests and synchronizing replicas. It is also responsible for interfacing with its private copy of the underlying storage that actually holds the device data, *Log Structured Virtual Disk* (LSVD). LSVD is backed by a file on the server file system that uses a dynamically growing file format. It provides checkpointing and versioning for high data integrity. A more in-depth discussion about LSVD is in Section 4.3.

4. IMPLEMENTATION

4.1 Reads and Writes

There are two operations that are issued by the driver — read and write. The driver keeps track of the most

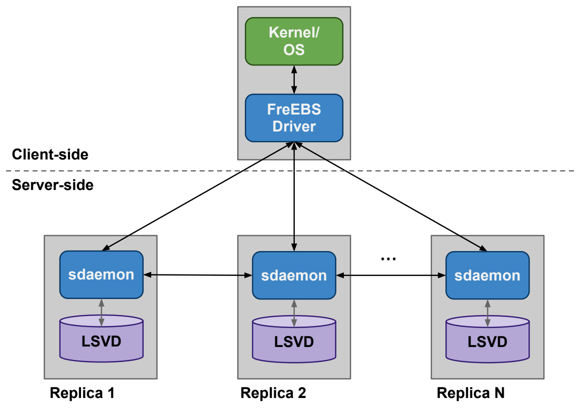


Figure 1: The system architecture for FreEBS. A driver on the client machine communicates with processes on the replicas on server machines to read and write data.

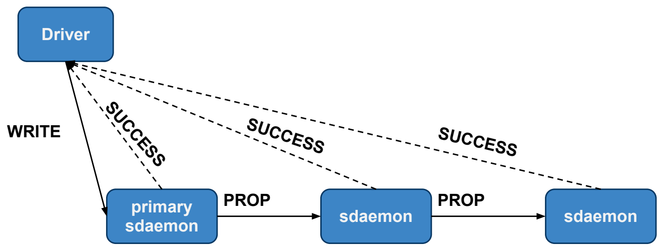
recent version for each replica in a list. This structure is updated on a write and used to decide which replica to request a read from.

Figure 2a shows the message flow on a write request. Notice that we utilize chained replication, where the closer the replica is to the head of the chain the more more up-to-date it is. The choice for this chained scheme is to reduce the network load to the driver. Thus, when the driver sends a write request, `sdaemon` issues the write to its local LSVD copy and then propagates the request to the next replica in the chain. After writing to its own LSVD, each replica sends back a response to the driver with a status and its current version number. The driver then updates its replica list with the version in the sent response. If a majority of replicas responds to the driver with a SUCCESS message, then the driver reports a success back to the kernel. Otherwise, it will report a failure.

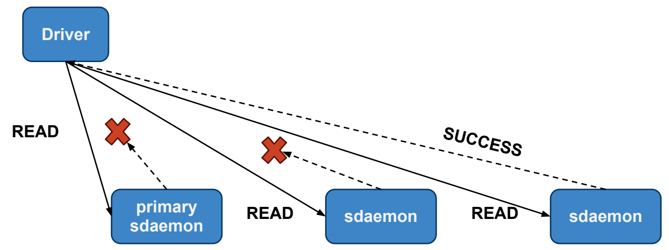
In contrast, a read only needs one replica to respond for a success. In Figure 2b we observe that when the driver issues a read request it issues the request to the replica with the most recent version. Typically this is the primary replica. If the replica responds with a SUCCESS, then the driver serves the request back to the kernel. Otherwise, it will issue the request to the next replica in the chain with the most recent version. I no replicas respond, then the driver reports a failure.

4.2 Replication

Replication is managed by `sdaemon` using write propagation and synchronization. Propagation was discussed in Section 4.1. Synchronization is performed whenever a replica joins the chain and also periodically to keep replicas near the tail from becoming increasingly out of date. Figure 3 shows how replication is performed. As we see, replica C sends a SYNC request with the



(a) The message flow for a write operation. A quorum of SUCCESS responses must be received for a successful write.



(b) The message flow for a read operation. A read request is issued to the replica with the most recent version until a SUCCESS is received or if no replicas with the most recent version responds with SUCCESS.

Figure 2: Message flows for read and write operations

version number of its local LSVD copy to the previous replica in the chain, replica B. Replica B then responds with all the writes from C.version to B.version. Replica C then applies all the writes to its LSVD. After the synchronization operation, Replica C's LSVD version is equal to B's version. Similarly, replica B initiates the synchronization procedure with replica A.

To handle failure, a replica controller is used to inform the driver and neighboring replicas that a failure has occured. This controller keeps track of the liveliness of each replica by recording the elapsed time since the last hearbeat message, which is sent every T seconds by each replica. After $2T$ seconds the controller flags the replica as *inactive* and after $4T$ seconds the replica is marked as failed. Upon failure, the controller informs the kernel and the neighboring replicas using update messages. Upon receipt of the update messages, the two neighboring replicas connect to each other. The latter replica sends a SYNC request, and then normal operation resumes. Meanwhile, the controller spins off a new `sdaemon` process that is appended to the end of the chain. Note that currently the replica controller is not implemented.

4.3 Log Structured Virtual Disk

Discussion on how versioning works, how we can retrieve arbitrary writes (change logs?) how we can easily extend this framework for snapshotting.

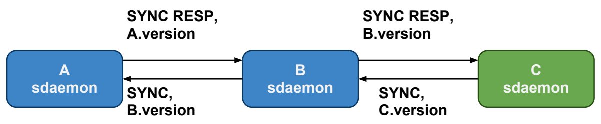


Figure 3: The message flow for a synchronization operation. Replica C sends a SYNC request to its predecessor, which responds with all the writes since C's version.

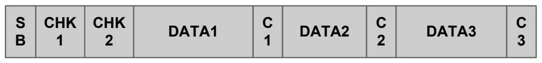


Figure 4: The LSVD file structure.

5. EVALUATION

5.1 File System Benchmarks

5.2 Checkpointing evaluation

6. CONCLUSION

7. REFERENCES

[1] Amazon elastic block store (ebs). `http://aws.amazon.com/ebs/`. Accessed: 2013-05-10.

[2] Drbd: Software develpment for high availability clusters. `http://www.drbd.org/`. Accessed: 2013-05-10.

[3] Brian Hellman, Florian Haas, Philipp Reisner, and Lars Ellenberg. The drbd user's guide. `http://www.drbd.org/users-guide-8.4/`. Accessed: 2013-05-10.

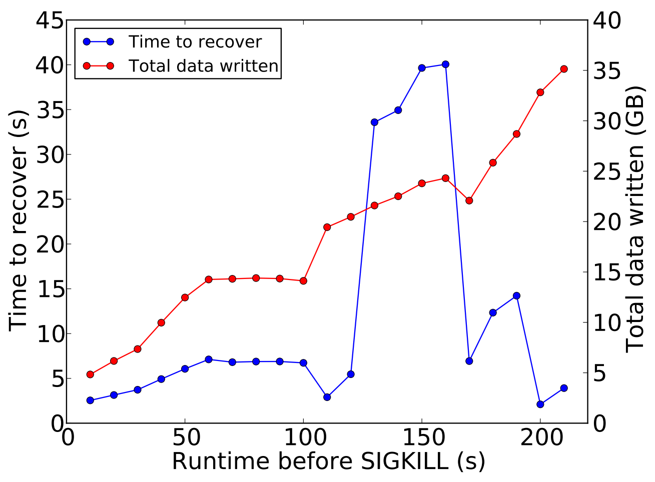


Figure 5: TODO