Logstor: a log-structured virtual storage

Wuyang Chung

wy-chung@outlook.com

***Abstract*—Logstor uses the idea from log-structured file system to build a virtual disk on top of a physical disk. It is a virtual disk so its size can be larger then the underlying physical disk. It's log-structured so it transforms write operations from upper layer file system to log append operations. Since the random write operations are transformed to log append, the random write operations are transformed to sequential write operations.**

Logstor is implemented by using FreeBSD's GEOM framework. This is the second version of this program. The differences with the previous version are:

- **Previous version runs in user-level and this version runs in kernel.**
- **Remove garbage collection and replace it with Stale Sector Recycle (SSR).**
- **Add new logstor commands snapshot and rollback.**

Snapshot command is used to take a snapshot of the virtual disk and the rollback command can revert the virtual disk back to its previous snapshot.

Keywords—GEOM, log-structured, virtual disk

I. Introduction

FreeBSD's GEOM framework makes the configuration of storage devices flexible and easy. What it does is to change the I/O requests from upper layer and then send the changed requests down to the lower layer. The change can be the data contents or the address where the data is to be written to. Logstor is implemented as a GEOM module. It only changes the address of the request. It adopts the idea from log-structured file system and changes the write requests from upper layer to log append requests to lower layer, i.e. to sequential write.

Historically file systems and disks have employed different terminology for storage units – namely blocks and sectors. To ensure clarity, this paper defines a block as the virtual unit of the upper-layer and a sector as the physical unit of the underlying disk. Hereafter, these terms are used to distinguish between the virtual and physical layers.

Logstor needs to store some metadata in order to create a virtual view of the disk to upper layer. There are three metadata in logstor. They are *forward map*, *inverse map* and *superblock*. Since data is written to different locations on the disk, a forward map from block address to sector address is required. This map lets logstor know where to read a block from the underlying disk. The forward map is very big and it cannot all be loaded in DRAM so logstor uses buffer cache to cache the recently used maps in DRAM. The inverse map is used to store the mapping from sector address to block address. It is needed for *stale sector recycle* (SSR). Logstor uses page table data structure to store the forward map on disk and the sector address of the root page is stored in superblock. There are also some other states and statistics data stored in superblock.

The next section will give a detailed information about the implementation of logstor.

II. Implementation

A. The inverse map

The inverse map stores the mapping from sector addresses to block addresses. It is stored at the end of each segment. Figure 1 illustrates the disk layout of logstor, which is divided into fixed-size segments. Segment 0 is reserved for the superblock, while the remaining segments are used for data storage, with the exception of their last sectors. The last sector of each segment is reserved for inverse map. Every data sector within a segment has a corresponding entry in this inverse map sector that stores its block address (i.e., the inverse mapping).

B. The forward map

The forward map stores the mapping from block addresses to sector addresses. Logstor utilizes a page table data structure to store this data on disk. To support snapshot and rollback commands, logstor requires at least four forward maps. As shown in figure 2, the sector addresses of the root pages are stored in the superblock. Each entry in a page directory (PD) page contains the sector address of the next level page.

Since page table pages are also appended to the log during forward map writes, each must be assigned a unique block address. The block address for a page table page consists of five fields:

- Field 1: A constant prefix (0xFF) identifying the block as a page table page.
- Field 2: The map number (two bits)
- Field 3: The depth of the page within the hierarchy.
- Field 4 & 5: The entry index numbers for the Page Directories (PD).

Using this encoding scheme, logstor can determine the exact disk location of any page table page based on its block address.

Because the forward map data is too large to be stored entirely in memory, logstor implements a buffer cache to store recently accessed page table pages.

C. The buffer cache

Figure 3 shows the structure of logstor buffer cache. Buffer cache is used to cache the recently used page table pages. Each buffer is linked to two queues. The hash queue and the depth queue. The hash queue enables rapid lookups to determine if a page is cached, while the depth queues manage the cache flushing process. When flushing the cache, we must flush from the highest depth queue to the lowest depth queue. Each buffer has a pointer to its parent buffer. When a buffer is written back to disk, its new sector address must be updated to its parent buffer. If it is the root buffer, its new sector address must be updated to the superblock. There are two depth-2 queues: the clean queue and the dirty queue. When a cache miss occurs,

logstor selects a victim buffer from the depth-2 clean queue and replaces it with the requested page. When the number of buffers in the clean queue drops below a predefined threshold, logstor initiates a cache flush to write all the modified buffers to disk.

D. The Stale Sector Recycle (SSR)

When writing data to disk, logstor uses stale sector recycling to find a stale sector on disk and replace it with new data. The following algorithm explains how to identify stale sectors. To determine if a sector is stale, the system must verify its status against a maximum of three forward maps.

```
inverse(s) → b
for (i = 0; i < n; ++i) {
   forward(i, b) → s'
   if (s == s')
      return not stale
}
return stale
```

E. Read and write operations

Figure 4 shows that during normal operation (when not performing snapshot) it uses two forward maps: the current and snapshot maps. When performing a read operation, the system first checks for a mapping in current map; if none is found, it then checks snapshot map to retrieve the sector address of the block. It then reads the data from that sector address. If no valid mapping exists, it returns a block of zeros to upper layer. When performing a write operation, the system appends the data to the end of the log. The write operation uses SSR to obtain the address of a stale sector, updates the current map with this sector address, and then writes the block to the specified sector.

F. The snapshot command

As illustrated in figure 5, logstor uses four forward maps during snapshot. When the snapshot command is received, the system will first create two empty maps: current_new and snapshot_new. It then merges the current and snapshot maps into snapshot_new. If a read operation is received during this process, the system searches for a sector address in the following order: current_new, then current, and finally snapshot. The search stops as soon as a valid mapping is found. If a write operation is received, the system uses SSR to obtain a stale sector, updates its address to the current_new map, and writes the block to that sector. Once the snapshot operation is complete, the system deletes the old current and snapshot maps, then renames current_new to current and snapshot_new to snapshot. When deleting a forward map, the system simply erases the sector address of its root page in the superblock. The sectors occupied by the deleted map will be recycled automatically by SSR.

G. The rollback command

Performing a rollback command is straightforward; the system simply erases the sector address of the current map's root page in the superblock and lets SSR recycle the map data. Since the current map is now empty, the virtual disk reflects only the snapshot view.

III. Conclusion

This paper has detailed the design and implementation of logstor, a log-structured storage solution built on the FreeBSD GEOM framework. By abstracting physical storage into a flexible virtual disk, logstor provides a scalable environment that supports advanced data management features such as thin provisioning, snapshots, and rapid system rollbacks.

References

[1] *Marshall Kirk McKusick, George V. Neville-Neil, Robert N.M. Watson, "The Design and Implementation of the FreeBSD Operating System," Addison-Wesley, 2004.*

[2] *Mendel Rosenblum, John K. OusterhoutAuthors, "The design and implementation of a log-structured file system," ACM Transactions on Computer Systems, Volume 10, No. 1, February 1992, Pages 26-52.*

[3] *Poul-Henning Kamp, "GEOM Tutorial."*

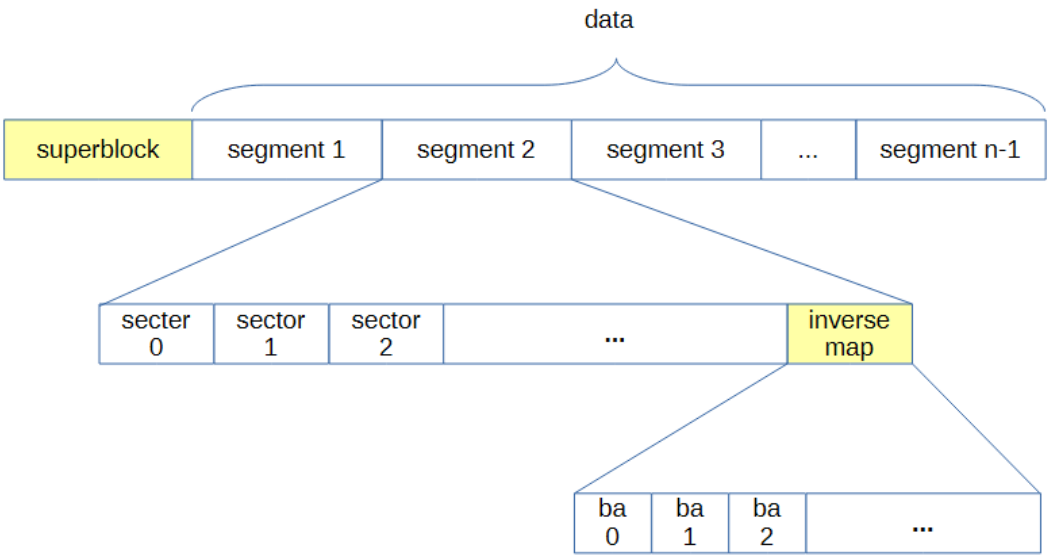

Figure 1. The disk layout of logstor

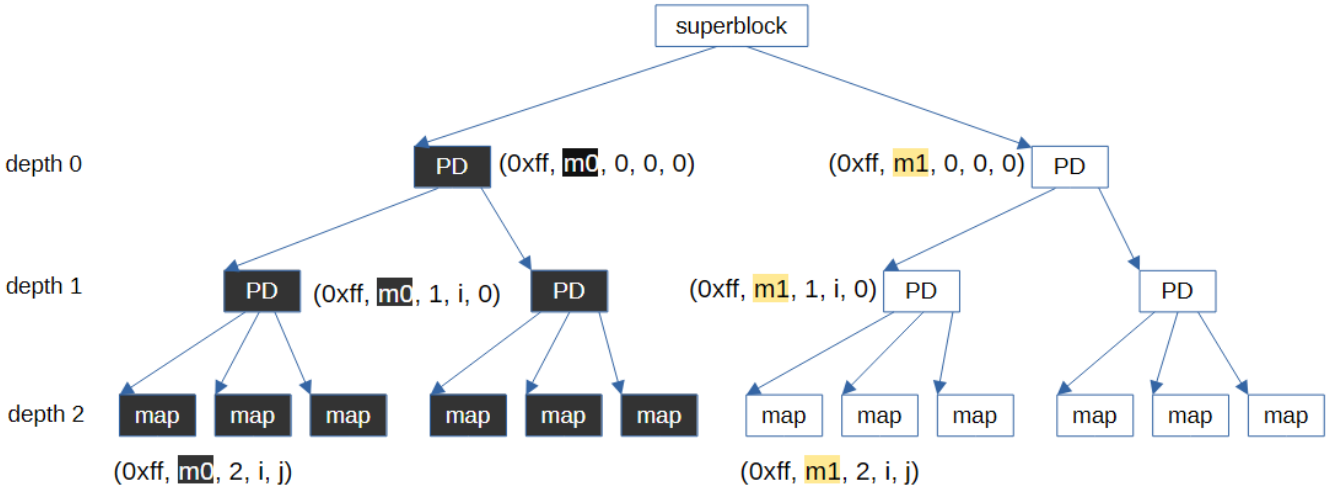

Figure 2. Forward map is stored in disk as a page table

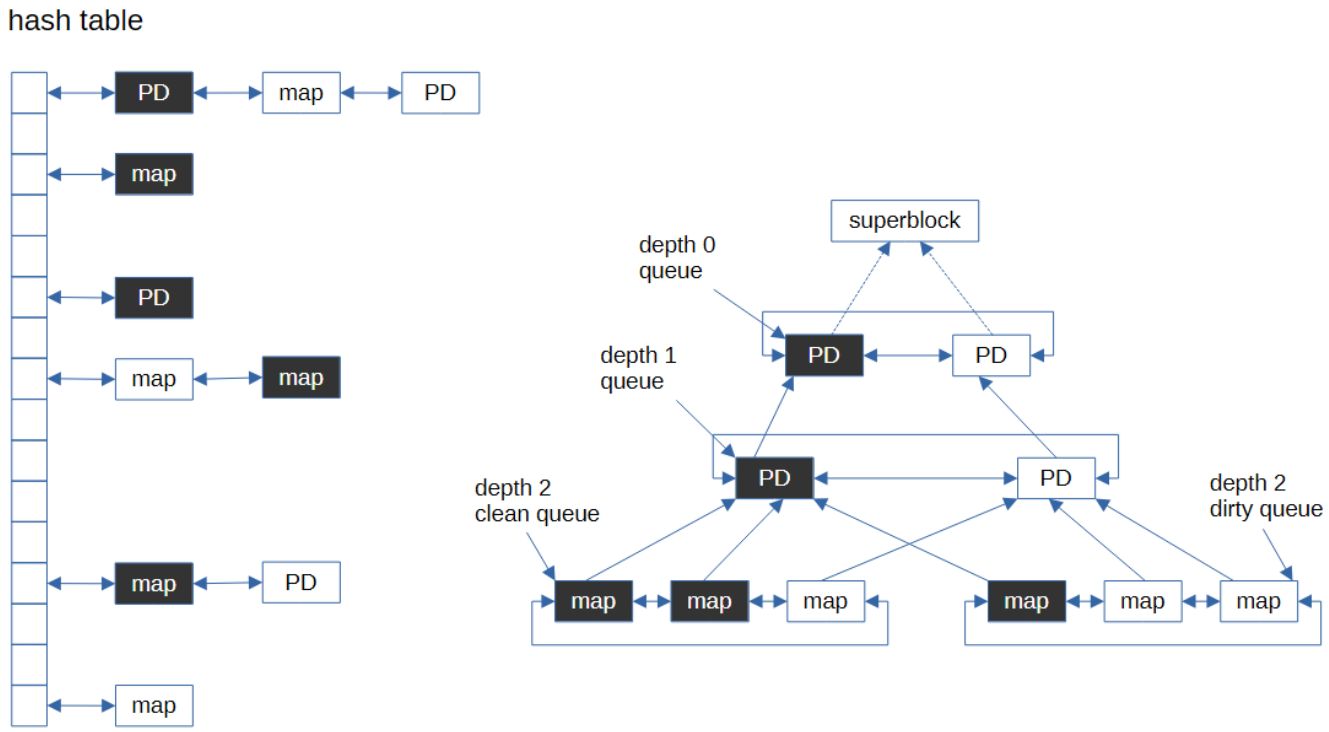

Figure 3. The structure of buffer cache

Figure 4. Two forward maps are used during normal operation

Figure 5. Four forward maps are used during snapshot command