# Detailed study on Linux Logical Volume Manager

Prashanth Nayak, Robert Ricci
Flux Research Group Universitiy of Utah

August 1, 2013

## 1 Introduction

This document aims to provide an introduction to Linux Logical Volume Manager (LVM); its working details and a set of changes that provide LVM with Time Chained Snapshots and Branching Capabilities. We further measure the read and write performance on Logical volumes created using LVM and the performance implications of adding support for time chained snapshots and branching capabilities.

## 2 Logical Volume Manager

The Linux Volume Manager provides us with a mechanism to virtualize disks. Virtual disks can be created using one or more physical hard drives. These disks have the flexibility to be grown, shrunk or moved from one hard drive to another. All this can be done using the interface provided by LVM while hiding the actual details of physical disk management.

Logical Volumes have the following advantages over physical volumes

1. **Flexible Capacity:** Since multiple disks can be aggregated into single logical volume files systems can extend over multiple disks.
2. **Resize-ability:** The size of the logical volume can be increased or decreased without reformatting the underlying disk.
3. **Disk Striping:** Logical volumes can stripe across two or more disks.
4. **Snapshots:** Logical volumes can be snapshotted without having to replicate the entire volume data.
5. **Mirroring:** Data in a logical volume can be mirrored to another logical volume.

### 2.1 LVM Components

1. **Physical Volume:** A hard disk or a partition which has LVM related configuration information written to it.
2. **Volume Group:** It is a collection of one or more physical volumes.
3. **Logical Volume:** It is the virtual disk that has been allocated space on the physical volumes contained in the volume group.
4. **Physical Extent:** Chunks of data the physical volume is divided into. They have the same size as the logical extents in the volume group.

5. **Logical Extent:** Chunks of data that the logical volume is divided into. [2]

## 2.2 LVM Design

LVM is designed to have three main components namely command interface, device mapper library and device mapper driver.

**Command Interface:** The LVM command interface sits in the userspace and provides commands to control the logical volumes, physical volumes and volume groups. It provides a set of commands that facilitate creation, deletion, resizing and other related activities on logical volumes, physical volumes and volume groups.

**Device Mapper Library:** It acts as an interface between the device mapper driver and user space applications and thus hides the differences between the various versions of the device mapper.

**Device Mapper Driver:** Device Mapper Driver provides a mapping from the logical volumes to the physical disk blocks. It maintains a device mapping table for each logical volume and uses it to map request to the corresponding physical disk blocks. The mapping provided by the device mapper could be Linear, Striped, Snapshot, Snapshot-Origin; based on which the IO requests are redirected to the appropriate underlying physical device.

## 2.3 Work-flows

In the following subsection the typical work-flows for Logical volume and Snapshot volume creation is explained.

### 2.3.1 Logical Volume Creation

The user initiates a logical volume creation request by specifying the name, size and the volume group to be used. This new logical volume is then assigned a set of free physical extents from the set of free extent that is maintained by LVM based on one of the allocation policies. The device mapper kernel module is then responsible to create a mapping table to maintain mappings from the logical volume to the underlying physical volumes, these mappings vary based on the kind of logical volume created. At boot time LVM scans its metadata in physical volumes within the volume groups and registers each logical volume and its mapping table with the device mapper which in turn registers it with the kernel. Thus any IO request from logical volumes is redirected through the device mapper, which maps it to the corresponding physical block device.[1]

### 2.3.2 Snapshot Volume Creation

The user initiates a snapshot creation by providing the name, size and the origin logical volume. LVM creates a logical volume similar to any other logical volume by following the same steps mentioned in the above subsection. It then creates another virtualization layer between the logical volume and physical volume to keep track of all the changes to blocks since the creation of the snapshot. Any write request to the original logical volume after creation of the snapshot results in a COW to the snapshot volume. Reads and writes to the snapshot volume and redirected appropriately by looking up the exception table in the additional virtualization layer that was created. This process is explained in detail in the following subsection. [1]

#### 2.3.3 Read/Write Flows

**Logical Volume**

1. Reads to the logical volume are directly sent down to the corresponding target device.
2. Writes to the logical volume may first trigger a COW to the snapshot device if it is already not written to after creation of the snapshot, in this case an exception table entry is added. Else it is directly sent down to the corresponding target device.

**Snapshot Volume**

1. Reads to the Snapshot are either sent to the origin device or sent down to corresponding target device based on the exception table entry.
2. Writes to the Snapshot are directly sent down to the corresponding target device.

## 2.4 Performance Evaluation of LVM

The test configuration consisted of LVM running on a Linux kernel, which was configured as the domain0 guest in a Xen virtualization environment. The exact testbed details is as flows

Version of Linux Kernel used: 2.6.18.
LVM version: 2.02.09.
Library version: 1.02.09.
Driver version: 4.7.0.
Disk Drive :$WDCWD5003ABYX$
Maximum Sustained Rate of the Disks : $128MB/s$.
Tool used to measure the Read/Write performance : Linux *dd* command.

The read/write performance is independent of the amount of data written. This was tested and confirmed with data sizes of 5GB, 10GB and 15GB. However the performance is greatly dependent on the block size used. LVM works at the 4KB block size granularity and hence using a block size of less that 4KB for writes triggers an additional COW overhead and affects write performance. In all of our experiments we make uses of 1MB block size to avoid the additional COW overhead. In all of the tests we make use of volumes, snapshots and branches of size 7GB. The amount of data read or written at any time is 5GB.

### 2.4.1 Read and write performance on Logical Volumes

The experimental setup consist of a 7GB logical volume with no snapshots associated with it. Both reads and writes were able to achieve the maximum sustained IO rate of 128MB/s. This performance was possible because there would be no exception table lookup associated with read or writes as no snapshots were created on the logical volume and read/write requests would directly go down directly to the target device.

### 2.4.2 Read and write performance on Snapshot Volumes

The experimental setup consist of a 7GB snapshot volume created based on a 7GB logical volume. Writes achieved the maximum sustained rate of 128MB/s as they directly went down on the target device however went through at 29.8MB/s because of the overhead of exception table lookup to locate the target device.

#### 2.4.3 Read and write performance on Logical Volumes in the presence of snapshots

The experimental setup consist of multiple 7GB snapshot volume created based on a 7GB logical volume. There is a continuous degradation of write performance depending on the number of snapshots created because of the COW overhead incurred for each write. The read performance remains constant and and does not depend on the number of snapshots as they directly go down to the corresponding target device.

Table 1: Read and write performance on Logical Volumes in the presence of snapshots

| Snapshot Count | 0 | 1 | 2 | 5 | 7 | 10 |
|---|---|---|---|---|---|---|
| Write | 128MB/s | 23.5MB/s | 8.3 MB/s | 6.1 MB/s | 5.2 MB/s | 4.1 MB/s |
| Read | 128MB/s | 128MB/s | 128MB/s | 128MB/s | 128MB/s | 128MB/s |

# 3 Changes to LVM

## 3.1 Immutable snapshots

LVM provides snapshots to support backup and recovery mechanism. However these snapshot are mutable allowing a snapshot volume to be directly mounted and used to service both write and read requests. Mutable snapshots do not support pure branching semantics; allowing snapshot data to change leaves the user without the ability to create multiple branches originating from the same snapshot. The changed version of LVM makes these snapshots immutable and hence the snapshot volumes can serve only read requests. It also allows separation of snapshotting and branching functionality.[3]

## 3.2 Branching

The changed version of LVM introduced a notion of Branches. A branch is a mutable target that can be created off immutable snapshots. All writes to the branch are directly sent down whereas reads can be served from the branch volume if the block exist else it is served form the branch origin. Thus immutable snapshots and mutable branches split the snapshotting functionality provided by LVM into distinct features.[3]

## 3.3 Time chained snapshots

LVM Snapshots were directly associated to its origin device hence any writes to the snapshot would trigger a Copy-On-Write to each of the snapshots created on that snapshot origin. This had a liner impact on write performance on the snapshot origin; larger the number of snapshots larger the number of COW that were initiated. The changed version of LVM made snapshots to be time chained. Hence a write to the snapshot origin would only result in one COW to the latest snapshot that was created on the snapshot origin resulting in a small constant degradation in write performance. However there was a small effect on the read performance on snapshots; in worst case the read request would have to traverse trough all the newer snapshots and the finally the origin to locate the block.[3]

## 3.4 Performance Evaluation

The test configuration consisted of LVM running on a Linux kernel, which was configured as the domain0 guest in a Xen virtualization environment. The exact testbed details is as flows

Version of Linux Kernel used: 2.6.18.
LVM version: 2.02.09.
Library version: 1.02.09.
Driver version: 4.7.0.
Disk Drive :$ST3250310NS$
Maximum Sustained Rate of the Disks : $105MB/s$.
Tool used to measure the Read/Write performance : Linux *dd* command.

The read/write performance is independent of the amount of data written. This was tested and confirmed with data sizes of 5GB, 10GB and 15GB. However the performance is greatly dependent on the block size used. LVM works at the 4KB block size granularity and hence using a block size of less that 4KB for writes triggers an additional COW overhead and affects write performance. In all of our experiments we make uses of 1MB block size to avoid the additional COW overhead. In all of the tests we make use of volumes, snapshots and branches of size 7GB. The amount of data read or written at any time is 5GB.

### 3.4.1 Read Performance in the presence of Snapshots

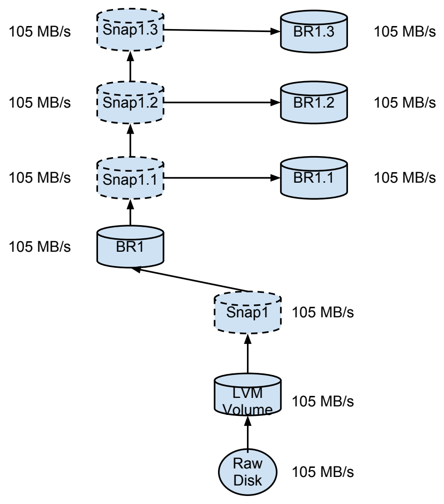

Figure 1:

The experiment setup is as shown in Figure 1. The numbers beside each block device indicates the read performance on the corresponding device. In the absence of any writes the read performance matched the Maximum Sustained Rate of $105MB/s$. This performance was possible because the exception tables for the snapshots and branches would be all empty and no significant time would be spent on lookups.

### 3.4.2 Write Performance in the presence of Snapshots

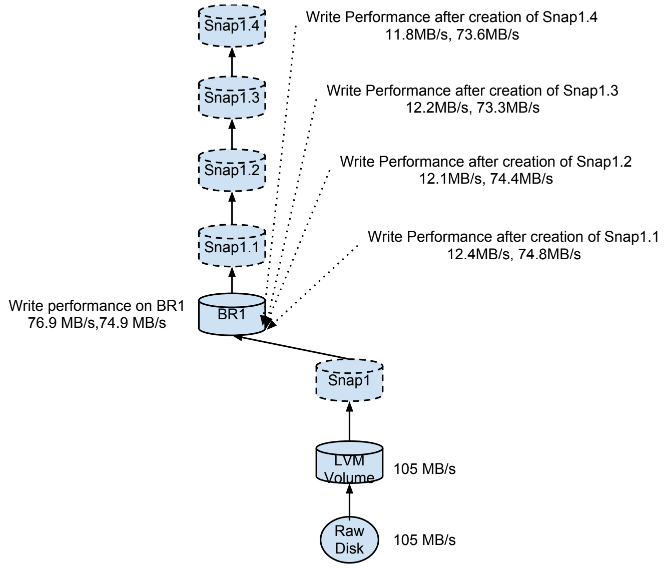

Figure 2:

The experiment setup is as shown in Figure 2. In the absence of any snapshots or branches the writes to the raw disk or LVM volume achieve the Maximum Sustained rate of 105MB/s. When the first branch BR1 was created on Snapshot Snap1 the writes to BR1 achieved a maximum performance of 76.9MB/S. This 30% loss in performance was due to creation of exception table entry for each data block that was written to the branch.
After Snapshot Snap1.1 was created on Branch BR1 the first write achieved a performance of 12.1MB/s and the next write achieved a performance of 74.4MB/s. This pattern continued for each subsequent snapshot creation. This significant (about 90%) drop in performance is because of the exception table lookup and Copy-On-Write overhead for every new block that is written. Once the block is copied to the latest snapshot the following writes to those blocks only incur a exception table lookup overhead; this explains the write performance of 74.8MB/s.

### 3.4.3 Write Performance on Branches created off Snapshots

The experiment setup is as shown in Figure 3. The numbers besides each block device indicates the write performance on the corresponding device. The branch BR1.1 created off snapshot Snap1.1 achieved a maximum write performance of 67.0MB/s. A similar performance was observed for all other branches that were created off other snapshots Snap1.2, Snap1.3 and Snap1.4. This 35% degradation in performance is because of the new exception table entry that is created for each and every block that is written.

### 3.4.4 Worst case read performance of the Snapshots.

The experiment setup is as shown in Figure 4. After branch BR1 is created we fill it with 7GB of data and create a snapshot Snap1.1 over it. We write 1 GB of data on BR1 starting from a seek position of 1000, this results in 1GB of data begin

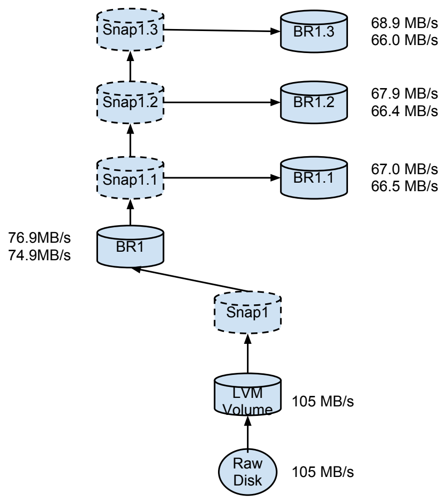

Figure 3:

copied to Snap1.1 as part of COW. In the next step we create snapshot Snap1.2 and write 1GB of data on BR1 starting from a seek position of 2000. This procedure is repeated for every snapshot (Snap1.3, Snap1.4, Snap1.5, Snap1.6, Snap1.7) that is taken. The resulting read performance on each snapshots is shown besides the corresponding snapshot device in Figure4. As the number of snapshots increase the number of exception table lookup also increase hence read performance is affected.

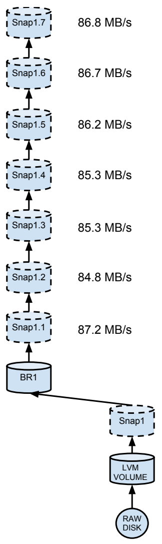

Figure 4:

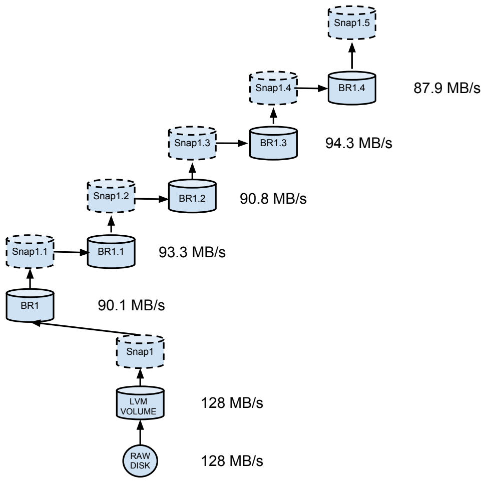

Figure 5:

### 3.4.5 Achieving constant performance using snapshots and branches

The experiment setup is as shown in Figure 5. The numbers besides each block device indicates the write performance on the corresponding device. The branches BR1.1, BR1.2,BR1.3 created off corresponding snapshot Snap1.1, Snap1.2,Snap1.3 achieved write performance of about 90MB/s (sustained IO rate was 128MB/s). A similar performance was observed for all other branches that were created in the similar manner. This 35% degradation in performance is because of the new exception table entry that is created for each and every block that is written on the branch.

## 4 Conclusion

1. Traditional LVM provides mutable snapshots thus combining the snapshotting and branching functionally.
2. Mutable snapshots does not allow the user to create multiple alternative branches originating from the same source volume.
3. Each snapshot in traditional LVM is directly associated with its source volume hence writes to source volume result in COW overhead to each of the snapshots resulting in huge degradation in performance. This performance degradation is directly proportional to the number of snapshots as demonstrated in the performance evaluation in section 2.4.
4. Immutable snapshots and mutable branches provide clean separation of snapshotting and branching functionality, allowing users to create multiple branches originating from the same source volume.
5. Time chained snapshots significantly reduces the COW overheads associated with snapshotting in traditional LVM. By using a combination of snapshots and branches a constant degradation in performance of about 35% can be achieved as demonstrated in section 3.4.5.

6. Both traditional LVM and changed LVM rely on COW technology to provide snapshot and branching support. Internally both of them use mapping tables to track different versions of block data on snapshot and branch volumes. These COW overheads and mapping table lookups do not make LVM suitable for fine grained high frequency snapshotting.

## References

[1] Bhavana Shah.(2006). *Disk Performance of Copy-On-Write Snapshot Logical Volumes* (Master's Thesis). The University of British Columbia.

[2] A.J Lewis.(2006). *LVM HOWTO* Retrieved from http://tldp.org/HOWTO/LVM-HOWTO/index.html

[3] Prashanth Radhakrishnan.(2008). *Stateful-Swapping in Emulab network testbed* (Master's Thesis). The University of Utah.