Linux Storage Replication

Lars Ellenberg

LINBIT HA Solutions GmbH
Vienna, Austria
What this talk is about

• What is replication
• Why block level replication
• Why replication
• What do we have to deal with
• How we are dealing with it now
• Where development is headed
Linux Storage Replication

- Replication Basics
- DRBD 8 Overview
- DM-Replicator
- DRBD 9
- Other Ideas
Linux Storage Replication

- Replication Basics
- DRBD 8 Overview
- DM-Replicator
- DRBD 9
- Other Ideas
Standalone Servers

- No System Level Redundancy
- Vulnerable to Failures

Important Systems

Node 1  Node 2  Node 3
Application Level Replication

- Special Purpose Solution
- Difficult to add to an application after the fact
Filesystem Level Replication

- Special Filesystem
- Complex
- Replicate on dirty?
- ... on writeout?
- ... on close?
- What about metadata?
- Resilience?
Shared Storage (SAN)

- No Storage Redundancy

Important Systems

Node 1  Node 2  Node 3

FC, iSCSI

Shared data
Replication capable SAN

- Application agnostic
- Expensive Hardware
- Expensive License costs
Block Level Replication

- Storage Redundancy
- Application Agnostic
- Generic
- Flexible
SAN Replacement Storage Cluster

- Storage Redundancy
- Application Agnostic
- Generic
- Flexible
Linux Storage Replication

- Replication Basics
- DRBD 8 Overview
- DM-Replicator
- DRBD 9
- Other Ideas
How it works: Normal operation

Application

Data blocks

Primary Node

Replicate

Acknowledge

Secondary Node

Data blocks

Write I/O

Read I/O

Write I/O

Read I/O

Replicate

Acknowledgment
How it works: Primary Node Failure

Application

Data blocks

Write I/O

Replicate

Acknowledge

Primary Node

Secondary Node

Read I/O

Read I/O
How it works: Secondary Node Failure

Application

Data blocks

Primary Node

Data blocks

Offline Node
How it works: Secondary Node Recovery

Application

Data blocks

Primary Node

Data blocks

Secondary Node

Resync

Resync

Acknowledgement

Acknowledgement

Read I/O

Read I/O

LINBIT
YOUR WAY TO HIGH AVAILABILITY
What if ...

- We want additional replica for desaster recovery
  - we can stack DRBD
- The latency to the remote site is too high
  - stack DRBD for local redundancy, run the high latency link in asynchronous mode, add buffering and compressing with DRBD proxy
- Primary node/site fails during resync
  - Snapshot before becoming sync target
It Works.

• Though it may be ugly.

• Can we do better?
Linux Storage Replication

- Replication Basics
- DRBD 8 Overview
- DM-Replicator
- DRBD 9
- Other Ideas
Generic Replication Framework

- Track Data changes
  - Persistent (on Disk) Data Journal
  - “global” write ordering over multiple volumes
  - Fallback to bitmap based change tracking
- Multi-node.
  - many “site links” feed from the journal
- Flexible Policy
  - When to report completion to upper layers
  - (when to) do fallback to bitmap
Current „default“ reference implementation

- Only talks to “dumb” block devices
- “Software RAID1” allowing some legs to lag behind
- No concept of “data generation”
- Cannot communicate metadata
- Not directly suitable for failover solutions
- Primary objective: cut down on “hardware” replication licence costs, replicate SAN-LUNs in software to desaster recovery sites.
Replicating smarter, asynchronous

- Detect and discard overwrites
  - shipped batches must be atomic
- Compress
- Compress XOR-diff
- Side effects
  - Can be undone
  - Checkpointing of generic block data
  - Point in time recovery
Replicating smarter, synchronous

- Identify a certain Data Set Version
- Start from scratch
- continuous stream of changes
- Data Generation Tags, \textit{dagtag}
  - which clone (node name)
  - which volume (label)
  - who modified it last (committer)
  - modification date (position in the change stream)
Colorful Replication Stream

Data Set Divergence

Primary Node Changes

atomic batch discarding overwrites
Advantages of the Data Generation Tag scheme

- On handshake, exchange *dagtags*
  - Trivially see who has the best data even on primary site failure with multiple secondaries possibly lagging behind
- Communicate dagtags with atomic (compressed, xor-diff) batches
  - allows for daisy chaining
- keep dagtag and batch payload
  - Checkpointing: just store the *dagtag*.
DRBD 9

- Replication Basics
- DRBD 8 Overview
- DM-Replicator
- DRBD 9
- Other Ideas
Stretched cluster file systems?

- Multiple branch offices
- One cluster filesystem
- Latency would make unusable
- But when
  - keeping leases and
  - inserting lock requests into the replication data stream
  - while having mostly self-contained access in the branch offices
- It may feel like low latency most of the time, with occasional longer delays on access.
- Tell me why I'm wrong :-(
Comments?

lars@linbit.com

http://www.linbit.com
http://www.drbd.org

If you think you can help,

we are Hiring!