Disaster Recovery

Where it stands on Linux

James Bottomley

SteelEye Technology

3 August 2004

What Is Disaster Recovery?

- Can mean a multitude of things.
- From the simple weekly backup and the transport of tapes to an offsite storage place, to
- A fully automated instantaneous geographic takeover from a live standby site when a disaster strikes the primary site.
- Disaster Recovery is any system which allows recovery of operation in the face of a systemic, site wide failure.
- This distinguishes it from High Availability which is the recovery from individual points of failure within a site.

Distinction between Disaster Recovery and High Availability

- High Availability is all about fast recovery from any failure, however small.
  - There's little cost to doing a recovery in a HA environment.
  - recoveries are automated as far as is possible
- Disaster Recovery should be performed *only* if there's a real need
  - Can take a long while, lose data, be expensive to effect and have manual steps that can't be automated
  - for this reason often require manual authorisation that a Disaster really has occurred.

Distinction between Disaster Recovery and High Availability Continued

- To generalise:
  - High Availability is about protecting the applications (the data being assumed to be safe).
  - Disaster Recovery is about protecting the Data (the applications being assumed to be recoverable later).
- Thus, all Disaster Recovery architectures are data centric.
- With application recovery a secondary requirement.

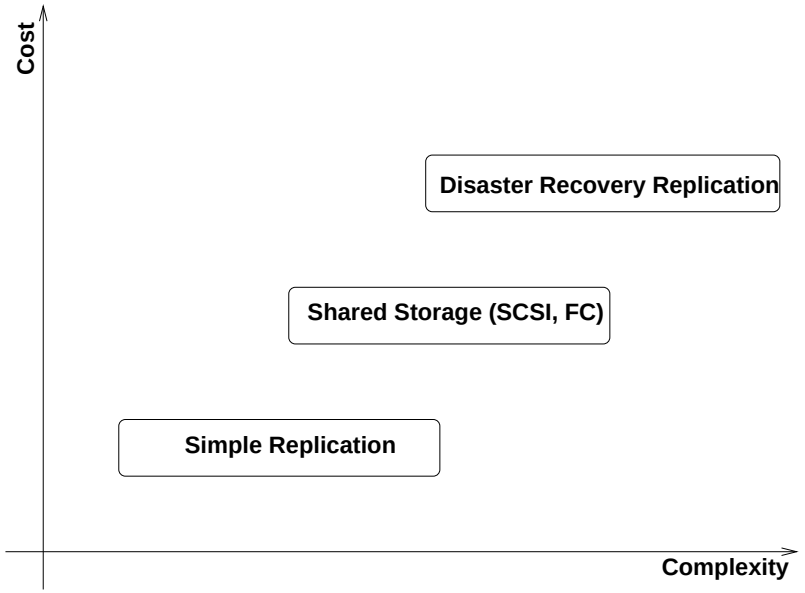
Key Elements of Disaster Recovery

- The primary point is that some form of backup is available, sufficiently far away so as not to be affected by the disaster.
- Thus, making tapes and transporting them offsite constitutes a disaster recovery solution.
- Making tapes and leaving them on-site does *not*.
- Recovery from the disaster may lose data.
  - Again, this distinguishes it from High Availability, where one of the key elements is no loss of committed transactions.

Disaster Tolerance Criteria

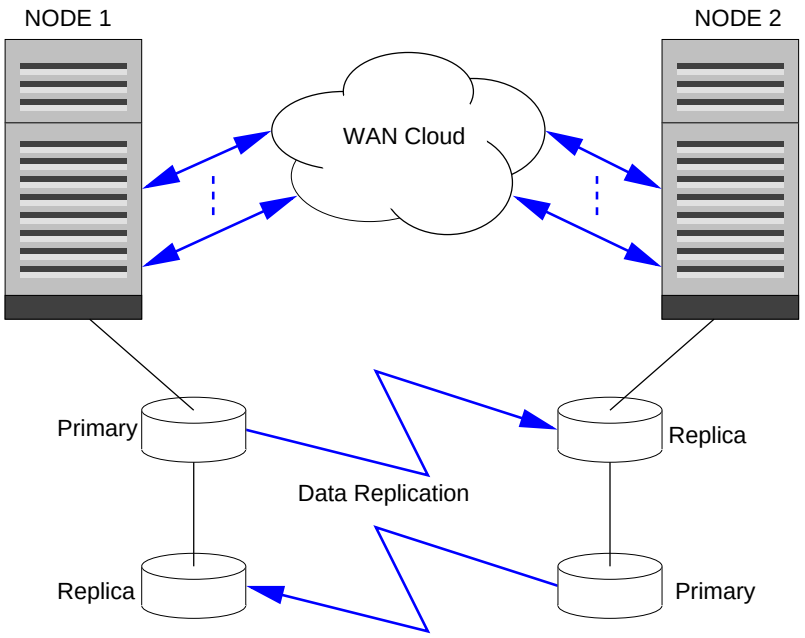
- There are two key criteria for determining the nature of a Disaster Recovery solution:
  1. How much data are you willing to lose? and
  2. How quickly do you need to become operational.
- If the answers to both these are measured in days, then probably a simple offsite tape rotation backup is sufficient for all your needs.
- If you need more stringent limits, then you probably require the continuous live backups afforded by replication.

The Paradox of Replication



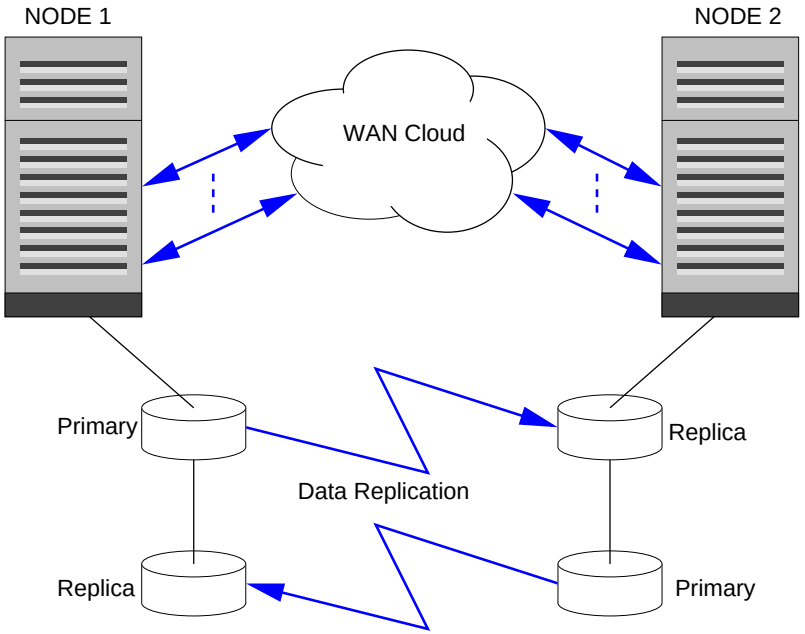
- Replication technologies occupy both the low and the high ends of the market
- The low end is primarily as a cheap alternative to shared storage in HA clusters.
- Concentrate on High End. technology

Replication



- This is a standard replicating system
- two nodes joined by some type of network
- The internal storage of the nodes is replicated by software across the network.

Replication, Continued



- Essentially, this is a distributed RAID-1 (mirror) system
- RAID-1 is old hat technology. The secret for disaster recovery is in other parts:
  - Asynchronicity
  - Logging

At What Level Should Replication Operate?

- Replicators can operate either at the Application level, the file level or at the block level.
- Examples are:
  - Application Level: MySQL data replicator
  - File Level: snapshot backup file-system
  - Block Level: Just about every replication technology (drbd, md/nbd etc).
- Block Level replicators are far and away the most popular because they don't depend either on application data formats or on file-system layouts (i.e. they're the most general).
- Format specific replicators can be better tuned.

Replication and Transaction Integrity

- All applications and file-systems commit data in integral units called “transactions”.
- For the replication stream to respect transaction boundaries (and thus the integrity of the data) it must respect the *ordering* of the data in the stream.
- For a block replicator, this means the order of the blocks as they come in must be strictly observed in the order they’re committed to the replica device.

How Big a Network Pipe Do You Need?

- One of the essential prerequisites for establishing your network requirements is knowing your data volume.
- There are two useful figures which characterise this:
  - The Average Bandwidth: This is simply the total data throughput averaged over a long time period (like a week).
  - The Sustained Peak: This takes the same time period, but calculates the data rate on an hourly basis. The sustained peak is the largest of these.
- The network bandwidth should lie between these two figures.

Flavours of Replication

- Replication comes essentially in three flavours:
  - **Synchronous** The simplest kind: Data is not acknowledged as being committed until it is safely on both the primary and secondary devices.
  - **Asynchronous** Data may be acknowledged as committed when it is safely on the primary, but may be still in-flight to the secondary.
  - **Periodic** This isn't really a continuously replicating system. The system is mostly operated on the primary only. Periodically, the differences between the primary and the secondary are sent to the secondary to bring it up to date.

Asynchronous Replication

- The data that has been acknowledged as committed (to the primary) but not yet sent to the secondary must be cached within the primary.
- This cache is primarily used to smooth out bursts of data that temporarily go over the available network bandwidth.
- Once the asynchronous cache is full, the application will be flow controlled (slowed down) to the actual network speed. The replication becomes effectively synchronous.
- Sizing the cache becomes important. Usually it should be based on the difference between Network Bandwidth and Sustained Peak.

Logging

- The primary purpose of logging is to protect the integrity of the replication and to resume as fast as possible after the problems are sorted out.
- The usual integrity problems are:
  1. Prolonged network outages (where the communications outage causes the mirror to break).
  2. Non Disaster failures of the primary (where the data is not transferred and the service is eventually restarted on the primary).
- Logging records the data that needs to be sent to the replica in order to bring the two volumes into sync.

Important Considerations in Log Placement

- The key problem with a transaction log is that it must be written to at the *same* time as the local data volume.
- Whenever dependent write activity like this is going on, the basic rule of thumb is "separate spindles".
- this means that the intent log should be on a separate physical disc (or set of discs) from the data.
- The reason for this is head rattle: If two areas of the disc are in use by I/O operations that depend on each other, the disc head must move backward and forward between them for each dependent operation.

Log Volatility

- The previous slide assumes a non-volatile log (i.e. the Log is stored on a device)
- However, it is possible to store the log *only* in the memory of the primary.
- Such a placement is called a volatile log.
- The key disadvantage is that the log is lost if anything happens to the primary.
- Temporary crash of the primary, or any other transient effect necessitating a reboot loses the log.
- Once the log is lost, the only way to get the primary and replica back into agreement is a full resynchronisation.

Transaction Logging

- A transaction log is basically a time ordered log of data (and location).
- The time ordering is what preserves transaction integrity.
- As soon as the data is safely in the transaction log, it can be acknowledged as committed (asynchronous) before it reaches either the primary or replica devices.
- Since the transaction log contains the data, any required log replay preserves the integrity of the replica.
- The time ordered log may contain multiple copies of the *same* sector, hence the log could grow larger than the actual volume.

Transaction Logging II

- Transaction logs need to be large and may grow without bound when communication is lost with the replica.
- Usually, transaction logs have a fixed size area devoted to them, so once they overflow, you cannot replay the log and have no choice but to synchronise every block on the disc.
- Can actually *speed* up the application by having the transaction log on a fast (say SSD) device.
- However, always need some back end daemon clearing the log and applying the data to both the primary and the secondary.

Transaction Logging III

- this back end daemon works from the back of the log while data is committed to the front causing head rattle.
- Every piece of data goes once to the log and then again to the primary device and the secondary. Thus the I/O throughput of the system is degraded by the additional writes.
- The transaction log is effectively the cache for asynchronous writes referred to in a previous slide.

Intent Logging

- An intent log is simply a record of blocks that differ between the primary and replica (the name comes from Intention to Write log).
- since data is not recorded in the intent log, the intention to write *and* the actual data must be committed on the primary before the write can be acknowledged (two I/O operations).
- The intent log can be a simple bitmap (one bit per block), so it's very small and a fixed size (can never overflow).
- When the mirror is broken and restored, only the changed data needs be transmitted, however many times it has changed in the interim.

Intent Logging II

- However, during a replay from the intent log, data is sent to the replica out of order, so the replica is *unusable* until the replay is complete.
- Usually during normal application operation "hot spots" develop. If the log clearing daemon is careful about ageing sectors, most hot spots wind up having their bits already set in the log. Thus no need to do a log write.
- For a prolonged outage, the amount of data transferred for a resynchronisation is usually much less for an intent log than for a transaction log.
- Intent logged systems require a *separate* cache for data acknowledged but not yet committed to the replica.

Periodic Replication

- Like automated offsite backup.
- *Requires* logging to work efficiently
- Usually pick quiet times (e.g. at night) to ship changed data from primary to replica.
- Best solution for narrow pipes if you calculate your daily bandwidth will go over the requirements.
- Works best with intent logging (for multiple changes to the same data only the latest change is transmitted).

Resynchronisation Tricks

- Obviously, a full Resynchronisation (especially over a narrow bandwidth WAN) is one of the most expensive operations on a mirror.
- However, two conditions often apply:
  1. Data is blank (usually because the entire disc isn't full)
  2. Data is the same on the primary and replica.
- Therefore can speed up resynchronisation by:
  1. having a special signal that conveys an empty block.
  2. before transmitting the block from primary to replica, have the replica transmit the md5sum of its block. If they match, high probability that they contain the same data, so no need to resend it.

The Failback Scenario

- After a disaster has struck and operations have been established at the replica site, the operation will eventually need to be returned to the original primary site when the disaster is alleviated
- Problem is that although you can record the changes you made to the replica, you need to combine this with the record of changes made to the primary to deduce the full set of data needed for a resync.
- Using an intent log makes this easy: you simply or the two intent logs together and the resulting log contains the set of all data that needs to be replayed.
- Thus, with an intent log, you can dispense with the requirement for a full replay on failback.

Hybrid Logs

- There are specific primary advantages and disadvantages to each of the logging techniques:

	Advantages	Disadvantages
Intent Log	small size, never overflows	replica unusable during replay
Transaction Log	replica usable but out of date during replay	large size, may overflow

- So the question arises, can you combine the technologies somehow to get the best of both while minimising the disadvantages?

Hybrid Logs II

- There exists a standard technique to combine the advantages of the two logging techniques.
- Relies on transaction log being much larger than the intent log.
- If transaction log overflows, or needs to be used for failback it can be converted to an intent log in the space the transaction log used to occupy.
- Conversion is simple and can be done while the mirror is active.

Converting a Transaction to an Intent Log

- Procedure is:
  1. create an in-memory image of the intent log, initially all clean.
  2. begin taking data out of the transaction log from the beginning and mark it dirty in the in-memory intent log
  3. When you've pulled enough data to fit the intent log, write the in-memory copy to the intent log, make the mirror use it and replay the rest of the transaction log through it.
- Obviously, can also convert back again while mirror is operational.

Linux Replication Solutions

- Solutions exist both in Open Source and Closed Source.
- Both Open Source solutions are intent log based.
- Veritas rumoured to have SRL replication technology available but not released on Linux which is transaction log based.
- Replication technology is invasive to the operating system, so principal disadvantage with closed source solutions is getting timely kernels that match the distributor;
- secondary problem is that full kernel replacement often invalidates distribution vendor's support agreements.

MD/NBD

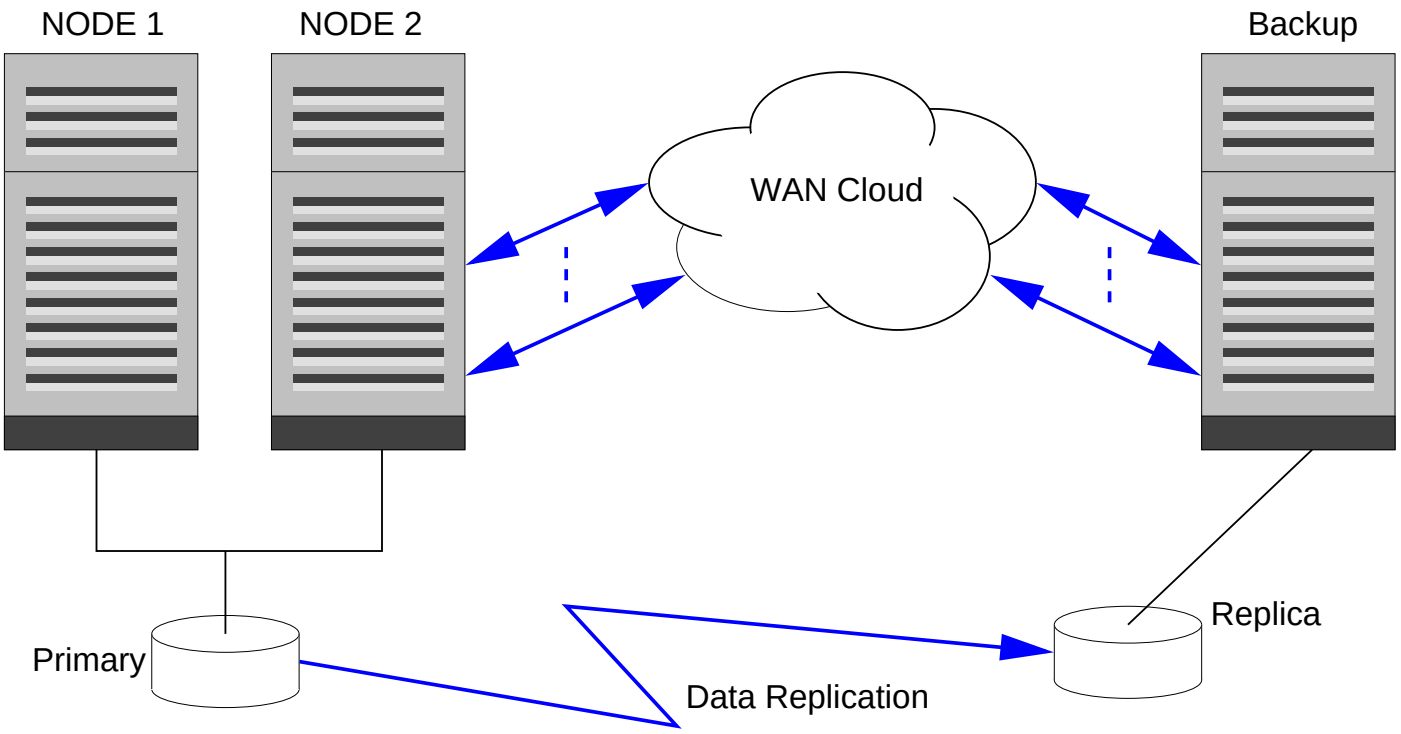
- Work for 2.6 is being sponsored by SteelEye Technology, Inc.
- Based on existing in-kernel md (Redundant Array) and nbd (network block device).
- proposed solution involves adding a non-volatile intent log and asynchronous capability to the existing in-kernel md driver.
- patches to implement this capability are being reviewed on the kernel mailing lists for inclusion into 2.6.
- Forms the basis of our LifeKeeper Disaster Recovery Solution.

DRBD

- Project of Philippe Rensner, currently being worked on by SUSE.
- Adds a completely new driver to the kernel
- system is a simple mirror with a volatile intent log and asynchronous capability.
- Project is not currently on inclusion track for 2.6

Stretch Clusters

- This is a concept that receives much attention.
- Idea is to have a local cluster replicating to a remote system.



Stretch Clusters II

- Local cluster protects against transient failures (provides high availability)
- Remote replication provides backup in case true disaster strikes (bringing down local cluster completely)
- Perfectly feasible with current technology *provided* the log is non-volatile.

Conclusions

- Disaster Recovery using Replication is a viable solution today.
- It can be implemented today using completely open sourced components.
- Subtleties in the implementation often leads to service oriented offerings to assist implementing organisations.
- For all around protection, use the stretch cluster concept.