Shared Storage Clusters

James Bottomley
SteelEye Technology

Introduction to High Availability using commodity shared storage hardware

This talk gives an alternative perspective on High Availability (HA) to the approach taken by the Linux−HA group.There are conventionally two approaches to Implementing HA: Entire system in software (the Linux−HA approach) or entire system in hardware (e.g. the AT&T 3B20)

Shared storage clustering is a marriage of the two approaches.Through judicious choice of commodity hardware, several of the morechallenging problems inherent in a software only HA solution may befinessed (that is, they a rendered non−existent by the correcthardware configuration). At the same time, shared storage clusteringprovides the flexibility and low total cost of ownership expected froma software based solution.

This talk will discuss the following topics:

- Problems which may be finessed by shared storage: cluster partitioning and membership, reduced storage requirements, lock management, volume replication strategies and transaction losses, N down to 1 failure, Write intensive application (e.g. E−Commerce).
- Problems introduced by shared storage: File System repair after failure (specifically the need for a Journalling FS), cluster must be co−located (maximum separation is 2km using a SAN).
- Problems specific to the Linux implementation of shared storage clusters: Multi−initiator SCSI and SCSI reservation handling in the Linux kernel. IP switchover without MAC address changing.
- Integrating the Linux−HA meta−clusters approach to enable geographic dispersion of shared storage clusters for disaster recovery.

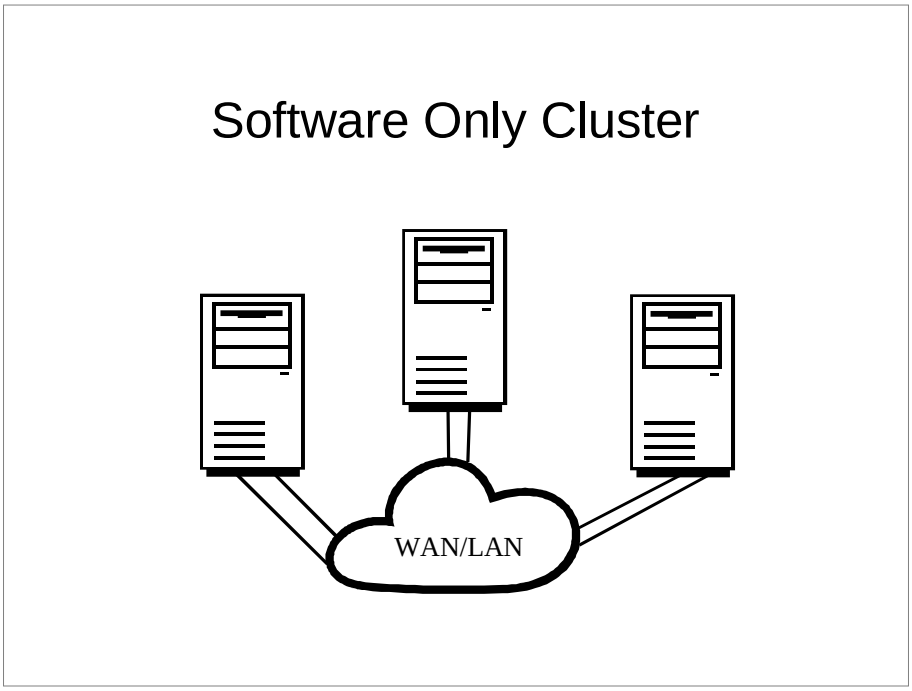
The Types of High Availability

- Fully Hardware Based − Fully redundant machines (AT&T 3B20) or special Cluster Hardware.
- Fully Software Based − Clustering (Linux−HA project)
- Hybrid − Shared Storage Clusters, build from commodity Hardware, supply clustering software

Fully hardware based is too expensive for small commercial installations to use − special hardware coupled with low volumes makes them ~10x more expensive than commodity. No interoperability standard=>all components from one vendor=> drives prices way up. This is a well understood approach, high cost and special drivers make it unviable in the low cost linux world.

Fully software is a nice approach, making clusters wholly commodity based but pays a price in terms of data protection (discuss later)

Hybrid tries to build a hardware cluster from Commodity components. This adds slightly to the cost but is still far cheaper than the Fully Hardware approach. Using commodity HW means cluster can be built from components supplied by competing vendors (competition drives prices down)



Typical configuration of a software only cluster: Individual nodes connected by a network, using replicators to mirror the individual disc devices. This (by design) looks exactly like just a bunch of computers connected to a net, except that they have connection redundancy.

Fully Software Clusters

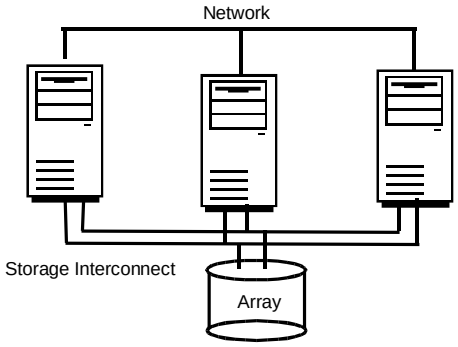
- Advantages:
  - Cheap
  - Easy to disperse Geographically
- Disadvantages:
  - Data concurrency
  - Fault detection and recovery usually Quorum Based

Fully software clusters usually have to have some type of network replicator. This gives geographic dispersion ability over WAN − great advantage for disaster recovery.

Replicators mean data on primary is always slightly out of date with respect to secondary (depends on Bandwidth and latency of network). Special steps must be taken to preserve transaction integrity. Network prone to outages=>special steps taken to avoid complete replay

Entirely network communication based=>Fault detection and recoveryhard. Usually done by a quorum mechanism (each node has a vote > 50% votes form cluster) => can only recover if > 50% of cluster survives (can mitigate using tie breakers like Quorum Disc).

Shared Storage Clusters



Storage interconnect shows redundancy. Network interconnect may be redundant.
Cluster may also export services to the WAN cloud

Shared Storage Clusters

- Advantages:
  - No replication: Data is instantly transferrable between nodes, no transaction loss.
  - All nodes see storage: Cascading protection is much easier
  - Not quorum based (down one failure)
- Disadvantages:
  - Shared storage costs more
  - Clusters are more complex (learning barrier)

Instant accessibility of the data is because on a failure, any connected node may instantly pick up the exact state of the data belonging to the crashed node and repair it. For databases this means there is no transaction loss. Shared storage clusters are ideal for applications with a high write throughput (e.g. electronic commerce)

Recovery is one of the most vulnerable times in a cluster because recovery procedures usually place great resource pressure on the system performing it. A cascade is where and individual (usually one among many) application may fail during the restore phase because of resource pressure. The alternatives are to re−schedule the restore for a later time when others have completed (the serial approach) or to restore the application on a different system immediately (the cascade approach). Any cluster which implements parallel recovery must employ one of these procedures,

The costs of shared storage clusters are related to the sorage array and the necessity for redundancy in the array connections

Complexity is severe for shared SCSI (unique ID’s etc) but should lessen for SAN technology (every SAN is a huge shared SCSI installation).

Protecting the Shared Storage

- All nodes see the storage, so need to ensure protected access
  - Use reservations to guarantee this
  - Operating System must respect reservations
  - All nodes must be able to access storage − needs multi−initiator environment
  - Reservations may be dropped for a variety of reasons, holding node must reassert if dropped
- The above are all problems in Linux.

Each array can be partitioned into many (up to 64) Logical Units (or LUNs). Reservations may be held separately on each LUN => different nodes may own different LUNs. This allows symmetric activity in the cluster.

Storage Ownership

- Need to assign exclusive ownership of storage
  - Do this at the LUN level using SCSI reservations
- Need to break reservations held by failed nodes (support for bus/device/LUN resets)
- Need to detect reservation stealing (prevent "Split Brain" clusters

Reservations are tricky things to maintain. SCSI 2 mandates that storage drop them on bus/device/LUN reset. This means that on error recovery by any node they must be restored (bus reset drops reservations on entire bus, device reset drops reservations for all LUNs on an array, LUN reset is SCSI 3 and not well supported). Need to re−assert reservations as soon as loss is detected => either as part of unit attention negotiation or pro−actively on bus reset detection.

Failed reservation breaking can only be done by issuing a reset from user level. Linux has no support for this (although RedHat 2.2.16−3 does support this feature via the SCSI generic device).

"Split Brain" occurs when all communication paths fail but the nodes are still connected to the storage=>they all try to acquire ownership and busily try to break each other's reservations. Finally there is one winner and all the others detect that their reservations have been pre−empted and relinquish ownership. Can mitigate problem by providing multiple redundant communication paths, or by doing communication over the SCSI bus (e.g. target mode or IP over Fibre).

Linux and Reservations

- Linux treats reservation conflict as an error
  - Error retries reset the bus, thus breaking the reservation
- Can use SCSI generic interface to issue commands (but not messages)
  - No way to issue any form of reset message to break the reservation
- Can patch kernel for above and implement reservation handling at user level

The required patch is already in RedHat kernel 2.2.16−3. With this patch, a user level daemon can be constructed to "watch" the storage. Periodically it issues a reservation request for all storage that is supposed to be reserved. If the reservation was dropped, this re−acquires it. If the reservation has been stolen, the daemon gets a RESERVATION_CONFLICT return and can act accordingly (either quietly take the hierarchy out of service or more drastically panic the node).

Linux Buffer Cache Problems

- The buffer cache does not purge buffers on unmount
  - If a volume transfers between nodes and back, a remount may not see the changes because of stale buffer information
  - Must manually purge buffers via ioctl on unmount
- Partition table may be inaccessible at boot
  - Need an ioctl to re−read the partition table before mounting the drive.

The ioctl to purge the buffer cache is BLKFLSBUF. This must be issued on an unmount of any protected filesystem.

The partition table may not be read at boot time because a reservation was held on the storage. If you attempt to access a partition of a device whose partition table is unknown, linux will not attempt to read the partition table again, it will merely return a not configured error. Before accessing a partition you must issue a BLKRRPART ioctl to force a re−read of the partition table.

Shared Storage Hardware

- Interconnect
  - Shared SCSI (68 pin cables), 12m max length
  - Fibre Channel (several wiring types) up to 2km max length
  - Should be redundant
- Array
  - Redundancy: RAID level >=1
  - RAID must be internal (not host based)

RAID state must be stored *inside* the array (Otherwise it cannot properly be shared among all the nodes in the cluster). Host based RAID does not work in a shared storage environment (except under very limited conditions and with host co−operation).

Shared SCSI

- Old technology (about 8 years)
- Bus is a matched impedance circuit
  - Needs correct termination
  - More devices more problems (any device failure usually sinks the entire bus)
- Devices must be "daisy chained"
  - No known Host Adapter supports this
  - Any interconnect failure kills the entire bus
- Devices must have unique identifier

Technique is easy. Usually viewed as esoteric not because of difficulty but because few people actually consider doing this. Therefore, there is a huge learning curve to setting up a shared storage cluster with shared SCSI.

Problems can be mitigated by using a storage device with many many busses (e.g. EMC Symmetrix) so that each node is connected to storage by a point to point connection (provides fault isolation and lessens impedance matching problems)

Point to point configurations have specific problems with SCSI 2 reservations and cannot be supported (yet) in linux.

Fibre Channel

- New Technology
- Bus is wired like a network
  - Requires a Hub or Fabric Switch
  - Hub provides segment isolation (bad devices don't drag entire bus down)
  - Network status lights make for easy fault isolation.
- Soft loop ID gets around unique device number

Wiring is simple and lightweight (Fibre is best − 2 tiny cables per node) but copper is not much worse (greatly reduced length limitation though).

Fault isolation and detection is a breeze with the hub/card lights. That having been said we did run into some early teething troubles where adding bad cards to the loop caused all other devices to drop off. The fault detection lights didn't mark the cards as bad and they had to be manually found by more traditional techniques.

Soft loop ID means that any initiator joining the SAN negotiates for an unoccupied Fibre ID before becoming fully logged in.

Path Redundancy

- To avoid single point of failure need multiple paths to storage
  - This requires advanced kernel support (path switchover or load balancing)
- Problems with reservation breaking: resets don’t propagate across a split bus
  - Use SCSI 3 reservations, or
  - SCSI 3 mandate for bus device resets

No Linux support at all for this.

Could modify a software RAID driver to support this (it is, after all a type of RAID which says "pick a path, if error pick another"), or use a hardware solution like the Vortex FC card which implements a dual FC port with internal (and invisible) path switchover

SCSI 3 reservations are prersistent across resets (and power fails if the storage supports this). They are also accessed using a key rather than the initiator number so are path independent−−−Ideal solution.

The down side is that SCSI−3 reservations are complex => array vendors don’t like them. Therefore, use the "get out" clause in SCSI 3: All devices should drop reservations on receiving a device reset on any port (note not a bus reset, just a device reset). Therefore a device reset may be used to break reservations.

Unfortunately, in linux at the moment the reset choice is a recommendation to the host adapter which usually ends up choosing a bus reset.

Storage Mapping

- Need to map storage between nodes
  - Must find out that /dev/sdc on node 1 is /dev/sdf on node 2
- Can’t use partition labels (can’t read partition table if reservation is held)
- Use device help
  - World Wide Name (inquiry page 0x83)
  - Serial Number (inquiry page 0x80)

This mapping is required ahead of time so that the cluster knows exactly what to do on a failure.

The World Wide name is guaranteed (by the SCSI−3 standard) to be independent of the path taken to get to the storage. However, not many arrays implement this so fall back on the Serial Number if they don’t. Note the serial number identifies the array not the LUN (solve by tacking on LUN number) and may be path dependent.

The final wrinkle in linux is that the device names may not be the same across reboots (add/remove device causes /dev/sd{x} to shuffle) => always make sure storage device is known by its identifier mapping, not by its device name.

Other Linux Shared Storage Issues

- Shared SCSI
  - Only 8 LUN support
  - Mid Layer crafted for single initiator only
  - No target mode
- Fibre Channel
  - Driven by SCSI subsystem so exactly the same issues, except more acute
  - 16 Device maximum

This addresses kernel issues in 2.2

Issues still exist in 2.4

This is a problem for shared SCSI, but this may be dismissed as a niche market. However, SAN looks exactly like a much larger shared SCSI cluster, therefore these problems will be a killer for SAN support, particularly as it is driven by exactly the same SCSI layer.

More than 8 LUN support can be added by making some of the static tables in the SCSI subsystem linked lists and allowing the low layer max_lun field to dictate instead of a compile time limit.

Can alter current mid layer to be more reset friendly−−−implement backoff delay on reset reciept to prevent reset storms etc.

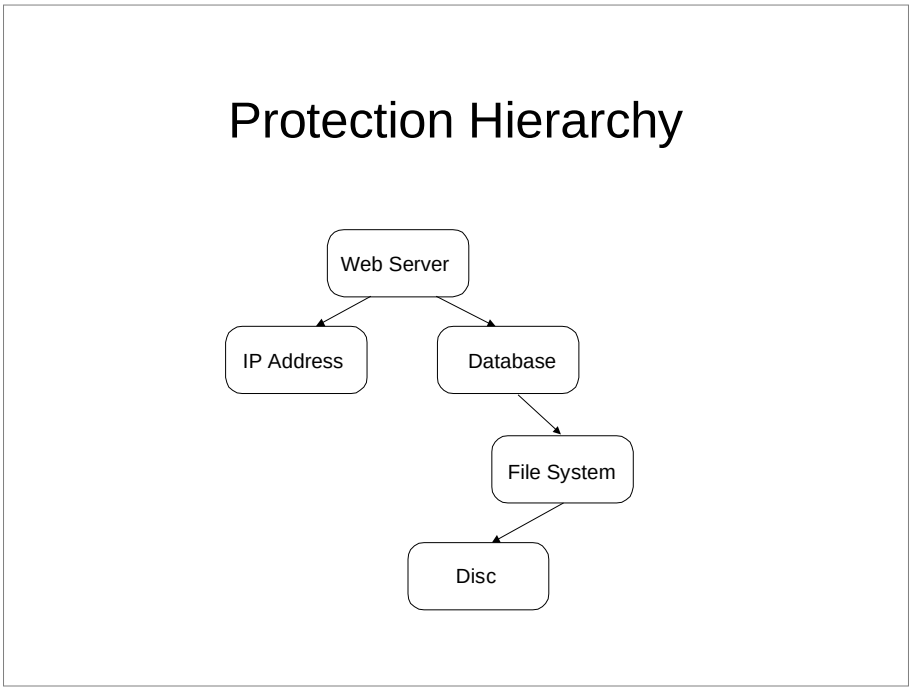
Target mode a problem. Each low layer driver would need to be enhanced to allow it to respond as a target (this is not a minor undertaking). Probably allow low layer to advertise support via the interface.

Solve the 16 device limit on a FC driver using the channel number.

Cluster Operation

- Recover identical resources (e.g. FileSystems, IP etc. and rely on application to recover and restart).
- Cluster must agree on node to perform recovery
- Application must be able make itself sane and continue where it left off.

Present an overview of the well known operation mechanism for all clusters: Cluster recovers resources, application is expected to restore its own state. Good cluster software also throws a protection harness around the application (see later).



A protection hierarchy: The arrows represent the relationship "Is required for me to function", so the webserver requires both an IP address and the database, the database requires a filesystem etc.

The hierarchy restores itself by beginning at the lowest dependency and working upwards.

Data Recovery

- Same problem as block replication: must recover filesystem fast
  - Could finesse by bypassing FS (raw devices)
  - Mitigate delay by using Journalling FS
- Better than replication: recovering node sees same data as failed node
  - No committed transaction loss
  - Ideal for write intensive applications

Both shared and non−shared storage clusters have the same issues with data recovery as far as the FS repair goes. (This actually depends on the replicator: usually block based, but could be FS or even SQL based).

Shared storage guarantees no committed transaction loss. For databases, and hence electronic commerce shopping cart and B to B applications, this is a huge win.

Cluster Partitioning

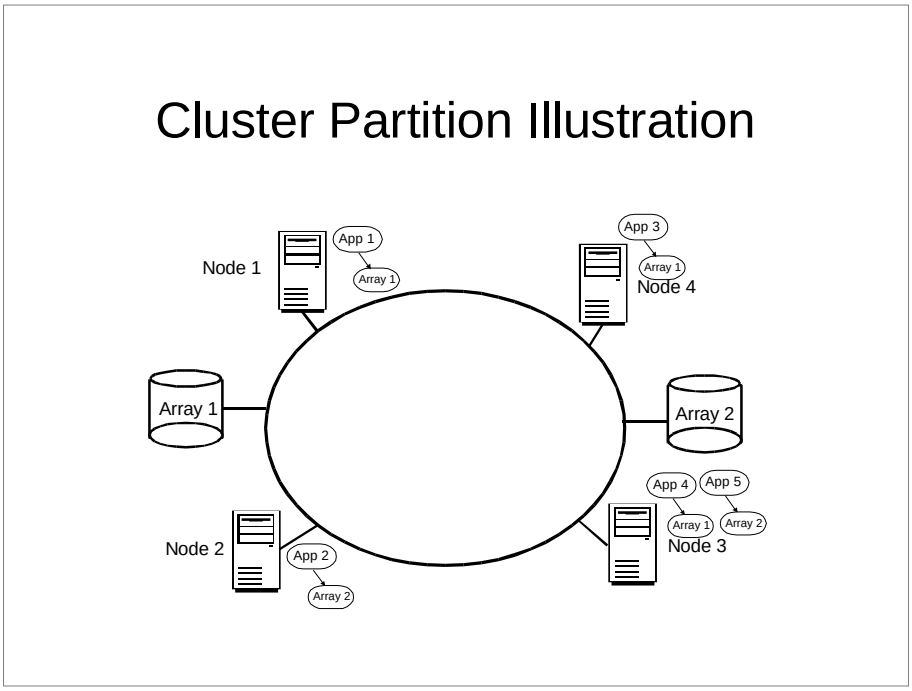
- Partition model is Resource Driven
- No Need for Quorum: Disc resource is arbitrator
  - Only resource with access to disc may recover
  - Partitoned cluster is fine (even desirable)
  - Problems when comms lost but all nodes see storage (solve by target mode or IP over FC)

In the VAX clusters model, each node has a vote, a cluster may not form without more than 50% of the available votes. This model is wasteful (in a 4 node single vote cluster, at least 3 nodes must be up to satisfy the condition). May mitigate this by varying the number of votes given to individual nodes or by using additional tie breakers (the quorum disc).

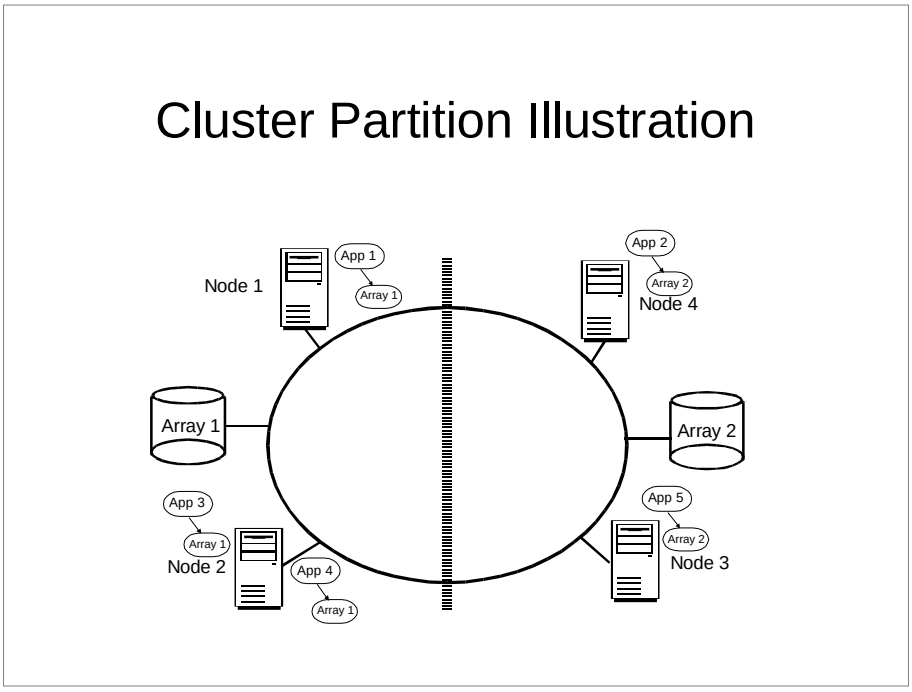
Shared storace cluster is resource driven: Any node is eligible to bring any hierarchy into service providing it can ensure exclusive access to all the resources.

The shared storage resource is usually the gate for this: Unless a node has access to the disc over the SAN ring, it cannot bring the resource into service and is not therefore eligible to recover the hierarchy. Therefore, only nodes capable of recovering the resource participate in the cluster.

Partitioning is legitimat: may occur with two or more arrays: if partition of SAN splits the arrays, one side will recover the hierarchies on array 1 and the other side will reccovery hierarchies on array 2. This is a fully functional and legitimate partitioned cluster.



Five applications active on four nodes. Each application is attached either to array 1 or to array 2.



Five applications are still active on four nodes but now the applications have rearranged themselves to take the accessibility of the arrays into account.

Recoverability in this situation is governed by the hierarchy composition−−−could easily have created an unrecoverable hierarchy (one that needed both array 1 and array 2 for instance).

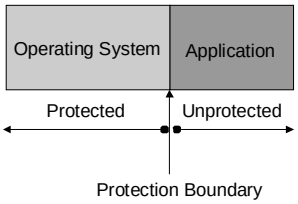
Can also encode requirements rules for other resources (e.g. IP address must be able to ping the WAN router).

Protecting Applications

- Assume application can recover from crash
  - Provide resources and application will recover itself correctly
- Enhance protection by monitoring
  - Monitor based on services performed by application, select a given table from a database, request a known page from a webserver.
- Try to move protection boundary into application

The protection boundary is the point beyond which the cluster software (and the application harness) cannot go. Therefore beyond this point it must be left to the application to "do the correct thing".

Application Protection Boundary



- Try to move boundary into application by monitoring
- Can only go so far without application support

The diagram shows the protection boundary at the application boundary. This is the case for hardware based high availability which has no application specific components at all.

Software based HA can tailor monitors specifically to the application (necessitates a protection harness for each application) which moves the protection boundary some way into the application. However, although better it too can only go so far in monitoring

Case Study: London Stock Exchange Failure

- On 5 April 2000, The LSE computer system failed and was taken down for a reboot
  - Down time was 8 hours, finally up at 3:45pm
  - This is the end of the tax year in Britain
  - Actual direct costs (traders fees, savings tax, transaction fees) estimated in the millions
  - Indirect cost (credibility, individual losses) almost unmeasurable

Everyone in the HA business likely had a friend from the UK comment that "the LSE could have done with your clustering software". Well, let's look and see if this is correct.

Preventing the Disaster

- Could any of the following have prevented the problem from occurring
- Hardware HA? No
- Software HA? No
- Shared Storage Clusters? No

The lesson is that No failover based system could prevent the Disaster because it was caused by data contamination

Analyse What Went Wrong

- At 4:30am a database purge program got bogged down and was still running when trading began
- Start of day assumed only one record per share reflecting the current price
- Contamination of the database forced the LSE to shut down.

Share prices are stored as a "time series", a sequence of time indexed records, the most current one representing the actual price. Since the purge was still running at start of day, obsolete price data was mixed with the correct ones. Without reliable data, no trading could be done and the data stream had to be shut down.

The problem was in the Application

- The database contained contaminated data
  - No amount of switching over or component protection fixes this
  - Could only have prevented this if the protection boundary was far enough into the application to detect the problem, but cannot be done without application help
- Could use Disaster Recovery techniques to return to uncontaminated data

Once the protection boundary goes beyond monitoring to pro−active failure detection, it becomes a battle of wits with the application to think of all the possible failure modes and encode checks for them. Can use computer analysis techniques on simple applications to check that all error conditions have been thought of. Analysis tends to be unwieldy for complex applications

Bottom line: person writing the checks must be intimately familiar with the application. Even so, how do you know they thought of everything.

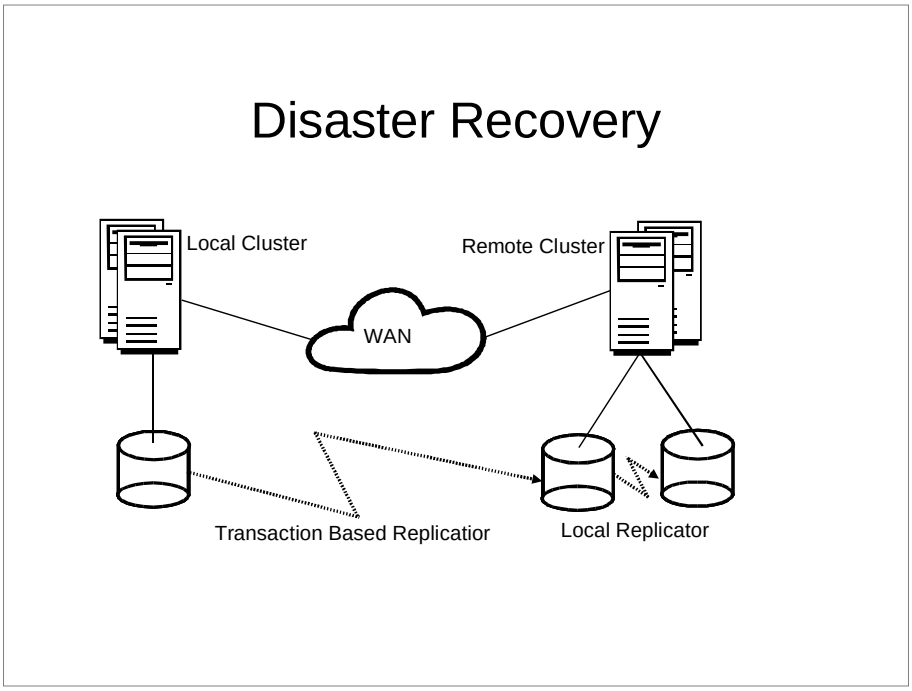
Disaster recovery is a different breed of High Availability technique. It involves restoring the application to a known good state.

Checkpoint and Restore

- Periodically dump the state of the application
- On automatic (or Manual) failure detection, if ordinary recovery fails, restore from checkpoint
- Restore procedure involves data loss
- Expensive on storage
- Would have limited LSE down time

Checkpoint and restore is simple, can be done with or without application co−operation. The down side is that you need at least three times the storage the application requires (you need at least two rotating checkpoints in case disaster strikes while a checkpoint is being taken).

Note that checkpointing still cannot guard fully against application error: if the application itself caused the error (by internal bug, design fault, etc.) unless the casue is determined and circumvented, the problem will likely recur in the same circumstances even when restored from the checkpoint.



The remote replicator must be transaction based (i.e. transactions are send in the order they are committed to the local storage) to ensure replica integrity

Remote cluster has local replicator (may also have this in local cluster). Its job is to replicate the database to storage. The replica is then broken off and retained as the checkpoint to be wheeled out again if the application needs to be restored to a known state.

Disaster Recovery

- Based on Local and Remote Replicators
  - Local replicator can be software RAID 1 (mirror)
  - Remote replicator comes from e.g. Linux−HA
- Cannot use shared storage (2km limitation)
- Use hybrid approach
  - Shared storage for local cluster
  - Linux−HA for geographic cluster

Remote replicator usually Asynchronous (WAN latency is too high for synchronous mode) => a certain number of transactions are committed locally but in flight to the remote. On a crash, these in−flight transactions are lost.

Local replicator usually synchronous (Bus bandwidth is high enough for this). A good local replicator can have the mirror broken for backup purposes and bring in either a fresh volume for complete reply or a previous backup volume for incremental replay

Because of geographic distances, even a SAN won't reach => cannot use resource as tie breaker, must use other techniques e.g. VAX quorum.

Conclusions

- Shared storage solves a lot of cluster problems (quorum, partitioning, replication delays)
- Shared storage introduces others (2km length, configuration)
- Failover clusters have inherent weaknesses in the protection boundary
- No one technique solves all problems, must combine from each field