Achieving High Availability and Data Protection with Linux

James Bottomley

SteelEye Technology

22 March 2007

What Is Availability?

- Availability measures the ability of a given service to operate.
  - Defined as the fraction of time for which the service you are exporting is available for use.
  - So a service with 99.99% Availability must be down for no more than 52 minutes per year.
  - 99.999% is no more than 5 minutes and 15 seconds per year.
- Usually, in any system, availability decreases as the complexity increases.
- Any system which takes action to increase availability beyond what would ordinarily be possible may be termed Highly Available.

Class of Nines

- When the availability is shown as a percentage (or just as a decimal), the number of initial nines is called the availability class of the service
- Thus, $A = 0.9999$ or 99.99% is an availability class of four (or four nines)
- $A = 0.99999$ or 99.999% is an availability class of five (or five nines).
- and so on.

Determining Availability

- This is actually one of the really hard things to do.
- Uptime U is defined as the average time to a failure
- Downtime D is defined as the time between experiencing the failure and getting the system working again.
- Obviously, the Availability A becomes

$$A = \frac{U}{U + D}$$

- But in order for this to be meaningful, you need to know what U and D are in your environment

Finding the Availability of Your Service

- Knowing what your Availability is and comparing this against your service level requirements can be critical to evaluating your need for High Availability.
- So how do you go about doing this?
- Manufacturers sometimes quote figures like
  - Mean Time To Failure (MTTF) which is exactly the same as the Uptime U.
  - and Mean Time Between Failures (MTBF) Which is $U + D$
- So we can easily plug the manufacturer supplied figures into the availability equation?

Finding the Availability of Your Service (2)

- **No**: the MTTF measures **hardware** uptime in a specific environment; it doesn't include things like
  - power failures (electric company, or even someone accidentally tripping over the server's power cord),
  - environmental damage (like the A/C unit floods the lab),
  - human error (the admin spills coffee on the server).
  - or Software failure (the web server crashes).
- So whatever your Uptime is, it will be less than the Quoted MTTF.

Finding the Availability of Your Service (3)

- Now the really big one, finding your **Downtime**:
  - For simple failures, like the database crashing, your admin can simply restart it, but how long does it take?
  - If the server has really failed, how long does it take you to find another one and get it up and running?
  - If you use a co-location service how long does it take to get someone out there?
- The Downtime is really the biggest unknown in finding your Availability.

Clustering and High Availability

- The simplest way to improve the Availability of a system is to have a duplicate waiting to take over if anything goes wrong.
- This duplication describes the simplest form of Active/Passive cluster.
- Here, the Down time of the Service is the Time it takes the Passive node to detect the failure plus the time it takes to recover the service.
- This is often termed the "Availability Equation" (but more accurately, it is the downtime equation)

$$D = T_{\mathrm{detect}} + T_{\mathrm{recover}}$$

Clustering and High Availability (2)

- So, if you have a Service Level requirement, what a cluster really does for you is quantify exactly the Downtime D.
- Thus, it eliminates a huge quantity of uncertainty from your enterprise.
- However, note that implementing cluster still **doesn't** give you any handle at all on your Uptime U.
- Therefore, you still cannot predict your Availability, even with a cluster, unless you know your Uptime.
  - All you've done is controlled your Downtime.

Clustering and High Availability (3)

- The reason clustering implementation is so important is precisely because the cluster cannot control Uptime.
- The only way to control uptime is by careful implementation and deployment of the cluster. This is why things like:
  - Hardware burn in,
  - Redundancy in communications and storage,
  - Multiple redundant power supplies,
  - All the traditional uptime lengtheners

  are still important in cluster deployment.

Case Studies: Crashing Web server

- My web server currently crashes once a week, but I'm going to implement clustering to give me a five nines (99.999%) uptime.
- No, and here's why:
  - Webster crashing once a week give at **most** an Uptime of a week.
  - Suppose the cluster can reduce the downtime to, say, ten seconds (five to detect and five to recover).
  - That gives an overall availability of

$$\frac{7 \times 24 \times 60 \times 60}{(7 \times 24 \times 60 \times 60) + 10} \approx 0.9999834$$

  - or only 99.998% uptime (four nines).

Morals from the Case Study

1. A cluster isn't a silver bullet for solving all your problems.
2. However, it can get you a significant fraction of the way there
   - Even in the example of an appalling one week uptime, if the sysadmin takes ten minutes to notice the problem and restart the server, that's an Availability of 99.90% (or three nines).
   - Implementing clustering got you a whole order of magnitude better.
3. You must clearly understand where your problems lie to understand if a cluster solution can meet your expectations.

Users and Failure Tolerance

- Sometimes, Availability is a misleading measure, and Downtime is the true quantity users care about.
  - It's all about perception.
- In the web server example: a regular user who complains to the admin once a week to get the service restarted regards the service level as unacceptable.
- However, if the cluster can restart it in ten seconds, he only has time to notice the failure and click again to get the service restored.
- A similar service glitch could be caused by the Internet or DNS resolution or a host of other problems between the user and the service, so the user will tolerate this level of downtime.

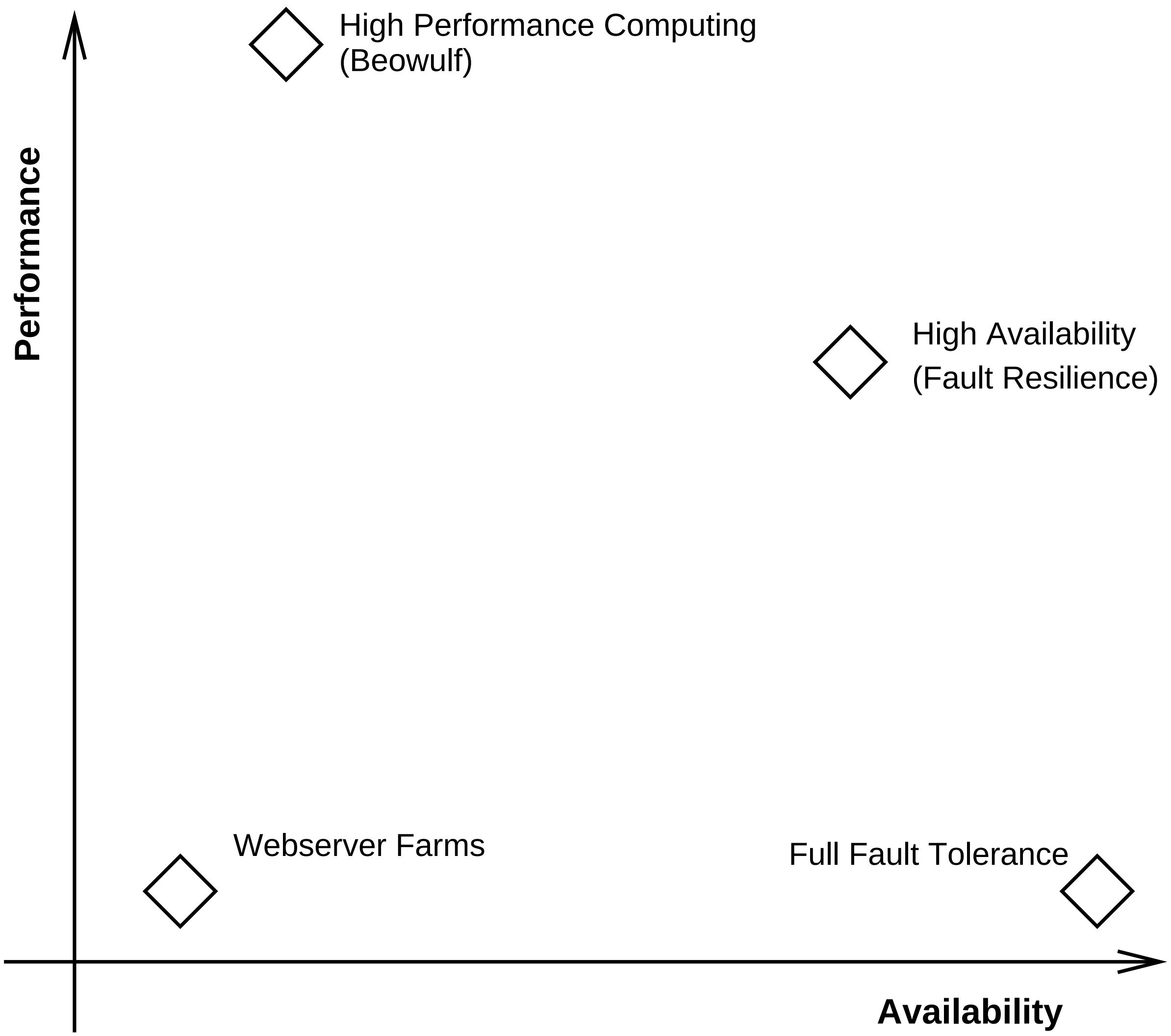
Five things to know before you start

- Do you currently have a service problem that you're trying to solve?
- What is the nature of the service you're trying to protect and how do its users react to outages?
- Do you care about absolute Availability, or is controlling the Downtime sufficient?
- If you care about availability, can you accurately measure your current uptime?
- For any given HA solution, can you estimate the maximum Downtime it will give your applications **before** you install it?

Types of Clusters

- Clustering really falls broadly into three categories
  - High Performance Computing (HPC). Cluster Many machines together to get much higher processing power than a single machine (e.g. Beowulf).
  - High Availability (HA). Cluster machines together to produce the availability of a set of services (e.g. LifeKeeper, Heartbeat, Cman, etc.)
  - Fault Tolerant (FT). Similar to HA but with much more strict availability requirements
- and one that isn't really a cluster at all, but is often regarded as one
  - Web server Farms

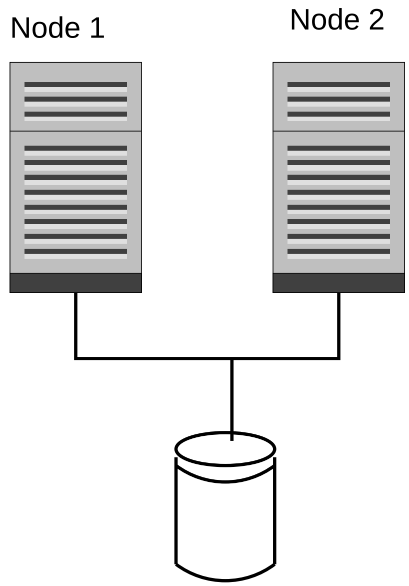
Graphing Availability versus Performance



The Difference Between Fault Tolerance and Fault Resilience

- The difference between these two terms indicates how the **users** of the service see the failure and recovery.
- **Fault Resilience**: Means that users may notice some disruption:
  - Web users may see a transient failure to find the page.
  - Database users may see a single error committing a transaction.
- **Fault Tolerance**: means that the users notice no disruption in the service at all (although they may notice a "pause" while recovery occurs).

Shared Storage Clusters



- Two or more computers connected by a shared storage bus (either SPI or FC).
- An external (and redundant) array to provide data to either node.

- Not the only model: could also provide data availability via replication

Clusters and Fault Tolerance

- There's no clustering technology available in the market today that provides full fault tolerance for every service.
- Originally, full fault tolerance was achieved
  - by running multiple copies of the application in a cluster
  - splitting inputs to go to each application
  - comparing outputs to check for errors (should be identical)
  - if one server fails, the other instances keep the service running exactly until there are none left.
- However, even in today's clusters, some (but not all) services are still fault tolerant.

High Availability and Failover Clusters

- Most clusters on the market today are failover.
- This means that they require **no** changes to the application to permit failover.
- Instead, they throw a harness around the application to control restarting it on another server.
- This approach relies on the application being coded in a crash resilient fashion, by:
  - not keeping critical data in memory,
  - placing all non-volatile data on permanent storage in an atomic (or recoverable) fashion,
  - ensuring that data is actually on permanent storage before signalling completions back to the user.

Fault Tolerance and Failover Clusters

- By and large, if the cluster relies on the crash resilience of the application, it is not going to be fault tolerant
  - primarily because uncommitted (and unacknowledged) data may be lost requiring the user to retransmit it.
  - for example a row insert in a database table may be returned with an error indicating the client should try again.
  - sometimes the user's client will do the retry without showing the failure to the user, giving the perception of fault tolerance.
  - but most of the time, the user's application will have to be coded to do the retry itself.

Fault Tolerance and Failover Clusters (2)

- Some services which are truly **stateless** may be recovered in a fault tolerant manner even in failover clusters.
- The classic example is NFS
  - designed by Sun with true statelessness built into the protocol
  - no persistent information about the client is retained in the server process.
  - If information (for example file writes) is lost, the NFS client automatically retries.
- Of course, extra stateful protocols (like locking) are layered on top of NFS which do cause fault tolerance problems.

Understanding the Consequences of Fault Resilience

- First and foremost, are the problems it may cause on a failover important to you (i.e. what will your users see in the face of a failure)
  - For web servers, usually the answer is “no”
  - Users are trained to press “reload” if the web page times out or comes back with a web server error.
  - A web page that shows an internal error from the database may be less welcome.
- Secondly, is there anything you can do in the way the client application works to improve the user experience?
  - Retry transient transaction errors automatically, for instance.

Important points

- Your need to know if the service you are protecting is fault tolerant or fault resilient.
- If it is fault resilient, you need to understand:
  - the impact of the failover to the service being exported from the cluster
  - The visibility of this impact to the external users of the service.
- If you also control the client side of the service, you should plan your implementation to take into account service anomalies caused by failover.

Monitoring

- Every cluster (without exception) provides the ability to monitor health at a node level.
  - so node failures may be spotted and corrected.
- some clusters also provide the ability to monitor individual applications and even restart them locally if they have failed.
  - this is essential, because applications can fail more often than the node (e.g. the web server crashes every week example)
  - Local recovery is important (because it can decrease downtime and minimise disruption).

Customising your Cluster Environment

- Cluster vendors try to provide off the shelf recovery tools for typical applications
  - web servers, databases, file exports (NFS or SMB/CIFS) etc.
- However, in a complex environment you often have custom applications that the cluster vendor won't support out of the box.
- In this case you need to know what options are available to you to support your application
  - does the cluster provide an easy way to protect and monitor arbitrary applications?
  - Does this come as an extra, or is it available with the base product.

Looking Beyond the Software

- As we learnt previously: a cluster helps you minimise Downtime. It cannot help you with uptime.
- However, uptime is extremely important to availability.
- Thus, as well as implementing clustering to improve Downtime, you should assess your cluster hardware for ways to improve uptime.
- Key to this is eliminating Single Points of Failure (SPOF).
  - Cluster wide SPOFs must be eliminated entirely
  - Individual Node SPOFs should be assessed to see if eliminating them would improve uptime.

Cluster Single Points of Failure

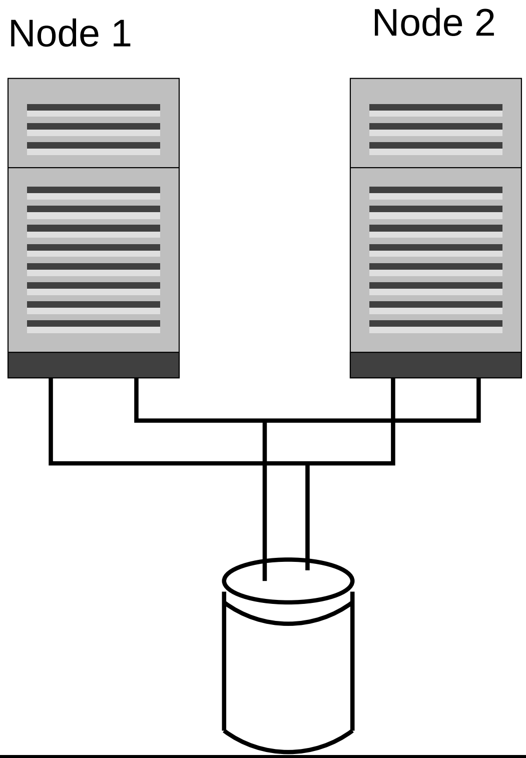
- In a shared storage cluster, the real SPOF is the storage.
- Make sure that the external array is configured as a RAID
- Not only that, but make sure it's RAID 1 (mirroring)
  - RAID 5 is cheaper, but in a double fault situation it may end taking the array offline **and corrupting your data**.
  - RAID 1 preserves data integrity (but still takes the array offline) in the double fault case.
- replication provides a cheap method of eliminating the storage SPOF (separate copies of the data in each node).

Node Single Points Of Failure

- In the shared storage cluster, the first and most obvious Node SPOF is the connection to storage.
- The next most common is the power supply (the non-silicon components often burn out or fail).
- Almost equally common is the failure of mechanical devices like Fans
  - Particularly nasty in todays world of hotter, faster and actively cooled CPUs
  - For example, a top of the line P4 will overheat and burn out in less than a second if its heat-sink fan fails.

Multi Path

- In the standard shared storage cluster, if a link to storage fails, the application loses contact with the data and the cluster must fail to another node that can still access it.



- This happens surprisingly often (cables get trodden on, dust gets into transceivers etc.)
- Can obviously eliminate this by having more than one connection to the storage per node (called Multi path).

The Costs

- Replication is essentially free.
- External Storage arrays cost about $3,000+ (FC arrays begin at about $5,000)
- Multi Path, starts at about $10,000 and the sky is the limit for truly Rolls-Royce solutions.
- Redundant Power Supplies and Redundant Fans only found in higher end servers (not as add on items to low cost servers), will drive server costs up by $3-5,000.

Managing your Systems

- Systems management is integral to SPOF elimination
- It is no use at all to buy a fully redundant system and then keep it in a cupboard and never monitor it.
  - redundancy will protect you when the first failure occurs.
  - the second failure will take down your server (or cluster if it's in the shared array).
  - maintaining and replacing failed redundant components is essential to preserving uptime.
- if you have no way of monitoring your server's redundant components, you may just as well opt for cheaper hardware and allow the cluster to manage the Downtime instead.

Choosing your Cluster

- Once you understand what you're trying to achieve (and what you might need to modify or alter to achieve it)
- and you have identified either your availability or Downtime goals
- you are ready to begin selecting an HA cluster product.
- Ideally, if you haven't already purchased your hardware, the cluster vendor should be able to guide you through choosing this.
- if you do already have your hardware, make sure you validate it with the cluster vendor first.

Cluster Products For Linux

- LifeKeeper (`http://www.steeleye.com`) available on all enterprise Linux distributions, no proprietary modules, Resource Driven, Cross Platform.
- heartbeat (`http://www.linux-ha.org`) now shipped with SLES10, Quorate.
- and FailSafe (was shipped with SLES, now defunct)
- Mission Critical (Shipped with RHEL3 and below, now defunct), Quorate
- Cman (Shipped with RHEL 4)
- Veritas Cluster Server (requires proprietary modules, limited distro support), Resource Driven

Replication

- Clustering provides application Availablility
- Replication provides true distributed Data availability
- Replication maintains complete and up to date copies of your data in multiple locations
  - If these locations are separated by several miles (or even several thousand miles) this gives true disaster recovery.
- Replication is not true Continuous Data Protection

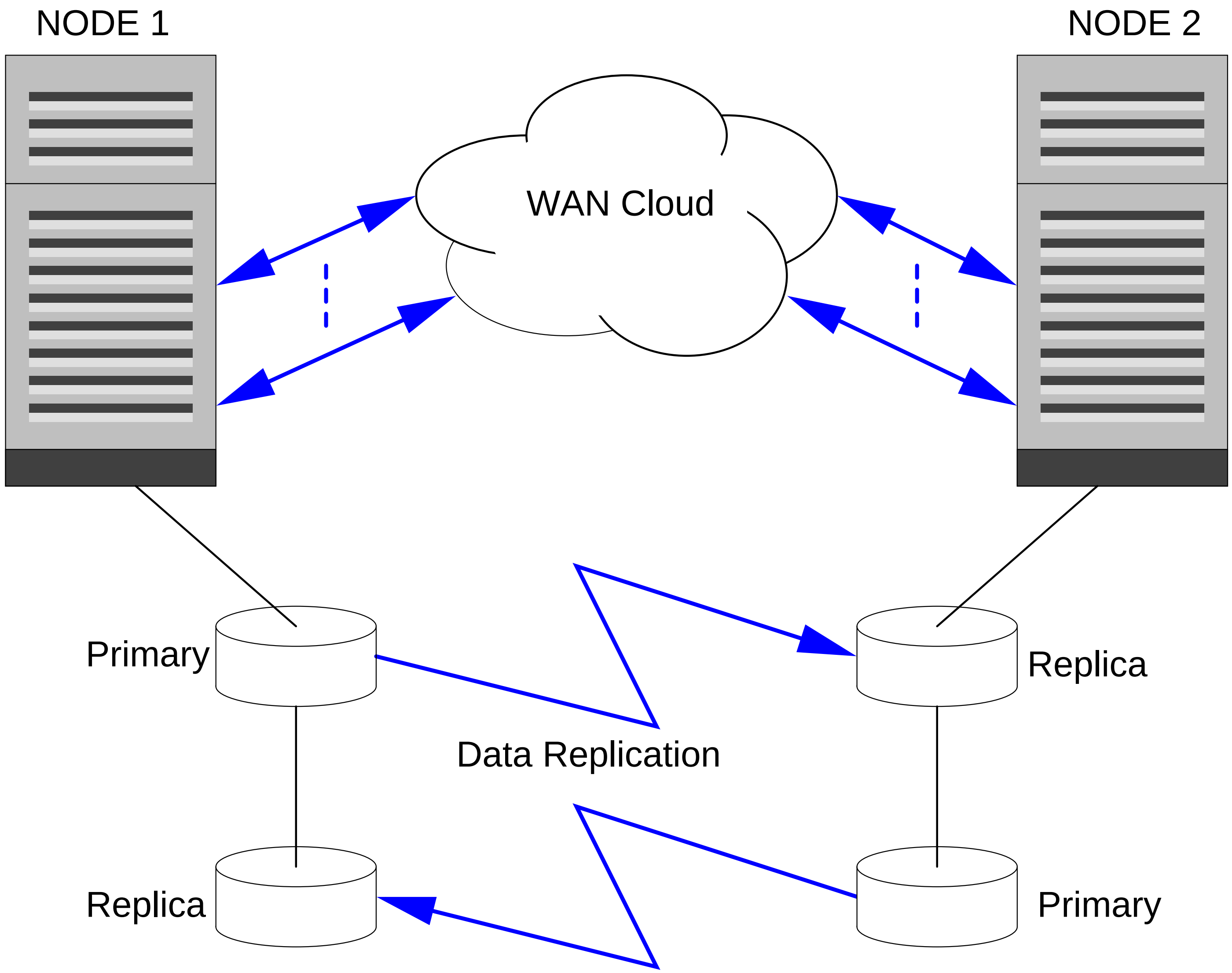
Replication Products

- SteelEye Data Replication: Based on md/nbd, no out of kernel modules, available in All distributions with recent (2.6.14+) kernels (SLES10, RHEL5), intent log based.
- drbd: requires out of kernel, but open source module, intent log based
- Veritas SRL: requires proprietary kernel module, relatively limited kernel support, transaction log based.

How does Replication Work

- Replication can operate at three levels
  - Application: SQL Replicatiors like that in MySQL
  - Filesystem: No Linux Examples (too many filesystems), Windows Replicators like XOSoft.
  - Block level: `drbd`; `md/nbd`; Veritas SRL.
- Block level replication is the most generally useful
- Application level replication can be the most efficient
  - because it understands the layout and meaning of the data
  - However, need separate replicator for **every** application!

The Basics of Replication



Replication Features

- Logs
  - Transaction: Records both Data and Position information
  - Intent: Merely records unsynchronised blocks
- Synchronous replication
  - Write is not reported as committed to the application until the data is on both primary and replica
  - round trip latencies make this useful only in a LAN environment

Replication Features (2)

- Asynchronous replication
  - Write is reported as committed when data is safely on Primary Media only—Replica data may still be in-flight
  - Logging is a requirement to make asynchronous replication crash proof
  - transaction integrity is preserved by correct ordering of the replica writes.
  - Most useful feature is ability to free replication from latency concerns and the ability to fully utilise the available bandwidth of the network
  - Ideally suited to WAN replication over large distances.