SQL Anywhere High Availability

Jason Hinsperger
Product Manager
hinsperg@ianywhere.com

TECHWAVE 2007
SYBASE USER TRAINING & SOLUTIONS CONFERENCE

August 6–10 | Mandalay Bay | Las Vegas, Nevada

Agenda

High Availability Background Information

Clustering

- SQL Anywhere Cluster Support

Database Mirroring

- What is Database Mirroring
- How mirroring works in SQL Anywhere 10
- An example of mirroring

High Availability Goals

- Who needs HA and why?
- How much does down time cost?
- Is recovery speed or data state more important?
- What will the maintenance schedule look like?

HA IS NOT A REPLACEMENT FOR A GOOD BACKUP/RECOVERY PLAN!

High Availability Options

- **Hardware**
  - Several physical machines (nodes), appearing as one logical entity to the consumer
  - When one of the nodes fails, the others pick up its services so they continue to be available to the consumer
- **Software**
  - Application made available to the consumer where location is abstracted
  - If the application the consumer is using fails, they are automatically pointed to another instance of the application
- **Procedures**
- **Policies**

Agenda

High Availability Background Information

Clustering

- SQL Anywhere Cluster Support

Database Mirroring

- What is Database Mirroring
- How mirroring works in SQL Anywhere 10
- An example of mirroring

High Availability Options - Clustering

- High Availability Cluster
  - Primary purpose is to improve availability of the services it provides
- Load Balancing Cluster (server farm)
  - Primarily used to improve performance, but also provides HA features
  - Workload comes through a frontend, which distributes it to a set of backend servers
- **SQL Anywhere provides only HA in a cluster**

Clustering Definitions

- **Resource**
  - Element managed by cluster service (eg. hard disk, application, ip address)
- **Resource Groups**
  - Two or more resources that are managed as a single unit
- **Shared Storage**
  - Some sort of disk used to provide quorum information and share data among the nodes in a cluster

Clustering Definitions

- **Failover**
  - When a resource (or resource group) fails, resulting in a loss of service, that resource (or resource group) is restarted on another component in the system
- **Failback**
  - When a node in a cluster comes back on-line, this is the act of the cluster server restarting a resource (or resource group) on that node

Clustering – Operating options

Active/Passive

- One node provides services, another node is idle and takes over if the first node fails

Active/Active

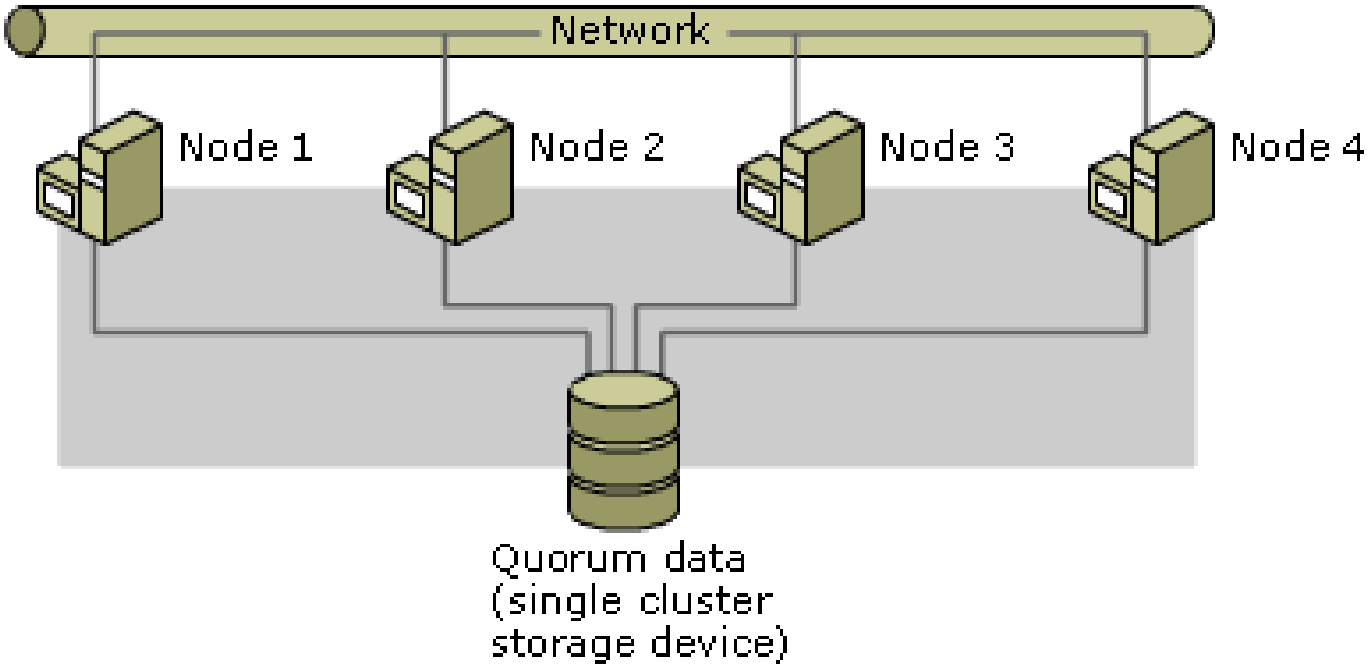
- Both nodes provide service, and if one fails, the other must take over the failed services

Cluster Hardware Requirements

- At least 2 host machines (nodes)
- Interconnection of nodes
  - Serial crossover cable
  - Ethernet crossover cable
- Shared Storage
  - SAN, NAS, iSCSI, RAID Arrays
  - Must be accessible by all nodes

Cluster Configuration

- Single quorum, shared disk cluster



Clustering With SQL Anywhere

All versions of SQL Anywhere can be set up to run in most HA cluster environments

- From SQL Anywhere 5.5 up to SQL Anywhere 10.0

No custom programming is required for this

Most cluster management software has a method for making a regular application clusterable.

Generic Cluster Server Agents - MSCS

- Create a local service (dbsvc.exe utility)
- Run the Cluster administrator and add a new “Generic Service” resource
- Resource should be dependant on
  - Cluster IP
  - Cluster Name
  - Shared Storage – SA database is stored here
- Bring service online
- Detailed instructions available online at:
  - http://www.ianywhere.com/developer/technotes/asa_cluster_db_service.html

SA Cluster - Limitations

Less control over how failover occurs

- If shutdown takes too long, service may be killed by cluster software, requiring recovery when failover occurs
- Only one copy of db file on shared disk
  - RAID 1 or RAID 5 maintains redundant storage, but in same location
- Cannot disperse nodes geographically without third party hardware/software and increased complexity

SA Veritas Cluster Server Agents

Same steps as with MSCS, the only difference is the cluster management tool

- Create database service/daemon on each node
- Use cluster manager to create a ‘generic service’ resource
  - Set up dependencies Bring service online

OR

SQL Anywhere 10.0 Includes Custom Resource Types for Veritas Cluster Server

SA Veritas Cluster Server Agents

Two Separate Agents

- **SAServer Agent** – Can setup, monitor and control a server process in the cluster
  - Provide server start, monitor and stop commands

- **SADatabase Agent** – Can setup, monitor and control a specific database running on a server in the cluster
  - Provide database file, database name, server name and utility database password
    - Can provide a different server name on each node in the cluster if desired

Agenda

High Availability Background Information

Clustering

- SQL Anywhere Cluster Support

Database Mirroring

- What is Database Mirroring
- How mirroring works in SQL Anywhere 10
- An example of mirroring

Database Mirroring

Using two or more (up to three for SA) servers to increase availability of the database

- Consists of
  - Primary server
  - Mirror server
  - Arbiter server

Clients only see (and connect to) 1 server

Database Mirroring – Definitions

Primary Server

- Current active server

Mirror Server

- Current standby server

Arbiter Server

- Determines who is the primary server

Quorum

- For a server to become the primary server it must have quorum – it and one other server must agree that it is (or should become) the primary

Database Mirroring - Benefits

- **When an arbiter is present, failover is automatic**
- **No transactions lost if running in synchronous mode**
- **Failover is fast – log has already been applied**
- **No special hardware requirements**
- **No special software requirements**
- **Operational servers can be geographically diverse**
- **Servers can run mirrored and non-mirrored databases**

Database Mirroring - Operation

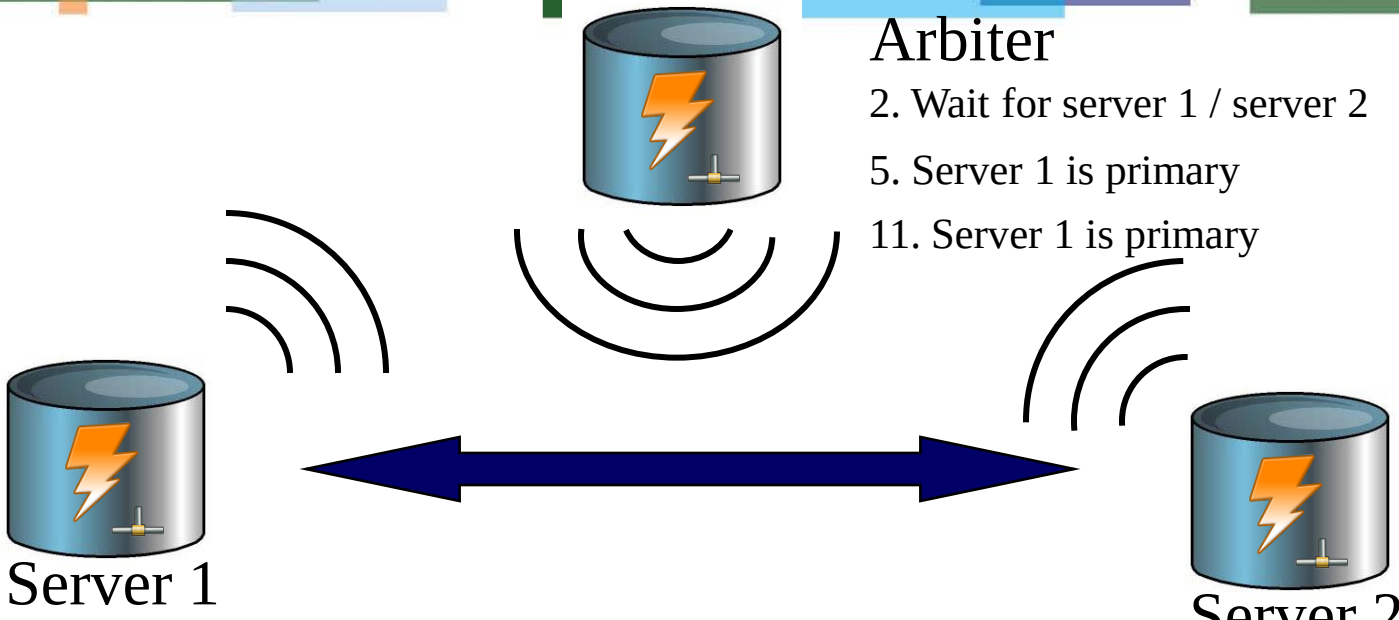
Server startup

- Examine and, if required, apply any logs in the same directory as the current log file
- Determine whether or not to become the primary server

Mirror Server

- In a constant state of recovery – applies log pages as it receives them from the primary server

Mirroring Startup



Arbiter

2. Wait for server 1 / server 2

5. Server 1 is primary

11. Server 1 is primary

Server 1

1. Look for arbiter/server 2

3. Connect to arbiter

4. Can I become primary?

6. Yes – Begin accepting connections

13. Start sending changes to server 2

Server 2

7. Look for arbiter/server 1

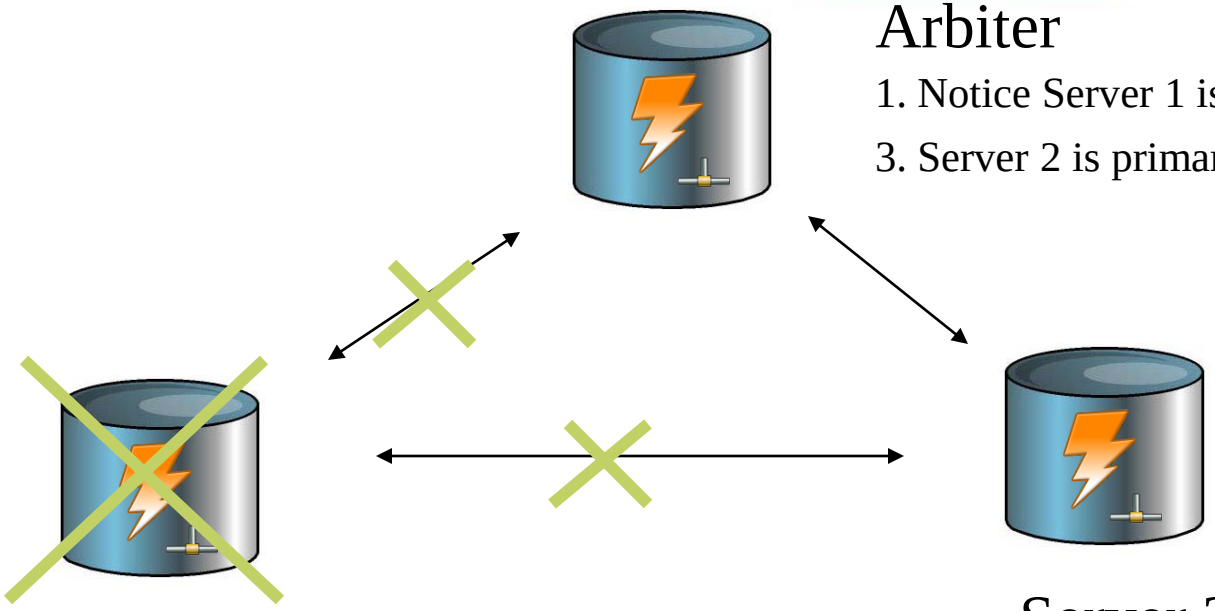
8. Connect to arbiter

9. Connect to server 1

10. Can I become primary?

12. No – Apply changes from server 1 and Standby

Failover Scenario 1 – Loss of Server 1



Arbiter

1. Notice Server 1 is gone – it is no longer primary
3. Server 2 is primary

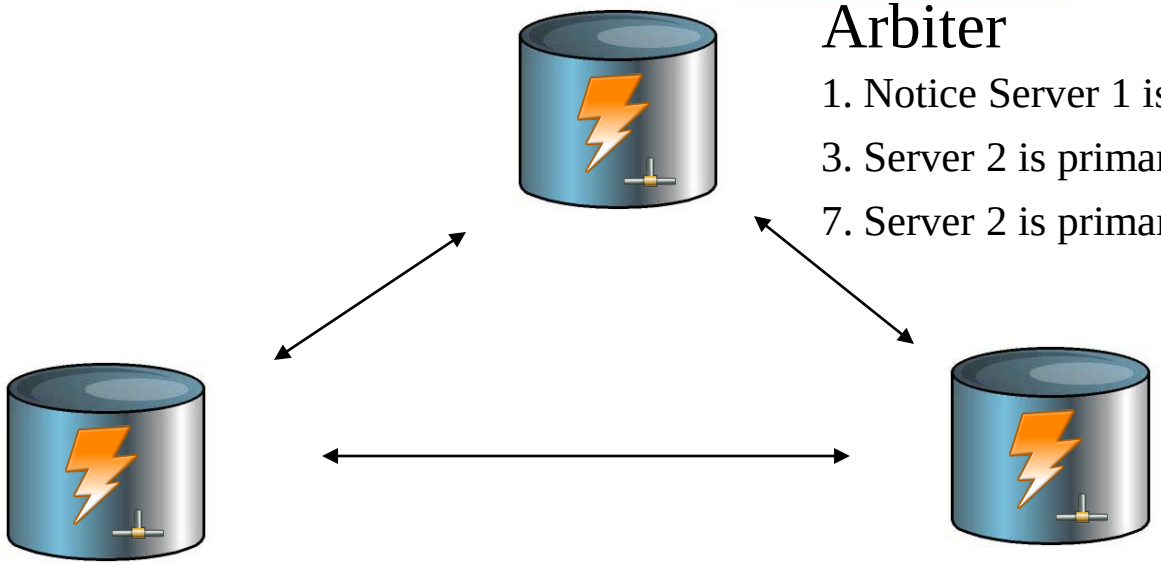
Server 1

Server 2

1. Notice Server 1 is gone
2. Can I become primary?
4. Yes – Begin accepting connections

Server 2 becomes primary server

Failover Scenario 1 – Server 1 Restarts



Arbiter

1. Notice Server 1 is gone – it is no longer primary
3. Server 2 is primary
7. Server 2 is primary

Server 1

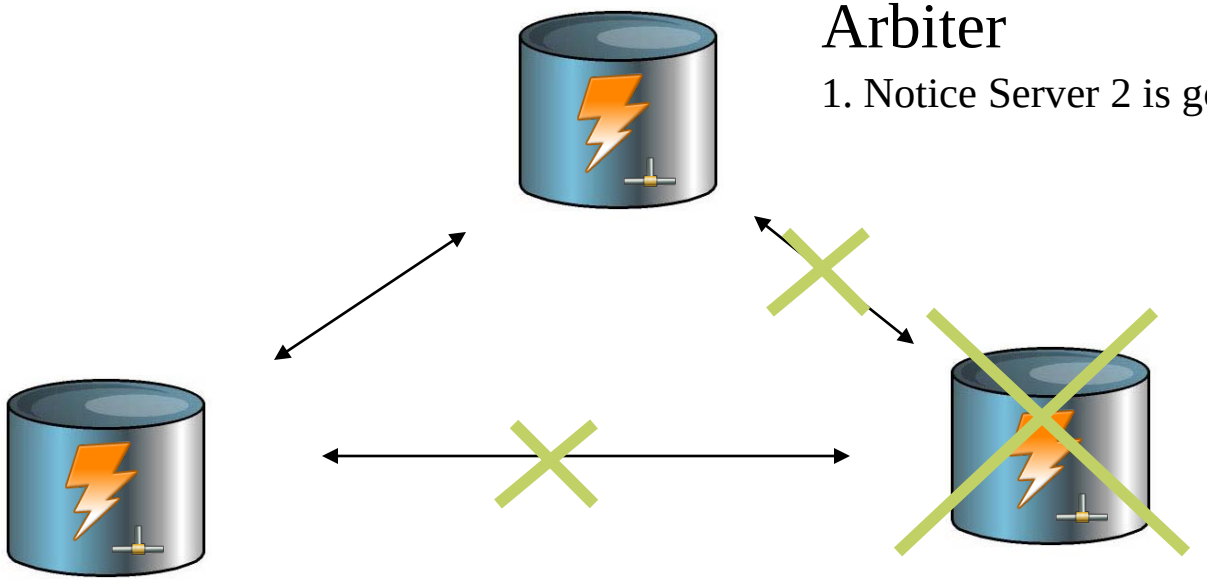
5. Server 1 comes back online
   (reconnects to arbiter and server 2)
6. Can I become primary
8. No – Apply changes from server 2 and Standby

Server 2

1. Notice Server 1 is gone
2. Can I become primary?
4. Yes – Begin accepting connections
9. Start sending changes to server 1

Server 2 remains active primary server

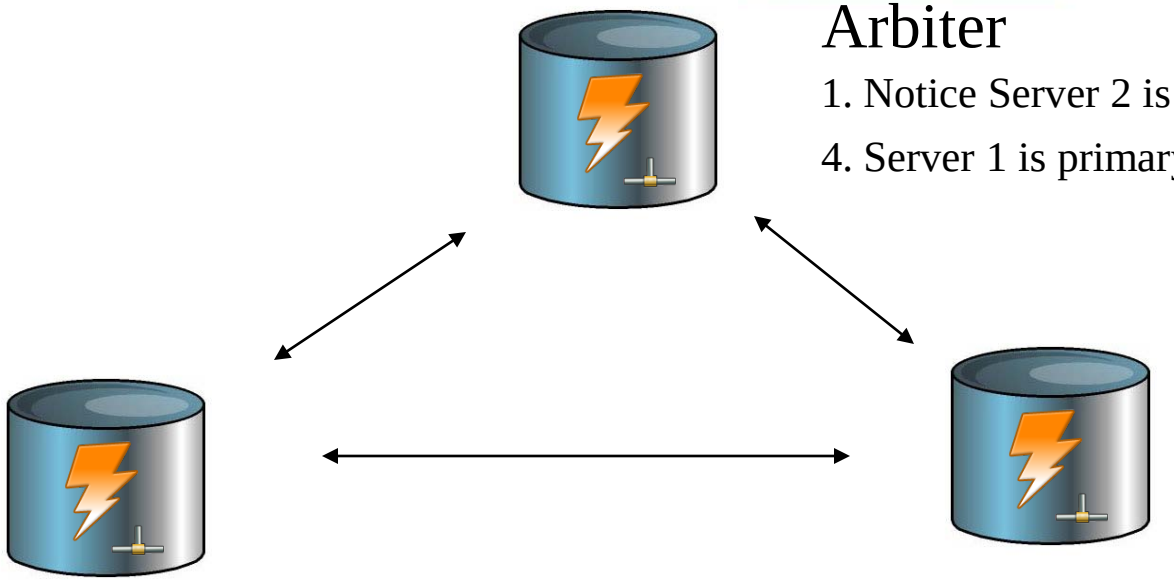
Failover Scenario 2 – Loss of Server 2



1. Notice Server 2 is gone, stop sending changes

No change in availability

Failover Scenario 2 – Server 2 Restarts



Arbiter

1. Notice Server 2 is gone
4. Server 1 is primary

Server 1

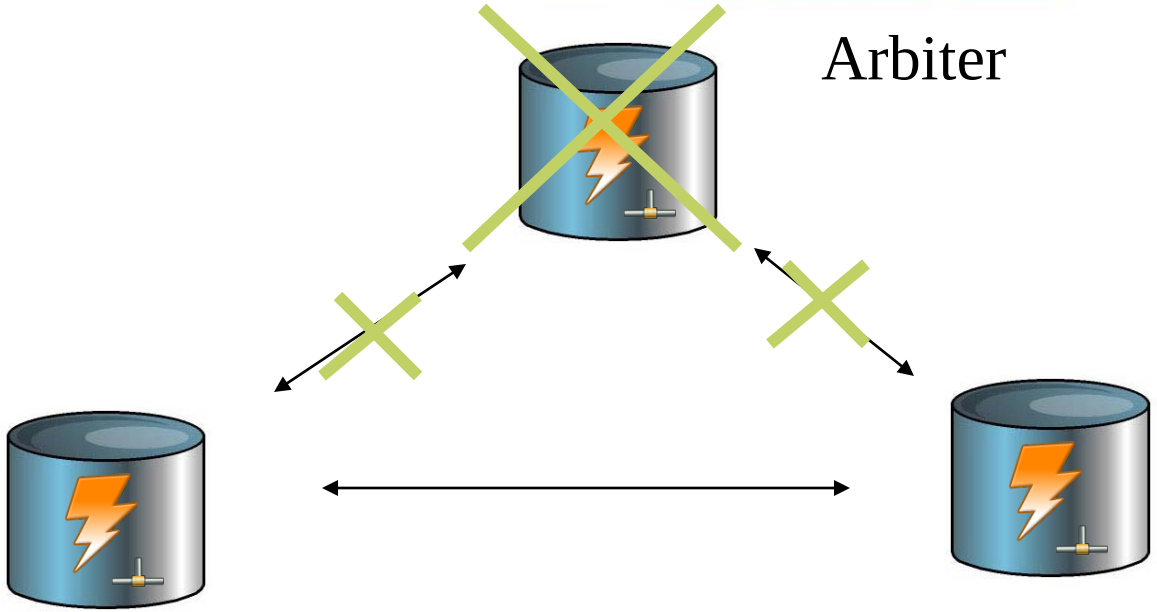
1. Notice Server 2 is gone, stop sending changes
6. Resume sending changes to server 2

Server 2

2. Server 2 comes back online (reconnects to arbiter and server 1)
3. Can I become primary
5. No – Apply changes from server 1 and Standby

No change in availability

Failover Scenario 3 – Loss of Arbiter

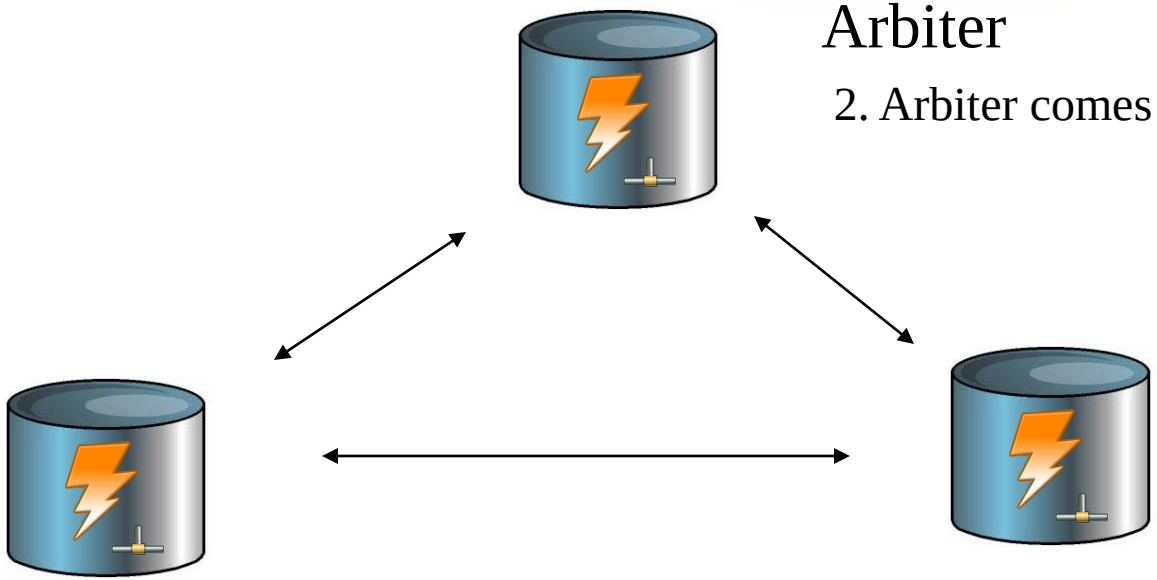


1. Notice Arbiter is gone

1. Notice Arbiter is gone

No change in availability

Failover Scenario 3 – Arbiter restarts



Arbiter

2. Arbiter comes back online

Server 1

1. Notice Arbiter is gone

3. Reconnect to Arbiter

Server 2

1. Notice Arbiter is gone

3. Reconnect to Arbiter

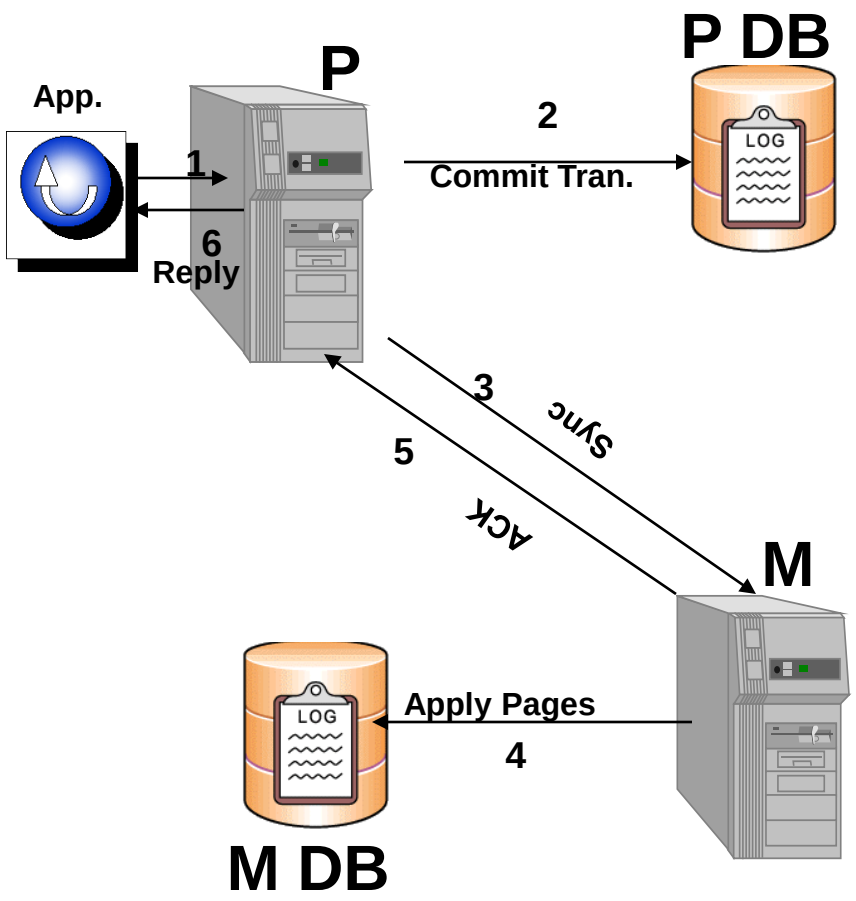
No change in availability

Database Mirroring – Modes

Synchronous mode

- Changes committed on the primary server are sent to the mirror and must be acknowledged before the primary responds to the client
- Provides **transaction safety**
- Potential lag at commit time if transactions are large

Synchronous Mode Architecture



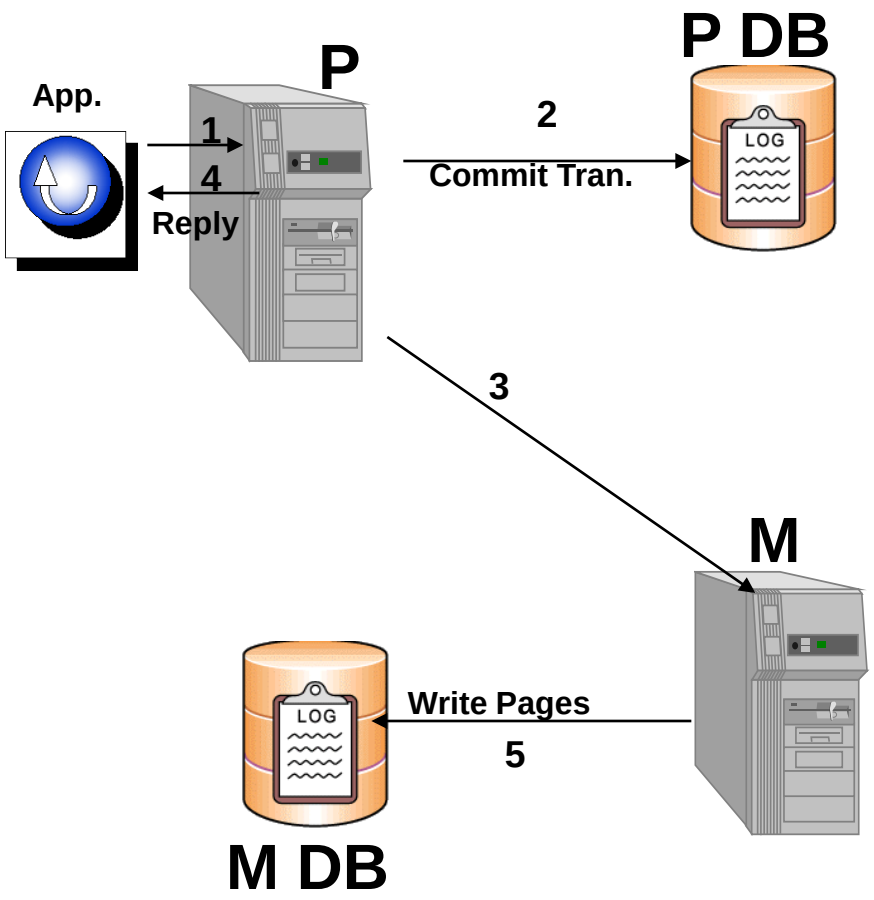
1. Application connects and executes transactions.
2. Application commits transactions on primary, updating transaction log.
3. Primary sends the new log pages to the mirror server.
4. Mirror server applies the new log pages to the mirror database and transaction log.
5. Mirror server sends acknowledgement.
6. Primary responds to the application.

Database Mirroring – Modes

Asynchronous mode

- Changes committed to the primary server are sent to the mirror, but no acknowledgement is required before responding to client
- Provides **higher performance**
- Potential loss of transactions if failover occurs before mirror receives/applies changes sent from primary
- By default, failover is not automatic
  - "Autofailover = yes" option can be set to force mirror to come up automatically if primary goes down

Asynchronous Mode Architecture



1. **Application connects and executes transactions**
2. **Application commits transactions on primary, updating transaction log.**
3. **Primary sends the new log pages to the mirror server.**
4. **Primary responds to the application.**
5. **Mirror server applies the new log pages to the mirror database/log when it receives them.**

Database Mirroring – Modes

Page

- Asynchronous mode with some improvements
- Log pages are sent when they are full, instead of on commit
  - Reduces traffic spikes and improves performance of primary
- “pagetimeout” parameter can be used to define how long to wait before sending a page that is not full
  - Reduces chances that a committed transaction could be lost if primary goes down
- “Autofailover = yes” option required for automatic failover
- Synchronize_mirror_on_commit option
  - Can be set for a transaction to force synchronous mode for that transaction

Database Mirroring - Setup

1. Stop database server
2. Copy db and log file to mirror server
3. Update command line options for primary and mirror servers
4. Build command line for arbiter server
5. Start the servers (order doesn't matter)

Database Mirroring – Setup

Server command line options

- -xp

  partner={partner_conn}; - connection string for partner server

  arbiter={arbiter_conn}; - connection string for arbiter server

  mode=[sync/async/page]; - mirroring mode

  auth=auth_str; – authentication string used by arbiter

  autofailover=[yes/no] – should mirror come up automatically if primary goes down

  pagetimeout=n – max. time to wait before shipping log page to mirror

- -xf <state file>

  Specify location of file for maintaining state information

Database Mirroring – Setup

Server command line options (cont'd)

- -xa auth=<auth_strings>;dbn=<database_names>
  - Used only by arbiter and lists auth strings and db names used in the mirror setup
- -sn <database_name>
  - Specifies an alternate server name for a single database
  - The alternate server name allows the server to listen for connections to a specific database/servername combination

Client connection parameters

- None – use the same connect string as is used currently
  - Server name must be the one specified by the –sn switch on the server
  - If crossing subnets, specify ip addresses for both servers

Database Mirroring – Setup Example

Primary Server

- `dbsrv10.exe -c 4M -n srvpr asademo.db -x tcpip{port=2638}`

  `-xf asa_mirror.ini`

  - State file location – state files help cluster co-ordination and co-operation

  `-xp partner={ENG=srvmr;links=tcpip{port=2639;timeout=1}};`

  - Partner server connection parameters
  - use a low timeout value to reduce startup and failover time

  `arbiter={ENG=srvar;links=tcpip{port=2640;timeout=1}};`

  - Arbiter connection parameters

  `mode=sync;`

  - The mirror environment will run in synchronous mode

  `auth=abc`

  - This authentication string is used by the arbiter to validate a servers participation in the mirroring setup

  `–sn asademo`

  - All servers must have the same alternate server name

Database Mirroring – Setup Example

Mirror Server

- `dbsrv10.exe -c 4M –n srvmr asademo.db -x tcpip{port=2638}`
  `–xf asa_mirror.ini –o mirror.out`
  `-xp`
  `partner={ENG=`**`srvpr`**`;links=tcpip{port=2638;timeout=1}};arb`
  `iter={ENG=srvar;links=tcpip{port=2640;timeout=1}};mode=s`
  `ync;auth=abc`
  `–sn asademo`

Arbiter Server

- `dbsrv10 -n srvar -x tcpip(port=2640)`
  `-xa auth=`**`abc`**`;dbn=`**`asademo`**
  - Specify the authentication string for participation in the mirror environment, as well as the alternate server name of the database being mirrored

Database Mirroring – State Files

- Each server in mirror system maintains a state file

```
[asatest]
Owner=server2
State = synchronizing
Mode = asynchronous
Sequence = 7
```

- Owner
  - Which server in the system is primary
- State
  - **Synchronizing** – the mirror server is requesting/receiving log pages from the primary server
  - **Synchronized** – the server is up to date
- Mode
  - Synchronous/Asynchronous/Page
- Sequence
  - How many times has failover occurred

Database Mirroring – Administration

Server properties

- ServerName – Can be used by client to determine which server in mirror system is active

Database properties

- MirrorState – null/synchronizing/synchronized
- PartnerState – null/connected/disconnected
- ArbiterState – null/connected/disconnected

Database Mirroring - Administration

db_property	Value	Description
MirrorState	Null	If connected to a database that is not mirrored
MirrorState	synchronizing	If the mirror server is not connected or has not yet read all of the primary's log pages
MirrorState	Synchronized	If the mirror server is connected and has all changes that have been committed on the primary server.

Database Mirroring - Administration

db_property	Value	Description
PartnerState	Null	If connected to a database that is not mirrored
PartnerState	Connected	If the mirror server is connected to the primary server
PartnerState	Disconnected	If the mirror server is not connected to the primary server

Database Mirroring - Administration

db_proprety	Value	Description
ArbiterState	Null	If connected to a database that is not mirrored
ArbiterState	Connected	If the arbiter server is connected to the primary server
ArbiterState	Disconnected	If the arbiter server is not connected to the primary

Database Mirroring – Administration

ALTER DATABASE <db> FORCE START

- Force <db> to become primary server, from the mirror server
- Requires a connection to utility_db on the mirror server

ALTER DATABASE SET PARTNER FAILOVER

- Initiate failover from the primary to the mirror, from the primary server

Preferred Server

- If all servers are running, the preferred server will become primary
- -xp partner={…};auth=x;arbiter={…};**preferred=yes**

Database Mirroring – Administration

MirrorServerDisconnect Event

- Fires on primary when connection from primary to mirror/arbiter is lost
- MirrorServerName event parameter – name of server that was lost

MirrorFailover Event

- Fires when a server becomes the primary database server

Database Mirroring – Administration

- Sample mirrorserverdisconnect event

```
CREATE EVENT mirror_server_unavailable TYPE MirrorServerDisconnect
HANDLER BEGIN
   CALL xp_startmail ( mail_user ='John Doe', mail_password ='mypwd' );
   CALL xp_sendmail( recipient='DBAdmin', subject='Database failover
   occurred', "message"='The following server is unavailable in the mirroring
   system: ' || EVENT_PARAMETER( 'MirrorServerName' ) );
   CALL xp_stopmail ( );
END;
```

Database Mirroring – Administration

- Sample mirrorfailover event

```
CREATE EVENT mirror_failover TYPE MirrorFailover
HANDLER BEGIN
   CALL xp_startmail ( mail_user =‘John Doe', mail_password ='mypwd' );
   CALL xp_sendmail( recipient='DBAdmin', subject='Database failover
   occurred', "message"='The server property( ‘ServerName' ) has become the
   primary server in the mirroring system: ' || );
   CALL xp_stopmail ( );
END;
```

Database Mirroring – Client Side

Same connection string

- Same server name
- May have to specify all hosts ip information and port ranges
  - eg. Uid=dba;pwd=sql;eng=altsrvrname;**links=tcpip{ip1;ip2;...}**

Application Considerations

- Client disconnected on failover – application must reconnect, and client must resubmit any uncommitted transaction
- If running in an asynchronous mode, develop a procedure to deal with potentially lost transactions
- Try to keep transactions short
- Watch out for commits in stored procedures (including ddl), except at the end – applicable in non-mirrored environments as well

Database Mirroring - Restrictions

- Must use network server (dbsrv)
- LOAD TABLE is not permitted
- Must connect to the utility_db in order to stop the server
- Only TCP/IP connections between the mirror servers is permitted
- Cannot use http server support (ip address of server changes on failover)

Database Mirroring - Restrictions

- Scheduled Events
  - Scheduled events will run on the mirror if failover completes before the scheduled start time of the event
- Cannot truncate log on backup
  - Mirror server may not be present
  - Must use rename, and delete logs later on primary
    - Eg. Using a scheduled event
  - Primary lets mirror know when rename occurs and mirror renames as well
    - When rename occurs, mirror is notified of oldest log on primary and deletes any log files it has that are older

Database Mirroring – Backup Sample

```
CREATE SERVER backup_tree CLASS 'directory‘ USING 'root=c:\backup';
CREATE EXTERNLOGIN DBA TO backup_tree;
CREATE EXISTING TABLE backup_files AT 'backup_tree;;;.'


CREATE EVENT cleanuplogs SCHEDULE START TIME '12:00am' EVERY 3 HOURS
HANDLER BEGIN
   declare dbmirror_state char(64);
   select property('mirrorstate') into dbmirror_state;
   IF dbmirror_state = 'synchronized' THEN
      delete from backup_files where datediff( week, today(), access_date_time) >= 1;
   END IF
END
```

Database Mirroring

- DEMO

Database Mirroring - Performance

Use similar/same hardware to guarantee consistent performance in case of failover

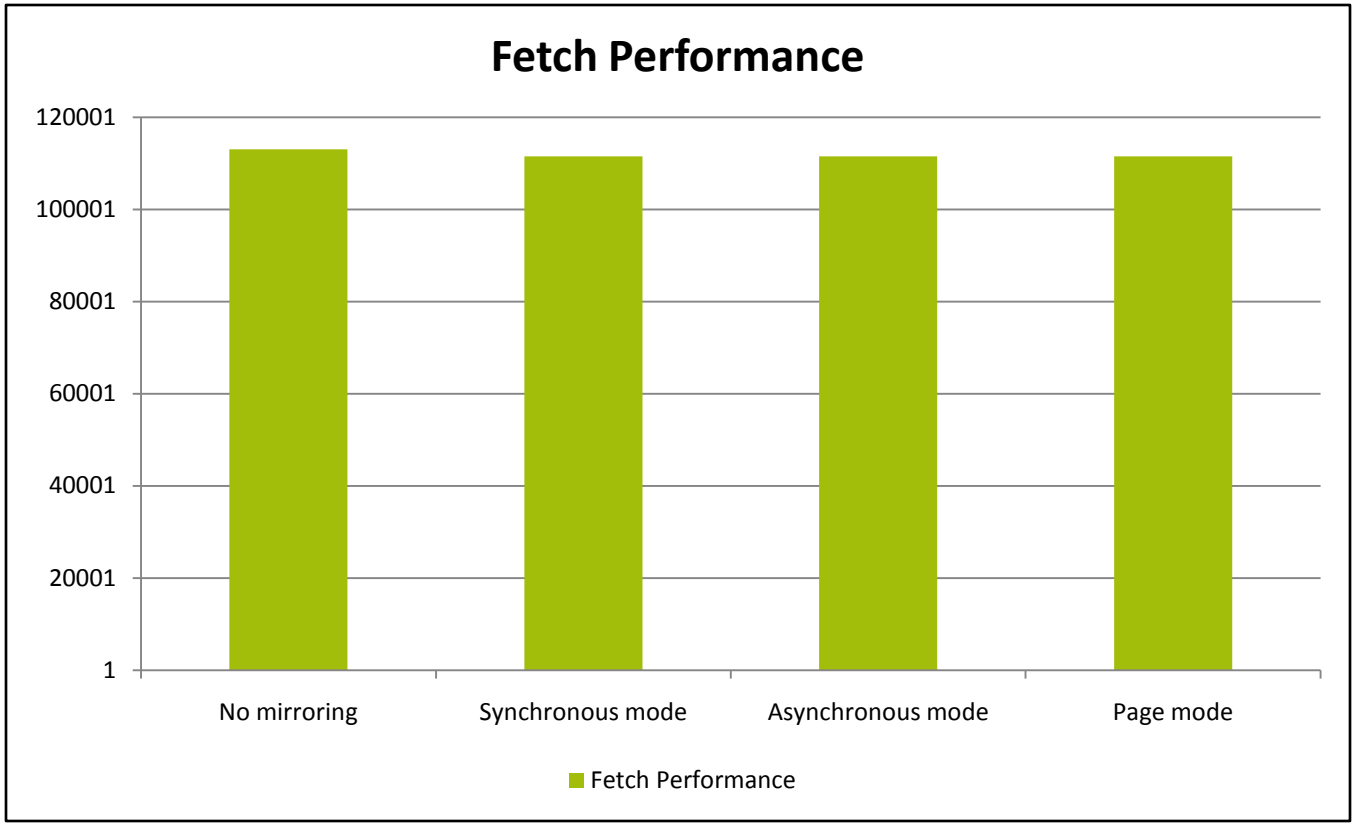
Queries

- No impact on performance
- Mirror server utilization is typically low

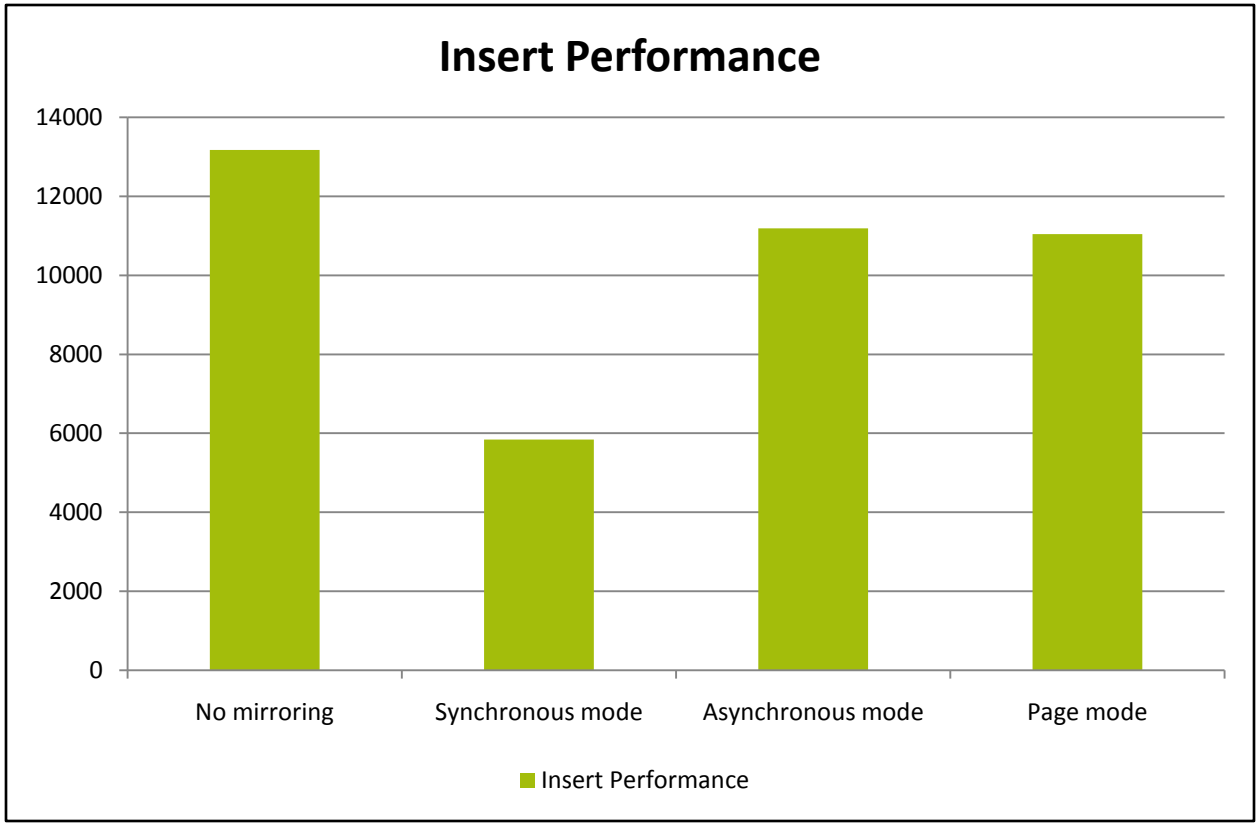
Insert/update/delete

- Performance generally slower than equivalent non-mirrored environment
- Asynchronous mode faster than synchronous mode
- Highly dependant on network throughput and latency of connection between mirrored servers

Database Mirroring – Fetch Performance



Database Mirroring – Insert Performance



Database Mirroring – Futures

- Mirror availability for read-only access
  - Offload reporting and queries from primary to mirror
    - Improves scalability and concurrency of applications

Questions?

- Thank You!

Jason Hinsperger
hinsperg@ianywhere.com
519-883-6492