YottaDB™ Intermediate

Comsan Chanma (Neo)

T.N. Incorporation Ltd.

About Me

- Department Manager, SE-Excellence Performance Lab Team, T.N. Incorporation Ltd.
- 17 years' experience on GT.M and YottaDB database (PROFILE CBS Project).
- SE Data Engineer Team Leader.
- Performance Testing Specialist.

Agenda

2st Day.

- YottaDB Replication
- YottaDB Replication Mechanism
- YottaDB Replication Command
- YottaDB Replication & Recovery
- YottaDB Replication : Failure

YDB Replication

Theory of Operation

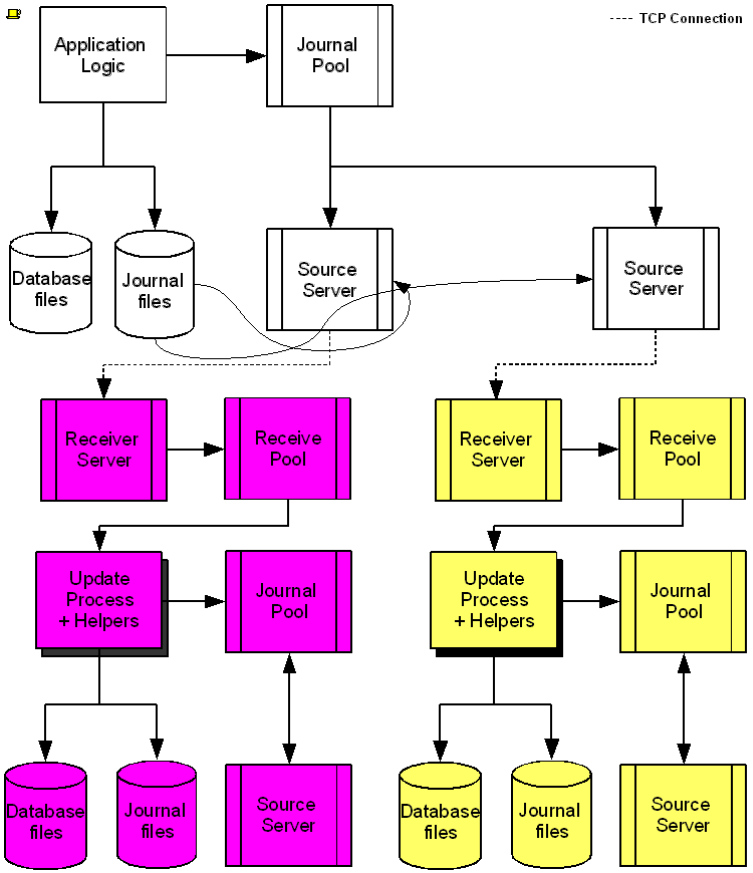
- YDB database replication provides the ability to implement continuous application availability, using a primary and secondary system, in case of complete system failure in one or more of the following components:
  - Either the primary or secondary system
  - The network between systems
  - The network between clients and the primary system

YDB Replication

- When replication is turned on for a database region, all updates to that region on a primary system replicate in near real-time on the database of a secondary system.
- The following steps characterize database updates. The first two steps occur with or without replication:

1. The journal file is written.
2. The database is updated.
3. The logical (M-level) journal file entry is delivered to a replication Source Server which in turn delivers it to the secondary system.

YDB Replication Architecture



YDB Replication

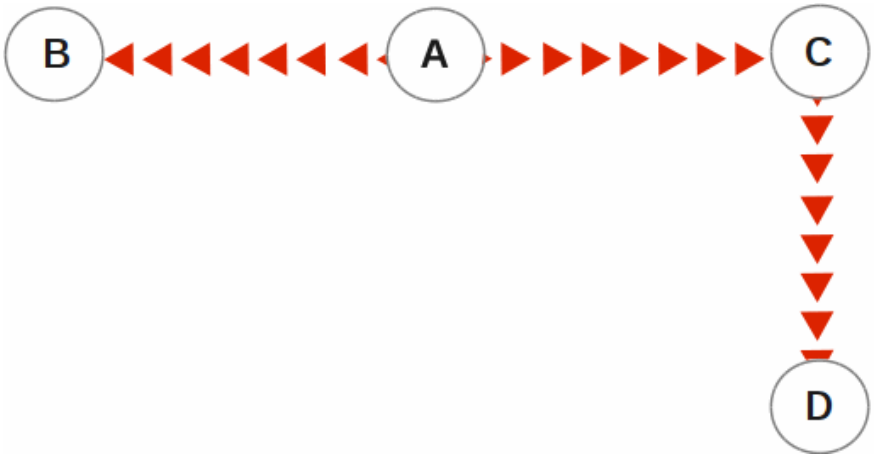
- YDB replicates the database by transporting the control records and M–level update journal records generated at the primary system to the secondary system and applying them there.
- Transaction commits at the primary system and data transfers to the secondary system occur asynchronously.
- If the secondary system or the communication link fails, it can lag behind the primary system until the two systems reestablish communication.
- The journal entries are replicated as units related to a database transaction, i.e., within a transaction fence.

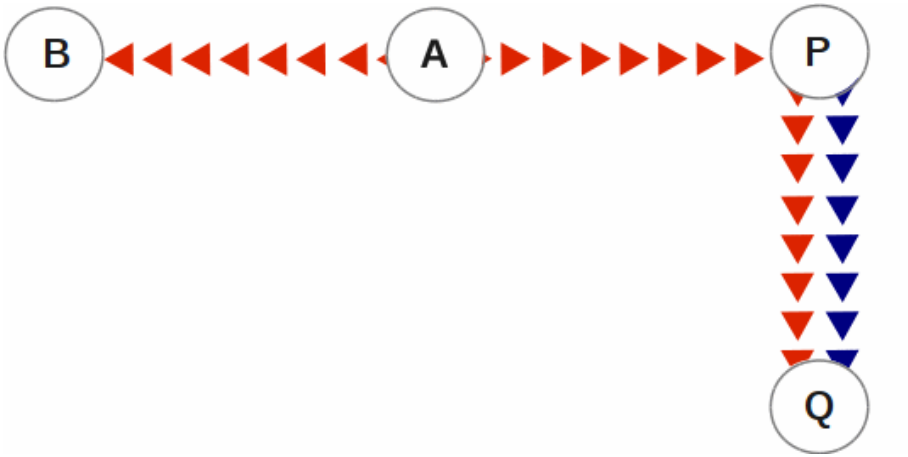
YDB Replication

- For optimum recovery, the replicated updates are moved to the secondary system at a rate as close to their creation rate as possible.
- For optimum performance on the primary system, disk I/O related to the replication process should be zero.
- To achieve this, the network connection and the software subsystem of the secondary must have adequate bandwidth for peak update rates on the primary.

Type of Replication

- Business Continuity (BC) replication
- Supplementary Instance (SI) replication





YDB Replication Mechanism

YDB Process

- Database replication requires before-image journaling to be enabled and turned ON.
- A YDB process writes journal records to a journal buffer and then flushes them to a disk.
- The Source Server process transports each record from the Journal Pool to the secondary system via a TCP/IP connection.

Journal Pool

- The Journal Pool, in memory, contains copies of journal records that the Source Server must send to the secondary system.
- If the Journal Pool overflows, the Source Server detects the condition and automatically obtains the journal records from the journal files and sends them to the secondary system in journal sequence order.
- Resynchronization is implemented with the same mechanism. At reconnection time, transactions needed for resynchronization are read from the journal files and sent to the secondary in journal sequence order.

Journal Pool(Cont.)

- The journal sequence uniquely determines a global transaction sequence.
- The journal sequence is recorded in the JNL_SEQNO field of the journal update records.
- The JNL_SEQNO is also copied to the reg_seqno field of the file header of each replicated region updated in the transaction.
- All journal update records for a single transaction have the same JNL_SEQNO.
- The JNL_SEQNO helps synchronize the systems when the systems are not synchronized.

Source Server

- Before any database activity, the Source Server must be started in either active or passive mode so it can transfer journal records from the Journal Pool to the communication channel.
- The Source Server sets up the shared structures of all replicated regions and creates the Journal Pool.
- Before replication can occur, every replicated database region must be rundown and the Source Server must be started.

Source Server(Cont.)

- Active Mode
  - the Source Server computes the startup value of JNL_SEQNO from the REG_SEQNO in the file headers.
  - The startup value is the last saved consistent state (the maximum REG_SEQNO in any file header).

Source Server(Cont.)

- Passive Mode
  - the Source Server acts as a stand-by server, waiting to activate in case of a failover.
  - When a passive server is started, it computes the startup JNL_SEQNO.
  - When a passive server is activated, it establishes a connection with the specified Receiver Server, and operates as a Source Server in active mode.

Source Server(Cont.)

- The sequence number from which the Source Server starts transmitting updates to the secondary is jointly determined by the Source Server and Receiver Server.
- The Source Server automatically locates the journal update records from either the Journal Pool or the journal file.
- If you specify a filter, the Source Server will send the replication stream to the filter, and the output of the filter, to the Receiver Server on the secondary system.

Source Server(Cont.)

- The sequence number from which the Source Server starts transmitting updates to the secondary is jointly determined by the Source Server and Receiver Server.
- The Source Server automatically locates the journal update records from either the Journal Pool or the journal file.
- If you specify a filter, the Source Server will send the replication stream to the filter, and the output of the filter, to the Receiver Server on the secondary system.

Source Server(Cont.)

- When the Source Server receives a message to stop sending journal records, it stops until it receives a restart request.
- A keep-alive protocol ("heartbeat") between the Source Server and Receiver Server detects problems with the communication channel.

Receiver Server

- The Receiver Server receives the journal records sent by the Source Server from the primary and puts them in the Receive Pool for future processing.
- An Update Process then updates the database from the Receive Pool.
- Upon startup, the Receiver Server creates the Receive Pool, starts the Update Process, and waits until the Update Process informs it of the reference point for starting (or restarting) the transmission of journal records from the primary.
- Once it receives the request, the Receiver Server sends a request to the Source Server to start or restart transmitting journal records from the specified reference point.

Receiver Server(Cont.)

- If any application filter is active, the Receiver Server collects the received journal records into groups of records that belong to the same transaction, and inputs each group into the filter. Then, the Receiver Server puts the filter output into the Receive Pool.
- When receiver pool is exceed the threshold, the receiver server sends a message to the Source Server to stop sending journal records.
- When the Receiver Server detects that the Receive Pool has adequate free space, it sends the Source Server a message to restart the process of sending journal records from the point that the process stopped.
- The Receiver Server participates in the keep-alive protocol by responding to every heartbeat message it receives from the Source Server with another heartbeat message.

Server Shutdown

- At shutdown, a Source Server in the active mode performs the following:
  - Flushes the dirty database cache buffers of all replicated regions
  - Transmits as many pending updates as possible to the Receiver Server within the specified time limit
  - Deletes the Journal Pool
  - Breaks the connection with the Receiver Server
  - Exits
- Under normal operation, the Source Server should be shut down only after all M processes accessing the replicated regions have terminated.

Server Shutdown(Cont.)

- At shutdown, the Receiver Server sets a flag in the Receive Pool that signals the Update Process to shut down.
- When this flag is set, the Update Process flushes the database cache buffers of all replicated regions, sets a flag in the Receive Pool to inform the Receiver Server that the task is complete, and exits.
- The Receiver Server confirms that the Update Process has exited, deletes the Receive Pool, breaks the connection with the Source Server, and exits.

Application Instance

- YDB's design allows the primary and secondary systems to operate on the same machine as separate instances.
- Different instances of an application are distinguished by the value of the environment variable ydb_gbldir which defines a YDB Global Directory.
- A process accessing a database file in an instance must use the Global Directory of that instance.

Filter

- Both Source and Receiver Servers can invoke filters.
- In the typical environment, the machine with the newer software will invoke filters.
- The filter should accept as input the logical database updates associated with a transaction and return the corresponding updates under the old or new schema, based on the purpose of the filter.

Statistics

- YDB provides the following replication statistics:
  - The number of database transactions in the replication queue (i.e., the backlog of transactions not yet shipped to the secondary system)
  - The speed at which to send database transactions to the secondary system

Failover & Database Synchronization

- When a primary system goes down and a secondary system takes over as the new primary. Un-replicated "in flight" transactions that do not appear in the new primary, may exist in the old primary.
- When the former primary comes up as the new secondary it will have transactions on its database that do not exist on the database of the new primary.
- To achieve database consistency, it must roll back the database to a synchronization point, or a transaction known to exist on the new primary.

Failover & Database Synchronization

- To support rollback/recovery to a known synchronization point, the Source Server and the Update Process store records indicating the last point when the two systems were linked by replication, and the mode of each system (active or passive) at the time.
- When a system is in the primary role, it retains a record of the last database transaction sent to the secondary.
- When a system is in the secondary role, it retains a record of the last database transaction received from the primary.

Application Architecture

- To create a robust dual-site architecture, pay attention to the issue of database consistency :
  - Package all database updates into transactions that are consistent at the level of the application logic using the YDB TSTART and TCOMMIT commands.
  - Ensure that internally driven batch operations store enough information in the database to enable an interrupted batch operation to resume from the last committed transaction.
  - If the application cannot or does not have the ability to restart batch processes from information in the database, copy a snapshot of the database to the secondary system just before the batch starts.
  - Ensure that externally driven batch processing also has the ability to resume.

YDB Replication Command

Controlling Replication

- System variable
  - **ydb_repl_instance** (gtm_repl_instance) specifies the location of the replication instance file when database replication is in use.
  - **ydb_repl_instname** (gtm_repl_instname) specifies a replication instance name that uniquely identifies an instance.
  - **ydb_repl_instsecondary** (gtm_repl_instsecondary) specifies the name of the replicating instance in the current environment. YottaDB uses $ydb_repl_instsecondary if the -instsecondary qualifer is not specified.

Controlling Replication (Cont.)

- **Turning Replication On/Off** (standalone access)

```
$ mupip set {-file db-file | -region reg-list}
-replication={ON | OFF}
```

- **Creating the Replication Instance File**

```
$ mupip replicate -instance_create -name=<instance name>
[-noreplace] [-supplementary] [-noqdbrundown]
```

- **Displaying and Changing the Replication Instance File**

```
$ mupip replicate
-edit[instance] {<instance-file>|-source -jnlpool}
{-show [-detail]|-change [-offset=] [-size=] [-value=]}
[-name=<new-name>] [-[no]qdbrundown]
```

Controlling Replication (Cont.)

- Starting the Source Server

```
$ mupip replicate -source -start {-secondary=<hostname:port>|-passive}
[-buffsize=<Journal Pool size in bytes>] [-filter=<filter command>]
[-freeze[=on|off] -[no]comment[='"<string>"']
[-connectparams=<hard tries>,<hard tries period>,
<soft tries period>, <alert time>, <heartbeat period>,
<max heartbeat wait>] -instsecondary=<replicating instance name>
[-[no]jnlf[ileonly]] -log=<log file name> [-log_interval=<integer>]
{-rootprimary|-propagateprimary} [{-updok|-updnotok}]
[-cmplvl=<compression level>] [-tlsid=<label>]
[-[no]plaintextfallback]
[-renegotiate_interval=<minutes>]
```

Controlling Replication(Cont.)

- Shutting Down the Source Server

```
$ mupip replicate -source -shutdown [-instsecondary=<instance_name>]
[-timeout=<timeout in seconds>] [-zerobacklog]
```

- Activating a Passive Source Server

```
$ mupip replicate -source -activate -secondary=<hostname:port>
-log=<log file name> [-log_interval=<integer>]
[-connectparams=<hard tries>,<hard tries period>,
<soft tries period>,<alert time>,<heartbeat period>,
<max heartbeat wait>]
-instsecondary=<instance_name> {-rootprimary|-propagateprimary}
```

Controlling Replication(Cont.)

- Deactivating an Active Source Server

```
$ mupip replicate -source -deactivate -instsecondary=<instance_name>
```

- Stopping the Source Filter

```
$ mupip replicate -source -stopsourcefilter
```

- Checking Server Health

```
$ mupip replicate -source -checkhealth
[-instsecondary=<instance_instance>] [-he[lpers]]
```

Controlling Replication(Cont.)

- Changing the Log File

```
$ mupip replicate -source -changelog -log=<log file name>
[-log_interval=<integer>] -instsecondary=<instance_name>
```

- Enabling/Disabling Detailed Logging

```
$ mupip replicate -source -statslog={ON | OFF}
[-log=<log file name>]
```

- Reporting the Current Backlog

```
$ mupip replicate -source -showbacklog
```

Controlling Replication(Cont.)

- Starting the Receiver Server

```
$ mupip replicate -receiver -start -listenport=<port number>
-log=<log file name> [-log_interval="[integer1],[integer2]"]
[-autorollback[=verbose]] [-buffsize=<Receive Pool size in bytes>]
[-filter=<filter command>] [-noresync]
[-stopsourcefilter] [-updateresync=</path/to/bkup-orig-repl-inst-file>
{[-resume=<strm_num>|-reuse=<instname>]} [-initialize] [-cmplvl=n]
[-tlsid=<label>]
```

- Starting the Update Process

```
$ mupip replicate -receiver -start
{-updateonly|-helpers[=m[,n]]}
```

Controlling Replication(Cont.)

- Stopping the Update Process and/or the Receiver Server

```
$ mupip replicate -receiver -shutdown [-helpers | -updateonly]
[-timeout=<timeout in seconds>]
```

- Checking Server Health

```
$ mupip replicate -receiver -checkhealth
```

- Changing the Log File

```
$ mupip replicate -receiver -changelog -log=<log file name>
[-log_interval="[integer1],[integer2]"]
```

Controlling Replication(Cont.)

- Enabling/Disabling Detailed Logging

```
$ mupip replicate -receiver -statslog={ON|OFF}
[-log_interval="[integer1],[integer2]"]
```

- Reporting the Current Backlog

```
$ mupip replicate -receiver -showbacklog
```

- Rolling Back the Database

```
$ mupip journal -rollback
{[-fetchresync=<port number>|-resync=<JNL_SEQNO>]
[-rsync_strm=<strm_num>]} -losttrans=<extract file> -backward *
```

Controlling Replication(Cont.)

Other command

```
$ mupip journal -show=header <jnl_file>

$ mupip set -jnlfile <jnl_file>
-[no]prevjnlfile=<jnlfilename> [-bypass]
```

YDB Replication & Recovery

Implementing Replication & Recovery

- System Requirements
  - Primary/Secondary Status Identification
  - Failover
    - There is only one primary at any one time, and database updates only occur on the primary system. If there is no primary system, the application is not available.
    - Messages received from clients during a failover are either rejected, so the clients will timeout and retry, or are buffered and sent to the new primary.
    - If the former primary continues operation or is back online after a crash, it must operate as a secondary.
  - Network Link between Systems
  - Database Repair

Implementing Replication & Recovery

- Procedures
  - All database updates to replicated regions must occur only on the primary system.
  - GT.M will replicate all changes to the database on the primary system to the database on the secondary system.

Procedures

- Situations
  - Normal Operation
  - Dual-Site to Single-Site
  - Failure
  - Dual-Site Failures
  - Complex Dual-Site Failure

Normal Operation

- General
  - Determine current and prior system status (primary/secondary).
  - Perform necessary database recovery/rollback. Reconcile rolled back transactions automatically or manually.
  - Create new journal files. Although not required, this will simplify system administration.
  - Start the YDB replication Source Server for this instance. On the secondary, bring up the Source Server in passive mode. On the primary, bring up the Source Server in active mode.

Normal Operation(Cont.)

- Start the YDB Receiver Server if this is the secondary instance.
- If starting up as primary, start the application servers. The application servers can also be started on the secondary to facilitate a faster failover; however, it must be guaranteed that they do not perform updates of any replicated region on the secondary.
- If starting up as primary, and the state of the database indicates that batch operations were in process when the system went down, restart batch operations.

Normal Operation(Cont.)

- **Startup Single-Site**
  - Perform database recovery/rollback to the last consistent state, since the system may previously have crashed.
  - Create new journal files.
  - Start the Source Server.
  - Start the application servers.
  - If the state of the database indicates that batch operations were in process, restart batch operations.

Normal Operation(Cont.)

- Secondary Starts from Backup of Primary
  - Load/restore the database.
  - Create new journal files without back pointers to previous generations of journal files with the -noprevjnlfile flag.
  - Start the passive Source Server and then the Receiver Server with the -updateresync qualifier, along with the other startup qualifiers for the Receiver Server.
  - Start the passive application servers, if appropriate.
  - Since the primary should not need to be rolled back to a state prior to the start of the backup, the generation link on the primary can be cut in the journal files created by the online backup command on the primary system.

Normal Operation(Cont.)

- Secondary Starts after a Shut Down or Crash
  - Recover/rollback database to last consistent state.
  - Create new journal files.
  - Start passive Source Server, and then the Receiver Server.
  - Start the passive application servers, if appropriate.

LAB 1.1

Dual-Site to Single-Site

- **Normal Failover**
  - If you have a choice, choose a time when database update rates are low to minimize the clients that may time out and retry their messages, and when no batch processes are running.
  - The external system responsible for primary/secondary status identification should be told that Site B should now be the primary system and Site A should now be the secondary.

Dual-Site to Single-Site(Cont.)

- **Normal Failover**
  - On A:
    - Stop the application servers.
    - Shut down the replication Source Server with an appropriate timeout. (default 30 seconds.)
    - Perform the rollback for the primary restarting as secondary.
    - Wait for B to become functional as the primary, and then perform a FETCHRESYNC ROLLBACK BACKWARD.
    - Create new journal files.
    - Start the passive Source Server, and then the Receiver Server.
    - Start the passive application servers, if appropriate.

Dual-Site to Single-Site(Cont.)

- **Normal Failover**
  - On B:
    - Shut down the Receiver Server with an appropriate timeout.
    - Create new journal files.
    - Make the passive Source Server active.
    - Start the application servers (if they were previously passive, make them active).
    - If the state of the database indicates that batch operations were in process, restart batch operations.
    - Begin accepting online transactions.

LAB 1.2

YDB Replication : Failure

Failure

- Network Failures
  - If the network from clients to the primary fails, and the network from the clients to the secondary is still functioning, this warrants a failover from the primary to the secondary.
  - If the network from clients to both the primary and secondary fails, the application is no longer available.
  - If the network between primary and secondary fails, no action is required to manage YDB replication.
  - If the network from the clients to the secondary fails, no action is required to manage YDB replication.

Failure(Cont.)

Secondary Fails

- When the secondary comes back up after failure, it will still be the secondary.
- Refer to the preceding “Secondary Starts After a Shut Down or Crash”.

Failure(Cont.)

Primary Fails

- The external control mechanism should detect that the primary has failed, and take action to switch the secondary to primary mode, and either route transactions to the new primary (former secondary) or notify clients to route transactions to the new primary.
- If the former primary did not respond to certain transactions, one cannot be certain whether they were processed and whether or not the database updates were committed to the former secondary.

Failure(Cont.)

Primary Fails

- On new primary (former secondary)
  - Stop the Replication Server.
  - Create new journal files.
  - Switch the Source Server from passive to active mode.
  - Start the application servers, or if they were passive, they should be activated.
  - If the state of the database indicates that batch operations were in process, restart batch operations.
  - When the new secondary (the former primary) comes back up, perform a FETCHRESYNC ROLLBACK BACKWARD.
  - Create new journal files.
  - Start the Source Server in passive mode.
  - Start the Receiver Server to resume replication.
  - As appropriate, start the passive application servers.

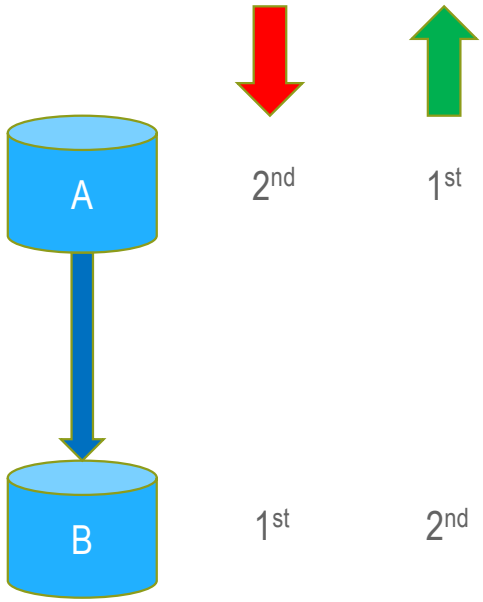
Dual-Site Failure

Secondary (Site B) Fails First

- Site A Recovers First

On Site A:

- Rollback the database to the last committed transaction (last application-consistent state).
- Create new journal files.
- Start the Source Server.
- Start the application servers. Application availability is now restored.
- If the state of the database indicates that batch operations were in process, restart batch operations.

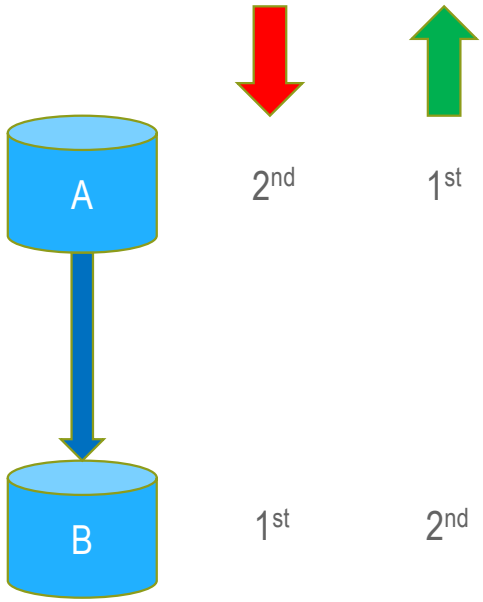


Dual-Site Failure(Cont.)

Secondary (Site B) Fails First (Cont.)

On Site B:

- Rollback the database to the last committed transaction.
- Create new journal files.
- Start the Source Server in passive mode.
- Start the Receiver Server. Dual-site operation is now restored.
- Start the passive application servers, as appropriate.



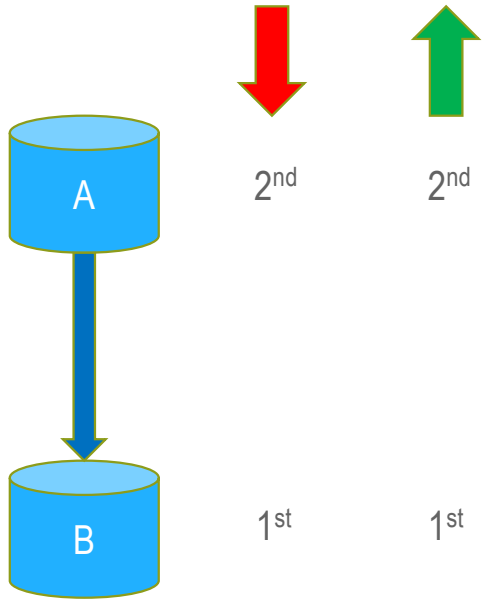
Dual-Site Failure(Cont.)

Secondary (Site B) Fails First (Cont.)

- Site B Recovers First

  On Site B:

  - Rollback the database to the last committed transaction (last application-consistent state).
  - Create new journal files.
  - Start the Source Server.
  - Start the application servers. Application availability is now restored.
  - If the state of the database indicates that batch operations were in process, restart batch operations.

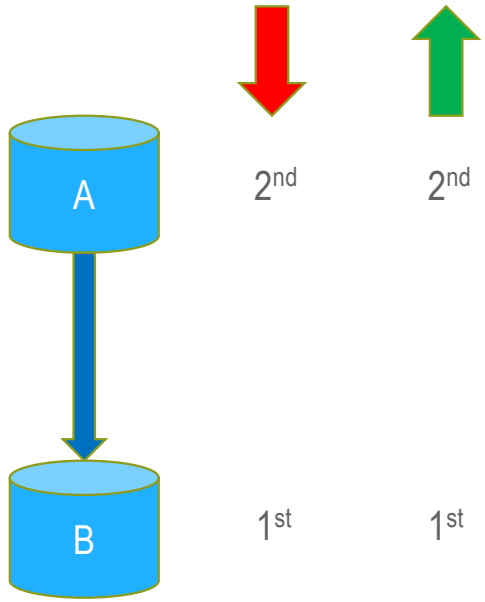


Dual-Site Failure(Cont.)

Secondary (Site B) Fails First (Cont.)

On Site A:

- Perform a FETCHRESYNC ROLLBACK BACKWARD.
- Create new journal files.
- Start the Source Server in passive mode.
- Start the Receiver Server to resume replication as the new secondary. Dual-site operation is now restored.
- Start the passive application servers, as appropriate.



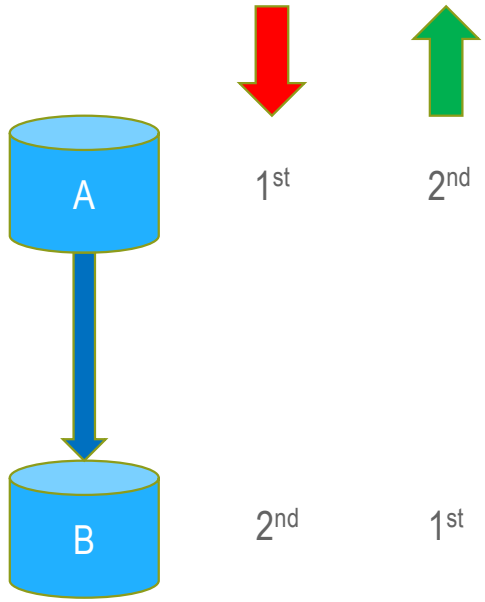
Dual-Site Failure(Cont.)

Primary (Site A) Fails First

- Site B Recovers First

  On Site B:

  - Roll back the database to the last committed transaction (last application-consistent state).
  - Create new journal files.
  - Start the Source Server.
  - Start the application servers. Application availability is now restored. If the state of the database indicates that batch operations were in process, restart batch operations.

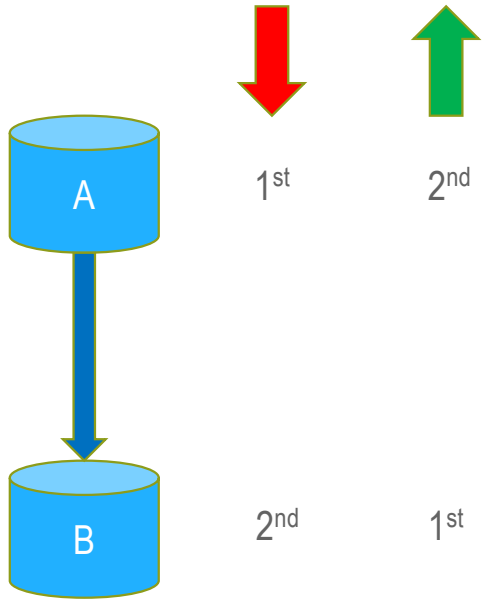


Dual-Site Failure(Cont.)

Primary (Site A) Fails First (Cont.)

On Site A:

- Perform a FETCHRESYNC ROLLBACK BACKWARD.
- Create new journal files.
- Start the Source Server in passive mode.
- Start the Receiver Server to resume replication as the new secondary. Dual-site operation is now restored.
- Start the passive application servers, as appropriate.



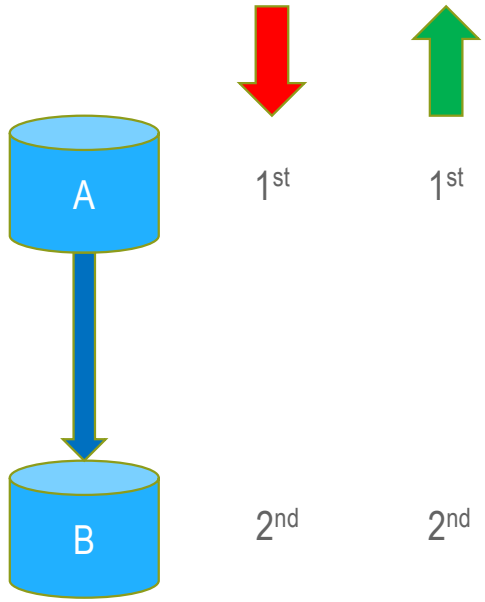
Dual-Site Failure

Primary (Site A) Fails First (Cont.)

- Site A Recovers First

  On Site A:

  - Rollback the database to the last committed transaction (last application-consistent state).
  - Create new journal files.
  - Start the Source Server.
  - Start the application servers. Application availability is now restored.
  - If the state of the database indicates that batch operations were in process, restart batch operations.

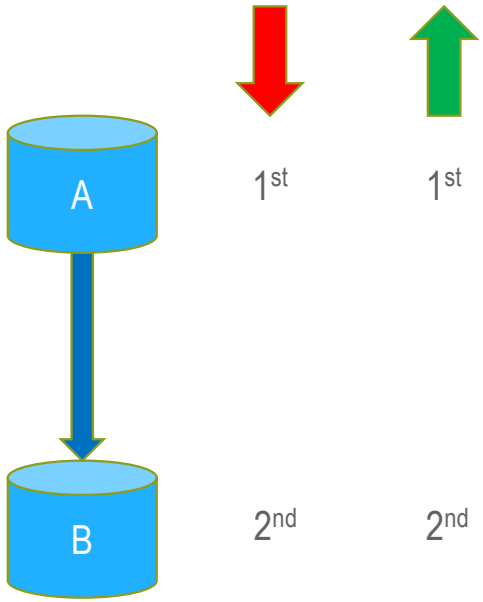


Dual-Site Failure(Cont.)

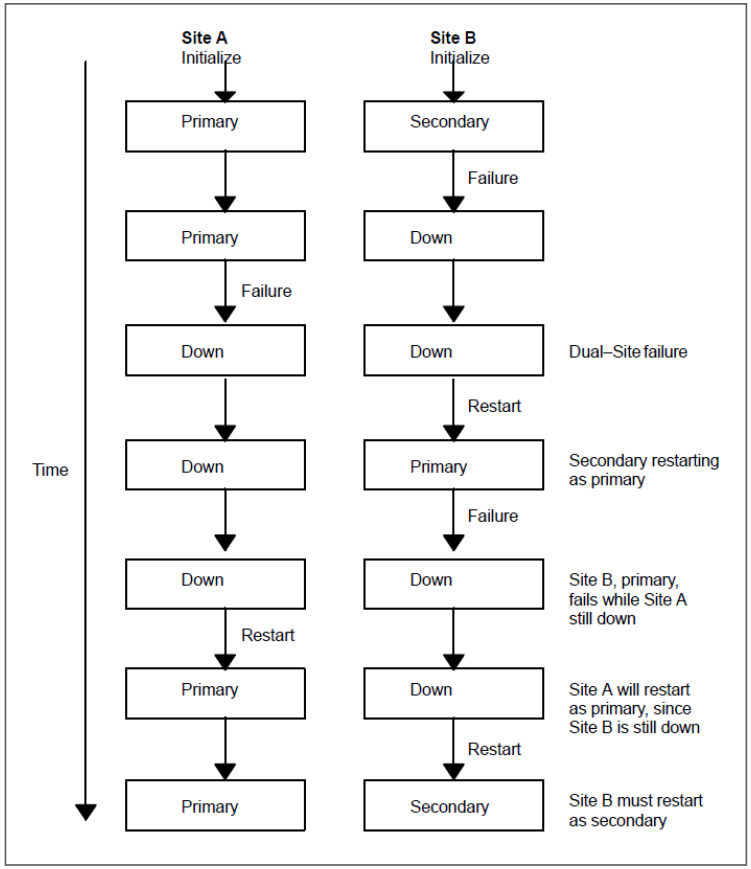
Primary (Site A) Fails First (Cont.)

On Site B:

- Rollback the database to the last committed transaction.
- Create new journal files.
- Start the Source Server in passive mode.
- Start the Receiver Server. Dual-site operation is now restored.
- Start the passive application servers, as appropriate.



Complex Dual-Site Failure



Question and Answer