

How to Implement Multi-way Active/Active Replication – SIMPLY

The easiest way to ensure data is always up to date in a 24x7 environment is to use a single global database. This approach works well if your application does not use frequent database interactions, and some latency is expected, for example because the application is browser-based. An internet store is a typical example of such an application in a 24x7 environment.

A single global database does not work well if your application requires frequent database interactions, especially if application interaction heavily impacts user productivity. In that case it makes more sense to have a local database – for example per region – so that users can always have low-latency access to the database. If users then still need access to all available data, you end up with an active/active replication environment to synchronize changes across multiple independent databases.

This paper discusses considerations for an active/active replication environment, as well as how to implement such an environment using the data replication product HVR – High Volume Replicator.

Figure 1 shows the example used throughout this paper: a 6-way active/active environment with one active database on every continent except Antarctica.

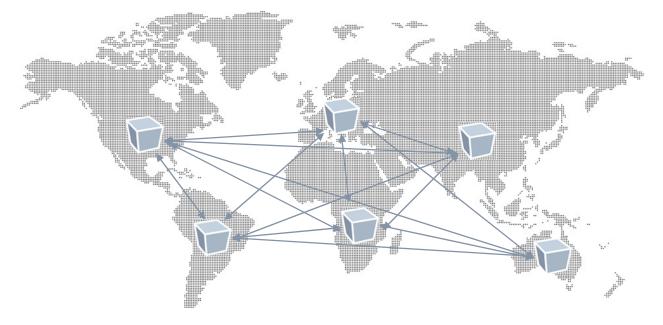


Figure 1. 6-way active/active replication across the world

Active/active replication considerations

An active/active replication environment can always introduce data-level conflicts that you will have to deal with. However there are a number of potential conflicts that you can avoid by preparing the application for active/active replication.



Primary and unique keys

OLTP applications often use a unique sequence value to populate a primary key. In a multi-way active/active scenario there are different options to avoid potential conflicts caused by duplicate sequence values:

- Use composite keys, using a unique database identifier as part of the key to ensure there never will be conflicts between rows created in different databases. Such a change may have a significant impact on the application if today there are single-column keys in place.
- Add a unique site-prefix/suffix to every key. If keys are numeric today using a numeric prefix or suffix can work. Such a change may impact key generation in the application, and can have an impact on the way unique indexes are distributed which may impact data lookup timing.
- Start with a unique sequential number on every site and increment sequence values by at least the number of sites in the active/active configuration. In the example with 6 databases on every content North-America could start with 1, South-America with 2, etc. and simply increment sequence values by 6. This approach may again impact index distribution if not all sites are generating rows at the same rate which can impact data lookup timing.

Unique keys in your application may have similar considerations, but in some cases you know you may end up with conflicts that you will have to deal with.

Triggers

Many applications use database triggers to populate column values or even entire rows in tables. In most replication environments triggers fire when the application performs a change, but they don't fire when the replication process applies a change. In a multi-way active/active replication scenario that is no different.

If you never want the replication to fire triggers then you may be able to take advantage of a database capability to disable triggers per session, but not every database exposes such an option to external replication products. As an alternative you may have to modify all triggers to ensure they don't fire during replication, for example because replication is run by a dedicated database user that is not used otherwise.

Truncate operations

Truncate operations are uncommon in an OLTP system, but they do sometimes occur. A truncate operation is a DDL statement that only logs the execution of the statement in the transaction log and not the data that was affected by the statement. In a multi-way active/active replication environment a user may be entering data into a table in one database while the table is truncated in another database. This scenario can lead to data inconsistencies with some databases containing fewer records than others. Watch out for truncate operations in your application, and if they do occur ensure you understand how these propagate to the various databases.

Cascade deletes

A cascade delete or nullify on delete operation in a replication environment is similar to a database trigger: the application probably relies on the cascade operation to proceed, but replication should not perform the cascade delete since the delete/update of child records is captured separately. You may have to work around cascade deletes by implementing triggers instead (that also take replication into

How to Implement Multi-way Active/Active Replication - SIMPLY



account). Depending on the database technology you use you may be forced to then disable constraints or use deferred constraints that perform constraint checking at commit time rather than as rows are being manipulated since the parent delete record likely comes first in sequential order.

Unavoidable conflicts

Fundamentally in an active/active environment that is kept in sync using asynchronous replication technology there may always be conflicts. In the example from Figure 1 there is no database locking to prevent a user from changing a row in the North American database while another user is changing the same data in the African database (there may always be application logic that forces segregation of changes to the data). When these changes are propagated to all environments all databases should be in sync, and one of the users should be notified that their change was not processed successfully.

Likewise there are unavoidable conflicts with deletes and updates that a typical application will have to deal with in an active/active scenario.

Introducing HVR – High Volume Replicator

HVR – High Volume Replicator – is a technology that captures transactions with very low latency and little overhead directly out of the transaction logs of commonly used relational databases. Transaction data is compressed before moved over the wire through a central hub into one or more target systems.

In addition to relational databases HVR can capture changes out of Salesforce, and HVR supports filebased replication out of directories or a SharePoint system.

Data is always compressed across the wire using a highly efficient compression scheme, and optionally the software can be configured to encrypt data as it moves between source and target(s).

HVR is used in multiple use cases:

- Real-time Business Intelligence and reporting offload in which HVR keeps one or more copies of a source OLTP system in sync in real-time, often in a heterogeneous database environment.
- Cloud integration, in which organizations store some data on-premise and some data lives in the cloud.
- Multiple types of migrations such as hardware changes, software upgrades, database changes as well as on-site to cloud migrations.
- High Availability (HA) and Disaster Recovery (DR) implementations, typically over a wide-area network.

Active/active replication is a special case of a DR implementation in which multiple sites are active concurrently.

Multi-way Active/Active Replication with HVR

Thanks to HVR's well thought-through architecture configuring multi-way active replication is surprisingly simple. Some replication products require careful configuration on every environment but with HVR the entire setup is created and maintained from a single central environment.

This section provides a step-by-step guide on how to setup an environment for the 6-way active/active replication example in Figure 1.



Installation and initial configuration

In order to take maximum advantage of HVR's compression every database system that participates in the replication must have an installation of the HVR software. One of the installations will be designated as the hub, and for high availability reasons you should designate at least one alternative location as a backup hub that can take over the replication in case the primary system fails.

The HVR software is only about 50 MB in size and installs in minutes. Every locations that does not act as a hub must run the HVR remote listener process on a designated port that must be accessible from other servers in the environment.

For more information and initial configuration refer to the HVR User Manual.

Setting up the environment

HVR's Graphical User Interface connects to a database schema in a relational database that is called the hub (see Figure 2).

HVB HVR					
Evis HVR Eile ⊻iew <u>H</u> elp					
Hub Machines	Location	Class	Node	Description	
- 🖪 localhost					
È- ∏ hvrhub					
Location Configuration					
🚰 Channel Definitions					
🚱 Scheduler					
Actions Attributes					ą
Channel Group Table Location Action					

Figure 2. HVR GUI connected to a hub.

The first step in setting up the replication is to define the Location Configuration, specifying the credentials and environment information on the various locations (see Figure 3).

HVR SOFTWARE
DEAL-TIME DATA DEDI ICATION

Location properties for	eu			×
Location				
Location eu				
Description Europe				
Connection Group Mem	bership			
Connect to HVR on remo	ote machine			
Node europe.corp		Login	hvr	
Port 4343		Password	•••••	
Class	Database Conne	ction		
Oracle	ORACLE_HOME	/u01/app	/oracle/product/11.2.0/db_1	
Ingres / Vectorwise	ORACLE_SID	db112		
SQL Server				
DB2	RAC	SCAN	15	21 🔺
🔘 Teradata	-	Service		
Paraccel	User	hvrdb		
Greenplum		nvrub		
File / FTP / Sharepoint	Password	•••••	•••••	
Salesforce				
	Test Connection	ОК	Cancel	lelp

Figure 3. Location definition in HVR.

This must be done for all 6 locations in this example.

Next define a channel. A channel is a logical group of tables that should be replicated together, for example because there is enforced referential integrity between the tables. A channel often maps all or a subset of the tables in a schema. In this example we use distapp, Distributed Application, as the channel (see Figure 4).

New Channel				
Channel distapp				
Description	Distributed application			
	OK Cancel			

Figure 4. Creating a new channel.

The Channel is a logical definition of what tables are replicated and how.



To map the logical definition to physical connection details HVR uses the concept of a Location Group within the context of a Channel. For a six-way active/active replication implementation effectively only one Location Group is required, since all databases all play an identical role in the replication (see Figure 5).

👿 New Gro	oup				×		
Channel	distapp	istapp 👻					
Group Name	ACTIVE						
Description	Single location	n for all active systems					
Group Mem	bership						
Location	n Class	Node	Description				
🗸 af	oracle	africa.corp	Africa				
🗸 as	oracle	asia.corp	Asia				
🔽 au	oracle	australia.corp	Australia				
🔽 eu	oracle	europe.corp	Europe				
🔽 na	oracle		North America				
🔽 sa	oracle	sourth-america.corp	South America				
			ОК	Cancel	Help		

Figure 5. New Location Group with all 6 databases as members.

Next identify the tables taking part in the replication. This examples uses the sample SH schema in the Oracle database. In the channel, right-mouse click Tables and choose Table Select from the pop-up menu. Select any one of the databases to connect to if all schemas are identical, or simply use the local database as the starting point.

HVR SOFTWARE
REAL-TIME DATA REPLICATION

Table Selection for distap	p and na		×
Only Current Schema			
Base Table Name H	IVR Table Name	Match	
cal_month_sales_mv		Only in Database	
channels		Only in Database	
costs		Only in Database	
countries		Only in Database	
customers		Only in Database	
dimension_exceptions		Only in Database	
dr\$sup_text_idx\$i		Only in Database	
dr\$sup_text_idx\$k		Only in Database	
dr\$sup_text_idx\$n		Only in Database	
dr\$sup_text_idx\$r		Only in Database	
fweek_pscat_sales_mv		Only in Database	
products		Only in Database	
promotions		Only in Database	
sales		Only in Database	
sales_transactions_ext		Only in Database	
supplementary_demographic		Only in Database	
times		Only in Database	
	Add Repla	ce Delete I	Details Close
			1

Figure 6. Select tables to be included in the channel.

Identify which tables should be part of the Channel and click Add (see Figure 6). HVR will retrieve the table definition including primary/unique key definitions. If there is no key HVR will show a no-key icon in the list of tables. HVR can deal with tables without keys and avoid duplicate row updates in case there would be duplicate rows, but heavy update/delete activity on a table with no key or index can seriously hurt performance.

Next, on the Location Group ACTIVE, define a New Action DbCapture which will apply to all locations in this group. Enable log-based capture (see Figure 7). There are many additional options that are useful for specific cases. Please refer to the HVR User Manual to get full details on the various options and when to use them. This example defines actions at the Location Group level but actions can also be defined for the entire channel, or for a specific table.



Wew Action: DbCapt	ure	—
	Group ACTIVE Location *	
✓ /LogBased	/IgnoreSessionName	
/ClusterThread	JignoreCondition	A
/QuickToggle	/IgnoreUpdateCondition	
/ToggleFrequency	🗦 🔳 /HashBuckets	
KeyOnlyCaptureTable	/HashKey	
NoBeforeUpdate	Coalesce	
NoTruncate		
Regular Text		
		OK Cancel Help

Figure 7. Creating a DbCapture action.

Then create a new DbIntegrate action for the single location group for all tables (see Figure 8).

www. Action: DbIntegrate		
Channel distapp Group A Table Configuration Action	CTIVE	
DbProc	/ResilientUpdate	
OnErrorSaveFailed	/ResilientDelete	
OnErrorBlockLocation	/ResilientDeleteCondition	<u></u>
/TxBundleSize	🚑 📃 /ResilientWarning	
/TxSplitLimit	NoTriggerFiring	
CycleByteLimit	🚖 📃 /SessionName	
/Burst	/DbProcDuringRefresh	
BurstCommitFrequency	- /Coalesce	
Resilient	🧾 /Journal	
ResilientInsert	Delay	
Regular Text		
		OK Cancel Help

Figure 8. Creating a DbIntegrate action.

DbIntegrate also has many options that are useful in specific situations. For example ResilientDelete can be used if there are enabled cascade delete constraints on tables and the errors resulting from the deletion of rows that were already removed should be ignored.



Finally, add a CollisionDetect action (see Figure 9) to ensure the databases remain in sync even if there are multiple concurrent operations on the same data in different databases. With CollisionDetect enabled HVR will automatically keep track of the recent history of changes to ensure the most recent committed change wins.

New Action: CollisionDetect	—
Channel distapp Group ACTIVE	
Table * Location *	
Configuration Action	
/TreatCollisionAsError	/DetectDuringRefresh
/TimestampColumn	🖸 /Context
/AutoHistoryPurge	
Regular Text	
	QK Cancel Help

Figure 9. Adding a CollisionDetect action.

These few steps are all that is required to define a six-way active/active environment.

Initializing replication

With the Channel Definition complete use HVR Load to initialize the replication. The Advanced Options tab shows a number of options that may be selected/deselected in some cases (see Figure 10).

	HVR SOFTWARE
HVR Load for channel distapp	
 Create or Replace Objects Drop Objects 	
Locations Advanced Options	
Object Types Image: State Tables Image: Change Tables Image: Database Triggers Image: Database Triggers	Capture Rewind None Back to Parallelism for Locations None
Table Name Base Table Name	/R Load Close Help
nyhoad -n oracle nyhub/ ((QD/Q.KQK)	a uistapp

Figure 10. Running HVR Load to initialize the replication.

For example if the channel was active before but then got modified to include more tables then Table Enrollment and Supplemental Logging have to be created for these tables, but other options can be deselected. Also note the option to rewind the replication to a point in time. This can be useful if for example you used a cold database export to initialize the other environment(s) and you wanted to capture any transactions since then. Please note that if you rewind the replication then all transaction logs between the point in time to which rewind is set and the current point in time must be accessible to HVR on the location that is known to the database (for example Oracle keeps this information in a view (G)V\$ARCHIVED_LOG).

For the first initialization of a channel simply use all defaults. Please note that HVR Load may add supplemental logging to the tables in the channel which requires an exclusive lock on the table. If an application is actively performing DML on the tables then running HVR Load can take a long time because it is waiting to acquire the locks.

HVR can be fully instructed from the command line, so almost all interactions with the GUI have a command-line equivalent. Whenever appropriate HVR will show the command-line equivalent of a GUI action in the status bar (e.g. see the hvrload command in the status bar in Figure 10).



HVR Load will create two jobs per physical location under the Scheduler node to run the replication: one for capture, and one for integrate.

Initial load

HVR provides the ability to perform the initial load through an operation called HVR Refresh. Refresh will simply take all data from one of the databases and load in bulk into one or more targets. Tables are truncated before data is loaded to ensure data will be in sync.

HVR Refresh can be run when the application is active, and the options under Online Refresh control how HVR deals with in-flight transactions during the refresh to ensure there are no collisions and post initial load and once the replication has caught up all systems are in sync (see Figure 11). HVR Refresh also provides the option to create tables that don't yet exist, or recreate tables that were modified.

HVR Refres	h for channel distap	р					• 🗙
Location				Location	ı		
Location	Class Node			Loc	ation Class	Node	
🗖 af	oracle			🔽 af	oracle		
as	oracle			as 🗸	oracle		
au	oracle			🕨 🔽 au	oracle		
eu	oracle			eu 🗸 eu	oracle		
🗸 na	oracle			na 📃	oracle		
sa	oracle			V sa	orade		
Show All				Show	All		
Table Name	Base Table Na						
Refresh Data							
Bulk Granulation Row by Roy	arity w Granularity				Parallelism for L		
Verbose	W Grandiancy				Parallelism for T	ables 4	•
Fire Trigger	s during Refresh				Online Refre	esh	
Apply All		-			Skip Previou	is Capture and Int	egration
					Only Skip Pr	evious Integration	1
Refresh Db	Sequences				Do not Skip	Changes	
Create Abse	nt Tables						
With Key							
With Unicod	e Datatypes						
Recreate Mi	ismatched Tables						
Schedule Re	fresh Jobs						
Taskname refr]					
nvrrefresh -qb	-P4 -p5 -qrw -cbk -	r na -l af -l as	-l au -l eu -l sa	Refresh -h oracle hvrhub/!{	Schedule Qb/Q.KqR}! dista	Close	Help

Figure 11. HVR Refresh options.



Use Parallelism for Locations and Parallelism for Tables to limit the amount of time it takes to run the refresh. Depending on the network speed and the amount of data that must be loaded HVR Refresh can take a long time. In the GUI HVR will show progress graphically (see Figure 12), and when run through the command line output is reported on standard out. There is also an option to Schedule Refresh Jobs in which case output is stored as part of the scheduler logging.

👿 Refresh R	esult				
Summary	Output				
Table Name	Target	State	Rows on na	Duration	
channels	af				
channels	as				
channels	au				
channels	eu				
channels	sa				
costs	af				
costs	as				
costs	au				
costs	eu				
costs	sa				
countries	af				
					1
Analyzing Dat	amodel and I	ndexes			
analyzing but	amouel and 1	indexed			Cancel
					Cancel

Figure 12. Refresh results in the HVR GUI.

Once Refresh finishes all locations will have table definitions and data as of a recent point in time.

Running the replication

The jobs to start the replication were created by HVR Load (see Figure 13). HVR will report the latency of data changes which in this case – given the jobs have not run before – is the amount of time elapsed since HVR Load was completed.



WE HVR									- P ×
<u>File View H</u> elp									
Hub Machines	Job	State	Retries	Recent Error	Latency	Router Rows	Router Bytes		
🖻 🕞 localhost	distapp-cap-af	SUSPEND			21m 17s				
⊡-@hvrhub		SUSPEND			21m 17s				
		SUSPEND			21m 17s				
					21m 17s				
	distapp-cap-eu								
🗄 🔚 🔚 distapp	distapp-cap-na				21 m 17 s				
🖕 🔚 distapp-cap	distapp-cap-sa	SUSPEND			21m 17s				
··· 🔜 distapp-cap-af	distapp-integ-af	SUSPEND			21m 17s				
··· distapp-cap-as	distapp-integ-as	SUSPEND			21m 17s				
distapp-cap-au distapp-cap-eu	distapp-integ-au	SUSPEND			21m 17s				
distapp-cap-eu	distapp-integ-eu				21m 17s				
distapp-cap-sa	distapp-integ-na				21m 17s				
E- 📴 distapp-integ					21m 17s				
···· 🛄 distapp-integ-af	distapp-integ-sa	SUSPEND			21m 1/s				
···· 🔜 distapp-integ-as									
··· 🔜 distapp-integ-au									
distapp-integ-eu distapp-integ-na									
distapp-integ-sa									
and apply integration									
Actions Attributes									ą
Job/Group Attribute									
DISTAPP-CAP retry_delay 10 1200									
DISTAPP-INTEG retry_delay 10 1200									
SYSTEM quota_speed 20 1									
STSTEM duota_speed 201									

Figure 13. Jobs created by HVR Load.

To start the jobs, simply use the option in the pop-up menu on the Scheduler or the Channel to Trigger all jobs (see Figure 14). Start the scheduler if it is not running yet.

HVR HVR												- P ×
<u>F</u> ile <u>V</u> iew	<u>H</u> elp											
Ub Machines	s ub .ocation C Channel D @distap		Trigger Suspend Unsuspend New Attribute Delete	Job distapp-cap-af distapp-cap-au distapp-cap-au distapp-cap-au distapp-cap-au distapp-cap-aa listapp-integ-au listapp-integ-au listapp-integ-au listapp-integ-au	SUSPEND SUSPEND SUSPEND SUSPEND SUSPEND SUSPEND SUSPEND SUSPEND	Retries	Recent Error	Latency 24m 3s 24m 3s	Router Rows	Router Bytes		
	tributes											ą
Job/Group	Attrib											
DISTAPP-CAP	retry	_delay 10 1200										
DISTAPP-INTE	G retry	_delay 10 1200										
SYSTEM	quota	a_speed 20 1										

Figure 14. Trigger all jobs in the scheduler.

If all is well then all jobs should start and depending on the activity in the system since HVR Load completed the jobs should all catch up quickly and report RUNNING (see Figure 15).



HVR HVR								- # ×
<u>File View H</u> elp								
	distapp-cap-as distapp-cap-au distapp-cap-eu distapp-cap-na	RUNNING RUNNING RUNNING RUNNING RUNNING RUNNING	Retries	Recent Error	Latency 1s 1s 2s 2s 2s 2s 2s 2s 2s 2s 2s 2	Router Rows	Router Bytes	
Actors Attribute Job/Gróup Attribute DISTAPP-CAP retry_delay 10 1200 DISTAPP-INTEG retry_delay 10 1200 SYSTEM quota_speed 20 1								8



The system is now active and changes made in any one of the databases will flow to all other databases.

Conclusion

Active/active replication is a relatively complex challenge that requires upfront planning to avoid unnecessary data conflicts when users change independent databases that must be synchronized. In addition some replication tools will require configuration per server which makes a multi-way active/active replication environment exponentially more complex with an increasing number of sites participating in the replication.

HVR is different. Thanks to its innovative hub architecture the setup of a multi-way active/active replication environment is surprisingly simple. Post product installation only a few steps have to be performed to enable the replication. With all systems equal in the environment configuration details are all defined at the Location Group level, with all active sites being a member of the group.

If you are facing an active/active replication challenge then you should take a look at HVR. Visit the website at <u>http://www.hvr-software.com</u> for more information or to request a free trial.

Supported sources	Supported targets
Oracle, all editions, including Amazon RDS	All supported sources
SQL Server, all editions, including Azure	Teradata
DB2 on Linux, Unix and Windows	Actian Vector (Vectorwise)
Ingres	Actian Matrix (ParAccel)
Flat files, including Hadoop	Pivotal Greenplum
Sharepoint	Pivotal Hawq
	Amazon Redshift