

HVR REALTIME DATA INTEGRATION

Reliable at any speed

White Paper

by HVR SOFTWARE BV

HVR is a robust and reliable database integration solution. Database changes are captured in one (or many) databases and streamed through to keep other replicated databases identical.

HVR's performance and scalability allows it to replicate very large databases in real-time. And its elegant design and rich functionality means complex replication schemes can be implemented with minimum effort. Strong tools are provided for comparing and refreshing replicated tables and for monitoring replication scheduling.

HVR Software by HVR Software bv, the supplier of HVR, is an Amsterdam-based IT company specialized in developing database software. HVR Software bv was founded in 1985 and initially developed applications for higher education institutions. Customers now include businesses in many sectors, such as logistics, telecommunications, public services, publishing and financial services.

HVR History HVR was originally developed for TNT Post, the world's 4th biggest logistics company. Replication was needed by various systems between branch and regional offices to make data available to the whole organization. Following HVR's successful implementation at TNT Post, the HVR product was launched.

- Features** The key features of HVR:
- Supports multiple DBMS's (Oracle, SQL Server, Ingres)
 - Log-based capture
 - Sub-second transactional replication
 - File replication
 - Easy to manage (low cost of ownership)

- Scalability across 1000+ databases
- High transport speed over WAN
- Ability to replicate between different data models
- 24x7 uptime
- Very flexible and customizable.

Heterogeneous Replication HVR can replicate Oracle, Microsoft SQL Server and Ingres databases. In addition to database replication, HVR can replicate files. This means file changes can be replicated alongside the stream of database changes. As well as replicating between databases of the same type, HVR can replicate between any mixture of the supported databases, for example replication from Oracle to Microsoft SQL Server or Oracle to Ingres. Such heterogeneous replication is performed without any reduction of HVR's performance or functionality.

How is HVR Used? HVR has a proven track record in business situations where there is need for quick and reliable database management. It is used in the following solutions:

- **Stand-by Database**
A stand-by database can be configured to avoid downtime due to an emergency on the production server or due to planned downtime. If an emergency happens the application can be re-connected to the stand-by database. The stand-by database can also be used to avoid planned downtime, for example while rebuilding table indexes or while upgrading the DBMS. By switching over to the stand-by database HVR can avoid downtime during these operations.
- **Workload Distribution**
HVR can replicate into a data warehouse or a reporting database. Regular users will use the production database, whereas the reporting database will be used for reporting and analytical purposes. The target database for workload distribution could be the same DBMS as the production database, or it could be a different DBMS.
- **Bridge between Databases**
HVR can provide a bridge between different databases. For example, between the internet database and back office database. These data-models could be quite different, so HVR must consolidate and transform the data during replication.
- **Enterprise Replication Bus**
As an enterprise replication bus, HVR is used as strategic middleware which connects all the business processes.

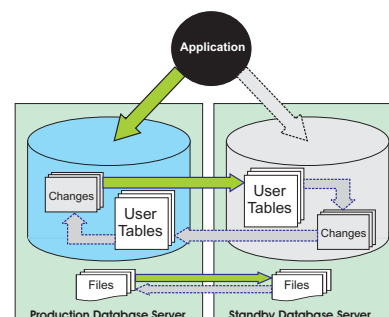
These uses will be described in more detail in the following pages of this white paper. The subsequent pages describe the technical architecture of the HVR.

Standby Database

Most companies have mission critical database applications. HVR can protect these systems by continuously replicating to a standby database. Operation can be switched to the standby database at any time; either in an emergency or perhaps just every night to avoid planned downtime. The standby database does not have to be in the same building. HVR's extreme efficiency over the network means it can easily be located over the WAN. This reduces the risk that an emergency which affects the main database will also affect the standby.

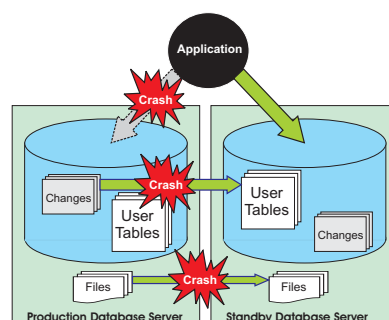
Replication to Standby

HVR replicates changes from the production database to the standby database keeping it up to date. Only changes are transmitted to the standby database not the entire database. Each change made to the database tables is queued. The queue is using either the DBMS's own system or by writing a row to a special capture table. The replicated change is then sent in a stream to the standby database. If there are files associated with the database then these can be replicated in a file replication channel alongside the database changes.



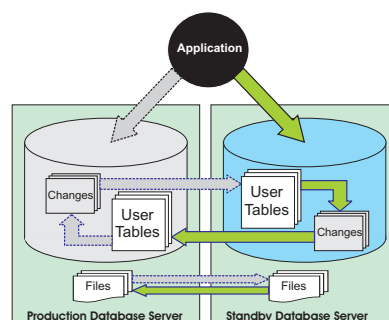
Emergency Switchover

If an emergency happens and the production database is no longer available, then the application can be immediately reconnected to the standby database. HVR makes sure that the standby database is up to date.



Replicating Back From Standby

Application users continue working on the standby database instead of the production database. If the production database becomes available again, HVR will replicate changes back from standby to the production database. At an opportune moment the DBA's may switch back to the production database. Only the DBA's will know whether the production database (machine A) or the hot standby database (machine B) is used. Application downtime will be virtually eliminated.



Standby Example

In one case a telecom company needed a system upgrade (new hardware, new DBMS version and an application revision). This would normally have required a conversion time of one week. Downtime this long is unthinkable for a telecom company. The company was able to quickly install HVR and the hardware for a standby machine, switch the user to the standby machine, perform the conversion on the production machine and switch their user back. Even the database's datamodel changed (HVR had to replicate between tables with different layout for this). The ambitious upgrade project was completed with only seconds of downtime. This meant that the company met its goal of 99.999% availability for that year.

24x7 Availability

In order to achieve true 24 hour x 7 day availability, various kinds of downtime must be eliminated.

What are the Benefits of HVR over Hardware Replication like EMC and Veritas

There are many kinds of emergencies. Both HVR and hardware replication systems (e.g. EMC and Veritas) protect against hardware failures such as a broken disk or burnt-out CPU. But hardware redundancy (Raid disk or extra CPUs) often guard against this already.

If power supply is interrupted then the standby could also be affected. Hardware replication systems use the network so inefficiently that it is prohibitively expensive to put the standby in a different building. HVR's standby can run over the existing network links and can be put in another town.

But the most serious cause of downtime is operator error or if a DBMS table becomes corrupt. HVR replicates at a logical level above these problems. So the replica will remain consistent. Hardware replication however will blindly replicate the inconsistency to the standby database.

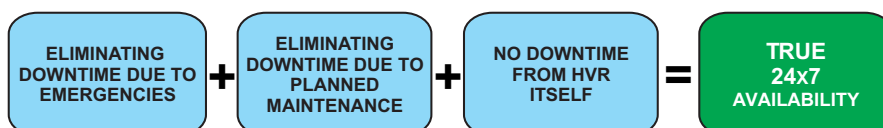
Eliminating Downtime Due to Planned Maintenance and System Upgrades

A database needs downtime in order to rebuild table indexes. Also the DBMS version and operating system will eventually need to be upgraded. By switching over to a standby database, HVR can avoid downtime when these time-consuming operations need to be performed. For upgrading the database application itself, HVR can also be used; its flexibility means that it can replicate between databases with a different table layout; some tables of the upgraded database may have new columns in old tables.

No Downtime for HVR itself

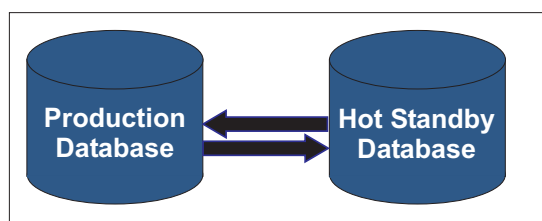
You don't need any downtime when installing HVR. Even for a large database, HVR is able to build a new replication target while at the same time end users are making changes to the original database. Sometimes HVR has to queue changes in database tables. Whether it does depends on which features are enabled. But even in this case HVR make sure that it remodifies these tables 'on the fly' so that you never need downtime for HVR itself.

Adding up to TRUE 24x7 availability



Another Standby Example

A typical telecom company needs 24x7 availability. All tables need to be replicated from production database to standby database. The hardware and database layouts are identical on both sides. Normally, replication is performed in one direction. When a switchover is done, the replicated data is momentarily flowing in both directions (bi-directional).



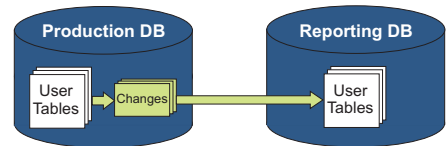
- Replication Distance: 26 km
- High transaction rate
- 200 replicated tables
- Bi-directional replication
- Provides TRUE 24x7 availability

Workload Distribution

Workload distribution aims at distributing the workload over several databases; for example, running reports on a reporting database or feeding data into a data warehouse. Regular end users will use the production database whereas the reporting database will be used for reporting and analytical purposes. This means that end user performance is not affected by reporting and reporting is not interrupted by end user locks. The target database for workload distribution could be the same DBMS as the production database, or it could be a different DBMS, e.g. reporting tools may run better on a Microsoft SQL Server database.

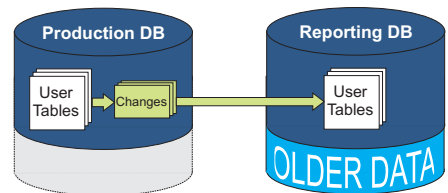
Basic Reporting Database

The reporting database could be on the same machine as production database or on a different machine. The reporting database can also be the standby machine.

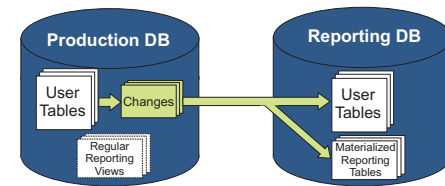


Purging Older Data From Production Database

Instead of copying all changes from production database to reporting database, HVR can be configured to ignore the purging of older data from the production database. The reporting database will increase in size, but the production database is kept small and runs faster. The purged data is still available in the replicated database for historical reporting purposes.



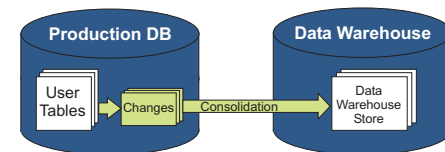
Reporting Database with Materialized Reporting Views



Often reports query into SQL Views. When these Views are complicated and the database is large, then the reports can run slowly because of the extra runtime joins. HVR offers the ability to run the reports against the Materialized Reporting Tables, instead of the SQL

Views. These Materialized Reporting Tables contain the same data as the Views, but they are real tables instead. When HVR replicates changes to the user tables, it also changes the data in the Materialized Reporting Tables. This makes reports run faster.

Replication to Data Warehouse



Changes made to the production database are consolidated and transformed and subsequently fed into the data warehouse. The business logic for consolidation and transformation is applied during replication; no temporary

tables are needed and no extra i/o is performed.

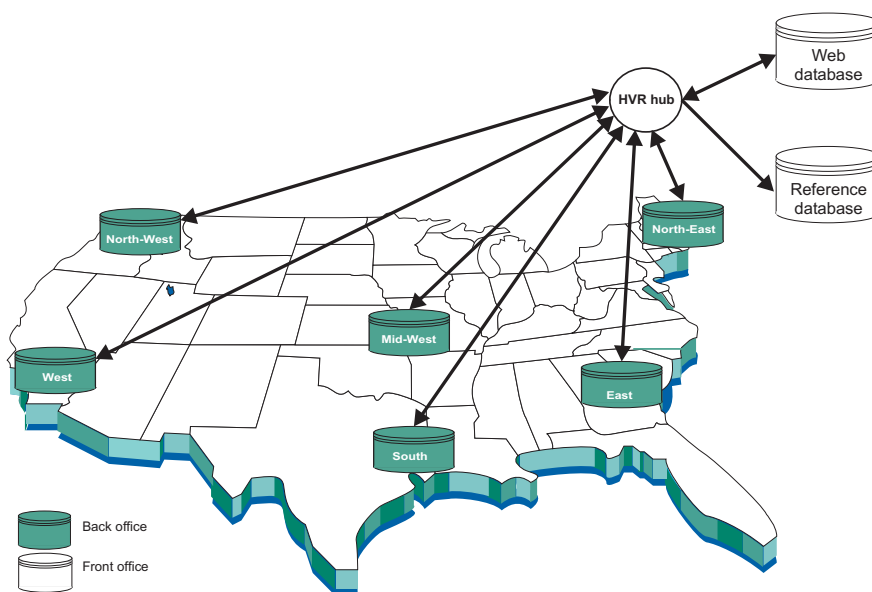
Bridge between Different Systems

For e-Services a bridge is needed between the backoffice database inside the firewall and the e-Commerce or web services on the other side. HVR provides this bridge and controls a backbone between the front office and back office. In this case the front and backoffice can be quite different; HVR must consolidate and transform the data during replication. The following requirements are essential for bridge to e-Services: replication must be flexible as it must transform data between different DBMS systems. The maintenance of these replication processes must be easy as these processes must evolve with the applications which they connect.

Connecting Many Databases In many cases, data from several operating systems need to be connected to a central database. Headquarters may need to gather daily sales information from local stores into a central database.

If many databases are involved, HVR automatically queues changes at the hub and distributes them into the target database. This hub functionality improves control over the different data streams. It also makes it possible to simply plug in new database targets without interrupting the existing data flow.

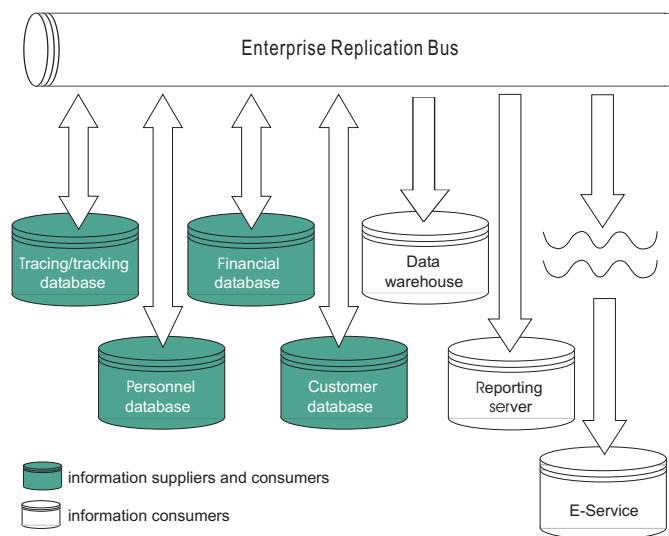
e-Services Example Another large telecom company has a billing database in each of its branch offices across the US. Their data must also be available in an internet database, so customers can access it to check their balances. Furthermore, the data must be send to a reference database for reporting and analysis purposes. HVR provides the solution by capturing each change and sending it to the Internet database and the reference database. An additional complication was that some databases were Ingres databases and others Oracle databases. HVR has to convert this data to fit the target database.



Enterprise Replication Bus

An Enterprise Replication Bus is when replication moves beyond being a simple connection somewhere in the organization. Instead data replication becomes strategic middleware connecting the business processes together. The bus is the implementation of a true replication strategy providing not only the technical tools but also the business framework.

Combining various replication categories, Enterprise Replication Bus creates one large databus. The data moves up and down using different replication techniques serving various purposes. It feeds for instance web services with operational data and the data warehouse with consolidated data. Data is shared in realtime across heterogeneous platforms and databases.



Replication Bus Example

A company selling computers on the Internet requires a good deal of data movement to have the process run as efficiently as possible. For each transaction, a credit card database must be queried, inventory levels reflected, shipping and receiving database updated, and perhaps a customer database loaded. HVR's replication software is capable of meeting all the data movement requirements of this enterprise replication bus, even across dissimilar hardware and software.

Database Capture

HVR provides two mechanisms to capture database changes. These mechanisms are log-based capture and trigger-based capture. This table compares their benefits:

	Log-based Capture	Trigger-based Capture
Mechanism	<ul style="list-style-type: none"> • Normally HVR reads from on-line logging. For Oracle this means redo log files and for Ingres this means the Ingres log file. • If HVR falls behind it jumps back to archived logging. This means the archive-redo files for Oracle and journal files for Ingres. • HVR only uses read access to DBMS files; it avoids the database locking system. • Physical logging records are converted to logical changes before being sent over the network. 	<ul style="list-style-type: none"> • Each replicated table has a pair of HVR capture tables. Every change is written into one of these capture tables by a HVR database trigger. • The replication job constantly 'toggles' these triggers so that they write into a different pair of capture tables. This toggling avoids contention with locks held by end-user and also protects transaction atomicity. • The replication job then processes all changes queued in the capture tables, exploiting the bundling to maximize throughput. • Finally the job truncates the capture queue tables and starts a new toggle.
Special Replication Properties	Only plain capture for specified tables, but special replication properties are still available on integrate side.	Various capture-side properties available including: <ul style="list-style-type: none"> • Collision handling for bi-directional replication. • Row capture-restrictions and horizontal partitioning. • Injection of business logic into capture triggers.
Overhead	Zero	Approximately 3% for interactive systems, increasing to 15% for batch work.
Replication Latency	Less than 1 second	About 1 minute

Managing Checkpoints, Journals and Archive Redo Files

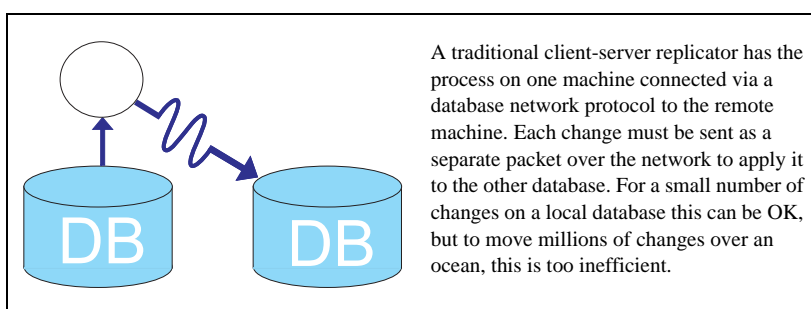
HVR also provides a DBA tool to manage the checkpoint, journal and archive redo log files. Each side has an existing backup/recovery regime that depends on these files, but HVR's log file may need to 'fall back' to reading the journals or archived redo-logfiles. The DBA tool will automatically purge these files only when they are no longer needed by the backup/recovery regime and have also been released by HVR capture.

No Overhead from Collision Detection

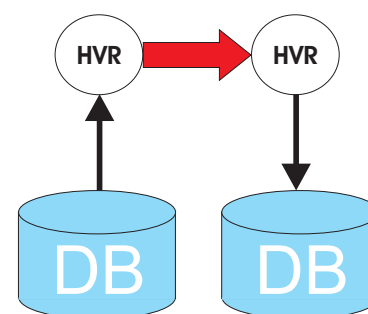
A collision is when a change is made simultaneously to two replicated versions of the same row in different databases. Some replication products need extra tables to detect these database collisions. Maintaining these tables during capture can impose significant replication overhead. HVR's collision detection is much more efficient; collision detection would only be enabled on the specific database and on the tables where bi-directional replication can occur. If the replicated table contains timestamp columns, then HVR is able to exploit these to detect collisions without needing to maintain any extra structures for collision detection.

Network Transport

To replicate data, HVR must transport changes between different databases, which could be on different machines. HVR's transport is very efficient, this is because of its advanced symmetrical architecture.



In contrast, HVR has a symmetric architecture. An HVR runtime process is running on each machine and is connected locally to each replicated database. The two HVR processes are connected to each other using a socket connection. This symmetrical process architecture is superior because it allows parallelism. One HVR process can be reading data from a database while the other process is writing changes into the target database.



Compression HVR can also perform compression. When moving data across the network, HVR uses a high-rate of compression. One process compresses data before sending it into the socket. Meanwhile the other process uncompresses it and puts it into the database. HVR's compression ratios are high (more than 90%) because it has tailor-made compression algorithms which exploits specific information about the compressed data which HVR knows. For example it knows the datatypes and how wide each table is. This beats traditional compression programs such as WinZip™. However, the key reason for HVR's efficiency over a network is the way it moves data: it is sending a whole queue of changes over the network as a single stream, bundling many changes into a single network packet, instead of sending each row in a packet and then waiting for a response packet.

Encryption When sending data across a network, HVR also has a feature to enable SSL encryption. This allows HVR to replicate data securely over a public network.

Coalescing Changes to the Same Row If multiple changes are made to the same row in a short time, these can be coalesced. Coalescing means, for example, that an insert and two updates to the same row can be merged into a single insert. Enabling HVR's coalesce option reduces the number of changes that need to be transported or applied, which further boosts replication speed.

Minimal Replication Hops Another explanation for HVR's speed is that it minimizes 'replication hops' using its streaming architecture. A replication hop is when changes touch disk between the capture database and replication target. Multiple hops incur disk I/O which adds to overhead. HVR does need to queue changes on disk, so the system can keep running if one target database is offline. But HVR only does this once and the queues are highly compressed. So every megabyte of captured data HVR will perhaps use a 100 kilobyte hop.

Reliability

24x7 Uptime HVR is designed to guarantee the strength that is essential for this kind of mission-critical program. It uses a variety of techniques to ensure that replication is continuous and uninterrupted. One technique is independent jobs. If data is being replicated to many databases, then the replication for each database is handled by an independent job. If that database is unavailable, then that specific job will fail, but the other replication jobs will continue. This insulates other physical objects from errors in other physical objects. It is also possible with HVR to group tables that need replicating into replication groups called channels. A channel is a group of tables that need to be replicated together, for example the product list tables could be replicated separately from the order tables in the same database. If multiple channels are defined for replication, then HVR creates a separate set of jobs for each of these channels. If an error occurs when replicating one table, then the set of jobs for that table will fail, but the jobs for the tables in the other channel will continue replicating and won't be affected. This insulates the logical elements in the system from failure in other logical parts of the system.

Collision Resolution Data integrated into target tables can be erroneous due to the following reasons:

- Database collisions due to bi-directional replication
- Database errors
- Application errors
- Operator mistakes.

Users can choose from a range of error or collision resolution policies to detect collisions efficiently and resolve them. The selected policy is followed whenever errors or collisions occur. HVR offers a wide range of standard policies.

Avoiding Locking Contention HVR uses a variety of locking techniques to ensure that there is absolutely no locking contention between HVR and end users making changes. One strategy is dual buffering. This is the toggle system used for capturing changes, which ensures that if end users are making changes to one table, HVR is always busy working on a different table where no end users are active.

Avoiding Distributed Deadlock Another locking issue is distributed deadlock. This is where two users attempt to make changes to different objects, but make the change in such an order, they have to wait for each other. Normally, if both users are working in the same database, then the DBMS will detect it automatically, but if a deadlock happens during replication, then no DBMS can detect this and replication can potentially hang further. To solve this, HVR analyses all the replication flows before configuring replication. At this time it is able to analyse any potential locking loops, which would deadlock and reconfigure replication topology to ensure that distributed deadlock is absolutely impossible.

Guaranteed Full Database and Transaction Consistency HVR guarantees the consistency and integrity of the replicated data using the following techniques:

- The HVR uses its own special two phase commit protocol; this variant is non-blocking so that it doesn't impact the system's uptime.
- Replicated databases include mechanisms that confirm replicated data has arrived and is integrated.
- Changes are not replicated until they are committed.
- The changes are re-played on the target database in the same order that they occurred on the capture database.

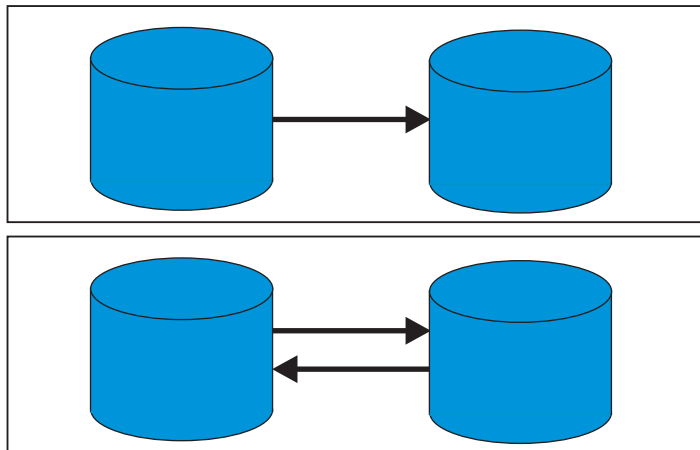
Replication Topology

One-Directional or Bi-Directional Replication

HVR is able to replicate both in a single direction - all changes are replicated from a master database to a slave database, or bi-directional way. Bi-directional means that changes can be made in several databases and the replicated data travels in both directions.

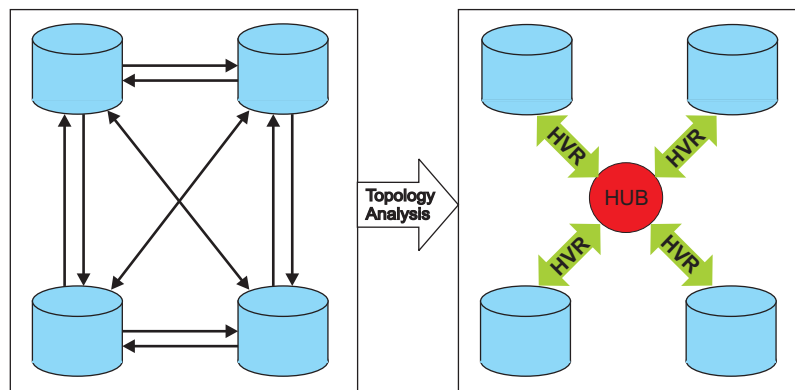
Point-to-Point Topology

For simple dataflows, e.g. data going from one database to a second database, HVR bypasses: data is sent straight between the replicated databases without a hop through another database.



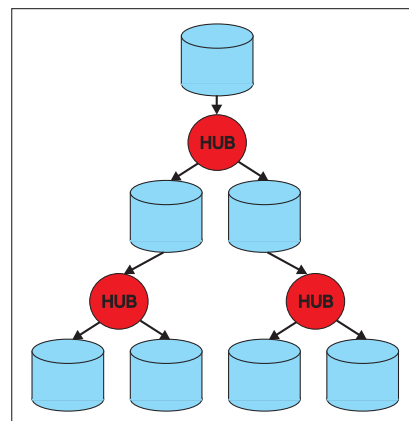
Routed Topology

For more complicated dataflows, e.g. when replication needs to move in a mesh between many different databases, HVR routes the data through a router point. The router is a queuing system, changes from each of the databases are directed into this queuing system and from the queuing system the changes are then moved onwards to the target databases.



The benefit of routing when many databases are involved, is that it simplifies the operator's job; the spaghetti of replication streams becomes vastly simplified. As well as improving operator control, the router model makes it possible for new replication targets to be plugged into an existing replication scheme without interrupting the rest of the application.

Cascade Replication If replication involves many databases and travels over long distances, cascade replication can also be configured. This means that changes are integrated into one database and then re-captured again so the changes cascade into another replication scheme. Cascade replication is more difficult to manage than a routed topology, so routing is often recommended instead when it is feasible. When cascade is used it is often used in combination with routed replication (as illustrated in the diagram).

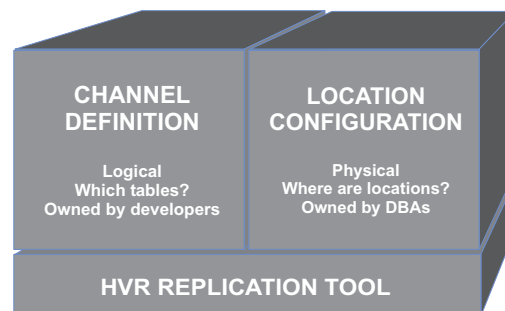


Change Control

Channel Definition and Configuration

Designing and implementing the replication of a data source or database is a highly structured process. It is divided in the following two areas:

- Channel definition. Contains the functional requirements. It answers questions such as: which tables must be replicated? How must these tables be replicated?
- Location configuration. Includes the technical parameters such as the actual network node names and the actual database names.



This division also applies to designing and implementing changes to existing replication processes. As a result developers can concentrate on their developing tasks and test their replication designs, while database administrators can focus on their administrative tasks

Quality Assurance for Replication Definition

The division of the channel into definition and configuration halves is important for testing. Replication can be tested on a test version of the database before it is put into production. Nobody wants to use drag and drop to choose tables for replication in the middle of the night on a production database. Instead, all of this information can be tested by the developer, passed to the acceptance testing group and when quality assurance is completed it can be applied to the production environment. The only information which needs reconfiguring by the operators is the physical information such as the database name and the network addresses. The application-specific information: the list of tables that need replicating, is delivered to them as a package and is not their responsibility.

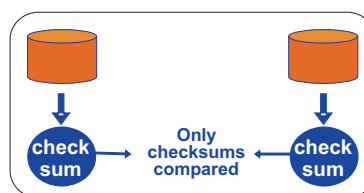
Example

A software supplier builds and delivers a complete Student Information System (SIS) for colleges and universities. This product requires replication both internally and to share data with other applications running at the customer. The software supplier is able to build an HVR channel definition to replicate its application and sell this as an extra module for the SIS package. To install this replication module, the college or university would not need any detailed knowledge of the tables in the SIS application or knowledge about setting up HVR. The only thing that would have to be reconfigured on each site would be the channel configuration, e.g. database names and network addresses. The replication module is sold to each customer as a blackbox package.

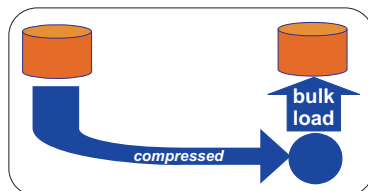
Refresh and Compare

As well as actual replication (capturing each change and applying it to another database), HVR supports refresh and compare. Database refresh is when two databases need to be brought into sync with each other. This is necessary before replication is enabled and may also be necessary if a disaster occurs to one of the replicated databases. Database compare allows the operator to see whether two large databases are identical or not. This is an important and handy tool for the operator. It could be that external auditors require an independent check that two independent databases are actually in sync. Also if replication has been running for a long period of time (years), it would still be sensible to periodically do a compare between the databases to double-check that they are identical. HVR supplies two flavours of refresh and compare.

Bulk Compare Bulk compare means that HVR performs checksums on each of the tables in the replication channel. Only once these checksums have been built are the numbers compared. The actual data does not travel across the network so this is very efficient for large databases over a WAN.



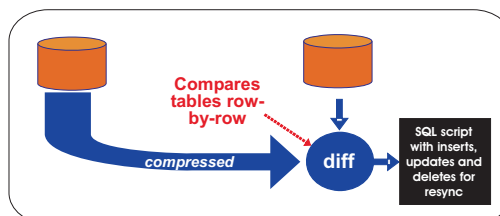
Bulk Refresh



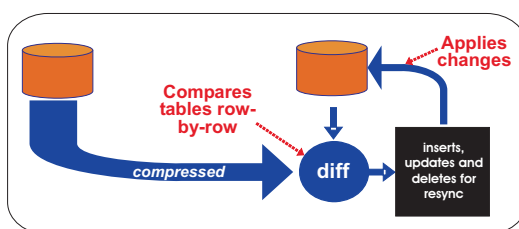
Bulk refresh means that HVR extracts the data from all the tables in one database, compresses it, brings it across the network, uncompresses it and loads that data into the target database. After the data has been bulk loaded into the target database, the indexes are reinitialized. The bulk refresh for different tables happens in parallel, so this operation is extremely fast and efficient.

Row-Wise Compare

Row-wise compare means that HVR extracts the data from one database, compresses it and on the target database it compares those changes row by row with the changes in the target database. For each difference detected an SQL statement is written: an insert, update or delete. The resulting SQL script will tell the operator which steps are needed in order to resync the two databases.



Row-Wise Refresh



Row-wise refresh is the same as row-wise compare, the data is brought across the network compressed and compared row by row. But the resulting SQL statements are applied directly to the target database in order to resynchronise the tables.

Very Fast and Very Flexible

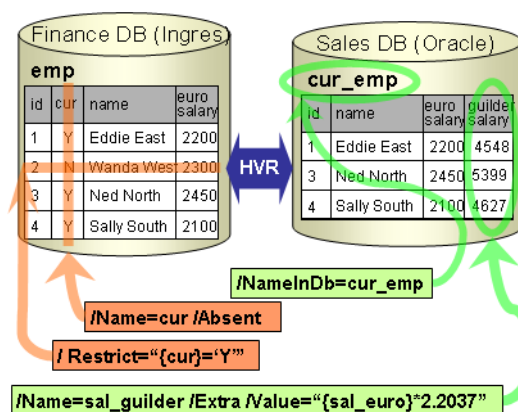
All of these flavours of refresh and compare will work faster than normal refresh tools if the two databases have the same DBMS and have the same table layout. They will work just as fast if the databases have different DBMS's (one database is Oracle and one database is Ingres). Even if the database layouts differ, for example one table has more columns in one database than in another database, then HVR is still able to perform its refresh and compare.

Replicating between Different Database Types

Replication is still possible between two databases even if their data models or DBMS's differ. For example if one database is Oracle and the other is Ingres, or if the tables have different names or different number of columns in the two databases.

Replicating between Different DBMS's

Performance is unchanged regardless of whether HVR is used in a homogeneous environment (databases from the same vendor) or in a heterogeneous environment (databases from different vendors). Also all of HVR's features are still available in a mixed environment. This includes bi-directional replication and collision handling. You do not need any DBMS gateway or special ODBC connection to join the two different types of databases. HVR connects directly to each database and performs any necessary data type conversions automatically.



Restrictive Replication

HVR has an option to replicate only some changes in a given table but to ignore changes to other rows. This can be used, for example, to allow old rows to be purged from a production database but to be kept in a reporting database. This feature can also be used to partition a central database so that different sections of each table are replicated into different decentral databases.

Extra, Missing or Renamed Columns

Replication can be configured to know that the table or column name in one of the replicated databases differs from the name in the other database. HVR can also replicate between tables where one table has extra columns, or some columns are missing. A default value or SQL expression can be configured to calculate a new value for the missing column.

Rolling Upgrade

HVR's ability to replicate between databases with different data models can be used for rolling updates. Typically an application upgrade involves adding tables and adding and removing columns to existing tables. A rolling update is where instead of upgrading all the databases simultaneously, only some of the databases are upgraded. During the rolling upgrade HVR is used to replicate between the new databases which have been upgraded and the databases which have not been upgraded. A rolling upgrade can be used to minimize downtime during maintenance work.

Transforming and Consolidating

Replication can be configured to transform or consolidate data during replication. No temporary tables are needed and such transformation or consolidation has no affect on replication performance.

Refresh and Compare

HVR can not only replicate between databases which differ in the way described on this page, but it can also compare and refresh between such tables. HVR refresh and compare take account of the renamed columns. Automatically convert between the DBMS datatypes and will cut or paste extra columns as appropriate. None of these operations have any effect on the performance.

Maximum Operator Control

The HVR is an easy-to-use application that can be adapted to a wide range of requirements. Maximum control means:

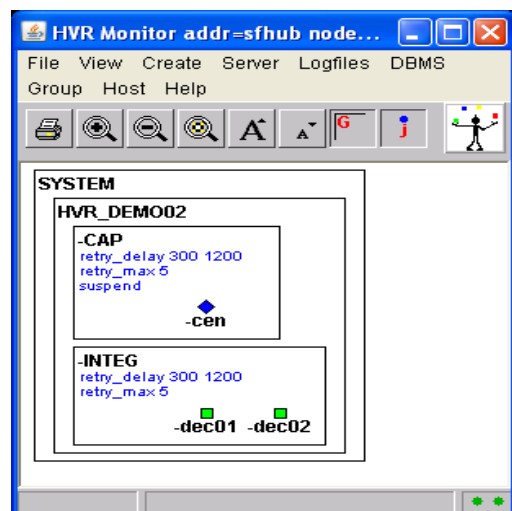
- Change control and ease of implementation
- Easy operator control
- Flexibility.

Sophisticated Scheduler A sophisticated scheduler allows database administrators to tune performance, to balance application or system loads, to schedule jobs, and to monitor and control job execution. Jobs can be scheduled by time intervals or linked to the occurrence of particular events.

As data can be transmitted over different network segments, data transmission can be organised to avoid overloading busy network segments or to use one particular segment for more than one transmission at the same time.

Single Point of Control A single point of control centralizes the administration, scheduling and control of jobs. All replication changes are affected from this central point. The replication catalogs are also centralized in this one hub database.

HVR Monitor For maximal operator control, HVR has a graphical user interface. This user interface is implemented as JAVA applet, which can run inside any browser (e.g. Internet Explorer). The JAVA applet provides a realtime animated overview of all the replication jobs running for replication. This provides a single point of control regardless of how many databases are involved or whether the data is going inwards, outwards or in a mesh topology.



Animated Overview of Replication Each replication job is represented as a small square, which changes colours according to its state. When data is travelling through HVR the replicated job becomes green. If a replication error occurs, the job will become pink (retry error) and if retry fails, it will eventually become red (failed). The state changes of a job are displayed realtime to the operator.

If the tables in a replicated database has been grouped into channels then the jobs for each channel will appear in separate groups in the user interface. The jobs for each channel will be stacked up on top of each other. The JAVA applet provides a number of functions. Firstly, it provides an easy overview, so that an operator can see at a glance the state of replication: if jobs are running too slowly, whether the system has been suspended.

Drag and Drop Control A function of the applet is to control the scheduling of replication. Drag and drop can be used to configure replication scheduling, e.g. a timeout attribute can be dropped onto a job in order to send an alert if a replication job's performance is

below a certain agreed service level. Drag and drop control can also be used to suspend jobs or to control job retry for network errors.

Drilldown Problem Diagnosis If a problem happens then the operator can click into the applet in order to see the replication logfile for that job. It can also check the DBMS logfile, perform a network ping or telnet onto the affected machine. This gives the operator a headstart so he can zoom in on a problem and quickly analyse whether the issue is in fact from replication or whether the problem occurred because of a application problem, network failure or an operating system error.

Interface with Enterprise Monitoring System The HVR would normally be configured to automatically raise errors into the enterprise's network management system where they will be handled according to the enterprise's normal processes. Various network management systems are supported, including Unicenter NSM and OpenView. The HVR monitor doesn't replace this enterprise monitoring system, rather it complements it as a handy adhoc way for the operator to monitor replication.

Platform Support

HVR supports various platforms and DBMS's. Mixed operating systems and DBMS's can be used in one implementation. All of HVR's features, performance and reliability is still available in a mixed environment.

Supported Operating Systems The HVR supports the following heterogeneous networks:

- HP-UX for PA-risc and Itanium
- Solaris for Sparc
- IBM AIX
- Linux for x86 and x86-64
- Microsoft Windows for x86 and x86-64
- OpenVMS for Alpha

Supported Data Storage The HVR supports the following data storage:

- Ingres
- Oracle
- Microsoft SQL Server
- File replication

“Objects connected with replication must not GAIN dependencies: instead replication must be a gateway, allowing data from heterogeneous sources to be combined MORE EASILY”.

Further Information

For further information, please contact:

Email info@hvr-software.com
Internet: www.hvr-software.com

HVR Software bv
Haaksbergweg 45
1101 BR Amsterdam
The Netherlands

Telephone +31 20 312 7512
Fax +31 20 312 7509