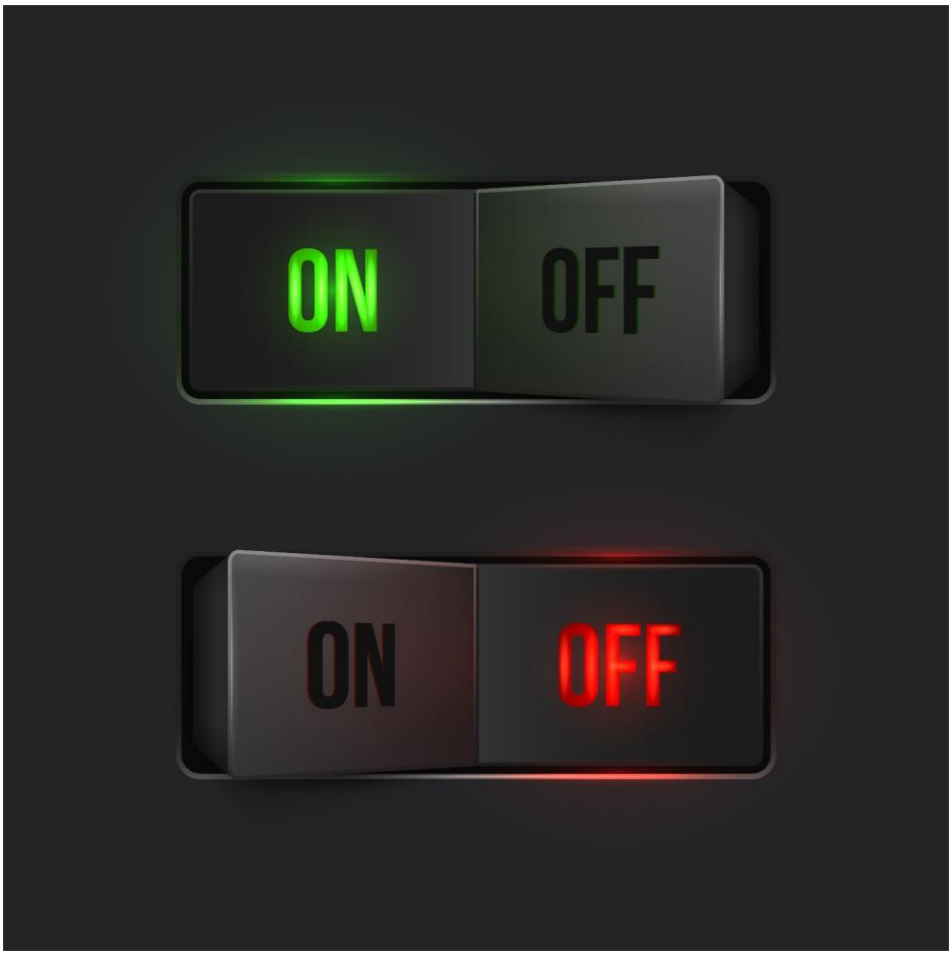
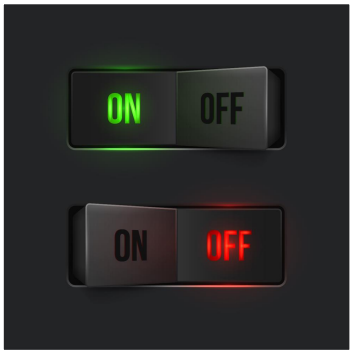
Switching Replication Engines with Zero Downtime and Less Risk

A Gravic, Inc. White Paper



Executive Summary



A mission-critical application often runs on redundant systems to ensure that it is always available to its users. This type of system may be configured as an active/passive pair, in which one system actively runs the production workload while the other system is passively standing by, ready to take over application processing in the event that the production system fails. Alternatively, the system can be configured as an active/active pair, in which both systems are actively processing transactions. In either configuration, a data replication engine is used to keep the databases of the two systems synchronized.

Sometimes, companies may decide either to switch data replication engines or to upgrade to a new version of the existing data replication engine. With mission-critical applications, it is necessary to do so without taking the applications down – a *zero downtime migration* (ZDM). Furthermore, it is imperative that a backup copy of the database is always available, and ready to take over if the production database fails. Additionally, the backup database must be kept synchronized with the production database while the data replication engine is being migrated (where no test node is present).

HPE Shadowbase software offers several methods, approaches, and techniques to solve these problems. One preferred approach is *Version Independence*, which completely avoids interoperating between different software versions. This scalable method works well for companies with only two systems available through proof and verification steps, where the test can occur over time and encompass an unlimited number of tables. This solution works well for production environments with limited resources.

A better solution exists for environments with three nodes. This uni-directional solution keeps the proof and verification process independent from the production environment until much later in the migration process. An old disaster recovery standby (DRS) system is always available for production failover with the original configuration and working database while the new environment is built and tested.

For companies seeking a full hardware refresh of both the production and original DRS systems, an entirely new production/DRS uni-directional configuration can be deployed and tested with four nodes. This solution offers the least amount of impact to existing production, far less risk, and only affects the production system much later in the migration process, yet is the most complex solution.

In bi-directional environments, all users can be moved from one system to another without changing the data replication engine's configuration, and all environments are automatically synchronized, meaning all nodes are kept synchronized during the entire process. The project is typically more complicated, since each vendor has its own algorithm for managing the bi-directional data oscillation problem, and data replication engines do not usually interoperate properly to avoid oscillation. This environment poses challenges and issues that must be anticipated and properly handled.

The *jagged edge* problem occurs for all replication engine migrations. How does a company update versions of a data replication engine during a switch-over and ensure the new replication engine matches the exact point where the switch-over occurred? A thorough understanding of the algorithms used by both engines is needed, so that all data is properly replicated once, and only once. This paper discusses the two most common methods for handling the transition from one replication engine to another: The *Brute Force Method* and the *Transactional Replication Method*.

Switching replication engines with zero downtime is a topic with immense complexity. In this paper, we discuss how a data replication engine can be changed/upgraded without taking either the application or the database offline. This approach improves availability over other methods that do require an outage, and eliminates the inherent risk normally associated with this type of change.[1]

[1]For additional information, please see the related Gravic white paper, *Using HPE Shadowbase Software to Eliminate Planned Downtime via Zero Downtime Migration*.

Table of Contents

Table of Figures

Switching Replication Engines with Zero Downtime and Less Risk

Redundant Systems

In a general case, the need to replace or upgrade a data replication engine can occur on a single system (e.g., a local data integration replication environment), as well as across systems (e.g., a classic business continuity architecture). This paper focuses on the more common case of intersystem data replication environments; however, the approach applies to any data replication environment.

As shown in Figure 1, changes to the active database in an active/passive configuration must be replicated to the database of the passive system so that the two systems are always synchronized. These changes are replicated with a uni-directional data replication engine.

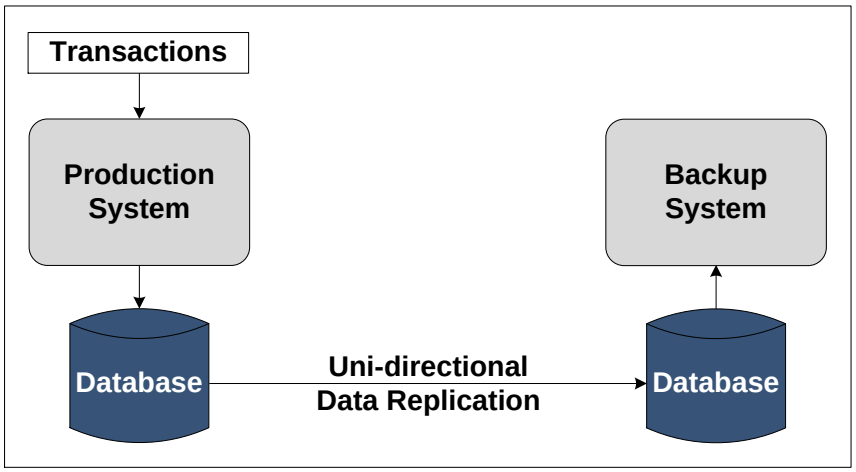


Figure 1 – An Active/Passive Redundant System

In an active/active configuration, changes made to either database are replicated to the other database via bi-directional replication, as shown in Figure 2.

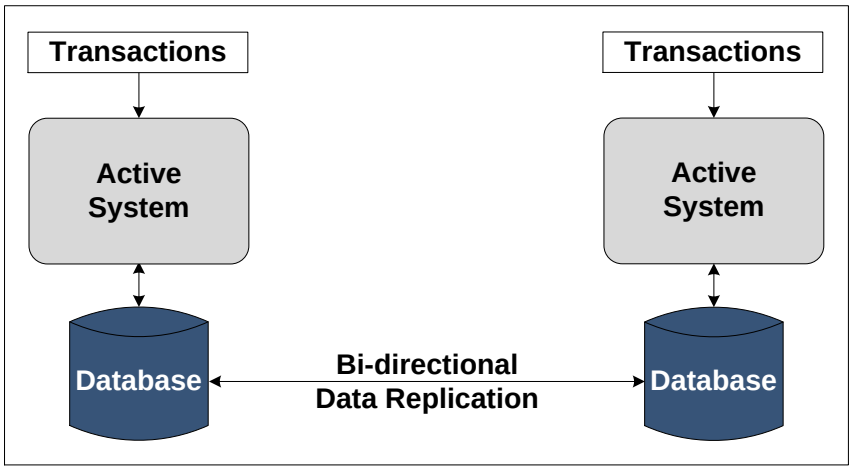


Figure 2 – An Active/Active Redundant System

Changing the Replication Engine

There are cases where the replication engine may need to be changed/upgraded. For instance, the need may arise to migrate from the HPE NonStop Remote Database Facility (RDF) that supports an active/passive system to the HPE Shadowbase product suite to upgrade to an active/active system. Alternatively, it may be necessary to upgrade from one version of an HPE Shadowbase replication engine to a more recent version.

If the application is critical to the users of the system, it is important to migrate from one data replication engine to another with no application downtime – a zero downtime migration (ZDM). It is also important to do so in a

manner that reduces/eliminates the risk inherent in this change; the ZDM approach satisfies this need. In this paper, we focus on upgrading HPE NonStop system environments, although this approach applies equally well to other platform/database data replication environments.

Interoperating Versions – A Dangerous Mix

In HPE NonStop systems, changes to the source database are appended to an audit trail set of files by the NonStop Transaction Management Facility (TMF) for audited databases. Figure 3 shows version 6100 (v6100) of a data replication engine reading the changes from the source database's audit trail and sending them to the target system to be applied to the target database. In this way, the target database is kept synchronized with the source database.

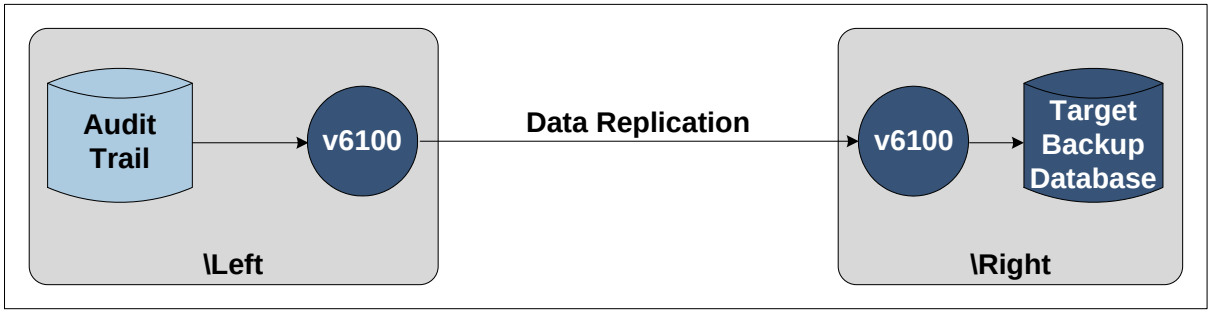


Figure 3 – Replicating Data from a Source Database to a Target Database

The concept is the same for other transactional databases. For example, an Oracle database provides a redo log and DB2 provides a journal log that contains all changes to the source database that can be used as a source of input for a replication engine.

Warning: Do Not Attempt!

Figure 4 shows what may appear to be a straightforward solution to this migration *(it is not, and should be avoided, as will be shown)*. Assume there are two versions of a data replication engine – version v6100, and version v6400. Version v6100 is the current version of the replication engine. The objective is to migrate to version v6400.

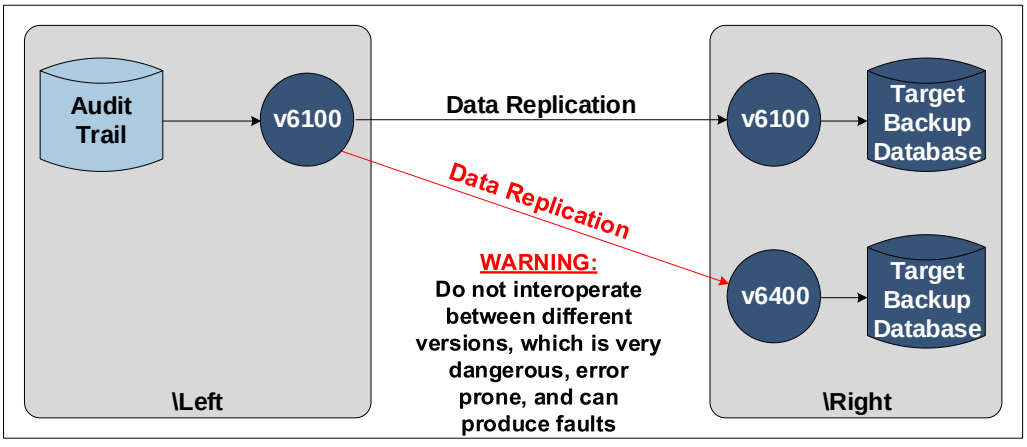


Figure 4 – Data Replication Engine Migration by Intermixing Versions

As shown in the figure, the migration can be accomplished by installing version v6400 on the target system and beginning data replication from the original version v6100 on the source system (System \Left) to the new version v6400 on the target system (System \Right).

When the database has been migrated from being replicated by version v6100 to being replicated by version v6400 to the target system, version v6400 can be installed on the source system, which then can take over from version v6100 and continue replication (see Figure 5).

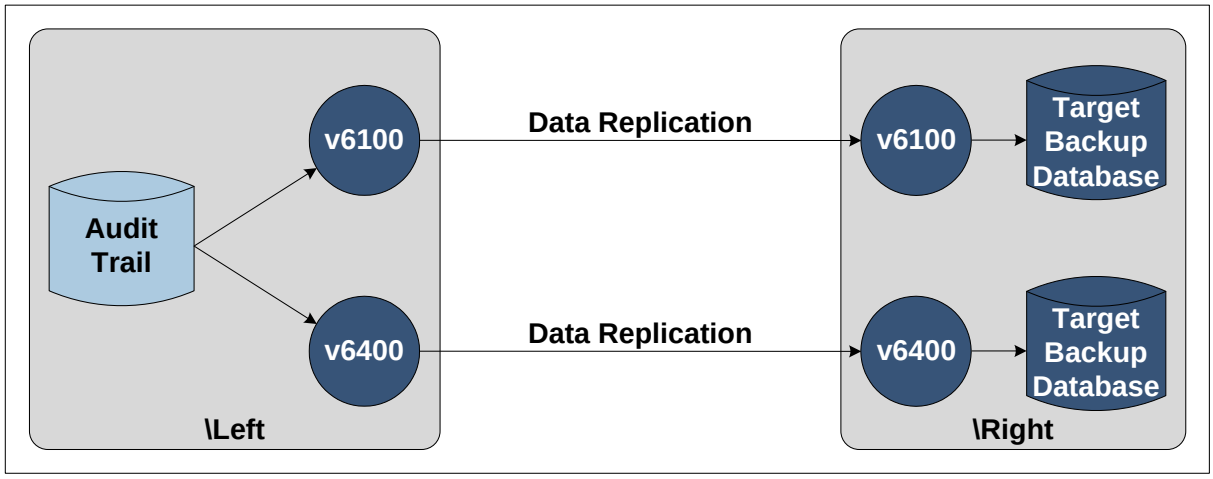


Figure 5 – v6400 Takes Over Replication

Once the database has been completely moved over to version v6400, version v6100 of the data replication engine now can be retired (Figure 6). Replication has been migrated to version v6400 without having to take down the application.

However, this migration requires either different data replication engines or different versions of the same replication engine to interoperate with each other, as originally shown in Figure 4. Such a sequence is fraught with peril, since the various versions have to be installed, tested, activated, and then interoperate for various time periods during the process. It is also possible that replication errors will arise, since the different versions were not necessarily intended to be interoperable. Additionally, this sequence requires a series of changes to both the source and the target data replication environments, each of which can introduce errors and impact the current production environment if not performed properly. Testing this sequence of changes ahead of time is tricky and time-consuming.

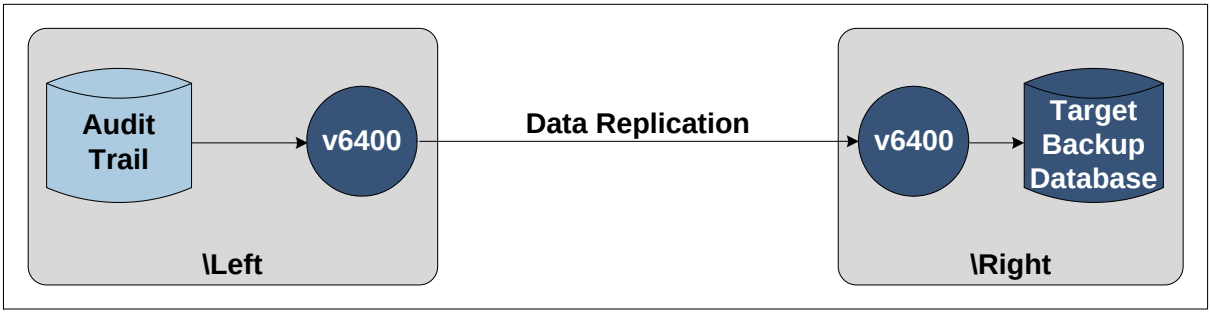


Figure 6 – v6100 Is Retired

For instance, with HPE Shadowbase software, approximately two releases were produced each year during the past decade. It is impractical to certify each previous version of HPE Shadowbase software interoperating with each newer version and vice versa. The coding changes required can be quite extensive to revise the message structures upward and downward as well as the effort to fully exercise and test each combination. A similar situation exists if the data replication products are from different vendors. It is a risky business having them interoperate at all. Additionally, the relative usage of each combination would be for a very small population of the installed base – it is limited to only those that want to attempt this form of upgrade.

Therefore, the interoperation of different data replication engines or their versions is a dangerous procedure. It is not known how well they will work together or if they even will. It is likely that this configuration will be error-prone and will produce replication faults. And, training the staff on the two versions can be confusing.

So, having discussed what not to do, what is the recommended approach? A method is needed to install and configure a new parallel replication environment alongside the previous/existing replication environment so that the new environment can be tested and certified while the existing environment continues to run. Once trusted, the new environment can slowly “take over” the replication load from the previous environment until

all traffic has been moved to the new environment. At that time, the previous environment can be shut down and decommissioned. This method is the essence of the zero downtime migration process, and avoids the pitfalls of other migration approaches.

Version Independence – The Preferred, Best Practice Approach

An alternative migration that does not require version interoperation is reflected in the following figures. These examples assume an application with 1,000 tables initially is being replicated by version v6100 of the data replication engine.

Version Independence – Detailed Steps

As shown in Figure 7, a test environment for version v6400 is established before migrating to version v6400. Of course, an initial checkout of the new replication engine (version v6400) should be undertaken in a non-production test environment before the sequence begins. The following sequence assumes that the initial checkout has already been completed and describes one approach for replacing the previous version (v6100) in the production environment. These examples assume that there are only two working systems, system \Left (production) and system \Right (standby).

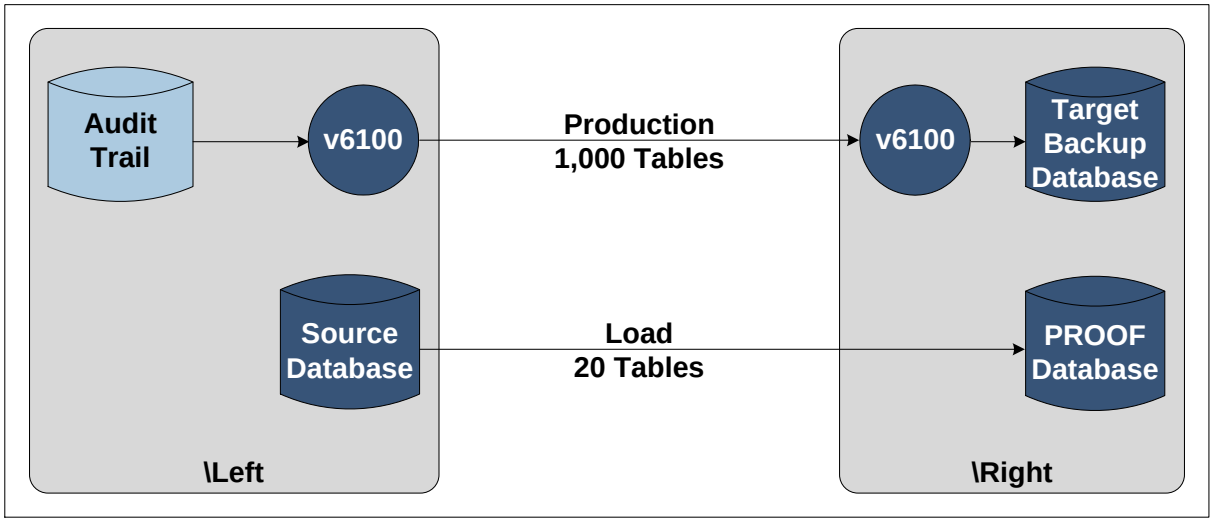


Figure 7 – A PROOF Temporary Target Database with 20 Tables

A tentative target database (PROOF) is created, and a representative sample of 20 tables are loaded from the production database into PROOF. The tables can be loaded offline via a tool (e.g., backup/restore), partially online via a utility (e.g., FUP or SQL LOAD on an HPE NonStop system), or online via the HPE Shadowbase Online Loader (SOLV), which is an online loading utility for loading source files/table into target files/tables while the application continues to run. SOLV thereby allows both the source and the target databases to be online and active while the source data is copied into the target environment.[2] As shown in Figure 8, data is replicated by the new data replication engine (v6400) to the 20 tables in PROOF, and the updated data is compared to the production copy of the target database. A database comparison tool like the HPE Shadowbase Compare product can be used to verify the databases; it examines and compares the databases online while they are being updated.[3]

[2]For additional information describing offline vs online loading and the Shadowbase SOLV product, please see the *Shadowbase SOLV Product Suite Brief*.

[3]For additional information describing the Shadowbase Compare product, please see the *Shadowbase Database Compare Technology Solution Brief*.

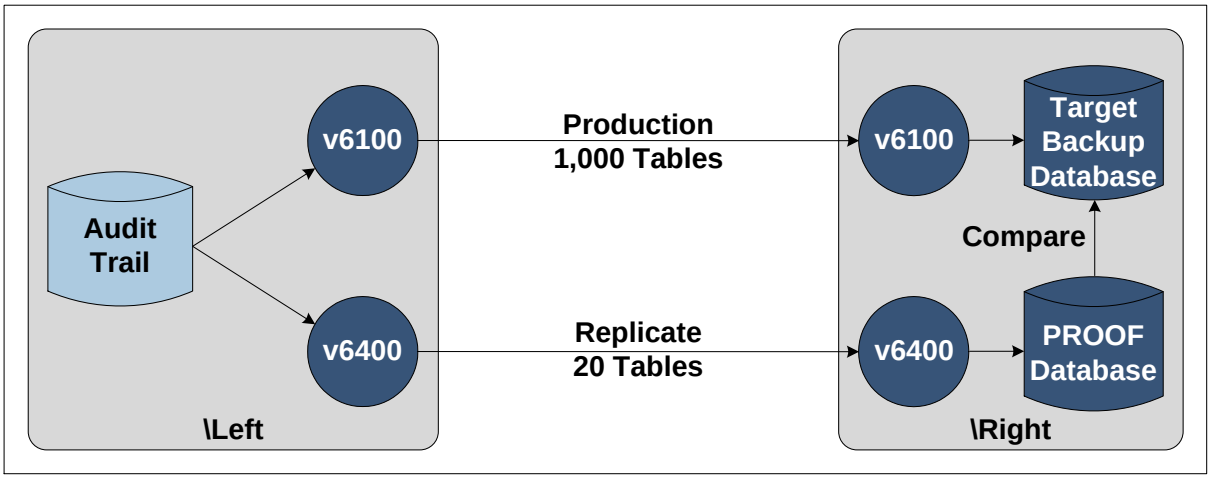


Figure 8 – Test the Replicated Temporary Target Database

PROOF replication is run for a time period to ensure that the new version v6400 of the replication engine is operating properly in this quasi-production environment. If the results do not match, the discrepancies must be resolved and the test rerun. If the replication test is successful, it could be expanded to more tables, (e.g., 80 tables, as shown in Figure 9).

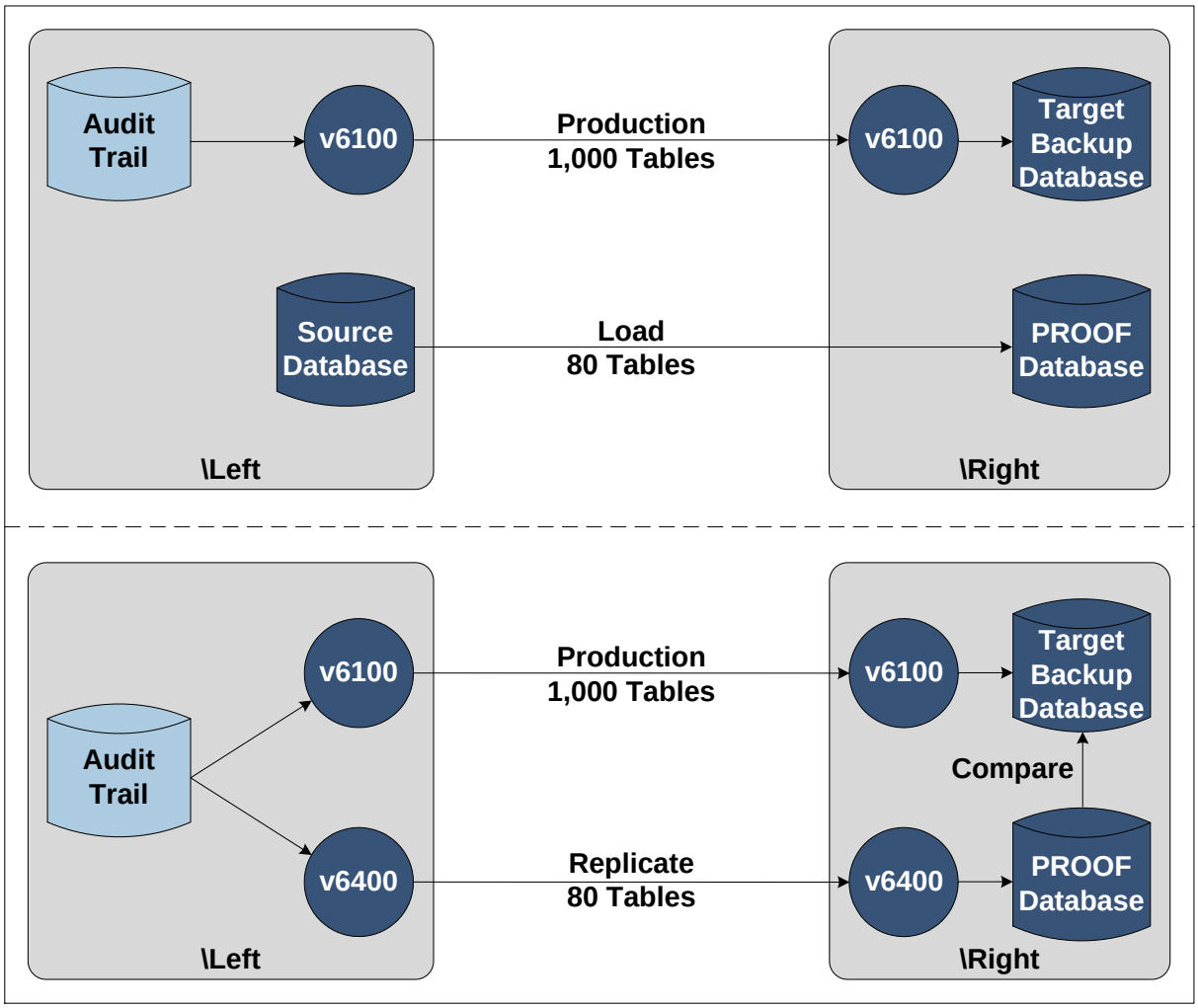


Figure 9 – Replicate 80 Tables to PROOF and Compare

If the 80-table replication test works properly for a time period, then the new version v6400 of the data replication engine is now validated to be working properly, and therefore, is a trusted and *known-working* environment. If disk space is at a premium, the successfully compared tables in the PROOF database can replace the original target database tables, and they can be removed from the version v6100 configuration. This step allows the data to be moved over to the new environment a few tables at a time, which reduces the risk of a *big-bang* (all-at-once) migration. The HPE Shadowbase product suite allows a migration of the database objects from one replication configuration to another, gradually over time, with minimal impact to the current replication environment. Only the files/tables that are being migrated are impacted. The new environment can be verified each step of the way to certify that the new environment is properly functioning before cutting over to it.

As shown in Figure 10, the test then can be expanded to replicate all 1,000 tables. If it is successful, the PROOF database can replace the production database, the data replication engine version v6400 can replace version v6100, and version v6100 can be retired, as shown in Figure 11. The 1,000 tables now are being replicated to the target system with the new data replication engine version v6400.

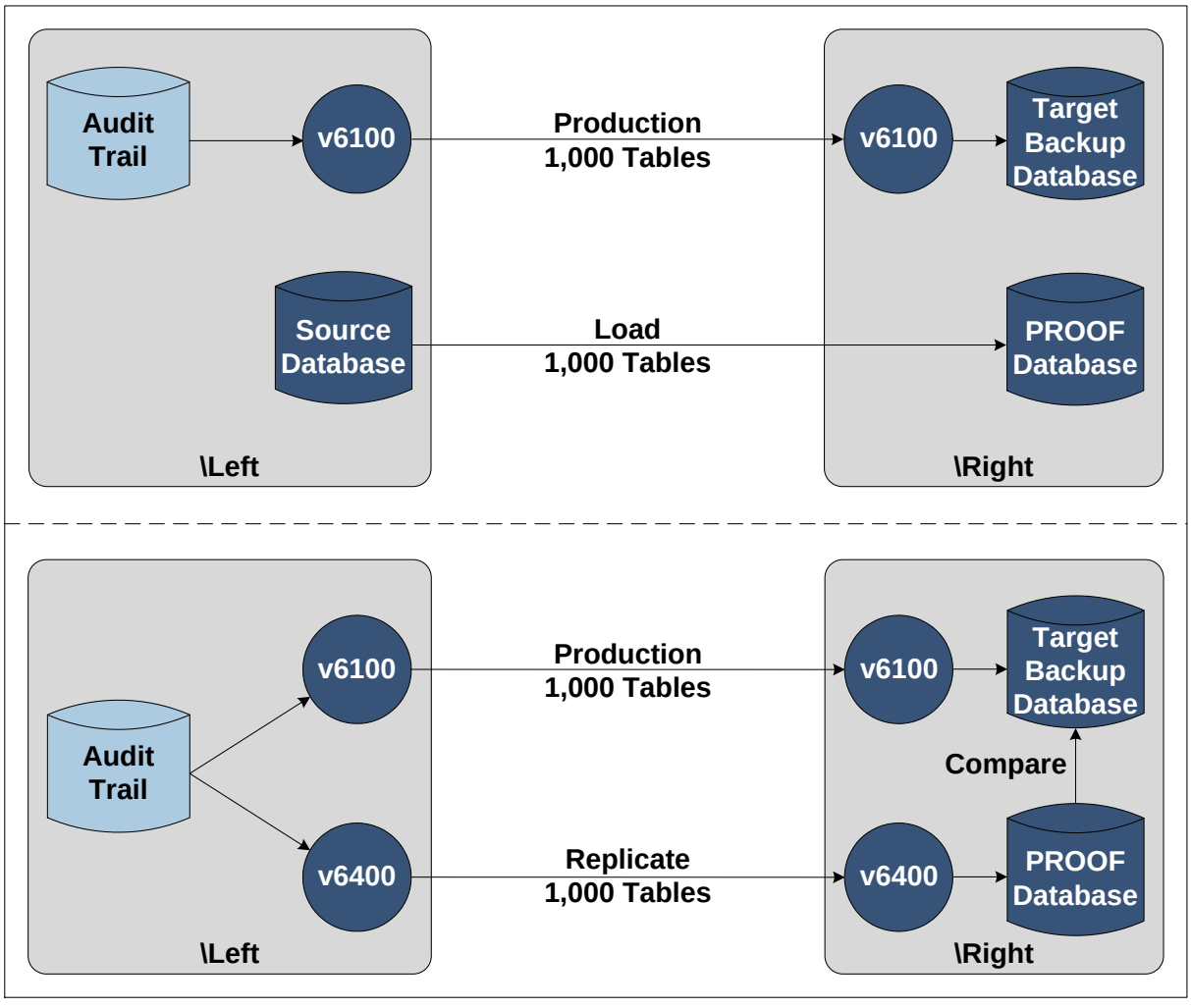


Figure 10 – Replicate 1,000 Tables to PROOF and Compare

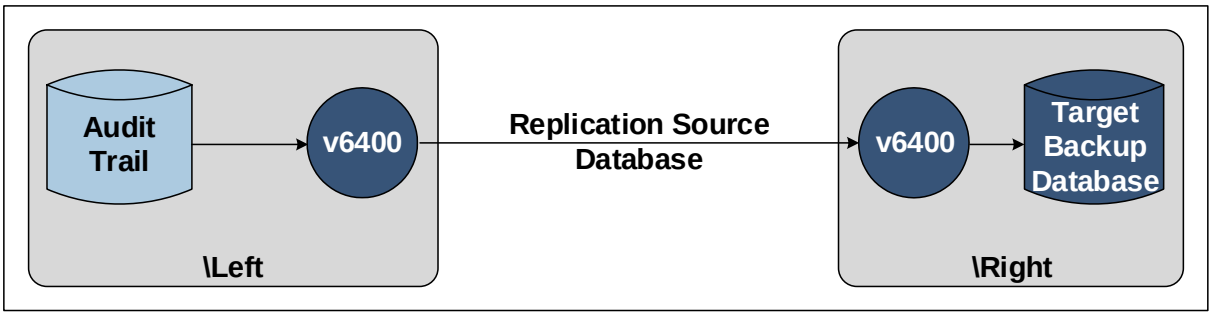


Figure 11 – Configure New Version of Replication Engine

During the entire migration, the target database was available as a full and complete backup to the production database. At any point, if an error with the new environment is identified, the PROOF testing can be stopped without affecting the production replication environment. The replication engine was migrated from version v6100 to version v6400 with no application downtime and with continuous availability of the target database – a zero downtime migration. Additionally, interoperation between the original and new replication engines was never required, which is a significant reduction in risk from trying to force interoperation.

If disk space is limited, or if the environment is running as an active/active architecture, then as each PROOF test completes, the validated tables should be removed from the original (v6100) replication environment and added to the new (v6400) replication environment using the production version of the target database. This step allows for a phased cutover from the original environment to the new one over a period of time dictated by the speed in which the system operations team can complete the PROOF tests, rather than via a classic all-at-once big-bang scenario. However, most companies will wait for a hardware refresh to update replication engines, which provides additional isolation of the data replication engine migration process, and further reduces the risk of impacting the production environment.

Version Independence – Two Node Step Summary

To summarize the sequence of steps described in the Version Independence process when using two nodes:

1. Perform the initial checkout of the new replication engine version in a non-production test environment.
2. If OK, on the production target system create a test database (PROOF) and load a representative sample of 20 tables from the production source database (e.g., using Shadowbase SOLV).
3. Configure and start replication using the new replication engine version from the production source database to the PROOF target database for the selected tables.
4. Run in this mode for a while until representative data changes are processed by the new replication engine.
5. Compare the updated data in the PROOF target database with the production target database (e.g., using Shadowbase Compare). Alternatively, compare the data in the PROOF database against the production source database.
6. If the PROOF database compares OK, repeat steps 3, 4, and 5 with an increasing number of test tables.
7. If OK, then incrementally add more tables to the PROOF target database, removing them from the original production replication version configuration.
8. Once all tables have been migrated to the new replication engine version, decommission the original replication version.

Of course, variations on this theme to accommodate specific environment and requirements are possible.

Version Independence – Two Node Pros and Cons

It is helpful to compare and contrast the benefits (pros) and issues (cons) for each approach discussed in this paper. For the simplest case of Version Independence when using two nodes:

- Pros
  1. Avoids the risks associated with replication engine version interoperation. In some cases, version interoperation is not even possible (e.g., when the replication engines are from different vendors, which is an especially important aspect for bi-directional replication environments).
  2. Avoids (or at least minimizes) any application outage that might otherwise be required to convert to the new replication engine.
  3. First can be performed in a test manner on the production environment before taking over production replication. The new replication engine can be fully tested and proved to be configured and working properly before starting the upgrade process. There is not a risky big-bang cutover to the new replication engine environment without having first verified correct operation. Additionally, when the cutover does occur, it is to a *known-working* environment.
  4. Incrementally can be performed, with a small set of files/tables to start, and grow at the speed or schedule that the staff feels is appropriate.
  5. Scaling the new replication engine environment can be accomplished slowly and methodically, validating that the environment is properly configured to handle the load and minimally affects the existing production environment before more load is added.
  6. If anything goes wrong, fall back to the original replication engine is simpler and easier and occurs with a smaller data set than the traditional big-bang approach.
  7. Can be accomplished during normal staff working hours (if desired) and does not require an off-hours effort to deploy the migration project.
- Cons
  1. "Thrill-seekers" may not appreciate the Version Independence process, since it has a tendency to make the upgrade less of an "adrenaline-filled roller-coaster ride."
  2. Using the same two nodes for the migration that are being used for production processing increases the chances of adverse impact to the production environment. This issue is addressed in the following sections.

Protecting the Standby System

One problem with the Version Independence process when using two nodes is that it uses the production and DR Standby (DRS) systems as the location(s) where the migration takes place. This is a reasonable approach for environments where only two systems/environments are available. However, for environments that can leverage a third system during the migration (perhaps as part of a hardware refresh sequence), this approach is more desirable, since it lessens the risk of any issues occurring to the production environment itself during the migration. Fewer/no changes are initially needed on the production source system, and a valid backup system (the original DRS) is always available for production failover with its original configuration and database intact, if it is needed. In other words, *migration isolation* is achieved.

Protecting the Standby System When Using Uni-directional Replication – Three Node Step Details

This problem can be solved by not touching the existing/original replication environment during the migration process. The plan is to create a new DRS with the new data replication engine on a third node. To accomplish this plan, the new replication engine is configured on the production source system replicating to the new DRS system (the third node).

When the new DRS has been created, loaded, and synchronized, its database contents are compared with those of the original DRS to ensure that the new DRS is correct, as shown in Figure 12. The original DRS is only shut down if the comparison shows that the new DRS is correct. Alternatively, for this approach or any discussed in this paper, the compare could be performed between the production database and the new DRS's database. In this way, the company can be absolutely certain that the new replication engine is functioning

properly, without having to rely on whether the original replication engine was maintaining a correct and complete copy of the database on the original DRS. (Oftentimes, in our experience, it is not.)[4]

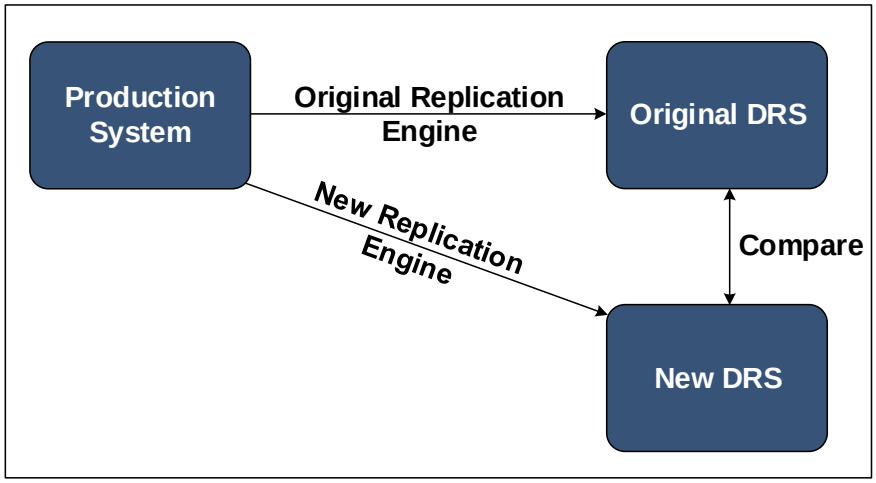


Figure 12 – Creating a New Target Database from the Production System

Ideally, similar to other approaches discussed in this paper, before shutting down the original replication engine and the original DRS, ideally, a failover to the new DRS should be performed to ensure that it is a complete and correct environment for which the application can run. While running on the new DRS, reverse replication can be used to keep the production database synchronized. Once it has been established that the new DRS is fully functional, production processing could be returned to the production database. The original data replication engine and the original DRS then can be shut down at this point.

An alternative approach is shown in Figure 13. The main benefit of this approach is that it does not affect the production environment until much later in the migration process. The new DRS is created by replicating to it from the original DRS, using the new data replication engine. Of course, the original DRS should be validated and accurately match the production environment via a complete compare sequence before using it as the source. When the replication testing is complete and the databases compare successfully, the original DRS can be shut down, since the new DRS is a known-valid copy. The new replication engine is installed on the production system, and the new DRS is then kept synchronized with the production system via the new data replication engine.[5] Note that this approach requires that the original DRS is known to be a consistently correct copy of the production database.

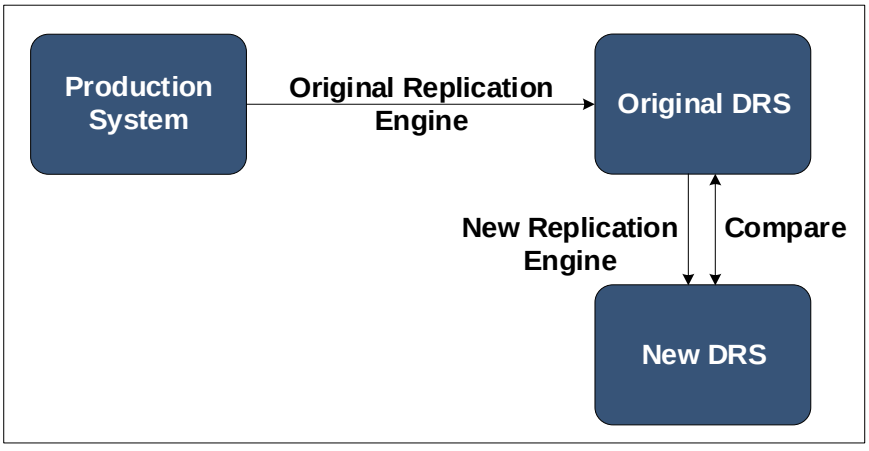


Figure 13 – Creating a New Target Database from the Standby System

[4]We appreciate that this is a bold statement to make. However, it is indeed our experience that many of the data replication target environments we encounter do not accurately reflect the true state of the source database. To confirm our statement, please run periodic compare operations (for example with Shadowbase Compare) to validate that your target matches your source; your auditors (and customers if you need to failover) will thank you.

[5]For this and the other approaches discussed, knowing precisely where to start the new replication engine for replicating from the source environment's change log to the new target environment is a complex task, and is discussed later in this paper in the section, "Solving the *Jagged Edge* Problem."

This approach is often preferred, because it does not require changes to the production system while the new DRS environment is built and validated. Instead of the production system having to support two replication engines, the original replication engine is simply replaced with the new replication engine when the upgrade takes place. Furthermore, the original DRS is typically less busy than the production system and has the extra capacity to perform the additional replication to the new DRS. As mentioned previously, before decommissioning the original environment, an additional check should be made to compare the production database to the new DRS database to be absolutely certain that both match. If they do not, the migration is paused, the cause identified/fixed, and the upgrade sequence is restarted – all without affecting the production environment and with minimal impact to the original DRS.

Version Independence – Three Node Step Summary

To summarize the sequence of steps described in the Version Independence process when using three nodes:

1. Perform the initial checkout of the new replication engine version in a non-production test environment.
2. If planning to use the production system as the source for the migration:
    a. If initial checkout OK, on the new DRS target system create a test database (PROOF) and load a representative sample of 20 tables from the production source database (e.g., using Shadowbase SOLV).
    b. Configure and start replication using the new replication engine version from the production source database to the PROOF target database for the selected tables.
    c. Run in this mode for a while until representative data changes are processed by the new replication engine.
    d. Compare the updated data in the PROOF target database with the production target database (e.g., using Shadowbase Compare). Alternatively, compare the data in the PROOF database against the production source database.
    e. If the PROOF database compares OK, repeat the steps above with an increasing number of test tables.
    f. If OK, then incrementally add more tables to the PROOF target database, removing them from the original production replication version configuration.
    g. Once all tables have been migrated to the new replication engine version, decommission the original replication version and original DRS.
3. If planning to use the original DRS as the source of the migration:
    a. If initial checkout OK, on the new DRS target system create a test database (PROOF) and load a representative sample of 20 tables from the original DRS database (e.g., using Shadowbase SOLV). First, ensure that the original DRS database is a correct and consistent copy of the production database.
    b. Configure and start replication using the new replication engine version from the original DRS database to the PROOF target database for the selected tables.
    c. Run in this mode for a while until representative data changes are processed by the new replication engine.
    d. Compare the updated data in the PROOF target database with the original DRS database (e.g., using Shadowbase Compare). Alternatively, compare the data in the PROOF database against the production source database.
    e. If the PROOF database compares OK, repeat the steps above with an increasing number of test tables.
    f. If OK, then incrementally add more tables to the PROOF target database and to the new replication engine configuration.
    g. Once all tables are added to the new replication engine version and the new DRS target database compares successfully, install the new replication engine on production and configure it to directly replicate to the new DRS target database.
    h. Validate that the new replication engine is working properly (e.g., compare the production source database and the new DRS target database after running for a while), and then decommission the original replication version and original DRS.

Of course, variations on this theme to accommodate specific environment and requirements are possible.

Version Independence – Three Node Pros and Cons

For the case of Version Independence when using three nodes:

- Pros
  1. The three node Version Independence process provides the same benefits as the two node process, while also reducing the impact to the production node (or at least impacting it far later in the migration process).
  2. There is minimal impact to the original DRS, and it is available the entire time the migration is taking place as a failover backup, if it is needed.
  3. Once the migration has taken place, the original DRS system is available for failback, if it is needed.
  4. Once the migration to the new replication engine and new DRS has occurred, the newly created data can be reverse replicated into the original DRS to keep it synchronized. If a subsequent failback is needed, it can be accomplished without requiring a reload of the original DRS.
- Cons
  1. The three node Version Independence process still has an impact on the production node that can be avoided when using a four node Version Independence process (as discussed in the following section).

Version Independence – Four Node Step Details

Yet another approach is shown in Figure 14. This approach is often used if the company is doing a full hardware refresh of both the production and the original DRS systems. An entirely new production/standby configuration is purchased and is thoroughly tested with both the application and new data replication engine, including failover and failback, etc. In such cases, the operating system and other subsystems on the refresh hardware are often new(er), and hence additional extended testing of this new environment is warranted. The Version Independence process accommodates a testing cycle of arbitrary duration, allowing the staff to full certify that the new environments are properly functioning before beginning the migration process.

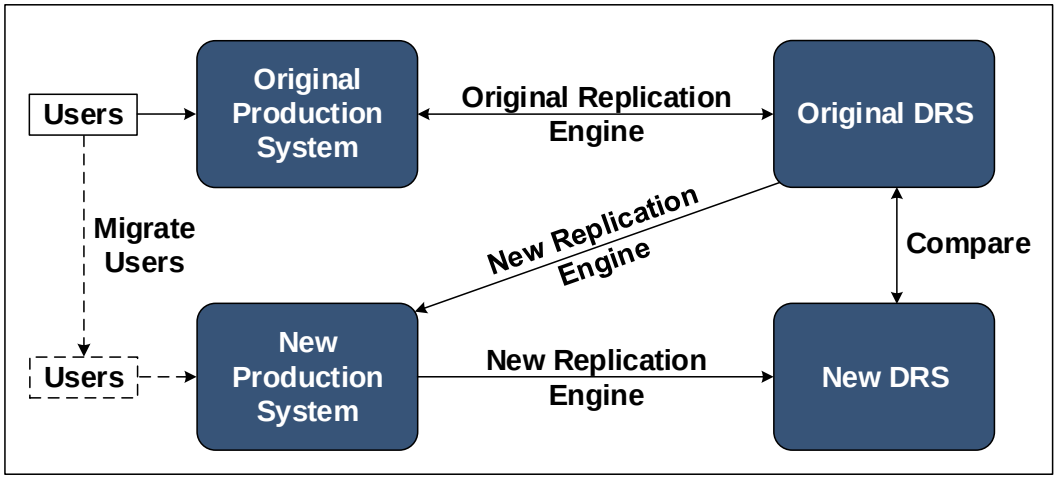


Figure 14 – A Full Uni-directional Hardware Refresh

To begin the migration process, the new production system is loaded from the original DRS to avoid affecting the production system, assuming the original DRS is known to be a correct and consistent copy of the production database. Then the new replication engine is installed on the original DRS and both of the new systems, and replicates the data changes from the original DRS to the new production system, as well as the new production system's changes to the new DRS. The contents of the new DRS are compared to those of the original DRS (or the original production system for absolute certainty). If the contents are correct, users are migrated in a controlled fashion from the original production system to the new production system; of course, they can be brought over all at once if that is desirable. Once this step is completed, the original production system and original DRS system can be decommissioned.

Note in this example that the changes to the new production system are not being reverse replicated into the original production system. Reverse replication would be helpful if a failback to the original production system

is needed to preserve all of the newly created data before the failback occurred. If a failback occurs, the newly generated data will not be present in its database. This problem can be avoided by reverse replicating the user's changes from the new production system to the original production system for the users that have been cut over. If all users are not cut over at the same time, care must be taken to avoid re-replicating these user changes from the original production system through the original DRS to the new production system (e.g., avoid a circular network map). Either implement the data replication cut-off in the new replication engine that is replicating from the original DRS to the new production system (e.g., using data content filtering), or use the preferred bi-directional replication approach (as discussed in the following sections on bi-directional replication).

Version Independence – Four Node Step Summary

To summarize the sequence of steps described in the Version Independence process when using four nodes:

1. Perform the initial checkout of the new replication engine version and the new systems in a non-production test environment. Since both the production and original DRS are being replaced with a new production and new DRS system, full application and data replication engine testing, (including failover and failback, etc.) should be performed on the new systems. Do not continue until all tests successfully complete.
2. If OK, install and configure the new replication engine on the original DRS, the new production system, and the new DRS.
3. Create the new database on the new production system and the new DRS.
4. Load the new database on the new production system and the new DRS (e.g., using Shadowbase SOLV). First, ensure that the original DRS database is a correct and consistent copy of the original production database. If it is not, rectify that issue before performing the migration.
5. Configure and start replication using the new replication engine version from the new production system to the new DRS as well as from the original DRS database to the new production system.
6. Run in this mode for a while until representative data changes have been processed by the new replication engine all the way through the new production system to the new DRS.
7. Compare the updated data in the new production database with the original DRS database (e.g., using Shadowbase Compare). Alternatively, compare the data in the new production database against the original production source database. Do the same for the new DRS database.
8. If the databases compare OK, migrate the users from the original production environment to the new production environment.
9. If reverse replication is needed, stop replication from the original DRS to the new production system, and replicate from the new production system back to the original production system to keep its database synchronized. If desired (and this is recommended), the original production system can then continue to replicate to the original DRS to keep the original DRS in sync as well. Of course, if the data schemas change from the original format to a new format, configure those changes into the reverse replication path (new production system to original production system).
10. When the new production system, new replication engine, and new DRS are performing satisfactorily, decommission the original production system, original replication engine, and original DRS.

Of course, variations on this theme to accommodate specific environment and requirements are possible.

Version Independence – Four Node Pros and Cons

For the case of Version Independence when using four nodes:

- Pros
  1. The four node Version Independence process provides the same benefits as the three and two node processes, while adding in far less impact to the original environment. In our experience, the four node Version Independence process is the safest and most reliable approach available today, since it requires no impact to the production application environment/database, and it affects the original DRS much later in the migration process.
  2. Once the migration has taken place, both the original production and the original DRS systems are available for failback, if they are needed.
  3. If reverse replication is used (a best practice), the original production system and original DRS are kept up-to-date with the data changes made at the new environments. Reverse replication is very helpful, if a failback to the original environment is needed.

4. Once the migration to the new production system and new DRS has occurred, this new mode can run for as long as necessary to prove that the new environment is properly working. The original environments then can be decommissioned when the new environments are known to be correct.

- Cons
  1. The four node process perhaps is more complex than some of the simpler processes previously described; however, it introduces far less risk and far later in the migration process.

Any upgrade or migration project can be risky. This risk is compounded when full application services must be provided while the upgrade or migration takes place. The Version Independence process does an excellent job of mitigating or eliminating this risk while this process occurs, allowing staff members to proceed at their own pace and schedule, when they feel the new environment has been proven, and when they are comfortable with migrating. The Version Independence process is being used now in real production environments to attain these benefits and eliminate these risks.[6]

Protecting the Standby System when Using Bi-directional Replication

The sequence is more complicated if bi-directional replication is being used between the production database and the target database during an upgrade to a new bi-directional replication engine. In these cases, the two replication engines do not typically interoperate and know of the changes each is making/replicating, thereby increasing the potential for (incorrect) data oscillation between the nodes. Changes made by the original DRS simply cannot be replicated to the new DRS because the replication engines are different. Therefore, for a two-node environment, it is recommended to create a third environment, preferably on a third node, as shown in Figure 15.

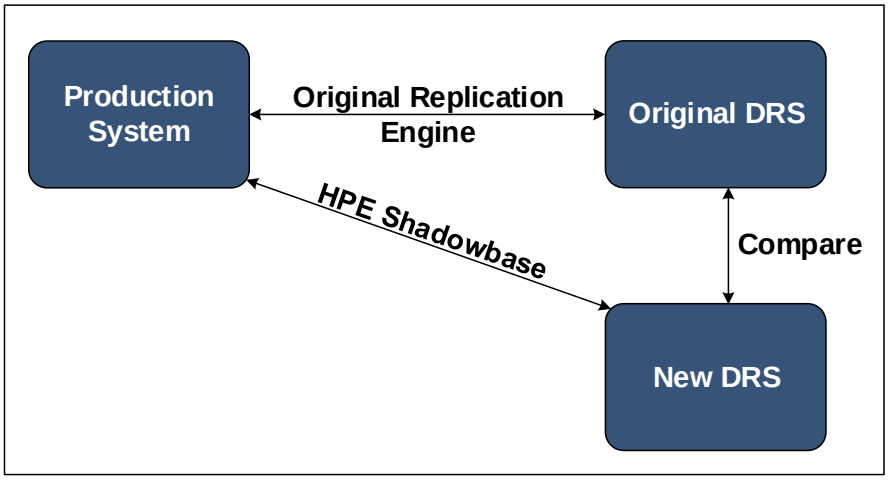


Figure 15 – Creating a New Target Database from the Production System Using Bi-directional Replication

In this figure, bi-directional replication via the original replication engine is used between the production system and the original DRS, and the goal is to migrate to the new replication engine with a new DRS. After the new DRS is available and has been fully tested, the new replication engine is configured, and the new DRS database is loaded and synchronized; the environment is ready for the migration process to begin. This sequence and effort is similar to the one previously discussed (Figure 12).

Application changes made at the production system essentially are uni-directionally replicated to the original DRS via the original replication engine, as well as to the new DRS via the new replication engine. Each bi-directional replication engine takes care not to *ping-pong* back the changes to the production system.

Application changes made to the database at the original DRS system are replicated to the production system via the original replication engine, where they update the production database. These changes appear to the new replication engine as if they are “application” changes (because the new replication engine did not make them), and are subsequently routed through the production system to the new DRS by the new replication

[6]Please see Shadowbase Zero Downtime Migration Case Studies for supporting examples.

engine. The production system is thus acting as a router (called a *route-through* node) for these changes, and routes them to the new DRS system via the new replication engine. Since the new replication engine is bi-directionally replicating between the production system and the new DRS, it knows not to route back these changes to the production system (classic bi-directional cut-off).

Eventually, the application changes made at the new DRS follow a similar, although reversed, approach to update the production database. Eventually, the changes that are applied by the new replication engine are routed through the production system by the original replication engine to the original DRS. In this way, all three of the systems remain synchronized with each other, with a change made at any of them properly reflected in all three databases. During this time, each DRS database/system is ready to take over if the production system experiences a fault.

An alternative approach is shown in Figure 16. This approach is preferred by many companies because it does not impact the production node until very late in the migration process, after the new DRS is built, deployed, synchronized, and proven to be functioning correctly.

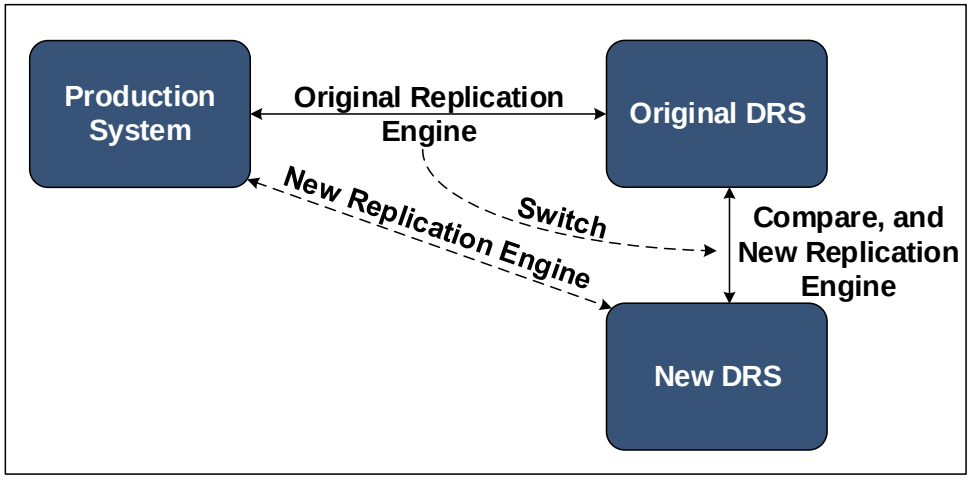


Figure 16 – Creating a New Target Database from the Original DRS Using Bi-directional Replication

The new DRS is created via the original DRS database replicating to it with the new bi-directional data replication engine (the solid lines/arrows in Figure 16). By using bi-directional replication, changes made to either DRS are reflected in the other DRS. These changes are ultimately replicated to the production system from the original DRS via the original replication engine.

When the new DRS is synchronized and ready to take over (i.e., the databases are compared and are correct), the new replication engine is installed and started on the production system, if not done previously (the dashed lines/arrows in Figure 16). The original replication engine and the original DRS then can be shut down, since the new DRS is a known-valid copy.

The new replication engine begins replicating from production to the new DRS at the point where the original replication engine was shut down. Similarly, the new engine is configured to reverse replicate from the new DRS to production from the point where it shut down when it was previously replicating to the original DRS by using the same restart point, or a point somewhat earlier in the change log.[7] From that point forward, the production system and the new DRS are kept synchronized via the new replication engine (the dashed lines/arrows in the figure).

Yet another approach is shown in Figure 17. This approach often is used if the company is doing a full hardware refresh of the production and DRS systems. An entirely new production/standby configuration is purchased, configured with the new replication engine, and thoroughly tested. The new production system is loaded from

[7]The point where the old replication engine shuts down can be a timestamp, audit trail position, or other point where it is known that all transactions prior to this point were replicated by the old replication engine. As the new replication engine, HPE Shadowbase software then can be configured to start replicating from this point, and taking responsibility for replicating all transactions in the audit trail. HPE Shadowbase software also can be configured to replicate *transactions-in-flight* at the point of shutdown (i.e., the so-called *jagged edge*, as discussed later in this paper). This process is much more complicated if all nodes are actively processing transactions and making changes to their databases when the switchover occurs. Contact us for additional details if contemplating this approach.

the original DRS and configured to replicate changes made to it back to the original DRS as well as to the new DRS. The new DRS then can be loaded from the new production system.

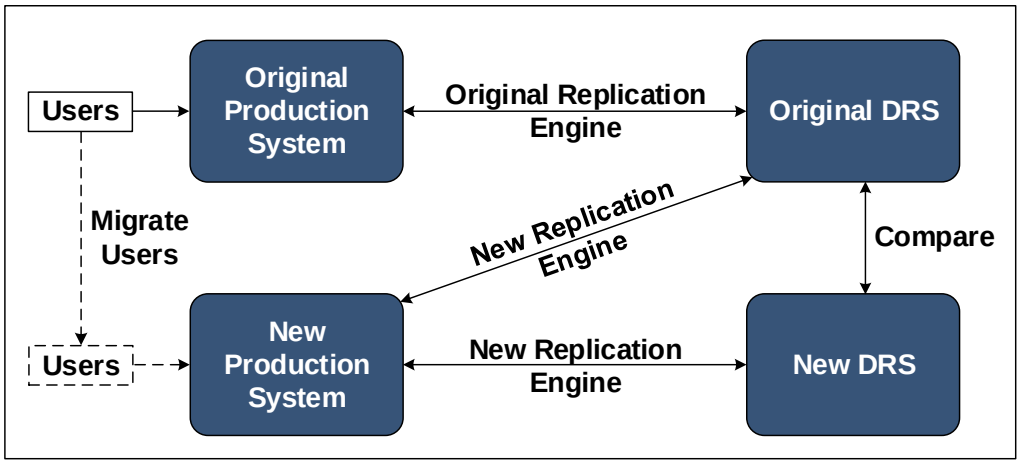


Figure 17 – A Full Bi-directional Hardware Refresh

Once all databases are loaded and synchronized, customers often run an extra set of tests with the new production system and new DRS to verify full application processing across end of day, week, month, etc. If database changes are made in the new environment that are not needed back in the original environment, the reverse replication link to the original DRS needs to be shut down while this step occurs.

When ready to move forward with the migration, the contents of the new DRS are compared to those of the original DRS (or to the production database), and if they are correct, users are migrated in a controlled fashion from the original production system to the new production system. Changes made to either production system are bi-directionally replicated to the other one by the original DRS, because it is not acting as a route-through node. Once the user migration has been completed, the original production system and original DRS system can be left running as a failback environment, and then eventually decommissioned once the new environment is fully trusted.[8]

Version Independence – Bi-directional Considerations

Bi-directional replication causes additional considerations for the migration process:

1. When switching from one vendor's bi-directional replication engine to another's, special care must be taken to make sure the replication engines interoperate properly, without causing infinite-loop data oscillation. This goal typically is achieved by setting up the original DRS as a route-through node.
2. At the customer's choice, the new replication engine connection back to the original DRS can be shut down after the users are migrated to the new production system. If left operational, the new replication engine can continue to run between the new production system and the original DRS, which keep the original production environment synchronized with any data changes, is a failback is needed.
3. It is simplest to migrate all users from the original production environment to the new one at the same time, which provides a cleaner shutdown and takeover point for the new replication engine. If migrating users slowly and in batches, care must be taken to avoid data oscillation and potential *data collisions* (same data being updated at the same time at more than one node). When configured correctly, the replication engines manage the data oscillation problem; the data collision problem should be handled via a partitioned set of moves so that any data item can only be updated in one place at a time. Otherwise, data collision identification and subsequent resolution algorithms must be added into the migration sequence.

Version Independence – Bi-directional Pros and Cons

Bi-directional replication environments add in additional complexity, but also provide additional capability:

[8] As discussed in the previous footnote, the sequence is simpler if all users migrate from the old production system to the new production system at the same time; otherwise, data oscillation cut-off filters need to be established. Additionally, the *jagged edge* issue in the restart points also needs to be addressed (as discussed later in this paper). Contact us for additional information if this approach applies to you.

- Pros
  1. Bi-directional environments allow the users to be moved from one application copy/system to another, without changing the data replication engine's configuration in order to reverse replicate the changes after cutover; this feature already is provided by the data replication engine.
  2. Bi-directional environments automatically keep all environments synchronized, meaning there is no data loss after the migration takes place, if a subsequent failback to the original environment is needed.
- Cons
  1. Bi-directional environments are more complex, and each vendor typically has its own/internal algorithms for managing the bi-directional data oscillation problem. These algorithms are typically specific and unique to each vendor, and the data replication engines typically do not interoperate properly to avoid oscillation. Hence, when faced with this issue, the best approach is to designate a route-through node to avoid any data oscillation issues (as discussed in the previous section).
  2. The entire migration becomes more complex if users are migrated in batches, potentially causing data collisions subsequently to occur. Data collisions must then be identified and resolved (if using asynchronous replication), or avoided via request/data partitioning or via synchronous replication.[9]

Solving the *Jagged Edge* Problem When Switching Replication Engines

The so-called *jagged edge* problem occurs when switching from one replication engine to another, although it may also occur when upgrading the replication engine version, if this aspect of the replication engine has been changed across versions. It is especially important when the original and new data replication engines are from different vendors. To properly resolve this issue, a thorough understanding of how transactionally-based engines replicate data is needed, along with reviewing various algorithms that resolve the issue, based on the type of engine that is being replaced.

Jagged Edge and Audit Trail Background

Figure 18 repeats an earlier figure (Figure 5); but now we delve into the details of switching over from v6100 to v6400 of the data replication engine (while the new replication engine catches up to the proper point where the switch-over occurred), without missing or repeat-replicating any data, or causing the target database to roll back to its original version.

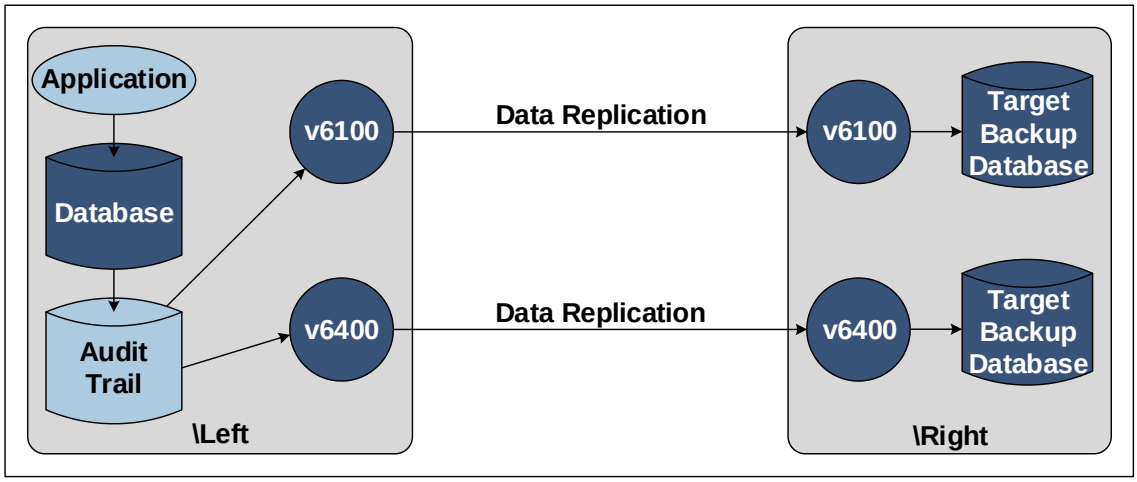


Figure 18 – v6100 to v6400 Switch-Over

[9] Contact us for further details on these more complex configurations.

The approach used to switch over from one replication engine to the other depends on the method each individual replication engine uses when it replicates, with the most important consideration being the method the original replication engine uses to track its restart point. The new data replication engine must adapt accordingly. Although other methods are also in use, we now discuss two common methods, and how properly to switch over when each is in use:

- *The Brute Force Replication Method*
- *The Transactional Replication Method*

For this discussion, assume that the "Target Backup Database" in Figure 18 represents the actual complete target database – the changes for a specific file/table (or individual partition in an HPE NonStop system) are replicated by either v6100 or v6400, but not both.

In Figure 18, the audit trail represents the log of all changes made to the source database; these database changes are applied by all of the applications on the source environment. Although the audit trail may be comprised of multiple separate log files (e.g., a MAT sequence of log files along with one or more AUX sequences of log files on an HPE NonStop), it can be processed as a singular and sequential listing/queue of the change events across all files and tables in "roughly" *ascending time order*. Note that the landed/received order prevails, if it differs slightly from the assigned chronological order of each event in the aggregated trail.[10]

Processing the audit trail in sequential/landed order yields a stream of transactional activity that has occurred against the source database. For any individual file/table (or partition on an HPE NonStop), the landed order reflects with absolute certainty the correct and consistent order that the events occurred in for each individual file or table. On other platforms, the landed order may need to be re-sorted; however, the sorted order reflects the correct and consistent event order. In either case, replaying the events in the correct and consistent order results in a correct and consistent target database.

Often (and always on HPE NonStop), the landed order represents the simultaneous interthreading of many transactions occurring at the same time. On some platforms/databases, a re-sorted order returns the change events in transactional order, based on the commit or abort completion time, such that all events for a transaction are returned in a sequential group for each transaction. Regardless, replaying the events in the order received (whether landed or sorted), results in a correct and consistent target database.

When processing the audit trail, each replication engine typically maintains a persistent "restart point" that reflects the position where it needs to restart, if it stops/fails and subsequently restarts. This position reflects the point (or can be used to derive the point) in the audit trail where the replication engine needs to start reading in order to guarantee that it does not miss/skip any data. In the best data replication engine designs/implementations, it does not replay any data against the target database, because it can cause data consistency issues at the target during the restart/catchup sequence.

Figure 19 reflects a simplified view of the layout of the internal events inside the audit trail (or the sequence of events, because the audit trail is aggregated). In this figure:

- The audit trail is depicted as a sequenced queue of change data events, where each event has an associated transaction ID assigned to it. The front of the queue (removal/extraction point) is to the right; the end of the queue (insertion/arrival point) is to the left.

- Time moves from right to left. The oldest/earliest events are to the right in the queue; the newest/latest events are to the left in the queue.

[10] One reason that each change event's landed order may differ from the assigned chronological order of when the change occurred has to do with the flushing operation on certain systems. For example, the landed order of the events may be batched in memory prior to flushing, with the flush (chronological) time assigned when the batch is actually written to disk. When several separate processes are performing this sequence in parallel, the landed order may not always match each block's assigned flush times.

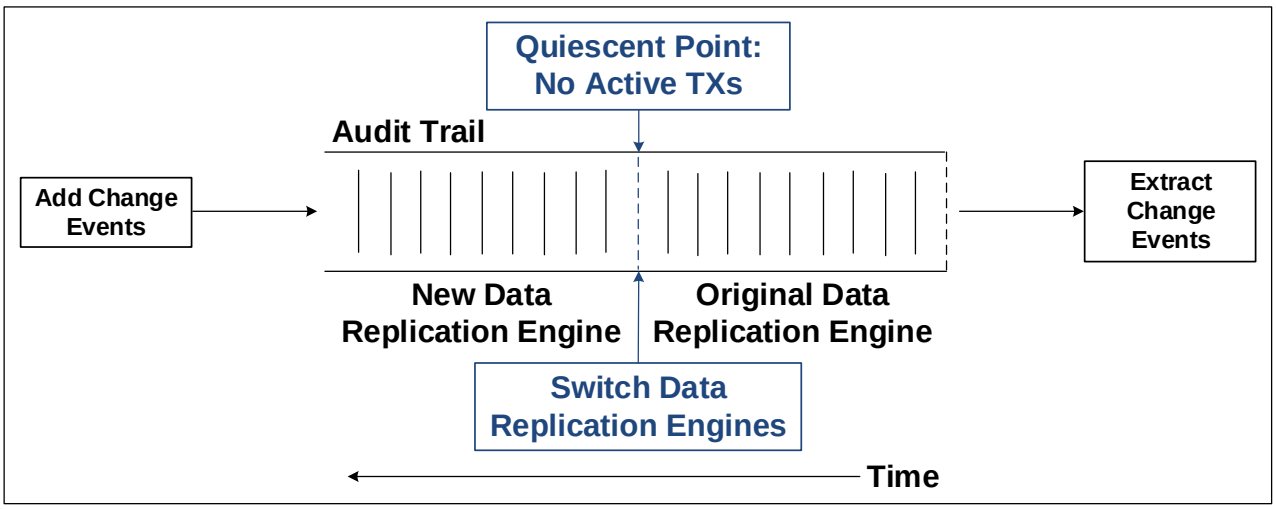


Figure 19 – Simplified View of the Audit Trail When the Application is Stopped at the Quiescent Point

If the application can be quiesced at a point where all transactions have ended (the "Quiescent Point: No Active TXs" in the figure), the original data replication engine takes responsibility to replicate the events to the right of the quiescent point, and the new data replication engine takes responsibility to replicate the events to the left of the quiescent point, with no transactions spanning it. It is fairly simple; however, since this paper is about avoiding an application outage while the migration or upgrade takes place, this "clean" switch-over point does not exist for applications/transactions that remain active during the switch-over process.

For this complex case, a more advanced algorithm is required, where the application is active when the switch-over takes place, as shown in Figure 20. In this figure:

- The audit trail again is depicted as a queue that grows with change data events from right to left (time increases from right to left).
- The switch-over point is selected to be at Timestamp(1). This is the point the original data replication engine is stopped/shutdown.
- The abort timer(2) represents a go-back interval, from the switch-over point, that are discussed below.
- TX(3) represents a transaction that started before the abort timer(2) go-back interval, that commits before the switch-over point.
- TX(4) represents a transaction that started and aborts before the switch-over point.
- TX(5) represents a transaction that started before the switch-over, but does not end until much later after the switch-over.
- TX(6) represents a transaction that started before the switch-over, but does not commit (at the Commit(7) point) until after the switch-over point.
- TX(8) represents a transaction that starts after the switch-over point and ends at some later time.

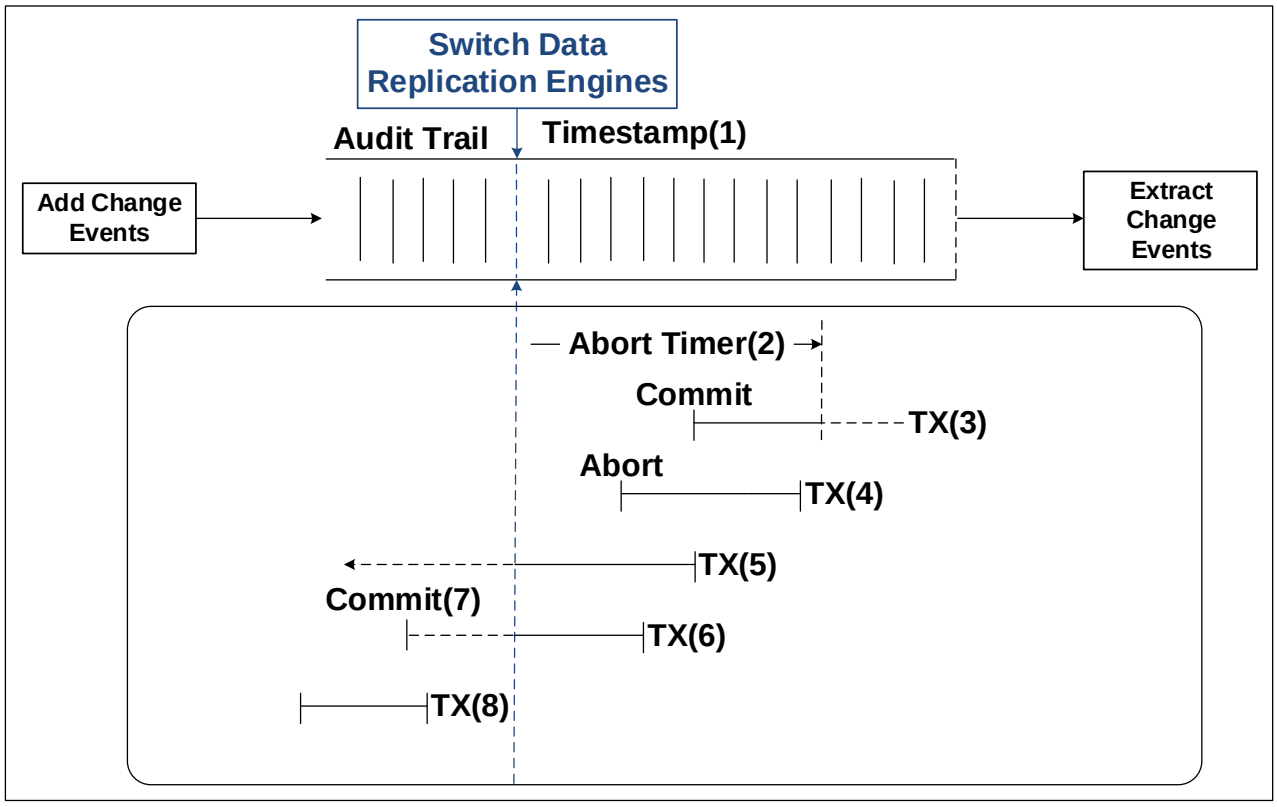


Figure 20 – View of the Audit Trail when the Application Remains Active Across the Switch-Over

The method to provide a clean, consistent, and complete switch-over from the original data replication engine to the new data replication engine depends on the method the original data replication engine uses to replicate the audit trail data.

The Brute Force Replication Method

If the original data replication engine uses the *Brute Force Method* to replicate:

- The original data replication engine takes responsibility to replicate and apply/commit all transactional events that have occurred *before* the switch-over point.
- In some cases, this replication may only include transactions that have committed before the switch-over point, excluding those that have aborted before the switch-over point.
- For transactions that are in progress at the switch-over point (e.g., TX(5) and TX(6) in the figure), the original data replication engine replicates and applies only those events that occurred before the switch-over point, performing a *commit* for each before shutting down. Since these transactions may ultimately abort, the new data replication engine is responsible for special processing of the transactions to apply all follow-on data and any backout events that might occur after the switch-over point.
- The new data replication engine takes responsibility to replicate all new transactions forward (i.e., the events for those transactions that start after the switch-over point).

The Brute Force Method does not maintain true data consistency as it commits and materializes events for transactions that may ultimately abort, as well as pre-commit transaction events for those that are in process at the switch-over point that ultimately commit. However, any such inconsistencies are short-lived, only until those transactions eventually complete (commit or abort) after the switch-over to the new data replication engine.

The Transactional Replication Method

If the original data replication engine uses the *Transactional Replication Method* to replicate:

- The original data replication engine takes responsibility to replicate and apply/commit all transactional events that have *completed* before the switch-over point.
- In some cases, this replication may only include transactions that have committed before the switch-over point, excluding those that have aborted before the switch-over point. Regardless, only those transactions that have committed are materialized in the target database at the switch-over point.
- For transactions that are in progress at the switch-over point (e.g., TX(5) and TX(6) in the above figure), the original data replication engine either does not replay them, or aborts them when it shuts down at the switch-over point. In some cases, the original data replication engine may replay the data for transactions that are in progress as of the switch-over point. However, before it shut downs, it retrieves the backout events from the target side audit trail and applies them for any transactions that were not completed before the switch-over point. This process effectively matches performing an abort for the transactions that are in progress at the time of the switch-over.
- The new data replication engine thus must take responsibility to replay:
  - All transactions that were in progress at the switch-over point.
  - All transactions that start after the switch-over point.

Although it is relatively straightforward to replicate all new transactions created after the switch-over point, identifying those that were in progress at the switch-over point can be complex. Although other methods exist, one simple algorithm to identify these transactions is to use the DBMS's "auto abort" time as an interval (Abort Timer(2) in Figure 20).

The abort timer is a DBMS-maintained timer that insures no transaction lasts (exists/runs) for more than a set period of time (or audit trail duration). When an abort timer exists, it is known that no transaction can last more than that amount of wall clock time relative to the change's time in the audit trail. Hence, when the new data replication engine goes back from the switch-over point to at least the abort timer amount of time in the audit trail, it knows that no transactions started before this time are still active at the switch-over point. The new data replication engine computes the abort timer interval from using the switch-over point's timestamp (Timestamp(1) in the figure), deducts the abort timer setting from this timestamp (e.g., deducts 2 hours from the timestamp if the abort timer is 2 hours), goes back that far in audit. It then reads forward to the switch-over point, tracking all transactions encountered. The new data replication engine disregards all transactions that end before the switch-over point since the original data replication engine took responsibility to replicate them. For transactions still in progress at the switch-over point, the new data replication engine replicates them as well as all additional transaction events encountered from the switch-over point forward.

The Transactional Replication Method maintains full data consistency at the target as each transaction is replayed only once at the target and in proper order. Of course, additional algorithms exist for managing the jagged edge at the switch-over point, and these algorithms are more complex for bi-directional replication environments, but the main tenets remain the same. The original data replication engine takes responsibility for replicating all completed transactions before the switch-over point, with the new data replication engine taking responsibility for replicating all in progress transactions at the switch-over point as well as any new transactions that started after the switch-over point.

Summary

Sometimes it is necessary to change or update a data replication engine. Properly undertaken, a data replication engine migration imposes no downtime on applications or users, and the databases all remain consistent, complete, and up-to-date during the process. This is called a zero downtime migration (ZDM).

Since there is no big-bang cutover, and the original and new data replication engines do not need to interoperate, the ZDM technique results in greatly reduced risks for error as well as staff stress levels during the migration process. The migration can take place at normal working times when the staff is at their best, rather than late at night or on weekends, thereby reducing migration costs. It can even occur over an extended time period, with the existing backup database continuously available and fully synchronized with the production database, ensuring full application and database availability during the migration process as discussed in the three and four node examples.

This technique is similar to and leverages the HPE Shadowbase Zero Downtime Migration technique that companies have used for decades to upgrade their applications, database schema formats, file and table locations/indices, operating systems, or to perform a hardware refresh.[11] Application outages that either are planned to support an upgrade/migration, or caused by poorly executed upgrades/migrations, are outdated and should no longer occur. Use the HPE Shadowbase ZDM technique to attain continuous application availability with no risk across even the most disruptive migrations and upgrades.

There are numerous approaches and methods to thwart common problems faced during a migration: *Version Independence* (avoids interoperating different software versions); utilizing three nodes (for partial hardware migrations) utilizing four nodes (for a full hardware refresh); and bi-directional environments (keeps all nodes synchronized as the migration takes place). Also, the so-called jagged edge problem that occurs when performing any replication engine migration must be resolved.

Switching replication engines with zero downtime is obviously never easy. With proper preparation, planning, time, and project assessment, all migrations should go flawlessly. HPE Shadowbase software enables companies to leverage these migration methods with proven solutions and implementations. The HPE Shadowbase team is interested in discussing your specific project plans and help you realize them.[12]

[11] For additional information, please see the Gravic white paper, *Using HPE Shadowbase Software to Eliminate Planned Downtime via Zero Downtime Migration*.

[12] Questions or comments? Please contact us.

International Partner Information

Global

Hewlett Packard Enterprise
6280 America Center Drive
San Jose, CA 95002
USA
Tel: +1.800.607.3567
www.hpe.com

Japan

High Availability Systems Co. Ltd
MS Shibaura Bldg.
4-13-23 Shibaura
Minato-ku, Tokyo 108-0023
Japan
Tel: +81 3 5730 8870
Fax: +81 3 5730 8629
www.ha-sys.co.jp

Gravic, Inc. Contact Information

17 General Warren Blvd.
Malvern, PA 19355-1245
USA
Tel: +1.610.647.6250
Fax: +1.610.647.7958
www.shadowbasesoftware.com
Email Sales: shadowbase@gravic.com
Email Support: sbsupport@gravic.com

Hewlett Packard Enterprise Business Partner Information

Hewlett Packard Enterprise directly sells and supports Shadowbase Solutions under the name ***HPE Shadowbase***. For more information, please contact your local HPE account team or visit our website.