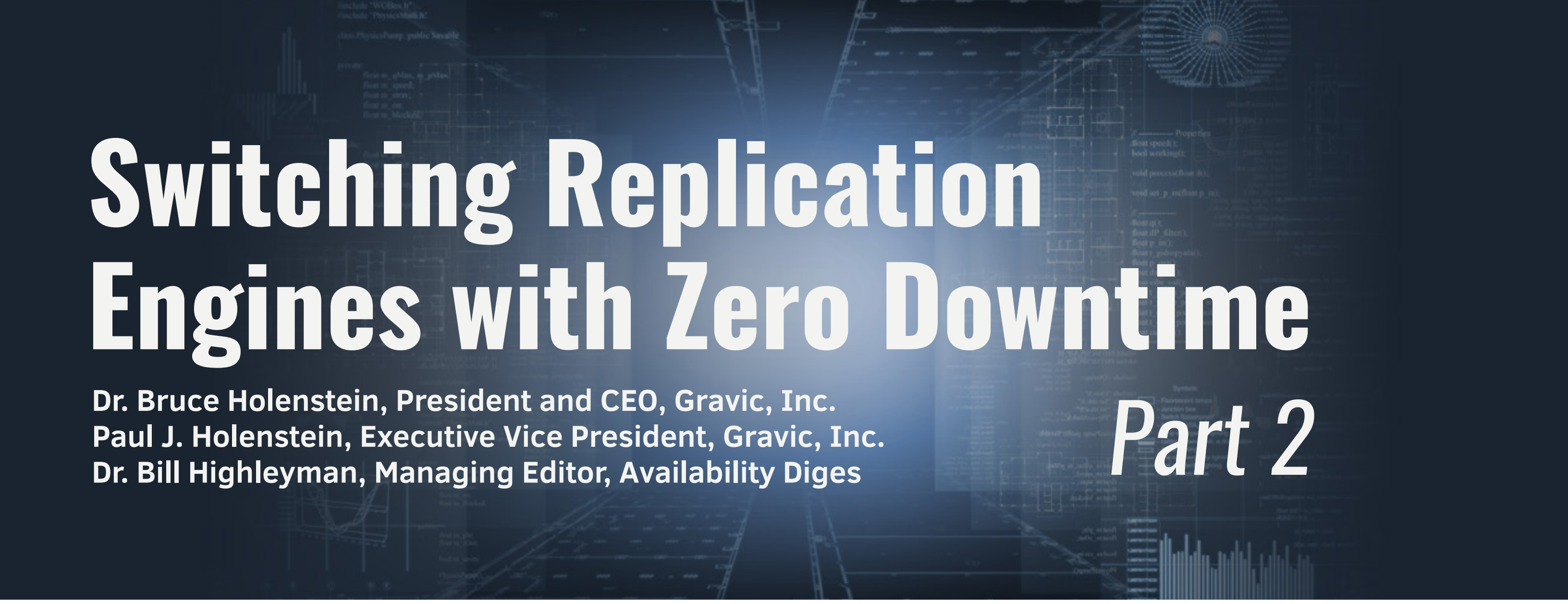

Switching Replication Engines with Zero Downtime

Part 2

Dr. Bruce Holenstein, President and CEO, Gravic, Inc.
Paul J. Holenstein, Executive Vice President, Gravic, Inc.
Dr. Bill Highleyman, Managing Editor, Availability Diges

In Part 1 of this article series, we described how an existing data replication engine could be replaced in a two-node scenario with a new version or with a different replication engine without taking an outage. As pointed out in that article, it is important to avoid requiring that different replication engines communicate with each other. Interoperation between different versions is very dangerous, error-prone, and can produce faults. In Part 1, we showed how a replication engine could be changed without interoperating between differing versions. Let us first review these procedures, and then we will look at eliminating the versioning problem by using additional nodes. We also look at protecting the standby system while the replication engines are being switched.

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 HPE 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 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 HPE 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 as described above:

Pros

1. Avoids the risks associated with replication engine version interoperation. In some cases, version interoperation is not even possible (i.e., 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 be required to convert to the new replication engine.
3. Can be performed in a test manner on the production environment first, 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. Can be performed incrementally, 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, which minimally affects the existing production environment before more load is added.
6. If anything goes wrong, falling back to the original replication engine is simpler, 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 approach is reasonable for environments where only two systems/environments are available. However, this approach is more desirable for environments that can leverage a third system during the migration, since it lessens the risk of any issues occurring to the production environment itself during the migration. (A third node is often available by using a development system, or loaning one for a short period.) 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.)

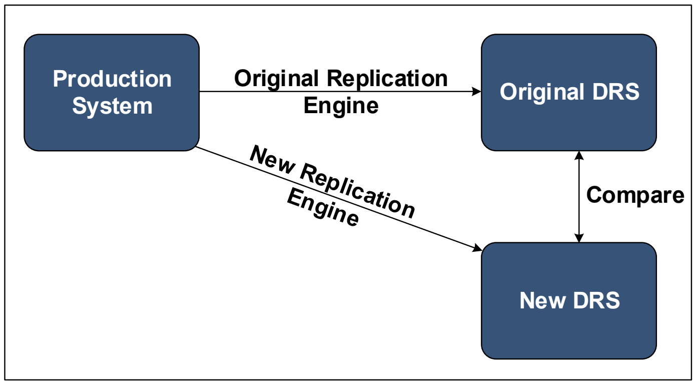

Figure 12 – Creating a New Target Database from the Production System

Ideally, similar to other approaches discussed in this paper, before shutting down the original replication engine and the original DRS, 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 can then be shut down at this point.

GRAVIC® Shadowbase

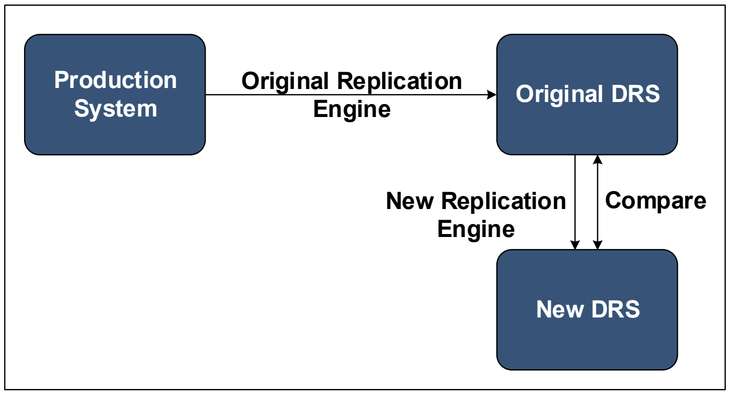

Figure 13 – Creating a New Target Database from the Standby System

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 via a complete compare sequence before using it as the source to ensure that it accurately matches the production environment. 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.[1] Note that this approach requires that the original DRS is known to be a consistently correct copy of the production database.

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 HPE 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.

[1] For this approach and the others 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 in the next article, Part 3.

c. Run in this mode 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 HPE 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 HPE 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 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 HPE 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), 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 (making four nodes available for the migration). 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., which is represented by the lower set of systems in the figure (New Production System and New DRS). In such cases, the operating system and other subsystems on the refresh hardware are often new(er); therefore, additional extended testing of this new environment is warranted. The Version Independence process accommodates a testing cycle of arbitrary duration, allowing the staff to fully certify that the new environments are properly functioning before beginning the migration process. Only after this certification is achieved, can the actual migration process take place (i.e., loading the New Production System from the Original DRS).

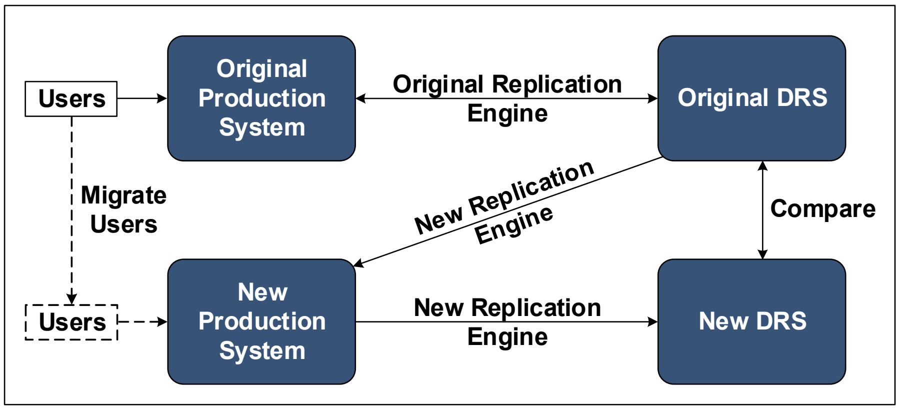

Figure 14 – A Full Uni-directional Hardware Refresh

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 users' 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 HPE 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 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 HPE 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 it 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 occurs 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.

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.

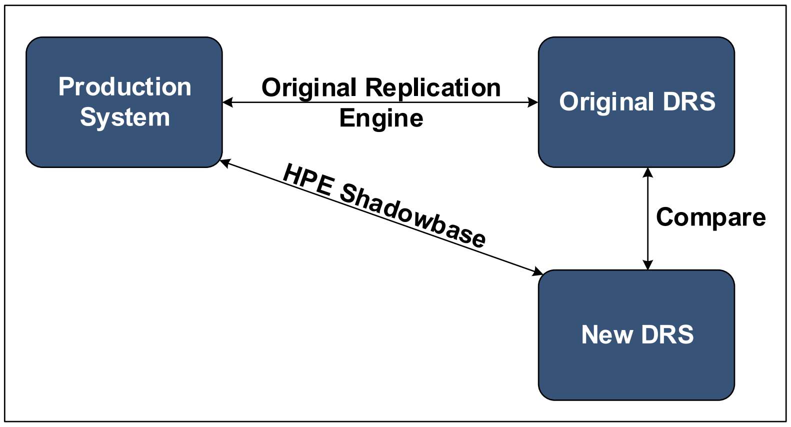

Figure 15 – Creating a New Target Database from the Production System Using Bi-directional Replication

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 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.

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.

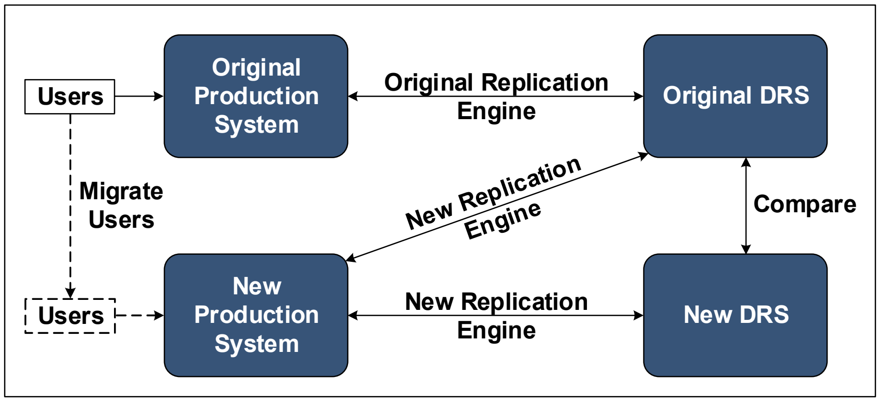

Figure 17 – A Full Bi-directional Hardware Refresh

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[2]. 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).

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

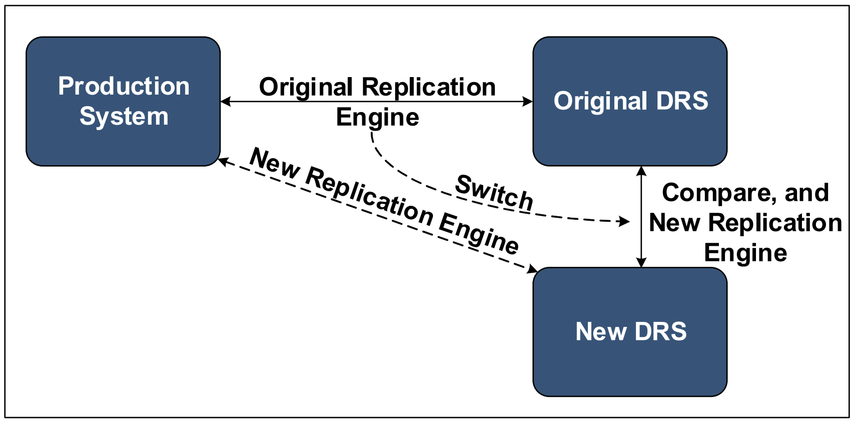

Figure 16 – Creating a New Target Database from the Original DRS Using Bi-directional Replication

[2] 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 take 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 Gravic for additional details if contemplating this approach.

is purchased, configured with the new replication engine, and thoroughly tested. The new production system is loaded from 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.

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.

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, if 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:

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.[3]

Summary

Sometimes it is necessary to change or update a data replication engine. Properly undertaken, such a migration will impose no downtime on either applications or users. We call this a zero downtime migration.

In this Part 2, we have discussed how the Version Independence of the migration can be improved using one or two additional nodes. This migration technique is similar to the HPE Shadowbase Zero Downtime Migration (ZDM) technique that customers have been using for decades to upgrade their applications, database schema formats, file and table locations (or indices), operating systems, or perform a hardware refresh.[4]

In the next article, Part 3, we will discuss how to switch replication engines without missing or re-replicating any data (solving the so-called jagged edge problem).

Paul J. Holenstein is Executive Vice President of Gravic, Inc. He is responsible for the HPE Shadowbase suite of products. The HPE Shadowbase replication engine is a high-speed, uni-directional and bi-directional, homogeneous and heterogeneous data replication engine that provides advanced business continuity solutions as well as moves data updates between enterprise systems in fractions of a second. It also provides capabilities to integrate disparate operational application information into real-time business intelligence systems. Shadowbase Total Replication Solutions® provides products to leverage this technology with proven implementations. HPE Shadowbase software is built by Gravic, and globally sold and supported by HPE. Please contact your local HPE account team for more information, or visit **https://www.ShadowbaseSoftware.com**. To contact the authors, please email: SBProductManagement@gravic.com.

Dr. Bruce D. Holenstein leads all aspects of Gravic, Inc. as President and CEO. He started company operations with his brother, Paul, in 1980. His technical fields of expertise include algorithms, mathematical modeling, availability architectures, data replication, pattern recognition systems, process control and turnkey software development.

Dr. Bill Highleyman brings years of experience to the design and implementation of mission-critical computer systems. As Chairman of Sombers Associates, he has been responsible for implementing dozens of real-time, mission-critical systems - Amtrak, Dow Jones, Federal Express, and others. He also serves as the Managing Editor of The Availability Digest (availabilitydigest.com). Dr. Highleyman is the holder of numerous U.S. patents and has published extensively on a variety of technical topics. He also consults and teaches a variety of onsite and online seminars.
Find his books on Amazon. Contact him at billh@sombers.com.

[3] Contact Gravic for further details on these more complex configurations.

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