Only On NonStop

Data Replication on Steroids

John R. Hoffmann
Manager of NonStop Development
Shadowbase Product Group

Mr. Hoffmann, Manager of NonStop Development, leads development for the Shadowbase Products Group on NonStop. John's experience with NonStop systems spans more than 30 years and began early in his career on the original Tandem NonStop System. John specializes in developing highly available, fault tolerant, real-time systems. He is a member of Gravic Labs, the company's intellectual property group, and is active in the fields of data replication and synchronization. He is a co-author on upcoming volumes in the series: Breaking the Availability Barrier. He received his BS in Honors Mathematics from McGill University, Canada.

The replication of data between geographically dispersed databases has many purposes. It is used for high availability to keep standby systems synchronized and for continuous availability to synchronize multiple active systems cooperating in a common application.[1] Data warehouses and data marts depend upon data replication to keep their data stores current. Legacy applications are integrated in real-time using efficient, event-driven processing by feeding the database changes to a downstream application as they occur. Data locality and accessibility are improved by allowing an application to maintain a local, synchronized copy of critical data generated by another application.

Transaction-oriented file and database management systems usually provide a change queue that holds all of the changes that have been made to the database. A replication engine follows the change queue and sends the changes to a target system to keep the target system's database synchronized with the source.

Most database management systems do not make the changes within a transaction available in the change queue until the transaction commits. In these systems the replication engine must wait for a transaction to commit before it begins to send its updates to the target system. We call this serial replication since transactions are sent to the target database one at a time.

Some database managers periodically send the change queue file to the target system. In these systems, no changes are typically available to the target database (that is, they are not materialized) until the entire queue file is received at the target system and closed. Then only committed transactions are available. Even if the database management system allows the change queue to be replicated before the commit occurs, the data is generally queued on the target system until a commit arrives for one of the transactions in the change queue. The committed transaction is then serially applied.

HP NonStop systems implement the change queue as the TMF Audit Trail for the Enscribe, SQL/MP, and SQL/MX file systems. The Audit Trail is different from other database management systems' change queues in that it makes changes immediately available for transmission as well as application into the target environment. Therefore, individual changes are replicated immediately and committed as soon as their transactions are committed at the source database. We call this concurrent replication since the replication of multiple, even related, transactions are in progress at any given time.

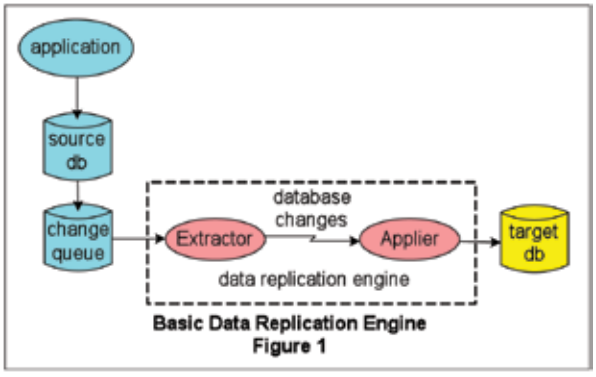
Concurrent replication has many efficiency advantages over serial replication. These advantages are described in this paper. The highly efficient technique of concurrent replication is commercially available only on NonStop systems because of the unique structure of and access to the TMF Audit Trail coupled with replication-engine extensions that allow the replication engine to replay the target transaction mix concurrently with source transaction processing. It is the specialized nature of the NonStop Audit Trail that allows data replication to perform as if on steroids.[2,3]

Data Replication 101

Replication Engine Architecture

Before we dive into serial and concurrent replication issues, it is important to understand the fundamentals of data replication.

Data replication is conceptually simple. It depends upon



Basic Data Replication Engine
Figure 1

1 What is Active/Active?, Availability Digest; October 2006.

2 We call this "data replication on steroids" because concurrent replication provides great performance enhancements, though without the negative side effects that accompany the medical use of steroids.

3 In some database systems, specialized event capture routines are added to the database to detect and replicate I/O events as they occur. The most common technique uses database trigger capture or a similar method. Triggers are a useful approach as they are granted access to the event data as it occurs rather than having to await a commit. However, they extend the source application's transaction path length because trigger work is typically performed under the source transaction. Triggers are generally used for specialized and localized replication environments since they typically require very high-speed interconnects so as to minimize the impact on the source application processing.

being able to track changes to a source database via some sort of change queue. The change queue may be created by the database manager, by the application, or via other techniques such as database triggers. In NonStop systems, the change queue is the TMF Audit Trail, which records every change made to the source database.

As shown in Figure 1, the replication engine provides an Extractor process that follows the change queue and reads each change that was made to the source database. Changes are sent over a communication channel to an Applier process that applies the changes to the target database.

Depending upon the nature of the change queue, a "change" may be an entire transaction (as in most database management systems) or a single update within a transaction (as in the NonStop TMF Audit Trail).

Since changes occur at the source database and are independently applied to the target database after they were made to the source database, this method of replication is called asynchronous replication.[4] The target database lags the source database by a short time. This delay is called replication latency. Another form of replication, synchronous replication, avoids replication latency but has other limitations. Synchronous replication is not considered in this paper.[5]

Database Consistency

In many applications, the target database must be available for use during the replication process. If it is the database of a standby system, it may be used for queries and reporting. If it is a database in an active/active system, it must be able to provide read/write activity to local applications while at the same time supporting updates by one or more replication engines.

Therefore, the target database must always be in a consistent state. A consistent relational database must satisfy the following constraints:[6]

- Primary Key: The rows in every table in the database must each have a unique primary key.
- Referential Integrity: Every child row must have a parent row. That is, every foreign key in a row must be the primary key of an existing row.
- Data Constraints: User-defined relationships between rows or columns in the same or different tables must be correct.
- Data Validity: The value of every data field must reflect the latest update for that field.

In transaction processing systems, related updates are included within the scope of a transaction. The ACID[7] properties of a transaction guarantee that the source database is left in a consistent state following the commit of each transaction. It is the responsibility of the transactional replication engine to ensure that this same consistency is enforced at the target database.

Natural Flow

Consistency is achieved if the natural flow of the source system's update sequence is preserved. In other words, the sequence of updates made to the target database is the same as that sequence at the source database. If updates are applied in random order, older updates might overwrite newer updates, child rows may exist without parent rows, and so on. The database contents would be wrong and in some cases unusable. In short, the target database would not be a consistent copy of the source database.

The simple replication engine of Figure 1 guarantees natural flow and a consistent target database because changes are sent over the replication channel to the target database in the same order as they were applied to the source database. Therefore, changes are applied to the target database in the same order, thus guaranteeing consistency.

Within a transaction-processing system, there are two types of natural flow to be considered – intertransaction natural flow and intratransaction natural flow.

Intertransaction Natural Flow

Intertransaction natural flow is the more important of the two. Intertransaction natural flow requires that transactions are committed in the same order on the target system as they were committed on the source system.

Data being modified within the scope of a transaction is generally not available to applications (except for the case of "dirty reads"). However, once the transaction is committed, the data affected by the transaction is now available to the target applications.

If transactions are allowed to commit out of order, the consistency constraints are easily violated. For instance, a newer update may be overwritten by an older update. A child row may be inserted before its parent row is inserted. The sum of a field in a set of rows may not equal the accumulated value contained in another row.[8]

Intratransaction Natural Flow

Since the results of updates within the scope of a transaction are not viewable to applications until the transaction commits, the violation of natural flow order of a specific transaction's updates is often not a problem.

4 Chapter 3, Asynchronous Replication, Breaking the Availability Barrier: Survivable Systems for Enterprise Computing, AuthorHouse; 2004.
5 Chapter 4, Synchronous Replication, Breaking the Availability Barrier: Survivable Systems for Enterprise Computing, AuthorHouse; 2004.
6 Holenstein, Paul J., Holenstein, Bruce D., and Highleyman, Wilbur H., Referential integrity, consistency, and completeness loading of databases, U.S. Patent 7,949,640: May 24, 2011.
7 Atomic – all updates are made, or none are. Consistent – each transaction leaves the database in a consistent state. Independent – the execution of a transaction is unaffected by other transactions being simultaneously executed. Durable – the results of a transaction survive any subsequent fault. For additional information, see J. Gray and A. Reuter, Transaction Processing Concepts and Techniques, Morgan Kaufmann; 1993.
8 This is not third-normal form but is often implemented for efficiency reasons.
9 In NonStop systems, changes to a single file or table partition are always handled by the same DP2 disk process and are recorded in the proper order in the Audit Trail. However, changes to file or table partitions handled by different DP2 processes will be written to the Audit Trail in an indeterminate order. Thus, intratransaction referential integrity is generally not available on NonStop systems except when the related I/O events affect data in the same partition, unless some form of sequence field is added by the application; and the replication engine reorders events into this sequence before applying them to the target database.

However, there are some special cases in which the violation of natural flow within a transaction could lead to the contamination of the database. For instance, if a transaction makes multiple updates to the same field, and if the updates are not applied to the target database in the proper order, the data field may be left with an older value.

If the target database enforces referential integrity, but the source database does not, transactions that have completed successfully on the source database may be aborted on the target database.[9]

Serial versus Concurrent Replication

Today's asynchronous replication technologies encompass three general types of replication:

Hardware replication, which is performed at the database system level by replicating disk blocks. These replication products are not transaction-oriented and are not considered further.

Serial software replication, in which all updates within a transaction are held in the change queue and are not available to the replication engine until commit time.

Concurrent software replication, in which updates within a transaction as well as the transaction commit are available to the replication engine as soon as they are applied to the source database.

Serial Replication

With serial replication, a transaction is typically not visible to the replication engine until it is committed at the source database. The entire transaction is then accessible in the change queue and is replicated to the target system.

Only one transaction is actively being replicated and applied to the target database at any given time. The replication of all data within a transaction must wait until all previously committed transactions are replicated. Transactions are applied in the proper order since the commits are received from the change queue in the order they were committed at the source.

Most database management systems impose serial replication on the replication facility, since the database changes are not available until the transaction is committed.

Concurrent Replication

HP NonStop systems support concurrent replication. When a source database update is made, TMF enters it into the Audit Trail, from where it is immediately accessible by the replication engine. Concurrent replication reads updates as they become available in the Audit Trail and immediately transmits and applies them to the target database.

Therefore, when the target system receives the source's commit for a transaction that is being replayed into the target database, the transaction's updates were already applied to the target database under a target transaction that was begun upon the receipt of the first update or upon the receipt of an explicit begin-transaction command. All that is necessary is to commit the target transaction. Transactions are committed in natural flow order because the commits are applied to the target database in natural flow order. Consequently, intertransaction natural flow is ensured.

As opposed to serial replication, the replication engine typically has many transactions in progress at any one time. The transaction mix at the target matches the transaction mix that originally occurred at the source, offset by the replication latency time lag. This mix has the desirable property of having the target database go through the same simultaneous data changes as the source did, offset by the replication lag time.

Comparing Serial and Concurrent Replication

Both techniques have strengths and weaknesses. An advantage of serial replication is that aborted transactions have no impact on the target database. They are simply not applied. In addition, serial replication is simpler to implement since only one transaction is being actively replicated at a time. However, since the replication and application of the data, whether serial or concurrent, are performed by today's commercially available replication engines, concurrent replication should pose no challenge for end users.

The Achilles' heel of serial replication is its inability to have more than one transaction at a time in progress. This inability has four important negative impacts – increased RPO (recovery point objective – the potential loss of transactions should the source system fail), uneven loading of the replication communication channel and the target system, a negative impact from long-running transactions, and the lack of extensibility. Concurrent replication solves these problems.

Recovery Point Objective

With serial replication, the replication process typically does not begin until the source system has committed the transaction. At that point, all of the updates within the scope of the transaction must be extracted from the change queue,[10] sent to the target system, and applied to the target database. Therefore, the replication latency is the time required to extract all of the updates from the source system, send them to the target system, apply them to the target database, and commit them.[11]

The latency time to replicate a transaction of any size with concurrent replication is simply the time to replicate the commit. The reason is that the events inside that transaction were already replicated and applied immediately after they occurred on the source.

This high degree of overlap substantially reduces RPO when concurrent replication is used. For instance, if a transaction has four updates, serial replication will cause the target system to lag the source system by five event replication times (four updates plus a commit). Concurrent

10 As noted earlier, some database engines do not even make events visible until they are committed; Oracle's Log Miner is an example.

11 In some cases, vendors moved the change queue to the target system. By doing this move, the event replication time has been shortened by eliminating communication time.

replication will cause the target system to lag the source system by only one event replication time (the commit). This time lag represents the amount of data that might be lost should the source system fail. Therefore, the RPO for serial replication in this example will be about five times greater than the RPO for concurrent replication.

Target System Loading

With serial replication, the replication load on the target system comes in bursts. The replication channel is idle until a transaction commits on the source system. At that point, all of the updates and the commit are sent to the target system as a complete transaction. The target system proceeds from being idle to suddenly being busy applying all of these updates, and then its replication activity disappears until the next complete transaction is received.

With concurrent replication, replication activity follows the natural flow of update activity at the source database. Whenever an update or a commit is made at the source database, it is also made at the target database. The database update activity at the target system is substantially the same as it is at the source system.

Thus, with concurrent replication, database I/O activity is as smooth at the target system as it is at the source system, and the same procedures used to tune the source system are applicable to the target system. With serial replication, the "bursty nature" of replication at the target system imposes peak loads not seen at the source system, and totally different tuning procedures are required.

The bursty nature of serial replication also impacts the communication channel. Rather than imposing a fairly smooth load on the replication communication channel, serial replication communication comes in bursts, which causes queuing at the communication channel.

The uneven loading imposed on both the communication channel and the target system by serial replication further increases the RPO penalty. To reduce this penalty, one must significantly increase the communication channel capacity and the CPU and disk capacity on the target system to offset the loss of parallelism that concurrent replication provides. This increase allows each event replication to be completed faster, thus offsetting the cost of idle time on the communication channel and at the target system.

Large Transactions

Many applications mix normal transaction activity and batch processing. Normal transaction activity is characterized by transactions with only a few updates each. Batch processing often binds hundreds, thousands, or even millions of updates within a single transaction.

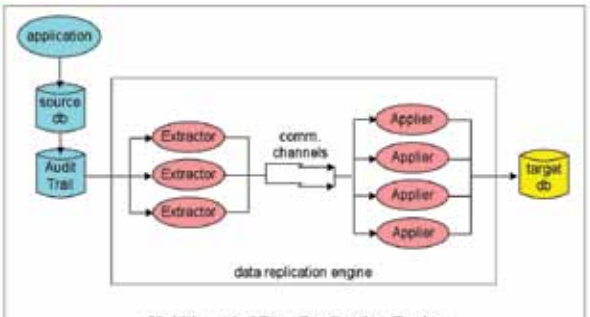
This type of transaction causes a significant problem when serial replication is used. When the batch transaction commits, the replication channel is often committed to that transaction until the hundreds or thousands or millions of updates are sent and applied to the target database. During that time, newly committed transactions must queue at the source system and wait until the batch transaction has committed before they are replicated.

For instance, assume that the replication channel handles 500 events per second (two milliseconds per event) to transmit and apply events to the target database. A normal transaction with four updates and a commit takes ten milliseconds to replicate with serial replication. If a batch transaction comes along with 100,000 updates, the replication channel could be committed to that batch transaction for 200 seconds (over three minutes). Transactions that commit after the commit of the batch transaction may be "stuck" on the source side and delayed up to 200 seconds. This type of transaction not only delays the state of the target system substantially, but it significantly increases RPO (the amount of data that is lost), since in this example, over three minutes of transactions could be lost should the source system fail during the batch replication.

Because concurrent replication follows the natural flow of transactions from the source database, all 100,000 updates were already sent and applied to the target database when the batch transaction committed. It only takes one event time – two milliseconds – to replicate the commit in order to complete the replication of the batch transaction. Other transactions that completed while the large batch transaction was in progress are replicated at the same time as the batch data. Large transactions have no negative impact on replication when concurrent replication is used.

Extensibility

What if the volume of data that needs to be replicated exceeds the capacity of the data replication channel? The bottleneck could be the Extractor, the communication channel, or the Applier. If the capacity of the replication channel needs to be increased, the component representing the bottleneck must be multithreaded so that two or more components share the load, thus removing the bottleneck, as shown in Figure 2. Depending upon where the bottleneck is, one may have to configure two or more Extractors, two or more communication channels, two or more Appliers, or any combination of them.



Multithreaded Data Replication Engine
Figure 2

With serial replication, multithreading has no value. Only one transaction is in progress at a time. There is work only for one Applier and one Extractor. Serial replication

12 Chapter 10, Referential Integrity, Breaking the Availability Barrier: Survivable Systems for Enterprise Computing, AuthorHouse; 2004.

13 NonStop systems have had this advantage for decades. It is only now that some database vendors are realizing the advantages of this capability.

Data Replication on Steroids

continued from pg.

engines are not readily extensible without the possibility of the Appliers colliding on data being applied, potentially causing data collisions.

Concurrent replication takes advantage of multiple replication threads since many transactions are being sent and applied simultaneously. However, now changes flow over multiple paths from the source database to the target database. If nothing is done, there is no guarantee that commits would arrive at the target database in the same order that they occurred at the source database. The natural flow of transactions would be disturbed, and the target database may be left in an inconsistent state.

This problem is solved via intelligent Appliers. An Applier must coordinate with its peers to ensure that it does not commit a transaction for which it is responsible before all prior transactions are committed.[12]

Summary

Concurrent replication has important advantages over serial replication. It provides shorter RPOs and is not impacted by long transactions that bring replication to a halt in a serial replication engine. It produces the same database activity profile on the target system, whereas serial replication converts the natural flow of updates to a bursty profile. Concurrent replication engines are extensible and scalable by making them multithreaded, whereas serial replication engines are bound by a single thread and are generally not extensible without compromising target database integrity.

Concurrent replication depends upon the ability to replicate the natural flow of transaction activity from the source database to the target database. This ability requires that source database updates are immediately available to the replication engine once they are made. Most database management systems do not make the updates within the scope of a transaction available until the transaction commits. Only NonStop systems make updates immediately available in the Audit Trail to third-party ISV replication engines. Only NonStop supports the many advantages of concurrent replication. Only NonStop supports data replication on steroids.[13]

If the benefits of concurrent replication are important to your application environment, look for replication products that allow you to exploit concurrent replication's many advantages.