

## Estimating Data Collision Rates

August 2007

### Asynchronous Replication and Data Collisions

Active/active systems rely in large part on bidirectional asynchronous data replication to keep the database copies in the application network synchronized.

With asynchronous replication, changes to the database are logged in a Change Queue of some sort. This Change Queue can be, for instance, the transaction log maintained by a transaction monitor, a log of changes maintained by the application, or a log of changes built via database triggers. As changes are entered into the Change Queue at the source system, they are subsequently sent to one or more target systems, where they are applied to the target's database.

There is necessarily a time delay between when a change is posted to the source database and when it is applied to the target database. This time delay is called replication latency. During this time, data collisions may occur if bidirectional replication is being used (as it will be in an active/active system).

A data collision occurs when the same record or row is changed at two different systems within the replication latency time. Neither system is yet aware that the other system is making a change to that record or row. The updated record or row is replicated by each system to the other system, where it overwrites the original change made by that system. The result is that the record or row is different in each system, and both are wrong.

There are several techniques that one may find applicable for avoiding or for automatically resolving data collisions. Many of these are covered in our earlier article, [Collision Detection and Resolution](#), April, 2007. If data collisions cannot be avoided or automatically resolved, they must be resolved manually.

Manual resolution can be a time consuming task. It is therefore often desirable to be able to estimate what the data collision rate for an application will be before entering into an active/active project. In this article, we derive a simple equation for doing just that and discuss the proper ways to use it.

This result was originally published by Jim Gray,<sup>1</sup> and this article relies heavily upon his work with some extensions.

---

<sup>1</sup> Gray, J.; et al.; "The Dangers of Replication and a Solution," *ACM SIGMOD 96*, pp. 173-182; June, 1996.

## The Result

For those readers who do not wish to dig through the mathematics of this derivation, we present the result first. It is

$$\text{collisions per second} = \frac{(d-1) u^2}{d D} L$$

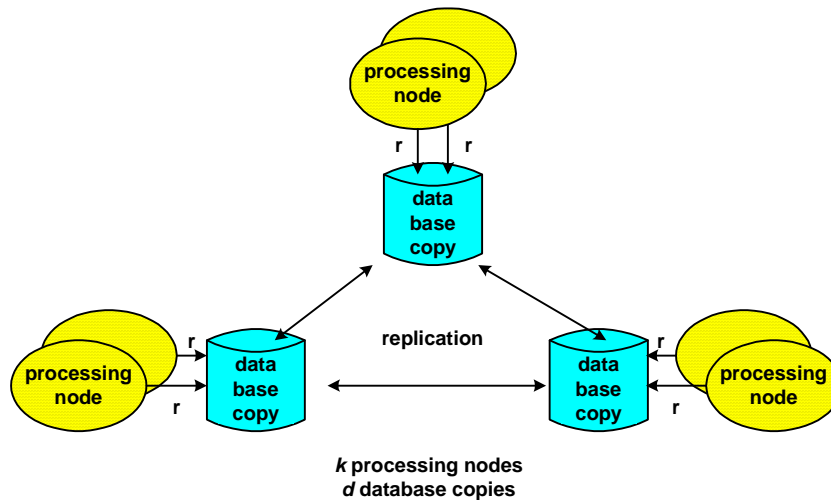
where

- $d$  is the number of database copies in the application network.
- $u$  is the total system row or record update rate (updates/second).
- $D$  is the size of the database in terms of rows or records.
- $L$  is the replication latency time (seconds).

Note that the collision rate increases as the square of the database update rate,  $u$ . It increases linearly with replication latency,  $L$ , and decreases with the database size,  $D$ . It is a monotonically increasing function of the number of database copies,  $d$ , in the application network, starting at zero for one database copy ( $d = 1$ ) since there are no collisions if there is only a single database copy and asymptotically approaching one as the number of database copies increases.

## The Model

The active/active architecture which we will analyze comprises an application network of  $k$  processing nodes and  $d$  database copies (for instance, six processing nodes with three database copies). It makes no difference whether the database copies are directly attached to some of the processing nodes, are configured as network attached storage (NAS), are geographically distributed databases in a storage area network (SAN), or are any combination of these.



**An Active/Active Application Network**

The database comprises a series of files and/or tables containing a total of  $D$  rows or records that are subject to updating. Each database copy in the application network contains a copy of all of these files and tables. It is initially assumed that update activity is uniformly distributed across the entire database. The analysis is extended later to cover the case of nonuniform update distribution and "hot spots."

Each of the  $k$  nodes is assigned to a specific database copy. Each node generates  $r$  updates per second and applies these updates to its assigned database copy. These updates are then replicated asynchronously to the other  $d-1$  database copies in the application network, where they are applied after a time delay equal to the replication latency,  $L$ . It is assumed that the system is balanced so that each database copy is fed by an equal number of nodes (i.e., there are  $k/d$  processing nodes assigned to each database copy).

Based on the above, the system update rate is the number of processing nodes multiplied by the nodal update rate. That is,  $u = kr$ .

## The Analysis

Consider a database copy A. It is being updated by  $k/d$  nodes. Each node is generating  $r$  updates per second. Thus, database copy A is being updated at a rate of  $kr/d$  updates per second. During a replication time,  $L$ , database copy A will receive  $krL/d$  updates:

$$\text{primary updates to a database copy during a replication interval} = \frac{k}{d}rL$$

During the same replication latency interval, the number of updates being made to the other  $(d-1)$  database copies in the application network is:

$$\text{updates made to the other database copies during a replication interval} = (d-1)\frac{k}{d}rL$$

The updates made to the database copies are distributed over the entire database of  $D$  rows and records. The probability that a specific update made to database copy A will collide with another update made to one of the other database copies during the replication interval is equal to the number of updates made to the other database copies during this interval divided by the size of the database:

$$\begin{aligned} &\text{probability that a specific update to database A will collide with an update} \\ &\text{made at another database copy} = \frac{(d-1)k}{D} \frac{rL}{d} \end{aligned}$$

Since database copy A has made  $krL/d$  updates during the replication latency interval  $L$ , the number of collisions in other database copies caused by database A updates is

$$\text{number of collisions created by updates to database copy A} = \left(\frac{k}{d}rL\right) \left[\frac{(d-1)k}{D} \frac{rL}{d}\right] = \frac{(d-1)}{D} \left(\frac{k}{d}rL\right)^2$$

Database copy A is just one of  $d$  nodes that is making updates. Therefore, the total number of collisions in the application network is

$$\text{number of collisions in the application network} = d \frac{(d-1)}{D} \left(\frac{k}{d}rL\right)^2 = \frac{(d-1)}{d} \frac{1}{D} (krL)^2$$

This is the number of collisions that will be generated during the replication latency time  $L$ . Dividing this by  $L$  gives us the collision rate:

$$\text{collision rate in the application network} = \frac{1}{L} \frac{(d-1)}{d} \frac{1}{D} (krL)^2 = \frac{(d-1)}{d} \frac{1}{D} (kr)^2 L$$

As noted earlier, the term  $kr$  is the system update rate  $u$ . This relationship can therefore be written as

$$\text{collision rate in the application network} = \frac{(d-1)u^2}{d} \frac{L}{D}$$

## Examples

Let us consider an application network with three database copies supporting a network-wide update rate of 100 updates per second to a database containing 10,000,000 rows and/or records. Let us also assume a replication latency time of 0.5 seconds. The network-wide collision rate will be

$$\text{collision rate} = \frac{2(100)^2}{3 \cdot 10^7} \cdot 0.5 = .00033 \text{ collisions/second} = 1.2 \text{ collisions/hour}$$

If there were only two database copies in the application network, the collision rate would be 0.9 collisions per hour.

For three database copies, if the update rate were to increase to 1,000 updates per second, the collision rate would increase to 120 collisions per hour, or two per minute. Carrying this further, if the database size were only one million records or rows, the collision rate would be 1,200 collisions per hour, or one every three seconds.

## Hot Spots

We have assumed in this analysis that all updates may cause collisions and that collisions are evenly distributed across the entire database. However, nature is not generally so kind. There are several cases where care must be taken in choosing the parameters which we have used.

### ***File/Table Hot Spots***

It is quite likely that some files or tables will be updated much more frequently than others and therefore will contribute a disproportionate share to data collisions. It is therefore wise to calculate the data collision rate individually for each file and table. This affects the choice of values for the update rate,  $u$ , and for the file or table size,  $D$ . The update rate should be the rate at which the specific file or table is updated; and the database size,  $D$ , should be the number of records or rows in that file or table.

### ***Dormant Rows or Records***

Some files or tables may contain records or rows that are unlikely to be updated, whereas the other records or rows may be actively updated. For instance, in a general ledger file, accounts for the current month may be frequently updated; whereas accounts for previous months exist for reporting and audit purposes only and may be infrequently updated, if at all.

In such a case, the file or table size  $D$  should reflect just that portion of the file or table which will be actively updated.

### ***Not all Actions Collide***

It is quite possible that not all update actions will cause collisions. Inserts probably will not cause collisions. Application dependencies may make certain kinds of updates relatively collision-free.

For instance, transactions against a nonreloadable gift card will probably not cause collisions since there is only one gift card. There cannot be (at least legally) more than one transaction at a time in progress for that card.

In this case, the file or table update rate,  $u$ , should only reflect those updates that can create collisions.

## Summary

Data collisions are an important and undesirable byproduct of bidirectional asynchronous replication. They should be thoroughly understood and analyzed before venturing into an active/active system implemented with asynchronous replication, especially if automatic collision resolution is not possible for a set of collisions that can happen at a high rate.

Using our previous example, if manual resolution is required, one collision per hour may well be acceptable. Two collisions per minute would certainly keep a team of people busy. If there is going to be a collision every three seconds, automatic collision resolution would probably be a requirement.

Though synchronous replication will avoid data collisions, an equivalent mechanism will create network deadlocks instead due to lock latency, the synchronous equivalent of asynchronous replication latency. This is Rule 32 of *Breaking the Availability Barrier*.<sup>2</sup>

*Lock latency deadlocks under synchronous replication become collisions under asynchronous replication.*

Chapter 9 of that volume, Data Conflict Rates, delves much more deeply into this topic. It analyzes data collisions during asynchronous replication and network deadlocks during synchronous replication. A variety of replication engine architectures are studied, including those that broadcast updates as soon as they are applied to the source database (our model in the above analysis), those that broadcast updates only following a successful commit, and those that send updates serially to the database copies rather than by broadcast.

---

<sup>2</sup> W. H. Highleyman, P. J. Holenstein, B. D. Holenstein, *Breaking the Availability Barrier: Survivable Systems for Enterprise Computing*, AuthorHouse; 2004.