25.2.6. Synchronous Replication

Streaming replication is by default asynchronous. Transactions on the primary server write commit records to WAL, yet do not know whether or when a standby has received and processed those changes. So with asynchronous replication, if the primary crashes, transactions committed on the primary might not have been received by any standby. As a result, failover from primary to standby could cause data loss because transaction completions are absent, relative to the primary. The amount of data loss is proportional to the replication delay at the time of failover.

Synchronous replication offers the ability to guarantee that all changes made by a transaction have been transferred to at least one remote standby server. This is an extension to the standard level of durability offered by a transaction commit. This is referred to as semi-synchronous replication.

When synchronous replication is requested, the commit of a write transaction will wait until confirmation that the commit record has been transferred successfully to at least one standby server. Waiting for confirmation increases the user's confidence that the changes will not be lost in the event of server crashes but it also necessarily increases the response time for the requesting transaction. The minimum wait time is the roundtrip time from primary to standby.

Read only transactions and transaction rollbacks need not wait for replies from standby servers. Subtransaction commits do not wait for responses from standby servers, only final top-level commits. Long running actions such as data loading or index building do not wait until the very final commit message.

25.2.6.1. Basic Configuration

Synchronous replication must be enabled on both the primary and at least one standby server. If synchronous replication is disabled on the master, or enabled on the primary but not enabled on any slaves, the primary will use asynchronous replication by default.

We use a single parameter to enable synchronous replication, set in `postgresql.conf` on both primary and standby servers:

```
synchronous_replication = off (default) | on
```

On the primary, `synchronous_replication` can be set for particular users or databases, or dynamically by applications programs.

If more than one standby server specifies `synchronous_replication`, then whichever standby replies first will release waiting commits.

Turning this setting off for a standby allows the administrator to exclude certain standby servers from releasing waiting transactions. This is useful if not all standby servers are designated as potential future primary servers. On the standby, this parameter only takes effect at server start.

25.2.6.2. Planning for Performance

Synchronous replication usually requires carefully planned and placed standby servers to ensure applications perform acceptably. Waiting doesn't utilise system resources, but transaction locks continue to be held until the transfer is confirmed. As a result, incautious use of synchronous replication will reduce performance for database applications because of increased response times and higher contention.

PostgreSQL allows the application developer to specify the durability level required via replication. This can be specified for the system overall, though it can also be specified for specific users or connections, or even individual transactions.

For example, an application workload might consist of: 10% of changes are important customer details, while 90% of changes are less important data that the business can more easily survive if it is lost, such as chat messages between users.

With synchronous replication options specified at the application level (on the master) we can offer sync rep for the most important changes, without slowing down the bulk of the total workload. Application level options are an important and practical tool for allowing the benefits of synchronous replication for high performance applications. This feature is unique to PostgreSQL.

25.2.6.3. Planning for High Availability

The easiest and safest method of gaining High Availability using synchronous replication is to configure at least two standby servers. To understand why, we need to examine what can happen when you lose all standby servers.

Commits made when synchronous_replication is set will wait until at least one standby responds. The response may never occur if the last, or only, standby should crash or the network drops. What should we do in that situation?

Sitting and waiting will typically cause operational problems because it is an effective outage of the primary server. Allowing the primary server to continue processing in the absence of a standby puts those latest data changes at risk. How we handle this situation is controlled by `allow_standalone_primary`. The default setting is `on`, allowing processing to continue, though there is no recommended setting. Choosing the best setting for `allow_standalone_primary` is a difficult decision and best left to those with combined business responsibility for both data and applications. The difficulty of this choice is the reason why we recommend that you reduce the possibility of this situation occurring by using multiple standby servers.

When the primary is started with `allow_standalone_primary` enabled, the primary will not allow connections until a standby connects that also has `synchronous_replication` enabled. This is a convenience to ensure that we don't allow connections before write transactions will return successfully.

When `allow_standalone_primary` is set, a user will stop waiting once the `replication_timeout` has been reached for their specific session. Users are not waiting for a specific standby to reply, they are waiting for a reply from any standby, so the unavailability of any one standby is not significant to a user. It is possible for user sessions to hit timeout even though standbys are communicating normally. In that case, the setting of `replication_timeout` is probably too low.

The standby sends regular status messages to the primary. If no status messages have been received for `replication_timeout` the primary server will assume the connection is dead and terminate it. This happens whatever the setting of `allow_standalone_primary`.

If primary crashes while commits are waiting for acknowledgement, those transactions will be marked fully committed if the primary database recovers, no matter how `allow_standalone_primary` is set. There is no way to be certain that all standbys have received all outstanding WAL data at time of the crash of the primary. Some transactions may not show as committed on the standby, even though they show as committed on the primary. The guarantee we offer is that the application will not receive explicit acknowledgement of the successful commit of a transaction until the WAL data is known to be safely received by the standby. Hence this mechanism is technically "semi synchronous" rather than "fully synchronous" replication. Note that replication still not be fully synchronous even if we wait for all standby servers, though this would reduce availability, as described previously.

If you need to re-create a standby server while transactions are waiting, make sure that the commands to run pg_start_backup() and pg_stop_backup() are run in a session with synchronous_replication = off, otherwise those requests will wait forever for the standby to appear.

18.5.5. Synchronous Replication

These settings control the behavior of the built-in *synchronous replication* feature. These parameters would be set on the primary server that is to send replication data to one or more standby servers.

`synchronous_replication` (`boolean`)

> Specifies whether transaction commit will wait for WAL records to be replicated before the command returns a "success" indication to the client. The default setting is `off`. When `on`, there will be a delay while the client waits for confirmation of successful replication. That delay will increase depending upon the physical distance and network activity between primary and standby. The commit wait will last until the first reply from any standby. Multiple standby servers allow increased availability and possibly increase performance as well.
>
> The parameter must be set on both primary and standby.
>
> On the primary, this parameter can be changed at any time; the behavior for any one transaction is determined by the setting in effect when it commits. It is therefore possible, and useful, to have some transactions replicate synchronously and others asynchronously. For example, to make a single multistatement transaction commit asynchronously when the default is synchronous replication, issue `SET LOCAL synchronous_replication TO OFF` within the transaction.
>
> On the standby, the parameter value is taken only at server start.

`synchronous_replication_timeout` (`boolean`)

> If the client has `synchronous_replication` set, and `allow_standalone_primary` is also set, then the commit will wait for up to `synchronous_replication_timeout` milliseconds before it returns a "success", or will wait forever if `synchronous_replication_timeout` is set to -1.
>
> If a standby server does not reply for `synchronous_replication_timeout` the primary will terminate the replication connection.

`allow_standalone_primary` (`boolean`)

> If `allow_standalone_primary` is not set, then the server will not allow connections until a standby connects that has `synchronous_replication` enabled.
>
> `allow_standalone_primary` also affects the behaviour when the `synchronous_replication_timeout` is reached.

25.5.2. Handling query conflicts

….

Remedial possibilities exist if the number of standby-query cancellations is found to be unacceptable. Typically the best option is to enable `hot_standby_feedback`. This prevents `VACUUM` from removing recently-dead rows and so cleanup conflicts do not occur. If you do this, you should note that this will delay cleanup of dead rows on the primary, which may result in undesirable table bloat. However, the cleanup situation will be no worse than if the standby queries were running directly on the primary server. You are still getting the benefit of off-loading execution onto the standby and the query may complete faster than it would have done on the primary server. `max_standby_archive_delay` must be kept large in this case, because delayed WAL files might already contain entries that conflict with the desired standby queries.

…

18.5.6. Standby Servers

These settings control the behavior of a standby server that is to receive replication data.

`hot_standby` (`boolean`)

> Specifies whether or not you can connect and run queries during recovery, as described in Section 25.5. The default value is `off`. This parameter can only be set at server start. It only has effect during archive recovery or in standby mode.

`hot_standby_feedback` (`boolean`)

> Specifies whether or not a hot standby will send feedback to the primary about queries currently executing on the standby. This parameter can be used to eliminate query cancels caused by cleanup records, though it can cause database bloat on the primary for some workloads. The default value is `off`. This parameter can only be set at server start. It only has effect if `hot_standby` is enabled.

….