Framework Deployment Guide

Transaction Serialization

5/6/2025

Transaction Serialization

Configuration Server can process data change (write) requests so that concurrent data changes are processed correctly and do not interfere with each other. Transaction serialization prevents data change transactions from overlapping and potentially causing a loss of data integrity. It involves the deferral of transaction data change requests so that each request can be processed completely, without impacting, or being impacted by, other requests.

> **Important**
>
> Transaction Serialization might be required when a non-Genesys application has been used to modify the configuration.

This section describes several ways that you can use transaction serialization to preserve data integrity without jeopardizing system performance. You must enable transaction serialization before you can configure any of the features described in this section.

Overview

Without transaction serialization, Configuration Server starts processing data change requests as soon as they arrive, regardless of any other requests that are being processed. If the first request being processed changes something upon which another request being processed depends, problems can include corrupt data, application faults, or even system shutdown.

Transaction serialization solves this problem by ensuring that only one data change request per session is being processed at a time. Configuration Server can accept new requests while processing a request, but does not start the next one until the first request is finished. Data write and data read requests are stored in the same queue and processed in FIFO (first-in, first-out) order. Data read requests from a client whose previous write requests are deferred must also wait for the preceding request to finish before being processed. This ensures that a client can view updated data from the database.

There is one queue for each client making a request. Configuration Server and Configuration Server Proxy clients are treated the same, so there is no potential for overlap and interference between clients.

Connect/Authorize requests are an exception. They are not considered transaction requests and Configuration Server processes them immediately. Internal requests are not meant to be deferred, even those that generate data change. They are processed immediately, but the change notification sent to the clients is deferred. Some of the internal requests that do not get deferred are:

- Updates to last-login and last-login synchronization information
- Password changes

- Forced password changes as a result of the Change password on next login feature

Enabling Transaction Serialization

To enable transaction serialization, set the option serialize-write-transactions in the **[system]** section of the Configuration Server Application object to `true`. If this option is not configured or is set to `false`, requests are not deferred.

> ### Warning
> Use extreme caution if you decide to disable transaction serialization after having first enabled it. Doing so will release all deferred requests and Configuration Server might become overwhelmed. Genesys recommends that you not disable this feature unless asked to.

Expiration of Deferred Requests

> ### Important
> Transaction serialization must be enabled (serialize-write-transactions is set to `true`) to use any of the options in this section.

Requests are classified into three categories: general clients' requests; requests from Configuration Server Proxies reconnecting to the master Configuration Server; and internal requests, generated by Configuration Server itself when processing other requests. Expiration and handling is different for each type, as described later in this section.

Deferred requests from disconnected clients are removed from the queue. If the client disconnected from the master Configuration Server, a response is sent immediately that the client has disconnected. General and Configuration Server Proxy requests are discarded. Internal requests are processed.

General Requests

To specify the time interval after which general requests expire, set deferred-requests-expiration to the desired time interval, in seconds. A value of `0` (zero) means that the requests never expire.

If a deferred general request has not been processed during the specified time interval, it expires. The request is removed from the queue, and the requesting client notified with the following message

```
ERROR CFGReadOnlyOperationalModeActivated/Transaction expired
```

Deferred non-transaction requests, such as read requests, that expire are removed from the queue and processed as usual.

Requests from Configuration Server Proxies in Process of Reconnecting

Requests coming from Configuration Server Proxies in the process of reconnecting have much higher expiration time, determined automatically, based on the value of proxy-load-timeout.

If the deferred request from a reconnecting Configuration Server Proxy expires, the proxy server is disconnected, and must reconnect to the master Configuration Server from the start.

Expiration of Internal Requests from Configuration Server

Internal requests are password change requests triggered by the Change Password On Next Login feature, or an overridden account expiration because of login and last-login notifications. These requests never expire, and are processed even if the corresponding clients disconnected while the requests were deferred.

Transaction Throttling

Transaction throttling addresses the problem of poor system performance because of, for example, an accumulation of data change notifications generated by massive data updates in the outgoing data buffers of Configuration Server and/or in the incoming data buffers of the clients, or a low bandwidth connection between Configuration Server and its clients (including Configuration Server Proxy). As a result, there was sometimes a significant delay in notification processing, excessive memory usage, and ultimately disconnections due to slow client responses.

When the throttling feature is enabled, unprocessed transaction requests are deferred in the same way as they are during **transaction serialization**. Processing of the deferred requests occurs in the same manner (that is FIFO, complete one request before starting the next). The functionality is invisible except for some delay due to the deferral. Data read requests from other clients who do not submit transaction requests, remain unaffected in both the timing of the processing of their requests and their processing order.

Implementation

To enable transaction throttling, do the following options in the **[system]** section of the Configuration Server Application:

1. Ensure that transaction serialization is enabled (serialize-write-transactions is set to `true`). If transaction serialization is disabled, transaction throttling cannot be used.
2. Set throttle-updates-interval in the **[system]** section of the Configuration Server Application to the desired time interval.

> **Tip**
> When configuring this option, keep in mind that if actual load consistently exceeds the rate specified by this option for a significant time, deferred unprocessed requests will accumulate in the input queue and will be eventually cancelled as defined by the value of the deferred-requests-expiration option. To avoid this happening, consider

> adjusting throttle-updates-interval accordingly, to account for the expected actual load.

Transaction Deferral for Primary or Backup Configuration Server Proxy Loading and Reloading Data

This feature defers all transaction data requests whenever a primary or backup Configuration Server Proxy is loading (or reloading) data from the master Configuration Server. This ensures that the notifications related to the load request are sent only after the full load is completed. Otherwise, because of the size of the data being loaded, notifications might be sent before the loading is complete, potentially harming the data in the proxy server's memory. After the loading is complete, all deferred requests are released (in FIFO order), and processed.

Implementation

This feature is always enabled, so long as transaction serialization is also enabled (serialize-write-transactions=`true`).