# Performance Related Configuration Settings

The following settings have a significant impact on Flux performance. Setting them to these recommendations optimizes the engine for performance, at the expense of functionality, loss of history, or logging details.

- If you are not seeing the amount of concurrency you expect (always <=10 but you are expecting more) - consider that having a runtime configuration file (that is referenced within the engine configuration properties file) impacts where concurrency is set for a Flux engine.
  - First, make absolutely sure that the engine configuration properties file entry for RUNTIME_CONFIGURATION_FILE points to a valid and accessible runtime configuration properties file.
- If a runtime configuration file is defined in the engine configuration, concurrency is set using CONCURRENCY_THROTTLE in the runtime configuration file.
- If no runtime configuration file is configured, concurrency is set using CONCURRENCY_LEVEL in the engine configuration file.
- If a runtime configuration file is being used, and no CONCURRENCY_THROTTLE is set within that runtime configuration file, the engine will run with it's concurrency set to 10 regardless of what the engine's CONCURRENCY_LEVEL is set to.

- Check the size of the **FLUX_AUDIT_TRAIL** table. If it is > 100,000 rows consider truncating it and reducing the number of audit trail records being written by using the **AUDIT_TRAIL_FILTER** setting. Large audit trail tables can slow overall Flux performance.
- Check that the **FLUX_CLUSTER** table does not have old or duplicated entries. Truncating the **FLUX_CLUSTER** table while the engine is down, and then restarting the engine, can improve performance if old entries are present.
- Set **SERVER=false** (In most cases this is not an option since this disables the Flux engine's REST API, Flux agents, and the operations console web application)
- Set **INTERNAL_LOGGER_LEVEL=INFO** (to reduce the amount of logging)
- Make sure - if using the default Flux internal logger - that it is set as follows: **logger_types.0=INTERNAL_ASYNCHRONOUS.** Having it set to **INTERNAL_SYNCHRONOUS** will make blocking-writes to the log, which is useful in developing and debugging workflows but not in production.
- Set **CLUSTER_NETWORKING_ENABLED=false** (but note that this disables agents and disables the bytes sent/received displayed on the console while file transfers are executing).
- Set **CACHE_TYPE=NONE** to stop using the cache for workflows.
- Minimize or eliminate the use of prescripts and postscripts since these involve the interactive execution of code within workflows and reduce performance
- Set **RUN_HISTORY_ENABLED=false** to turn off run history data collection
- Set **FLOW_CHART_DEADLINES_ENABLED=false** to turn off checking for flowchart deadlines
- Set **AUDIT_TRAIL_FILTER.0=** (and no other audit trail filters are present or all others are commented out)
- Reduce the property **SYSTEM_DELAY=+3m** to **SYSTEM_DELAY=+5s**. The SYSTEM_DELAY is the maximum amount of time the engine sleeps when it has nothing to do (i.e., it has completed all workflows or it has just started up). Reducing this time makes the system more aggressive in looking for work to process in between intervals when all work has finished.
- (For builds of 8.0.13) If potentially executing many workflows that can be shared across a Flux cluster, and those workflows initiate at the same time based on their timer triggers, set the property **RANDOM_JOB_SELECT=TRUE.** This property randomizes the returned rows to reduce the contention between engines. If the workflows are returned to both engines in the same order at the same time, the engines attempt to process the same workflows at the same time - which Flux addresses by stopping the second (or greater) engine from claiming the workflow. This creates a lot of needless database contention. The reason RANDOM_JOB_SELECT is not by default set to TRUE is that some customers reply on the order returned for their application-specific processing.

Specific to SQL Server Installations:

Using the connection string parameter selectMethod=cursor was required in old releases of SQL Server. In more recent versions, it is only needed if large result sets are being returned back to the client (generally more than 100 records at a time), and only if the client has insufficient memory to handle the returned result set. If this is turned on, the server returns results at 100 rows per call but at the expense of server performance. In testing Flux, the performance hit seems to be around 35-40% for Puts by themselves as illustrated in the SelectMethod.png attachment below. Puts and execution is also improved - as seen in SelectMethod 2.png. Removing this parameter from the SQL Server connection string should provide some significant performance gain.

Per the Microsoft website at http://msdn.microsoft.com/en-us/library/ms378988.aspx :

'If this property is set to "cursor," a database cursor is created for each query created on the connection for TYPE_FORWARD_ONLY and CONCUR_READ_ONLY cursors. This property is typically required only if the application generates very large result sets that cannot be fully contained in client memory. When this property is set to "cursor," only a limited number of result set rows are retained in client memory. The default behavior is that all result set rows are retained in client memory. This behavior provides the fastest performance when the application is processing all rows.'