**Threadpool model for Continuous Query:**

Continuous Queries are the queries that are registered in the query engine and are executed till the query or the engine is stopped. Each Continuous Query (CQ) is run in a separate thread. But CQ spends a lot of time waiting for various external events like window expiry, cache update, etc. Threads allotted for each CQ is tied up even when the CQ is waiting idle.

Threadpool model for Continuous Query has a common threadpool which is used to execute all the continuous queries. When the Continuous Query needs to wait for external events, they are moved to wait queue and the thread is freed for use by other continuous queries that need thread for execution. This model enables the query engine to use very less thread resources compared to Thread per CQ model.

BE Query Engine allows the user to choose the Continuous Query Threading model that is suitable for their needs. Users can easily switch between the two threading models by defining the continuous query threadpool size as the Query Engine startup parameter. When a value greater than zero is entered for the threadpool size, Threadpool model for continuous query is used while a value less than or equal to zero makes the Query Engine to use Thread per CQ model. The engine startup property that is used to choosing the CQ Threading model is given below:

be.agent.query.continuous.threadpoolsize=<Threadpool size>

**Filter Optimization using Index aware Filters:**

Query Engine optimizes the query execution by pushing down all the filter conditions down to the source. Query Engine serializes the filter conditions and sends it over the wire to each of the Cache Server. Filter conditions are deserialized and are reconstructed on the Cache Server. Filtering is done by iterating over all the records in the cache and applying the filter conditions on each of the record and choosing all the records that satisfy the filter conditions. This enables the query engine to bring only the data that satisfy the filter conditions and improves the performance of the query engine manyfolds by reducing the amount of data that is transferred over the network.

By introducing Index Aware filter usage, the record filtering process on the cache server can be made to take advantage of any index defined on the concept. The Filter predicates are decomposed into a tree of primitive comparison and logical operators like Equals, Greater than, Less than, And, Or, etc and each of these operators are mapped to their equivalent Index aware filter implementation offered by the Cache library. These Index aware filters are serialized and sent to the cache server where the cache library uses the index to gather all the records that matches the filter conditions and is sent over the network to the query engine. By using the index aware filters, the time taken for filtering the records that match the filter conditions is improved by orders of magnitude. Index Aware Filter optimizer can be enabled by setting the following Query Engine property to true in the startup command/cdd properties.

be.agent.query.enable.filter.optimizer=<true/false>