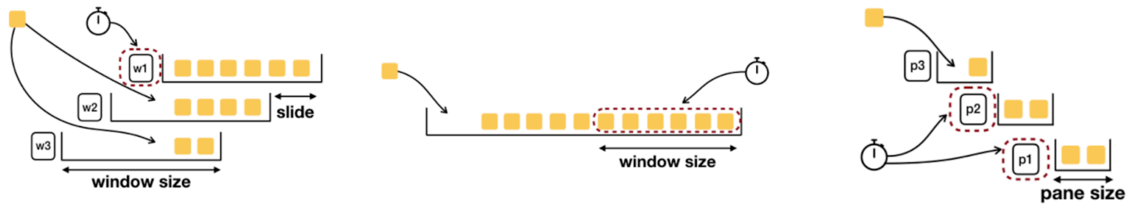
Window evaluation strategies in Flink using the RocksDB state backend

Introduction

Window operators are integral to streaming applications. They enable evaluation of blocking operators, such as aggregations and joins on streams. Further, they allow expressing computations on the most recent stream history, so that applications can be continuously updated with fresh results. Windowing splits an unbounded input stream into a series of bounded sets of records, which we simply refer to as *windows*. Flink's default window operator represents each window as a key-value pair on RocksDB, where the key is a unique bucket id and the value represents the window contents. While this representation is simple, it is not always efficient, especially when windows have overlap. In this project, you will design, implement, and evaluate alternative window operators in Apache Flink and compare their performance with Flink's default windowing mechanism.

Background

We define a window operator with a *size* parameter, which indicates how many time units belong to a window, and an optional *slide* parameter, which indicates how often a new window starts. A particular window can be identified by its `start` and `end` timestamps. A window *evaluation function* defines the computation logic to be applied to the window contents. We say that a window *triggers* when the system's notion of time arrives at its end timestamp. Evaluation functions can be applied eagerly, upon receiving a new record that belongs to the window, or lazily, on trigger.
A window operator is responsible for grouping incoming records into buckets and making the evaluation function results available in the output whenever a window triggers. The figure below shows three alternative window strategies. Regardless of the strategy used, the operator performs two types of processing: (i) upon receiving a new record on its input, and (ii) upon receiving a trigger.



Window ID: On *record*, map to corresponding windows. On *trigger*, find the window ID.

Record buffer: On *record*, append to state. On *trigger*, find records within window bounds.

Slicing: On *record*, map to corresponding pane. On *trigger*, find the pane IDs within window bounds.

The **window ID** strategy organizes records into windows by assigning them IDs (start or end timestamp). When a record arrives at the input, the operator calls an assigner function that computes a list of at most `size/slide` window IDs the record belongs to. The record is inserted to the corresponding state of each window in the list. On trigger, window contents can be easily retrieved using the ID.

The **record buffer** strategy stores incoming records in a buffer ordered by timestamp. When a window triggers, the operator scans the buffer and retrieves all records whose timestamp falls inside the window bounds.

Slicing organizes records into smaller units, called *panes*. A pane is the maximum shareable unit across windows and its size is computed as `gcd(size, slide)`. This guarantees that a record belongs to only one pane and that every window can be composed by a set of consecutive panes. When a record arrives, the operator calls an assigner function that computes the pane ID and adds the record to its state. On trigger, it retrieves `size/paneSize` panes to assemble the window contents. If the evaluation function supports pre-aggregation, it can be eagerly applied on record arrival to maintain partially aggregated results per pane. Those are combined into the global aggregate when the window fires.

Project goal

The goal of this project is to design and implement two alternative window operators in Flink using the *record buffer* and *slicing* strategies. You will have to think about how to best represent the state of these operators as key-value pairs in RocksDB and how to minimize latency when a new record arrives and when the window triggers. You will evaluate the performance of the new operators and Flink's default implementation with respect to the window length, ratio of length to slide, and using different window evaluation functions.

Required skill set

- Experience with Java programming.

Where to start

- Read the paper "Efficient Window Aggregation with General Stream Slicing": https://hpi.de/fileadmin/user_upload/fachgebiete/rabl/publications/2019/GeneralStreamSlicingEDBT2019.pdf. This paper evaluates different window strategies in memory. In this project, you will extend this analysis for window state backed by RocksDB.
- Clone the Flink source code and find the classes that implement the window operators. How is the state represented and accessed in RocksDB?