KIP-969: Support range interactive queries (IQv2) for versioned state stores

This page is meant as a template for writing a KIP. To create a KIP choose Tools->Copy on this page and modify with your content and replace the heading with the next KIP number and a description of your issue. Replace anything in italics with your own description.

Status

Current state: Under Discussion

Discussion thread: here

JIRA: KAFKA-15348

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

The main goal is to support interactive queries in the presence of versioned state stores (KIP-889) in AK. This KIP is the successor of KIP-960 and KIP-968 For this KIP, the following query types are considered to be implemented:

Range Queries

1. key-range latest-value query
2. key-range with lower bound latest-value query
3. key-range with upper bound latest-value query
4. all-keys (no bound) latest-value query
5. key-range query with timestamp (upper) bound
6. key-range with lower bound with timestamp (upper) bound
7. key-range with upper bound with timestamp (upper) bound
8. all-keys (no bound) with timestamp (upper) bound
9. key-range query with timestamp range
10. key-range query with lower bound with timestamp range
11. key-range query with upper bound with timestamp range
12. all-keys (no bound) with timestamp range
13. key-range query all-versions
14. key-range query with lower bound all-versions
15. key-range query with upper bond all-versions
16. all-keys query (no bound) all-versions (entire store)

Public Interfaces

In this KIP we propose the public classes, **MultiVersionedRangeQuery** that will be described in the next section. More over a method will be added to the **VersionedKeyValueStore** interface.

Proposed Changes

For supporting range queries, **MultiVersionedRangeQuery** class is used.

- The methods are composable. The *fromTime* and *toTime* methods specify the time range while the *withLowerKeyBound* and *withUpperKeyBound* methods specify the key bounds.
    - If a user applies the same time limit multiple times such as MultiVersionedRangeQuery.withLowerKeyBound(k1).from(t1).from(t2), then the last one wins (it will be translated to MultiVersionedRangeQuery.withLowerKeyBound(k1).from(t2)).
    - Defining a query with time range (empty, t1] will be translated into [0, t1]
    - Defining a query with time range (t1, empty) will be translated into [t1, MAX)
    - A query with no specified time range will be translated into [0, MAX). It means that the query will return all the versions of all the records with specified key range.
- As explained in the javadocs, the query returns all valid records within the specified time range.

    - The *fromTime* specifies the starting point. There can be records which have been inserted before the *fromTime* and are valid in the time range.
    - The *toTime* specifies the ending point. Records that have been inserted at *toTime* are returned by the query as well.
- No ordering is guaranteed for the return records.

MultiVersionedRangeQuery

```
package org.apache.kafka.streams.query;


/**
 * Interactive query for retrieving a set of records with keys within a specified key range and time
 * range.
 */

@Evolving
public final class MultiVersionedRangeQuery<K, V> implements
    Query<KeyValueIterator<K, VersionedRecord<V>>> {

  private final Optional<K> lower;
  private final Optional<K> upper;
  private final Optional<Instant> fromTime;
  private final Optional<Instant> toTime;

  private MultiVersionedRangeQuery(
      final Optional<K> lower,
      final Optional<K> upper,
      final Optional<Instant> fromTime,
      final Optional<Instant> toTime) {
    this.lower = lower;
    this.upper = upper;
    this.fromTime = fromTime;
    this.toTime = toTime;
  }

   /**
   * Interactive range query using a lower and upper bound to filter the keys returned. * For each
   * key the records valid within the specified time range are returned. * In case the time range is
   * not specified just the latest record for each key is returned.
   * @param lower The key that specifies the lower bound of the range
   * @param upper The key that specifies the upper bound of the range
   * @param <K> The key type
   * @param <V> The value type
   */
   public static <K, V> MultiVersionedRangeQuery<K, V> withKeyRange(final K lower, final K upper);


   /**
   * Interactive range query using a lower bound to filter the keys returned. * For each key the
   * records valid within the specified time range are returned. * In case the time range is not
   * specified just the latest record for each key is returned.
   * @param lower The key that specifies the lower bound of the range
   * @param <K>   The key type
   * @param <V>   The value type
   */
  public static <K, V> MultiVersionedRangeQuery<K, V> withLowerKeyBound(final K lower);

  /**
   * Interactive range query using a lower bound to filter the keys returned. * For each key the
   * records valid within the specified time range are returned. * In case the time range is not
   * specified just the latest record for each key is returned.
   * @param upper The key that specifies the lower bound of the range
   * @param <K>   The key type
   * @param <V>   The value type
   */
  public static <K, V> MultiVersionedRangeQuery<K, V> withUpperKeyBound(final K upper);

  /**
   * Interactive scan query that returns all records in the store. * For each key the records valid
   * within the specified time range are returned. * In case the time range is not specified just
```

```
   * the latest record for each key is returned.
   * @param <K> The key type
   * @param <V> The value type
   */
  public static <K, V> MultiVersionedRangeQuery<K, V> allKeys();

  /**
   * Specifies the starting time point for the key query. The range query returns all the records
   * that are valid in the time range starting from the timestamp {@code fromTime}.
   * @param fromTime The starting time point
   */
  public MultiVersionedRangeQuery<K, V> fromTime(Instant fromTime);

  /**
   * Specifies the ending time point for the key query. The range query returns all the records that
   * have timestamp <= {@code toTime}.
   * @param toTime The ending time point
   */
  public MultiVersionedRangeQuery<K, V> toTime(Instant toTime);

  /**
   * The lower bound of the query, if specified.
   */
  public Optional<K> lowerKeyBound();

  /**
   * The upper bound of the query, if specified
   */
  public Optional<K> upperKeyBound();

  /**
   * The starting time point of the query, if specified
   */
  public Optional<Instant> fromTime();

  /**
   * The ending time point of the query, if specified
   */
  public Optional<Instant> toTime();

}
```

Examples

The following example illustrates the use of the VersionedKeyQuery class to query a versioned state store.

Imagine we have the following records

put(1, 1, time=2023-01-**01**T10:00:00.00Z)

put(1, null, time=2023-01-**05**T10:00:00.00Z)

put(2, 20, time=2023-01-**10**T10:00:00.00Z)

put(3, 30, time=2023-01-**12**T10:00:00.00Z)

put(1, 2, time=2023-01-**15**T10:00:00.00Z)

put(1, 3, time=2023-01-**20**T10:00:00.00Z)

put(2, 30, time=2023-01-**25**T10:00:00.00Z)

```
// example 1: MultiVersionedRangeQuery without specifying any time bound will be interpreted as all versions
final MultiVersionedRangeQuery<Integer, Integer> query1 =
        MultiVersionedRangeQuery.withKeyRange(1, 2);
final StateQueryRequest<KeyValueIterator<Integer, VersionedRecord<Integer>>> request1 =
        inStore("my_store").withQuery(query1);
```

```
final StateQueryResult<KeyValueIterator<Integer, VersionedRecord<Integer>>> versionedRangeResult1 =
kafkaStreams.query(request1);

// Get the results from all partitions.
final Map<Integer, QueryResult<KeyValueIterator<Integer, VersionedRecord<Integer>>>> partitionResults =
versionedRangeResult.getPartitionResults();
for (final Entry<Integer, QueryResult<KeyValueIterator<Integer, VersionedRecord<Integer>>>> entry :
partitionResults.entrySet()) {
    try (final KeyValueIterator<Integer, VersionedRecord<Integer>> iterator = entry.getValue().getResult()) {
        while (iterator.hasNext()) {
          final KeyValue<Integer, VersionedRecord<Integer>> record = iterator.next();
          Integer key = record.key;
          Integer value = record.value.value();
          Long timestamp = record.value.timestamp();
                  Long validTo = record.value.validTo();
              System.out.println ("key,value: " + key + "," +value + ", timestamp: " + Instant.ofEpochSecond
(timestamp)+ ", valid till: " + Instant.ofEpochSecond(validTo));
        }
     }
}
/* the printed output will be
    key,value: 1,1, timestamp: 2023-01-01T10:00:00.00Z, valid till: 2023-01-05T10:00:00.00Z
    key,value: 1,2, timestamp: 2023-01-15T10:00:00.00Z, valid till: 2023-01-20T10:00:00.00Z
    key,value: 1,3, timestamp: 2023-01-20T10:00:00.00Z, valid till: now
    key,value: 2,20, timestamp: 2023-01-10T10:00:00.00Z, valid till: 2023-01-25T10:00:00.00Z
    key,value: 2,30, timestamp: 2023-01-25T10:00:00.00Z, valid till: now
*/

// example 2: The value of the records with key range (1,2) from 2023-01-17 Time: 10:00:00.00Z till 2023-01-30
T10:00:00.00Z


MultiVersionedRangeQuery<Integer, Integer> query2 = MultiVersionedRangeQuery.withKeyRange(1, 2);
query2 = query2.fromTime(Instant.parse(2023-01-17T10:00:00.00Z)).toTime(Instant.parse(2023-01-30T10:00:00.00Z));

final StateQueryRequest<KeyValueIterator<Integer, VersionedRecord<Integer>>> request2 =
        inStore("my_store").withQuery(query2);

final StateQueryResult<KeyValueIterator<Integer, VersionedRecord<Integer>>> versionedRangeResult2 =
kafkaStreams.query(request2);

// Get the results from all partitions.
final Map<Integer, QueryResult<KeyValueIterator<Integer, VersionedRecord<Integer>>>> partitionResults2 =
versionedRangeResult2.getPartitionResults();
for (final Entry<Integer, QueryResult<KeyValueIterator<Integer, VersionedRecord<Integer>>>> entry :
partitionResults.entrySet()) {
    try (final KeyValueIterator<Integer, VersionedRecord<Integer>> iterator = entry.getValue().getResult()) {
        while (iterator.hasNext()) {
          final KeyValue<Integer, VersionedRecord<Integer>> record = iterator.next();
          Integer key = record.key;
          Integer value = record.value.value();
          Long timestamp = record.value.timestamp();
                   Long validTo = record.value.validTo();
              System.out.println ("key,value: " + key + "," +value + ", timestamp: " + Instant.ofEpochSecond
(timestamp)+ ", valid till: " + Instant.ofEpochSecond(validTo));
                }
     }
}

/* the printed output will be
    key, value: 2,30, timestamp: 2023-01-25T10:00:00.00Z, valid till: now
    key, value: 1,3, timestamp: 2023-01-20T10:00:00.00Z, valid till: now
    key, value: 1,2, timestamp: 2023-01-15T10:00:00.00Z, valid till: 2023-01-20T10:00:00.00Z
    key, value: 2,20, timestamp: 2023-01-10T10:00:00.00Z, valid till: 2023-01-25T10:00:00.00Z
*/
```

Compatibility, Deprecation, and Migration Plan

- Since this is a completely new set of APIs, no backward compatibility concerns are anticipated.
- Since nothing is deprecated in this KIP, users have no need to migrate unless they want to.

Test Plan

The range interactive queries will be tested in versioned stored IQv2 integration test (like non-versioned range queries). Moreover , there will be unit tests where ever needed.

Rejected Alternatives

The initial plan was to provide ordering based on key and/or timestamp, which is removed from the KIP and may be provided by subsequent KIPs based on user demand.