KIP-747 Add support for basic aggregation APIs

Status

Current state: *"Under Discussion"*

Discussion thread: here

JIRA: TBD

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

Motivation

Analytics applications use aggregations over data as a fundamental building block. Streaming frameworks like Spark provide functions like min, max, sum and many more as part of the Spark SQL aggregations. Kafka streams library supports count as part of KGroupedStream, KGroupedTable, TimeWindowedKStream and SessionWindowedStream. For other aggregations, application developers need to develop their own custom aggregators. Most applications have standard templates for many such functions. In some cases, the implementation may not be optimal and it spreads across many applications which may become hard to debug.

This document proposes some of the basic aggregation functions that will be useful for many applications. As we gain experience with this, we can expand the APIs to include other functions.

Public Interfaces

Aggregation APIs

```
public interface KGroupedStream<K, V> {


        <VR extends Number> KTable<K, VR> min(Function<V, VR> func,
                                      final Materialized<K, Long, KeyValueStore<Bytes, byte[]>> materialized);
        <VR extends Number> KTable<K, VR> min(Function<V, VR> func, final Named named,
                                      final Materialized<K, Long, KeyValueStore<Bytes, byte[]>> materialized);




        <VR extends Number> KTable<K, VR> max(Function<V, VR> func,
                                      final Materialized<K, Long, KeyValueStore<Bytes, byte[]>> materialized);
        <VR extends Number> KTable<K, VR> max(Function<V, VR> func, final Named named,
                                      final Materialized<K, Long, KeyValueStore<Bytes, byte[]>> materialized);



        <VR extends Number> KTable<K, VR> sum(Function<V, VR> func,
                                      final Materialized<K, Long, KeyValueStore<Bytes, byte[]>> materialized);
        <VR extends Number> KTable<K, VR> sum(Function<V, VR> func, final Named named,
                                      final Materialized<K, Long, KeyValueStore<Bytes, byte[]>> materialized);


}

public interface TimeWindowedKStream<K, V> {
```

```
        <VR extends Number> KTable<Windowed<K>, VR> min(Function<V, VR> func,
                                      final Materialized<K, Long, WindowStore<Bytes, byte[]>> materialized);
        <VR extends Number> KTable<Windowed<K>, VR> min(Function<V, VR> func, final Named named,
                                      final Materialized<K, Long, WindowStore<Bytes, byte[]>> materialized);



        <VR extends Number> KTable<Windowed<K>, VR> max(Function<V, VR> func,
                                      final Materialized<K, Long, WindowStore<Bytes, byte[]>> materialized);
        <VR extends Number> KTable<Windowed<K>, VR> max(Function<V, VR> func, final Named named,
                                      final Materialized<K, Long, WindowStore<Bytes, byte[]>> materialized);



        <VR extends Number> KTable<Windowed<K>, VR> sum(Function<V, VR> func,
                                      final Materialized<K, Long, WindowStore<Bytes, byte[]>> materialized);
        <VR extends Number> KTable<Windowed<K>, VR> sum(Function<V, VR> func, final Named named,
                                      final Materialized<K, Long, WindowStore<Bytes, byte[]>> materialized);

}

public interface SessionWindowedKStream<K, V> {



        <VR extends Number> KTable<Windowed<K>, VR> min(Function<V, VR> func,
                                      final Materialized<K, Long, SessionStore<Bytes, byte[]>> materialized);
        <VR extends Number> KTable<Windowed<K>, VR> min(Function<V, VR> func, final Named named,
                                      final Materialized<K, Long, SessionStore<Bytes, byte[]>> materialized);



        <VR extends Number> KTable<Windowed<K>, VR> max(Function<V, VR> func,
                                      final Materialized<K, Long, SessionStore<Bytes, byte[]>> materialized);
        <VR extends Number> KTable<K, VR> max(Function<V, VR> func, final Named named,
                                      final Materialized<K, Long, SessionStore<Bytes, byte[]>> materialized);



        <VR extends Number> KTable<Windowed<K>, VR> sum(Function<V, VR> func,
                                      final Materialized<K, Long, SessionStore<Bytes, byte[]>> materialized);
        <VR extends Number> KTable<Windowed<K>, VR> sum(Function<V, VR> func, final Named named,
                                      final Materialized<K, Long, SessionStore<Bytes, byte[]>> materialized);

}

public interface KGroupedTable<K, V> {



        <VR extends Number> KTable<K, VR> sum(Function<V, VR> func,
                                      final Materialized<K, Long, KeyValueStore<Bytes, byte[]>> materialized);
        <VR extends Number> KTable<K, VR> sum(Function<V, VR> func, final Named named,
                                      final Materialized<K, Long, KeyValueStore<Bytes, byte[]>> materialized);
}
```

The scala interfaces will also be changed accordingly.

Proposed Changes

The functionality of the changes are self-explanatory. The new APIs return the min, max and sum of of the values in the current KStream or KTable. The API provides a functional interface (func) to extract the value for which the min/max/sum should be calculated. The return value (VR) is upper bounded to be a Number as the operations make sense only for subclasses of the Number.

The following variants of the API that are supported by **count** is not supported by min/max/sum.

KTable<K, Long> count();

KTable<K, Long> count(**final** Named named);

Count API assumes that the "valueSerde" is Long. In the case of min/max/sum we should be able to support the other Number types also. As the type of return value of the "func" is not known at the time of building the topology, the user has to explicitly provide this using the Materialized parameter. The valueSerde cannot be null. One alternate way of supporting the above forms is to assume some generic serde and at run time inspect the return value of the func and install the right Serde. This adds to the complexity and hence not supported.

Also KTable version of the API supports only "sum" and it does not support min/max. KTable aggregations has the notion of "adder" and "subtractor" where state can be subtracted when there is deletion or change to the previous value that is already aggregated. For min and max, this implies that we store all the old values which could be very expensive. Hence, they are not supported by KGroupedTable.

The aggregations have an initializer which initializes the aggregate to some starting value. This requires the aggregation type which is not explicitly passed in. One possible way to infer this using "valueSerde" which is not very desirable as the the application could be using its own implementation of Serdes. There are two possibilities

- Provide an explicit parameter
- Assume a default type e.g., Long

This is an open issue and the APIs will be modified after the mailing list discussion.

Implementation details are described below.

KGroupedStream , SessionWindowedKStream, TimeWindowedKStream, SlidingWindowedKStream

We can model the implementation similar to how "count" is implemented. GroupedStreamAggregateBuilder holds the definition for countInitializer and countAggregator. Similarly we can have initializer and aggregator definitions for min/max and sum.

KGroupedTable

We can model the implementation similar to how "count" is implemented. KGroupedTableImpl class holds the definition for countInitializer, countAdder and countSubtractor. Similarly we can have the definition for sum.

Compatibility, Deprecation, and Migration Plan

N/A

Rejected Alternatives

See the discussion above