Apache Flink Side Inputs Proposal

Ventura Del Monte (venturadelmonte at gmail dot com), initial idea and doc by Aljoscha Krettek (aljoscha at apache dot org)

This document aims to revitalise the discussion about side inputs which has been abandoned for a while and to get consensus about side inputs semantics from an end-user perspective. Moreover, we should like to show a bunch of use-cases where an early-stage POC is already giving good results, with the hope that end-users may eventually benefit from this new feature.
As already stated in the past, side input idea comes from Apache Beam and should not be confused with broadcast set because a side input is neither necessarily broadcasted among operators nor a set.

Motivation

A native support to side inputs would help in dealing with the following use-cases:

- Joining an high-throughput stream with one or more static datasets in order to filter/enrich/combine them accordingly to the end-user business logic.
- Joining an high-throughput stream with one or more slowly evolving datasets; just as above but the side input is slowly evolving over time.
- Joining the main input with a continuously updated side input, i.e.: a machine learning model.
- Windowed-based join: having a keyed main input and a keyed side input, merging two windows according to some criterion may be meaningful from an user point-of-view.
    - As a sub-case, the side input is broadcasted: every side window will be joined with every main input window.

Although only the last use-case explicitly refers to keyed streams, there should be no restriction about the type of main and side inputs in other situations.

Types of side inputs

A side input is basically a stream (*DataStream, KeyedStream, WindowedStream, AllWindowedStream*) that is plugged as additional input of one operator. There are different plugging techniques which are just exploiting Flink partitioning operations:

1. forwarded side input (if the underlying data stream has parallelism>1);
2. broadcasted side input;
3. keyed side input (using a key translator for mapping side input key values to main input key values).

These side inputs can be combined with a main input as described below:

SIDE → ↓ MAIN	**Data Stream**	**Keyed Stream**	**Windowed Stream**	**All Windowed Stream**
Data Stream	B/F	B/F	B/F	B(1 to N)
Keyed Stream	B/F	B/F/K	B/F/K	B(1 to N)
Windowed Stream	B/F	B/F/K	B/F/K	B(1 to N)
All Windowed Stream	B (n to 1)	B (n to 1)	B (n to 1)	B(1 to N)

B - broadcasted | F - forwarded | K - keyed

The *degree of parallelism of each stream* has to be taken into account when using side inputs, i.e., if a side input has parallelism=1 then forwarding to a parallel operator is not possible. Note that a keyed side input can be linked to a keyed main input in two ways (as long as the parallelism is the same): by mapping one key of the side input to 1+ key(s) of the main input or by just forwarding it according to data locality.
Nevertheless, random keying for machine learning must be considered.

Side inputs processing

According to the usage of the side input, this latter can be considered ready for processing when:

- Static side input: no main input record is processed unless all side input(s) items are loaded. Meanwhile these side items are loaded, also main input elements must be buffered due to being in a streaming environment.
- Slowly-evolving side input: once defined an *update precondition* (i.e.: update the side input every N hours, asynchronously) and a *loading strategy* (i.e.: load all the static side input or load a portion of the side input), then no main input record is processed unless all side input(s) are loaded according the *loading strategy*. The execution of main records may be blocked while updating the side input. The processing is blocked with different granularity: if the main input is windowed then the current window has to be fully processed in order to have the side input updated. Meanwhile, if the input is not windowed, side input is updated as soon as possible.
- Continuously updated side input: no main input record is processed unless at least one item for each side input stream is loaded.
- Windowed join: as the main and the side inputs may have a different window definition, there are 3 different joining strategies:
    - Same window definition: join the last main input window with the last side one and then wait for two new windows.
    - Main input window has a smaller time length: join the current main window with the last side input window.
    - Side input window has a smaller time length: opposite.

Side inputs proposed API changes

Proposed API for creating a side input:

```
public class StreamingExecutionEnvironment {
. . .
public <TYPE> SideInput<TYPE> newBroadcastedSideInput(DataStream<TYPE> sideIn);
public <TYPE> SideInput<TYPE> newForwardedSideInput(DataStream<TYPE> sideInput);
public <TYPE, SIDE_KEY> KeyedSideInput<TYPE, KEY>
newKeyedSideInput(KeyedStream<TYPE, KEY> sideInput);
public <TYPE, W extends Window> AllWindowedSideInput<TYPE, W>
newAllWindowedSideInput(AllWindowedSide<TYPE, W> sideInput);
```

```
. . .
}
```

Meanwhile with the following API it is possible to bind a side input to a main input:

```
public class DataStream <T> {
. . .
public <TYPE, SELF extends DataStream<T>> SELF withSideInput(SideInput<TYPE>
sideIn);
. . .
}

public class KeyedStream <T, KEY> {
. . .
public <TYPE, SELF extends KeyedStream<T, KEY>> SELF withSideInput(SideInput<TYPE>
sideIn);

public <TYPE, SIDE_KEY, SELF extends KeyedStream<T, KEY>> SELF
withSideInput(KeyedSideInput<TYPE, SIDE_KEY> sideIn, KeyTranslator<KEY, SIDE_KEY>
translator);
. . .
}

public class AllWindowedStream <T, W extends Window> {
. . .
public <TYPE, SELF extends AllWindowedStream<T, W>> SELF
withSideInput(SideInput<TYPE> sideIn);
. . .
}

public class WindowedStream <T, K, W extends Window> {
. . .
public <TYPE, SELF extends WindowedStream<T, K, W>> SELF
withSideInput(SideInput<TYPE> sideIn);
public <TYPE, SIDE_KEY, SELF extends WindowedStream<T, K, W>> SELF
withSideInput(KeyedSideInput<TYPE, SIDE_KEY> sideIn, KeyTranslator<K, SIDE_KEY>
translator);
. . .
}
```

Regarding how to lookup a side input from an user defined function, a `getSideInput` method can be added to the UDF RuntimeContext:

```
DataStream<String> sideSource = ...
SideInput<String> sideInput = env.newBroadcastedSideInput(sideSource);

main
  .map(new RichMapFunction<String, String>() {
     public String map(String value) throws Exception {
        Iterable<String> side = getRuntimeContext().getSideInput(sideInput);
        System.out.println("SEEING MAIN INPUT: " + value + " with " + side);
        return value;
     }
  })
  .withSideInput(sideInput)
;
```

While dealing with windowed stream, the best way to lookup a side input would be throughout the Window Function Context described in FLIP-2.

Link to initial document and discussion

The aforementioned initial document written by Aljoscha is available here meanwhile this is the link to Apache Flink Mailing List thread.