dataflow/operators/aggregate

Created	@September 14, 2021 6:14 PM
Tags	

StateMachine

We assign each key k a state s. It starts with a default value. We keep receives (k, v) pairs. We take a function that takes `(k, v, &mut s)` to update the state with the newly arrived (k, v) pair. For each (k, v) pair it takes, multiple outputs can be generated. It returns a data type that can yield an iterator.

For instance, for key value aggregation, we can return `Option<K, S>`, where `S` is the data type of the state.

We ensure that that for (k, v) pairs will be fed to the provided function to update state with a timely order. For a timestamp `t' > t`, all (k, v) pairs with timestamp t are already used to update the state. But we do not guarantee order in KV pairs with the same timestamp.

```
// stash if not time yet
// since the state depends on the order of fold call
// if the arriving KV pairs' time is not less than that in the frontier
// we must stash the KV pair for fold to process it later
// to ensure timely order
if notificator.frontier(0).less_than(time.time()) {
    pending.entry(time.time().clone()).or_insert_with(Vec::new).extend(vector.drain(..));
    notificator.notify_at(time.retain());
}
else {
    // else we can process immediately
    // each input message arrvies with a CapabilityRef
    // we can create a new capability for the output if we want
    // or we can just use this CapabilityRef to create a new capability
    // for the operator (output port) to retain
    let mut session = output.session(&time);
    for (key, val) in vector.drain(..) {
        let (remove, output) = {
            let state = states.entry(key.clone()).or_insert_with(Default::default);
            fold(&key, val, state)
        };
        if remove { states.remove(&key); }
        session.give_iterator(output.into_iter());
    }
}
```

Aggregate

This is just a intra-time key-value aggregator, while StateMachine is an inter-time version.

When all (k, v) pairs with a timestamp t is received, we call the `emit` function that takes a key and the corresponding agg value and emits an output of any type `R`.

```
notificator.for_each(|time,_,_| {
                if let Some(aggs) = aggregates.remove(time.time()) {
                    let mut session = output.session(&time);
                    for (key, agg) in aggs {
                        // emit
                        session.give(emit(key, agg));
                    }
                }
            });
```