Flink 中的时间语义和 Watermark

左元

2021 年 7 月 22 日

尚硅谷大数据组

主要内容

- Flink 中的时间语义
- 设置 Event Time
- 水位线（Watermark）
- Watermark 的传递、引入和设定

时间（Time）语义

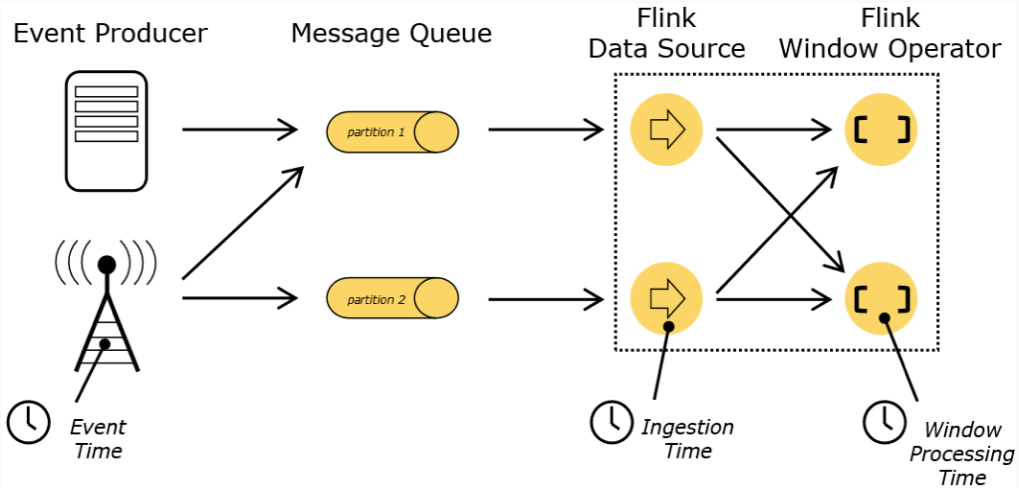

图 1: 时间语义

- Event Time（事件时间）：事件创建的时间（必须包含在数据源中的元素里面）
- Ingestion Time（摄入时间）：数据进入 Flink 的 source 算子的时间，与机器相关
- Processing Time（处理时间）：执行操作算子的本地系统时间，与机器相关

哪种时间语义更重要

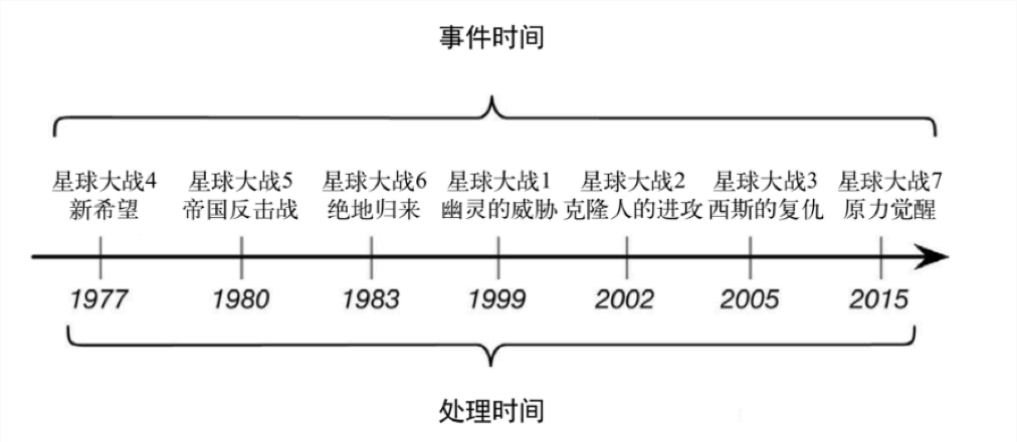

图 2: 星球大战

- 不同的时间语义有不同的应用场合
- 我们往往更关心事件时间（Event Time）

哪种时间语义更重要

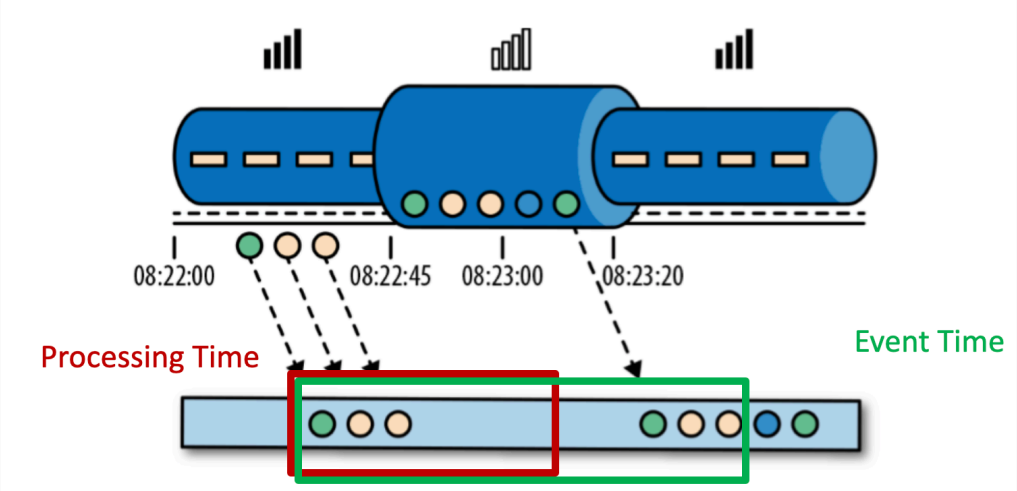

图 3: 打游戏

- 某些应用场合，不应该使用 Processing Time
- Event Time 可以从日志数据的时间戳（timestamp）中提取
  - 2017-11-02 18:37:15.624 INFO Fail over to rm
- Flink 1.12 默认使用事件时间，无需设置

乱序数据的影响

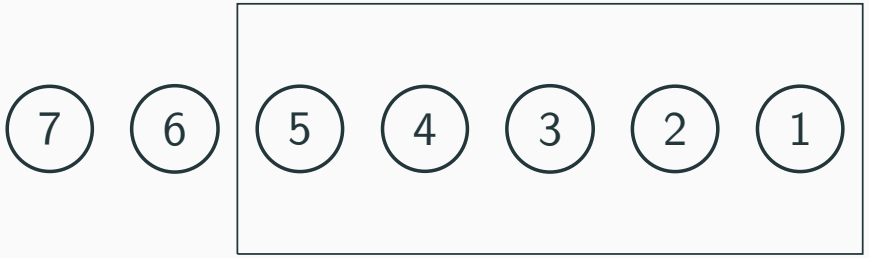

图 4: 理想情况

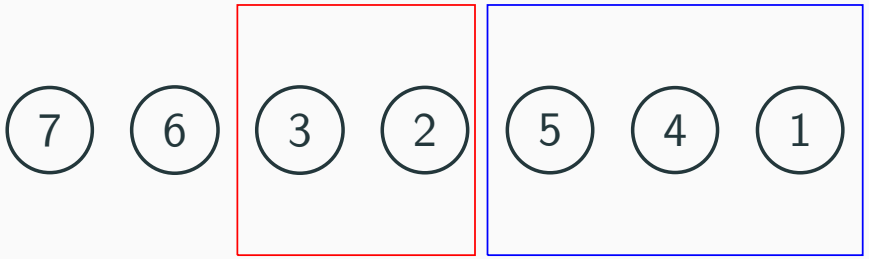

图 5: 乱序情况

乱序数据的影响

- 当 Flink 以 Event Time 模式处理数据流时，它会根据数据里的时间戳来处理基于时间的算子
- 由于网络、分布式等原因，会导致乱序数据的产生
- 乱序数据会让窗口计算不准确

水位线（Watermark）

- 怎样避免乱序数据带来计算不正确？
- 遇到一个时间戳达到了窗口关闭时间，不应该立刻触发窗口计算，而是等待一段时间，等迟到的数据来了再关闭窗口
- 要等多长时间？碰到含有 10000s 时间戳的事件，敢闭合 0s 5s 滚动窗口吗？

水位线（Watermark）

- Watermark 是一种衡量 Event Time 进展的机制（逻辑时钟），可以设定延迟触发
- Watermark 是用于处理乱序事件的，而正确的处理乱序事件，通常用 Watermark 机制结合 Window 来实现；
- 数据流中的 Watermark 用于表示 Timestamp 小于 Watermark 的数据，都已经到达了，因此，Window 的执行也是由 Watermark 触发的（水位线 >= 窗口结束时间）。
- Watermark 用来让程序自己平衡延迟和结果正确性。

水位线

Flink 认为时间戳小于等于水位线的事件都已到达

水位线

水位线是一种逻辑时钟

水位线

水位线由程序员编程插入到数据流中

水位线

水位线是一种特殊的事件

水位线

在事件时间的世界里，水位线就是时间

水位线

水位线 = 观察到的最大时间戳 - 最大延迟时间 - 1 毫秒

窗口

- 左闭右开区间
- $[0s, 5s]$ 不包含$5s$，其实是 $[0, 4999ms]$

水位线

水位线超过窗口结束时间，窗口闭合，默认情况下，迟到元素被抛弃

水位线

- Flink 会在流的最开始插入一个时间戳为负无穷大的水位线
- Flink 会在流的最末尾插入一个时间戳为正无穷大的水位线

Watermark 的特点

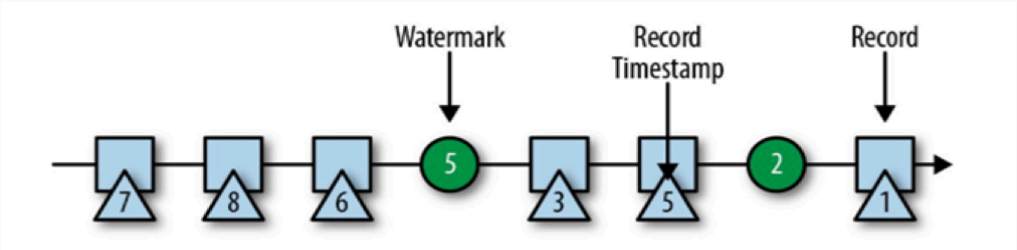

图 6: 水位线的特点

- Watermark 是一条特殊的数据记录，由程序员编程产生
- 水位线是流中的特殊事件，由程序员编程插入到数据流中
- Watermark 必须单调递增，以确保任务的事件时间时钟在向前推进，而不是在后退，(Watermark 就是当前数据流的逻辑时钟)
- Watermark 与数据的时间戳相关

Watermark 的传递

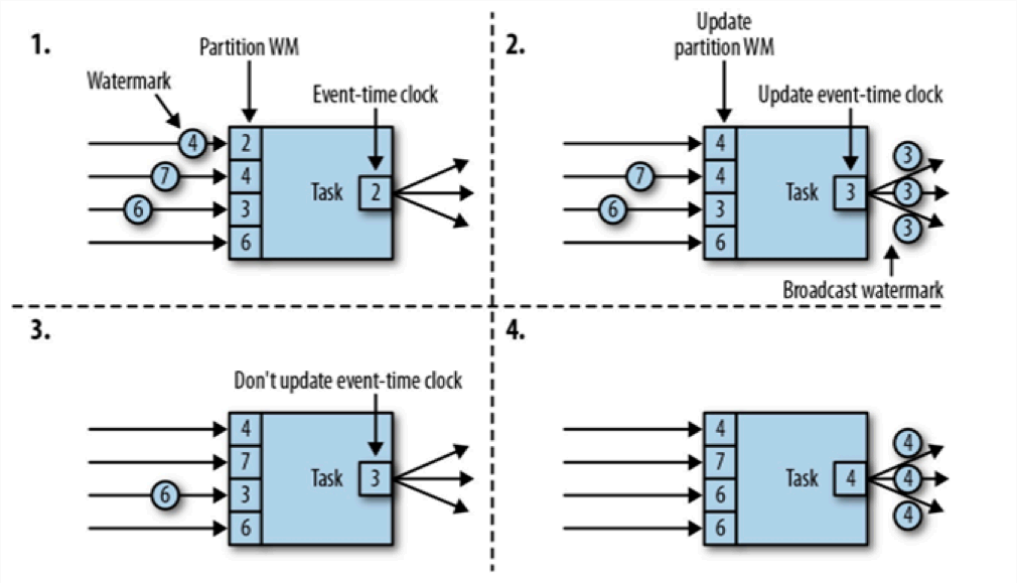

图 7: 水位线的特点

Watermark 的引入

- Event Time 的使用一定要指定数据源中的时间戳（单位是 ms）

```
.assignTimestampsAndWatermarks(
    WatermarkStrategy
        .<SensorReading>forBoundedOutOfOrderness(Duration.ofSeconds(5))
        .withTimestampAssigner(new SerializableTimestampAssigner<SensorReading>() {
            @Override
            public long extractTimestamp(SensorReading element, long recordTimestamp) {
                return element.timestamp;
            }
        })
)
```

Watermark 的引入

- 对于排好序的数据，不需要延迟触发，可以只指定时间戳就行了。

```
.assignTimestampsAndWatermarks(
    WatermarkStrategy.<SensorReading>forMonotonousTimestamps()
    .withTimestampAssigner(new SerializableTimestampAssigner<SensorReading>() {
        @Override
        public long extractTimestamp(SensorReading element, long l) {
            return element.timestamp;
        }
    })
);
```

自定义水位线

```java
@Public
public interface WatermarkGenerator<T> {

    /**
     * 每来一个事件都会调用，允许水位线产生器记忆和检查事件的时间戳。
     * 允许水位线产生器基于事件本身发射水位线。
     */
    void onEvent(T event, long eventTimestamp, WatermarkOutput output);

    /**
     * 周期性的调用（默认 200ms 调用一次），可能会产生新的水位线，也可能不会。
     *
     * 调用周期通过 ExecutionConfig#getAutoWatermarkInterval() 方法来配置。
     */
    void onPeriodicEmit(WatermarkOutput output);
}
```

周期性产生水位线

```java
public class BoundedOutOfOrdernessGenerator implements WatermarkGenerator<MyEvent> {

    private final long maxOutOfOrderness = 3500; // 最大延迟时间是 3.5s

    private long currentMaxTimestamp;

    @Override
    public void onEvent(MyEvent event, long eventTimestamp, WatermarkOutput output) {
        currentMaxTimestamp = Math.max(currentMaxTimestamp, eventTimestamp);
    }

    @Override
    public void onPeriodicEmit(WatermarkOutput output) {
        // 产生水位线的公式：观察到的最大时间戳 - 最大延迟时间 - 1ms
        output.emitWatermark(new Watermark(currentMaxTimestamp - maxOutOfOrderness - 1));
    }

}
```

不规则水位线的产生

```
1   public class PunctuatedAssigner implements WatermarkGenerator<MyEvent> {
2
3       @Override
4       public void onEvent(MyEvent event, long eventTimestamp, WatermarkOutput output) {
5           if (event.hasWatermarkMarker()) {
6               output.emitWatermark(new Watermark(event.getWatermarkTimestamp()));
7           }
8       }
9
10      @Override
11      public void onPeriodicEmit(WatermarkOutput output) {
12          // 不需要做任何事情，因为我们在 onEvent 方法中发射了水位线
13      }
14  }
```

Watermark 的设定

- 在 Flink 中，Watermark 由应用程序开发人员生成，这通常需要对相应的领域有一定的了解
- 如果 Watermark 设置的延迟太久，收到结果的速度可能就会很慢，解决办法是在水位线到达之前输出一个近似结果
- 而如果 Watermark 到达得太早，则可能收到错误结果，不过 Flink 处理迟到数据的机制可以解决这个问题

Q & A