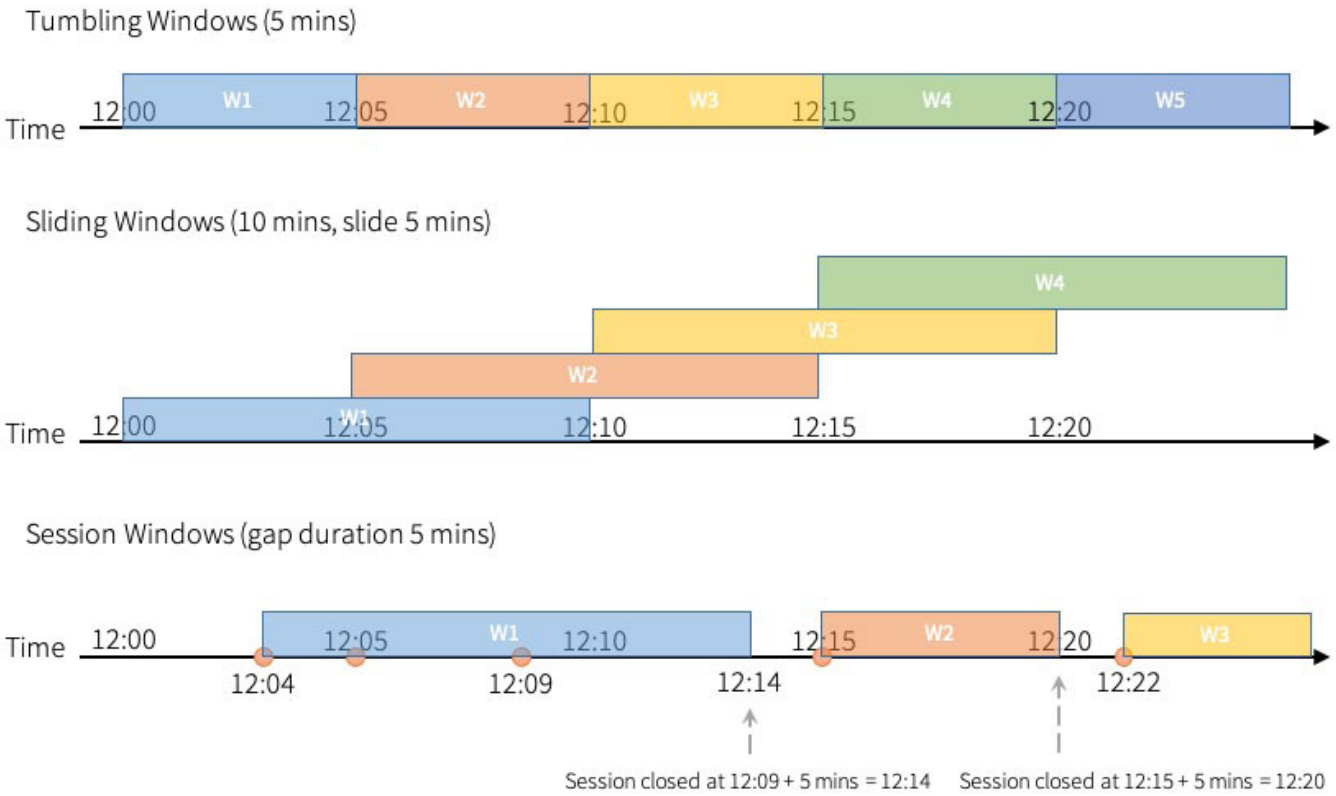
Apache Spark 3.2 内置支持会话窗口

Apache Spark™ Structured Streaming 允许用户在事件时间的窗口上进行聚合。 在 Apache Spark 3.2™ 之前，Spark 支持滚动窗口（tumbling windows）和滑动窗口（ sliding windows）。在已经发布的 Apache Spark 3.2 中，社区添加了“会话窗口（session windows）”作为新支持的窗口类型，它适用于流查询和批处理查询

什么是会话窗口



如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：过往记忆大数据

翻滚窗口是一系列固定大小、不重叠且连续的时间间隔。 一个输入只能绑定到一个窗口。

从“固定大小”的角度来看，滑动窗口类似于滚动窗口，但是如果滑动的持续时间小于窗口的持续时间，则窗口可以重叠，在这种情况下，可以将输入绑定到多个窗口。

与前两种类型相比，会话窗口具有不同的特征。会话窗口有一个动态大小的窗口长度，具体取决于输入。会话窗口以输入开始，如果在间隔持续时间内收到以下输入，则会话窗口将自身展开。如果在接收到最新输入后的间隔时间内没有收到任何输入，会话窗口就会关闭。这使您可以对事件进行分组，直到在指定的持续时间（不活动）内没有新事件。

它的工作原理类似于具有会话超时的网站上的会话 - 如果您登录网站并且在一段时间内没有显示

任何活动，则该网站将提示您保留登录状态并强制退出，如果您在这之后仍然不活动，那么会话就会关闭。但如果我们保持活动，会话超时就会延长。

将此应用于会话窗口：当新事件（例如流式作业）发生时，将启动新的会话窗口，并且在超时内的后续事件将包含在同一会话窗口中。每个事件都会延长会话超时时间，与其他时间窗口相比，它引入了不同的特性——会话窗口的持续时间不是静态的，而翻滚和滑动窗口的持续时间都是静态的。

如何在查询中使用会话窗口

在 Spark 3.2 版本之前，我们需要使用 flatMapGroupsWithState 来处理会话窗口。
您需要设计自己的逻辑来定义会话窗口以及如何在同一会话中聚合输入。 这带来了几个缺点：

- 我们不能利用内置的聚合函数，如 count, sum 等，并必须自己实现它们。
- 考虑到各种输出模式和输入的延迟，实现逻辑并非易事。
- PySpark 中没有 flatMapGroupsWithState，因此，我们需要通过 Java/Scala 精心设计查询。

现在，Spark 提供了与使用时间窗口（time windows）相同的用户体验。“在 Structured Streaming 中，在事件时间上表达这样的窗口只是执行一个特殊的分组”这句话仍然是正确的。对于滚动和滑动窗口，请使用 window 函数；对于会话窗口，引入了一个新的函数 session_window。

例如，在事件中的 eventTime 列上进行 5 分钟的滚动（非重叠）窗口可以描述如下。

```
# tumbling window
windowedCountsDF = ₩
  eventsDF ₩
    .withWatermark("eventTime", "10 minutes") ₩
    .groupBy(“deviceId”, window("eventTime", "10 minutes") ₩
    .count()

# sliding window
windowedCountsDF = ₩
  eventsDF ₩
    .withWatermark("eventTime", "10 minutes") ₩
    .groupBy(“deviceId”, window("eventTime", "10 minutes", "5 minutes")) ₩
    .count()
```

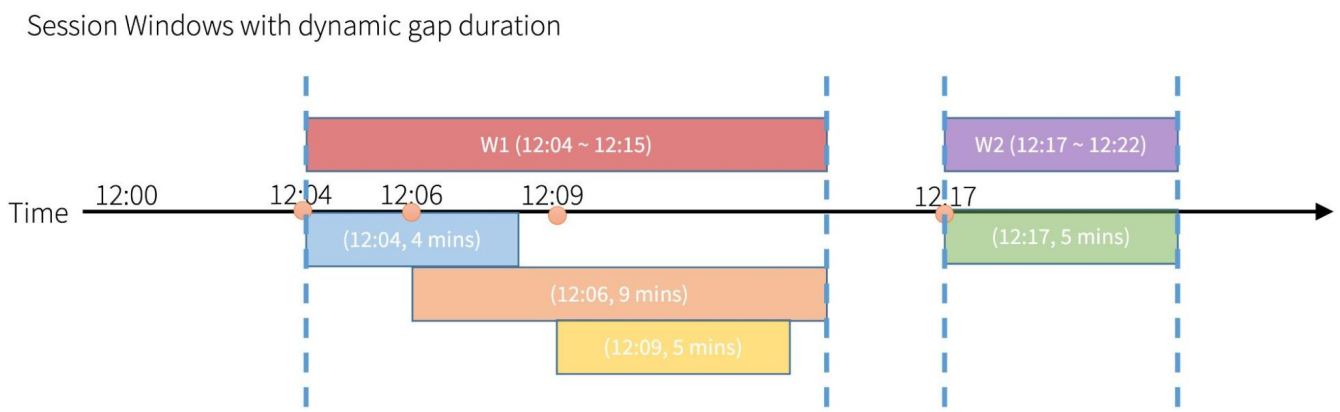
可以将 window 替换成 session_window 实现会话窗口：

```
# session window
```

```
windowedCountsDF = ₩
  eventsDF ₩
    .withWatermark("eventTime", "10 minutes") ₩
    .groupBy("deviceId", session_window("eventTime", "5 minutes")) ₩
    .count()
```

具有动态间隙持续时间的会话窗口

会话窗口除了具有相同的跨会话间隔持续时间之外，还有另一种类型的会话窗口，每个会话具有不同的间隔持续时间。 我们称之为“动态间隙持续时间（dynamic gap duration）”。



如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：过往记忆大数据

时间线下方的框表示每个事件及其间隔持续时间。一共有有四个事件，它们的（事件时间，间隔持续时间）对分别是（12:04，4 分钟）蓝色，（12:06，9 分钟）橙色，（12:09，5 分钟）黄色，以及（ 12:15，5 分钟）绿色。

时间线上方的框表示由这些事件构成的实际会话。您可以将每个事件视为一个单独的会话，并将具有交集的会话合并为一个。正如您看到的，会话的时间范围是会话中包含的所有事件的时间范围的“联合”。 请注意，会话的结束时间不再是会话中最新事件的时间 + 间隙持续时间。

新的 session_window 函数接收两个参数，事件时间列和间隙持续时间。

对于动态会话窗口，您可以在“session_window”函数中为“间隙持续时间”参数提供一个“表达式”。这个表达式应该解析为一个间隔，比如“5分钟”。由于“间隙持续时间”参数接收一个表达式，我们也可以使用 UDF。

例如，基于 eventType 列的动态间隙持续时间的会话窗口计数可以实现如下：

```
# Define the session window having dynamic gap duration based on eventType
session_window expr = session_window(events.timestamp, ₩
```

```
    when(events.eventType == "type1", "5 seconds") ₩
    .when(events.eventType == "type2", "20 seconds") ₩
    .otherwise("5 minutes"))

# Group the data by session window and userId, and compute the count of each group
windowedCountsDF = events ₩
    .withWatermark("timestamp", "10 minutes") ₩
    .groupBy(events.userID, session_window_expr) ₩
    .count()
```

原生支持会话窗口与 FlatMapGroupsWithState 比较

flatMapGroupsWithState
为实现会话窗口提供了更大的灵活性，但它需要用户编写一堆代码。例如，请参考 Apache Spark 上的 sessionization 示例，它通过 flatMapGroupsWithState 实现会话窗口。 请注意，Apache Spark 上的会话示例非常简化，仅适用于处理时间和附加模式。
通过原生支持会话窗口，处理事件时间和各种输出模式的总体复杂性被抽象了出来。

Spark 将原生支持会话窗口设置为覆盖一般用例的目标，因为它使 Spark 能够优化性能和状态存储。当我们的业务用例需要一个复杂的会话窗口时，我们可能仍然需要利用 flatMapGroupsWithState，例如，如果用例会话也应该在特定类型的事件上关闭，而不管是否处于不活动状态。

总结

我们已经介绍了流聚合中如何使用会话窗口，它也适用于批处理查询。通过学习如何使用新的 session_window
函数'，我们可以利用流数据聚合与时间窗口的知识，并能够处理会话窗口。这也使 SQL/PySpark 用户能够处理会话窗口，因为 flatMapGroupsWithState API 在 PySpark 中不可用并且不能表示为 SQL 语句。

当然，时间窗口的操作还有很多空间可以改进，在此之前还是需要使用 flatMapGroupsWithState API，社区计划在不久的将来研究自定义窗口操作。

本文翻译自：Native Support of Session Window in Spark Structured Streaming

本博客文章除特别声明，全部都是原创！
原创文章版权归过往记忆大数据（过往记忆）所有，未经许可不得转载。
本文链接: 【】（）