DTCC

2017第八届中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2017

Apache Beam 介绍

杨旭钧

Apache Beam 中文社区

Apache Beam概要介绍

什么是Apache Beam?

Apache Beam 编程模型

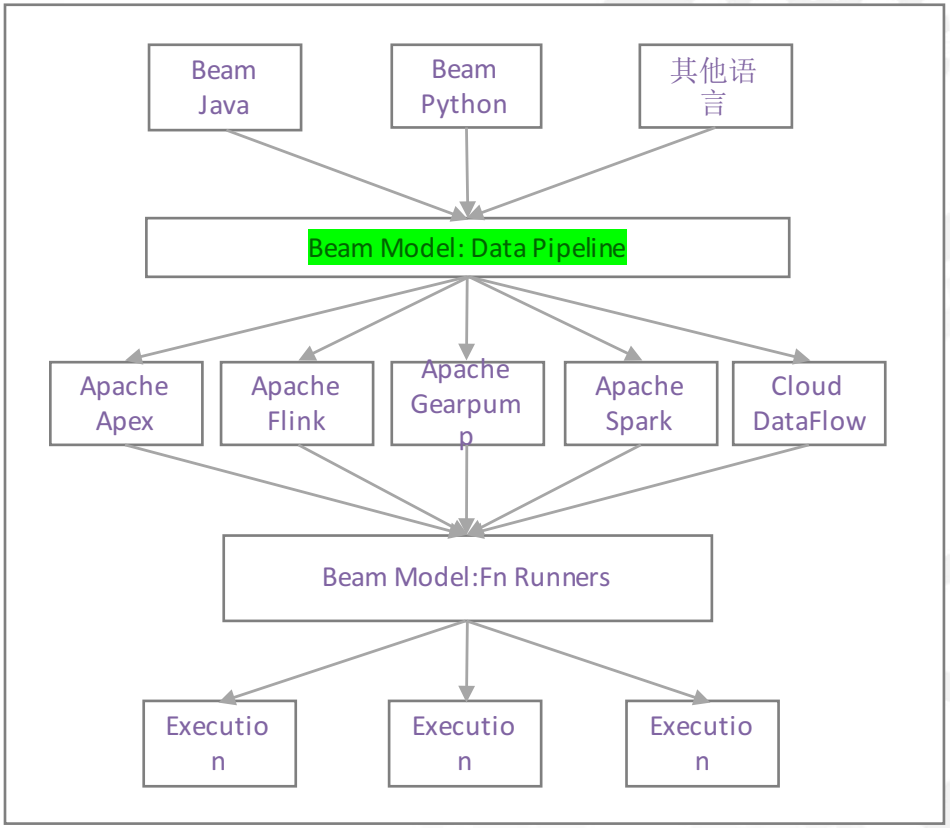
- What/Where/When/How

Beam Pipeline 的SDK

- Java、Python

内置一系列的Beam Runners

- Apache Apex
- Apache Flink
- Apache Gearpump
- Apache Kafka
- Apache Spark
- Google Cloud DataFlow



Beam支持的外部数据IO

- HDFS
- HBase
- JDBC
- MongoDB
- Elasticsearch
- Kafka
- Kinesis
- JMS
- MQTT
- Google - GCS, BigQuery, BigTable, Datastore

Apache Beam 模型介绍

- PCollection – 窗口内基于时间戳元素的并行集合。
- Sources & Readers – 产生时间戳元素的Pcollections 和水标。
- ParDo – 在Pcollection的元素上执行的并行处理函数。
- (Co)GroupByKey – shuffle & group {{K: V}} → {K: [V]}。
- Side inputs – PCollection 的全局视图，用于 broadcast / joins。

Apache Beam 模型介绍

- Window – 重新分配元素给窗口; 可能是数据依赖。
- Triggers – 基于窗口、水标、计数、延时进行流控。

Pcollection 介绍

Pipeline IO

读取输入数据—从文件中读取文本

```
PCollection<String> lines =
p.apply(TextIO.Read.from(“gs://some/inputData.txt"));
```

编写输出数据—保存数据到外部文件

```
PCollection<String> lines =
p.apply(TextIO.Read.from(“gs://some/inputData.txt"));
```

多对多输入和输出

Pcollection可从多个文件读取数据，也可以将数据写入到多个文件

PCollection功能特性介绍

- 不可变性：Pcollection一旦创建，无法添加、删除或更改里面的单个元素。在转换过程中，生成新的 Pipeline ，对原输入数据不做修改。
- 随机访问性：PCollection不支持随机访问单个元素。
- 大小与处理边界：Pcollection 没有大小限制，既可以是单机内存业可以是数据中心分布式数据集。另外，Pcollection可以是有界也可以是无界，例如，批量数据源可以代表有界数据源，流数据源可以代表无界数据源。
- 元素时间戳：每个元素都带有一个内置的时间戳，由Pcollection自动给他分配。

注意：在整个流式处理时，在任何一个时间点Pcollection 都是不可用的

获取内存中的数据创建PCollection

```
List<String> words = Arrays.asList(
      "apple",
      "apple",
      "apricot",
      "banana",
      "blackberry",
      "blackberry",
      "blackberry",
      "blueberry",
      "blueberry",
      "cherry");
PCollection<String> input = p.apply(Create.of(words));
```

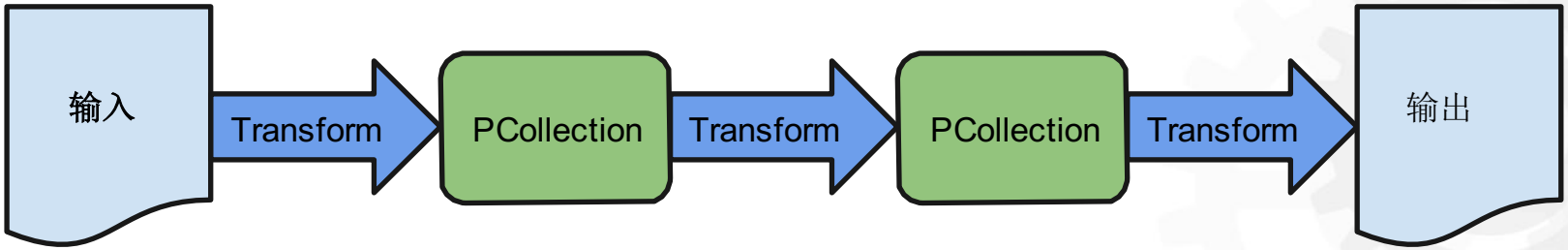
Pcollection进行简单的输入输出操作

```
PCollection<KV<String, List<CompletionCandidate>>> output =
    input.apply(new ComputeTopCompletions(2, recursive))
        .apply(Filter.by(
            new SerializableFunction<KV<String, List<CompletionCandidate>>, Boolean>()
{
              @Override
              public Boolean apply(KV<String, List<CompletionCandidate>> element) {
                return element.getKey().length() <= 2;
              }
            }));
```

Transformation 介绍

Transformation介绍

Transformation 用于对 Pipeline 中的数据进行处理操作。
将PCollection（或多个PCollection）作为输入，执行在该集合中每个元素上指定的操作，并生成新的输出Pcollection。



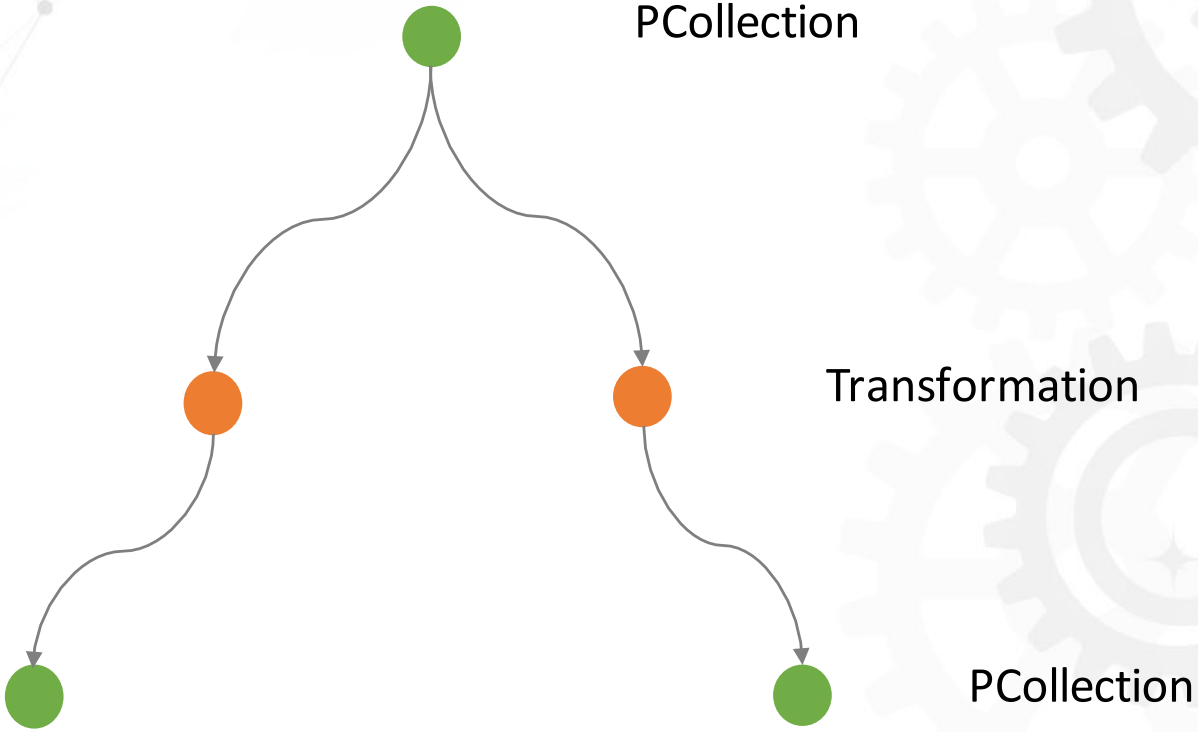
Transformation介绍

- 链式转换：上一个转换后的输出可以作为下一个转换的输入
- 嵌套转换：某个转换可以嵌套在另外的转换中

```
[Final Output PCollection] = [Initial Input PCollection].apply([First Transform])
.apply([Second Transform])
.apply([Third Transform])
```

Transformation介绍

一个Pcollectoin创建分支管道，拆分成多个PCollection进行计算



Pardo 介绍

- ParDo
- GroupByKey
- Combine
- Flatten 和 Partition

ParDo：Beam的通用并行处理方法。进行过滤(Filter)、格式化(Format)、抽取(Extract)、计算(Comput)等。

DoFn：是ParDo中具体执行计算的函数体。DoFn 处理PCollection中的元素，方法体中ProcessContext负责输入元素和输出元素的访问。

其他函数介绍

GroupByKey 介绍

- ParDo
- GroupByKey
- Combine
- Flatten 和 Partition

GroupByKey：是一个用于处理键值对集合的转换方法，类似于Map / Shuffle / Reduce-style算法的Shuffle阶段。最合适用于汇总包含具有相同键的多个对儿。

CoGroupByKey：CoGroupByKey连接两个或多个PCollection具有相同键类型的键/值，然后发出一组KV<K, CoGbkResult>对。

Combine 介绍

- ParDo
- GroupByKey
- Combine
- Flatten和Partition

Combine：是一个用于组合数据中元素或值集合的转换方法，类似于Map / Shuffle / Reduce-style算法的Shuffle阶段。最合适用于汇总包含具有相同键的多个对儿。Combine 组合了PCollection键/值对中的每个键的值。

扩展combine子类

```
public class AverageFn extends CombineFn<Integer, AverageFn.Accum, Double> {
  public static class Accum {
    int sum = 0;
    int count = 0;
  }

  @Override
  public Accum createAccumulator() { return new Accum(); }


  @Override
  public Accum mergeAccumulators(Iterable<Accum> accums) {
    Accum merged = createAccumulator();
    for (Accum accum : accums) {
      merged.sum += accum.sum;
      merged.count += accum.count;
    }
    return merged;
  }

}
```

扩展combine子类

```
public class AverageFn extends CombineFn<Integer, AverageFn.Accum, Double> {
  public static class Accum {
    int sum = 0;
    int count = 0;
  }

  @Override
  public Accum createAccumulator() { return new Accum(); }


  @Override
  public Accum mergeAccumulators(Iterable<Accum> accums) {
    Accum merged = createAccumulator();
    for (Accum accum : accums) {
      merged.sum += accum.sum;
      merged.count += accum.count;
    }
    return merged;
  }

}
```

Flatten和Partition 介绍

- ParDo
- GroupByKey
- Combine
- Flatten和Partition

Flatten：用于合并多个Pcollection对象到一个单Pcollection对象。

Partition：用于将一个Pcollection对象拆分成多个小的Pcollection。

Flatten 示例介绍

```
// Flatten takes a PCollectionList of PCollection objects of a given type.
// Returns a single PCollection that contains all of the elements in the PCollection objects in that list.
PCollection<String> pc1 = ...;
PCollection<String> pc2 = ...;
PCollection<String> pc3 = ...;
PCollectionList<String> collections = PCollectionList.of(pc1).and(pc2).and(pc3);

PCollection<String> merged = collections.apply(Flatten.<String>pCollections());
```

Partition 示例介绍

```
// Provide an int value with the desired number of result partitions, and a PartitionFn
that represents the partitioning function.
// In this example, we define the PartitionFn in-line.
// Returns a PCollectionList containing each of the resulting partitions as individual
PCollection objects.
PCollection<Student> students = ...;
// Split students up into 10 partitions, by percentile:
PCollectionList<Student> studentsByPercentile =
    students.apply(Partition.of(10, new PartitionFn<Student>() {
        public int partitionFor(Student student, int numPartitions) {
            return student.getPercentile()  // 0..99
                * numPartitions / 100;
        }}));

// You can extract each partition from the PCollectionList using the get method, as
follows:
PCollection<Student> fortiethPercentile = studentsByPercentile.get(4);
```

数据编码介绍

数据编码介绍

什么是数据编码？

将基本数据类型的数据对象序列化为字节字符串，用于后续的数据读取和写入操作

为什么要进行数据编码？

记录输入和输出数据对象的元数据信息和时间戳信息

数据编码类型映射

Java Type	Default Coder
Double	DoubleCoder
Instant	InstantCoder
Integer	VarIntCoder
Iterable	IterableCoder
KV	KvCoder
List	ListCoder
Map	MapCoder
Long	VarLongCoder
String	StringUtf8Coder
TableRow	TableRowJsonCoder
Void	VoidCoder
byte[ ]	ByteArrayCoder

Windows 函数介绍

窗口函数按照时间维度切分整个Pcollection，例如在某个窗口范围内，进行聚合操作。

窗口函数对于无限流数据处理是非常重要的，可将整个Pcollection细分为多个逻辑窗口。

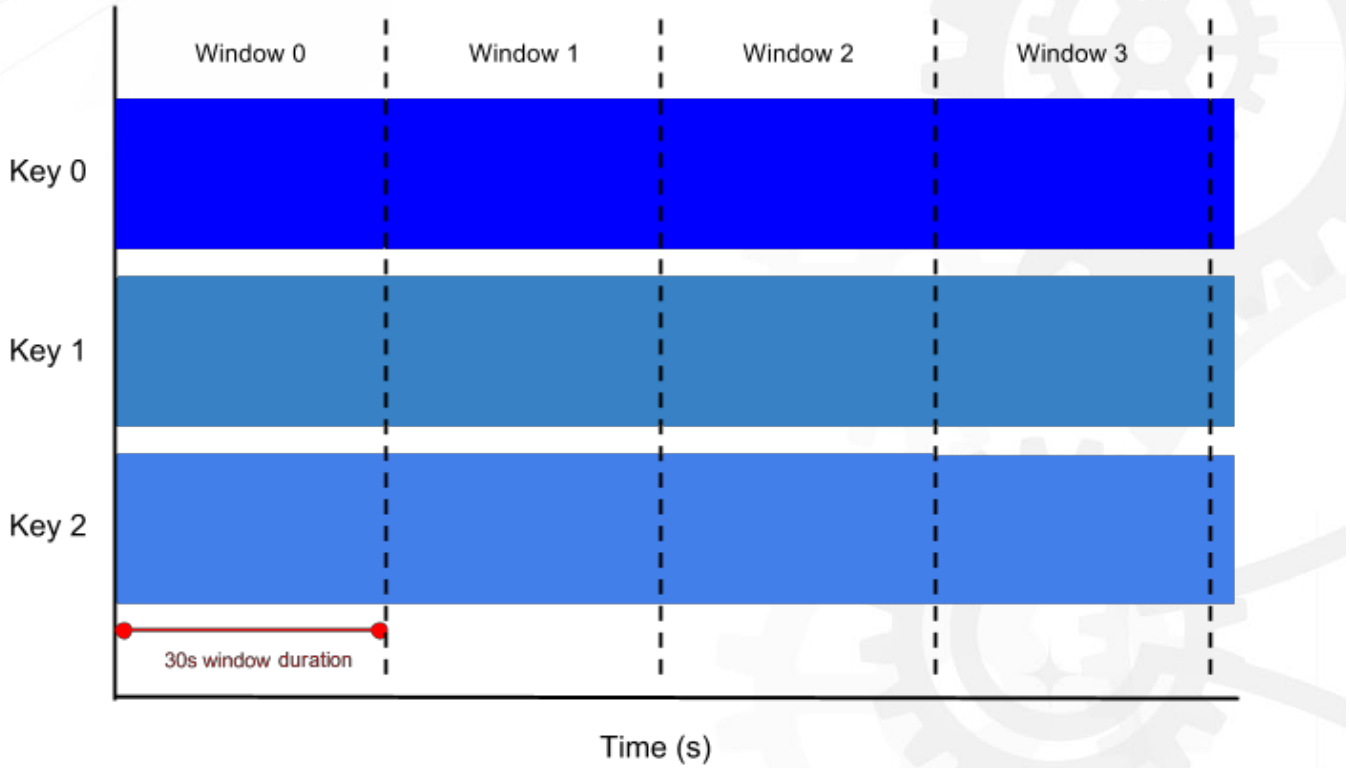
Windows介绍

Windows 函数分类

- 固定时间窗口
- 滑动时间窗口
- 每会话窗口
- 单一全局窗口
- 基于日历的窗口

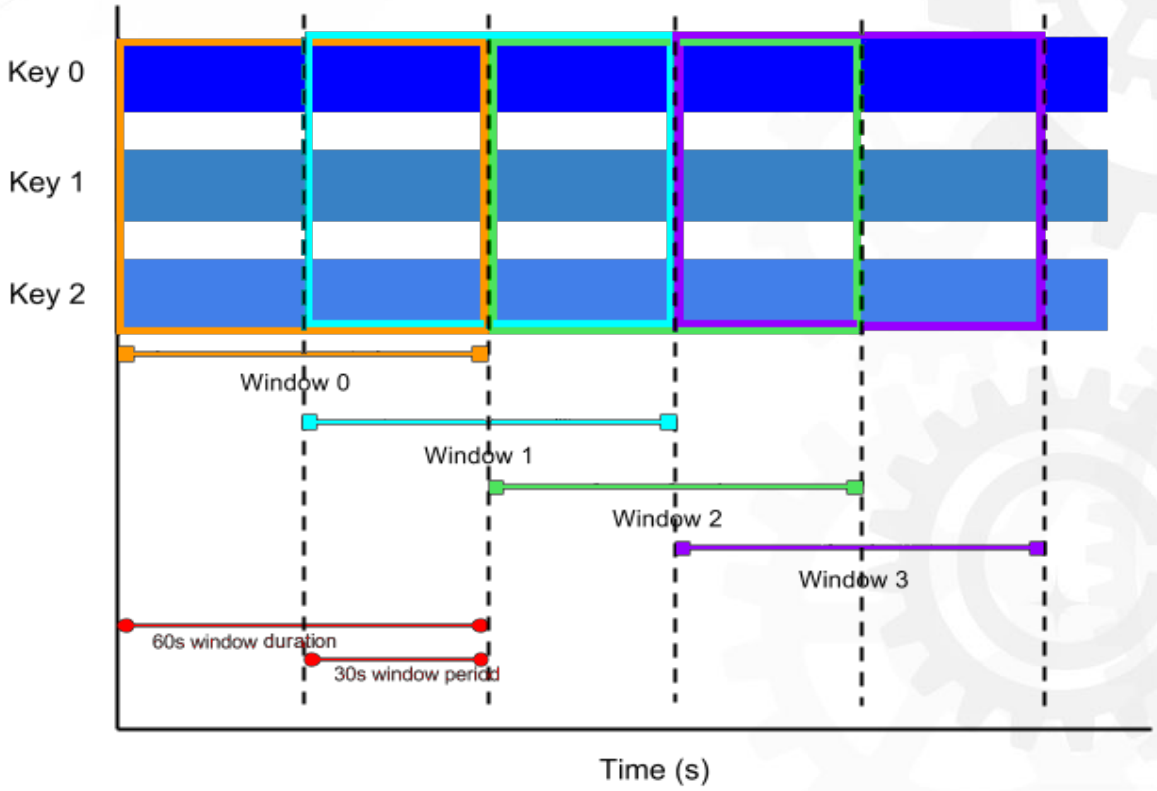
固定时间窗口

最简单的窗口形式是使用固定时间窗口：给定PCollection可能持续更新的时间戳，每个窗口可以捕获（例如）所有具有五分钟时间间隔的时间戳的元素。



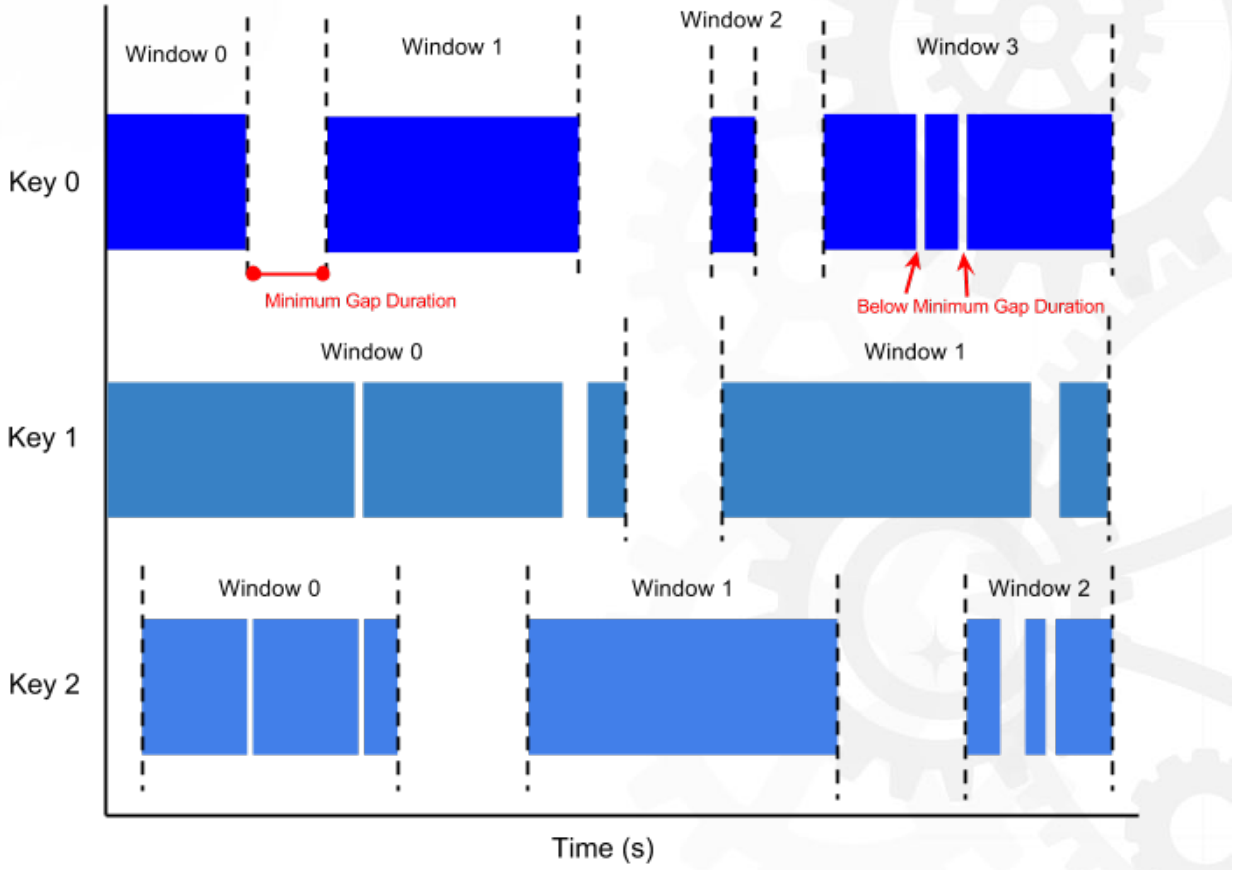
滑动时间窗口

滑动时间窗也代表在数据流中的时间间隔；然而，滑动时间窗口可以重叠。例如，每个窗口可能捕获五分钟的数据，但是每十秒钟会启动一个新窗口。



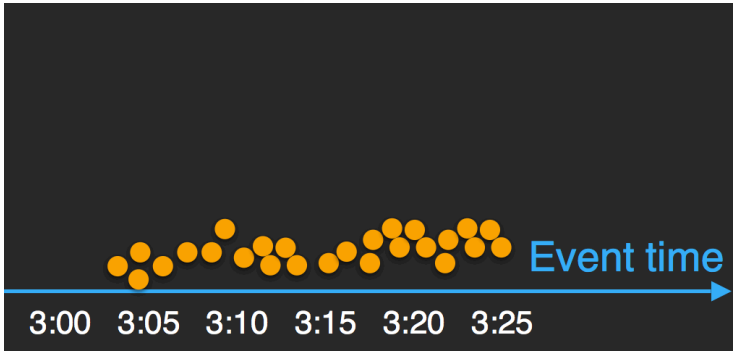
会话时间窗口

会话窗口函数定义了包含有另一元件的一定的间隙的持续时间内的元件的窗户。会话窗口适用于每个密钥，对于相对于时间不规则分布的数据是有用的。



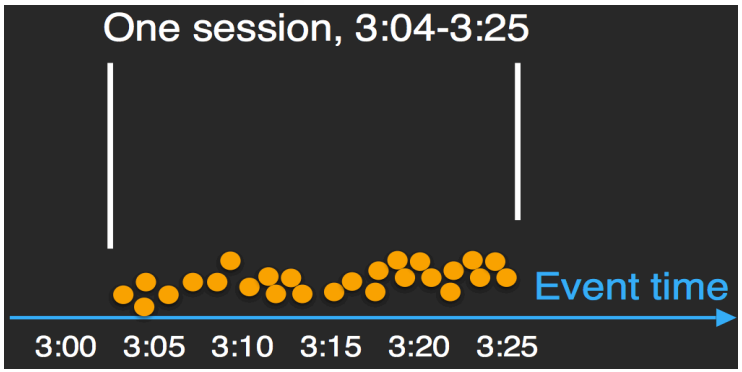
Windows和Triggers介绍

```
PCollection<KV<User,  Long>> userSessions  =
 clickstream.apply(Window.into(Sessions.withGapDuration(Minutes(3)))
 .triggering(
 AtWatermark()
 .withEarlyFirings(AtPeriod(Minutes(1)))))
```



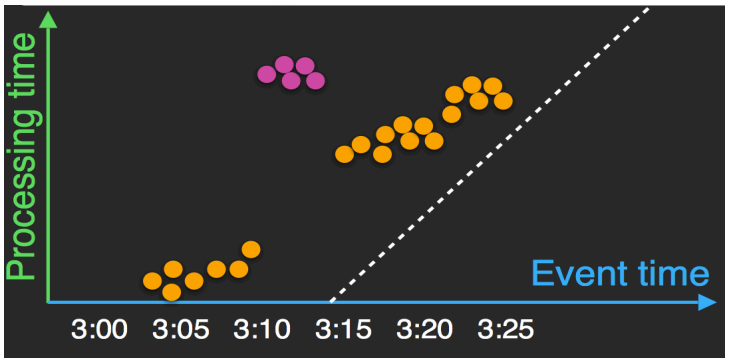
Windows和Triggers

```
PCollection<KV<User,  Long>> userSessions  =
 clickstream.apply(Window.into(Sessions.withGapDuration(Minutes(3)))
 .triggering(
 AtWatermark()
 .withEarlyFirings(AtPeriod(Minutes(1)))))
```



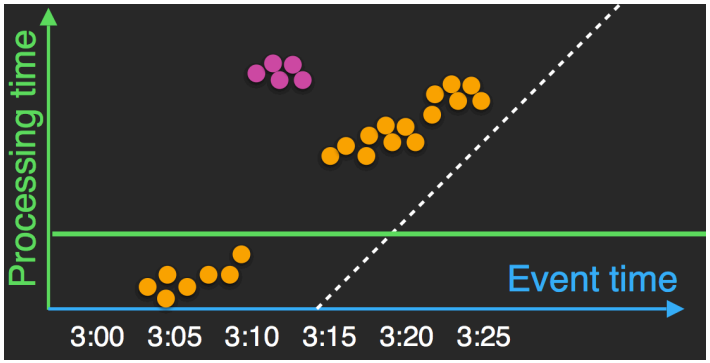
Windows和Triggers

```
PCollection<KV<User,  Long>> userSessions  =
 clickstream.apply(Window.into(Sessions.withGapDuration(Minutes(3)))
 .triggering(
 AtWatermark()
 .withEarlyFirings(AtPeriod(Minutes(1)))))
```



Windows和Triggers

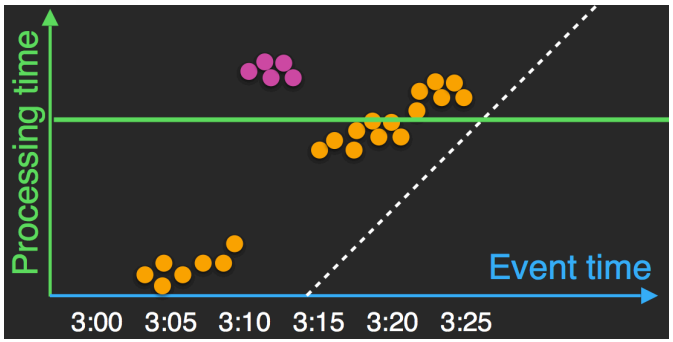
```
PCollection<KV<User,  Long>> userSessions  =
 clickstream.apply(Window.into(Sessions.withGapDuration(Minutes(3)))
 .triggering(
 AtWatermark()
 .withEarlyFirings(AtPeriod(Minutes(1)))))
```



(EARLY) 1 session, 3:04–3:10

Windows和Triggers

```
PCollection<KV<User,  Long>> userSessions  =
 clickstream.apply(Window.into(Sessions.withGapDuration(Minutes(3)))
 .triggering(
 AtWatermark()
 .withEarlyFirings(AtPeriod(Minutes(1)))))
```

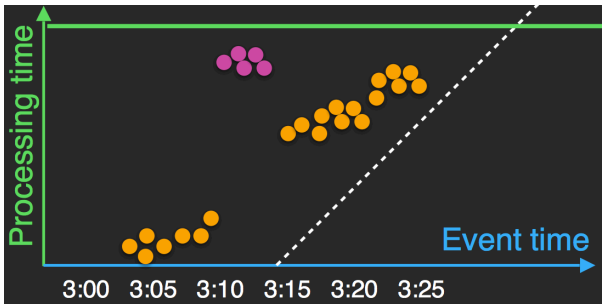


(EARLY) 2 sessions, 3:04–3:10 & 3:15–3:20

(EARLY) 1 session, 3:04–3:10

Windows和Triggers

```
PCollection<KV<User, Long>> userSessions =
 clickstream.apply(Window.into(Sessions.withGapDuration(Minutes(3)))
 .triggering(
 AtWatermark()
 .withEarlyFirings(AtPeriod(Minutes(1)))))
```



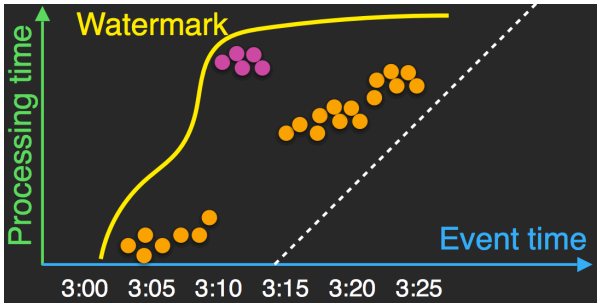
1 session, 3:04–3:25

(EARLY) 2 sessions, 3:04–3:10 & 3:15–3:20

(EARLY) 1 session, 3:04–3:10

Windows和Triggers

```
PCollection<KV<User, Long>> userSessions =
 clickstream.apply(Window.into(Sessions.withGapDuration(Minutes(3)))
 .triggering(
 AtWatermark()
 .withEarlyFirings(AtPeriod(Minutes(1)))))
```



1 session, 3:04–3:25

(EARLY) 2 sessions, 3:04–3:10 & 3:15–3:20

(EARLY) 1 session, 3:04–3:10

Beam生态系统介绍

流处理生态图

流处理生态组件比较

	Storm	Spark	Samza	Flink	Apex
Model	Native	Micro-batch	Native	Native	Native
API	compositional	declarative	compositional	declarative	compositional
FT	Record acks	RDD-based	Log-based	Checkpoints	Checkpoints
Guarantee	At-least-once	Exactly-once	At-least-once	Exactly-once	Exactly-once
State	Stateful operators	State as dstream	Stateful operators	Stateful operators	Stateful operators
Windowing	Time-based	Time-based	Time-based	Flexible	Time-based
Latency	Very-low	High	Low	Low	Very-low
Throughput	Medium	Very-High	High	Very-High	Very-High

Beam应用场景介绍

Beam 应用场景介绍

应用场景	日志/数据 采集转换加载	指标/KPI 抽取	实时数据分析	复杂流处理
数据类型	IT 基础设施 /数据库，应用日志，社交媒体, 金融/市场数据, Web 点击流, 传感器数据, 地理/位置数据			
数据库	各种数据库数据实时同步	大型机降负载	各种数据源实时汇总分析	
IT/业务	日志采集	业务指标实时报表	设备 / 传感器 运行智能化	
数字营销/市场	广告数据汇总分析	广告指标实时分析	用户参与度分析	买卖订单优化
金融服务	金融交易数据采集	金融数据指标评估	反欺诈监控,风险管理	订单数据审计
电子商务	在线用户参与数据汇总	消费者参与度实时分析	用户点击流分析	推荐引擎

Beam 开源社区介绍

- Apache Beam 官方网站

https://beam.apache.org

- Apache Beam 中文社区资料库

https://github.com/BloomSoftware/Beam

- The World Beyond Batch 101 & 102

https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101
https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-102

- 邮件列表!

user-subscribe@beam.apache.org dev-subscribe@beam.apache.org


THANKS
SequeMedia
盛拓传媒
ITPUB
ChinaUnix