Apache Flink可靠性基准测试方案

Technical Report

v0.3

郑莹莹，刘重瑞，许利杰

{zhengyingying14, liuchongrui15, xulijie}@otcaix.iscas.ac.cn

2017-05

**目录**

[1. 背景 3](#_Toc482736785)

[2. 目标 3](#_Toc482736786)

[3. 流处理系统特征 3](#_Toc482736787)

[3.1 无穷数据 3](#_Toc482736788)

[3.2 时间工具 4](#_Toc482736789)

[3.3 窗口机制 4](#_Toc482736790)

[3.4 状态与失效恢复 6](#_Toc482736791)

[3.5 Flink流处理流程 8](#_Toc482736792)

[4. 核心问题 9](#_Toc482736793)

[4.1 流式负载生成 9](#_Toc482736794)

[4.1.1 并发度（Parallelism） 9](#_Toc482736795)

[4.1.2 目标流速（TargetSpeed）及流速分布（SpeedDistribution） 10](#_Toc482736796)

[4.1.3 流式负载生成算法 11](#_Toc482736797)

[4.2 数据生成方式 11](#_Toc482736798)

[4.2.1 数据倾斜度（DataSkew） 12](#_Toc482736799)

[4.2.2 数据乱序程度（OutOfOrder） 13](#_Toc482736800)

[4.2.3 数据维度（DataDimension）及数据分布（DataDistribution） 13](#_Toc482736801)

[4.2.4 数据生成算法 13](#_Toc482736802)

[4.2.5 Source场景 14](#_Toc482736803)

[4.3 数据处理模式（window） 14](#_Toc482736804)

[4.3.1 影响参数分析 15](#_Toc482736805)

[4.3.2 问题场景分析 17](#_Toc482736806)

[5. 语义验证 19](#_Toc482736807)

[5.1 失效恢复验证 19](#_Toc482736808)

[5.2 计算结果准确性验证 20](#_Toc482736809)

[6. 测试流程 20](#_Toc482736810)

[6.1 具体应用选取 21](#_Toc482736811)

[6.2 参数组合测试 23](#_Toc482736812)

[6.3 测试报告生成 24](#_Toc482736813)

# 背景

当前，Flink等分布式处理框架被广泛应用于大数据的处理分析场景中。然而，应用在处理大数据时，经常出现内存溢出、I/O异常、任务无响应等运行时错误，这些错误会直接造成应用执行失败。同时，应用在处理流数据时经常出现数据丢失、数据重复计算、计算结果错误等情况。由于可靠性问题的多样性和复杂性，目前还没有一个通用的测试方法能够提前发现系统、应用和数据的潜在问题。本项目拟构造一个可靠性基准测试框架，通过构造典型应用，从数据源、数据处理及状态持久化三方面来对大数据系统流式应用进行可靠性分析，提供测试方案，从而提前暴露系统和应用存在的问题和缺陷。

# 目标

系统能否在有限时间内执行出正确结果是可靠性的表现。可靠性包括两点，一是计算的正确性，数据是否完整的被计算，未产生丢失，重复计算的情况，是否违背计算语义，失效情况下能否正常恢复；二是系统运行时的性能表现，是否会出现超时，内存溢出，I/O异常等运行时错误。

Flink可靠性基准测试框架的测试目标是验证以上两点。针对上述目标，可靠性基准测试框架的总体测试方案如下：

1. 构造典型应用：选取或构造有代表性的应用，尽可能的覆盖Flink中的DataSet/DataStream API, SQL, CEP等。
2. 流式负载及数据生成（Source）：通过高负载高流速的生成异常数据来验证流式应用的可靠性。
3. 计算逻辑分析：主要对流数据处理中的窗口应用进行计算分析，对其中容易出现错误的参数进行组合测试。
4. 语义验证方法：在相同数据集下，通过与批处理应用的计算结果进行对比来验证计算结果的正确性；通过对失效恢复中状态和检查点的持久化进行分析，验证Flink是否可以确保其设置的语义保障程度

# 流处理系统特征

相对于批处理系统，流处理系统有其不同的典型特征。

## 3.1 无穷数据

数据可分为有限数据和无穷数据。在批处理系统中，常常面对的是大量固定的数据集，然后在这些数据集上进行各种逻辑处理，得出结果。但是在流处理系统中，作为一种持续产生并且无穷的数据是主要处理对象，由于无穷数据的特征，许多在有限数据中的逻辑算子语义无法简单。

## 3.2 时间工具

当处理无穷无序数据时，时间工具是一切的基础。

如图1，一般的，事件的时间被分成两种。event time发生时间（事件实际发生时间），processing time处理时间（事件被计算处理时的时间）。虽然Flink还定义一种ingestion time事件摄入时间（事件进入Flink系统的时间），但其本质上的处理和event time类似。

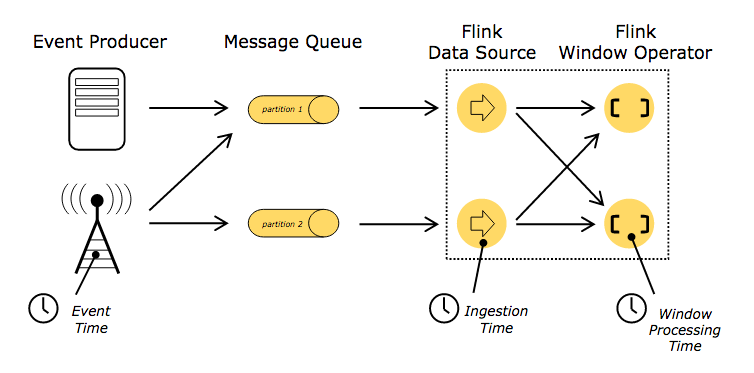


图1 Flink时间概念

在完美情况下，系统希望事件发生后立刻被处理，即event time和processing time永远相等，但是由于网络延迟，资源竞争等影响，processing time 将落后于event time，且落后的程度也不可预知。这种不确定性对于关系event time的逻辑处理是困难的。所以，应该设计相应时间工具来应对这种不确定性。

## 3.3 窗口机制

而在流系统中，处理的数据一般是无穷数据。对于一些聚合、连接等操作，由于数据是连续不断的，无法等到所有的数据结束才进行处理，但是可以定义这样一种操作，例如：统计过去的1分钟内有多少用户点击了我们的网页，每100个用户计算他们的平均停留时间……在这种情况下，必须定义一个范围，无论是基于时间还是基于事件个数，对这个范围内的数据进行合理计算，这个范围就叫做窗口。

所以，窗口就是用来对一个无穷的流设置一个有限的集合，在有限数据上进行操作的一种机制。下面以Flink流处理框架为例，简单介绍其窗口机制。

Flink流处理框架提供了丰富的窗口机制，首先将Flink的窗口进行分类，如表1所示。

表1 Flink窗口分类

|  |  |  |
| --- | --- | --- |
| Count |  | Tumbling Count Window |
| Sliding Count Window |
| Time | Event time | Tumbling Event Window |
| Sliding processing time window |
| Session Event time window |
| Processing time | Tumbling Event time Window |
| Sliding processing time window |
| Session processing time window |

根据驱动方式和时间产生方式，可以把window分成3类：基于事件个数的窗口，基于Event time的窗口以及基于Processing time的窗口。上述分类下的窗口又可以根据窗口移动方式分为以下3类：

1. 翻滚窗口(tumbling window)：按照相同的窗口大小将数据流切分成不重叠的窗口，这种窗口被成为翻滚窗口，翻滚窗口大小既可是时间长度也可以是事件个数。
2. 滑动窗口(sliding window)：按照指定的窗口大小，并以一定的步长在数据流中进行滑动，这种窗口称为滑动窗口，滑动窗口允许重叠存在。同样滑动窗口大小既可是时间长度也可以是事件个数
3. 会话窗口(session window)：将一段用户持续活动周期的一系列事件聚集到一个窗口中，由非活动的间隙（gap）分隔开，这样的窗口称为会话窗口。会话窗口只能是时间驱动的。

图2中，详细说明了每种窗口的对应语义。

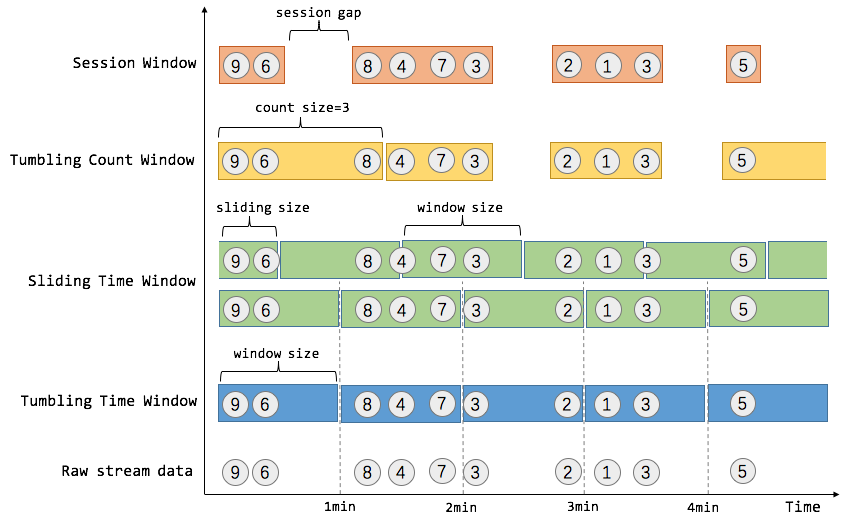


图2 Flink各窗口的语义

## 3.4 状态与失效恢复

Flink Job在运行过程中，每个Operator经历的阶段如图？所示。如果Operator以Failed状态结束，则需要系统恢复该Operator到失效之前的状态。Flink通过State和Checkpoint机制保证了失效恢复。

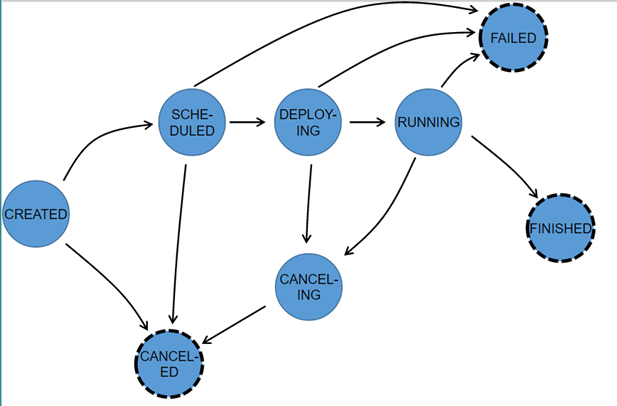


图3 Operator运行时状态

Flink提供两种基本的State类型： Keyed State和Operator State。

（1）Keyed State是与键值Key相关的，因此仅能用于KeyedStream上的函数和算子。Keyed State可以被看做已经被划分或分片后的Operator State。表2为Keyed State中使用的4种原语。

表 2 Keyed State类型

|  |  |
| --- | --- |
| **State** | **Meaning** |
| ValueState<T> | ValueState保存可以被更新和检索的值。这个值可以使用update(T)来设置值，使用T value()来检索值。 |
| ListState<T> | ListState保存一个列表元素。可以添加元素并且在所有当前存储的元素上检索Iterable。使用add(T)添加元素，使用Iterable<T> get()来获取检索的元素。 |
| ReducingState<T> | ReducingState保留一个单独的值，该值是添加到状态的所有值的聚合。与ListState接口类似，但是用于添加的元素的add(T)使用指定的ReduceFunction来实现聚合。 |
| FoldingState<T,ACC> | FoldingState保留一个单独的值，该值是添加到状态的所有值的聚合。与ReducingState相反的是，聚合类型可能与添加到state中的元素类型不同。接口与ListState类似，同样的add(T)使用指定的FoldFunction实现聚合。 |

（2）每个Operator State与一个并行的算子实例绑定。算子通过实现CheckpointedFunction接口或者ListCheckpointed<T extends Serializable>接口来保存状态信息。Flink Kafka就是使用的Operator State。KafkaConsumer的每个并行实例都包含一个Map作为他的Operator State，存放topic分区和偏移量。Operator State允许在并行度改变时重新分配算子实例中的状态。

这些状态信息的存储由StateBackend提供，共有三种方式，如表3所示。

表 3 StateBackend类型

|  |  |
| --- | --- |
| **StateBackend Type** | **Descriptor** |
| MemoryStateBackend | 状态值存储在Java堆中，并以哈希表（KeyMap数据结构）来存储这些K/V值。默认情况下使用该类型。 |
| FsStateBackend | in-flight数据存储在TaskManager内存中，检查点被触发后，将数据写入文件系统中。元数据存储在JobManager内存中。 |
| RocksDBStateBackend | In-flight数据存储在RocksDB中，检查点被触发后，将数据写入文件系统中。元数据存储在JobManager内存中。 |

上述状态信息可通过检查点（快照）的方式每隔一段时间进行一次持久化操作。当出现任务失败时，系统可以从这些检查点保存的状态值中恢复任务到失效前一刻的状态，以保障系统处理数据时的可靠性。系统中使用的检查点的属性如表4所示。

表 4 Checkpoint属性

|  |  |
| --- | --- |
| **Checkpoint Attribute** | **Description** |
| checkpointingMode | 语义保障的类型，包括“恰好一次”、“至少一次”  （默认情况下是“恰好一次”）。 |
| checkpointInterval | 快照被触发的时间间隔  （默认为-1，即不触发快照）。 |
| checkpointTimeout | 快照存活的最大时间  （默认为10分钟，即一次快照必须在十分钟内完成）。 |
| minPauseBetweenCheckpoints | 两次快照之间的最小暂停间隔  （默认为0） |
| maxConcurrentCheckpoints | 同时可能正在进行的最大检查点尝试次数 |

系统在进行检查点操作时，会在数据流中注入Barriers，通过Barriers将数据流中的记录隔离成一系列的记录集合，相邻Barriers之间的记录状态信息会被存入一个检查点中。每个Barrier都带有一个ID标识，并会随着数据流向下游Operator。当一个中间的Operator从它的所有输入流中接收到一个来自快照n的屏障，它会发送一个针对快照n的Barrier到所有它的输出流。

## 3.5 Flink流处理流程

我们以Flink流处理框架为例，简单介绍流处理的整个流程。如图4所示，事件从传感器、基站等地方生成之后，会以数据流的形式流入Flink系统，通过Source、Transformation以及Sink Operator的计算，最终流入数据库、文件系统等存储介质中。Flink Stream DataFlow中的Source/Transformation/Sink Operator通过调用相应的Function来实现用户逻辑的执行，同时通过设置不同的parallelism，可以实现多个subTask的并行执行。



图4 数据流图

综上，parallelism在系统运行中会影响系统任务的分配。例如，图5为并行度为1的数据流逻辑视图，这种情况下，每个Operator只能有一个线程来执行其Function中定义的计算逻辑。如果设置Source并行度为1，FlatMap、Window以及Sink的并行度为2，则优化后的数据流并行化视图如图6所示，此时有5个并行的线程执行Job。



图 5数据流逻辑视图



图 6数据流并行化视图

每个Operator的Parallelism都可以由用户来指定，其值确定的流程如图7所示。在Job运行时，会给每个Operator分配parallelism个线程执行其计算逻辑， parallelism的取值范围为 [1,NumTaskManagers\*NumSlotsPerTaskManager]。

接下来将从数据流经DataFlow中的各个阶段介绍Flink可靠性基准测试的测试方案。



图 7 Parallelism确定流程

# 核心问题

## 4.1 流式负载生成

为了提供高并发、流速可变的负载，Flink可靠性基准测试针对流式负载生成定义了以下几种相关属性，如表5所示。

表 5 流式负载生成属性

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| Parallelism | 任务并发度，即Source的subTask数目，决定负载生成的并发度。 |
| TargetSpeed | 目标流速，即每秒钟期望产生的数据量，决定负载生成的流速。 |
| SpeedDistribution | TargetSpeed的分布形式，即随机生成满足一定概率分布的TargetSpeed值，决定负载生成中的流速变化情况。 |

### 4.1.1 并发度（Parallelism）

Parallelism在系统运行中会影响系统任务的分配，通过设置不同的parallelism，可以实现多个subTask的并行执行，且subTask的数目对应线程数目，从而可以实现多并发的负载生成。如图8所示，当parallelism取值为2时，表示有两个Source并发的生成数据，每个Source是一个独立的线程，但执行的数据生成逻辑是相同的，即通过并发度的设置实现了独立同分布的负载生成。Parallelism的取值范围为[1, NumTaskManagers\*NumSlotsPerTaskManager]，其中NumTaskManager为Flink集群中TaskManager的数目，NumSlotsPerTaskManager为每个TaskManager中Slot的数目。如果设置parallelism超出了该取值范围，则会在运行Job时抛出异常：没有足够的Slot运行Job。



图 8 Parallelism决定负载并发度

### 4.1.2 目标流速（TargetSpeed）及流速分布（SpeedDistribution）

TargetSpeed被定义为每秒钟期望产生的数据量。如图9所示，在时间为t时， TargetSpeed 取值为100，即t时刻产生的数据量为100。通过设定TargetSpeed值可以控制每个Source生成数据的流速（每秒钟产生的数据量）。由于一条数据发送成功后才会继续发送下一条数据，且受分布式环境下节点通信造成的网络延迟等因素的影响，因此TargetSpeed只是一个期望的流速，不一定能够达到。



图 9目标流速

SpeedDistribution决定了负载生成中TargetSpeed值的分布方式。如图10所示，通过使用泊松分布、指数分布、均匀分布等概率分布方式，随机生成TargetSpeed值，从而模拟流式场景下的流速变化的情况。



图 10 流速分布

### 4.1.3 流式负载生成算法

综合上述属性，流式负载的生成算法的步骤如下：

1. 设置parallelism=N，提供N个负载并发生成数据的场景。
2. 设置targetSpeed取值范围[minSpeed, maxSpeed]，提供流速变化的范围。如果提供流速不变的场景，则需设置minSpeed=maxSpeed。
3. 设置speedDistribution=D，确定targetSpeed值的分布形式。
4. 使用D分布，在[minSpeed, maxSpeed]范围内生成一个随机数作为latestSpeed值（当前的TargetSpeed值）。
5. N个负载独立同分布的生成latestSpeed个数据。

如图11所示，基于上述算法，可靠性基准测试框架会在t时刻，由N个负载发生器分别生成latestSpeed条数据，且latestSpeed是在一定范围内满足一定分布的随机值。



图 11 流式负载生成

## 4.2 数据生成方式

针对流式负载生成的数据，根据异常数据的特征（数据倾斜、数据维度高、数据分布异常、数据稀疏等），总结数据的相关属性如表6所示。

表 6 数据生成属性

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| DataSkew | 数据倾斜度 |
| OutOfOrder | 数据乱序因子 |
| DataDimension | 数据维度，即一条数据的维度大小 |
| DataDistribution | 数据分布形式 |

### 4.2.1 数据倾斜度（DataSkew）

DataSkew用于描述Key的倾斜程度，取值为0~1的浮点数，且取值越接近1，表示数据倾斜程度越高。DataSkew通过控制Zipf分布的参数取值，来实现满足Zipf分布的不同倾斜程度的数据生成。图12为不同倾斜度取值下的数据分布情况。



图12 单参数Zipf分布

Flink Stream中的倾斜数据主要针对经过keyBy()操作对数据进行分流的情况。如果分流后的数据有严重的数据倾斜，会给某些任务节点带来计算压力。如图13，Window接收的数据为Map输出的结果通过哈希分区后传来的数据，如Window(1)接收Key值为3,5,7的数据，Window(2)接收Key值为1,2,4,6的数据。由于Key为3和7的数据较其他Key值对应的数据量过大，并且这些数据全部由Window(1)来处理，此时会导致该subTask处理的负载过大，其Window Buffer中存储的数据量会急剧增大，造成系统资源压力和任务计算压力。



图13 倾斜数据

### 4.2.2 数据乱序程度（OutOfOrder）

Flink对使用EventTime的应用进行数据处理时，需要根据时间戳来确定数据归属的窗口。数据流正常情况下是按照时间戳的顺序流入到Operator中，但是由于网络时延、背压等因素，可能会出现乱序的情况。OutOfOrder决定数据的乱序程度，即一个数据晚于正常情况下到达Operator的时间。如图14所示，数据流按照一定的顺序从Source到下游Operator。正常的事件处理顺序应该是1,2,3,4,5，然而由于数据传输过程中的网络时延，1,2,3,5到达Window，而数据4却迟到了T时刻，此时就是一个乱序的数据流。乱序的时间T由OutOfOrder来决定。OutOfOrder取值为0时，表示该数据流没有乱序发生（数据生成时不考虑乱序，但不排除数据实际传输中存在乱序）。



图 14 乱序程度

### 4.2.3 数据维度（DataDimension）及数据分布（DataDistribution）

DataDimension用于描述数据的维度，决定了一条数据的数据格式，如一个N维数据可表示为（Key,Value1,Value2,….ValueN-1,Timestamp），Timestamp是该数据生成时的时间戳。且该数据中的每一维的数据类型可以是基本数据类型、对象、数组等形式。当数据维度过高，且每个维度的数据过大时，负载产生的每一条数据都会占用较高的内存空间。如果数据需要经过序列化/反序列化操作，则会产生较高的处理时延。

DataDistribution用于决定Value的分布形式，Zipf分布、高斯分布、指数分布、均匀分布以及混合分布等概率分布方式。

### 4.2.4 数据生成算法

综合上述属性，数据生成算法的步骤如下：

1. 设置DataSkew=*s*，同时确定Key的取值范围[*minKey*, *maxKey*]；
2. 设置DataDimension=*N*，确定数据格式（*Key,Value1,Value2,….ValueN-1*）；
3. 设置DataDistribution，确定Value值的分布形式；
4. 设置OutOfOrder=t，确定乱序的程度，如果t=0，则不产生乱序数据；
5. 在[*minKey*, *maxKey*]范围内生成一个满足倾斜程度为*s*的Zipf分布的随机值作为当前的Key值；
6. 针对Key以外的其他value值，采用满足DataDistribution的分布形式随机生成数据。
7. 根据OutOfOrder来确定时间戳Timestamp，即某一条数据的时间戳生成为currentTimestamp+t。

如图15所示，基于上述算法，流入下游Operator的每条数据都有一定的数据格式，并由数据维度N来决定。数据格式为（Key,Value1,Value2,….ValueN-1，Timestamp）的数据中，Key由DataSkew来决定倾斜程度，Value根据DataDistribution在一定范围内生成满足一定分布的随机值。



图 15数据生成方法

### 4.2.5 Source场景

数据源通过组合流式负载属性和数据属性实现流式负载数据生成。根据这些属性值可以提供多种流式数据场景：

1. 当Parallelism取值较大时，可以提供高并发场景；
2. 当TargetSpeed取值较大时，可以提供高流速场景；
3. 当SpeedDistribution为泊松分布时，可以提供流速变化的负载生成；
4. 当DataSkew为0.9时，可以提供倾斜度较高的异常负载数据。
5. 当DataDimension较大时，可以提供数据维度较高的异常负载数据。
6. 当OutOfOrder大于0时，可以提供乱序数据。

## 4.3 数据处理模式（window）

当接收到上游Operator流出数据后，下一步执行计算逻辑。而在流处理系统中，由于无穷数据的特征，许多操作算子是基于window来进行计算的。这节主要分析基于window来进行的数据处理模式。

### 4.3.1 影响参数分析

为了进行测试和分析，首先需要知道有哪些参数会影响计算逻辑和结果。根据3.3节中提到的Window分类，分析其影响计算时间复杂度或空间复杂度的可变窗口参数，并总结为表7。

表7 窗口参数

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| window size | 窗口大小 |
| slide | 窗口滑动的步长 |
| gap | 窗口的非活动时间间隙 |

**窗口大小**：是所有窗口都需要设置的参数，窗口大小的单位可以是时间长度，也可以是事件的个数，具体取决于窗口语义。

**滑动步长**：滑动窗口独有的参数，根据滑动窗口的滑动步长和窗口大小，可以计算出每个事件分配的窗口个数。

**非活动时间间隙**：会话窗口独有的参数，当两个事件的时间距离大于非活动间隙，就将其分隔为两个会话窗口。

除了窗口类型和其参数这一影响因素外，流处理还有其他的参数需要考虑，如表8所示。

表8 其他参数

|  |  |
| --- | --- |
| **参数** | **描述** |
| parallelism | 并行度 |
| maxOutOfOrderness | 最大乱序时间 |
| allowedLateness | 允许迟到时间 |
| Function | 窗口触发时的运算逻辑。 |

**并行度**：并行度影响系统任务的分配，Flink一般在KeydStream上进行window处理，在之前我们一般会先进行keyby()函数的分区，用哈希函数将key值相同的事件分到同一个subtask上，如图16所示；如果不采用keyby分区，就只能采用windowAll，所有的事件将强制推到一个subtask中进行，即并行度为1，如图17所示。



图16 keyby与window函数



图17 windowAll函数

**最大乱序时间与允许迟到时间**：Flink中，如果我们关心事件的实际产生时间，那么我们就要使用Event time，但是由于网络延迟等环境因素的影响，具有先后关系产生数据，在实际算子处理时并不一定仍然保持相对顺序，即事件的到达可能是乱序的。Flink采用watermark机制来处理乱序，最大乱序时间即为watermark的一个参数，它允许一些事件在但最大乱序时间内到达，仍然被纳入窗口中，如图18所示。允许迟到时间allowedLateness也是一个处理消息乱序的机制，它允许窗口触发计算后，仍然保留一段时间，若此时有属于此窗口的事件到达，重新触发计算。

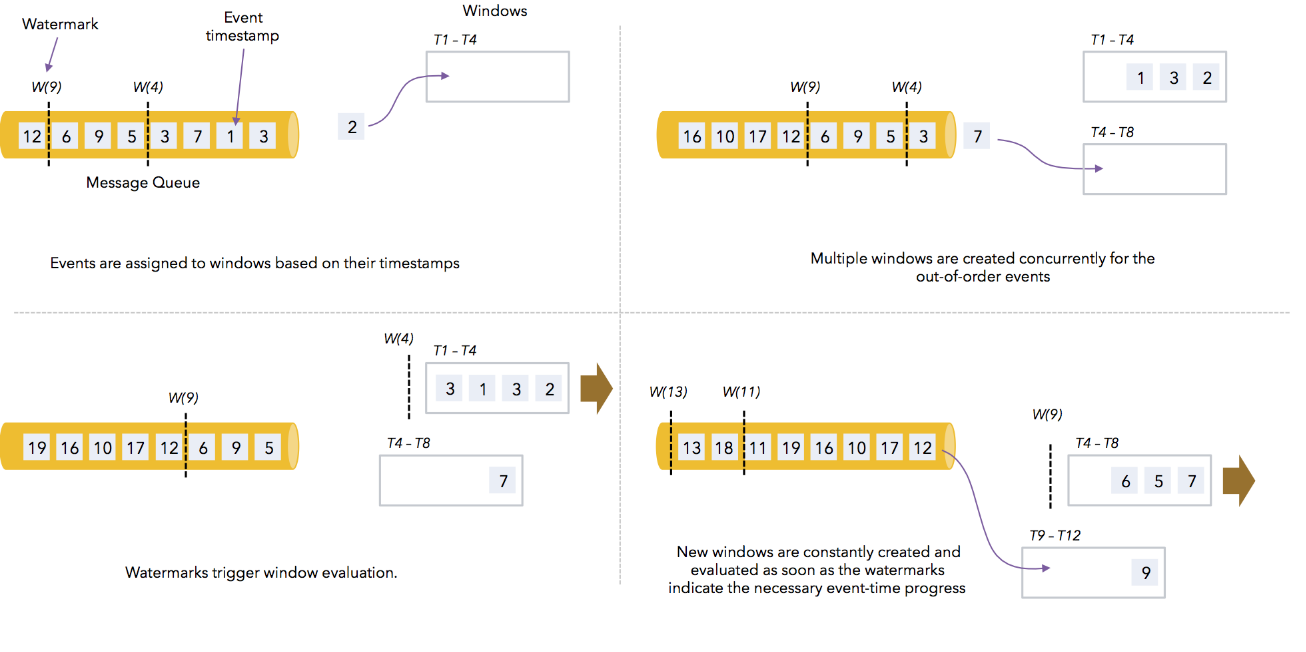


图18 Flink watermark机制

**窗口处理函数**：Flink中提供了一些简单的计算逻辑，如求和，求最值等，用户也可以定义更加复杂的处理逻辑。

### 4.3.2 问题场景分析

根据4.3.1节分析的窗口参数和其他参数，构造参数设置的逻辑图，如图19所示。

图19 设置影响参数的逻辑图

可知各种参数之间并不是互相独立的，但是仍然可以通过组合窗口参数与其他参数的值，提供多种流式数据窗口处理的场景，来验证Flink基于窗口处理逻辑的可靠性。

在不同的场景下，结合Flink流处理机制和资源，分析可能产生的问题场景：

1. **数据倾斜**：采用keyby函数分区时，相同key的事件将被推倒同一个subtask中，若数据存在倾斜，会导致某条流上的数据远大于其他流，那么该流的处理时间将会成为系统的性能瓶颈。如图20，我们使用keyby将源数据进行分区，key为1,3的事件将分在一个subtask中，而key为2,4的事件被分在另一个subtask中，此时前者的待处理数据远多于其他流。



图20倾斜数据对keyby分区的影响

1. **窗口大小与滑动值比例过高**：如图21所示，采用时间驱动的sliding window时，若window大小远大于滑动值，那么在每个元素分配窗口时会产生很多的窗口，在每个窗口中都会记录下这个事件的值，可能会有内存或性能问题。



图21窗口大小与滑动值比例过大

1. **会话窗口时间间隙不合理**：如图22，采用Session window时，若会话间隔时间过长，则会导致所有的事件都聚集到一个session window中，若数据一直存在内存中，而窗口又未达到触发条件，则可能产生内存问题。



图22 会话间隙参数影响

1. **事件乱序程度过于严重**：Flink 使用watermark和allowedLateness机制来处理乱序和迟到的事件，但是由于事件的到达时间不可预测，若最大乱序时间和允许迟到时间设置过小，则有些迟到事件被丢弃，导致结果错误；若最大乱序时间和允许迟到事件设置过大，则窗口会一直等待，占据内存，可能引发内存或性能问题。
2. **窗口计算复杂**：当窗口处理逻辑过于复杂时，窗口触发运算时需要消耗大量资源，可能导致此时间段流入算子的事件流无法及时处理，引起性能问题。

# 语义验证

语义验证包括计算结果的准确性验证和失效恢复验证。

## 5.1 失效恢复验证

Flink提供两种语义保障程度：“exactly-once”和“at-least-once”。在“exactly-once”模式下，所有的数据都只被计算一次；在“at-least-once”模式下，某些数据可能会被重复计算。

Flink首先需要满足以下几点要求：

1. Source需要由Kafka提供，并需要保存数据偏移量，确保数据源可重放；
2. 带有状态的Operator需要实现CheckpointedFunction或ListCheckpointed接口，并提供状态保存的计算逻辑；
3. 状态信息需要保存在文件系统中，确保状态的可靠性。

如果一个Operator有多个输入流，为了满足“exactly-once”的语义保障，Flink需要将这多个输入流进行Barrier对齐操作，即Operator在所有的输入流的中都接受到Barrier n时，才会处理这些数据，否则会一直在input buffer中缓存。在这种情况下，如果该Operator中Function的计算复杂度过高或其等待所有ID为n的Barriers到来的时间过长，则会在缓存中积累过多的数据，对系统造成一定的压力。

失效恢复验证的步骤如图23所示：（1）第一组测试中，人为宕掉Job中的Operator，触发Flink的失效恢复机制，并得到最终的计算结果1；（2）第二组测试中，使用相同的数据集进行测试，并得到计算结果；（3）对比两组测试的计算结果，验证失效恢复后是否能够保证结果的准确性。



图 23 失效恢复验证

## 5.2 计算结果准确性验证

为了验证流式应用的计算结果正确性，首先需要将测试使用的数据集保存到文件系统中，确保测试可重放。如图24所示，流式应用从文件中读取与批处理应用相同的数据集，并使用上述负载生成方式将这些数据由Source发送给下游的Operator，最后通过比较流式应用和批处理应用得到的计算结果，进而验证系统计算结果的准确性。



图 24 计算结果准确性验证

# 测试流程

当测试环境搭建完毕后，开始进行实际测试，测试的流程如图25所示。



图25 测试流程图

流式数据生成与结果验证在第4章均已介绍，这里不再赘述。只考虑选取具体应用，参数组合测试和报告生成3个部分。

## 6.1 具体应用选取

Flink可靠性基准测试框架开始测试时，需要选取具体待测的应用。所以，测试框架中需要事先提供覆盖Flink API的应用。表1中提供的应用选自Flink自身提供的Example、现有的性能测试基准和Stream基准中提供的应用，以及通过Flink提供的API自行构建的应用。表9中Window中的窗口类型简写为如下形式：Tumbling Window（TW），Sliding Window（SL），Session Window（SE）。

表 9 典型应用

|  |  |  |  |
| --- | --- | --- | --- |
| **应用类型** | **典型应用** | **应用描述** | **覆盖API** |
| Basic | WordCount | 统计文件中单词出现的次数 | DataSet API flatMap,keyBy,sum |
| PiEstimation | 使用蒙特卡罗方法估计Pi的值 | DataSet API中map,reduce |
| CollectionExecutionExample | 集合的join操作 | DataSet API中join,where,equalTo |
| **StreamWordCount** | 统计文件中单词出现的次数 | DataStream API中flatMap, keyBy, sum |
| ChainingSpeed | 计算供应链速度 | DataStream API中map,flatMap |
| StreamRepartition | rebalance操作 | DataSteam API中rebalance |
| **StreamUnion** | 将两个流速不同的数据流合并 | DataStream API中map, filter, union |
| Window  (TW,SL,SE) | **WindowWordCount**  **(TW,SL,SE)** | 使用Window机制，统计文件中单词出现的次数 | DataStream API中flatMap, keyBy, GlobalWindow, sum |
| **WindowJoin**  **(TW,SL,SE)** | 两个使用Window的数据流的join操作 | DataStream API中Join, TumblingEventTimeWindows, apply |
| TopSpeedWindowing  (TW,SL,SE) | 统计窗口的最大速度 | DataStream API中assignTimestampsAndWatermarks,keyBy,window,evictor,trigger,maxBy |
| **SessionWindowing**  **(SW)** | 使用Session窗口处理数据流 | DataStream API中keyBy, EventTimeSessionWindows, sum。 |
| **GroupedProcessingTimeWindow**  **(TW,SL,SE)** | 分组处理时间窗口的例子 | DataStream API中keyBy, SlidingProcessingTimeWindows, reduce。 |
| **CountAverage**  **(TW,SL,SE)** | 统计平均值，并使用Managed keyed state保存状态信息 | DataStream API中keyBy, flatMap, ValueState, window, apply |
| CEP | CEPExample | 在KeyedStream中检测模式start,middle(name = ”error”) - > end(name = ”critical”) |  |
| NFA | 非确定性有限自动机实现 |  |
| SQL | SQL Base | 基础子句，操作符和函数.基础子句包括select,groupby,join等；操作符包括like,exists,in等；函数包括count,max,decode等 |  |
| Benchmark | 主流的数据库评测基准测试集所提供的SQL 语句，计划主要采用3个，Big Data Benchmark(4)，TPC-H(23)和TPC-DS(99) |  |
| Typical SQL Query | 通过理论分析和实证分析，总结出的可能在特殊数据下表现异常的SQL语句，选取这些语句单独作为典型应用 |  |

上述应用的主要来源如下：

[1] flink-perf: <https://github.com/project-flink/flink-perf>

[2] HiBench: <https://github.com/intel-hadoop/HiBench>

[3] Yahoo streaming benchmark: <https://github.com/yahoo/streaming-benchmarks>

[4] dataArtisans: <https://github.com/dataArtisans/performance>

[5]AaltoStreamBench:

<https://github.com/wangyangjun/StreamBench/tree/master/StreamBench>

## 6.2 参数组合测试

流处理系统运行应用的配置参数包括系统参数和应用参数。系统参数指的是系统运行应用时可能会影响系统数据分配或任务分配等的参数；应用参数指的是应用或算法本身运行时需要的参数。参数（如，窗口大小、最大允许乱序时间、并行度、划分函数等）配置错误，同样也会引起错误的产生。因此，还需要考虑组合系统参数以及应用参数进行测试。

当我们对这些系统参数和应用参数进行组合测试时，会存在一个主要的问题：参数组合空间过大。假设一个应用有*n*个配置参数，其中第*i*个参数有*mi*个可选值，那么其参数组合空间为这*n*个数的乘积，其计算复杂度为*O*(*m1\*m2\**···*\*mn*)：

考虑到当参数个数过多时，会出现参数组合测试空间爆炸问题，为此，我们做了以下两个假设：

1. *n*个参数互相独立
2. 第*i*个参数的*mi*个可选值与性能/资源利用率正相关或负相关

在满足这两个假设的基础上，我们可以得出以下结论：参数在取得临界值时应用性能最差或资源消耗最高（可能会触发运行时错误）。在此假设基础上，我们提出了一个削减参数组合测试空间的贪心算法，如下：

1. 给定每个参数的具体取值范围
2. 选择每个参数的某一临界值进行组合，进行测试，并记录资源使用情况；
3. 改变一个参数的取值为另一临界值，进行测试，并记录资源使用情况；
4. 比较两次参数组合下的资源使用，选择性能较差的那个参数作为固定配置。
5. 返回第2步重复执行。直到出现异常或组合测试结束。





图 26 参数组合空间削减过程

图26给出了一个具有三个配置参数的例子。首先，选择三个参数的最小边界取值组合进行测试，即2-1-2；然后，将第一个参数的取值改为最大边界值进行测试，即100-1-2；比较应用在参数组合2-1-2和组合100-1-2下进行测试的资源使用情况，如果组合2-1-2的资源使用比100-1-2低，则固定第一个参数取值为100。接下来改变第二个参数的临界值为10，即参数组合为100-10-2，依次进行测试比较，最后得到d)中给出的最差参数组合100-10-1。

如果参数不满足以上两个假设，我们将使用二分查找的方式选择每个参数中的最差的配置，这种情况下的计算复杂度为*O*(*logm1*\**logm2*\*···\**logmn*)。

## 6.3 测试报告生成

在选定待测的应用，并给定测试数据以及参数配置后，可靠性测试框架执行参数组合测试，并在执行结束之后，提供测试报告。

测试报告的主要内容包括：

1. 配置信息：如应用名称、测试数据、系统参数以及系统参数等。
2. 运行指标：应用运行过程中使用的最大内存、CPU等资源占用情况。
3. 错误信息：应用运行过程中是否出现异常。

通过上述报告内容，可靠性测试框架给出测试结果以及简单的测试分析。测试人员可以从测试报告中分析得出：

1. 如果出现了异常，可以得到该应用出现错误的数据集以及具体的参数配置信息。
2. 如果没有发现异常，可以得到能够使该应用具有最差资源使用情况下的数据集和参数配置信息。

测试人员可以根据测试报告找到出现问题的配置参数组合，以及具有异常特征的数据集。那么在系统上线后的应用执行过程中，可以通过加强输入数据格式检测、避免极端配置参数组合等方式，尽可能的避免错误的出现。