细细品味 Storm

——Storm 简介及安装

精

华

集

锦

csAxp

http://www.xiapistudio.com/

2015 年 6 月 11 日

目录

1、Storm 简介.....2
1.1 主要特点.....2
1.2 基本概念.....4
1.3 基础架构.....5
1.4 工作原理.....7
1.5 亮点功能.....12
1.6 图形案例.....15
1.7 发展趋势.....18
2、Storm 安装.....18
2.1 版本选择.....18
2.2 安装 Zookeeper 19
2.3 安装 Storm.....20
2.4 集成 Kafka.....21
2.5 启动 Storm.....22
2.6 验证 Storm.....22
3、参考文献.....24
4、打赏小编.....25

1、Storm 简介

Storm 是由专业数据分析公司 BackType 开发的一个**分布式实时**数据处理软件，可以简单、高效、可靠地处理大量的数据流。Twitter 在 2011 年 7 月收购该公司，并于 2011 年 9 月底正式将 Storm 项目开源。Storm 被托管在 GitHub 上，目前最新版本是 0.9.0.1。软件核心部分使用 Clojure 开发，外围部分使用 Java 开发。Clojure（发音同 **closure**）是 Lisp 语言的一种现代方言。类似于 Lisp，Clojure 支持一种功能性编程风格，但 Clojure 还引入了一些特性来简化多线程编程（一种对创建 Storm 很有用的特性）。

Twitter 列举了 Storm 的三大类应用：

- 信息流处理（**Stream Processing**）
  Storm 可用来**实时处理新数据**和**更新数据库**，兼具**容错性**和**可扩展性**。
- 连续计算（**Continuous Computation**）
  Storm 可进行连续查询并把结果即时反馈给客户端。比如把 Twitter 上的热门话题发送到浏览器中。
- 分布式远程程序调用（**Distributed RPC**）
  Storm 可用来并行处理密集查询。Storm 的拓扑结构是一个等待调用信息分布函数，当它收到一条信息后，会对查询进行计算，并返回查询结果。举个例子 Distributed RPC 可以做并行搜索或者处理大集合的数据。

Storm 的主工程师 Nathan Marz 表示：Storm 可以方便地在一个计算机集群中编写与可扩展的实时计算，**Storm 之于实时处理，就好比 Hadoop 之于批处理**。Storm 保证每个消息都会得到处理，而且它很快——在一个小集群中，每秒可以处理数以百万计的消息。更棒的是你可以使用任意编程语言来做开发。

当然 Storm 也存在一些**缺点**：开源版的 Storm 有个最大的缺点，就是只支持单 Nimbus 节点，一旦 Nimbus 节点挂掉就只能重启，存在单点失效的问题；Clojure 是一个在 JVM 上运行的动态函数式编程语言，优势在于流程计算，Storm 的核心部分由 Clojure 编写，虽然性能上提高不少但同时也提高了维护成本。

1.1 主要特点

Storm 拥有低延迟、高性能、分布式、可扩展、容错等特性，可以保证消息不丢失，消息处理严格有序。Storm 的主要特点如下所示：

- 简单的编程模型。类似于 MapReduce 降低了并行批处理复杂性，Storm 降低了进行实时处理的复杂性。
- 可以使用各种编程语言。你可以在 Storm 之上使用各种编程语言。默认支持 Clojure、Java、Ruby 和 Python。要增加对其他语言的支持，只需实现一个简单的 Storm 通信协议即可。
- 容错性。Storm 会管理工作进程和节点的故障。
- 水平扩展。计算是在多个线程、进程和服务器之间并行进行的。
- 可靠的消息处理。Storm 保证每个消息至少能得到一次完整处理。任务失败时，它会负责从消息源重试消息。

- 快速。系统的设计保证了消息能得到快速的处理，使用 ØMQ 作为其底层消息队列。
- 本地模式。Storm 有一个“本地模式”，可以在处理过程中完全模拟 Storm 集群。这让你可以快速进行开发和单元测试。

Storm 集群由一个主节点和多个工作节点组成。主节点运行了一个名为“**Nimbus**”的守护进程，用于分配代码、布置任务及故障检测。每个工作节点都运行了一个名为“**Supervisor**”的守护进程，用于监听工作，开始并终止工作进程。Nimbus 和 Supervisor 都能**快速失败**，而且是**无状态**的，这样一来它们就变得十分健壮，两者的协调工作是由 **Apache ZooKeeper** 来完成的。

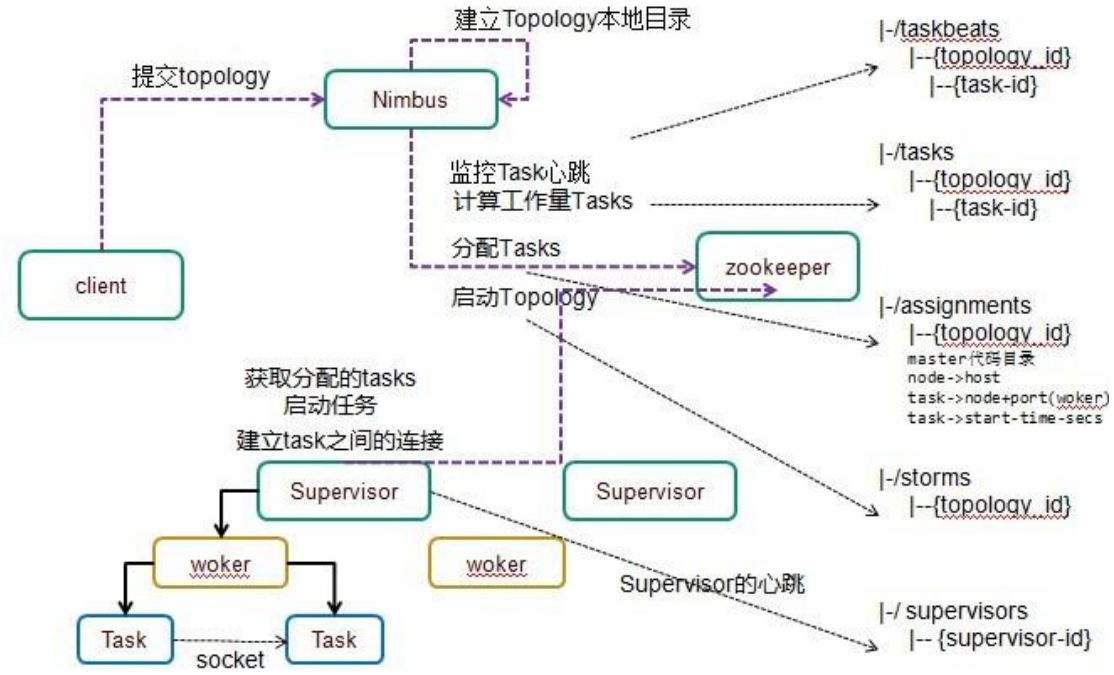


图 1.1 Storm 工作流程

第一步：客户端提交拓扑到 Nimbus。第二步：Nimbus 针对该拓扑建立本地的目录根据 topology 的配置计算 task，分配 task，在 zookeeper 上建立 assignments 节点存储 task 和 supervisor 机器节点中 woker 的对应关系。第三步：在 zookeeper 上创建 taskbeats 节点来监控 task 的心跳，启动 topology。第四步：Supervisor 去 zookeeper 上获取分配的 tasks，启动多个 woker 进行，每个 woker 生成 task，一个 task 一个线程；根据 topology 信息初始化建立 task 之间的连接；Task 和 Task 之间是通过 ZeroMQ 管理的；后整个拓扑运行起来。

Storm 的术语包括 Stream、Spout、Bolt、Task、Worker、Stream Grouping 和 Topology。**Stream** 是被处理的数据。**Sprout** 是数据源。**Bolt** 处理数据。**Task** 是运行于 Spout 或 Bolt 中的线程。**Worker** 是运行这些线程的进程。**Stream Grouping** 规定了 Bolt 接收什么东西作为输入数据。数据可以随机分配（术语为 Shuffle），或者根据字段值分配（术语为 Fields），或者 广播（术语为 All），或者总是发给一个 Task（术语为 Global），也可以不关心该数据（术语为 None），或者由自定义逻辑来决定（术语为 Direct）。Topology 是由 Stream Grouping 连接起来的 Spout 和 Bolt 节点网络。

可以和 Storm 相提并论的系统有 Esper、Streambase、HStreaming 和 Yahoo S4。其中和 Storm 最接近的就是 S4。两者**最大的区别**在于 Storm 会保证消息得到处理。这些系统中有的拥有**内建数据**存储层，这是 Storm 所**没有**的，如果需要持久化，可以使用一个类似于 Cassandra 或 Riak 这样的外部数据库。

1.2 基本概念

首先我们通过一个 Storm 和 Hadoop 的对比来了解 Storm 中的基本概念。

表 1.1 Storm 和 Hadoop 角色对比

	Hadoop	Storm
系统角色	JobTracker	Nimbus
	TaskTracker	Supervisor
	Child	Worker
应用名称	Job	Topology
组件接口	Mapper/Reducer	Spout/Bolt

接下来我们再来具体看一下这些概念。

- Nimbus：负责资源分配和任务调度。
- Supervisor：负责接受 nimbus 分配的任务，启动和停止属于自己管理的 worker 进程。
- Worker：运行具体处理组件逻辑的进程。
- Task：worker 中每一个 spout/bolt 的线程称为一个 task。在 Storm0.8 之后，task 不再与物理线程对应，同一个 spout/bolt 的 task 可能会共享一个物理线程，该线程称为 executor。

下面这个图描述了以上几个角色之间的关系。

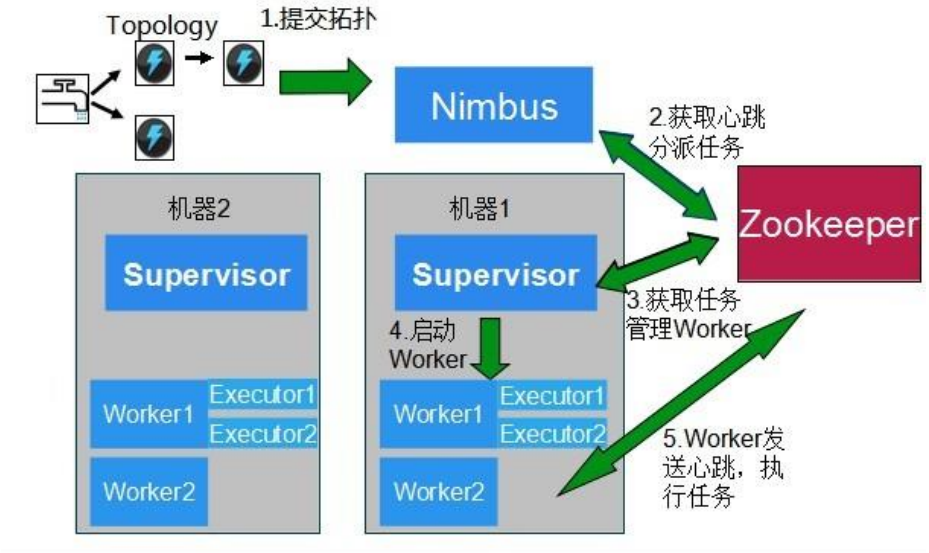


图 1.2 Storm 角色间关系

- Topology：Storm 中运行的一个实时应用程序，因为各个组件间的消息流动形成逻辑上的一个拓扑结构。
- Spout：在一个 topology 中产生源数据流的组件。通常情况下 spout 会从外部数据源中读取数据，然后转换为 topology 内部的源数据。Spout 是一个主动的角色，其接口中有个 nextTuple()函数，Storm 框架会不停地调用此函数，用户只要在其中生成源数据即可。
- Bolt：在一个 topology 中接受数据然后执行处理的组件。Bolt 可以执行过滤、函数操作、合并、写数据库等任何操作。Bolt 是一个被动的角色，其接口中有个 execute(Tuple input)函数,在接受到消息后会调用此函数，用户可以在其中执行自己想要的操作。
- Tuple：一次消息传递的基本单元。本来应该是一个 key-value 的 map，但是由于各个组件间传递的 tuple 的字段名称已经事先定义好，所以 tuple 中只要按序填入各个 value 就行了，所以就是一个 value list.
- Stream：源源不断传递的 tuple 就组成了 stream。

Hadoop 是实现了 MapReduce 的思想，将数据切片计算来处理大量的离线数据。Hadoop 处理的数据必须是已经存放在 hdfs 上或者类似 hbase 的数据库中，所以 Hadoop 实现的时候是通过**移动计算**到这些存放数据的机器上来提高效率的；而 Storm 不同，Storm 是一个流计算框架，处理的数据是实时消息队列中的，所以需要我们写好一个 topology 逻辑放在那，接收进来的数据来处理，所以是通过**移动数据**平均分配到机器资源来获得高效率。

Hadoop 的优点是处理数据量大（瓶颈是硬盘和 namenode，网络等），分析灵活，可以通过实现 dsl，mdx 等拼接 Hadoop 命令或者直接使用 hive，pig 等来灵活分析数据。适应对大量维度进行组合分析。其缺点就是慢：每次执行前要分发 jar 包，Hadoop 每次 map 数据超出阈值后会将数据写入本地文件系统，然后在 reduce 的时候再读进来。

Storm 的优点是全内存计算，因为内存寻址速度是硬盘的百万倍以上，所以 Storm 的速度相比较 Hadoop 非常快（瓶颈是内存，cpu）。其缺点就是不够灵活：必须要先写好 topology 结构来等数据进来分析。

Storm 关注的是数据**多次处理一次写入**，而 Hadoop 关注的是数据**一次写入，多次查询**使用。Storm 系统运行起来后是**持续不断**的，而 Hadoop 往往只是在**业务需要**时调用数据。

1.3 基础架构

Storm 集群类似于一个 Hadoop 集群。 然而你在 Hadoop 的运行“MapReduce job”，在 Storm 上你运行 “topologies”。 “job”和“topologies”本身有很大的不同，其中一个关键的区别是，MapReduce 的工作最终完成，而 topologies 处理消息永远保持（或直到你杀了它）。Strom 集群有主要有两类节点：主节点和工作节点。 主节点上运行一个叫做“Nimbus”的守护进程，也就是类似 Hadoop 的“JobTracker”。 Nimbus 负责在集群分发的代码，将任务分配给其他机器，和故障监测。

每个工作节点运行一个叫做”Supervisor”的守护进程。Supervisor 监听分配给它的机器，

根据 Nimbus 的委派在必要时启动和关闭工作进程。每个工作进程执行 topology 的一个子集。一个运行中的 topology 由很多运行在很多机器上的工作进程组成。

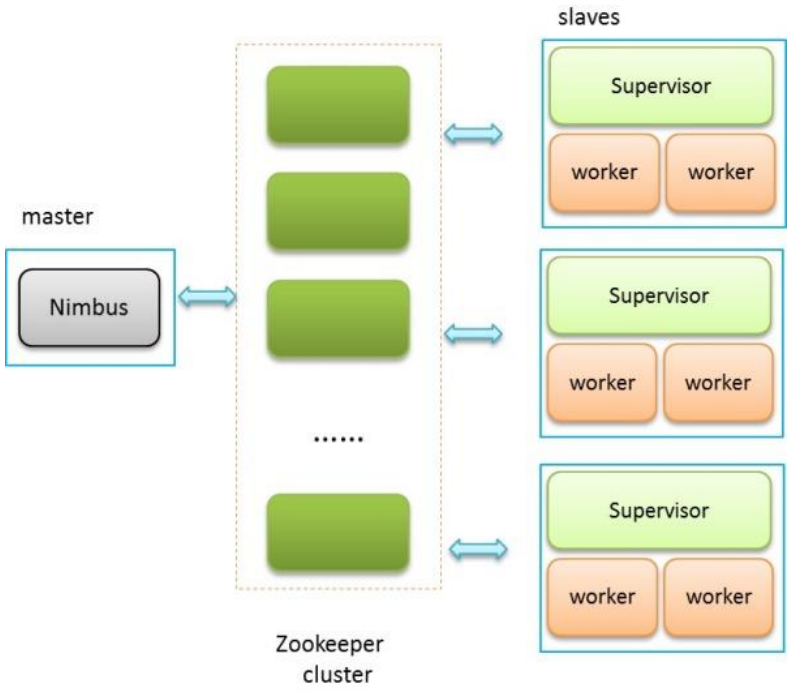


图 1.3 Storm 架构

Nimbus 和 Supervisors 之间所有的协调工作是通过一个 Zookeeper 集群。 此外，Nimbus 的守护进程和 Supervisors 守护进程是无法连接和无状态的；所有的状态维持在 Zookeeper 中或保存在本地磁盘上。这意味着你可以 kill -9 Nimbus 或 Supervisors 进程，所以他们不需要做备份。这种设计导致 Storm 集群具有令人难以置信的稳定性。

Storm 实现了一种数据流模型，其中数据持续地流经一个转换实体网络。一个数据流的抽象称为一个**流（stream）**，这是一个无限的元组序列。**元组（tuple）**就像一种使用一些附加的序列化代码来表示标准数据类型（比如整数、浮点和字节数组）或用户定义类型的结构。每个流由一个唯一 ID 定义，这个 ID 可用于构建数据源和接收器（sink）的拓扑结构。流起源于**喷嘴（spout）**，Spout 将数据从外部来源流入 Storm 拓扑结构中。

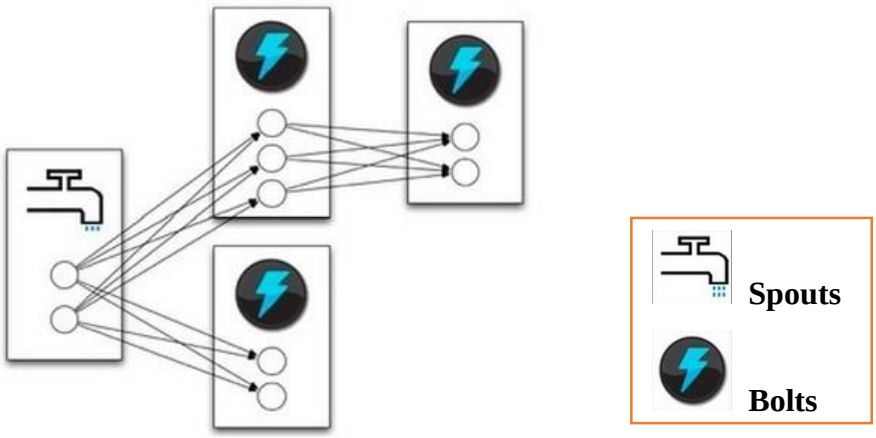


图 1.4 Storm 的拓扑结构

接收器（或提供转换的实体）称为**螺栓（bolt）**。螺栓实现了一个流上的单一转换和一个 Storm 拓扑结构中的所有处理。Bolt 既可实现 MapReduce 之类的传统功能，也可实现更复杂的操作（单步功能），比如过滤、聚合或与数据库等外部实体通信。典型的 Storm 拓扑结构会实现多个转换，因此需要多个具有独立元组流的 Bolt。Bolt 和 Spout 都实现为 Linux 系统中的一个或多个任务。

但是，Storm 架构中一个最有趣的特性是有保障的消息处理。Storm 可保证一个 Spout 发射出的每个元组都会处理；如果它在超时时间内没有处理，Storm 会从该 Spout 重新发射该元组。此功能需要一些聪明的技巧来在拓扑结构中跟踪元素，也是 Storm 的重要的附加价值之一。

除了支持可靠的消息传送外，Storm 还使用 ØMQ（ZeroMQ）最大化消息传送性能（删除中间排队，实现消息在任务间的直接传送）。ØMQ 合并了拥塞检测并调整了它的通信，以优化可用的带宽。

Storm 0.9.0.1 版本的第一亮点是引入了 Netty Transport。Storm 网络传输机制实现可插拔形式，当前包含**两种方式**：原来的 ØMQ 传输，以及新的 Netty 实现；在早期版本中（0.9.x 之前的版本），Storm 只支持 ØMQ 传输，由于 ØMQ 是一个本地库（native library），对平台的依赖性较高，要完全正确安装还是有一定挑战性。而且版本之间的差异也比较大；Netty Transport 提供了纯 JAVA 的替代方案，消除了 Storm 的本地库依赖，且比 ØMQ 的网络传输性能快一倍以上。

1.4 工作原理

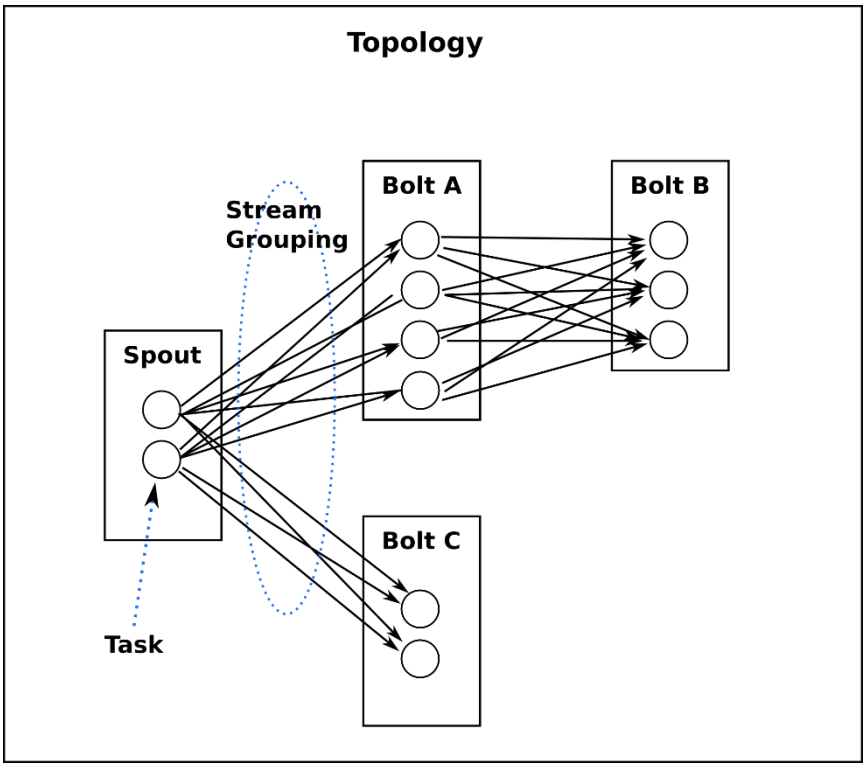


图 1.5 Storm 里面各个对象的示意图

【计算拓补：Topologies】

一个实时计算应用程序的逻辑在 Storm 里面被封装到 topology 对象里面，我把它叫做计算拓补。Storm 里面的 topology 相当于 Hadoop 里面的一个 MapReduce Job，它们的关键区别是：一个 MapReduce Job 最终总是会结束的，然而一个 Storm 的 topoloy 会一直运行，除非你显式的杀死它。一个 Topology 是 Spouts 和 Bolts 组成的图状结构，而链接 Spouts 和 Bolts 的则是 Stream groupings。

【消息流：Streams】

消息流是 Storm 里面的最关键的抽象。一个消息流是一个没有边界的 tuple 序列， 而这些 tuples 会被以一种分布式的方式并行地创建和处理。对消息流的定义主要是对消息流里面的 tuple 的定义，我们会给 tuple 里的每个字段一个名字。 并且不同 tuple 的对应字段的类型必须一样。 也就是说： 两个 tuple 的第一个字段的类型必须一样， 第二个字段的类型必须一样，但是第一个字段和第二个字段可以有不同的类型。在默认的情况下，tuple 的字段类型可以是：integer、long、short、byte、string、double、float、boolean 和 byte array。你还可以自定义类型——只要你实现对应的序列化器。

每个消息流在定义的时候会被分配给一个 id，因为单向消息流是那么的普遍，OutputFieldsDeclarer 定义了一些方法让你可以定义一个 stream 而不用指定这个 id。在这种情况下这个 stream 会有个默认的 id: 1.

【消息源：Spouts】

消息源 Spouts 是 Storm 里面一个 topology 里面的消息生产者。一般来说消息源会从一个外部源读取数据并且向 topology 里面发出消息：tuple。 消息源 Spouts 可以是可靠的也可以是不可靠的。一个可靠的消息源可以重新发射一个 tuple 如果这个 tuple 没有被 Storm 成功的处理， 但是一个不可靠的消息源 Spouts 一旦发出一个 tuple 就把它彻底忘了——也就不可能再发了。

消息源可以发射多条消息流 stream。使用 OutFieldsDeclarer.declareStream 来定义多个 stream，然后使用 SpoutOutputCollector 来发射指定的 sream。

Spout 类里面最重要的方法是 nextTuple 要么发射一个新的 tuple 到 topology 里面或者简单的返回如果已经没有新的 tuple 了。要注意的是 nextTuple 方法不能 block Spout 的实现，因为 Storm 在同一个线程上面调用所有消息源 Spout 的方法。

另外两个比较重要的 Spout 方法是 ack 和 fail。Storm 在检测到一个 tuple 被整个 topology 成功处理的时候调用 ack, 否则调用 fail。Storm 只对可靠的 spout 调用 ack 和 fail。

【消息处理者：Bolts】

所有的消息处理逻辑被封装在 bolts 里面。 Bolts 可以做很多事情：过滤、聚合、查询数据库等等等等。

Bolts 可以简单的做消息流的传递。复杂的消息流处理往往需要很多步骤， 从而也就需要经过很多 Bolts。比如算出一堆图片里面被转发最多的图片就至少需要两步：第一步算出每个图片的转发数量。第二步找出转发最多的前 10 个图片。(如果要把这个过程做得更具有扩展性那么可能需要更多的步骤)。

Bolts 可以发射多条消息流，使用 OutputFieldsDeclarer.declareStream 定义 stream，使用 OutputCollector.emit 来选择要发射的 stream。

Bolts 的主要方法是 execute，它以一个 tuple 作为输入，Bolts 使用 OutputCollector 来发射 tuple，Bolts 必须要为它处理的每一个 tuple 调用 OutputCollector 的 ack 方法，以通知 Storm 这个 tuple 被处理完成了。从而我们通知这个 tuple 的发射者 Spouts。 一般的流程是： Bolts 处理一个输入 tuple，发射 0 个或者多个 tuple，然后调用 ack 通知 Storm 自己已经处理过这个 tuple 了。Storm 提供了一个 IBasicBolt 会自动调用 ack。

【Stream groupings：消息分发策略】

定义一个 Topology 的其中一步是定义每个 bolt 接受什么样的流作为输入。stream grouping 就是用来定义一个 stream 应该如果分配给 Bolts 上面的多个 Tasks。

Storm 里面有 6 种类型的 stream grouping：

- Shuffle Grouping：**随机分组**，随机派发 stream 里面的 tuple，保证每个 bolt 接收到的 tuple 数目相同。
- Fields Grouping：**按字段分组**，比如按 userid 来分组，具有同样 userid 的 tuple 会被分到相同的 Bolts，而不同的 userid 则会被分配到不同的 Bolts。
- All Grouping：**广播发送**，对于每一个 tuple，所有的 Bolts 都会收到。
- Global Grouping：**全局分组**，这个 tuple 被分配到 Storm 中的一个 bolt 的其中一个 task。再具体一点就是分配给 id 值最低的那个 task。
- Non Grouping：**不分组**，表示 stream 不关心到底谁会收到它的 tuple。目前这种分组和 Shuffle grouping 是一样的效果，有一点不同的是 Storm 会把这个 bolt 放到这个 bolt 的订阅者**同一个线程**里面去执行。
- Direct Grouping：**直接分组**，这是一种比较特别的分组方法，用这种分组意味着消息的发送者指定由消息接收者的哪个 task 处理这个消息。只有被声明为 Direct Stream 的消息流可以声明这种分组方法。而且这种消息 tuple 必须使用 emitDirect 方法来发射。消息处理者可以通过 TopologyContext 来获取处理它的消息的 taskid（OutputCollector.emit 方法也会返回 taskid）。

【可靠性：Reliability】

Storm 保证每个 tuple 会被 topology 完整的执行。Storm 会追踪由每个 spout tuple 所产生的 tuple 树（一个 bolt 处理一个 tuple 之后可能会发射别的 tuple 从而可以形成树状结构），并且跟踪这棵 tuple 树什么时候成功处理完。每个 topology 都有一个消息超时的设置， 如果 Storm 在这个超时的时间内检测不到某个 tuple 树到底有没有执行成功， 那么 topology 会把这个 tuple 标记为执行失败，并且过一会会重新发射这个 tuple。

为了利用 Storm 的可靠性特性，在你发出一个新的 tuple 以及你完成处理一个 tuple 的时候你必须要通知 Storm。这一切是由 OutputCollector 来完成的。通过它的 emit 方法来通知一个新的 tuple 产生了，通过它的 ack 方法通知一个 tuple 处理完成了。

【任务：Tasks】

每一个 Spout 和 Bolt 会被当作很多 task 在整个集群里面执行。每一个 task 对应到一个线程，而 stream grouping 则是定义怎么从一堆 task 发射 tuple 到另外一堆 task。你可以调用 TopologyBuilder.setSpout()和 TopBuilder.setBolt 来设置并行度——也就是有多少个 task。

【工作进程：Workers】

一个 topology 可能会在一个或者多个工作进程里面执行，每个工作进程执行整个 topology 的一部分。比如对于并行度是 300 的 topology 来说，如果我们使用 50 个工作进程来执行，那么每个工作进程会处理其中的 6 个 tasks（其实就是每个工作进程里面分配 6 个线程）。Storm 会尽量均匀的工作分配给所有的工作进程。

【配置：Configuration】

Storm 里面有一堆参数可以配置来调整 nimbus、supervisor 以及正在运行的 topology 的行为，一些配置是系统级别的，一些配置是 topology 级别的。所有有默认值的配置的默认配置是配置在 default.xml 里面的。你可以通过定义个 Storm.xml 在你的 classpath 厘米来覆盖这些默认配置。并且你也可以在代码里面设置一些 topology 相关的配置信息——使用 StormSubmitter。当然，这些配置的优先级是：default.xml < Storm.xml < TOPOLOGY-SPECIFIC 配置。

在 Storm 中也有对于流 stream 的抽象，流是一个不间断的无界的连续 tuple，注意 Storm 在建模事件流时，把流中的事件抽象为 tuple 即元组，后面会解释 Storm 中如何使用 tuple。

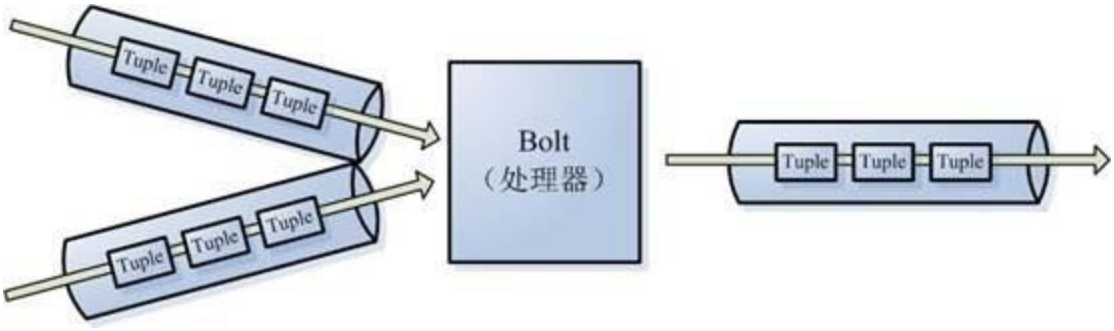


图 1.6 Bolt 处理器数据流

Storm 认为每个 stream 都有一个 stream 源，也就是原始元组的源头，所以它将这个源头抽象为 spout，spout 可能是连接 twitter api 并不断发出 tweets，也可能是从某个队列中不断读取队列元素并装配为 tuple 发射。

有了源头即 spout 也就是有了 stream，那么该如何处理 stream 内的 tuple 呢，同样的思想 twitter 将流的中间状态转换抽象为 bolt，bolt 可以消费任意数量的输入流，只要将流方向导向该 bolt，同时它也可以发送新的流给其他 bolt 使用，这样一来，只要打开特定的 spout（管口）再将 spout 中流出的 tuple 导向特定的 bolt，又 bolt 对导入的流做处理后再导向其他 bolt 或者目的地。

我们可以认为 spout 就是一个一个的水龙头，并且每个水龙头里流出的水是不同的，我们想拿到哪种水就拧开哪个水龙头，然后使用管道将水龙头的水导向到一个水处理器（bolt），水处理器处理后再使用管道导向另一个处理器或者存入容器中。

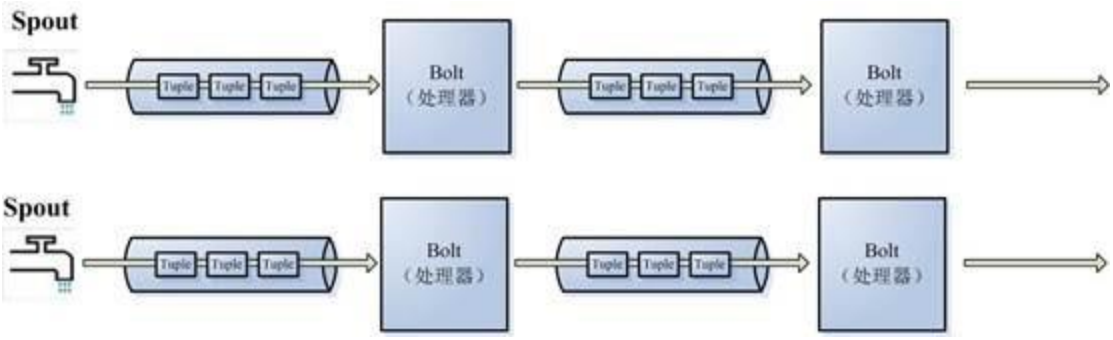


图 1.7 Spout 数据流向图

为了增大水处理效率，我们很自然就想到在同个水源处接上多个水龙头并使用多个水处理器，这样就可以提高效率。没错 Storm 就是这样设计的，看到下图我们就明白了。

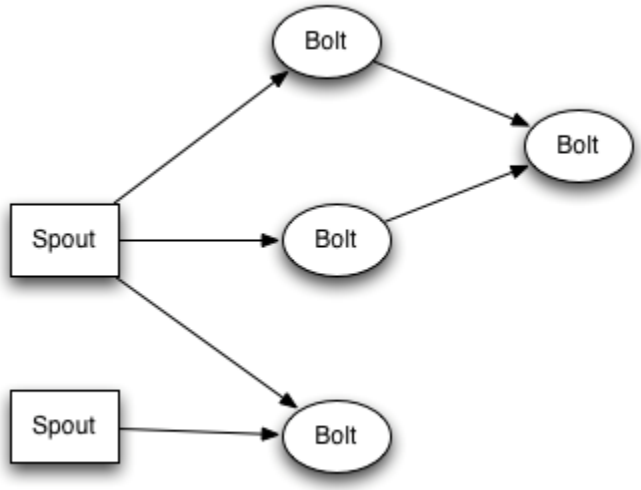


图 1.8 Topology 拓扑图

对应上文的介绍，我们可以很容易的理解这幅图，这是一张有向无环图，Storm 将这个图抽象为 Topology 即拓扑（的确，拓扑结构是有向无环的），拓扑是 Storm 中最高层次的一个抽象概念，它可以被提交到 Storm 集群执行，一个拓扑就是一个流转换图，图中每个节点是一个 spout 或者 bolt，图中的边表示 bolt 订阅了哪些流，当 spout 或者 bolt 发送元组到流时，它就发送元组到每个订阅了该流的 bolt（这就意味着不需要我们手工拉管道，只要预先

订阅，spout 就会将流发到适当 bolt 上）。

插个位置说下 Storm 的 topology 实现，为了做实时计算，我们需要设计一个拓扑图，并实现其中的 Bolt 处理细节，Storm 中拓扑定义仅仅是一些 Thrift 结构体（请 google 一下 Thrift），这样一来我们就可以使用其他语言来创建和提交拓扑。

Storm 则将流中元素抽象为 tuple，一个 tuple 就是一个值列表 value list，list 中的每个 value 都有一个 name，并且该 value 可以是基本类型，字符类型，字节数组等，当然也可以是其他可序列化的类型。拓扑的每个节点都要说明它所发射出的元组的字段的 name，其他节点只需要订阅该 name 就可以接收处理。

1.5 亮点功能

相比于 S4，Puma 等其他实时计算系统，Storm **最大的亮点**在于其记录级容错和能够保证消息精确处理的事务功能。

- Storm 记录级容错

首先来看一下什么叫做记录级容错？Storm 允许用户在 spout 中发射一个新的源 tuple 时为其指定一个 message id，这个 message id 可以是任意的 object 对象。多个源 tuple 可以**共用**一个 message id，表示这多个源 tuple 对用户来说是**同一个**消息单元。Storm 中记录级容错的意思是说，Storm 会告知用户每一个消息单元是否在指定时间内被完全处理了。那什么叫做完全处理呢，就是该 message id 绑定的源 tuple 及由该源 tuple 后续生成的 tuple 经过了 topology 中每一个应该到达的 bolt 的处理。举个例子。在图 1.9 中，在 spout 由 message 1 绑定的 tuple1 和 tuple2 经过了 bolt1 和 bolt2 的处理生成两个新的 tuple，并最终都流向了 bolt3。当这个过程完成处理完时，称 message 1 被完全处理了。

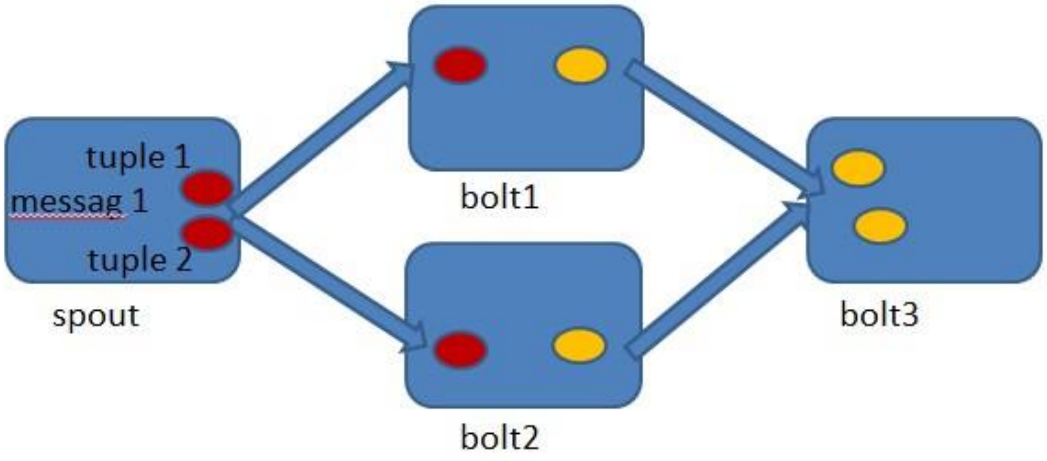


图 1.9 Storm 记录级容错原理图之一

在 Storm 的 topology 中有一个系统级组件，叫做 acker。这个 acker 的任务就是追踪从 spout 中流出来的每一个 message id 绑定的若干 tuple 的处理路径，如果在用户设置的最大超时时间内这些 tuple 没有被完全处理，那么 acker 就会告知 spout 该消息处理失败了，相反则会告知 spout 该消息处理成功了。在刚才的描述中，我们提到了“记录 tuple 的处理路径”，

如果曾经尝试过这么做的同学可以仔细地思考一下这件事的复杂程度。但是 Storm 中却是使用了一种非常巧妙的方法做到了。在说明这个方法之前，我们来复习一个数学定理。**A xor A = 0，A xor B…xor B xor A = 0**，其中每一个操作数出现且仅出现两次。

Storm 中使用的巧妙方法就是基于这个定理。具体过程是这样的：在 spout 中系统会为用户指定的 message id 生成一个对应的 64 位整数，作为一个 root id。root id 会传递给 acker 及后续的 bolt 作为该消息单元的唯一标识。同时无论是 spout 还是 bolt 每次新生成一个 tuple 的时候，都会赋予该 tuple 一个 64 位的整数的 id。Spout 发射完某个 message id 对应的源 tuple 之后，会告知 acker 自己发射的 root id 及生成的那些源 tuple 的 id。而 bolt 呢，每次接受到一个输入 tuple 处理完之后，也会告知 acker 自己处理的输入 tuple 的 id 及新生成的那些 tuple 的 id。Acker 只需要对这些 id 做一个简单的异或运算，就能判断出该 root id 对应的消息单元是否处理完成了。下面通过一个图示来说明这个过程。

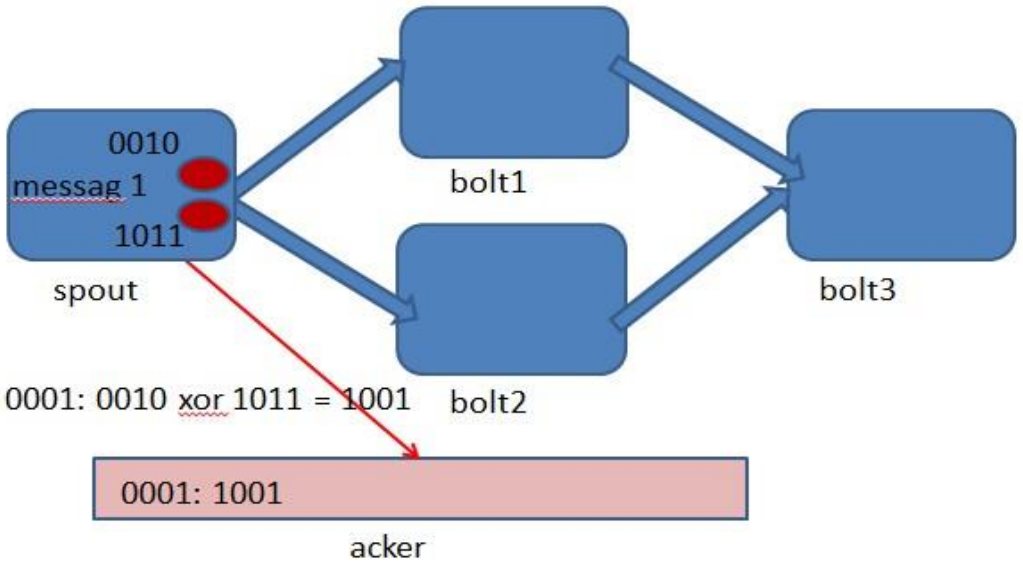


图 1.10 Storm 记录级容错原理图之二

图 1.10 spout 中绑定 message 1 生成了两个源 tuple，id 分别是 0010 和 1011。

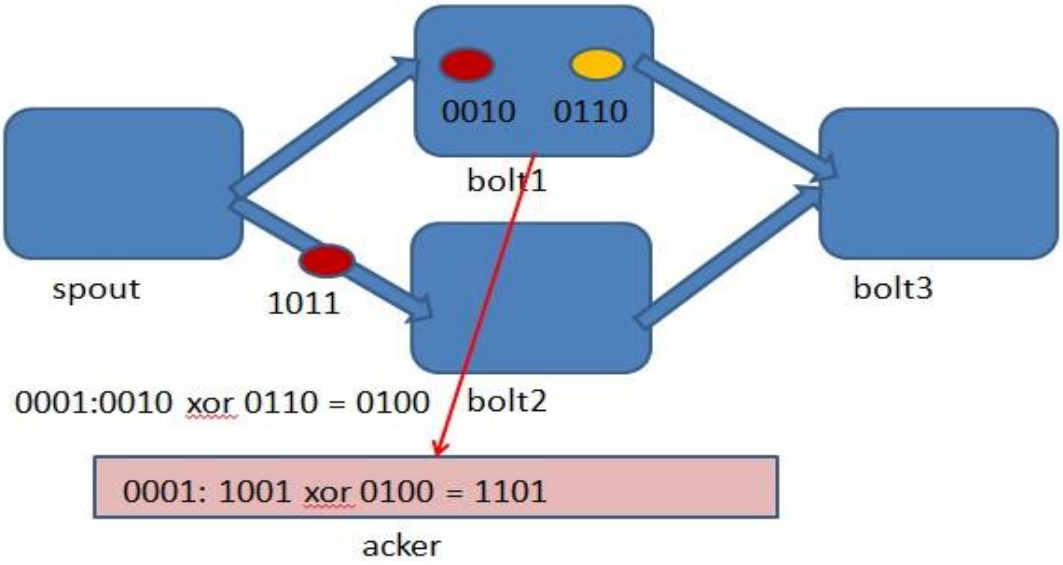


图 1.11 Storm 记录级容错原理图之三

图 1.11 bolt1 处理 tuple 0010 时生成了一个新的 tuple，id 为 0110。

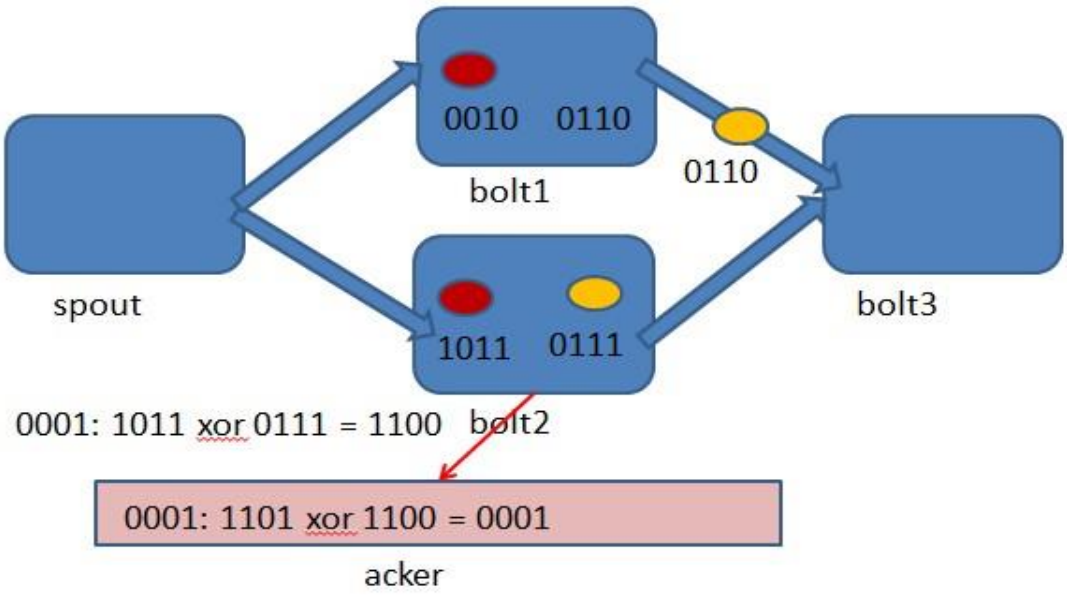


图 1.12 Storm 记录级容错原理图之四

图 1.12 bolt2 处理 tuple 1011 时生成了一个新的 tuple，id 为 0111。

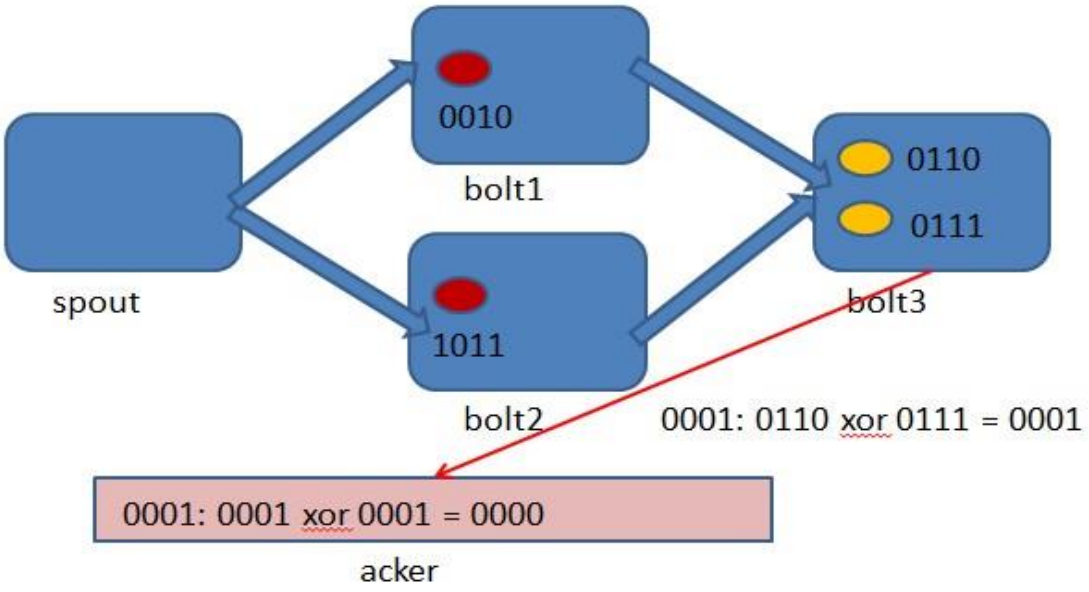


图 1.13 Storm 记录级容错原理图之五

图 1.13 bolt3 中接收到 tuple 0110 和 tuple 0111，没有生成新的 tuple。

可能有些细心的同学会发现，容错过程存在一个可能出错的地方，那就是，如果生成的 tuple id 并不是完全各异的，acker 可能会在消息单元完全处理完成之前就错误的计算为 0。这个错误在理论上的确是存在的，但是在实际中其概率是极低极低的，完全可以忽略。

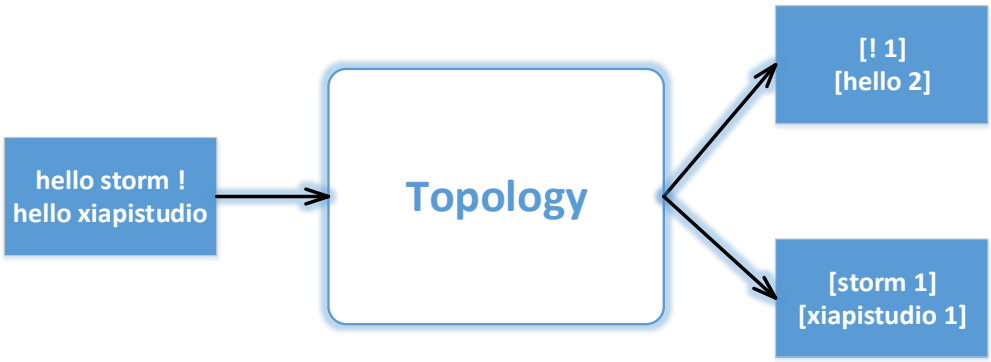
- Storm 的事务拓扑

事务拓扑（transactional topology）是 Storm0.7 引入的特性，在最近发布的 0.8 版本中已经被封装为 Trident，提供了更加便利和直观的接口。因为篇幅所限，在此对事务拓扑做一个简单的介绍。

事务拓扑的目的是为了满足对消息处理有着极其严格要求的场景，例如实时计算某个用户的成交笔数，要求结果完全精确，不能多也不能少。Storm 的事务拓扑是完全基于它底层的 spout/bolt/acker 原语实现的，通过一层巧妙的封装得出一个优雅的实现。个人觉得这也是 Storm 最大的魅力之一。

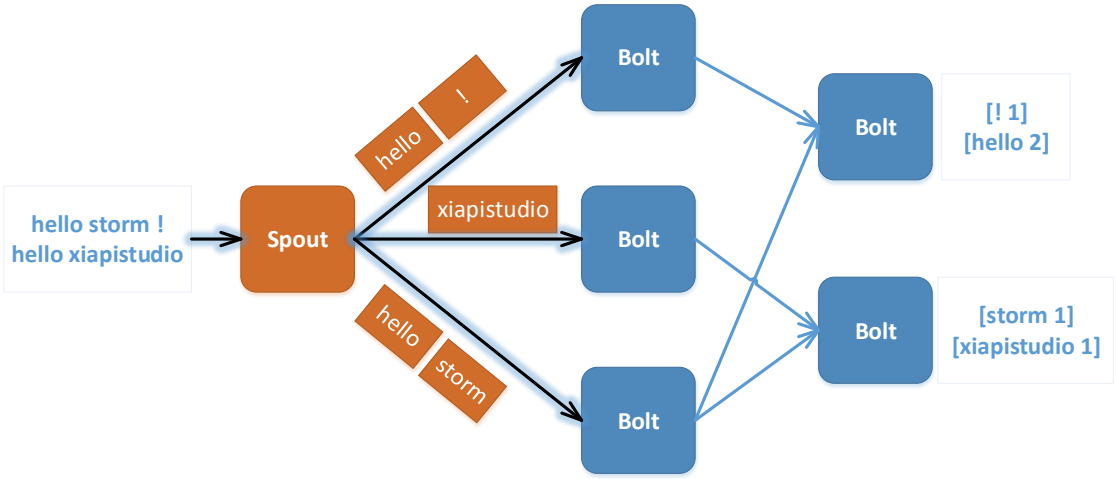
事务拓扑简单来说就是将消息分为一个个的批（batch），同一批内的消息以及批与批之间的消息可以并行处理，另一方面，用户可以设置某些 bolt 为 committer，Storm 可以保证 committer 的 finishBatch()操作是按严格不降序的顺序执行的。用户可以利用这个特性通过简单的编程技巧实现消息处理的精确。

1.6 图形案例

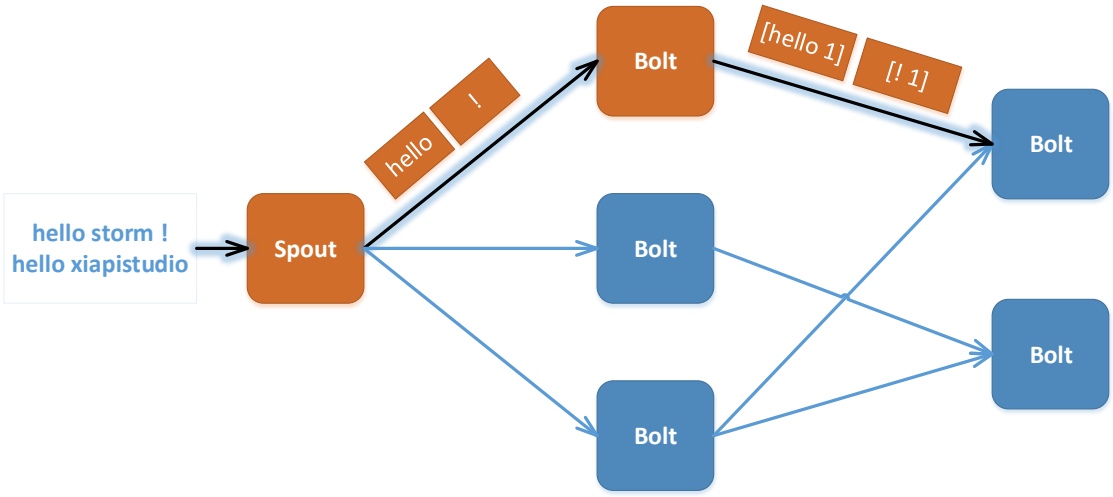


Storm 版的 WordCount 图

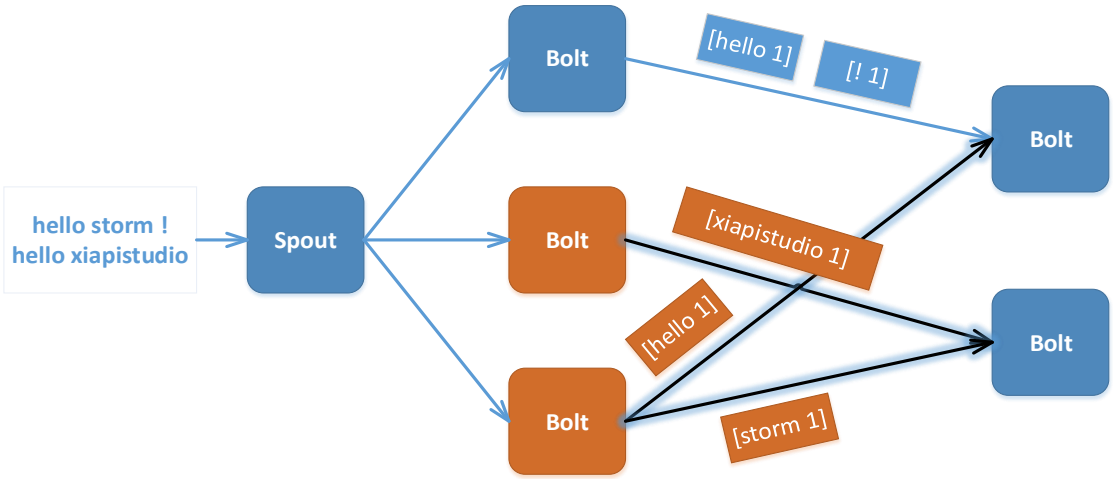
WordCount 例子在分布式计算中堪比 HelloWorld 的一个入门级程序，是每一个学习大数据处理，必先接触的一个程序，现在我们通过图形的方式了解 Storm 是如何实现 WordCount 的，相关的过程如下所示：



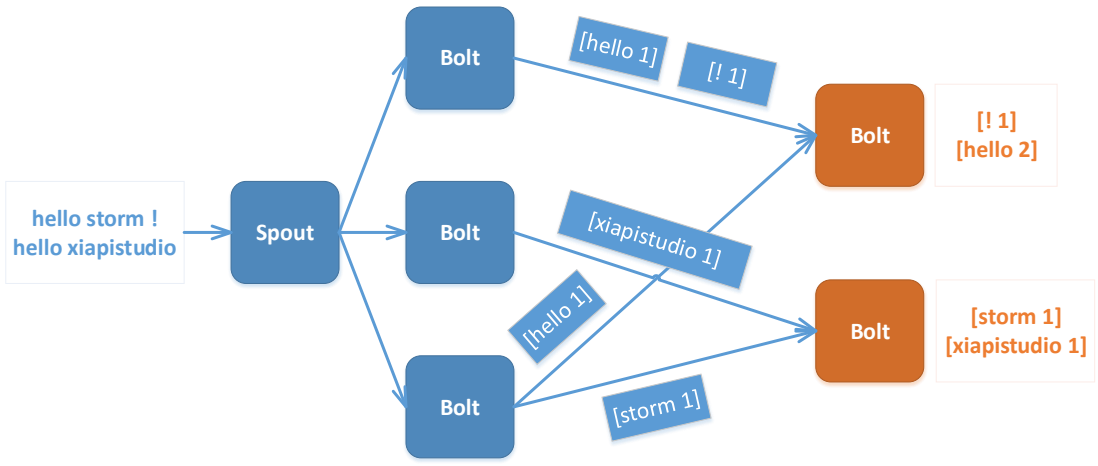
Spout 发送 Word 到 SetValBolt



其中一个 SetValBolt 处理 Word 的过程

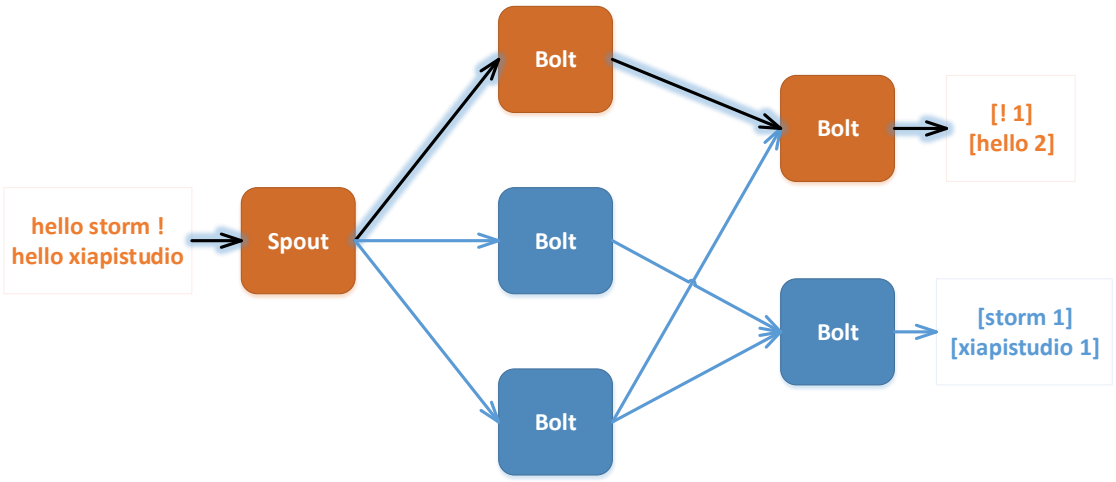


剩余两个 SetValBolt 处理 Word 的过程

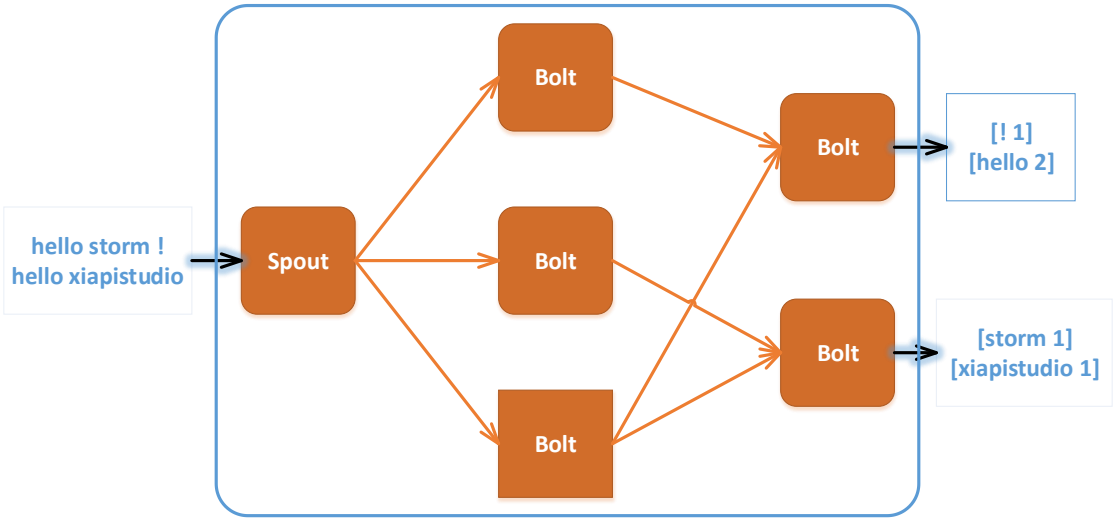


最后 CountBolt 对 Word 进行计数累加

下面我们在 WordCount 图形的基础上面对前面讲到的几个概念温习一下，下面我们看一下 Stream 在图形中是指什么？

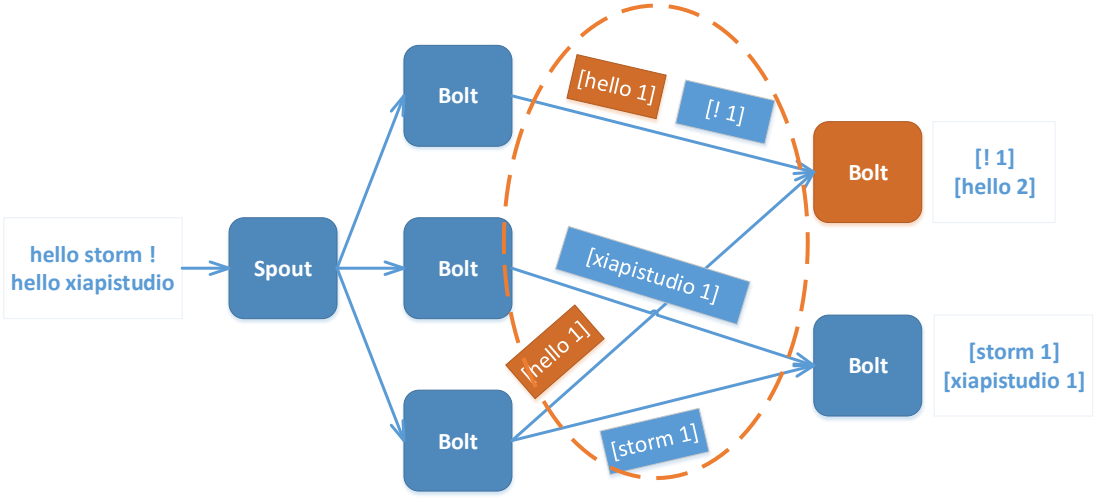


Storm 中的 Stream 模型



Storm 中的 Topology 模型

其中在图解过程中，大家会有一个疑问，再由 SetValBolt 到 CountBolt 过程中，为何两个“hello”单词会到同一个 CountBolt 进行处理呢？



Storm 中的 Grouping 模型

1.7 发展趋势

Storm 已经发展到 0.10.0 版本了，看一下四年多来，它取得的成就：

- 有 50 个大大小小的公司在使用 Storm，相信更多的不留名的公司也在使用。这些公司中不乏淘宝、百度、Twitter、Groupon、雅虎等重量级公司。
- 从开源时候的 0.5.0 版本，到现在的 0.9.0+。先后添加了以下重大的新特性：
  - 使用 kryo 作为 Tuple 序列化的框架（0.6.0）
  - 添加了 Transactional topologies（事务性拓扑）的支持（0.7.0）
  - 添加了 Trident 的支持（0.8.0）
  - 引入 Netty 作为底层消息机制（0.9.0）
  - 增加 Kafka、Hive、HBase 等插件（0.9.2-0.9.5）

Transactional topologies 和 Trident 都是针对实际应用中遇到的**重复计数**问题和**应用性**问题的解决方案。可以看出，实际的商用给予了 Storm 很多良好的反馈。

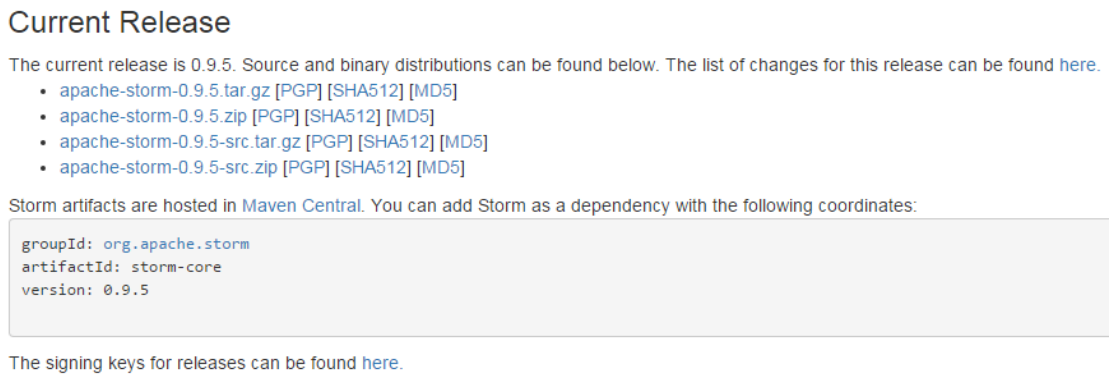
- 在 GitHub 上超过 4000 个项目负责人。Storm 集成了许多库，支持包括 Kestrel、Kafka、JMS、Cassandra、Memcached 以及更多系统。随着支持的库越来越多，Storm 更容易与现有的系统协作。

Storm 的拥有一个活跃的社区和一群热心的贡献者。过去两年，Storm 的发展是成功的。

2、Storm 安装

2.1 版本选择

官网：http://storm.apache.org/



点击“apache-storm-0.9.5.zip”，进入下载页面，使用下面命令下载：

```
wget http://apache.go-parts.com/storm/apache-storm-0.9.5/apache-storm-0.9.5.zip
```

另外 Storm 是在 0.9.0.1 以后，Storm 的安装变的非常简单，只需要安装 Java 和 Python 环境即可。如果使用最新的 Linux，可能现有的版本条件就可以满足，不然还需要额外的安装。目前我们的 Storm 0.9.5 需要的 Java 版本为 1.6 +和 Python 版本为 2.6.6 +。本文不在另行安装较少如何安装 Java 和 Python，等安装完毕后，使用 java –version 和 python –V 进行验证安装的版本是否符合要求。

2.2 安装 Zookeeper

ZooKeeper 是一个分布式的，开放源码的分布式应用程序协调服务，是 Google 的 Chubby 一个开源的实现，是 Hadoop 和 Hbase 的重要组件。它是一个为分布式应用提供一致性服务的软件，提供的功能包括：配置维护、名字服务、分布式同步、组服务等。

Storm 中使用 Zookeeper 主要用于 Storm 集群各节点的分布式协调工作，具体功能如下：

- 存储客户端提供的 topology **任务信息**，nimbus 负责将任务分配信息写入 Zookeeper，supervisor 从 Zookeeper 上读取任务分配信息；
- 存储 supervisor 和 worker 的心跳（包括**它们的状态**），使得 nimbus 可以监控整个集群的状态， 从而重启一些挂掉的 worker；
- 存储整个集群的**所有状态信息**和**配置信息**。

本文档系列主要是介绍 Storm，顾 Zookeeper 的安装，我们仅仅安装其单机模式，安装的版本为 Zookeeper-3.4.6，相关的网址如下：

官网：http://zookeeper.apache.org/

下载：wget http://apache.arvixe.com/zookeeper/zookeeper-3.4.6/zookeeper-3.4.6.tar.gz

下载完后解压，然后执行如下命令：

```
cd zookeeper-3.4.6/conf
cp ./zoo_sample.cfg    ./zoo.cfg
```

在接着修改 zoo.cfg 中的 dataDir 参数，指定到合理的位置，例如：

```
dataDir=/home/xiapistudio/realtime/data/zookeeper
```

从版本 3.4.0 开始，Zookeeper 提供了自动清理快照（snapshot）和事务日志的功能，需要在 zoo.cfg 配置文件中设置。

- autopurge.purgeInterval=1
- autopurge.snapRetainCount=3

其中，“autopurge.purgeInterval”这个参数制定了持久化日志清理频率，单位是小时，默认是 0，表示不开启自动清理功能；“autopurge.snapRetainCount”这个参数和前面的参数搭配使用，用于指定需要保留的持久化日志的文件数目，默认是保留 3 个。

当然设置的目录，必须预先自己创建，具体情况根据实际情况操作。剩下的就是启动单机的 Zookeeper，通过下面命令启动：

```
/home/xiapistudio/realtime/zookeeper-3.4.6/bin/zkServer.sh start
```

当启动完成后，可以使用 jps 命令检查是否启动，如果启动，存在“QuorumPeerMain”的进程，说明我们已经启动成功了，当然你可以用下面命令进入 ZK：

```
/home/xiapistudio/realtime/zookeeper-3.4.6/bin/zkCli.sh
```

温馨提示：Zookeeper 推荐部署奇数台服务器（根据 Zookeeper 的特性，2N+1 台的 Zookeeper 集群，当 N 个节点不能访问时，整个 Zookeeper 仍然是可用的）。

2.3 安装 Storm

首先从官网上下载最新的版本，我们使用的是 Storm0.9.5，本文的 Storm 是单机伪分布式，即 Nimbus 和 Supervisor 均在一个机器上面。在实际的工作中，Nimbus 和 UI 在一台机器，而其余的机器皆是 Supervisor。而 Storm 的扩展非常的简单，只要一台机器部署好了，把相关的文件复制过去，直接用命令启动即可。

1）从前面指定的官网上面下载相应的版本，然后解压到指定的目录，本文的目录为“/home/xiapistudio/realtime/apache-storm-0.9.5”中。

```
cd /home/xiapistudio/realtime
wget http://apache.go-parts.com/storm/apache-storm-0.9.5/apache-storm-0.9.5.zip ./
unzip apache-storm-0.9.5.zip
```

2）修改 conf/storm_env.ini，指定使用的 java 环境

```
# Environment variables in the following section will be used
# in storm python script. They override the environment variables
# set in the shell.
[environment]

# The java implementation to use. If JAVA_HOME is not found we expect java to be in path
JAVA_HOME:/home/xiapistudio/realtime/jdk1.6.0_45
```

温馨提示：在这里指定有个好处，当机器中存在多个 java 版本时，可以保证 Storm 程序的应用欢迎而不受影响。

3）修改 conf/storm.yaml，指定 strom 的以下几项配置

（1）storm.zookeeper.servers

Storm 集群使用的 Zookeeper 集群地址，其格式如下：

```
 storm.zookeeper.servers:
    - "127.0.0.1"
#    - " server2 "
```

如果 Zookeeper 集群使用的不是默认端口，那么还需要 storm.zookeeper.port 选项。

（2）storm.local.dir

Nimbus 和 Supervisor 进程用于存储少量状态，如 jars、confs 等的本地磁盘目录，需要提前创建该目录并给以足够的访问权限。然后在 storm.yaml 中配置该目录，如：

```
storm.local.dir: "/home/xiapistudio/realtime/data/storm"
```

（3）nimbus.host

Storm 集群 Nimbus 机器地址，各个 Supervisor 工作节点需要知道哪个机器是 Nimbus，以便下载 Topologies 的 jars、confs 等文件，如：

```
nimbus.host: "www.xiapistudio.com"
```

备注：这里可以是 hostname，也可以是 ip。

（4）supervisor.slots.ports

对于每个 Supervisor 工作节点，需要配置该工作节点可以运行的 worker 数量。每个 worker 占用一个单独的端口用于接收消息，该配置选项即用于定义哪些端口是可被 worker 使用的。默认情况下，每个节点上可运行 4 个 workers，分别在 6700、6701、6702 和 6703 端口，如：

```
supervisor.slots.ports:
    - 6700
    - 6701
    - 6702
    - 6703
```

温馨提示：每个配置的前面都有一个空格，在编辑的时候不能去除掉，不然会导致配置无法生效，这是默认规则。

2.4 集成 Kafka

本一步骤并不是必须的，如果你的系统需要从 kafka 里面读取数据进行处理，那么这时还需要在 storm 的 lib 里面增加点必要的 jar 包才可以。

假如你使用的 kafka 的版本为“kafka_2.9.2-0.8.1.1”，并且你所编写的 Topology 工程的依赖均为“provided”的 scope，则需要将涉及到的依赖 jar 包拷贝到 Storm 安装目录的 lib 文件夹下，包括：

- kafka_2.10-0.8.2.1.jar
- scala-library-2.10.4.jar
- metrics-core-2.2.0.jar
- storm-kafka-0.9.3.jar
- zookeeper-3.4.6.jar
- curator-client-2.8.0.jar
- curator-framework-2.8.0.jar
- curator-recipes-2.8.0.jar
- guava-18.0.jar

备注：记得是在启动集群前拷贝进去，不然无法生效。

2.5 启动 Storm

最后一步，启动 Storm 的所有后台进程。和 Zookeeper 一样，Storm 也是快速失败（fail-fast）的系统，这样 Storm 才能在任意时刻被停止，并且当进程重启后被正确地恢复执行。这也是为什么 Storm 不在进程内保存状态的原因，即使 Nimbus 或 Supervisors 被重启，运行中的 Topologies 不会受到影响。

Storm 集群分为 Nimbus 节点和 Supervisor 节点。

- Nimbus 节点：用于提交应用 Topology、管理整个 Storm 节点（将 Topology 的 Task 分配给 Worker、监控各个 Supervisor 节点的状态进行负载均衡等）。Nimbus 节点上不能运行 Worker。
- Supervisor 节点：负载从 Zookeeper 上获取、启动并运行任务。

因此相对而言，我们认为 Nimbus 并不需要 Supervisor 节点那么高的配置，在我们的测试环境中，Nimbus 的硬件配置只有 Supervisor 节点的一半。Storm UI 节点也不需要高配置，可以和 Nimbus 节点在同一台机器上。

1）在 Nimbus 节点启动 nimbus、storm-ui 和 logviewer：

```
cd /home/xiapistudio/realtime/apache-storm-0.9.5
mkdir logs
nohup bin/storm ui > logs/ui.out 2>&1 &
nohup bin/storm nimbus > logs/nimbus.out 2>&1 &
nohup bin/storm logviewer > logs/logviewer.out 2>&1 &
```

2）在各 Supervisor 节点启动 supervisor 和 logviewer：

```
cd /home/xiapistudio/realtime/apache-storm-0.9.5
mkdir logs
nohup bin/storm logviewer > logs/logviewer.out 2>&1 &
nohup bin/storm supervisor > logs/supervisor.out 2>&1 &
```

至此，Storm 集群已经部署、配置完毕，可以向集群提交拓扑运行了。

温馨提示：Storm 后台进程被启动后，将在 Storm 安装部署目录下的 logs 子目录下生成各个进程的日志文件。经测试，Storm UI 必须和 Storm Nimbus 部署在同一台机器上，否则 UI 无法正常工作，因为 UI 进程会检查本机是否存在 Nimbus 链接。为了方便使用，可以将 bin/storm 加入到系统环境变量中。

2.6 验证 Storm

启动 UI 后台程序，并放到后台执行，启动后可以通过 http://{nimbus host}:8080 观察集群的 worker 资源使用情况、Topologies 的运行状态等信息。当然也可以通过 jps 查看各节点

上面是否正常启动相应的 storm 服务。

当集群成功后，就可以向集群提交任务了。

1）启动 Storm Topology：

```
bin/storm jar storm.jar com.xiapistudio.DemoTopology arg1 arg2 arg3
```

其中，storm.jar 是包含 Topology 实现代码的 jar 包，com.xiapistudio.DemoTopology 的 main 方法是 Topology 的入口，arg1、arg2 和 arg3 为 com.xiapistudio.DemoTopology 执行时需要传入的参数。

2）停止 Storm Topology：

```
bin/storm kill {toponame}
```

其中，{toponame}为 Topology 提交到 Storm 集群时指定的 Topology 任务名称。

3、参考文献

感谢以下文章的编写作者，没有你们的铺路，我或许会走得很艰难，参考不分先后，贡献同等珍贵。

1） http://ju.outofmemory.cn/entry/118203
2） http://weyo.me/pages/techs/kafka-storm-integration/
3） http://blog.csdn.net/cjfeii/article/details/24706321
4） http://www.cnblogs.com/panfeng412/archive/2012/11/30/how-to-install-and-deploy-storm-cluster.html
5） http://www.aboutyun.com/thread-6854-1-1.html
6） http://blog.csdn.net/WeiJonathan/article/details/17762477
7） http://isuifengfei.iteye.com/blog/1998269
8） http://www.aboutyun.com/thread-7394-1-1.html
9） http://www.kuqin.com/system-analysis/20120906/330093.html
10） http://wenku.baidu.com/view/e222a8eb8bd63186bdebbc13.html
11） http://wenku.baidu.com/view/fed961e281c758f5f61f6711.html
12） http://wenku.baidu.com/view/3b695d5c7fd5360cba1adbf3.html
13） http://www.searchtb.com/2012/09/introduction-to-storm.html
14） http://blog.csdn.net/yangbutao/article/category/1314490
15） http://blog.csdn.net/hguisu/article/details/8454368

4、打赏小编

	编辑简介： 高级软件工程师（T5），河北工业大学硕士研究生，现在就职于百度在线网络技术（北京）有限公司。专注于大数据以及其相关研究，在离线计算和实时计算方面有较为深入的研究，积累了丰富的实战经验。热衷于知识分享，其细细品味系列教程深受网友喜爱。
姓名：解耀伟	网站：www.xiapistudio.com
笔名：虾皮	博客：http://www.cnblogs.com/xia520pi/
扣扣：461052034	邮箱：xieyaowei1986@163.com

从高考复习开始养成了总结的习惯，习惯于在学习的过程中，把相关的文章融会贯通，并加以实践，结合自己的实际情况把相关的内容整理成册，便于学习和总结。在这几年里陆陆续续分享了很多细细品味系列文章。

每一期文章都耗费了不少的心血，很多时候都是在星期天业余的时间完成，现在也建立了自己独立的网站：www.xiapistudio.com，需要一些资金来维持，同时也可以鼓励我写更多的好东西来分享。如果你看了本文章对自己有用，可以通过支付宝的形式来进行打赏，1 元、2 元、10 元皆可，多少并不重要，只要你感觉文章使你受益即可。

温馨提示：在转账时，可以写明“打赏虾皮”或者“打赏虾皮工作室”。我的支付宝已经进行实名认证，支付宝是的个人头像，请认准后再支付。