获取教材和讲义 PPT 等各种课程资料请访问 http://dblab.xmu.edu.cn/node/422

=课程教材由林子雨老师根据网络资料编著=

厦门大学计算机科学系教师 林子雨 编著

http://www.cs.xmu.edu.cn/linziyu

2013 年 9 月

前言

本教程由厦门大学计算机科学系教师林子雨编著，可以作为计算机专业研究生课程《大数据技术基础》的辅助教材。

本教程的主要内容包括：大数据概述、大数据处理模型、大数据关键技术、大数据时代面临的新挑战、NoSQL 数据库、云数据库、Google Spanner、Hadoop、HDFS、HBase、MapReduce、Zookeeper、流计算、图计算和 Google Dremel 等。

本教程是林子雨通过大量阅读、收集、整理各种资料后精心制作的学习材料，与广大数据库爱好者共享。教程中的内容大部分来自网络资料和书籍，一部分是自己撰写。对于自写内容，林子雨老师拥有著作权。

本教程 PDF 文档及其全套教学 PPT 可以通过网络免费下载和使用（下载地址：http://dblab.xmu.edu.cn/node/422）。教程中可能存在一些问题，欢迎读者提出宝贵意见和建议！

本教程已经应用于厦门大学计算机科学系研究生课程《大数据技术基础》，欢迎访问 2013 班级网站 http://dblab.xmu.edu.cn/node/423。

林子雨的 E-mail 是：ziyulin@xmu.edu.cn。

林子雨的个人主页是：http://www.cs.xmu.edu.cn/linziyu。

林子雨于厦门大学海韵园

2013 年 9 月

第 8 章 流计算

厦门大学计算机科学系教师 林子雨 编著

个人主页：http://www.cs.xmu.edu.cn/linziyu

课程网址：http://dblab.xmu.edu.cn/node/422

2013 年 9 月

第 8 章 流计算

随着大数据时代的到来，数据量急剧膨胀，业务也变得越加复杂。在业务中产生了源源不断的数据流，而数据的价值又随着时间的流逝而降低，如何实时处理海量流数据成为一大挑战。传统的数据库方案已不适合处理这样的数据，而流计算则可以持续地对流数据进行分析，实时得出有价值的信息。

本章内容首先介绍了什么是流计算，包括概念、处理模型和处理流程，并详细介绍了当前热门的开源流计算框架 Storm，内容要点如下：

- 流计算概述
- 流计算处理流程
- 流计算应用
- 流计算框架 Storm

8.1 流计算概述

8.1.1 什么是流计算

近年来，一种新的数据密集型应用已经得到了广泛的认同，这类应用的特征是：数据不宜用持久稳定的关系型模型建模，而适宜用瞬态数据流建模。这些应用的实例包括金融服务、网络监控、电信数据管理、Web 应用、生产制造、传感检测等等。在这种数据流模型中，单独的数据单元可能是相关的元组，例如网络测量、呼叫记录、网页访问等产生的数据。但是，这些数据以大量、快速、时变（可能是不可预知的）的数据流形式持续到达，由此产生了一些基础性的新的研究问题。

互联网从诞生的第一时间起，对世界的最大的改变就是让信息能够实时交互，数据库和高速网络的发展更是给互联网业务带来了实时性的改变。对于实时性要求很高的应用，若把持续到达的数据简单地放到 DBMS 中，再在其中进行操作，是不太现实的。传统的 DBMS 并不是为快速连续的存放单独的数据单元而设计的，而且也并不支持“持续处理”，而“持续处理”是数据流应用的典型特征。

随着大数据时代的到来，互联网业务的发展从初期数据量小、业务简单，到过渡期数据有所膨胀、业务较复杂，再到如今大数据时期数据量急剧膨胀，业务很复杂的情况。面对大数据，特别是流数据的实时化需求，传统的数据库技术方案已不能满足需求，成本高的同时并不能带来高效率，急需针对流数据的实时计算——流计算。

流计算，即针对流数据的实时计算。要了解流计算，就要了解两个概念：流数据和实时计算。

流数据，也称流式数据，是指将数据看作数据流的形式来处理。数据流是在时间分布和数量上无限的一系列动态数据集合体；数据记录是数据流的最小组成单元。数据流具有如下特征：

- 数据连续不断；
- 数据来源众多，格式复杂；
- 数据量大，但是不十分关注存储；
- 注重数据的整体价值，不要过分关注个别数据；
- 数据流顺序颠倒，或者不完整。

实时计算是针对大数据而言的。对于少量数据而言，实时计算并不存在问题，但随着数据量的不断膨胀，实时计算就发生了质的改变，数据的结构与来源越来越多样化，实时计算的逻辑也变得越来越复杂。除了像非实时计算的需求（如计算结果准确）以外，实时计算最重要的一个需求是能够实时响应计算结果，一般要求为秒级。

图 8-1 是一个流计算的示意图：实时获取来自不同终端的海量数据，经过流计算平台的不断地分析处理，整合获得有价值的信息。

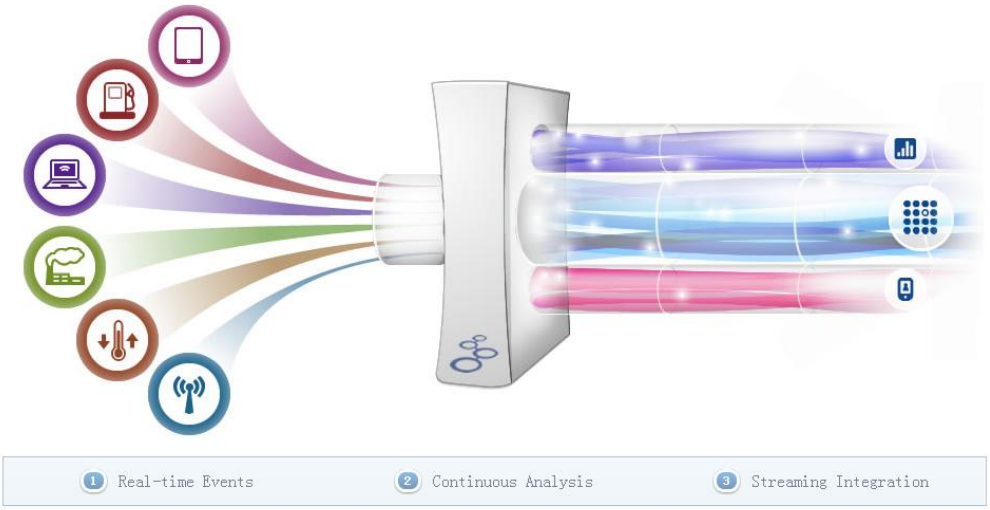

图 8-1 流计算示意图

总的来说，流计算来自一个信念：数据的价值随着时间的流逝而降低。所以，事件出现后必须尽快地对它们进行处理，最好数据出现时便立刻对其进行处理。发生一个事件就进行一次处理，而不是缓存起来成一批再处理。例如商用搜索引擎，像 Google、Bing 和 Yahoo！等，通常在用户查询响应中提供结构化的 Web 结果，同时也插入基于流量的点击付费模式的文本广告。为了在页面上最佳位置展现最相关的广告，就需要对用户数据进行实时分析，通过一些算法来动态估算给定上下文中一个广告被点击的可能性，从而能展示更佳的广告。为了及时处理用户反馈，需要一个低延迟、可扩展、高可靠的处理引擎。

8.1.2 数据流与传统的关系存储模型的区别

在数据流模型中，需要处理的输入数据（全部或部分）并不存储在可随机访问的磁盘或内存中，而是以一个或多个“连续数据流”的形式到达。数据流不同于传统的关系存储模型，主要区别有如下几个方面：

- 流中的数据元素在线到达；
- 系统无法控制将要处理的新到达的数据元素的顺序；
- 数据流的潜在大小也许是无穷无尽的；
- 一旦数据流中的某个元素经过处理，要么被丢弃，要么被归档存储。因此，除非该数据被直接存储在内存中，否则将不容易被检索。相对于数据流的大小，这是一种典型的极小相关。

8.1.3 流计算需求

对于一个流计算系统来说，它应达到如下需求：

- 高性能：处理大数据的基本要求，如每秒处理几十万条数据。
- 海量式：支持 TB 级甚至是 PB 级的数据规模。
- 实时性：必须保证一个较低的延迟时间，达到秒级别，甚至是毫秒级别。
- 分布式：支持大数据的基本架构，必须能够平滑扩展。
- 易用性：能够快速进行开发和部署。
- 可靠性：能可靠地处理流数据。

针对不同的应用场景，相应的流计算系统会有不同的需求，但是，针对海量数据的流计算，无论在数据采集、数据处理中都应达到秒级别的要求。

8.1.4流计算与 Hadoop

谈到大规模数据的处理，很容易想到 Hadoop 和 MapReduce。Hadoop 是大数据分析领域的王者，那么 MapReduce 模式能否胜任实时流计算系统的需求呢？

Hadoop 在本质上是一个批处理系统。数据被引入 Hadoop 文件系统 (HDFS) 并分发到各个节点进行处理。当处理完成时，结果数据返回到 HDFS 供始发者使用。Hadoop 的批量化处理是人们喜爱它的地方，但这在某些领域仍显不足，尤其是在例如移动、Web 客户端或金融、网页广告等需要实时计算的领域。这些领域产生的数据量极大，没有足够的存储空间来存储每个业务收到的数据。而流计算则可以实时对数据进行分析，并决定是否抛弃无用的数据，而这无需经过 Map/Reduce 的环节。

为了保证实时性，许多实时数据流处理系统都是专用系统，它们不得不面对可靠性、扩展性和伸缩性方面的问题。使用 MapReduce 的好处在于 Hadoop 帮助业务屏蔽了底层处理，上层作业不用关心容错和扩容方面的问题，应用升级也很方便。不过基于 MapReduce 的业务不得不面对处理延迟的问题。有一种想法是将基于 MapReduce 的批量处理转为小批量处理，将输入数据切成小的片段，每隔一个周期就启动一次 MapReduce 作业，这种实现需要减少每个片段的延迟，并且需要考虑系统的复杂度：

- 将输入数据分隔成固定大小的片段，再由 MapReduce 平台处理，缺点在于处理延迟与数据片段的长度、初始化处理任务的开销成正比。小的分段是会降低延迟，但是，也增加附加开销，并且分段之间的依赖管理更加复杂（例如一个分段可能会需要前一个分段的信息）；反之，大的分段会增加延迟。最优化的分段大小取决于具体应用。
- 为了支持流式处理，MapReduce 需要被改造成 Pipeline 的模式，而不是 reduce 直接输出；考虑到效率，中间结果最好只保存在内存中等等。这些改动使得原有的 MapReduce 框架的复杂度大大增加，不利于系统的维护和扩展。
- 用户被迫使用 MapReduce 的接口来定义流式作业，这使得用户程序的可伸缩性降低。

MapReduce 框架为批处理做了高度优化，系统典型地通过调度批量任务来操作静态数据，任务不是常驻服务，数据也不是实时流入；而数据流计算的典型范式之一是不确定数据速率的事件流流入系统，系统处理能力必须与事件流量匹配。数据流实时处理的模式决定了要和批处理使用非常不同的架构，试图搭建一个既适合流式计算又适合批处理的通用平台，

结果可能会是一个高度复杂的系统，并且最终系统可能对两种计算都不理想。

如Facebook通过对Hadoop/HBase进行实时化改造，使其具有了一定的实时处理能力（可参阅 Facebook 发布的论文《Apache Hadoop Goes Realtime at Facebook》），但这并不能算是一个较好的通用流计算解决方案。

因此，当前业界诞生了许多专门的数据流实时计算系统来满足各自需求，当然除了延迟，它们需要解决可靠性、扩展性和伸缩性等方面的挑战。

8.2 流计算处理流程

传统的数据操作（如图 8-2 所示），首先将数据采集并存储在 DBMS 中，然后通过查询和 DBMS 进行交互，得到用户想要的结果。这样的一个流程隐含了两个前提：

- 数据是旧的。当对数据做查询的时候，里面数据已经是过去某一个时刻数据的一个快照，这些数据可能已经过期了；
- 这样的流程需要人们主动发出查询。也就是说**用户是主动的，而 DBMS 系统是被动的**。

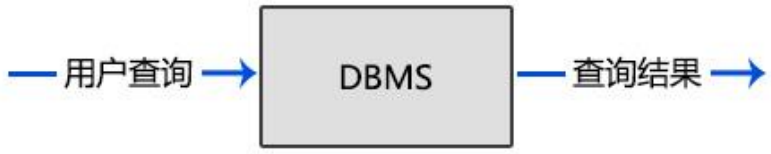

图 8-2 传统数据操作流程

对于流计算（如图 8-3 所示），其数据的处理流程一般有三个阶段：数据实时采集、数据实时计算、实时查询服务。

图 8-3 流计算的数据处理流程

8.2.1 数据实时采集

在数据实时采集阶段，由于现在分布式集群得到广泛应用，数据可能分散存储在不同的机器上，要处理这些数据，首先就要进行一个实时采集的过程，汇总来自不同机器上的数据。数据的实时采集要保证实时性、低延迟与稳定可靠。

目前有许多优秀的开源分布式日志收集系统均可满足每秒数百 MB 的数据采集和传输需求。如 Hadoop 的 Chukwa、Facebook 的 Scribe、LinkedIn 的 Kafka、Cloudera 的 Flume、淘宝的 TimeTunnel 等。

一般来说，数据采集系统基本架构有三个部分（如图 8-4 所示）：

- Agent: 主动采集数据，并把数据推送到 collector；
- Collector: 接收多个 Agent 的数据，并实现有序、可靠、高性能的转发；
- Store: 存储 Collector 的数据。

但对于流计算，一般在 Store 部分不进行存储，而是直接发送给流计算平台进行计算。

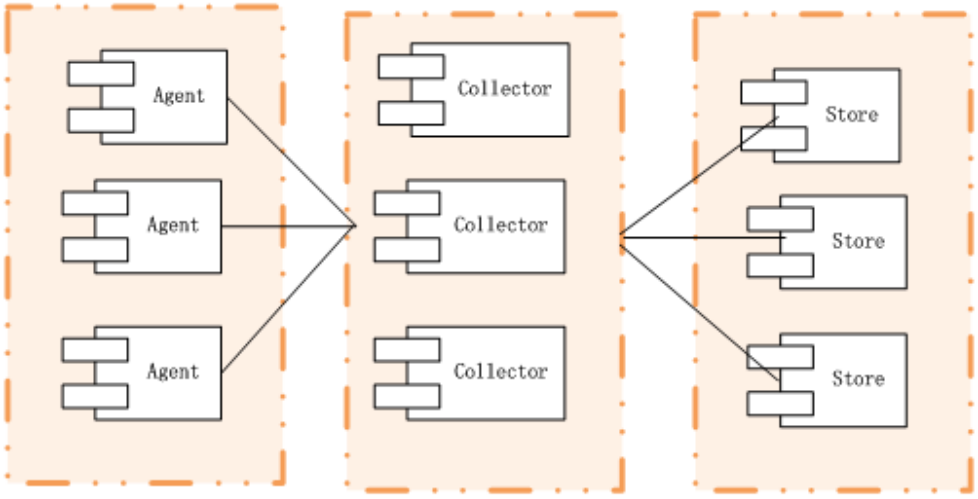

图 8-4 数据实时采集系统基本架构

8.2.2 数据实时计算

如图 8-5 所示，接收数据采集系统源源不断发来的实时数据后，流计算系统在流数据不断变化的运动过程中实时地进行分析，捕捉到可能对用户有用的信息，并把结果发送出去。数据实时计算与传统的数据操作的不同之处包括以下两个方面：

- 能对流数据做出实时回应；
- 用户是被动的，而 DBMS 是主动的。

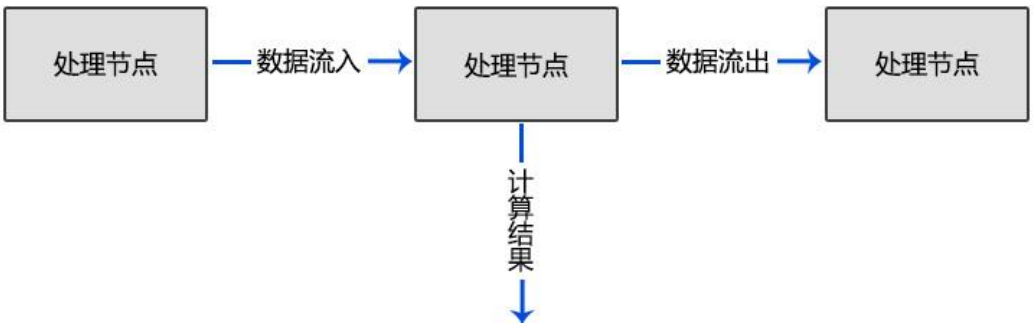

图 8-5 数据实时计算流程

实时数据经过处理节点后，产生的结果可作为另一个处理节点的输入数据，直至获取所需的计算结果。处理节点也可以将这些数据保存下来，以便下一阶段使用。

8.2.3 实时查询服务

理想情况下，流计算会将对用户有价值的结果实时推送给用户，这取决于应用场景。一

般而言，流计算的第三个阶段是实时查询服务，经由流计算框架得出的结果可供用户进行实时查询、展示或储存。

8.3 流计算的应用

流计算是针对流数据的实时计算，主要应用在产生大量流数据并对实时性要求很高的领域。

8.3.1 流计算的应用场景

如对于大型网站，活跃的流式数据非常普遍，这些数据包括网站的访问 PV（page view）/UV（unique visitor）、用户访问了什么内容、搜索了什么内容等。实时的数据计算和分析可以动态展示网站实时流量的变化情况，分析每天各小时的流量和用户分布情况，这对于大型网站来说具有重要的实际意义，不仅可用于网站的实时业务监控，也可以实现用户实时个性化内容推荐等。

流计算的应用场景有很多，总的来说，流计算一方面可应用于处理金融服务如股票交易、银行交易等产生的大量实时数据。另一方面流计算主要应用于各种实时 Web 服务中，如搜索引擎、购物网站的实时广告推荐，SNS 社交类网站的实时个性化内容推荐，大型网站、网店的实时用户访问情况分析等。

但从另一方面来说，并不是每个应用场景都需要实时流计算的。需要考虑是否对数据的实时性有迫切需求、是否更关注对当前数据的分析与响应。若处理动态流程（其特征更改得相当频繁）、非线性流程（其计时和顺序不可预测）和需要实时响应外部事件的流程，则流计算是最适合的。

8.3.2 流计算实例

（1）量子恒道

流计算的一大应用领域是分析系统。传统的分析系统都是分布式离线计算的方式，即将数据全部保存起来，然后每隔一定的时间进行离线分析，从而得出结果。但这样必然会导致一定的延时，这取决于离线计算的间隔时间和计算时长。特别是对于海量数据而言，即使是短时间内就可计算出结果，但离线计算间隔时间过长的话延时也相应增加。

但是，随着业务对实时性要求的提升，这样的模式已不太适合对于流数据的分析，也不太适用于需要实时响应的互联网应用场景。而通过流计算，能在秒级别内得到实时的分析结果，有利于根据当前所得到的分析结果及时地做出决策、调整。典型的搜索引擎、购物网站的广告推荐、社交网站的个性化推荐等，都是基于对用户行为的分析系统实现的。典型的代表还有网站访问数据的分析。接下来我们将以量子恒道的例子来说明流计算给分析系统带来的改变。

量子恒道是一家专业电子商务数据服务商，致力于为网商提供精准实时的数据统计、多维的数据分析、权威的数据解决方案。目前为超过百万的淘宝卖家提供数据统计分析服务。

随着用户和访问数据规模的不断增加，量子恒道也面临着巨大的挑战：实时计算处理数据超过 3T/日，分布式离线计算处理数据超过 20T/日。虽然分布式离线计算能满足大部分用户的需求，小时级的统计延时是可以接受的。但随着实时性要求的不断提升，特别是“双 11”、“双 12”这样需要实时数据分析支撑的应用场景，商家希望通过实时的网店访问情况来及时调整促销策略。如何实现秒级别的实时分析响应成为量子恒道的一大挑战。

网站访问数据是典型的流数据，针对流数据，量子恒道基于“Erlang（一种通用的面向并发的编程语言）+ZooKeeper（针对大型分布式系统的可靠协调系统）”开发了海量数据实时流计算框架 Super Mario 2.0。该流计算框架具有低延迟、高可靠性的特点。

与前面介绍的流计算的三个阶段相对应，Super Mario 2.0 的实时数据处理流程也可以用以下三个阶段来表示（如图 8-6 所示）：

- Log 数据由 TimeTunnel 在毫秒级别内实时送达；
- 实时数据经由 Super Mario 流计算框架进行处理；
- HBase 输出、存储结果。

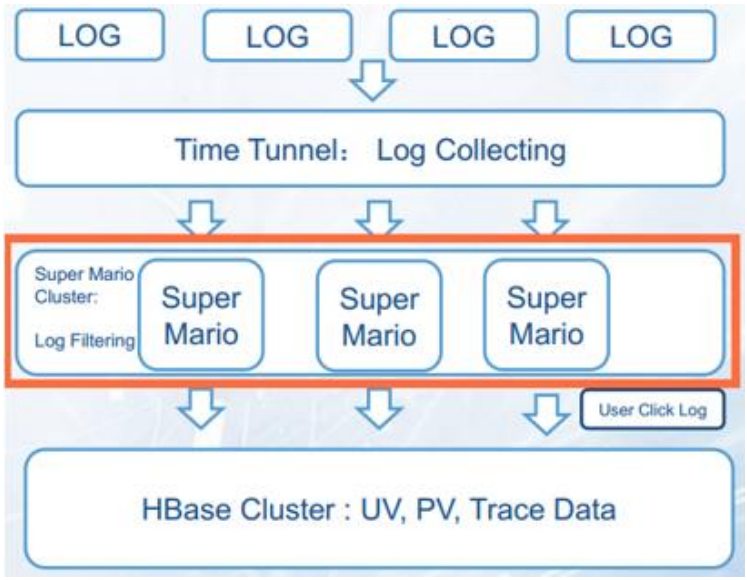

图 8-6 Super Mario 实时数据处理流程

通过 Super Mario 流计算框架，量子恒道可处理每天 TB 级的实时流数据，并且从用户发出请求到数据展示，整个延时控制在 2-3 秒内，达到了实时性的要求。

（2）IBM InfoSphere Streams

流计算不仅为互联网带来改变，也能改变我们的生活。IBM 的流计算平台 InfoSphere Streams（如图 8-7 所示），能够广泛应用于制造、零售、交通运输、金融证券以及监管各行各业的解决方案之中，使得实时快速做出决策的理念得以实现。以实时交通信息管理为例，Streams 应用于斯德哥尔摩的交通信息管理，通过结合来自不同源的实时数据，Streams 可以生成动态的、多方位的看待交通流量的方式，为城市规划者和乘客提供实时交通状况查看。

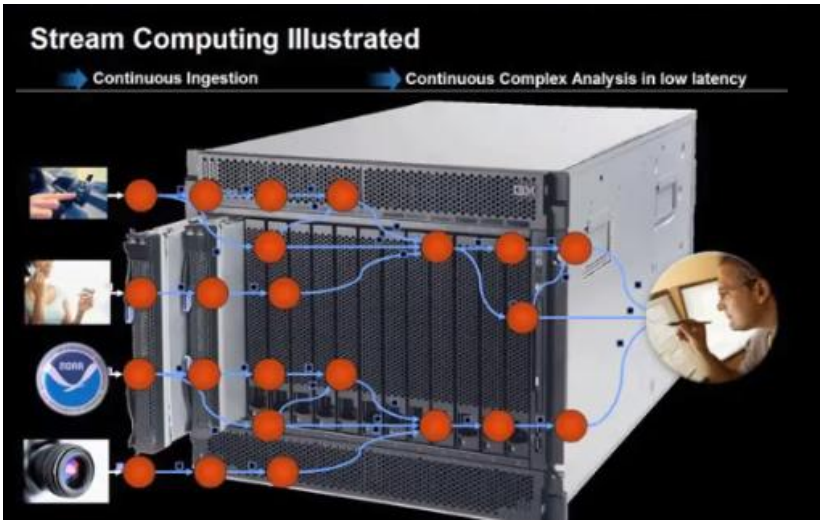

图 8-7 Streams 可汇总来自不同源的实时数据

通过实时流计算来分析交通信息是很有现实意义的。以提供导航路线为例，一般的导航路线计算并没有考虑交通状况，因为要处理如此庞大的实时信息就是一个极大的挑战。即便

计算路线时有考虑交通状况，往往也只是参考了以往的交通状况。而借助于实时流计算，不仅可以根据交通情况制定路线，而且在行驶过程中，也可以根据交通情况的变化实时更新路线，始终为用户提供最佳的行驶路线，如图 8-8 所示。

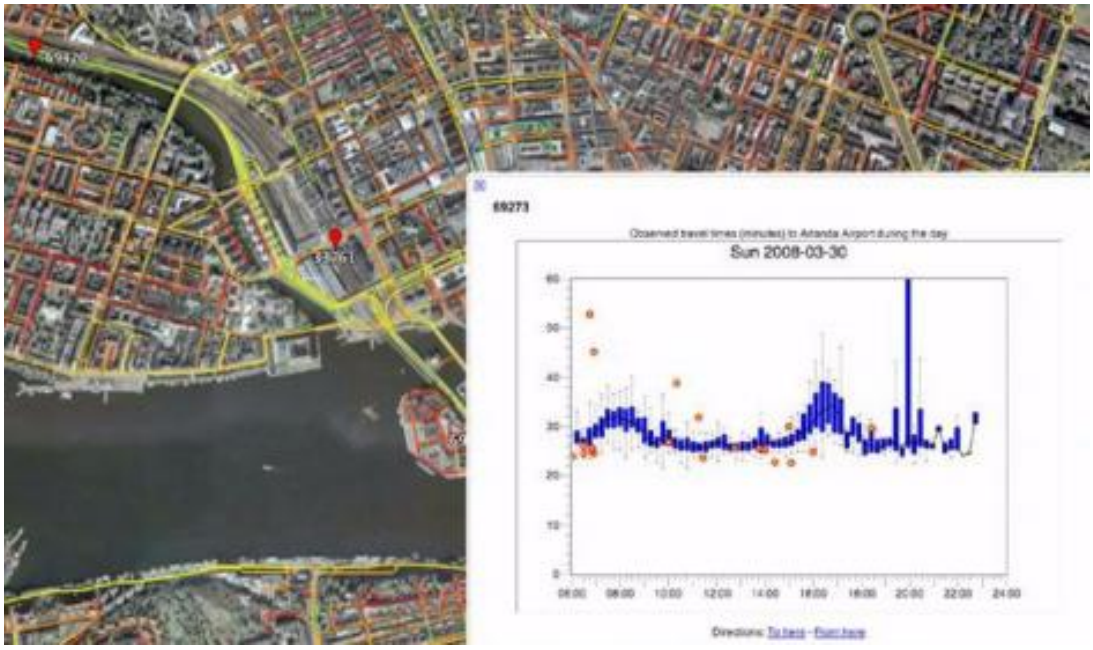

图 8-8 通过 Streams 分析实时交通信息

8.4 流计算框架 Storm

自从数据流出现以来，就有分析数据流并从中获取有用信息的需求。但是，直到几年前，仍然只有那些非常大的银行和政府机构能够通过昂贵的定制系统满足这种计算需求，如 IBM 推出的商业流计算系统 InfoSphere Streams，在政府部分与金融机构得以使用。

早在 InfoSphere Streams 出现之前，就有许多流计算技术的学术研究，如 Aurora，它是 MIT 等三所大学合作完成的项目。后来在 Aurora 的基础上开发了流式系统 Borealis，但该项目在 08 年已经停止维护。

流数据一般出现在金融行业或者互联网流量监控的业务场景，由于这些场景中数据库应用占据主导地位，因而造成了早期对于流数据研究多是基于对传统数据库处理的流式化，而对流式框架本身的研究则偏少，当时的工业界把更多的精力转向了实时数据库。

2010 年 Yahoo!开发的分布式流式处理系统 S4（Simple Scalable Streaming System）的开源，以及 2011 年 Twitter 开发的 Storm 的开源，改变了这个情况。S4 和 Storm 相比 Hadoop 而言，在流数据处理上更具优势。MapReduce 系统主要解决的是对静态数据的批量处理，

即当前的 MapReduce 系统实现启动计算时，一般数据已经到位。而流式计算系统在启动时，一般数据并没有完全到位，而是源源不断地流入。批处理系统一般重视数据处理的总吞吐量，而流处理系统则更加关注数据处理的延时，即希望流入的数据越快处理越好。

以往开发人员在做一个实时应用的时候，除了要关注应用逻辑计算处理本身，还要为了数据的实时传输、交互、分布大伤脑筋，但是，现在情况却大为不同。以 Storm 为例，开发人员可以快速地搭建一套健壮、易用的实时流处理框架，配合 SQL 产品或者 NoSQL 产品或者 MapReduce 计算平台，就可以低成本地做出很多以前很难想象的实时产品。

Yahoo! S4 与 Twitter Storm 是目前流行的开源流计算框架，各有其架构特点，相对而言，Storm 更为优秀。我们在此就以 Storm 为研究学习对象，学习其设计理念与架构特点。

8.4.1Storm 简介

Twitter Storm 是一个免费、开源的分布式实时计算系统，它可以简单、高效、可靠地处理大量的流数据。Storm 对于实时计算的意义类似于 Hadoop 对于批处理的意义，这一说法也得到了业内人士的认同。

Storm 是基于 Clojure 和 Java 开发的，可以访问其官方网站 http://storm-project.net/或 Github 项目主页 https://github.com/nathanmarz/storm 了解其更多信息。

Twitter 开发这样一款系统也是为了应对其不断增长的数据和实时处理需求。为了处理最近的数据，需要一个实时系统和批处理系统同时运行。当要计算一个查询时，需要查询批处理视图和实时视图，并把它们合并起来以得到最终的结果。

在 Twitter 中进行实时计算的系统就是 Storm，它在数据流上进行持续计算，并且对这种流式数据处理提供了有力保障。

同时，Twitter 采用分层的数据处理架构（如图 8-9 所示），由 Hadoop 和 ElephantDB（专门用于从 Hadoop 中导出 key/value 数据的数据库）组成批处理系统，Storm 和 Cassandra（混合型的非关系的数据库）组成实时系统，实时系统处理的结果最终会由批处理系统来修正，正是这个观点使得 Storm 的设计与众不同。

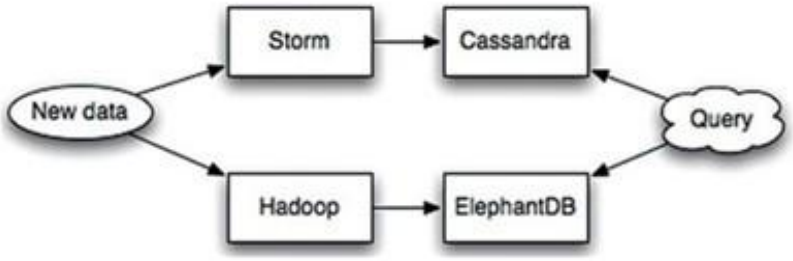

图 8-9 Twitter 数据系统分层处理架构

8.4.2Storm 主要特点

Storm 的主要特点如下：

- 简单的编程模型：Storm 降低了进行实时处理的复杂性；
- 支持各种编程语言：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 会自动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；
- 快速：系统的设计保证了消息能得到快速的处理；
- 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，这样可以快速进行开发和单元测试；
- 容易部署：Storm 集群易于部署，只需少量的安装和配置就可以运行。

Storm 的这些特点，特别是能可靠地处理消息，保证每条消息都能得到处理的特点，使其在目前的流计算应用中得到了广泛的使用。此外，Storm 支持本地模式，在单机上就可以进行安装、使用，大大降低了学习成本。

8.4.3Storm 应用领域

Twitter 列举了 Storm 的三大应用领域：

- 信息流处理（Stream Processing）：Storm 可以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 可以进行连续查询并把结果即时反馈给

客户，比如将 Twitter 上的热门话题发送到客户端；

- 分布式远程过程调用（Distributed RPC）：Storm 可以用来并行处理密集查询，Storm 的拓扑结构（后文会介绍）是一个等待调用信息的分布函数，当它收到一条调用信息后，会对查询进行计算，并返回查询结果。

除了这些领域，Storm 也可以应用于各类实时计算的应用场景。

8.4.4Storm 设计思想

Storm 对一些概念进行了抽象化，其主要术语和概念包括 Streams、Spouts、Bolts、Topology 和 Stream Groupings。

（1） Streams

如图 8-10 所示，在 Storm 对流 Stream 的抽象描述中，流是一个不间断的无界的连续 Tuple（元组，是元素有序列表）。这些无界的元组会以分布式的方式并行地创建和处理。

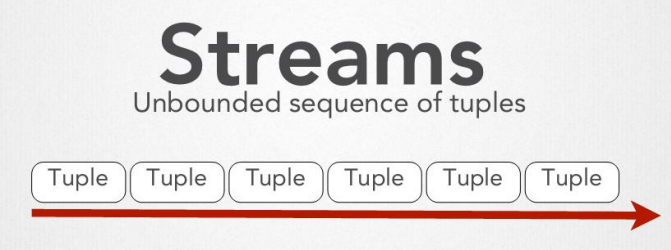

图 8-10 Streams:无界的 Tuples 序列

（2） Spouts

Storm 认为每个 Stream 都有一个源头，它将这个源头抽象为 Spouts。Spouts 会从外部读取流数据并发出 Tuple，如图 8-11 所示。

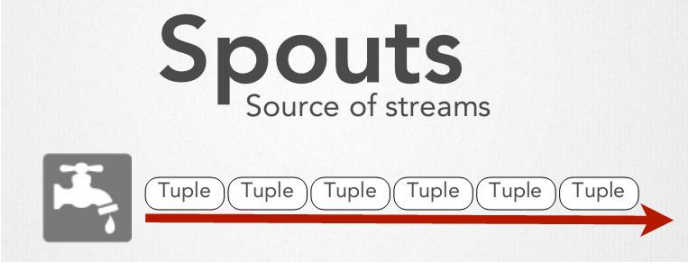

图 8-6 Spouts 数据源

（3） Bolts

如图 8-12 所示，Storm 将流的中间状态转换抽象为 Bolts，Bolts 可以处理 tuples，同时它也可以发送新的流给其他 Bolts 使用。Bolts 作为消息处理者，所有的消息处理逻辑被封装

在 Bolts 里面，处理输入的数据流并产生输出的新数据流。Bolts 中可执行过滤、聚合、查询数据库等操作。

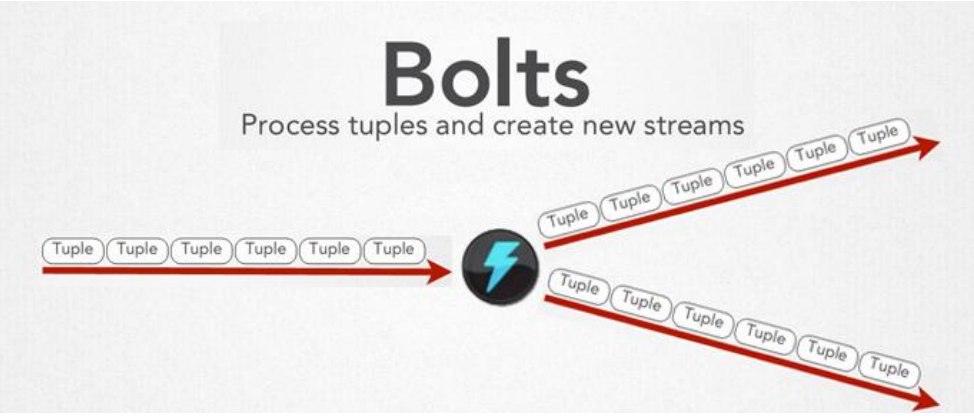

图 8-7 Bolts: 处理 tuples 并产生新的数据流

（4） Topology

为了提高效率，在 Spout 源可以接上多个 Bolts 处理器。Storm 将这样的无向环图抽象为 Topology，如图 8-13 所示。Topology 是 Storm 中最高层次的抽象概念，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 上进行处理。

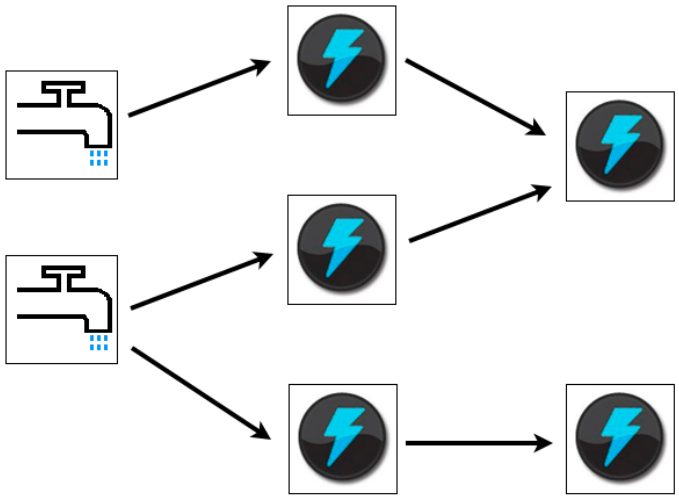

图 8-8 Topology 示意图

在 Topology 的实现上，Storm 中拓扑定义仅仅是一些 Thrift 结构体（Thrift 是基于二进制的高性能的通讯中间件），这样一来就可以使用其他语言来创建和提交拓扑。

Stream 中的每一个 Tuple 就是一个值列表。列表中的每个值都有一个名称，并且该值可以是基本类型、字符类型、字节数组等，当然也可以是其他可序列化的类型。

Topology 中的每个节点都要说明它所发射出的元组的字段的名称，这样其他节点只需

要订阅该名称就可以接收处理。

（5） Stream Groupings

消息分发策略，即定义一个 Stream 应该如何分配给 Bolts。目前 Stream Groupings 有如下几种方式：

- Shuffle Grouping：随机分组，随机分发 Stream 中的 Tuple；
- Fields Grouping：按字段分组，具有相同值的 Tuple 会被分发到对应的 Bolts；
- All Grouping：广播分发，每个 Tuple 都会被分发到各个 Bolts 中；
- Global Grouping：全局分组，Tuple 只会分发给 Bolt 中的一个任务；
- Non Grouping：不分组，与随机分组效果类似；
- Direct Grouping：直接分组，由 Tuple 的生产者来定义接收者。

通过这些消息分发策略，Storm 解决了两个组件（Spout 和 Bolt）之间如何发送 Tuple 的问题。

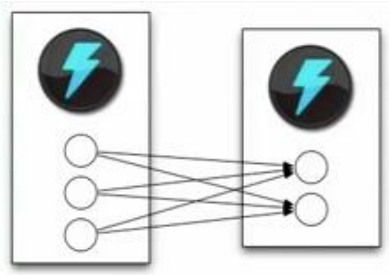

图 8-9 Stream grouping 示意图

图 8-14 中的箭头表示 Tuple 的流向，而圆圈则表示 Task，Task 就是具体的处理逻辑，每一个 Spout 和 Bolt 会被当作很多 Task 在整个集群里面执行，并且每一个 Task 对应到一个线程。通过一个完整的 Topology 示意图（如图 8-15 所示），可以了解 Stream Grouping 和 Task 在当中的作用。

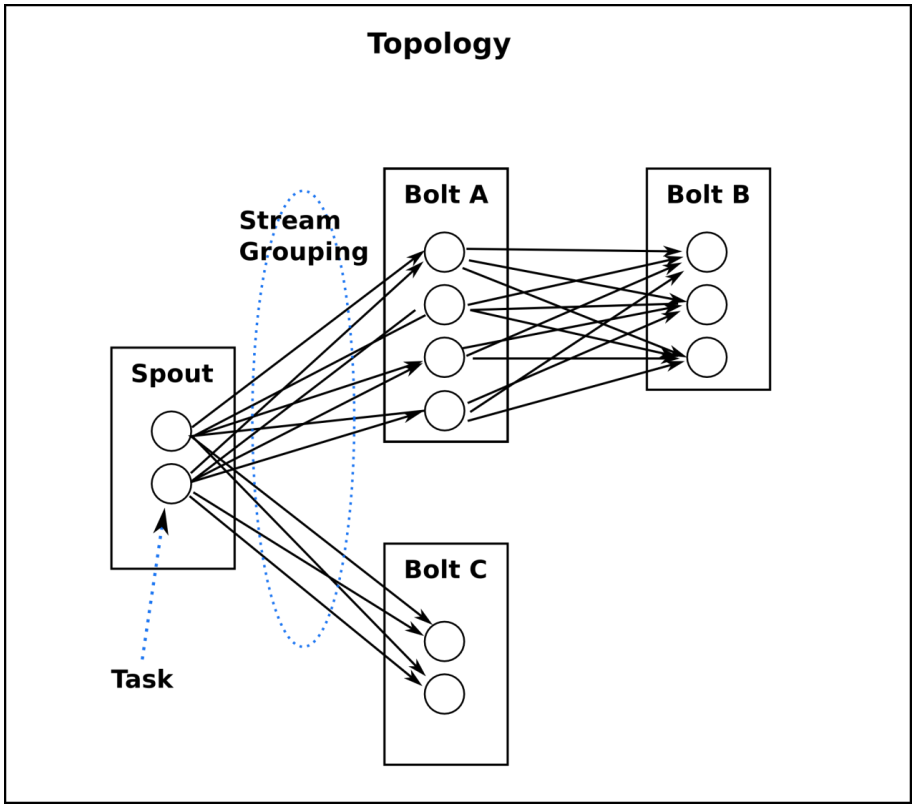

图 8-10 Topology 完整示意图

8.4.5Storm 框架设计

Storm 运行于集群之上，与 Hadoop 集群类似。但在 Hadoop 上运行的是“MapReduce Jobs”，而在 Storm 上运行的是“Topologies”。两者大不相同，一个关键不同是一个 MapReduce 的 Job 最终会结束，而一个 Topology 永远处理消息（或直到 kill 它）。

Storm 集群有两种节点：控制（Master）节点和工作者（Worker）节点。

Master 节点运行一个称之为“Nimbus”的后台程序，负责在集群范围内分发代码、为 worker 分配任务和故障监测。

每个 Worker 节点运行一个称之“Supervisor”的后台程序，监听分配给它所在机器的工作，基于 Nimbus 分配给它的事情来决定启动或停止工作者进程。

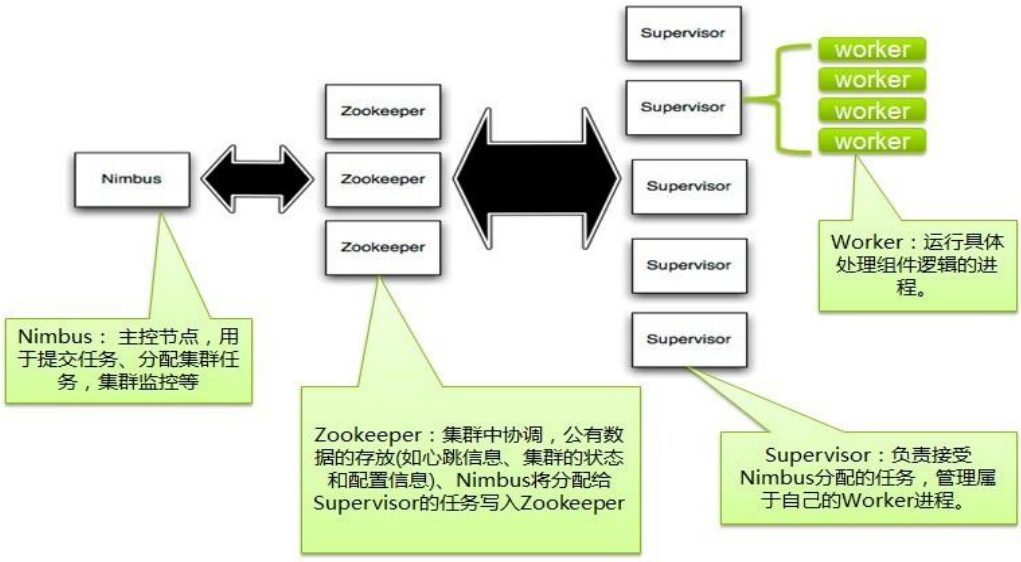

图 8-11 Storm 集群架构示意图

从图 8-16 可以看出，Storm 采用了 Zookeeper 来作为分布式协调组件，一个 Zookeeper 集群负责 Nimbus 和多个 Supervisor 之间的所有协调工作（一个完整的拓扑可能被分为多个子拓扑，并由多个 supervisor 完成）。

Nimbus 后台程序和 Supervisor 后台程序都是快速失败（fail-fast）和无状态的，所有状态维持在 Zookeeper 或本地磁盘中。

在这种设计中，master 节点并没有直接和 worker 节点通信，而是借助中介 Zookeeper，这样一来可以分离 master 和 worker 的依赖，将状态信息存放在 Zookeeper 集群内以快速回复任何失败的一方。

这意味着你可以 kill 杀掉 nimbus 进程和 supervisor 进程，然后重启，它们将恢复状态并继续工作，这种设计使得 Storm 极其稳定。

再来看看 Storm 的工作流程（如图 8-17 所示）：

- 首先定义 Topology，由客户端提交 Topology 到 Storm 中执行；
- Nimbus 建立 Topology 本地目录，将 Topology 分配到集群中进行处理（将分配给 Supervisor 的任务写入 Zookeeper 中）；
- Supervisor 从 Zookeeper 中获取所分配的任务，启动任务；
- Worker 节点中的 Task 执行具体的任务逻辑。

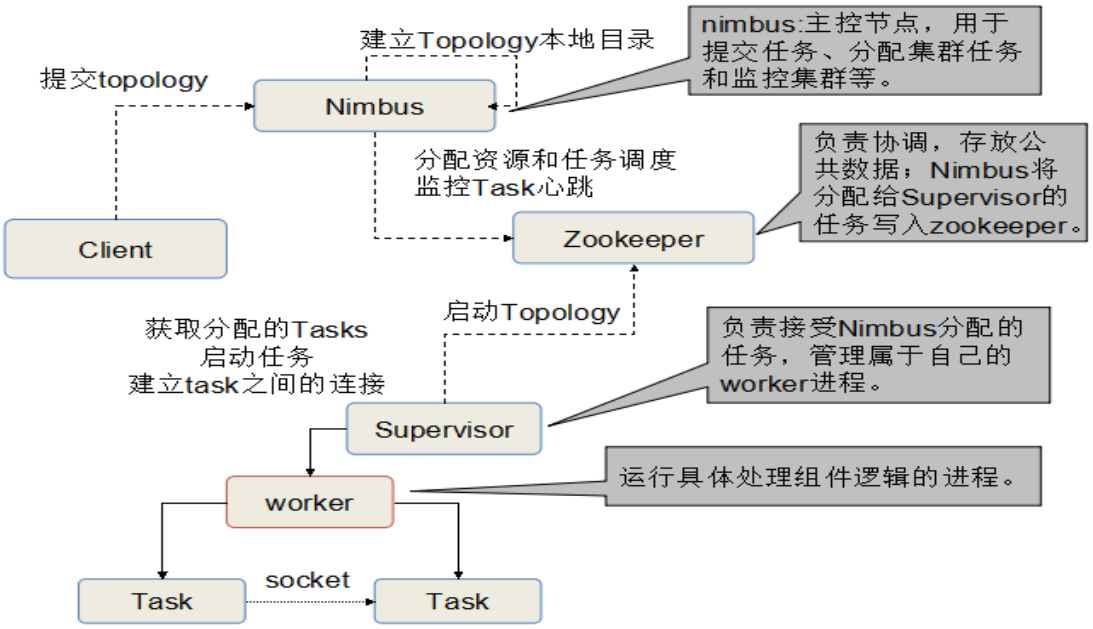

图 8-12 Storm 工作流程示意图

总的来说，流计算任务的整体逻辑在 Topology 中定义，然后便可提交到 Storm 中执行。

8.4.6Storm 实例

了解了 Storm 的设计思想和框架设计，下面以一个单词统计的实例来加深对 Topology 的认识。

Storm 编程模型非常简单，如下 Topology 代码即定义了整个单词统计的逻辑：

```
TopologyBuilder builder = new TopologyBuilder();

builder.setSpout("sentences", new RandomSentenceSpout(), 5);
builder.setBolt("split", new SplitSentence(), 8)
        .shuffleGrouping("sentences");
builder.setBolt("count", new WordCount(), 12)
        .fieldsGrouping("split", new Fields("word"));
```

图 8-13 单词统计 Topology 代码

代码中第一行新建了一个 Topology builder。

Builder.setSpout 是对 Spout 数据源的定义，方法中有三个参数，第一个参数定义 Spout 来源为“sentences”，表明要统计单词的来源；第二个参数定义 Spout 数据源的处理函数；参数三则定义了并发线程数。

紧接着代码包含两个 builder.setBolt 定义，同样有三个参数，并且每个 setBolt 同时定义

了消息分发策略。第一个 setBolt 定义了单词的分割，即从句子中提取出单词，并以随机分发的方式将 Tuple 分发给每个 Bolt。而第二个 setBolt 则定义了对这些分割后单词的处理，即计数，分发方式为“按字段分组”，只有具有相同 field 值的 Tuple 才会发给同一个 Task 进行统计，保证了统计的准确性。

从代码中也可以看出，Bolts 是通过订阅 Tuple 的名称来接收相应的数据，如第二个 setBolt 订阅了前一个 setBolt 分割后的单词数据。

Topology 中只是定义了整个计算逻辑，具体的处理函数则可以使用多种语言来完成。如 SplitSentence 方法中，代码 super(“python”, “splitsentence.py”)说明这个方法是使用 Python 语言来实现的。

```
public static class SplitSentence extends ShellBolt implements IRichBolt {

    public SplitSentence() {
        super("python", "splitsentence.py");
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("word"));
    }

    @Override
    public Map<String, Object> getComponentConfiguration() {
        return null;
    }
}
```

图 8-19 SplitSentence 类定义

方法中调用了 splitsentence.py 脚本（如图 8-20 所示），该脚本定义了一个简单的单词分割方法，即通过空格来分割单词。当然真正的单词分割逻辑没有这么简单，这里仅是通过这个简单的实例代码来快速了解其实现原理。分割后的单词通过 emit 的方法将 Tuple 发射出去，以便订阅了该 Tuple 的 Bolts 进行接收。

```
import storm

class SplitSentenceBolt(storm.BasicBolt):
    def process(self, tup):
        words = tup.values[0].split(" ")
        for word in words:
          storm.emit([word])
```

图 8-14 splitsentence.py 脚本

SplitSentence 类中的 declareOutputFields 方法定义了要输出的字段。进行”count”操作的 Bolts 接收其订阅的 Tuple 后，调用 WordCount 类来进行下一步的处理，如图 8-21 所示。

```
public static class WordCount extends BaseBasicBolt {
    Map<String, Integer> counts = new HashMap<String, Integer>();

    @Override
    public void execute(Tuple tuple, BasicOutputCollector collector) {
        String word = tuple.getString(0);
        Integer count = counts.get(word);
        if(count==null) count = 0;
        count++;
        counts.put(word, count);
        collector.emit(new Values(word, count));
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("word", "count"));
    }
}
```

图 8-15 WordCout 类定义

图 8-21 的类定义中的 execute 方法说明了单词统计的逻辑，即单词若已统计过，则计数加 1，否则置为 0。同时 declareOutputFields 方法定义了最终的输出字段：(“word”, “count”)。

下图表示一个句子经过上面单词统计流程后的统计结果图。

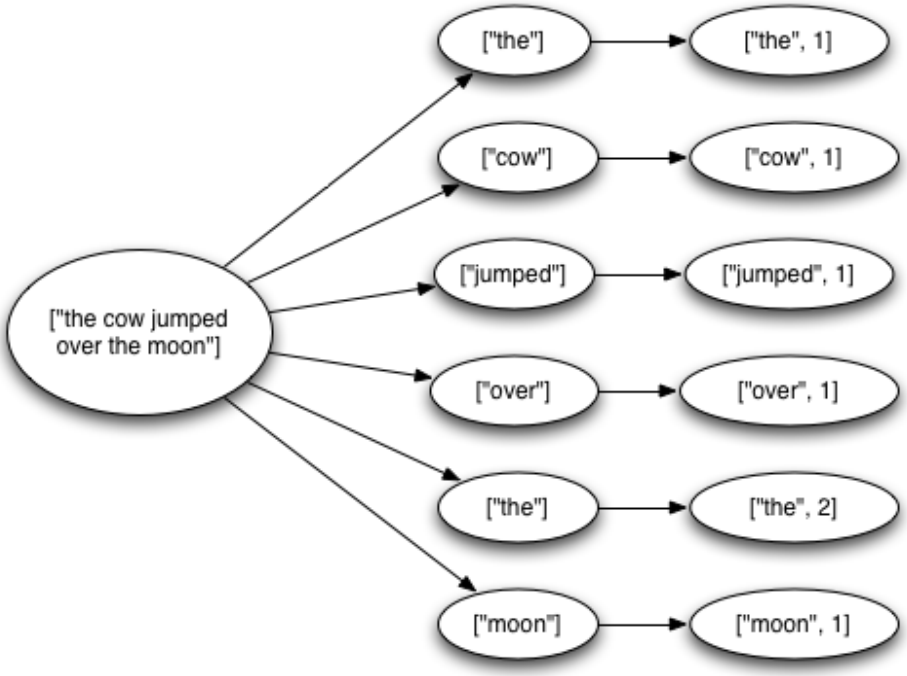

图 8-22 一个句子经单词统计后的统计结果示意图

现在让我们再来总结一下整个流程：

- 每个从 Spout 发送出来的消息（英文句子）都会触发很多的 Task 被创建；
- 用于分割单词的 Bolts 将句子分解为独立的单词，然后发射包含这些单词的 Tuple；

- 用于计数的 Bolts 接收 Tuple，并对其进行统计；
- 最后，实时的输出每个单词以及它出现过的次数。

这虽然是一个简单的单词统计，但对其进行扩展，便可应用在许多场景中，如微博中的实时热门话题。Twitter 也正是使用了 Storm 来实现这一功能。

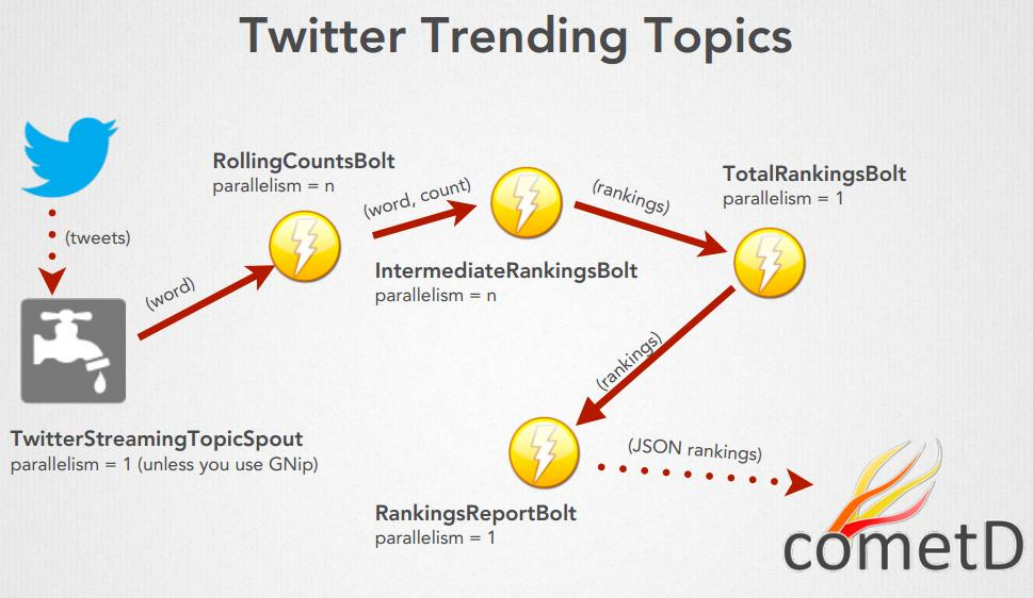

图 8-163 Twitter 实时热门话题处理流程示意图

Twitter 实时热门话题的处理流程与单词统计流程是相近的（如图 8-173 所示），不过 Twitter 实时热门话题使用了更多级的 Bolts。首先，以用户发布的 tweets 作为数据源，经过 TwitterStreamingTopicSpout 处理后，分发给 RollingCountsBolt（用于实现滑动窗口计数和 Top N 排序，网上有文章介绍了 Twitter 的这一 Rolling Count 算法：http://www.michael-noll.com/blog/2013/01/18/implementing-real-time-trending-topics-in-storm/ ）；然后，发出的（word，count）Tuple 再经过 IntermediateRankingsBolt 进行排序；最后，由 TotalRankingBolt 进行汇总得出总的热门话题排序。排序结果再交给 RankingsReportBolt 进行最后的处理，如进行筛选、提取、输出等。图 8-183 中的 cometD 是一项 Ajax 推送技术，表示处理后的排序结果以 JSON 的格式，结合 JavaScript 前端技术实时推送给 Web 前端展示给用户。

8.4.7 哪些公司在使用 Storm

Storm 自 2011 年发布以来，凭借其优良的实时流计算框架设计及开源特性，如今已经吸引了许多大型互联网公司的注意，并将其应用到了自身的实际项目中。下图展示了部分使

用 Storm 的公司和项目，典型用户如淘宝和阿里巴巴。

图 8-194 使用 Storm 的公司和项目

淘宝和阿里巴巴许多业务都需要实时流计算的支撑，如业务监控、广告推荐、买家实时数据分析等业务场景。淘宝数据部门开发的新架构已把 Storm 作为当中重要的一部分（如图 8-25 所示）。

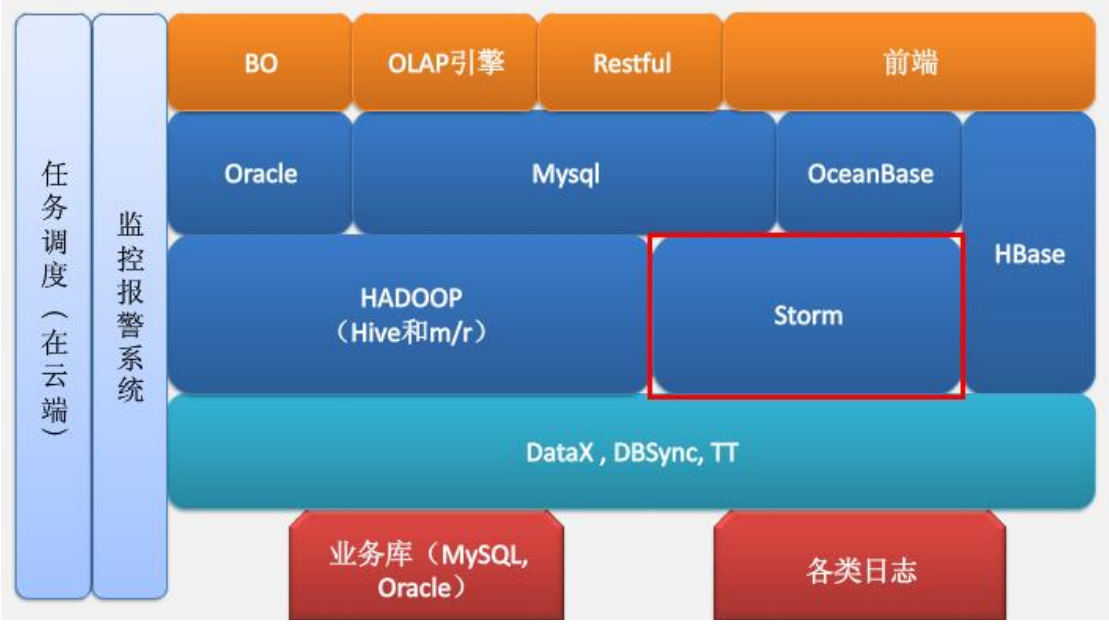

图 8-20 淘宝数据部门新架构示意图

8.4.8流计算框架汇总

目前业内已涌现出许多的流计算框架与平台，在此做一个小小的汇总。

第一类是商业级的流计算平台，代表如下：

- IBM InfoSphere Streams：商业级高级计算平台，帮助用户开发的应用程序快速摄取、分析和关联来自数千个实时源的信息 http://www-03.ibm.com/software/products/cn/zh/infosphere-streams/；
- IBM StreamBase：IBM 开发的另一款商业流计算系统，在金融部门和政府部门使用 http://www.streambase.com/。

第二类是开源流计算框架，代表如下：

- Twitter Storm：免费、开源的分布式实时计算系统，可简单、高效、可靠地处理大量的流数据 http://storm-project.net/ ；
- Yahoo! S4（Simple Scalable Streaming System）：开源流计算平台，是通用的、分布式的、可扩展的、分区容错的、可插拔的流式系统。http://incubator.apache.org/s4/；

第三类是公司为支持自身业务开发的流计算框架，虽然未开源，但有不少的学习资料可供了解、学习，代表如下：

- Facebook Puma：Facebook 使用 Puma 和 HBase 相结合来处理实时数据；
- DStream：百度正在开发的属于百度的通用实时数据流计算系统；
- 银河流数据处理平台：淘宝开发的通用流数据实时计算系统；
- Super Mario：基于 Erlang 语言和 Zookeeper 模块开发的高性能数据流处理框架。

此外，业界也涌现出了像 SQLstream 这样专门致力于实时大数据流处理服务的公司。

本章小结

本章首先介绍了什么是流计算，介绍了流计算产生的背景与流计算的基本概念，接着介绍了流计算的处理模型与处理流程，分析了 Hadoop 为代表的批处理能否胜任流计算的工作。接着对流计算的应用场景做了总结，并通过具体的实例来说明当前流计算框架的重要性。接下来，着重介绍了目前流行的开源流计算框架 Twitter Storm，包括其主要特点、应用领域、设计思想和框架设计，并且通过一个简单的实例来加深对 Storm 的认知。最后对当前流计算框架做了一个小小的汇总。

参考文献

[1] Beyond MapReduce：谈 2011 年风靡的数据流计算系统 http://www.programmer.com.cn/9642/

[2] 对互联网海量数据实时计算的理解 http://www.cnblogs.com/panfeng412/archive/2011/10/28/2227195.html

[3] Storm - As deep into real-time data processing as you can get in 30 minutes. http://www.slideshare.net/DanLynn1/storm-as-deep-into-realtime-data-processing-as-you-can-get-in-30-minutes

[4] Storm 实时流计算 http://wenku.baidu.com/view/7b24d0d49e3143323968937d.html

实时流式数据处理及应用 http://365cy.gotoip1.com/wp-content/uploads/2013/06/05xuzhengjun.pdf

附录 1:任课教师介绍

林子雨(1978－),男,博士,厦门大学计算机科学系助理教授,主要研究领域为数据库,数据仓库,数据挖掘.
主讲课程：《大数据技术基础》
办公地点：厦门大学海韵园科研 2 号楼
E-mail: ziyulin@xmu.edu.cn
个人网页：http://www.cs.xmu.edu.cn/linziyu