# Learning Apache Flink

## 1. Introduction to Apache Flink

随着分布式技术的不断发展，工程师们正试图将这些技术推向极限。早期，人们寻找更快、更廉价的方式处理数据。因此Hadoop被提出来了。人们使用Hadoop，并使用Hadoop生态系统作为他们的ETL工具。现在许多公司已经开始使用Hadoop，另外一些人需要以流式方式处理数据，因此刀子一些像Spark和Flink这些技术出现。因其有更快的处理数据的能力，快速的扩展性，支持机器学习和图计算的特点，在开发社区已经非常受欢迎。

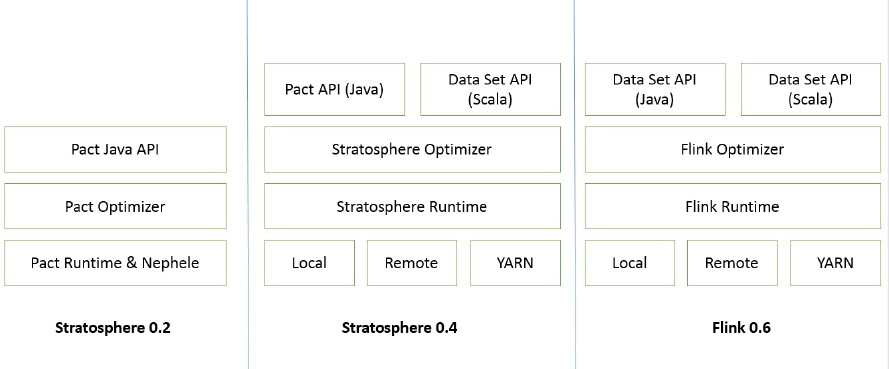
有些人在他们的日常生活中已经使用Spark了，或许他们想知道为什么我们还需要Flink？这种想法人们很自然的被人提出来。让我来用简洁的话来回到这个问题。第一点是Flink是真正的流处理引擎，它不是像Spark流处理引擎那样以更快的速度处理微批量数据。它把微批量数据的处理作为流式处理的一种特殊情况。同样通过本书我们会发现许多的不同。

这本书是关于更先进的技术中的Flink。在本章，我们将讨论一下主题：

* 历史
* 结构
* 分布式执行
* 特点
* 快速开始配置
* 集群配置
* 运行一个简单的应用程序

### History

Flink起源于柏林地区的大学研究的一个名叫Stratosphere项目，该项目的主要目标是建立下一代大数据分析平台。在2016.4.16日作为Apache旗下一个孵化项目。Stratosphere初始版本主要依据于Nephele的<http://stratosphere.eu/assets/papers/Nephele_09.pdf>论文。以下图展示了Stratosphere随着时间是怎么进化的。

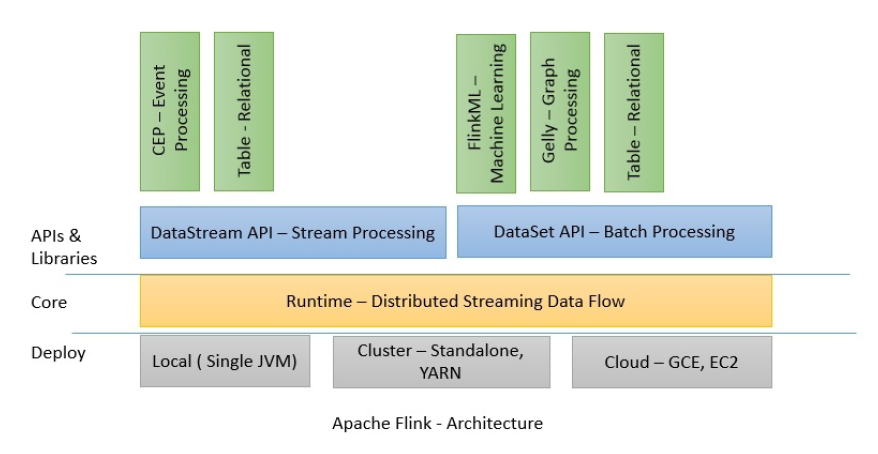


Stratosphere 最开始的版本主要包括运行时环境，优化器以及Java API。之后，随着该平台的不断成熟，它支持运行在本地和yarn上，从0.6版本以后，Stratosphere 被重新命名为Flink。之后的Flink版本不断聚焦于批量处理、流式处理、图计算、机器学习等。

Flink在0.7版本，一个非常重要的特点-streaming API被启动。Flink以前的版本仅仅支持Java API，在之后的版本也支持Scala API。让我们在下一节看看现在Flink的结构。

### Architecture

Flink 1.x 之后主要的结构由部署、主要的处理引擎以及APIs组成。我们可以很轻松的比较最后的版本和Stratosphere的结构来发现他的进化。以下图展示了组件，APIs和相关的库：



Flink具有层次结构，每一个组件都是特殊层中的一部分。每一层都是简历在其它层之上的抽象。Flink可以运行在本机、Yarn集群或者云上。运行时环境是Flink主要的数据处理引擎。它将通过API别写的程序转换成JobGraph。JobGraph是一个并行化的数据流，包含一些生产和消费数据的流任务

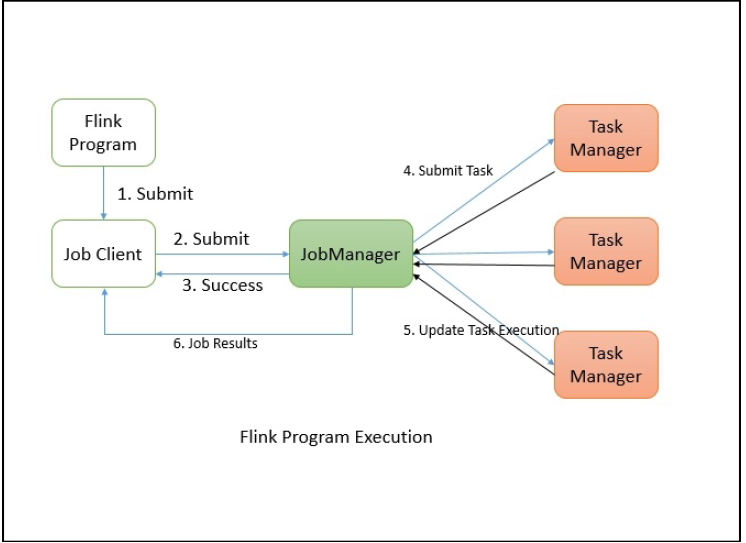
编程人员利用DataStream和DataSet APIs可以定义Job。JobGraph是在编写完成的程序中生成的。一旦JogGraph生成，DataSet API通过优化器生成优化的执行计划与此同时DataStream API通过流建立一个有效的执行计划。

这些优化的JobGraph然后会被提交到部署模式中的执行者。你可以选择部署在本地、远程或者Yarn集群。如果你已经拥有一个运行的Hadoop集群，最好使用Yarn来进行部署。

#### Distributed execution

Flink分布式执行主要由两个非常重要的进程组成：master和worker。当一个Flink程序在执行的时候，许多的进程参与执行，它们是Job Manager，Task Manager，Job Client。

以下图展示了一个Flink程序的执行：



Flink程序需要被提交打Job Client。Job Clinet将该作业提交给Job Manager，Job Manager的主要责任是负责资源的分配和作业的执行，最主要的事情是分配需要的资源，一旦资源被分配，task将被分配到各自的Task Manager。当接收到task，Task Manager 初始化一个进程去执行。在进程执行的过程中，Task Manager 不断汇报状态（启动，执行，完成）给Job Manager，一旦job执行完成，最后的结果将会返回给client。

#### Job Manager

master进程又名Job Manager，协调和管理程序的执行。它主要的职责包括作业的调度，检查点的管理，容错回复等等。

并行化环境下有许多master，他们共享他们的职责，以此来实现高可用。只有一个master是主，如果主master宕机了，会从其他的standby master节点选取一个作为leader master。

Job Manager由以下几个主要的组件组成：

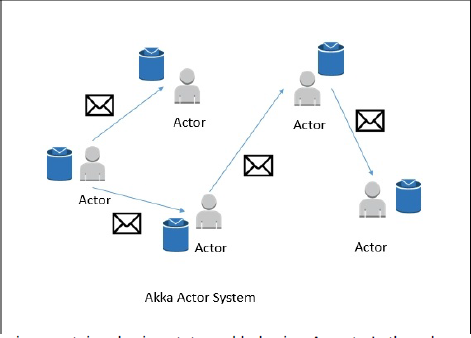
* Actor System
* Scheduler
* Check pointing

Flink内部使用Akka actor system 在job manager和task managers之间进行通讯。

##### Actor System

一个Actor System 是一个由不同角色的actors组成的容器。它提供例如调度，配置，日志等服务。它主要包含一个线程池，所有的actors都是从该线程池中进行初始化的。所有的actors都处于同一个层级结构。每一个新的actor都会给其分配一个parent。所有的actors通过消息系统进行交流。每一个actor都有自己的邮箱，它们通过该邮箱来读取信息。如果actors是local模式，那么消息将会在内存中进行共享，如果是远程模式，那么消息将会通过RPC进行调用。

每一个parent都需要对其所有的孩子进行监督。如果其中一个孩子出现问题，parent将会知道。如果孩子可以自己处理，那么parent将会重启该孩子，如果该孩子不能处理它的问题，那么它将会把该问题提升给自己的parent。



在Flink内部，一个actor是一个容器，它拥有自己的状态和行为。如果邮箱收到消息，他们他们将会不断的处理消息。它们的状态和行为是由他们所收到的消息决定的。

##### Scheduler

Flink中的执行者被定义为task slots。每一个task Manager需要管理一个或者多个task slots。在内部，Flink通过SlotSharingGroup和ColocationGroup决定哪些任务需要共享他们的task slot，哪些任务需要放置在特定的task slots。

##### Check point

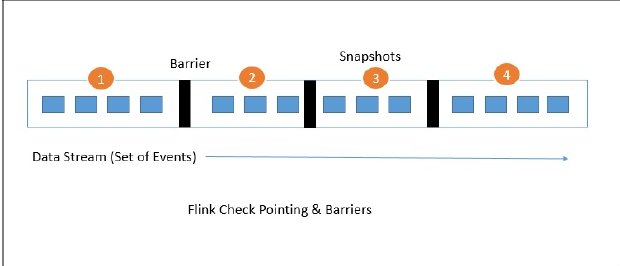
Flink通过check point来主要实现容错。它始终为分布式流数据和执行状态提供一致性快照。它受Chandy-Lamport算法的启发，但是已经根据Flink的需求进行了修改。更多关旭Chandy-Lamport算法间：<http://research.microsoft.com/enus/um/people/lamport/pubs/chandy.pdf>。

更精确的关于快照的实现基于以下文章：轻量级分布式数据流异步快照（<http://arxiv.org/abs/1506.08603>）。

容错机制不断地为数据流创造轻量级的快照。因此他们能够不断工作而不会过度负载。一般数据流的状态被保存在配置指明的地方，例如HDFS等地方。

在发生任何错误的情况下，Flink停止executors并重新启动它们，它们会从最后的检查点进行开始执行。

Stream barriers 是Flink 快照的核心元素，它们被摄入数据但不影响流浪。Barriers重来不会超越记录，它们将一组记录合并成一个快照。每一个barrier都有一个唯一的ID。以下图展示了barriers怎么被嵌入到数据流中进行快照的：

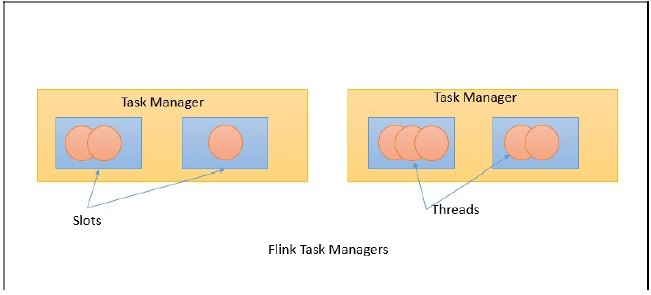


每一个快照状态都会向job Manager的检查点协调器进行汇报。当绘制快照时，Flink都会进行记录的校验以防止由于任何错误导致的数据重复处理。即使毫秒级的延迟是不可接受的，我们也可以选择在单个记录处理中选择低延迟。默认情况下，Flink只处理一次记录。 如果任何应用程序需要低延迟并且至少有一次处理就可以，我们可以关闭该触发器。 这将跳过校验并提高延迟。

#### Task Manager

Task managers是一些工作节点，它们在JVM中使用一个或者多个线程来执行任务。

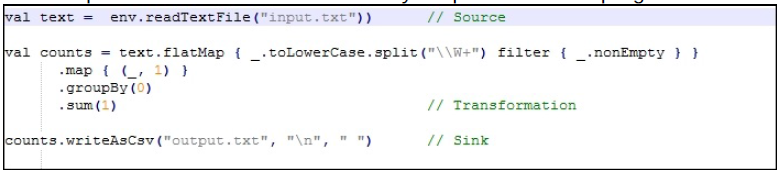
并行化的任务执行主要由每一个task manager中可用的task slots决定。每一个任务代表了被分配资源的集合到task slot上的映射。例如。如果一个task manager有4个slots，它们每一个slot将会得到task manager 内存的25%。每一个task slot可能有一个或者多个线程。在同一个slot中的线程之间共享他们的JVM。在同一个JVM中的task共享他们的TCP链接和心跳。



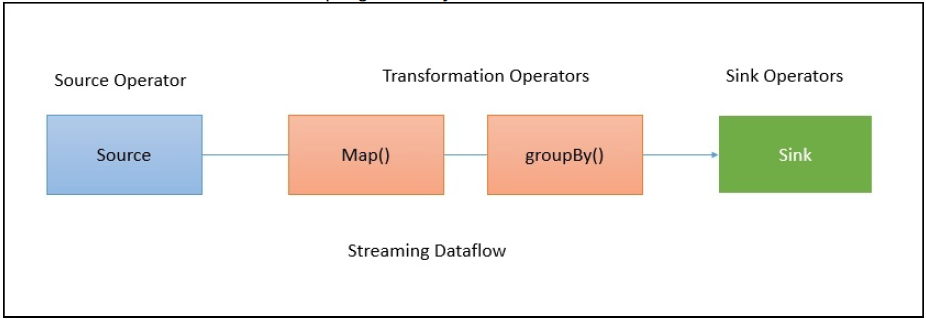
#### Job Client

Job Client不是Flink程序执行的一部分，但是他们是程序执行的开始点。Job Client主要负责接收用户提交的程序并创建数据流，然后提交该数据流到job manager去执行。当执行完成以后，job client将处理的结果返回给用户。

一个数据流是一个执行计划，考虑一个最简单的单词统计程序：

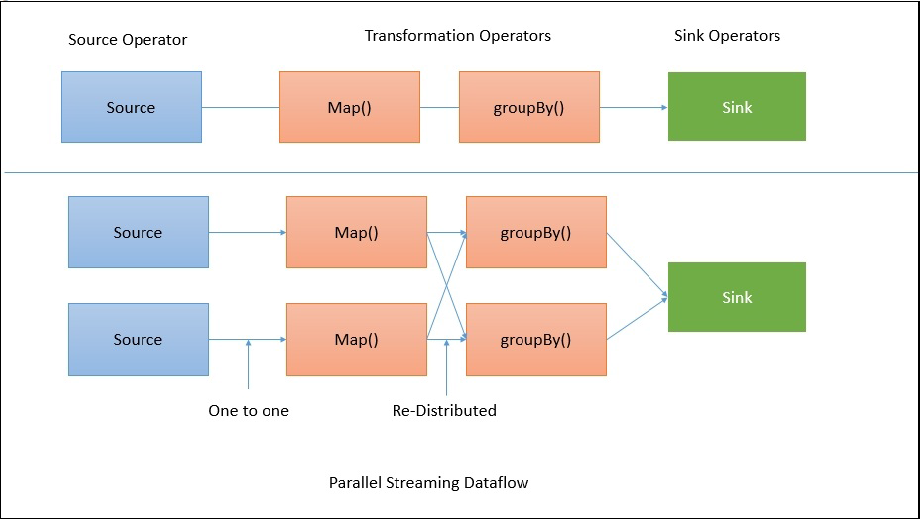


当client 接收到用户的程序，他们将其转换成数据流。上述程序的数据流图例如这个：



上图展示了一个程序一个程序怎么被转化成一个数据流的。模式数据流是分布式，并行化的。对于并行化的数据处理，Flink分解这些算子和流，算子被分解成子任务，流可以进行一对一分配或者重新分配。

数据流通过source到map算子由于不需要shuffle所以是直接分配，但是groupby()算子由于需要收集结果导致流重新分配。



### Features

#### Hight performance

Flink 可以实现高可用和低延迟，不像其他流式处理框架，例如spark，你不需要配置多种配置就可以达到更好的效果，flink相比于其他流处理框架，他的管道流处理给了他更好的性能。

#### Exactly-once stateful computation

在上一个部分我们已经讨论了，flink分布式的检查点机制保证饿了每一个记录精准一次处理。在高吞吐量应用的情况下，Flink为我们提供了一个开关，允许至少进行一次处理。

#### Flexible streaming windows

Flink支持数据驱动的windows，意味着我们可以根据时间，计数，会话来设计一个窗口，一个window也可以被我们自定义来检测时间流中特定的模式。

#### Fault tolerance

Flink的分布式轻量级快照机制有助于实现高度的容错性。 它允许Flink提供高吞吐量性能并保证交付。

#### Memory managerment

Flink在JVM内部提供自己的内存管理，使其独立于Java的默认垃圾收集器。 它通过使用散列，索引，缓存和排序有效地进行内存管理。

#### Optimizer

Flink的批量数据处理API经过优化，以避免诸如洗牌，分类等耗费内存的操作。 它还确保使用缓存以避免繁重的磁盘IO操作。

#### Stream and batch in one platform

Flink为批处理和流数据处理提供API。 因此，一旦您设置了Flink环境，它可以轻松托管流和批处理应用程序。 事实上，Flink的工作原理是Streaming，并将批处理视为流媒体的特例。

#### Libraries

Flink拥有丰富的库，可用于机器学习，图形处理，关系数据处理等。 由于其架构，执行复杂的事件处理和警报非常容易。 我们将在随后的章节中更多地了解这些库。

#### Event time semantics

Flink支持事件时间语义。 这有助于处理事件无序到达的流。 有时事件可能会延迟。 Flink的架构允许我们根据时间，计数和会话来定义窗口，这有助于处理这种情况

### Quick start setup

现在让我们来理解flink的结构方面的细节和其编程模型，让我们通过快速的配置来探索这些。Flink可以工作在windows和linux机器上。

首先我们需要下载flink二进制安装包，可以通过<http://flink.apache.org/downloads.html>进行下载。

在下载页面，你可以看到多种选项：



为了安装flink，你并不需要安装hadoop，但是当你需要连接hadoop的时候，为了兼容性，你需要下载对应的二进制版本。

#### Pre-requisite

Flink 需要java的支持，所以在你启动flink之前，需要保证java已经安装。

#### Installing on windows

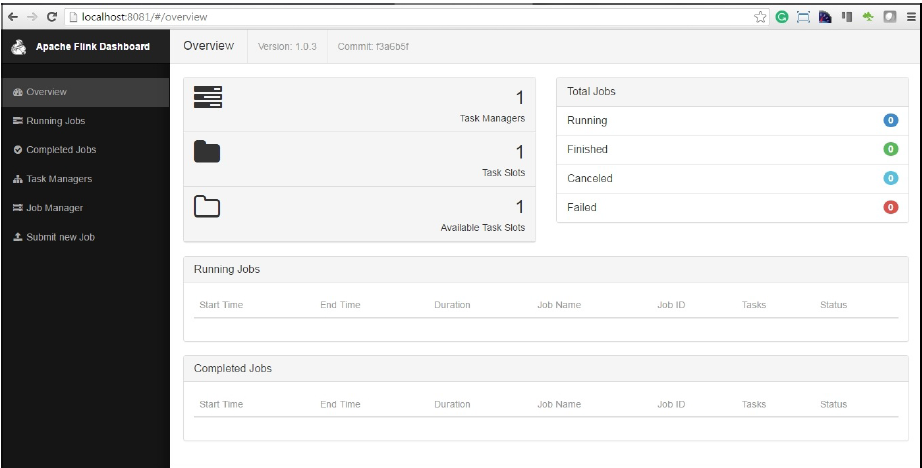
Flink安装是非常简单，仅仅需要解压文件，然后进入文件夹执行start-local.bat:

>cd flink-1.1.4

>bin\start-local.bat

然后你将会看到本地flink已经启动了。

你也可以通过web ui进行查看：<http://localhost:8081/>



你可以通过cltr+C进行停止flink。

#### Installing on Linux

与Windows安装类似，在linux上安装flink非常简单，我们需要下载二进制安装包，然后解压：

$sudo tar -xzf flink-1.1.4-bin-hadoop27-scala\_2.11.tgz

$cd flink-1.1.4

$bin/start-local.sh

与windows类似，你需要保证java已经在本机安装了，现在我们可以提交flink作业了，在linux上停止flink。我们可以执行写一下命令：

$bin/stop-local.sh

### Cluster setup

#### 待完成

最后来写，由于机器不足，不能实践。

### Running sample application

Flink二进制安装包中包含了一个简单应用的例子，让我们从这个简单的应用wordcount开始，现在我们运行该流应用，他通过netcat服务器特定端口读取数据。

因此我们首先需要通过以下命令在netcat服务器9000端口启动服务：

nc –l 9000

现在netcat服务器开始监听9000端口，当你键入的东西都会被发送到flink程序。

现在我们启动该流程序来监听netcat服务器，以下是启动命令：

bin/flink run examples/streaming/SocketTextStreamWordCount.jar --

hostname localhost --port 9000

08/06/2016 10:32:40 Job execution switched to status RUNNING.

08/06/2016 10:32:40 Source: Socket Stream -> Flat Map(1/1)

switched to SCHEDULED

08/06/2016 10:32:40 Source: Socket Stream -> Flat Map(1/1)

switched to DEPLOYING

08/06/2016 10:32:40 Keyed Aggregation -> Sink: Unnamed(1/1)

switched to SCHEDULED

08/06/2016 10:32:40 Keyed Aggregation -> Sink: Unnamed(1/1)

switched to DEPLOYING

08/06/2016 10:32:40 Source: Socket Stream -> Flat Map(1/1)

switched to RUNNING

08/06/2016 10:32:40 Keyed Aggregation -> Sink: Unnamed(1/1)

switched to RUNNING

这将会启动flink 作业的执行，现在我们在netcat控制台键入一些东西，flink程序将会处理它。

例如，在netcat server服务器上输入一下内容：

$nc -l 9000

hi Hello

Hello World

This distribution includes cryptographic software. The country in

which you currently reside may have restrictions on the import,

possession, use, and/or re-export to another country, of

encryption software. BEFORE using any encryption software, please

check your country's laws, regulations and policies concerning the

import, possession, or use, and re-export of encryption software,

to

see if this is permitted. See <http://www.wassenaar.org/> for

more

information.

你可以通过日志验证输出：

$ tail -f flink-\*-taskmanager-\*-flink-instance-\*.out

==> flink-root-taskmanager-0-flink-instance-1.out <==

(see,2)

(http,1)

(www,1)

(wassenaar,1)

(org,1)

(for,1)

(more,1)

(information,1)

(hellow,1)

(world,1)

==> flink-root-taskmanager-1-flink-instance-1.out <==

(is,1)

(permitted,1)

(see,2)

(http,1)

(www,1)

(wassenaar,1)

(org,1)

(for,1)

(more,1)

(information,1)

==> flink-root-taskmanager-2-flink-instance-1.out <==

(hello,1)

(worlds,1)

(hi,1)

(how,1)

(are,1)

(you,1)

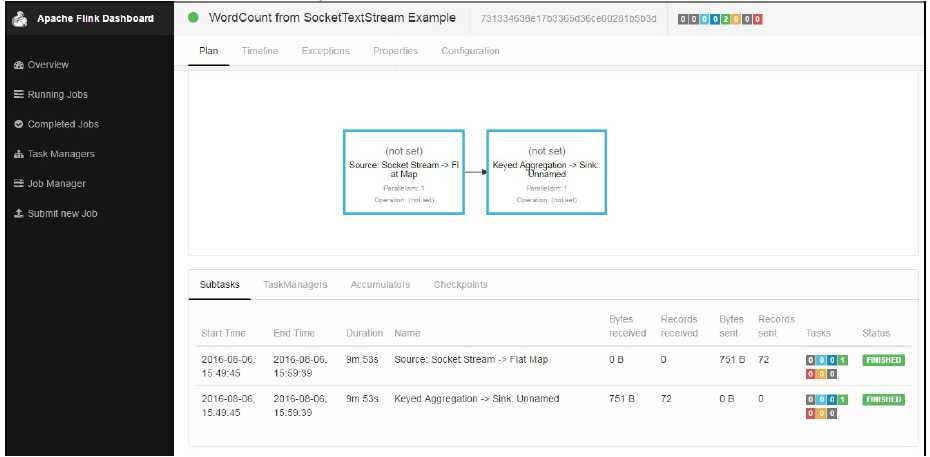
(how,2)

(is,1)

(it,1)

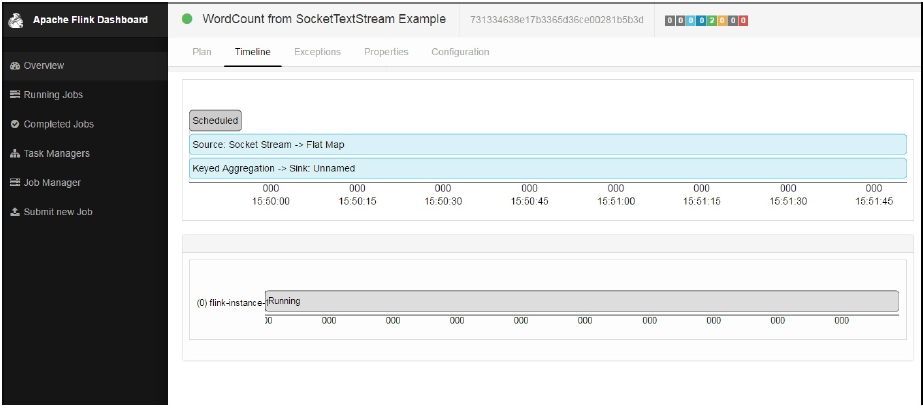
(going,1)

你也可以检查flink的web ui去观察该作业执行的性能。一下快照展示了该执行的数据流计划：



当前的作业，flink有两个算法，第一个算子是source算在，他通过socket流读取数据，第二个算在是转化算子，它聚集单词的计数。

我们也可以看到作业执行的时间线：



### Summary

在本章中，我们讨论了Flink如何开始作为一个大学项目，然后成为一个完整的企业级数据处理平台。 我们研究了Flink架构的细节以及它的流程模型是如何工作的。 我们还学习了如何在本地和集群模式下运行Flink。

在下一章中，我们将学习Flink的Streaming API并查看其细节，以及如何使用该API来解决我们的数据流处理问题。

## 2. Data Processing Using the DataStream API

## 3. Data Processing Using the Batch Processing API

## 4. Data Processing Using the Table API

## 5. Complex Event Processing

## 6. Maching Learning Using FlinkML

## 7. Flink Graph API – Gelly

## 8. Distributed Data Processing with Flink and Hadoop

## 9. Deploying Flink on Cloud

## 10. Best Practices