Twitter开源Summingbird：近原生编码下整合批处理与流处理

发表于 2013-09-04 15:15 | 5550次阅读 | 来源 CSDN | 6 条评论 | 作者 仲浩

云计算 大数据 开源 Twitter Summingbird Scalding Hadoop Storm

摘要：近日，Twitter开源了名为Summingbird的数据分析工具。区别于以往的更快、更准确节奏，Summingbird更注重于流处理与批处理的无缝整合，以及编程语言的原生化。

根据使用场景不同，大数据处理逐渐地向两个极端发展——批处理和流处理。其中流处理更注重于数据的实时分析，代表工具有Storm及S4。而批处理更注重于数据的长期挖掘，典型的工具则是从Google 三大论文上衍生的Hadoop。

随着数据的“爆棚”，各个公司在大数据的处理上可谓是绞尽脑汁，目的无非就是更快、更准。然而近日Twitter 开源的新工具Summingbird却打破了这个节奏，在流处理与批处理的无缝整合上更进了一步。

开发背景

众所周知，Twitter的系统基本上完成了面向服务的架构转变，而众多服务对数据处理也有着不同的需求，从而无可避免的出现这种情况：类似Trending Topics及搜索服务在开始时有实时处理的需求，而数据的价值却需要经过最终的深度挖掘——批处理。这样减少转换时开销的重要性就显而易见了，Summingbird应运而生。

Sunmmingbird

相关简介

Twitter在9月3日开源了一个名为Summingbird的大数据处理系统，通过整合批处理与流处理来减少它们之间的转换开销。

从Twitter对Summingbird的介绍还得知，开发者可以使用非常接近原生的Scala或者Java 在Summingbird上执行MapReduce作业，下面着眼一个使用纯粹Scala编写的word-counting示例：

```
def wordCount(source: Iterable[String], store: MutableMap[String, Long]) =
  source.flatMap { sentence =>
    toWords(sentence).map(_ -> 1L)
  }.foreach { case (k, v) => store.update(k, store.get(k) + v) }
```

而在Summingbird做word-counting则需要这样的代码

```
def wordCount[P <: Platform[P]]
  (source: Producer[P, String], store: P#Store[String, Long]) =
    source.flatMap { sentence =>
      toWords(sentence).map(_ -> 1L)
    }.sumByKey(store)
```

不难看出他们有着相同的逻辑和近乎完全相同的代码，然而不同的是，你既可以使用Summingbird项目做“批处理”（Scalding），也可以使用它做“实时处理”（使用Storm）；同时，你还可以使用两种模式的混合给应用程序带来无与伦比的容错性。

核心概念

Summingbird作业会产生两种类型的数据：流（stream）和快照（snapshot）。流包含了数据的所有历史，Store则是包含了系统在指定时间的快照。Summingbird核心通过众多组件实现：

- **Producer**——Producer是Summingbird的数据流抽象，用以传递给特定Platform做MapReduce流编译。
- **Platform**——Platform实例可以用于任何流MapReduce库的实现，Summingbird库包含了Platform对Storm、Scalding及内存处理的支持。
- **Source**——Source代表了一个数据的源，每个系统都对数据源有自己的定义，比如Memory平台将Source[T]定义为任何TraversableOnce[T]。
- **Store**——Store是Summingbird中流MapReduce进行“reduce”操作的场所，Store包含了所有键对应值聚合的快照。
- **Sink**——不同于Store，Sink允许你形成一个体现Producer值的非聚合流，sink是流而不是快照。
- **Service**——Service允许用户在Producer流中当前值上执行“lookup join”或者是“leftJoin”，被连接的值可以是来自另一个Store的快照，也可以是另一个Sink的流，甚至来自一些其它的异步功能。
- **Plan**—— Plan由Platform调用platform.plan(producer)产生，作为MapReduce流的最终实现。对于Storm来说，Plan就是个StormTopology实例，用户可以通过Storm提供的方法执行。对于Memory平台来说，Plan就是个内存Stream，包含了被传递Producer提供的输出内容。

详细内容请访问：Summingbird的核心概念

相关项目

Summingbird催生了大量的子项目，其中必须关注的有：

- **Algebird**——Scala的抽象代数库，Algebird中的众多数据结构都经Monoid实现，让他们能更好的进行Summingbird聚合。
- **Bijection**——Summingbird使用Bijection项目的Injection在不同客户端和执行平台间共享序列化。
- **Chill**——Summingbird的Storm及Scalding平台都使用了Kryo库做序列化处理，Chill是对Kryo一个很好的补充，其中包括许多可用配置选项，并且提供了Storm、Scala、Hadoop的使用模型。Chill同样也在伯克利Amp实验室的Spark中使用。
- **Tormenta**——Tormenta在Storm的Scheme及Spout接口上提供了类型安全层。
- **Storehaus**——Summingbird客户端通过 Storehaus的异步键值储存特性实现，Storm平台利用了 Storehaus的MergeableStore特性，以达到一些常用后备存储的实时聚合，包括memcache及Redis。

未来计划

- 支持更多的平台， Spark及 Akka“首当其冲”
- Producer图层的可插式优化
- 支持可过滤的数据源，比如 Parquet
- 为Producer基元注入更高级的数学及机器学习代码
- 通过相关项目，实现更多扩展
- 通过公共数据源释放更多的教程

更多详细信息访问Summingbird HomePage及Summingbird问题跟踪，Summingbird GitHub下载页。

扩展阅读：

- Testing
- Subpackage Descriptions
- Picking Monoids
- Long-Term Serialization
- The Storm Platform
- The Scalding Platform
- History and Motivation
- Core Concepts
- The Producer API
- Batch and Realtime
- The (deprecated) Builder API
- Frequently Asked Questions （审校/王鹏）