大数据系统　大作业展示

Toy-Spark

郝天翔　张洋

2019-12-16

选题简介

实现一个 Mini 版本的 Spark
（Spark 是一个内存计算框架）

我们的目标是保留 Spark 的以下特性：

- 用户只需编写串行程序
- 自动并行化和分布式执行
- 计算过程在内存中完成

在此基础上,希望尽可能多地提供和原始 Spark 相同的功能性，而非通过大量减少功能来保证性能。

实现情况

- 任务划分
- 节点间的通信与协作
- 血统机制与惰性计算
- 简单的 transformation 与 action
- 简单的需要进行 shuffle 的操作
- 支持读写 HDFS
- **支持 RDD 之间的关系为一般化的有向无环图**
  上个月时仅支持 RDD 之间的依赖关系是线性的
- 支持一次提交中包含多个子计算任务，复用部分数据
- 简单的需要多个 RDD 作为参数的操作
- **支持在内存中缓存 RDD**
  只有 RDD 之间是线性的时候才会有缓存的问题

实现的 API

- 不多于一个参数的 transformation：
  `map`、`filter`、`repartition`、`flatMap`、`distinct`、`groupByKey`、`reduceByKey`
- 两个参数的 transformation：
  `unionWith`、`intersectionWith`、`cartesianWith`、`joinWith`
- action：
  `reduce`、`collect`、`count`、`take`、`saveAsSequenceFile`、`saveAsSingleFile`
- 数据生成：`generate`、`read`
- 缓存相关：`save`

系统架构与运行流程

- 节点类型：master、worker
  master 和 worker 任务一致；此外 master 还负责协调
- 线程类型：manager、executor、communicator
  - manager：负责协调自己节点上的其它线程；处理 **控制通信**
  - executor：负责完成实际的计算（不再负责通信）
  - communicator：负责处理所有的 **数据通信** 请求
- 大致的运行过程
  1. 用户提交任务，设定配置；节点完成握手
  2. master 划分任务为 jobs，再划分 job 为 stages
  3. master 对 stages 按照依赖关系排序，确定 partition schema
  4. 逐个 stage 进行计算
  5. 最后结果汇总到 master 上，用 Scala 的原生数据类型返回用户

实现思路

- RDD 为线性依赖如何进行 stage 划分
- RDD 为 DAG 依赖如何进行 stage 划分
- 如何处理通信
- 如何实现内存中缓存

RDD 为线性依赖如何进行 stage 划分

在这个假定下，所有的计算总是若干个 transformation 再加一个 action

transformation 可以分成两类：

- 类似 `map` 的在求值过程中不关心其它分片的值，谓之 **direct dependency**
- 类似 `repartition` 的关心其它分片的值，谓之 **shuffle dependency**

对于线性依赖的 RDD，只需要在 shuffle dependency 处划分出新的 stage 即可

RDD 为 DAG 依赖如何进行 stage 划分

- 在这个假定下，用户一次提交可能有多个 job（希望得到多个输出）
- 首先对 job 进行排序，显然只需要使用用户在代码中提供的顺序即可
- 然后需要在 DAG 上找到所有被这个 job 依赖的 RDD，以树的形式表示
  DAG 上可能出现一个节点分裂的情况，在这一步中我们是在反向地进行 stage 划分，遇到节点分裂时看到的实际上是“节点合并”，此时让该节点在多次出现即可实现用树表示 RDD
- 对于每个需要两个参数的 transformation，记两个参数分别为 `lhs` 和 `rhs`
  我们总是让计算结果的 partition schema 和 `lhs` 一致，它们形成 direct dependency；计算结果和 `rhs` 之间形成 shu
  dependency
  遇到需要两个参数的 transformation 时，划分策略为先计算 `rhs` 再计算 `lhs`
- 现在不用处理分叉的情况了，按照线性依赖的处理手段进行处理即可

RDD 为 DAG 依赖如何进行 stage 划分（续）

依上述策略，得到的 stage 划分满足：

- stage 内部的依赖都是 direct dependency
- stage 计算过程中如果需要使用 shuffle dependency，则数据一定已经就绪
- stage 中没有分支也没有汇合，是一条链，可以顺序执行

RDD 为 DAG 依赖如何进行 stage 划分（例）

如何处理通信

- 每个节点会维护一个节点共用的 `sendingBuffer`
- 一个 stage 计算完成后就会把自己的数据存到 `sendingBuffer` 中
- 下游需要这个数据的 stage 会通过请求相应节点的 communicator 取到数据
- 数据被取一次之后就可以删除了
  这是因为我们假定了计算的依赖总是以树的形式存在的
  根据这个性质，也可以使用文件实现 `sendingBuffer`

如何实现内存中缓存

如果实际执行只能把 DAG 展开成树的话，RDD 无法被重用。

调用 RDD 的 save 方法即可让其在内存中缓存。

- 当一个 RDD 被调用 save 时，对其进行标记
  由于惰性求值，只能先进行标记，而不是立即计算
- 当一个有标记的 RDD 被计算出时，把数据放入 memCache 中
  memCache 在一个节点上共享
- 当要对一个 RDD 求值的时候，如果它在 memCache 中有对应的 entry，可以直接获取它的值；没有才需要进行计算
- 每个 job 开始时的 stage 需要重新进行划分
  因为有些 stage 只需要计算一部分，有些则可以完全不用计算
  这一步的划分依然得到一棵树，但是合理使用 save 可以让这棵树没有重复的分支

单元测试

对数据生成方法、Transformation和Action进行正确性测试

- filter + count
- map + collect
- repartition + reduce
- unionWith + count
- intersectionWith + collect
- cartesianWith + take
- saveAsSingleFile
- read

性能测试

- 将Spark examples中的SparkPi示例程序在Toy-Spark上进行实现，对比运行时间。
- Spark: 约0.4s。
- Toy-Spark: 约2.8s。

示例程序

- CalculatePi: 蒙特卡罗法计算Pi值。
- PrimaryMath: 蒙特卡罗法计算正方形内以边长为半径的半圆与四分之一圆相交部分的面积。

系统演示

- 使用蒙特卡洛的方法解阴影部分面积
- 用 Toy-Spark 跑 PageRank

分工

张洋

- 设计 Toy-Spark 的框架
- 实现 Toy-Spark 的框架
- PPT 制作与文档撰写

郝天翔

- HDFS 支持
- 一些 transformation 与 action 的实现
- 正确性测试与性能测试

Q & A