Design Of ToySpark

2019/10/24

张洋　　郝天翔

整体架构

- 节点分为 master 和 worker
  master 的工作和 worker 类似，但是多了管理的工作
- master 和 worker 运行的是同一个程序
  它们根据传入的配置在运行时知道自己的身份
- 节点具体要完成什么工作由程序决定
  以 C++ 实现为例，用户只要在 main.cpp 中指定工作即可编译后部署
- 一个 RDD 是计算过程中的“中间结果”
  RDD 包含若干个 partition
  partition 包含若干个 record
  RDD 之间会形成依赖关系，partition 之间也会形成依赖关系

目前假定的情况

- RDD 的依赖关系是线性的
  必然是连续若干个 transformation，然后以一个 action 结束
  如果时间允许，我们会移除这个假定，支持最通用的 DAG 形式
- 系统不会出现故障
- 计算过程中 partition 数不小于节点数

运行流程

准备阶段

1. 手动启动 master 节点

   master 节点会读取配置获知自己的身份，并等待来自 worker 的连接

2. 手动启动所有 worker 节点

   worker 节点会读取配置获知自己的身份，从中得到联系 master 的方法并发起连接

3. master 等待所有 worker 节点连接好后，准备阶段结束

任务划分阶段

1. master 根据计算任务的所有 transformation 和 action 得到 RDD 的依赖图
2. master 根据 RDD 的依赖图进行 ***stage*** 的划分
   stage：划分完成后，每个 stage 计算过程中每个线程之间不会产生依赖或干扰
3. master 将划分结果告知 worker 节点
4. worker 对每个 stage 计算出若干个 ***线程计算图***，每个 partition 对应一个线程
   线程计算图：一个线程应当如何完成当前 stage 的计算
5. 每个 worker 会：
   - 在每个 stage 中维护数目同当前节点 partition 数的 ***executor 线程*** 进行计算
   - 在整个计算流程中保留一个 ***manager 线程*** 负责协调

任务计算阶段

数据的获取与开启计算

1. stage 开始后，manager 创建出需要的所有 executor
2. executor 根据自己的 ID，推算 **获取数据的方法**
   - 如果是第一个 stage，则 ID 决定读取本地数据的哪一部分
   - 如果不是，则 ID 决定应该获取上游 partition 的哪些部分
3. 对于数据中的每一条 record，**非惰性** 地进行计算，并在线程计算图的每个节点上同步

任务计算阶段（续）

数据的同步与结束计算

1. executor 在线程计算图的最后一个节点上完成计算后，通知 manager 线程已完成
2. executor 开始等待下游 executor 获取自己的数据
3. **本地同步**：manager 在发现本地节点所有 executor 完成后，通知 master
4. **全局同步**：master 在发现所有节点完成后，发送开启下一 stage 的通知
5. executor 在下游全部 executor 获取完自己的数据后，结束掉自己的运行

结束阶段

- 我们可以将最后的 action 视作只有一个 partition 的特殊 stage，即可和前面采取类似的过程。
- 如果一个 worker 不再负责任何 partition，它会在发送完所有数据后结束运行。
- master 会在收集完数据之后用 C++ 的原生数据类型返回给用户。

TRANSFORMATION 和 ACTION 的实现

- `map` 生成的 RDD 会记下需要执行的 map 操作
- `filter` 生成的 RDD 会记下需要执行的 filter 操作
- `coalesce` 会重新划分 partition，可以通过随机哈希的方式决定下游 partition
- `reduce` 操作必须满足交换律和结合律，因此可以先在 partition 内部进行 reduce，然后再汇总到 master
- `collect` 操作会先在每个节点上合并，然后一起发送给 master

其中 `map` 和 `filter` 不会使 stage 断开，其它的都会。

THANKS

Q & A ?