11 MapReduce

- 大纲
  - MapReduce 编程模型
  - MapReduce 解决的典型问题
  - MapReduce 示例
  - 一个简短的历史
  - MapReduce 执行概述
  - Hadoop
- 为什么会产生 MapReduce 并行编程模型?
  - 大数据的呈现出指数增长速度
  - 大规模数据处理
    - 想要处理大量数据（>1TB）
    - 想要在成百上千个 CPU 上并行化
    - 想让这件事变得简单
- MapReduce
  - 一个简单而强大的接口，可实现大规模计算的自动并行化和分布，结合该接口的实现，可在大型商用 PC 集群上实现高性能。”
  - 更简单地说，MapReduce 是一种并行编程模型和相关的实现
  - **和MPI的差别——运行时环境不同**
    - MapReduce——PC——性能差，可靠性强
    - MPI——HPC
    - MapReduce是数据流引擎，MPI是通过传统网络进行消息传递的通信
  - 术语
    - job——“完整程序”——跨数据集执行 Mapper 和 Reducer
    - 任务——在数据切片上执行 Mapper 或 Reducer
      - 又名进行中的任务 (TIP)
    - Task Attempt ——尝试在机器上执行任务的特定实例
  - 例子
    - 跨 20 个文件运行“数单词”是一项工作
    - 要映射的 20 个文件意味着 20 个 map 任务 + 一些 reduce 任务
    - 至少会执行 20 次 map 任务尝试……如果机器一台崩溃，则更多次等
  - Task Attempt
    - 一个特定的任务将至少尝试一次，如果它崩溃可能会尝试更多次
      - 如果相同的输入反复导致崩溃，则该输入最终将被放弃

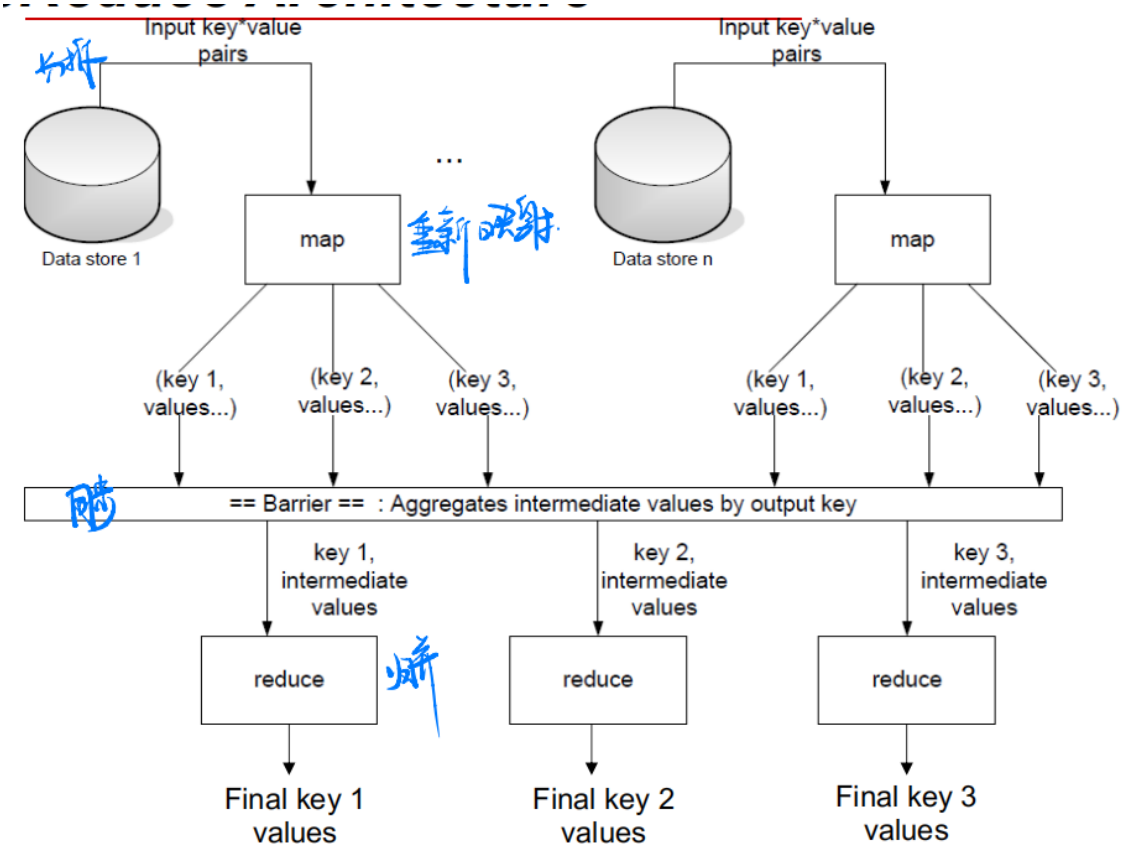
    - 在开启推测执行的情况下，可能会同时发生对一项任务的多次尝试
        - TaskInProgress 中的task ID 不是唯一标识符
- MapReduce编程模型
    - 使用特殊的 map() 和 reduce() 函数处理数据
        - map() 函数在输入中的每个项目上调用，并发出一系列中间键/值对
        - 与给定键关联的所有值都组合在一起
        - 在每个唯一键及其值列表上调用 reduce() 函数，并发出一个添加到输出的值
    - Map
        - 来自数据源的记录（文件中的行、数据库中的行等）以键*值对的形式输入到 map 函数中：例如，（文件名、行）
        - map() 产生一个或多个中间值以及来自输入的输出键

```
– map  (in_key, in_value) ->
   (out_key, intermediate_value) list
```

    - Reduce
        - 映射阶段结束后，给定输出键的所有中间值组合在一起形成一个列表
        - reduce() 将这些中间值组合为同一输出键的一个或多个最终值

```
– reduce (out_key, intermediate_value list) ->
  out_value list
```

    - MapReduce架构



    - MapReduce 运行时系统

    - 分区输入数据
        - 在不同节点之间分区和分发数据
    - 在一组机器上安排执行
        - Yarm——（资源管理器），先进先出。（许多研究工作）
    - 处理机器故障
        - 机器出现故障时重新安排任务。
    - 管理进程间通信
- 并行化
    - map() 函数并行运行，从不同的输入数据集创建不同的中间值
        - map和map之间没有通信，但是map和reduce之间有
    - reduce() 函数也并行运行，每个函数处理不同的输出键
    - 所有值都独立处理
    - 瓶颈：在 map 阶段完全完成之前，reduce 阶段无法启动
- 局部性
    - 主程序根据数据的位置划分任务：尝试将 map() 任务与物理文件数据放在同一台机器上，或者至少在同一个机架上
    - map() 任务输入分为 64 MB 块：与 Google 文件系统块大小相同
- **容错性**
    - Master 检测到 worker 故障
        - 重新执行已完成和正在进行的 map() 任务
        - 重新执行正在进行的 reduce() 任务
    - Master 注意到特定的输入键/值导致 map() 崩溃，标识他们并在重新执行时跳过这些值
        - 效果：可以解决第三方库中的错误！
- **优化**
    - 在 map 完成之前，reduce 不能启动
        - 单个slow disk控制器可以对整个过程进行速率限制
    - Master冗余地执行“慢动作”的map任务； 使用第一个完成的副本结果
    - “Combiner”功能可以与maper在同一台机器上运行
    - 在真正的 reduce 阶段之前发生一个 mini-reduce 阶段，以节省带宽
- 优点
    - 大大降低并行编程复杂度
        - 降低同步复杂度
        - 自动分区数据
        - 提供故障透明度
        - 处理负载平衡

- MapReduce可以解决什么问题
  - 阅读大量数据。
  - Map：从每条记录中提取您关心的内容。
  - 洗牌和排序。
  - 减少：聚合、汇总、筛选或转换。
  - 写下结果。
  - 大纲保持不变，但map并reduce更改以适应问题

以上内容整理于 幕布文档