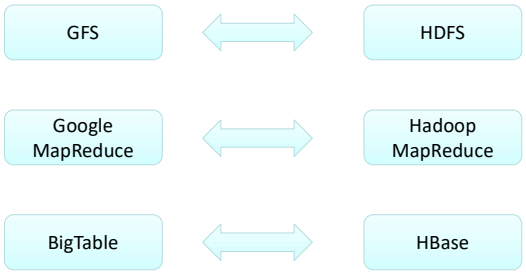
摘要

- Google的三驾马车
  - Google File System
  - Google MapReduce
  - Google Cloud Bigtable（Optional）

谷歌的三驾马车与Hadoop的简单对应

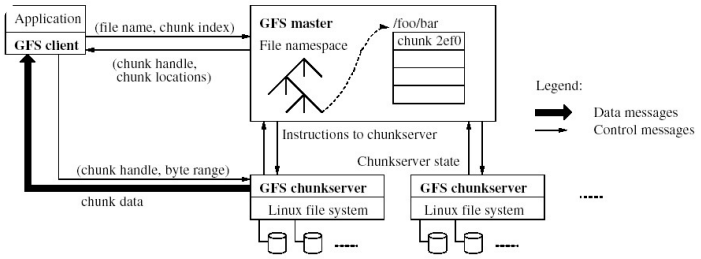


GFS的设计思路

- 将文件划分为若干块（Chunk）存储
  - 每个块固定大小（64M）
- 通过冗余来提高可靠性
  - 每个数据块至少在3个数据块服务器上冗余
- 通过单个master来协调数据访问、元数据存储
  - 结构简单，容易保持元数据一致性
- 不缓存数据

GFS的架构

- 一个GFS包括单个主服务器（master）和多个块服务器（chunk server）



Master节点的任务

- 实施Access Control，存储元数据（Matadata）
  - 元数据包括三类：
    - File and chunk namespaces
    - The mapping from files to chunks
    - The location of each chunk's replicas
- 文件系统目录管理与加锁
- 与ChunkServer进行周期性通信
  - 发送指令，搜集状态，跟踪数据块的完好性

Master节点的任务

- 数据块创建、复制及负载均衡
  - 对Chunk Server的空间使用和访问速度进行负载均衡
  - 对数据块进行复制、分散到ChunkServer上
  - 一旦数据块冗余数小于最低数，就发起复制操作
- 垃圾回收
  - 在日志中记录删除操作，并将文件改名隐藏
  - 缓慢地回收隐藏文件
  - 与传统文件删除相比更简单、更安全
- 陈旧数据块删除

单一Master问题

- 分布式系统设计告诉我们：
  - 这是单点故障
  - 这是性能瓶颈
- GFS的解决办法
  - 单点故障问题

采用多个（如3个）影子Master节点进行热备，一旦主节点损坏，立刻选举一个新的主节点服务

单一Master问题

- GFS的解决办法
  - 性能瓶颈问题

尽可能减少数据存取中Master的参与程度

不使用Master读取数据，仅用于保存元数据

客户端缓存元数据

采用大尺寸的数据块（64M）

数据修改顺序交由Primary Chunk Server完成

GFS架构的特点

- 采用中心服务器模式
  - 可以方便地增加Chunk Server
  - Master掌握系统内所有Chunk Server的情况，方便进行负载均衡
  - 由于只有一个中心服务器,不存在元数据的一致性问题

GFS架构的特点

- Chunk Server不缓存文件数据
  - GFS的文件操作大部分是流式读写，不存在大量的重复读写，且多数文件太大而无法缓存
  - Chunk Server上的数据存取使用本地文件系统，如果某个Chunk读取频繁，本地缓存于内存中。
  - 从可行性看，Cache与实际数据的一致性维护也极其复杂，不缓存避免了一致性维护问题

GFS架构的特点

- 在用户态下实现
  - 直接利用Chunk Server的文件系统，实现简单
  - 用户态应用调试较为简单，利于开发
  - 用户态的GFS不会影响Chunk Server的稳定性
- 提供专用的访问接口
  - 未提供标准的POSIX访问接口
  - 降低GFS的实现复杂度

GFS的容错方法

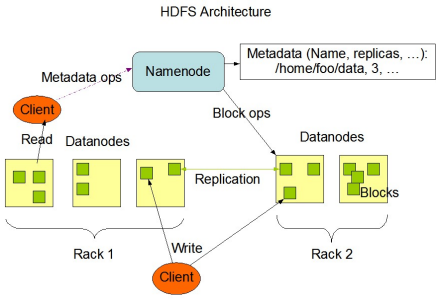
- Master容错
  - 三类元数据：命名空间（目录结构）、Chunk与文件名的映射以及Chunk副本的位置信息
  - 前两类通过日志提供容错，Chunk副本信息存储于Chunk Server，Master出现故障时可恢复
- Chunk Server容错
  - 每个Chunk有多个存储副本（replica, 通常是3个），分别存储于不通的服务器上
  - 每个Chunk又划分为若干Block（64KB），每个Block对应一个32bit的校验码，保证数据正确（若某个Block错误，则转移至其他Chunk副本）

副本放置方法

- 数据副本的排布影响可靠性和性能
  - 数据可靠性，可用性，网络带宽等
- 每个unique rack放置一个副本？
  - 利：高可靠，高性能
  - 弊：开销大
- HDFS的方法：
  - Put one replica on one node in the local rack, another on a node in a different (remote) rack, and the last on a different node in the same remote rack.

Hadoop Distributed File System (HDFS)

- 一种运行在商用硬件上的分布式文件系统
- 和传统分布式文件系统的主要区别之一在于其构建于低成本硬件上的容错性



GFS在Google中的部署

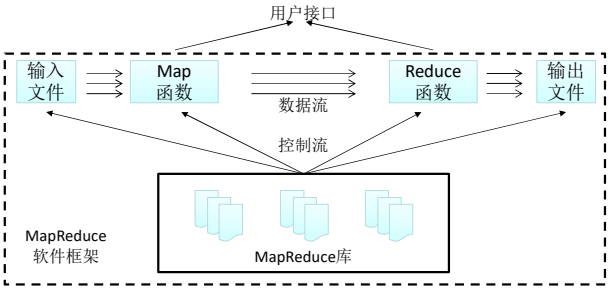
- 超过50个GFS集群
- 每个集群包含数千个存储节点
- 管理着PB(10^{15}Byte)级的数据

为什么需要 MapReduce？

- 计算问题简单，但求解困难
  - 待处理数据量巨大（PB级），只有分布在成百上千个节点上并行计算才能在可接受的时间内完成
  - 如何进行并行分布式计算？
  - 如何分发待处理数据？
  - 如何处理分布式计算中的错误？

MapReduce 框架

- 一个软件框架，处理海量数据的并行编程模式
- 在大数据时代，代表“数据并行”型计算模式



使用特别的用户接口来访问Map和Reduce资源。输入数据流经Map和Reduce函数，在软件库的控制流下产生输出结果。

Map和Reduce函数

- MapReduce提供给用户两个重要功能接口
  - Map（映射）和Reduce（化简）
  - 用户可重载这两个函数以实现不同形式的数据流操纵
- MapReduce中的数据流以特定结构呈现
  - <key, value>
  - 其中，“value”部分是实际数据，“key”部分只是被MapReduce控制器使用来控制数据流
  - 寻找合适的key是解决一个MapReduce问题的出发点

MapReduce

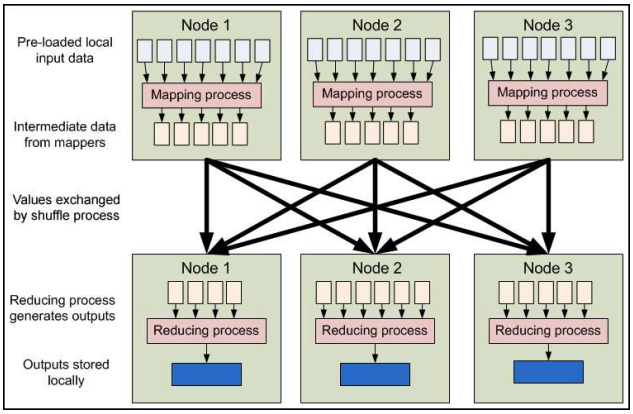
- Map把一个函数应用于集合中的所有成员，然后Reduce对结果集进行分类和归纳

(key_1, val_1) —Map函数→ List (key_2, val_2)

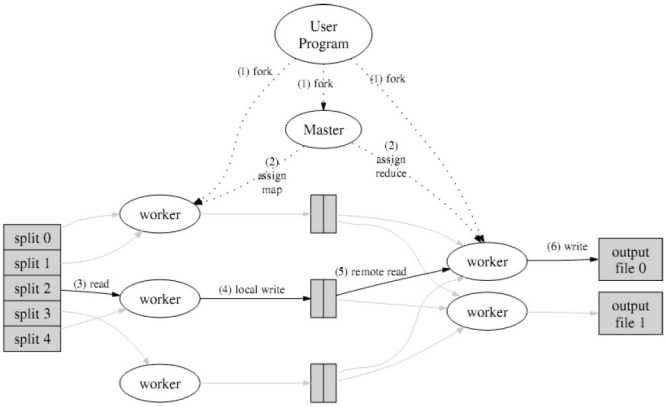
$(key_2, List(val_2))$ —Reduce函数→ List(val_2)

1. 对每个输入的<k,v>对并行地应用map函数，生成一系列中间<k,v>对
2. MapReduce库收集中间数据，基于键值对进行排序和统计
3. 分组并行地应用Reduce，产生值集合作为输出

MapReduce



Google MapReduce 执行流程



Bigtable

- 需存储的数据种类繁多
  - 网页，地图数据，邮件……
  - 如何使用统一的方式存储各类数据？
- 海量的在线服务请求
  - 数十亿的URL，用户规模巨大，每秒钟上千次查询
  - 如何快速地从海量信息中寻找需要的数据？
- 传统商用数据库无法解决如此大规模的结构化或半结构化的数据处理问题

Google的需求和假设

- 需求：
  - 高速数据检索与读取
  - 数据存储可靠性
  - 存储海量的记录（若干TB）
  - 可以保存记录的多个版本
- 假设：
  - 与写操作相比，读操作占绝大多数
  - 单个节点故障损坏是常见的
  - 磁盘是相对廉价的
  - 可以不提供标准接口

数据模型

- 行（Row）
  - 每行数据有一个可排序的关键字和任意列项
  - 字符串、整数、二进制串甚至可串行化的结构都可以作为关键字，称为行键（Row Key）
  - 按照行键“逐字节排序”实施有序化处理
  - 表内数据非常‘稀疏’，不同的行的列数可不同
  - URL是较为常见的行键，存储时需要倒排
    - 统一地址域的网页连续存储，便于查找、分析和压缩

mp3.baidu.com/index.asp→com.baidu.mp3/index.asp

数据模型

- 列(Column)
  - 特定含义的数据的集合，如图片、链接等
  - 可将多个列归并为一组，称为族（family）
  - 采用 族:限定词 (Qualifier)的规则进行定义
    - 例：“fileattr:owning_group”
  - 同一个族的数据被压缩在一起保存
  - 族是BigTable中访问控制的基本单元

数据模型

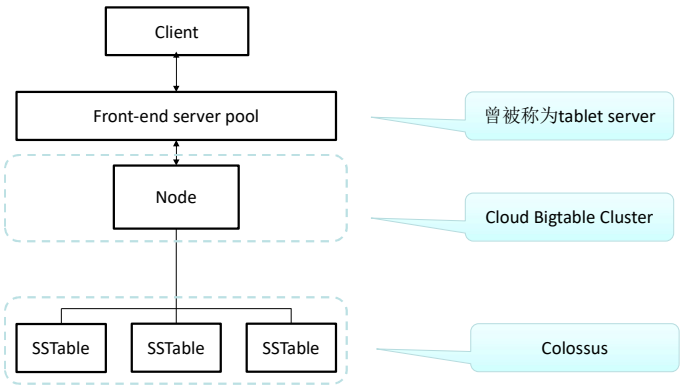
- 时间戳（Timestamp）
  - 保存不同时期的数据，如“网页快照”
- “A big table”
  - 表中的列可以不受限制地增长
  - 表中的数据几乎可以无限地增加

通过(row, col, timestamp)查询
通过(row, col, MOST_RECENT)查询

数据模型

- 无数据校验
  - 每行都可存储任意数目的列
    - BigTable不对列的最少数目进行约束
  - 任意类型的数据均可存储
    - BigTable将所有数据均看作为字符串
  - 数据有效性校验由构建于其上的应用系统完成
- 一致性
  - 针对同一行的多个操作可以分组合并
    - 不支持对多行进行修改的操作符

体系结构



Cloud Bigtable tablets

- 逻辑上的“表”被划分为若干子表（tablet）
  - 每个Tablet由多个SSTable文件组成
  - SSTable文件存储在GFS之上
- 每个子表存储了table的一部分行
  - 元数据：起始行键、终止行键
  - 如果子表体积超过了阈值（如200M），则进行分割

Cloud Bigtable nodes

- 数据不在Cloud Bigtable nodes上存储
- 每个节点都有指向一系列tablets的指针
  - 对tablets的重新均衡很快速
    - 因为数据并不迁移复制，只需要更新指针
  - Cloud Bigtable nodes的灾害回复也很快
    - 仅仅metadata需要被迁移到新节点
    - 即使Cloud Bigtable node失效，也没有数据损失