大数据技术原理与应用

第一章 大数据概述

大数据

1.1 大数据摩尔定律：根据IDC作出的估测，数据一直都在以每年50%的速度增长，也就是说每两年就增长一倍。

1.2 大数据的四个特征：

- 大量化
- 快速化
- 多样化
- 价值化

1.3 大数据带来的三个思维方式方面的变化：

- 全样而非抽样
- 效率而非精确
- 相关而非因果

1.4 两大核心技术： 分布式存储、分布式处理

1.5 大数据计算模式及其代表产品

表1-3 大数据计算模式及其代表产品

大数据计算模式	解决问题	代表产品
批处理计算	针对大规模数据的批量处理	MapReduce、Spark等
流计算	针对流数据的实时计算	Storm、S4、Flume、Streams、Puma、DStream、Super Mario、银河流数据处理平台等
图计算	针对大规模图结构数据的处理	Pregel、GraphX、Giraph、PowerGraph、Hama、GoldenOrb等
查询分析计算	大规模数据的存储管理和查询分析	Dremel、Hive、Cassandra、Impala等

云计算

云计算主要解决两个方面的问题：

1. 分布式存储与分布式处理
2. 虚拟化与多租户

1. 云计算概念

- 云计算实现了通过网络提供可伸缩的、廉价的分布式计算能力，用户只需要在具备网络接入条件的地方，就可以随时随地获得所需的各种IT资源

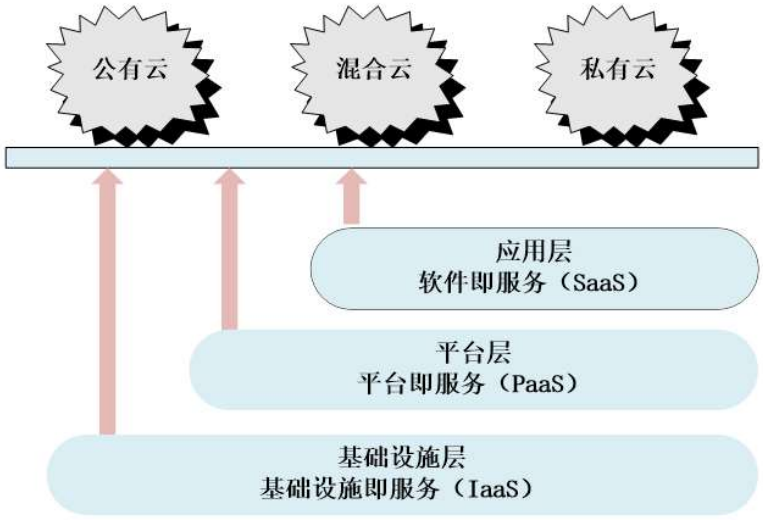


图1-7 云计算的服务模式和类型

物联网

1. 物联网概念

- 物联网是物物相连的互联网，是互联网的延伸，它利用局部网络或互联网等通信技术把传感器、控制器、机器、人员和物等通过新的方式联在一起，形成人与物、物与物相联，实现信息化和远程管理控制

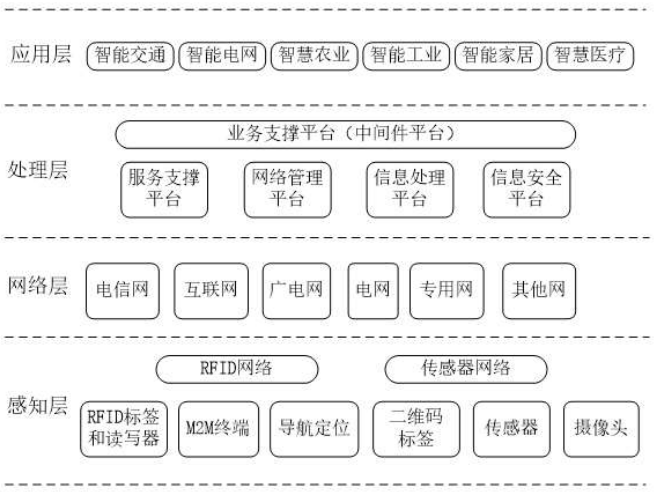


图1-9 物联网体系架构

三者关系

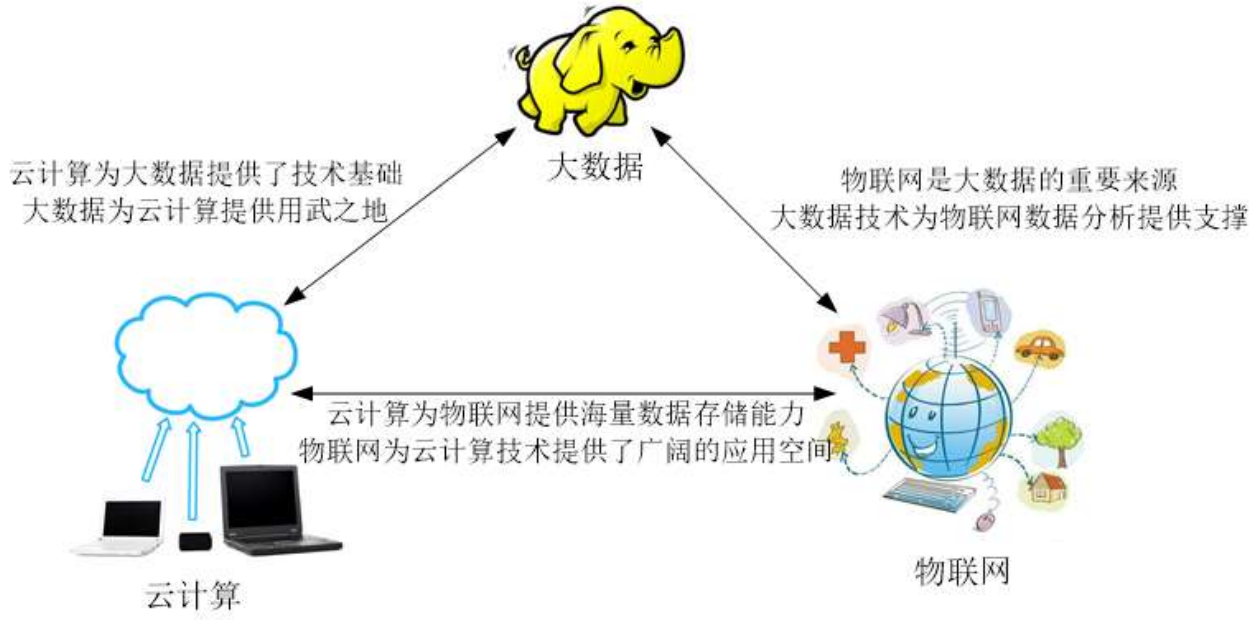
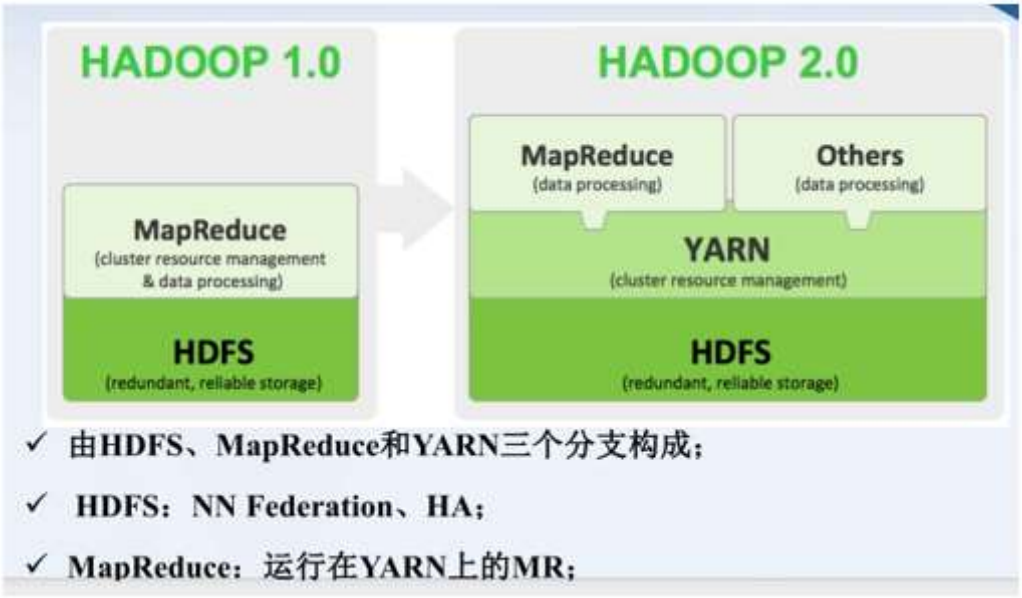


图1-9 大数据、云计算和物联网之间的关系

第二章 大数据处理架构Hadoop

版本

2.1 版本演进

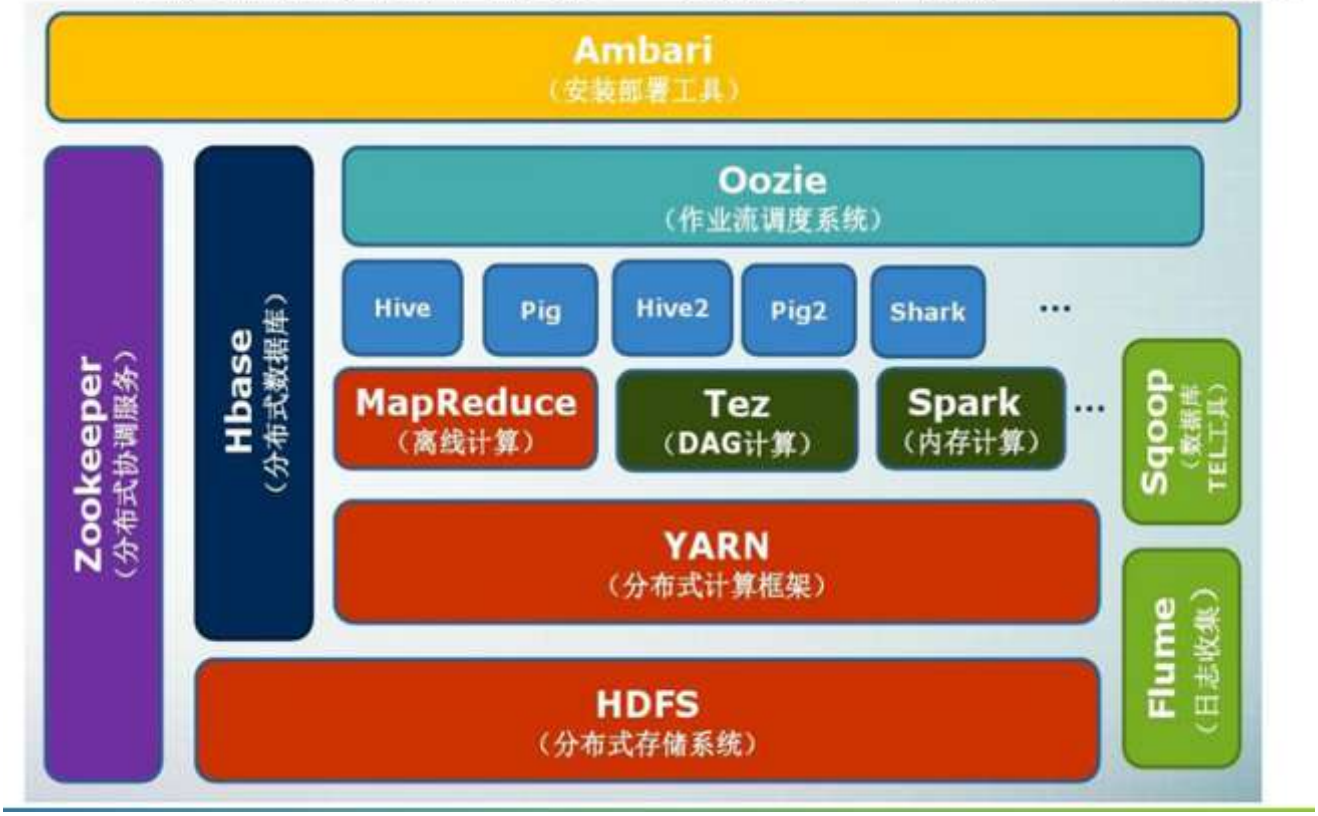


2.2 不同企业版本对比

厂商名称	开放性	易用性(★)	平台功能	性能(★)	本地支持	总体评价(★)
apache	完全开源、Hadoop就是托管在apache社区里面	安装：2 使用：2 维护：2	Apache是标准的Hadoop平台，所有厂商都是在apache的平台上面进行改进	2	没有	2
cloudera	与Apache功能同步，部分代码开源	安装：5 使用：5 维护：5	有自主研发的产品如：impala、navigator等	4.5	2014年刚进入中国，上海	4.5
hortonworks	与apache功能同步，也是完全开源	安装：4.5 使用：5 维护：5	是apache hadoop平台的最大贡献者，如Tez	4.5	没有	4.5
MapR	在apache的hadoop版本上面修改很多	安装：4.5 使用：5 维护：5	在apache平台上面优化很多、从而形成自己的产品	5	没有	3.5
星环	核心组件与apache同步、底层的优化比较多、完全封闭的一个平台	安装：5 使用：4 维护：4	有自主的Hadoop产品如Inceptor、Hyperbase	4	本地厂商	4

项目结构

Hadoop的项目结构不断丰富发展，已经形成一个丰富的Hadoop生态系统



- HDFS：分布式文件系统
- MapReduce：分布式并行编程模型
- YARN：资源管理和调度器
- Tez：运行在YARN之上的下一代Hadoop查询处理框架
- Hive：Hadoop上的数据仓库
- HBase：Hadoop上的非关系型的分布式数据库
- Pig：一个基于Hadoop的大规模数据分析平台，提供类似SQL的查询语言Pig Latin
- Sqoop：用于在Hadoop与传统数据库之间进行数据传递
- Oozie：Hadoop上的工作流管理系统
- Zookeeper：提供分布式协调一致性服务
- Storm：流计算框架
- Flume： 一个高可用的，高可靠的，分布式的海量日志采集、聚合和传输的系统
- Ambari：Hadoop快速部署工具，支持Apache Hadoop集群的供应、管理和监控
- Kafka：一种高吞吐量的分布式发布订阅消息系统，可以处理消费者规模的网站中的所有动作流数据
- Spark：类似于Hadoop MapReduce的通用并行框架

安装教程

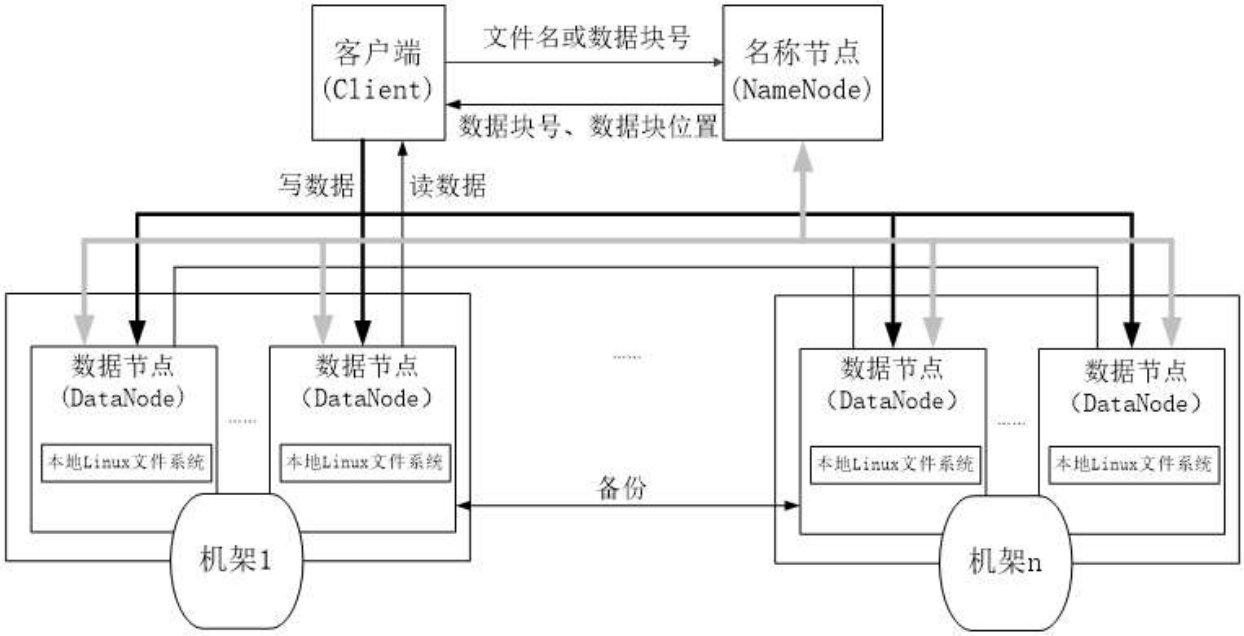
大数据技术原理与应用 第二章 大数据处理架构Hadoop 学习指南

第三章 分布式文件系统HDFS

> **只允许追加，不允许修改**

实践教程：大数据技术原理与应用 第三章 分布式文件系统HDFS 学习指南

体系结构



- 数据块
  支持大规模文件存储、简化系统设计、适合数据备份

- 名称节点（NameNode）

第二名称节点（SecondaryNameNode）为冷备份

1. FsImage：用于维护文件系统树以及文件树中所有的文件和文件夹的元数据
2. 操作日志文件EditLog：记录所有针对文件的创建、删除、重命名等操作

- 数据节点（DataNode）

优缺点（Hadoop1.0）

优点

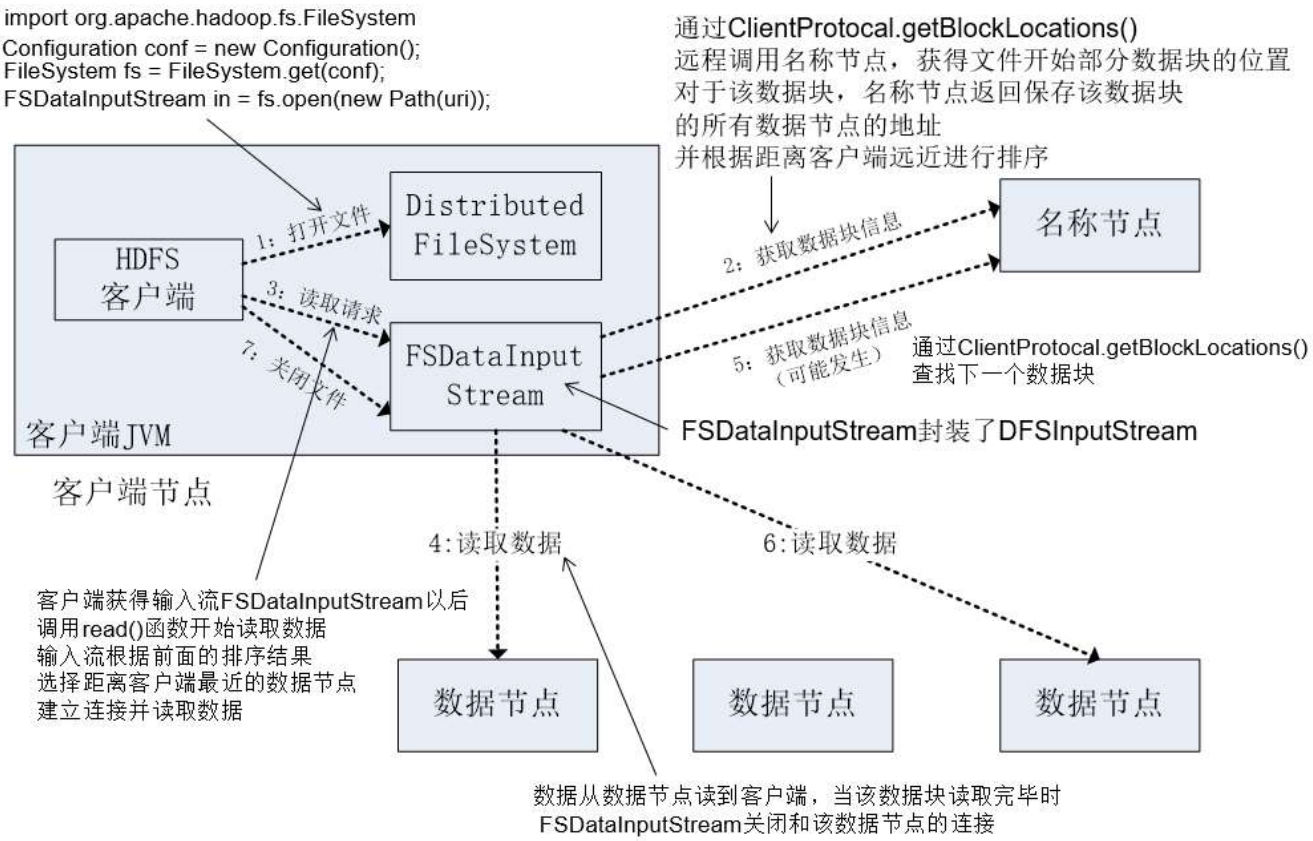
- 兼容廉价的硬件设备
- 流数据读写
- 大数据集
- 简单的文件模型
- 强大的跨平台兼容性

缺点

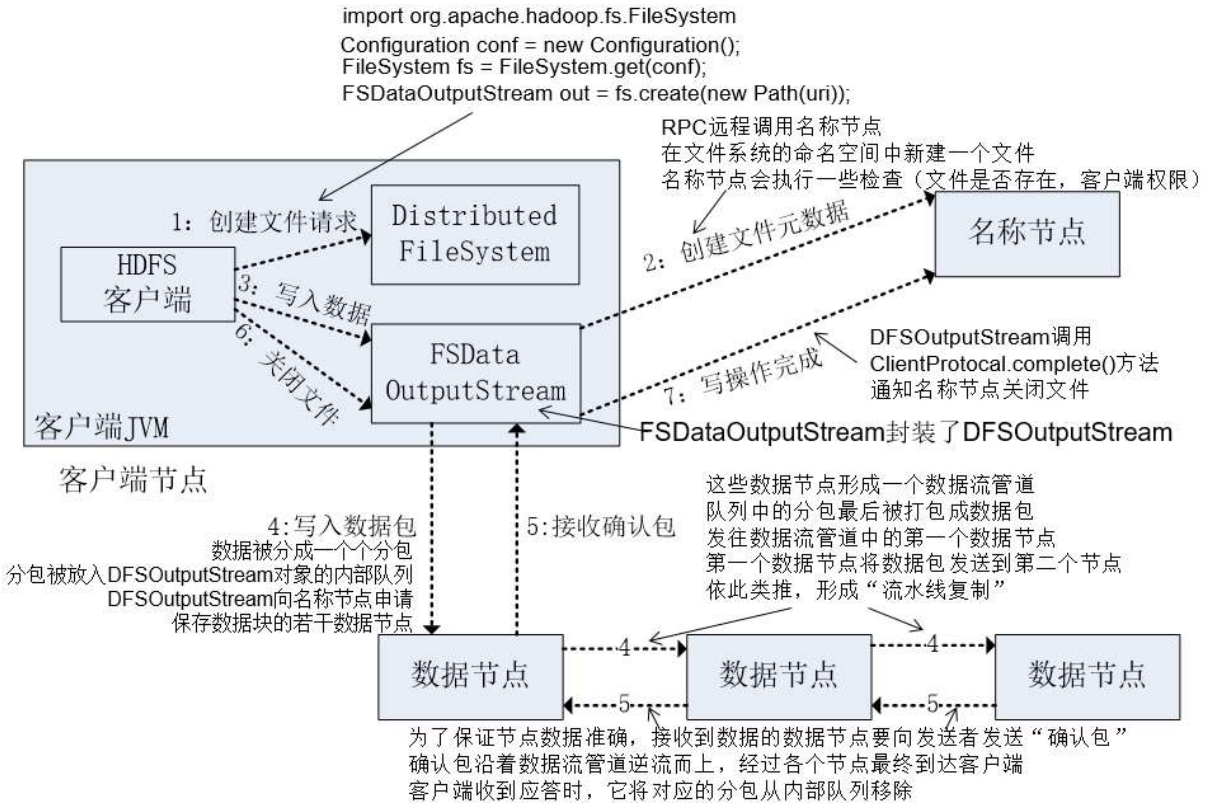
- 不适合低延迟数据访问
- 无法高效存储大量小文件
- 不支持多用户写入及任意修改文件

JavaAPI读/写数据过程

读数据过程

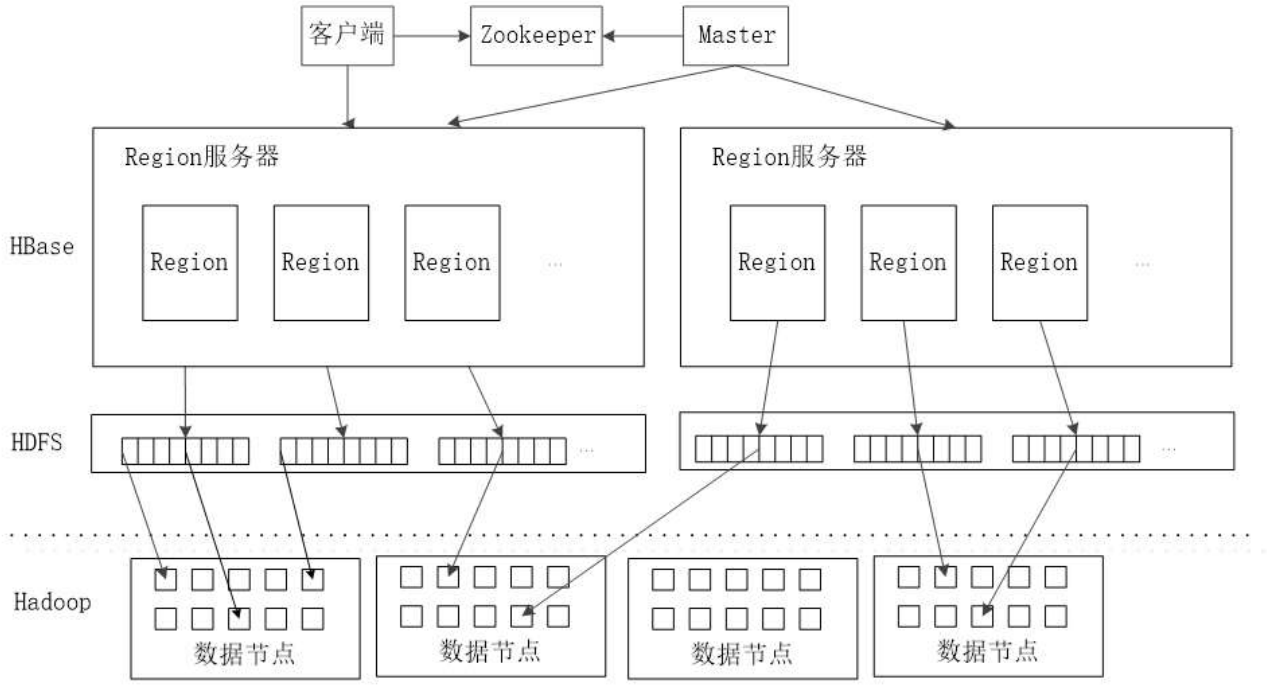


写数据过程



第四章 分布式数据库HBase

实践教程： 大数据原理与应用 第四章 分布式数据库HBase 学习指南



HBase的系统架构

与BigTable对比

-	BigTable	HBase
文件存储系统	GFS	HDFS
海量数据处理	MapReduce	Hadoop、MapReduce
协同服务管理	Chubby	Zookeeper

与传统关系数据库对比

-	传统关系数据库	HBase
数据类型	关系模型，各式丰富	模型简单，统一存储为**未经解释的字符串**
数据操作	操作丰富，涉及多表	只有增删查等简单操作，不存在表与表之间的关系
存储模式	行存储	列存储
数据索引	可以有多个索引	只有一个索引——行键（可以通过其他软件达到建立二级索引的目的）
数据维护	新值覆盖旧值	会保留旧版本
可伸缩性	很难横向扩展，纵向扩展有限	灵活的水平扩展

> 行式存储：适合事务操作（增删改查等）较多的场景
> 列式存储：更适合大数据分析工作（因为大部分分析工作不会同时分析所有的列）

数据模型

- **表**：HBase采用表来组织数据，表由行和列组成，列划分为若干个列族
- **行**：每个HBase表都由若干行组成，每个行由行键（row key）来标识。
- **列族**：一个HBase表被分组成许多“列族”（Column Family）的集合，它是基本的访问控制单元
- **列限定符**：列族里的数据通过列限定符（或列）来定位
- **单元格**：在HBase表中，通过行、列族和列限定符确定一个“单元格”（cell），单元格中存储的数据没有数据类型，总被视为字节数组byte[]
- **时间戳**：每个单元格都保存着同一份数据的多个版本，这些版本采用时间戳进行索引

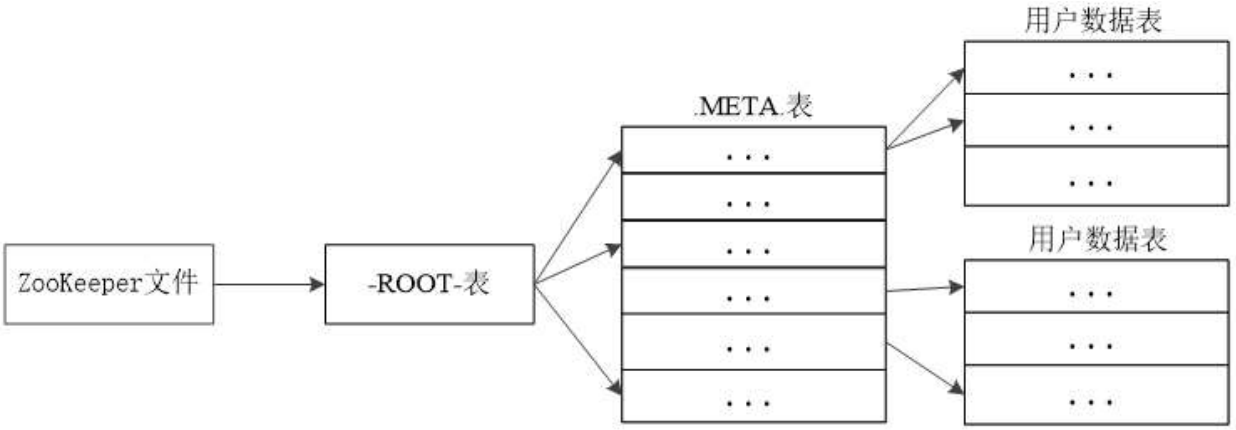
> 概念视图中多个列族集中在一起，为稀疏表；物理视图中不同列族分开，为非稀疏表。

功能组件

1. 库函数：链接到每个客户端
2. 一个Master主服务器，负责管理和维护HBase表的分区信息，维护Region服务器列表，分配Region，负载均衡
3. 许多个Region服务器，负责存储和维护分配给自己的Region，处理来自客户端的读写请求

> 数据读取通过Zookeeper进行，不依赖于Master主服务器

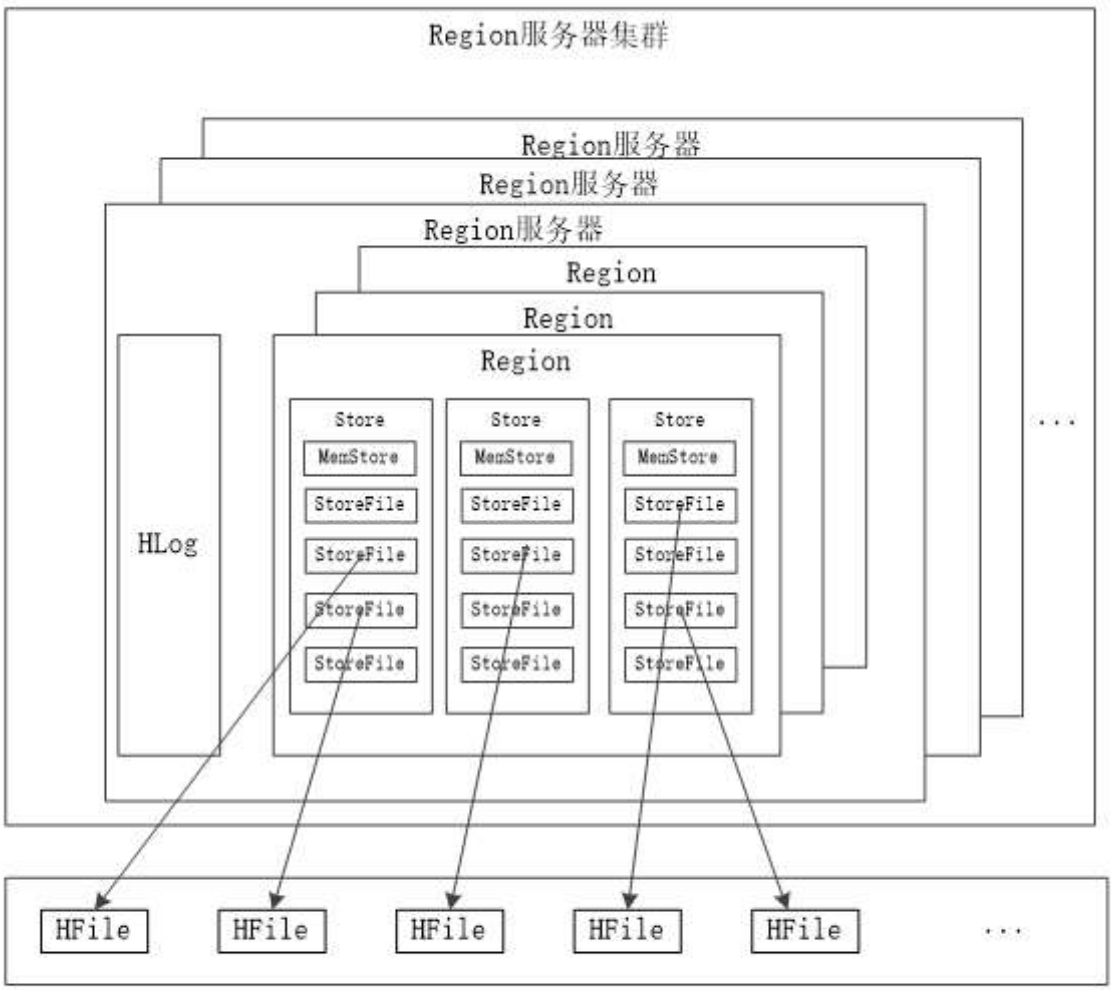
数据定位



层次	名称	作用
第一层	Zookeeper文件	记录了-ROOT-表的位置信息
第二层	-ROOT-表	记录了.META.表的Region位置信息，-ROOT-表只能有一个Region。通过-ROOT-表，就可以访问.META.表中的数据
第三层	.META.表	记录了用户数据表的Region位置信息，.META.表可以有多个Region，保存了HBase中所有用户数据表的Region位置信息

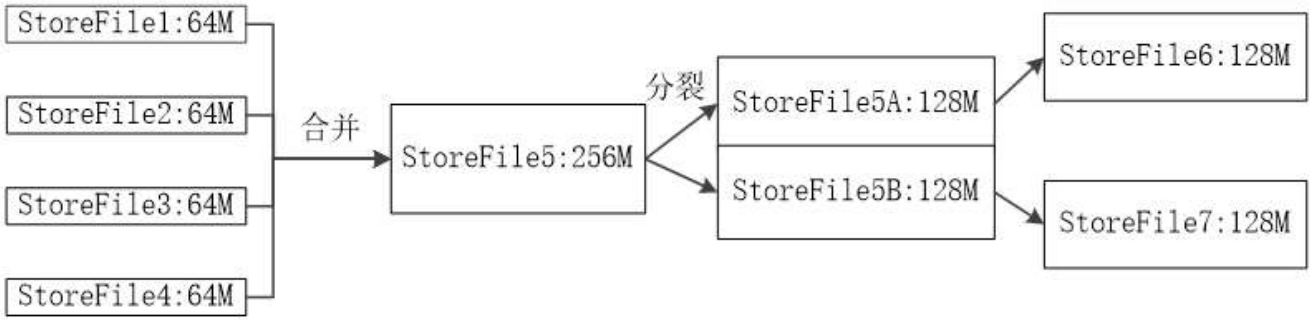
> 1. root表不允许分位多个region，meta表会被分为多个region
> 2. meta表全部放在内存
> 3. 客户端有位置信息缓存（惰性缓存，查找不到时才更新）

工作原理



Region服务器向HDFS文件系统中读写数据

> 一个Region中只使用一个HLog是为了提高正常工作时的写性能，异常时的读操作只是少数情况



StoreFile的合并和分裂过程

> StoreFile的分裂操作对应于Region的分裂

优化方法

1. 行键是按照字典序存储，因此，设计行键时，要充分利用这个排序特点，将经常一起读取的数据存储到一块，将最近可能会被访问的数据放在一块。 举个例子：如果最近写入HBase表中的数据是最可能被访问的，可以考虑将时间戳作为行键的一部分，由于是字典序排序，所以可以使用Long.MAX_VALUE - timestamp作为行键，这样能保证新写入的数据在读取时可以被快速命中。
2. 创建表的时候，可以通过HColumnDescriptor.setInMemory(true)将表放到Region服务器的缓存中，保证在读取的时候被cache命中。
3. 创建表的时候，可以通过HColumnDescriptor.setMaxVersions(int maxVersions)设置表中数据的最大版本，如果只需要保存最新版本的数据，那么可以设置setMaxVersions(1)。
4. 创建表的时候，可以通过HColumnDescriptor.setTimeToLive(int timeToLive)设置表中数据的存储生命期，过期数据将自动被删除，例如如果只需要存储最近两天的数据，那么可以设置setTimeToLive(2 * 24 * 60 * 60)。

第五章 NoSQL数据库

NoSQL与关系数据库的比较

比较标准	RDBMS	NoSQL	备注
数据库原理	完全支持	部分支持	RDBMS有关系代数理论作为基础；NoSQL没有统一的理论基础
数据规模	大	超大	RDBMS很难实现横向扩展，纵向扩展的空间也比较有限，性能会随着数据规模的增大而降低；NoSQL可以很容易通过添加更多设备来支持更大规模的数据
数据库模式	固定	灵活	RDBMS需要定义数据库模式，严格遵守数据定义和相关约束条件；oSQL不存在数据库模式，可以自由灵活定义并存储各种不同类型的数据
查询效率	快	可以实现高效的简单查询，但是不具备高度结构化查询等特性，复杂查询的性能不尽人意	RDBMS借助于索引机制可以实现快速查询（包括记录查询和范围查询）；很多NoSQL数据库没有面向复杂查询的索引，虽然NoSQL可以使用MapReduce来加速查询，但是，在复杂查询方面的性能仍然不如RDBMS
一致性	强一致性	弱一致性	RDBMS严格遵守事务ACID模型，可以保证事务强一致性；很多NoSQL数据库放松了对事务ACID四性的要求，而是遵守BASE模型，只能保证最终一致性
数据完整性	容易实现	很难实现	任何一个RDBMS都可以很容易实现数据完整性，比如通过主键或者非空约束来实现实体完整性，通过主键、外键来实现参照完整性，通过约束或者触发器来实现用户自定义完整性；但是，在NoSQL数据库却无法实现
扩展性	一般	好	RDBMS很难实现横向扩展，纵向扩展的空间也比较有限；NoSQL在设计之初就充分考虑了横向扩展的需求，可以很容易通过添加廉价设备实现扩展
可用性	好	很好	RDBMS在任何时候都以保证数据一致性为优先目标，其次才是优化系统性能，随着数据规模的增大，RDBMS为了保证严格的一致性，只能提供相对较弱的可用性；大多数NoSQL都能提供较高的可用性
标准化	是	否	RDBMS已经标准化（SQL）；NoSQL还没有行业标准，不同的NoSQL数据库都有自己的查询语言，很难规范应用程序接口；StoneBraker认为：NoSQL缺乏统一查询语言，将会拖慢NoSQL发展

比较标准	RDBMS	NoSQL	备注
技术支持	高	低	RDBMS经过几十年的发展，已经非常成熟，Oracle等大型厂商都可以提供很好的技术支持；NoSQL在技术支持方面仍然处于起步阶段，还不成熟，缺乏有力的技术支持
可维护性	复杂	复杂	RDBMS需要专门的数据库管理员(DBA)维护；NoSQL数据库虽然没有DBMS复杂，也难以维护

NoSQL数据类型

键值数据库（Redis）

相关产品	Redis、Riak、SimpleDB、Chordless、Scalaris、Memcached
数据模型	键/值对；键是一个字符串对象；值可以是任意类型的数据，比如整型、字符型、数组、列表、集合等
典型应用	涉及频繁读写、拥有简单数据模型的应用；内容缓存，比如会话、配置文件、参数、购物车等；存储配置和用户数据信息的移动应用
优点	扩展性好，灵活性好，大量写操作时性能高
缺点	无法存储结构化信息，条件查询效率较低
不适用情形	不是通过键而是通过值来查：键值数据库根本没有通过值查询的途径；需要存储数据之间的关系：在键值数据库中，不能通过两个或两个以上的键来关联数据；需要事务的支持：在一些键值数据库中，产生故障时，不可以回滚
使用者	百度云数据库（Redis）、GitHub（Riak）、BestBuy（Riak）、Twitter（Redis和Memcached）、StackOverFlow（Redis）、Instagram （Redis）、Youtube（Memcached）、Wikipedia（Memcached）

列族数据库（BigTable、HBase）

相关产品	BigTable、HBase、Cassandra、HadoopDB、GreenPlum、PNUTS
数据模型	列族
典型应用	分布式数据存储与管理；数据在地理上分布于多个数据中心的应用程序；可以容忍副本中存在短期不一致情况的应用程序；拥有动态字段的应用程序；拥有潜在大量数据的应用程序，大到几百TB的数据
优点	查找速度快，可扩展性强，容易进行分布式扩展，复杂性低
缺点	功能较少，大都不支持强事务一致性
不适用情形	需要ACID事务支持的情形，Cassandra等产品就不适用
使用者	Ebay（Cassandra）、Instagram（Cassandra）、NASA（Cassandra）、Twitter（Cassandra and HBase）、Facebook（HBase）、Yahoo!（HBase）

文档数据库（MongoDB）

相关产品	MongoDB、CouchDB、Terrastore、ThruDB、RavenDB、SisoDB、RaptorDB、CloudKit、Perservere、Jackrabbit
数据模型	键/值；值（value）是版本化的文档
典型应用	存储、索引并管理面向文档的数据或者类似的半结构化数据，比如，用于后台具有大量读写操作的网站、使用JSON数据结构的应用、使用嵌套结构等非规范化数据的应用程序
优点	性能好（高并发），灵活性高，复杂性低，数据结构灵活；提供嵌入式文档功能，将经常查询的数据存储在同一个文档中；既可以根据键来构建索引，也可以根据内容构建索引
缺点	缺乏统一的查询语法
不适用情形	在不同的文档上添加事务。文档数据库并不支持文档间的事务，如果对这方面有需求则不应该选用这个解决方案
使用者	百度云数据库（MongoDB）、SAP （MongoDB）、Codecademy （MongoDB）、Foursquare （MongoDB）、NBC News （RavenDB）

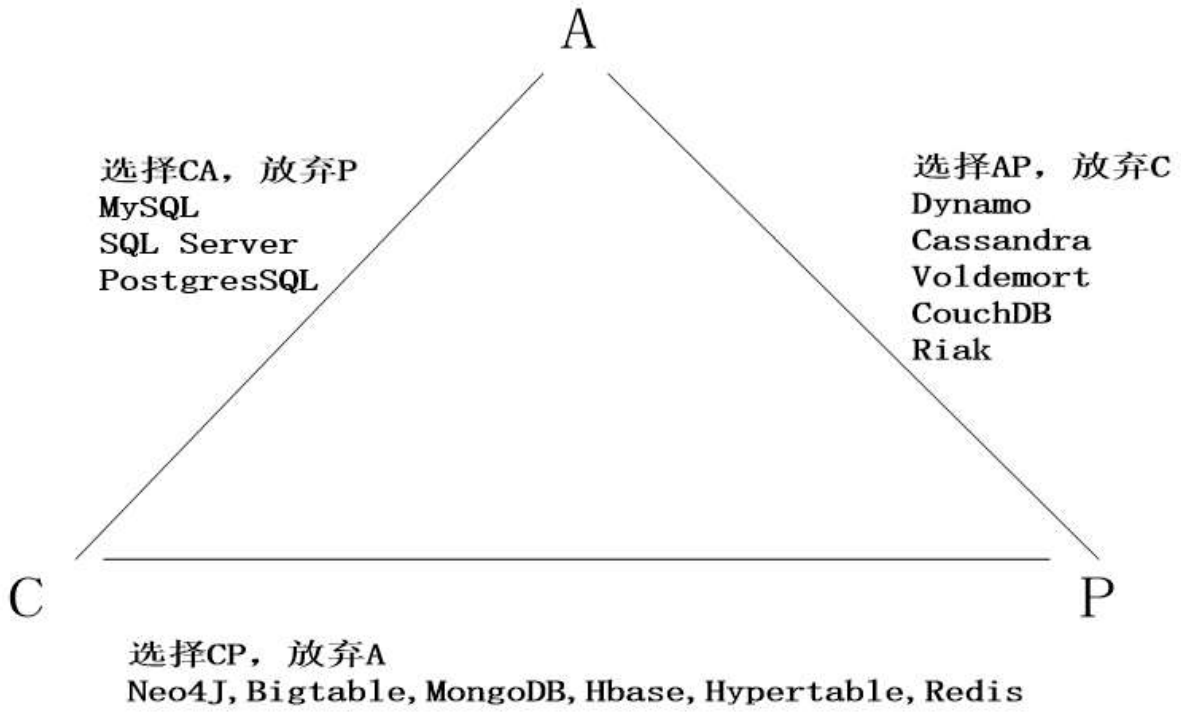
图形数据库

相关产品	Neo4J、OrientDB、InfoGrid、Infinite Graph、GraphDB
数据模型	图结构
典型应用	专门用于处理具有高度相互关联关系的数据，比较适合于社交网络、模式识别、依赖分析、推荐系统以及路径寻找等问题
优点	灵活性高，支持复杂的图形算法，可用于构建复杂的关系图谱
缺点	复杂性高，只能支持一定的数据规模
使用者	Adobe（Neo4J）、Cisco（Neo4J）、T-Mobile（Neo4J）

CAP

所谓的CAP指的是：

- C（Consistency）：一致性，是指任何一个读操作总是能够读到之前完成的写操作的结果，也就是在分布式环境中，多点的数据是一致的，或者说，所有节点在同一时间具有相同的数据
- A:（Availability）：可用性，是指快速获取数据，可以在确定的时间内返回操作结果，保证每个请求不管成功或者失败都有响应；
- P（Tolerance of Network Partition）：分区容忍性，是指当出现网络分区的情况时（即系统中的一部分节点无法和其他节点进行通信），分离的系统也能够正常运行，也就是说，系统中任意信息的丢失或失败不会影响系统的继续运作。



第六章 云数据库

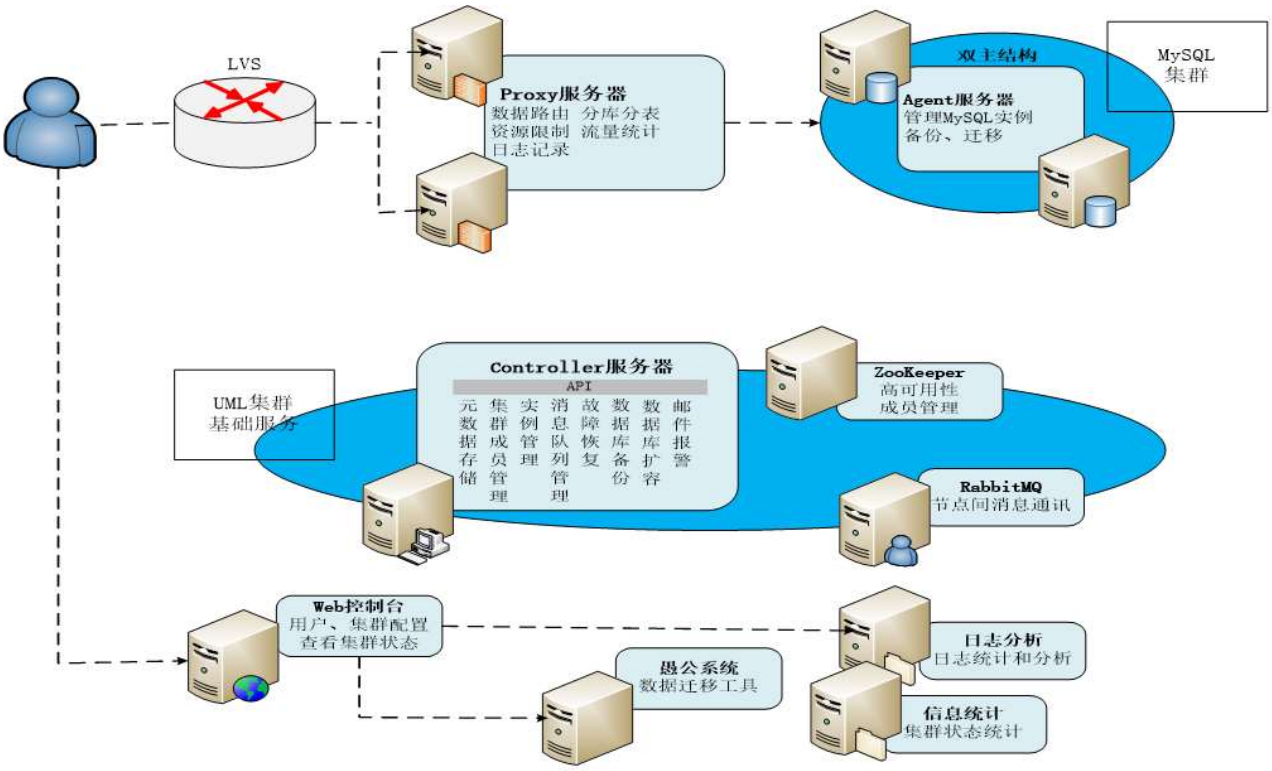
云数据库特性

- 动态可扩展
- 高可用性
- 较低的使用代价
- 易用性
- 高性能
- 免维护
- 安全

云数据库产品

企业	产品
Amazon	Dynamo、SimpleDB、RDS
Google	Google Cloud SQL
Microsoft	Microsoft SQL Azure
Oracle	Oracle Cloud
Yahoo!	PNUTS
Vertica	Analytic Database v3.0 for the Cloud
EnerpriseDB	Postgres Plus in the Cl
阿里	阿里云RDS
百度	百度云数据库
腾讯	腾讯云数据库

云数据库系统架构--阿里UMP（Unified MySQL Platform）系统为例



第七章 MapReduce

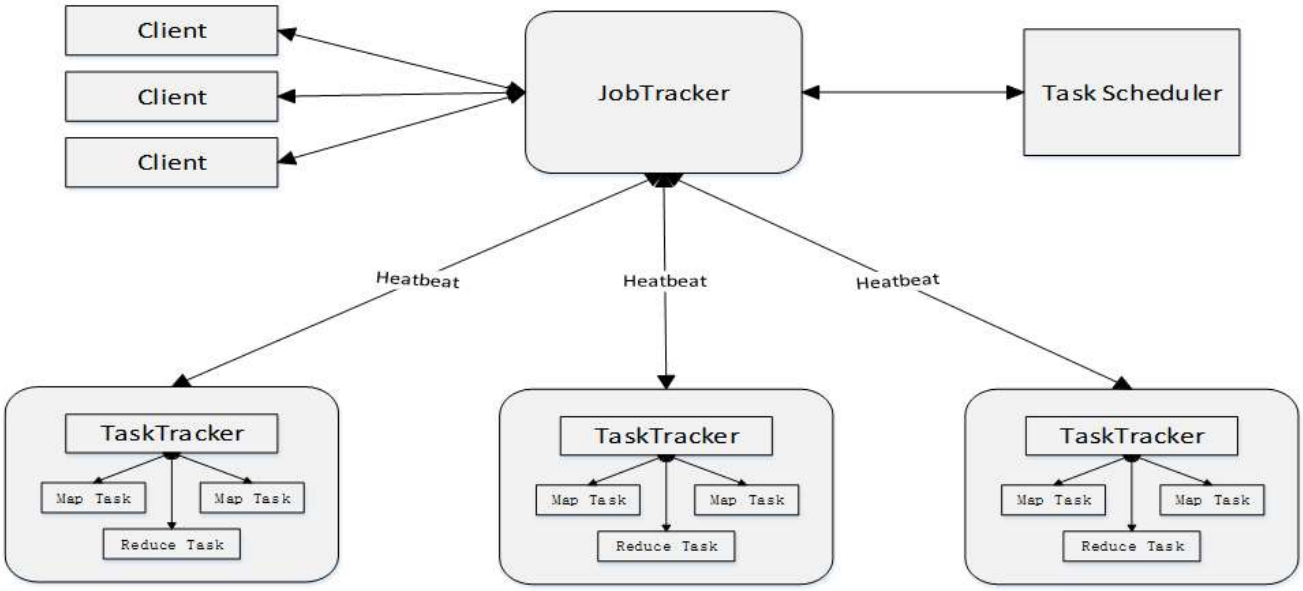
与传统并行计算框架的区别

-	传统并行计算框架	MapReduce
集群架构/容错性	共享式(共享内存/共享存储)，容错性差	非共享式，容错性好
硬件/价格/扩展性	刀片服务器、高速网、SAN，价格贵，扩展性差	普通PC机，便宜，扩展性好
编程/学习难度	what-how，难	what，简单
适用场景	实时、细粒度计算、计算密集型	批处理、非实时、数据密集型

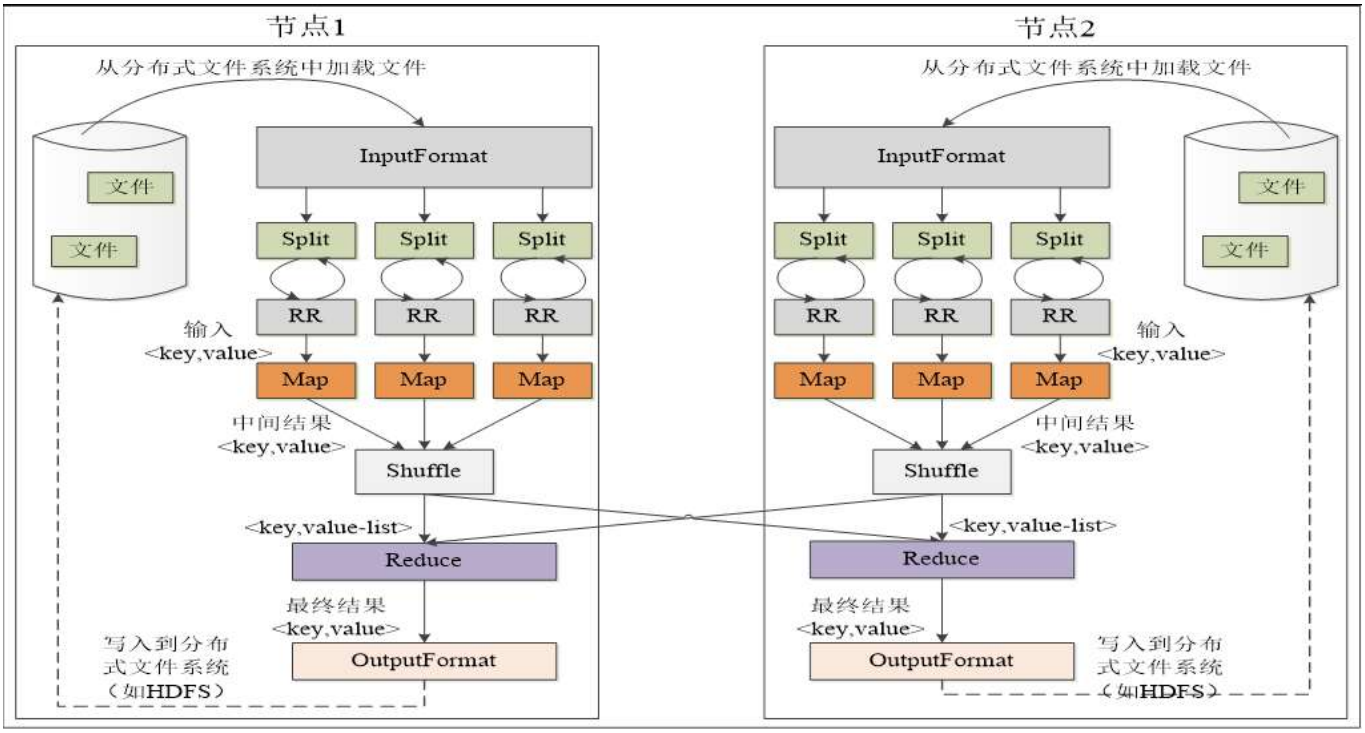
两个重要的理念

- 分而治之
- 计算向数据靠拢

体系结构



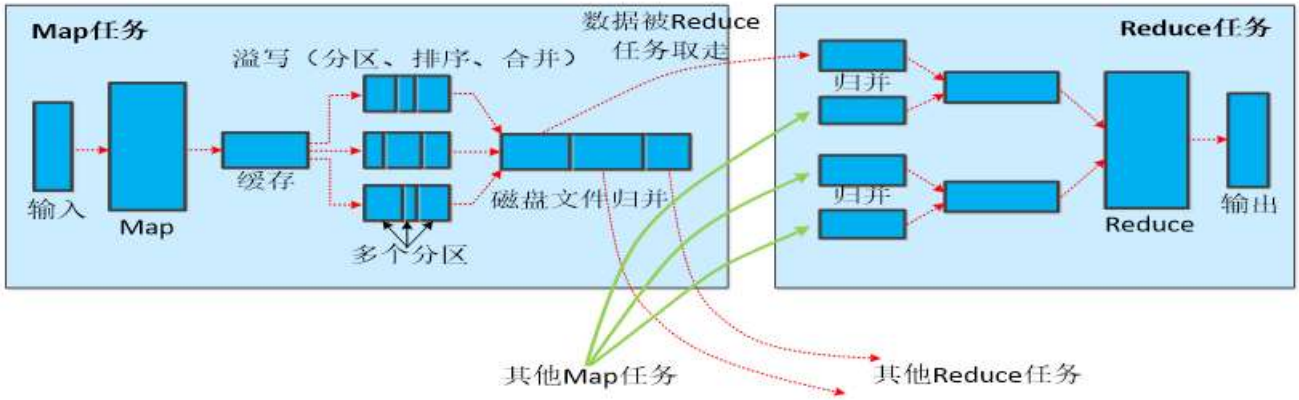
执行过程



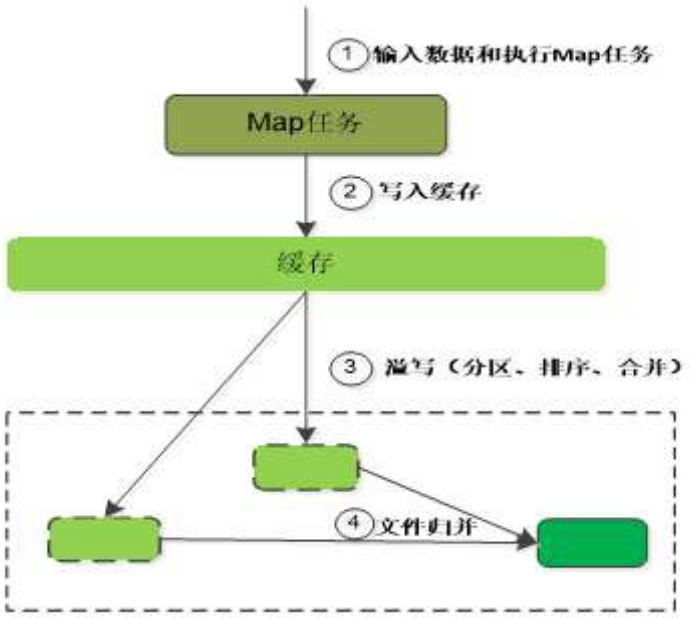
- split理想分片大小为一个HDFS块
- 最优的Reduce任务个数取决于集群中可用的reduce任务槽(slot)的数目

Shuffle过程

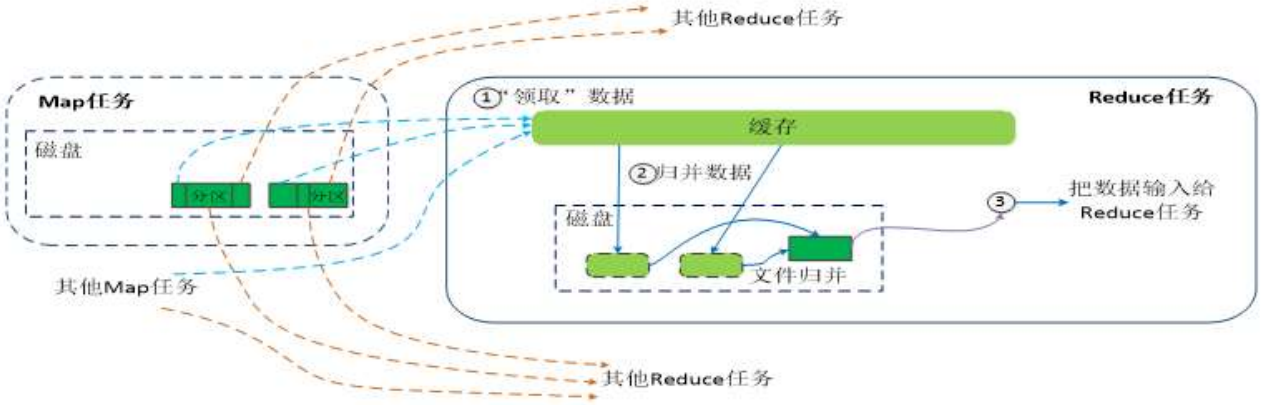
> 合并：多个<key, value>通过combiner函数计算得到一个<key, value>(执行过程中不一定会有这个操作)
> 归并：多个<key, value>生成一个<key, List<value>>



map shuffle过程

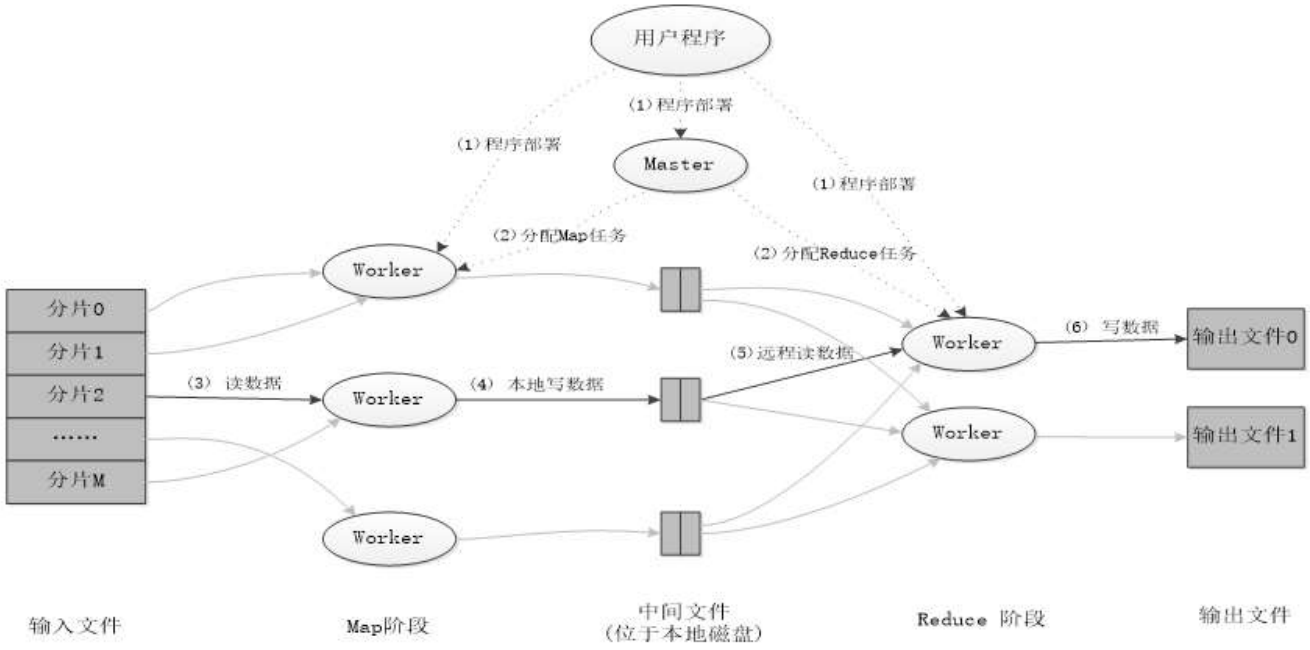


reduce shuffle过程



MapReduce应用程序执行过程

> map和reduce操作中间文件存储在本地磁盘，不放入分布式文件系统



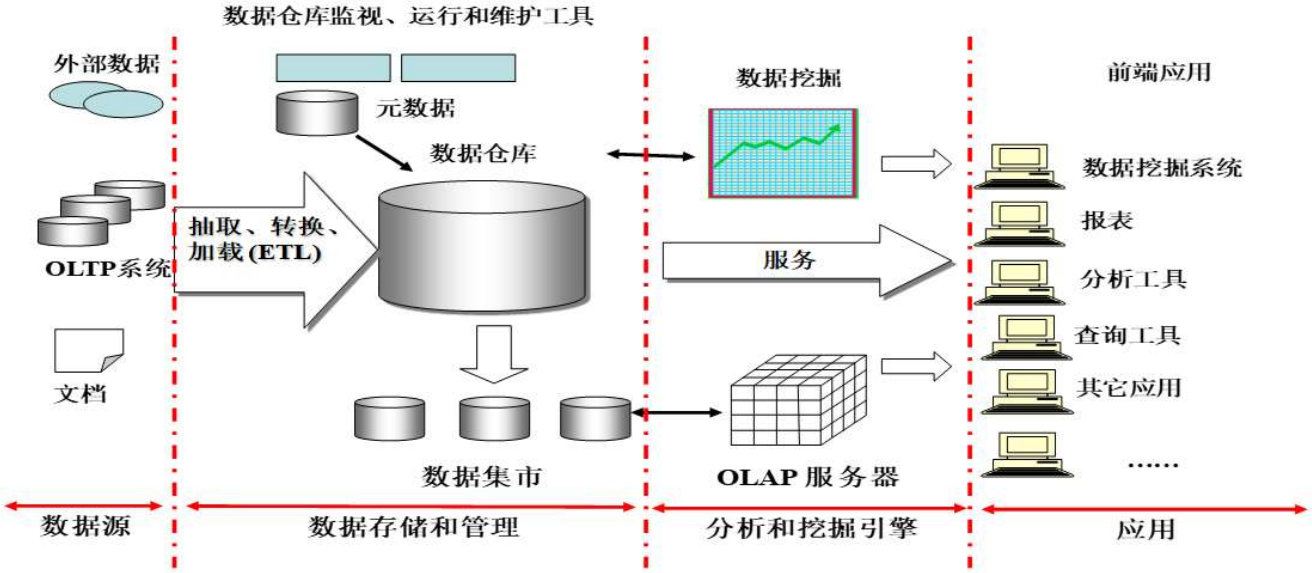
MapReduce任务执行的几种方式

1. Hadoop jar
2. Pig
3. Hive
4. Python
5. Shell脚本

第八章 基于Hadoop的数据仓库Hive

数据仓库的概念

数据仓库（Data Warehouse）是一个面向主题的（Subject Oriented）、集成的（Integrated）、相对稳定的（Non-Volatile）、反映历史变化（Time Variant）的数据集合，用于支持管理决策。



与Hadoop其他组件的关系

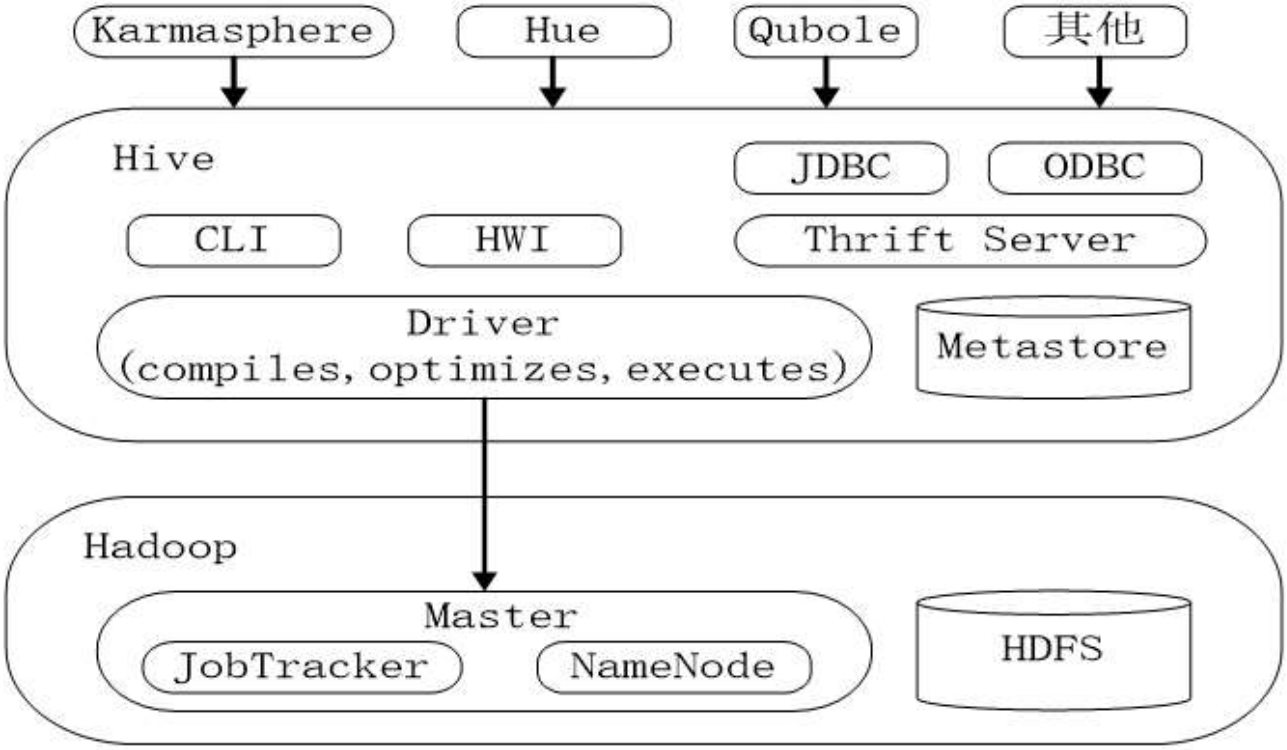
- Hive依赖于HDFS 存储数据
- Hive依赖于MapReduce 处理数据
- 在某些场景下Pig可以作为Hive的替代工具
- HBase 提供数据的实时访问

与传统数据库对比

对比项目	Hive	传统数据库
数据插入	支持批量导入	支持单条和批量导入
数据更新	不支持	支持
索引	支持	支持
分区	支持	支持
执行延迟	高	低
扩展性	好	有限

系统架构

- 用户接口模块包括CLI、HWI、JDBC、ODBC、Thrift Server
- 驱动模块（Driver）包括编译器、优化器、执行器等，负责把HiveSQL语句转换成一系列MapReduce作业
- 元数据存储模块（Metastore）是一个独立的关系型数据库（自带derby数据库，或MySQL数据库）



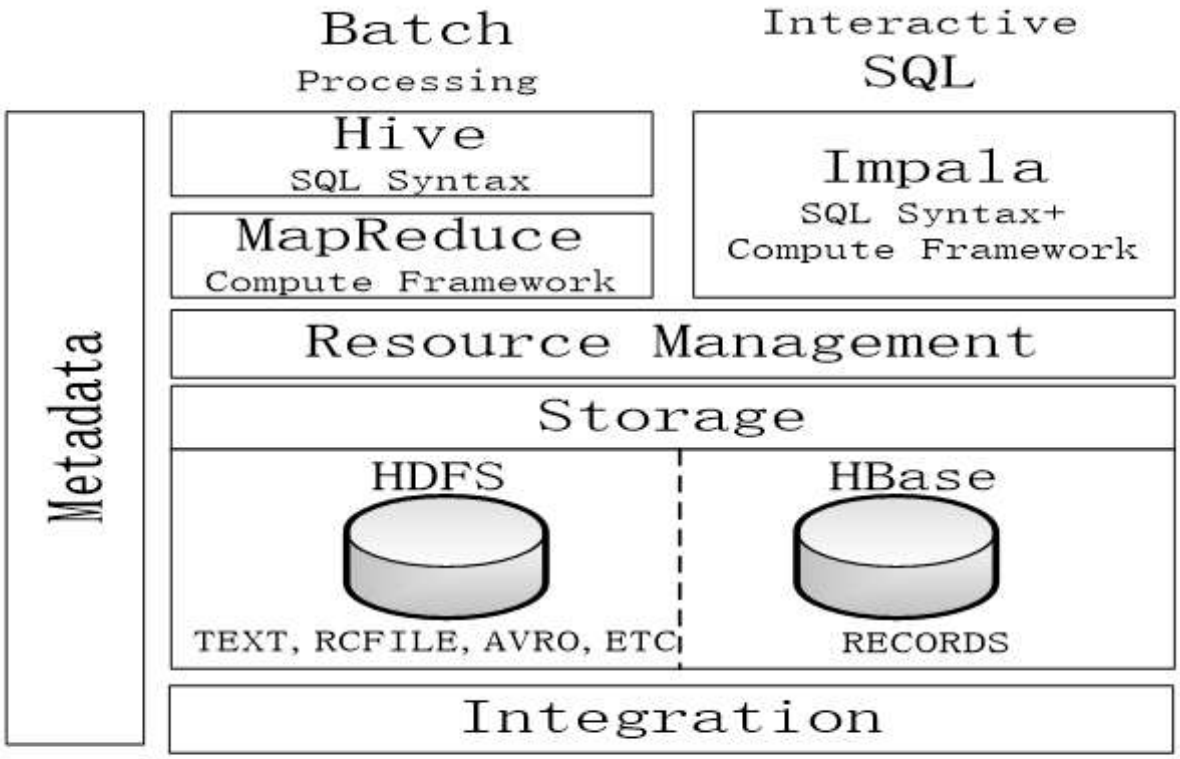
Impala和Hive对比

相同点

1. Hive与Impala使用相同的存储数据池，都支持把数据存储于HDFS和HBase中
2. Hive与Impala使用相同的元数据
3. Hive与Impala中对SQL的解释处理比较相似，都是通过词法分析生成执行计划

不同点

1. Hive适合于长时间的批处理查询分析，而Impala适合于实时交互式SQL查询
2. Hive依赖于MapReduce计算框架，Impala把执行计划表现为一棵完整的执行计划树，直接分发执行计划到各个Impalad执行查询
3. Hive在执行过程中，如果内存放不下所有数据，则会使用外存，以保证查询能顺序执行完成，而Impala在遇到内存放不下数据时，不会利用外存，所以Impala目前处理查询时会受到一定的限制



Impala与Hive的对比

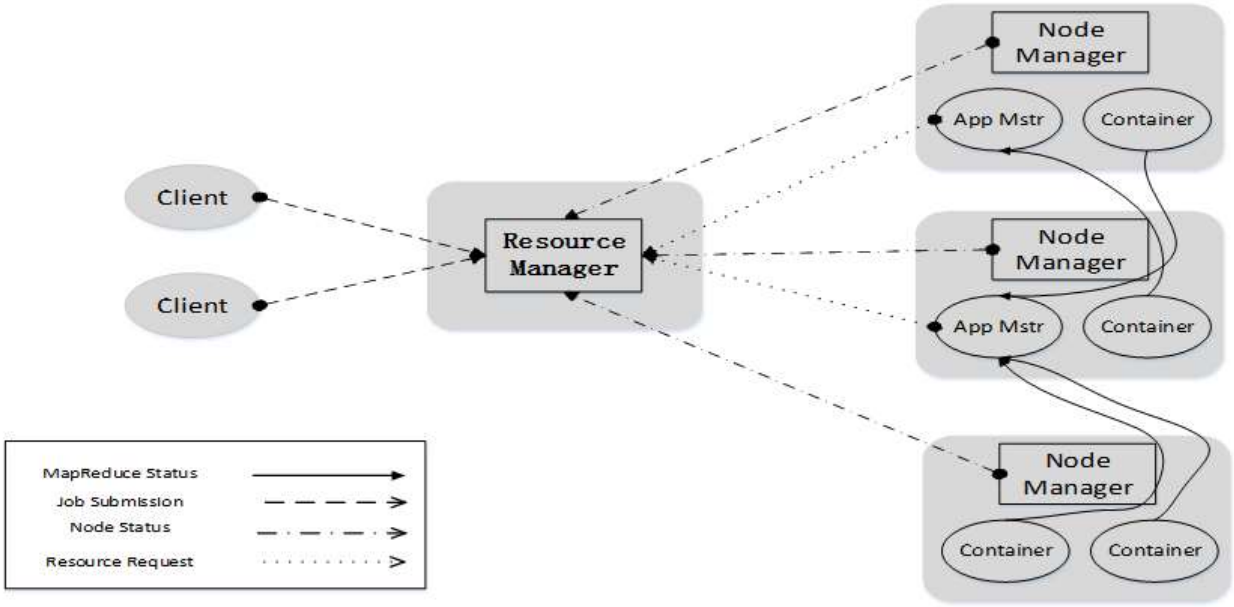
第九章 Hadoop架构再探讨

Hadoop架构从1.0到2.0的改进

组件	Hadoop1.0的问题	Hadoop2.0的改进
HDFS	单一名称节点，存在单点失效问题	设计了HDFS HA，提供名称节点热备机制
HDFS	单一命名空间，无法实现资源隔离	设计了HDFS Federation管理多个命名空间
MapReduce	资源管理效率低	设计了新的资源管理框架YARN

YARN

> YARN的目标就是实现“一个集群多个框架”，即在一个集群上部署一个统一的资源调度管理框架YARN，在YARN之上可以部署其他各种计算框架



- ResourceManager
  处理客户端请求
  启动/监控ApplicationMaster
  监控NodeManager
  资源分配与调度
- NodeManager
  单个节点上的资源管理
  处理来自ResourceManger的命令
  处理来自ApplicationMaster的命令
- ApplicationMaster
  为应用程序申请资源，并分配给内部任务
  任务调度、监控与容错

第十章 Spark

特点

- 运行速度快：使用DAG执行引擎以支持循环数据流与内存计算
- 容易使用：支持使用Scala、Java、Python和R语言进行编程，可以通过Spark Shell进行交互式编程
- 通用性：Spark提供了完整而强大的技术栈，包括SQL查询、流式计算、机器学习和图算法组件
- 运行模式多样：可运行于独立的集群模式中，可运行于Hadoop中，也可运行于AmazonEC2等云环境中，并且可以访问HDFS、Cassandra、HBase、Hive等多种数据源

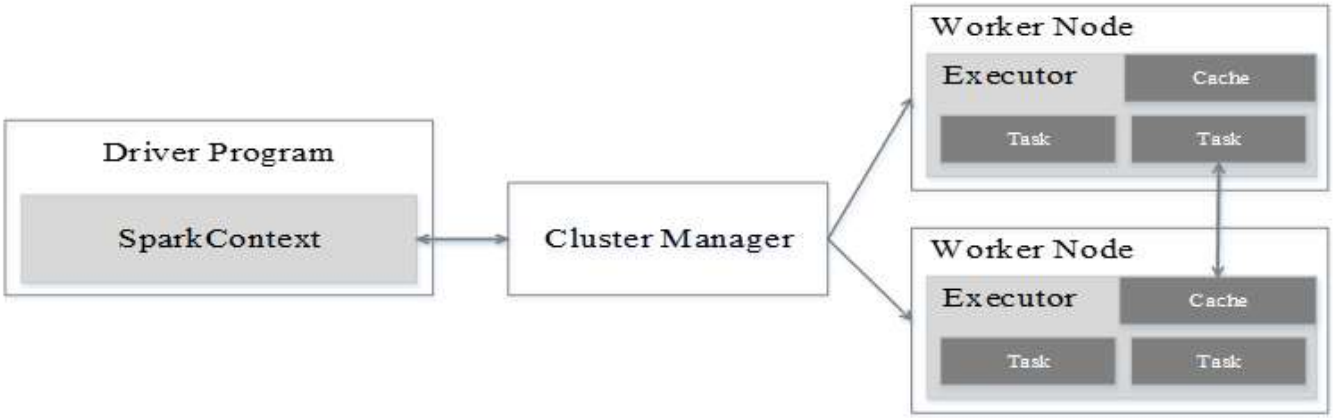
Spark组件与Hadoop生态中的其他框架对应关系

应用场景	时间跨度	其他框架	Spark生态系统中的组件
复杂的批量数据处理	小时级	MapReduce、Hive	Spark
基于历史数据的交互式查询	分钟级、秒级	Impala、Dremel、Drill	Spark SQL
基于实时数据流的数据处理	毫秒、秒级	Storm、S4	Spark Streaming
基于历史数据的数据挖掘	-	Mahout	MLlib
图结构数据的处理	-	Pregel、Hama	GraphX

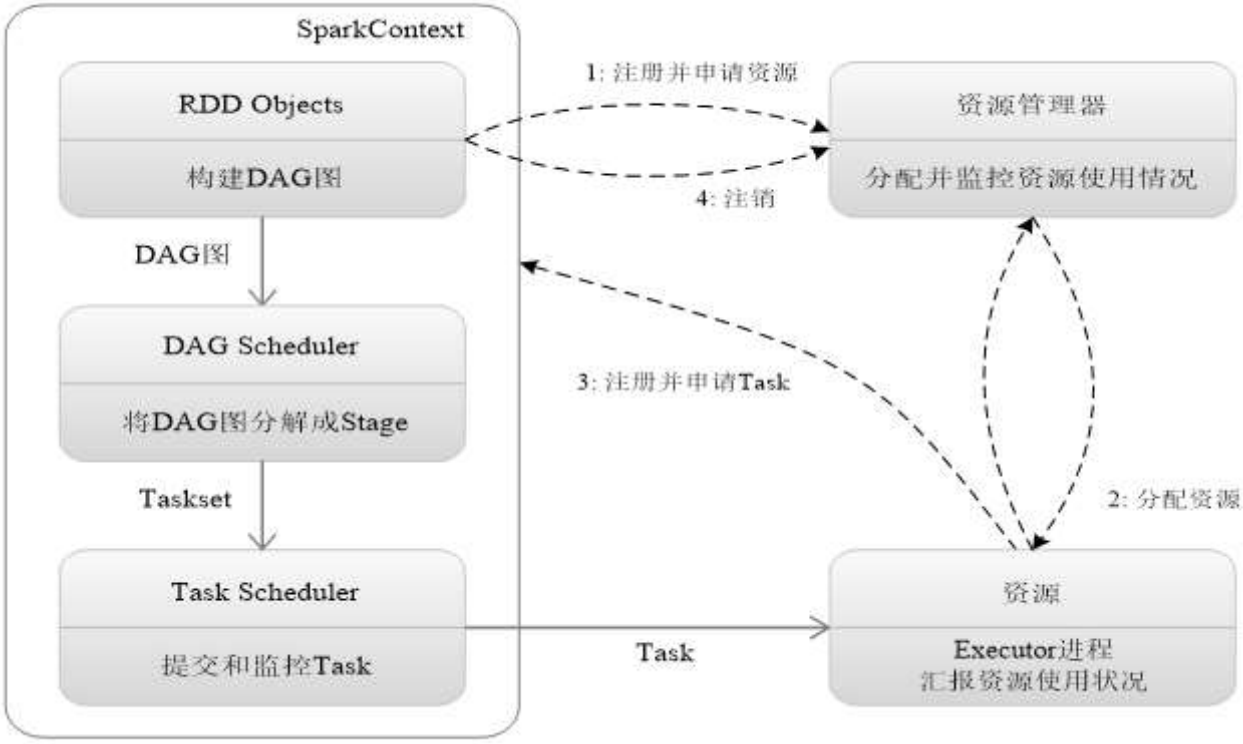
组成

- RDD：是Resillient Distributed Dataset（弹性分布式数据集）的简称，是分布式内存的一个抽象概念，提供了一种高度受限的共享内存模型
- DAG：是Directed Acyclic Graph（有向无环图）的简称，反映RDD之间的依赖关系
- Executor：是运行在工作节点（WorkerNode）的一个进程，负责运行Task
- Application：用户编写的Spark应用程序
- Task：运行在Executor上的工作单元
- Job：一个Job包含多个RDD及作用于相应RDD上的各种操作
- Stage：是Job的基本调度单位，一个Job会分为多组Task，每组Task被称为Stage，或者也被称为TaskSet，代表了一组关联的、相互之间没有Shuffle依赖关系的任务组成的任务集

运行框架

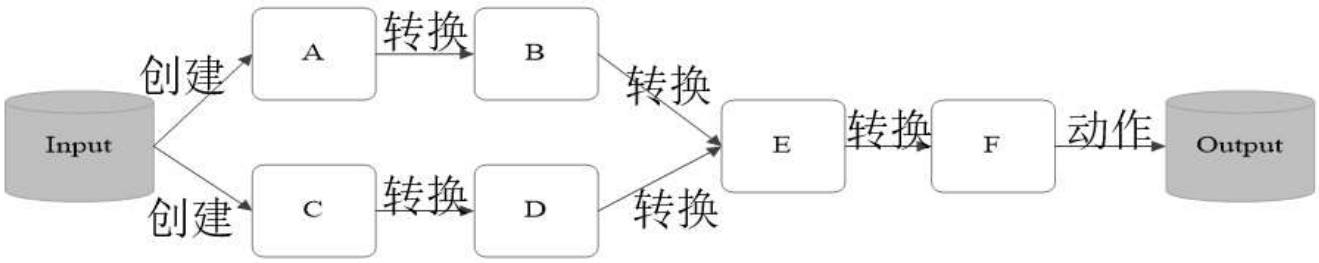


工作流程



RDD

- RDD的转换过程如下图，首先输入数据创建RDD，之后经过转换到达最终的RDD，最后输出结果。
- 整个执行过程中间不会产生输出数据，前一个RDD的结果作为后一个RDD的输入。
- RDD所有分区分为窄依赖和宽依赖两种，窄依赖表现为一个父RDD的分区对应于一个子RDD的分区或多个父RDD的分区对应于一个子RDD的分区，宽依赖则表现为存在一个父RDD的一个分区对应一个子RDD的多个分区。
- 所有RDD分区由宽依赖切分成多个Stage，所有资源按照Stage划分，隔离运行。



第十一章 流计算

概括

流计算秉承一个基本理念，即**数据的价值随着时间的流逝而降低**，如用户点击流。因此，当事件出现时就应该立即进行处理，而不是缓存起来进行批量处理。为了及时处理流数据，就需要一个低延迟、可扩展、高可靠的处理引擎。

流计算处理流程示意图

Storm和Hadoop架构组件功能对应关系

-	Hadoop	Storm
应用名称	Job	Topology
系统角色	JobTracker/TaskTracker	Nimbus/Supervisor
组件接口	Map/Reduce	Spout/Bolt

MapReduce批处理架构和Samza流处理架构的类比

-	MapReduce批处理框架	Samza流处理架构
数据层	HDFS	Kafka
执行层	YARN	YARN
处理层	MapReduce	Samza API

Storm、Spark Streaming、Samza三个流计算框架选型对比

- 从**编程的灵活性**来讲，**Storm**是比较理想的选择，它使用Apache Thrift，可以用任何编程语言来编写拓扑结构（Topology）
- 当需要在一个集群中把**流计算和图计算、机器学习、SQL查询分析等进行结合**时，可以选择**Spark Streaming**，因为，在Spark上可以统一部署Spark SQL，Spark Streaming、MLlib，GraphX等组件，提供便捷的一体化编程模型
- 当**有大量的状态需要处理**时，比如每个分区都有数十亿个元组，则可以选择**Samza**。当应用场景需要**毫秒级响应**时，可以选择**Storm和Samza**，因为Spark Streaming无法实现毫秒级的流计算