云计算

第3讲

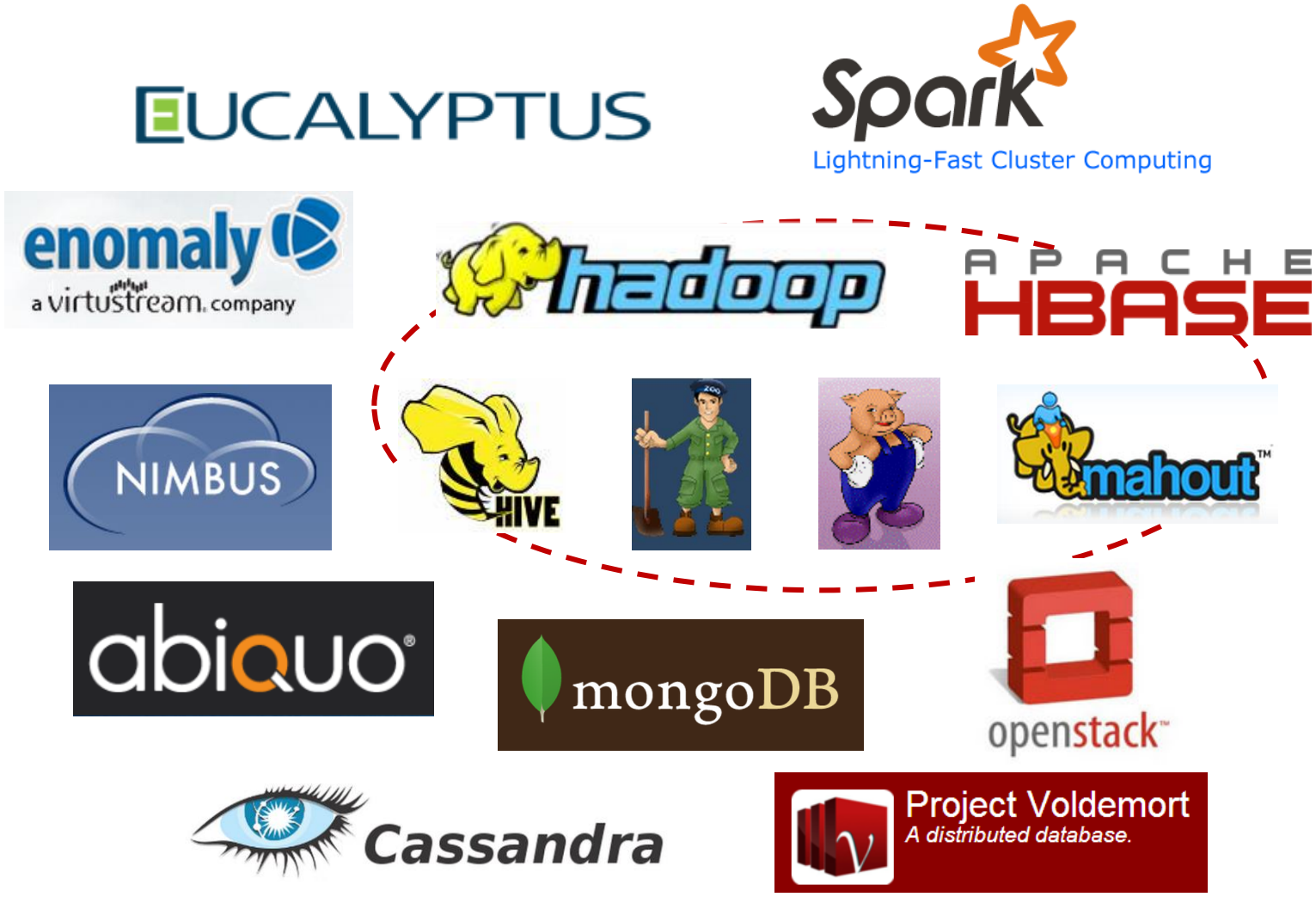
开源云平台

任桐炜，李传艺

南京大学软件学院

2017-09-20

丰富的开源云平台


EUCALYPTUS
Spark
Lightning-Fast Cluster Computing
enomaly
a virtustream company
hadoop
APACHE
HBASE
NIMBUS
HIVE
mahout
abiquo
mongoDB
openstack
Cassandra
Project Voldemort
A distributed database.

Hadoop

- Apache支持，Java语言，参照Google
- 组成：
  - Hadoop Common：整个项目的核心
  - HDFS：提供高吞吐量的分布式文件系统（对应GFS）
  - Hadoop YARN：任务调度和集群资源管理框架
  - Hadoop MapReduce：基于YARN的大型数据的分布式处理系统（对应MapReduce）
- 相关项目
  - Ambari：基于web的Hadoop集群监控系统
  - Avro：数据序列化系统
  - Cassandra：无单点故障的多主节点数据库
  - Chukwa：用于管理大型分布式系统的数据采集系统
  - HBase：支持结构化数据存储的分布式数据库（对应BigTable）
  - Hive：提供数据摘要和查询功能的数据仓库
  - Mahout：机器学习和数据挖掘库
  - Pig：可并行计算的高级数据流语言和框架
  - ZooKeeper：解决分布式系统中的一致性问题（模拟Chubby）

Hadoop项目网址

What Is Apache Hadoop?

The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing.

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

The project includes these modules:

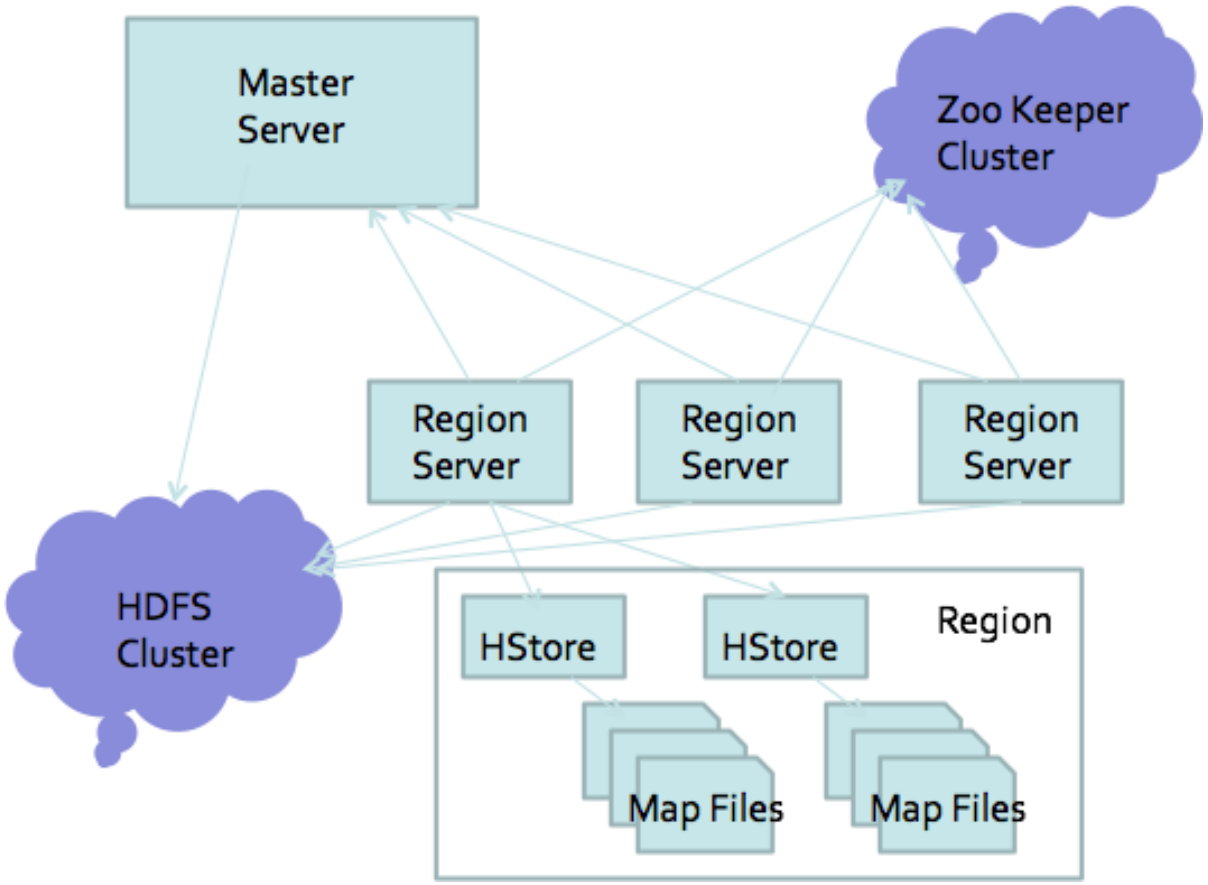
- **Hadoop Common**: The common utilities that support the other Hadoop modules.
- **Hadoop Distributed File System (HDFS™)**: A distributed file system that provides high-throughput access to application data.
- **Hadoop YARN**: A framework for job scheduling and cluster resource management.
- **Hadoop MapReduce**: A YARN-based system for parallel processing of large data sets.

Other Hadoop-related projects at Apache include:

- **Ambari™**: A web-based tool for provisioning, managing, and monitoring Apache Hadoop clusters which includes support for Hadoop HDFS, Hadoop MapReduce, Hive, HCatalog, HBase, ZooKeeper, Oozie, Pig and Sqoop. Ambari also provides a dashboard for viewing cluster health such as heatmaps and ability to view MapReduce, Pig and Hive applications visually alongwith features to diagnose their performance characteristics in a user-friendly manner.
- **Avro™**: A data serialization system.
- **Cassandra™**: A scalable multi-master database with no single points of failure.
- **Chukwa™**: A data collection system for managing large distributed systems.
- **HBase™**: A scalable, distributed database that supports structured data storage for large tables.
- **Hive™**: A data warehouse infrastructure that provides data summarization and ad hoc querying.
- **Mahout™**: A Scalable machine learning and data mining library.
- **Pig™**: A high-level data-flow language and execution framework for parallel computation.
- **Spark™**: A fast and general compute engine for Hadoop data. Spark provides a simple and expressive programming model that supports a wide range of applications, including ETL, machine learning, stream processing, and graph computation.
- **Tez™**: A generalized data-flow programming framework, built on Hadoop YARN, which provides a powerful and flexible engine to execute an arbitrary DAG of tasks to process data for both batch and interactive use-cases. Tez is being adopted by Hive™, Pig™ and other frameworks in the Hadoop ecosystem, and also by other commercial software (e.g. ETL tools), to replace Hadoop™ MapReduce as the underlying execution engine.
- **ZooKeeper™**: A high-performance coordination service for distributed applications.

http://hadoop.apache.org/

系统总体架构

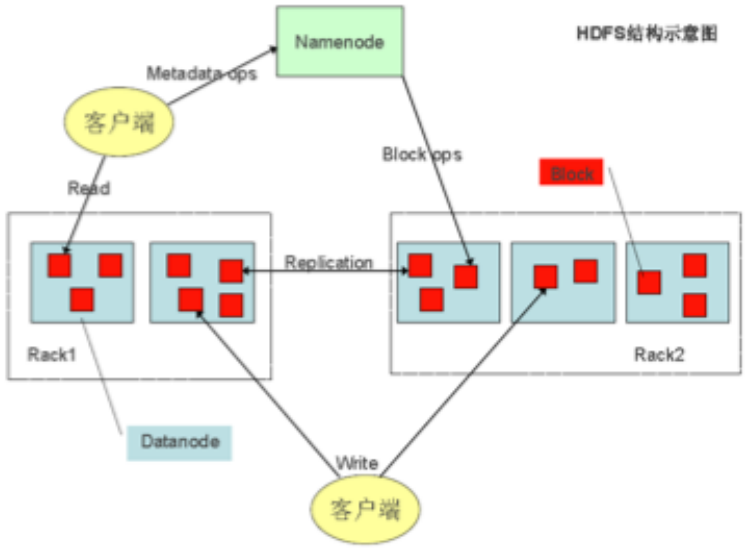

Master Server
Zoo Keeper Cluster
Region Server
Region Server
Region Server
HDFS Cluster
HStore
HStore
Region
Map Files
Map Files

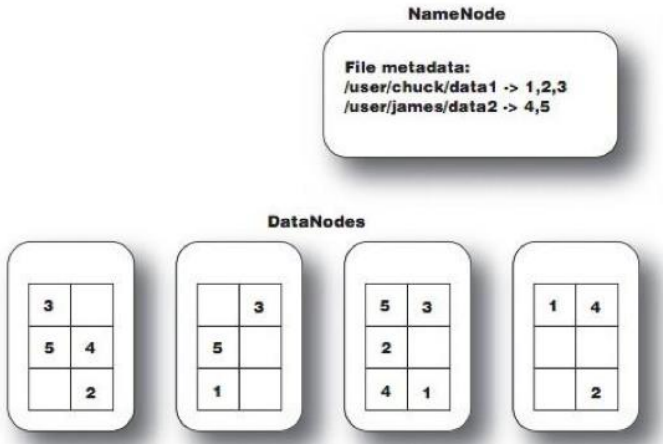
HDFS设计目标和前提

- 目标
  - 部署在廉价硬件之上，能够高容错、可靠地存储海量数据
- 前提
  - 硬件错误是常态而不是异常
  - 流式数据访问
  - 大规模数据集
  - 简单一致性模型
  - 移动计算比移动数据更划算
  - 异构软硬件平台间的可移植性

HDFS体系结构

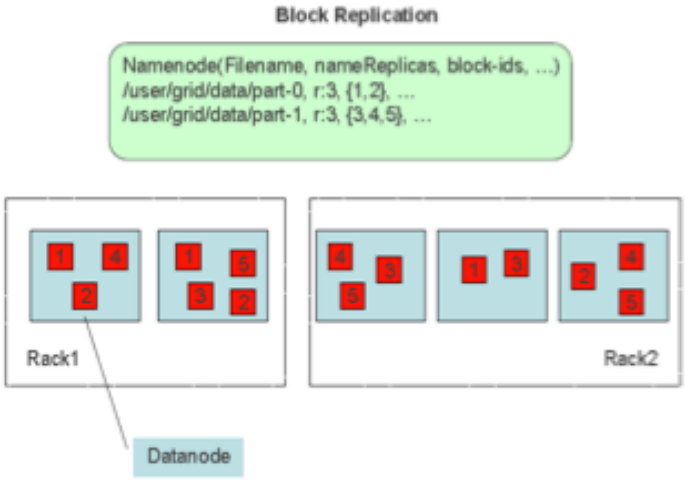
- 主从结构
  - **NameNode（1个）：** 管理文件系统的元数据
  - **DataNode（多个）：** 存储实际的数据





HDFS可靠性保障

- 冗余备份和副本存放
  - 每个数据块有一定数量的副本（副本数可配置）
  - 机架感知策略：本地+同机架+不同机架（开发中）
- 心跳检测和安全模式
  - **NameNode**周期性从每个**DataNode**接受心跳包（是否死机）和块报告（安全模式下检测）
- 数据完整性检测
  - 将校验和存在隐藏文件中
- 空间回收
- 元数据备份
  - 依赖映像文件和日志



HDFS性能提升

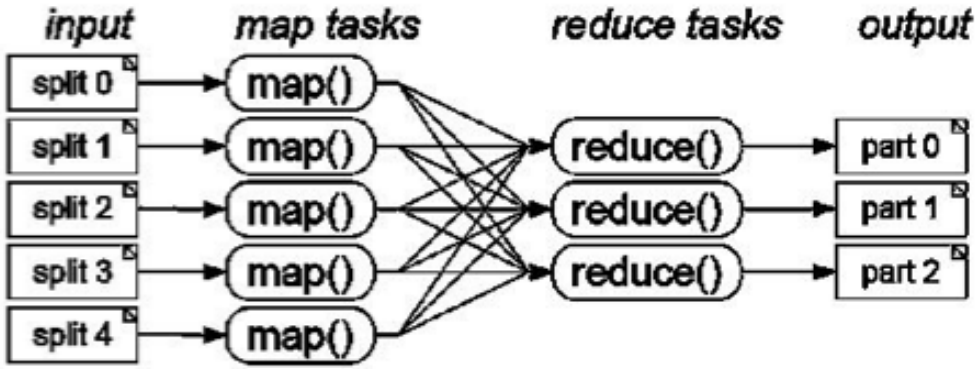
- 副本选择
  - 使用离程序最近的副本进行读操作
- 负载均衡
  - 当剩余磁盘空间很少时自动迁移数据到其它节点（待实现）
- 客户端缓存
  - 临时文件超过**64MB**时才正式写入数据
- 流水线复制
  - 客户端**=>**第一个**DataNode=>**第二个**DataNode=>**第三个**DataNode**

HDFS和GFS比较

	HDFS	GFS	原因
中心服务器模式	单一中心服务器模式，存在单点故障	从多台服务器中选择一台服务，损坏时可由另外一台替代	Hadoop缺少分布式锁服务
主节点故障	NameNode损坏后，恢复时需要花费一段时间获知DataNode的状态	Master损坏时，替补服务器可以快速获知Chunk Server的状态	
子服务器管理模式	DataNode通过心跳的方式告知NameNode其生存状态	Chunk Server在Chubby中获取独占锁表示其生存状态，Master通过轮询这些独占锁获知Chunk Server的生存状态	

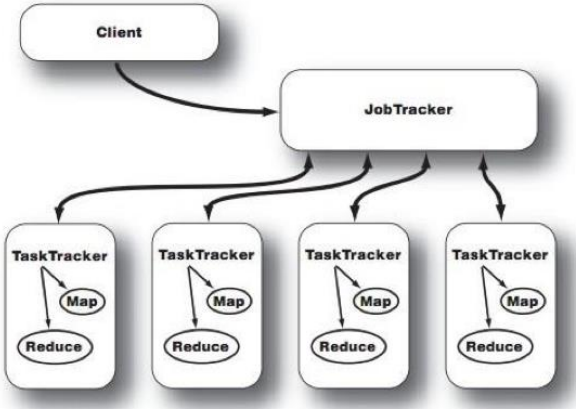
Hadoop MapReduce计算流程

- 基本要求
  - 待处理的数据集可以分解为可并行处理的小数据集
- 操作阶段
  - **Map阶段**
  - **Reduce阶段**



HMapReduce实现机制

- 由JobTracker和TaskTracker调度
  - JobTracker负责调度和管理TaskTracker
- 计算和存储共享节点
- 处理的小数据集不大于HDFS中的数据块
  - 每台计算机上可运行多个任务
- 连接（Combine）和分区（Partition）
  - 合并key相同的中间结果，可以直接使用Reduce函数
  - 将连接的结果按照key的范围划分
- 任务管道
  - Reduce的结果不一定需要合并，而是作为新的输入

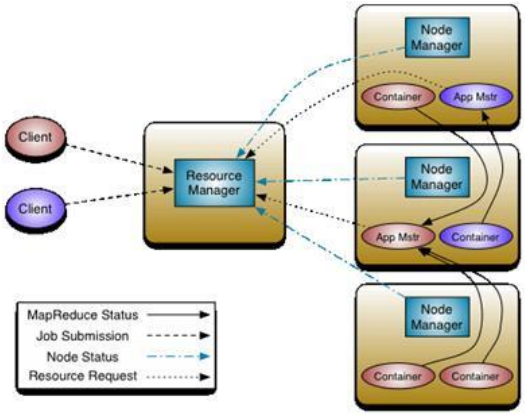


HMapReduce的缺陷

- JobTracker 是Map-reduce的集中处理点，存在单点故障
- 当 map-reduce job 非常多的时候，JobTracker会造成很大的内存开销，进一步增加了 JobTracker出现故障 的风险
  - HMapReduce通常只能支持不多于4000个节点
- TaskTracker 端以task的数目作为资源的表示过于简单，没有考虑到cpu/内存的占用情况
  - 例如当两个需要大内存的task被调度到了一起，很容易出现内存溢出
- TaskTracker端把资源强制划分为map task slot和reduce task slot
  - 如果当系统中只有map或reduce task时，会造成资源的浪费
- 代码难以阅读，一个class内包含太多事情，增加 bug 修复和版本维护的难度
- HMapReduce框架在进行任何变化（例如 bug 修复、性能提升等）时，都会强制让分布式集群系统的每一个用户端同时更新

基于YARN的Hadoop MapReduce

- 思想：将**JobTracker**两个主要的功能，资源管理和任务调度/监控，分离成单独的组件
  - **ResourceManager**：全局管理所有应用程序计算资源的分配，调度、启动和监控每一个 **ApplicationMaster**
  - **ApplicationMaster**：每个**Job**有一个负责该**Job** 生命周期内的所有工作
  - **NodeManager**负责 **Container** 状态的维护，并向**ResourceManager**保持心跳



HBase设计动机和目标

- 动机
  - 数据库系统无法适应大型分布式数据存储的需要
  - 改良的关系数据库（副本、分区等）难于安装与维护
  - 关系模型对数据的操作使数据的存贮变得复杂
- 目标
  - 空间的扩展只需要加入存储结点
  - 实现极大的、稀疏的“表”，存储在分布式文件系统上
  - 不支持SQL

HBase逻辑视图

- 行
  - 行关键字，时间戳，列
  - 表按照行键的“逐字节排序”顺序对行进行处理
  - 表内数据非常稀疏，不同的行的列数目不相同
  - 写操作时对行的操作是是“原子”的
- 列
  - 物理上将同“族”数据存储在一起
  - 数据可通过时间戳区分版本

Row Key	Time Stamp	Column Contents	Column Anchor		Column “mime”
			cnnsi.com	my.look.ca	
“com.cnn. www”	T9		CNN		
	T8			CNN.COM	
	T6	“<html>.. “			Text/html
	T5	“<html>.. “			
	T3	“<html>.. “			

HBase物理视图

Row Key	Time Stamp	Column Contents	Column Anchor		Column "mime"
			cnnsi.com	my.look.ca	
"com.cnn.www"	T9		CNN		
	T8			CNN.COM	
	T6	"<html>.. "			Text/html
	T5	"<html>.. "			
	T3	"<html>.. "			

Row Key	Time Stamp	Column: Anchor	
Com.cnn.www	T9	Anchor:cnnsi.com	CNN
	T8	Anchor:my.look.ca	CNN.COM

Row Key	Time Stamp	Column: mime
Com.cnn.www	T6	text/html

Row Key	Time Stamp	Column: Contents
Com.cnn.www	T6	"<html>.."
	T5	"<html>.."
	T3	"<html>.."

HBase中的区域

- 划分
  - 按水平方式划分，每个区域包含表中的一部分行
  - 每个区域包含一个随机id，区域内的行按行键排序
  - 当表大小超过阈值后，自动分割成两个相同大小的区域
- 管理
  - 区域服务器(Region Server)
    - 处理用户读写需求
    - 向主服务器报告状态，并取得需要服务的区域
    - 负责维护区域的分割
  - 主服务器(Master Server)
    - 指派区域服务器装载指定的区域
    - 恢复失效的区域服务器

HBase的区域服务器

- 存储
  - 每个列族对应一个**Hstore**
  - 一个**Hstore**包含多个**Map File**（类似于**B**树的结构）
- 写操作
  - 数据先缓存，达到一定数量后再批量写入
  - 写数据前“预写”日志，写入完成后在日志中标记
  - 一个区域服务器上的所有区域采用同一个日志
- 读操作
  - 先在缓存中查找，如果命中请求则直接服务
  - 如果存在多个版本，则返回最新的

HBase的区域服务器（续）

- 合并
  - 当达到周期或Map File数量超过阈值时，会进行合并
  - 合并可与无关的读写请求并发进行
  - 若读写请求与合并相关，则挂起读写操作直到合并完成
- 分割
  - 当区域超过阈值后，会按照行进行对半分割
  - 由主服务器确定接管的区域服务器
  - 被分割区域通过垃圾回收机制回收
  - 区域服务器在元信息表中生成子表元信息

HBase的元数据表

- 元数据表
  - 区域的元数据存储在另一个区域中
    - 以区域的唯一标识符作为行关键字
    - 包括区域中数据起止行、区域状态、区域服务器地址
  - 可包含多个区域
  - 所有元数据区域的元数据存储在根表里
- 根表(ROOT Table)
  - 只包含一个区域
  - 主服务器启动时扫描
    - 扫描根表获得所有元数据区域的位置
    - 扫描元数据区域得到区域的位置，并分配区域服务器

HBase的失效恢复

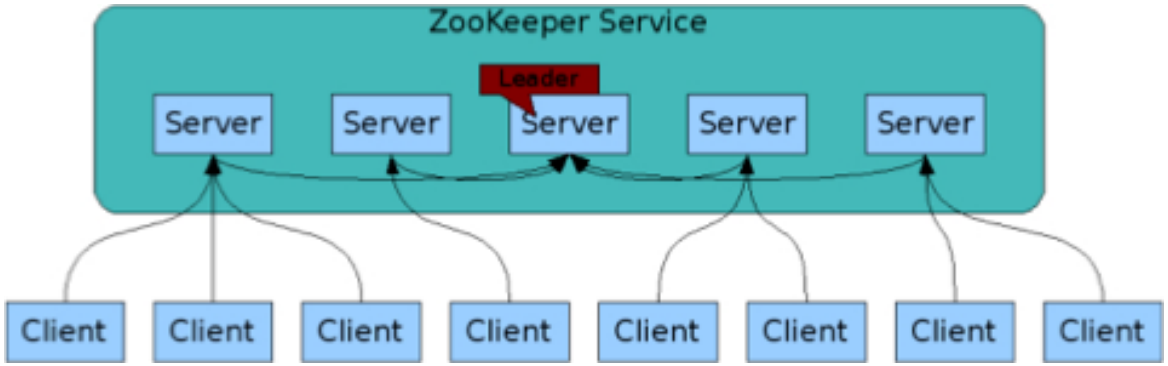
- 如果没有心跳，则主服务器判定区域服务器失效
- 主服务器将失效区域服务器所提供服务的区域重新分配给其它区域服务器
- 与**Google Bigtable**的区别
  - **Bigtable**中子表服务器即便和主服务器的连接断掉后，仍可以继续服务
  - 原因：缺少**Chubby**

Zookeeper的设计目标

- 目的
  - 大部分分布式应用需要一个主控、协调器或控制器来管理物理分布的子进程（如资源、任务分配等）
  - 缺乏通用的协调程序，需要各自编写
- 目标
  - 提供通用的分布式协调服务

Zookeeper的读写机制

- 由多个Server组成的集群
- 一个Leader，多个Follower
  - 每个Server都保存了一份数据副本
  - 全局数据一致
  - 分布式读写
  - 更新请求转发，由Leader实施

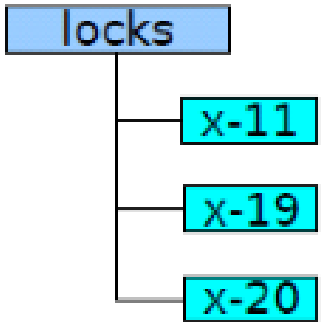


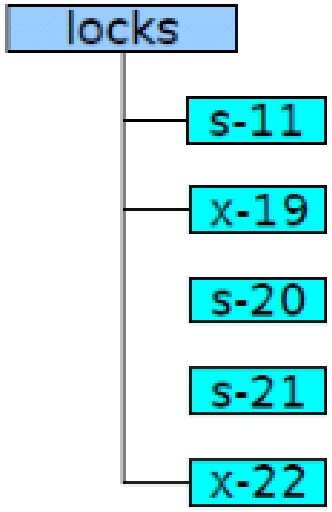
Zookeeper的使用约定

- 更新请求顺序执行
  - 来自同一个Client的更新请求按其发送顺序依次执行
- 数据更新原子性
  - 一次数据更新要么成功，要么失败。不存在部分数据写入成功或失败的情况
- 全局唯一数据视图
  - Client无论连接哪个Server，数据视图都是一致的
- 实时性
  - 在一定时间范围内，Client能读到最新数据

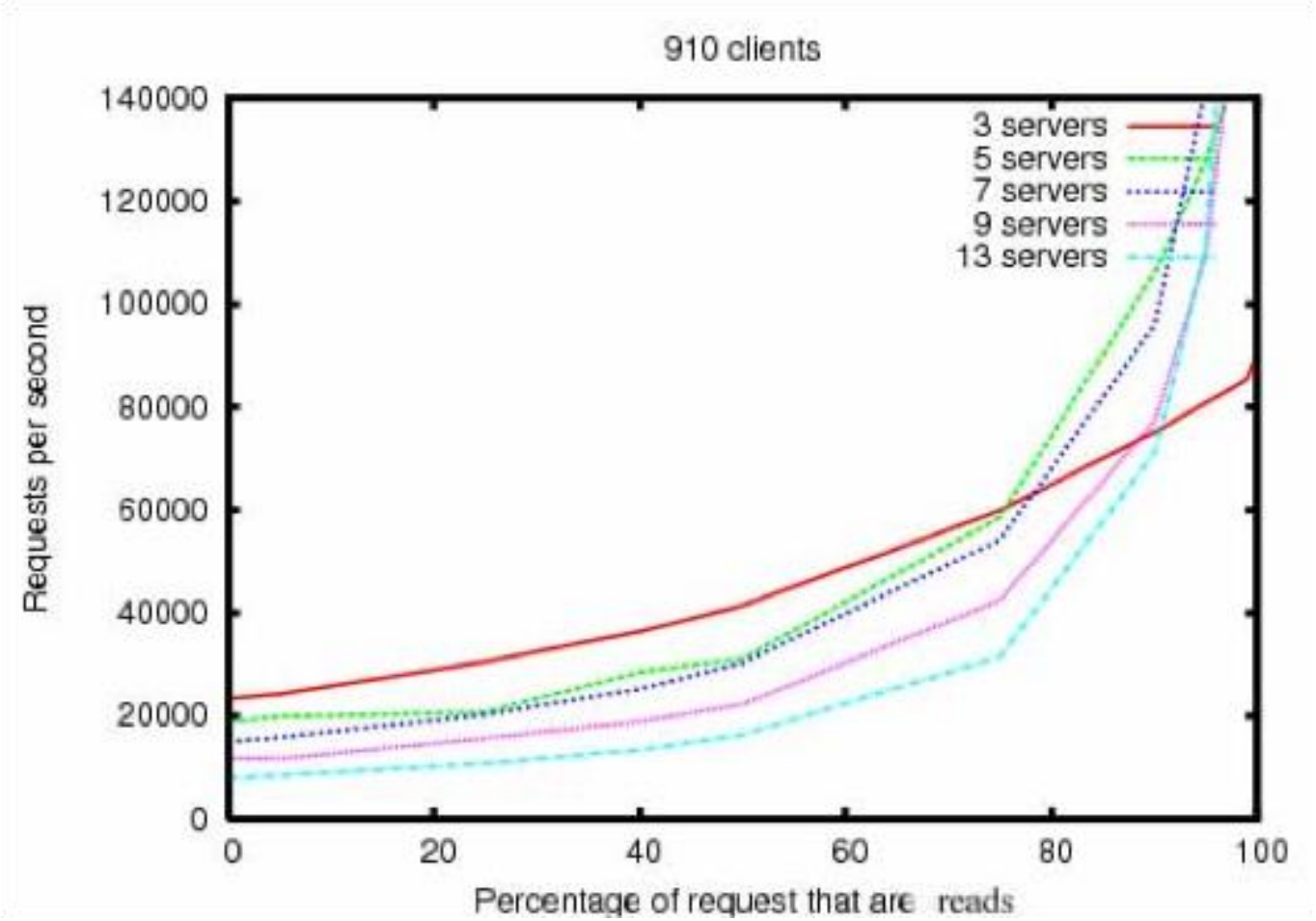
Zookeeper的功能

- **Leader的选举**
  - 在多个节点中选取**Master**
- 锁服务
  - 独占锁
    - 当分布式应用需要对资源独占使用
  - 共享锁
    - 当分布式应用需要对资源非独占使用
- 小数据存储





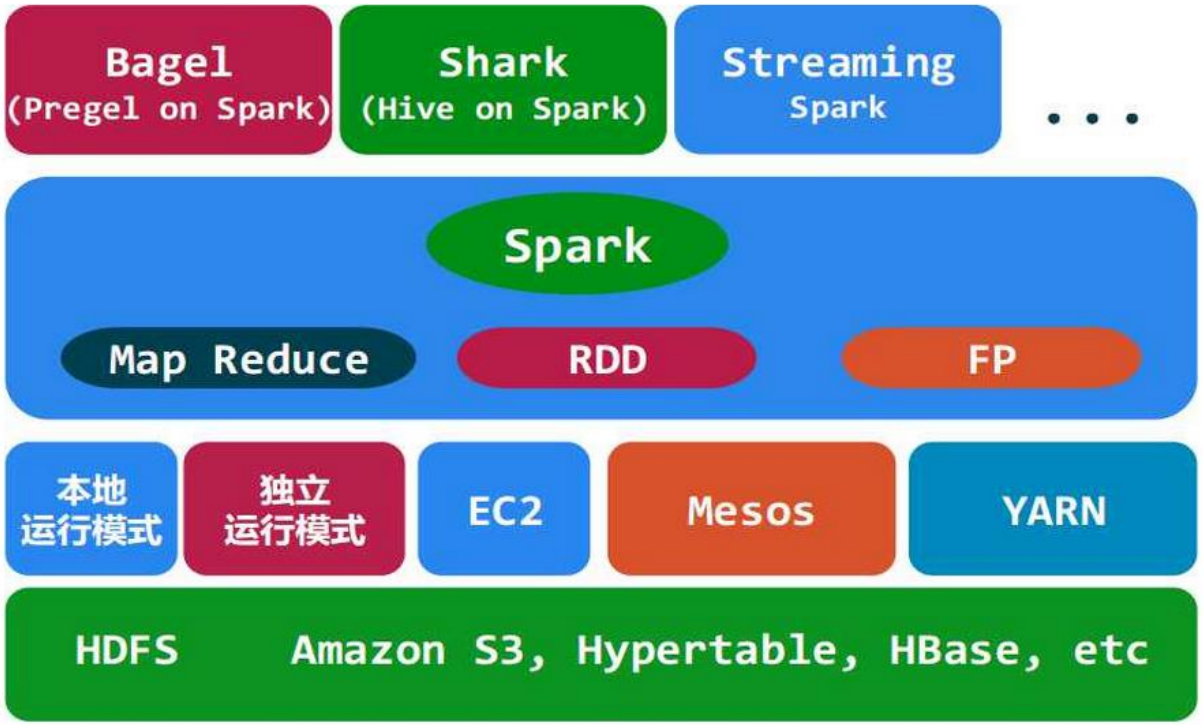
Zookeeper的性能



Spark

- **Spark是UC Berkeley AMP lab所开源的类Hadoop MapReduce的通用的并行计算框架**
- **Spark基于map reduce算法实现的分布式计算，拥有Hadoop MapReduce所具有的优点**
- **不同于MapReduce的是Job中间输出和结果可以保存在内存中，从而不再需要读写HDFS，因此Spark能更好地适用于数据挖掘与机器学习等需要迭代的map reduce的算法**
- Spark项目2010年开源，目前使用的有：Berkeley、Princeton、Klout、Foursquare、Conviva、Quantifind、Yahoo! Research、淘宝、豆瓣（python克隆版Dpark）等

Spark架构图


Bagel
(Pregel on Spark)
Shark
(Hive on Spark)
Streaming
Spark
. . .
Spark
Map Reduce
RDD
FP
本地
运行模式
独立
运行模式
EC2
Mesos
YARN
HDFS
Amazon S3, Hypertable, HBase, etc

Spark vs. Hadoop

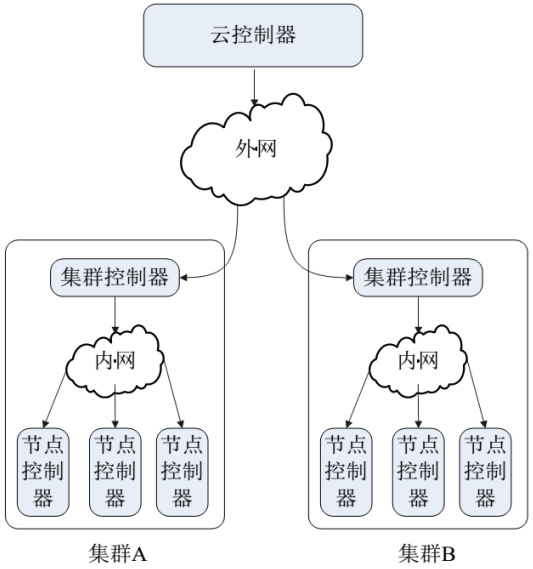
- Spark的中间数据放到内存中，对于迭代运算效率更高
  - Spark更适合于迭代运算比较多的ML和DM运算
- Spark比Hadoop更通用
  - Spark提供的数据集操作类型有很多种，比如map, filter, flatMap, sample, groupByKey, reduceByKey, union, join, cogroup, mapValues, sort,partionBy等多种操作类型，Spark把这些操作称为Transformations。同时还提供Count, collect, reduce, lookup, save等多种actions操作。
  - 这些多种多样的数据集操作类型，给给开发上层应用的用户提供了方便。各个处理节点之间的通信模型不再像Hadoop那样就是唯一的Data Shuffle一种模式。用户可以命名，物化，控制中间结果的存储、分区等。可以说编程模型比Hadoop更灵活。
- 容错性
  - 在分布式数据集计算时通过checkpoint来实现容错，而checkpoint有两种方式，一个是checkpoint data，一个是logging the updates。用户可以控制采用哪种方式来实现容错。
- 可用性
  - Spark通过提供丰富的Scala, Java，Python API及交互式Shell来提高可用性。

Spark与Hadoop的结合

- **Spark可以直接对HDFS进行数据的读写，同样支持Spark on YARN**
- **Spark可以与MapReduce运行于同集群中，共享存储资源与计算**
- **数据仓库Shark实现上借用Hive，几乎与Hive完全兼容**

Eucalyptus

- 设计原则：模块化
- 开发者：加州大学圣巴巴拉分校
- 实现语言：**Java**
- 对应商用系统：**Amazon AWS**
- 组成部分：
  - **Cloud Controller (CLC)**：负责管理整个系统
  - **Cluster Controller (CC)**：负责管理整个虚拟实例网络
  - **Node Controller (NC)**：控制主机操作系统及相应的 **hypervisor**
  - **Walrus (W)**：管理对 **Eucalyptus** 内的存储服务的访问
  - **Storage Controller (SC)**：实现 **S3** 接口，与 **Walrus**合作用于存储和访问虚拟机映像、内核映像、**RAM** 磁盘映像和用户数据

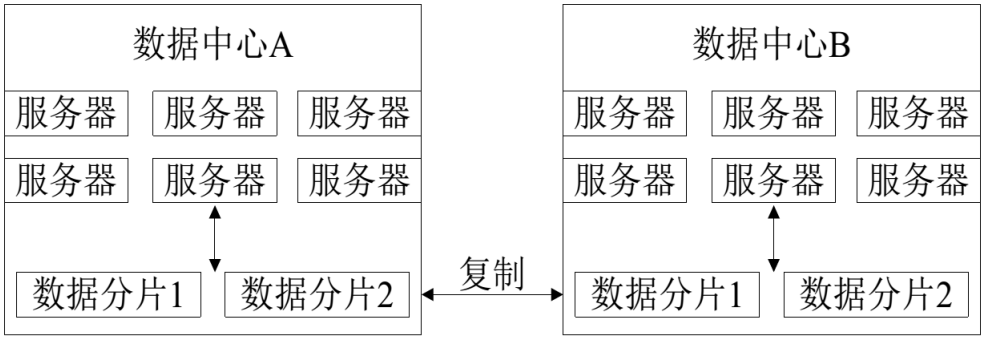


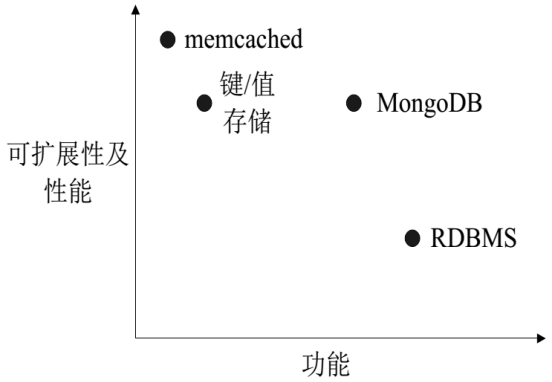
OpenStack

- 目标：为公共及私有云的建设与管理提供软件
- 开发者：**AT&T**，惠普，**IBM**，雅虎，思科……
- 对应商用系统：**Amazon EC2**，**VMWare**，……
- 实现语言：**Python**
- 优点：
  - 包含了一系列的项目：**Openstack-common**（通用程序Python库以及适用于所有OpenStack项目的组件），**Nova**（计算），**Swift**（存储），**Glance**（图形服务），**Keystone**（身份、令牌、分类及策略服务），**Horizon**（用户界面），**Quantum**（虚拟网络），**Cinder**（块存储）
  - 有大量的公司和人员支持：超过180家企业和400多位开发人员

MongoDB

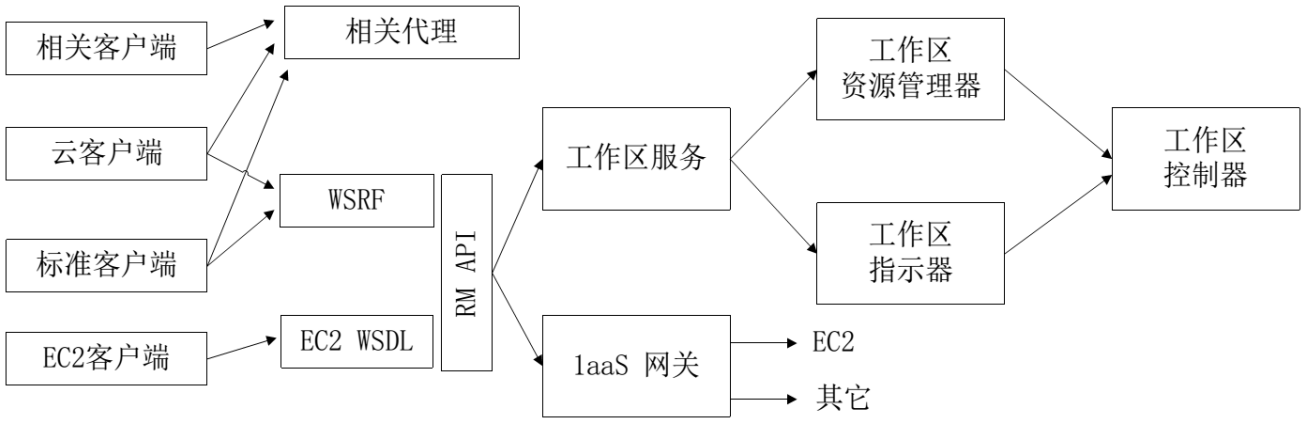
- 目标：构建基于分布式文件存储系统的数据库
- 开发者：**10gen**
- 对应商用系统：**Google App Engine**
- 实现语言：**C++**
- 优点：均衡性





Nimbus

- 目标：提供Iaas形式的云计算解决方案
- 开发者：网格中间件Globus
- 对应商用系统：Amazon EC2
- 组成部分：
  - Reference client：以命令行的方式访问服务
  - Web Services Resource Framework（ WSRF ）：Web服务资源框架
  - Resource Management API（ RM API ）：资源管理接口
  - Workspace：实际上就是一个计算节点

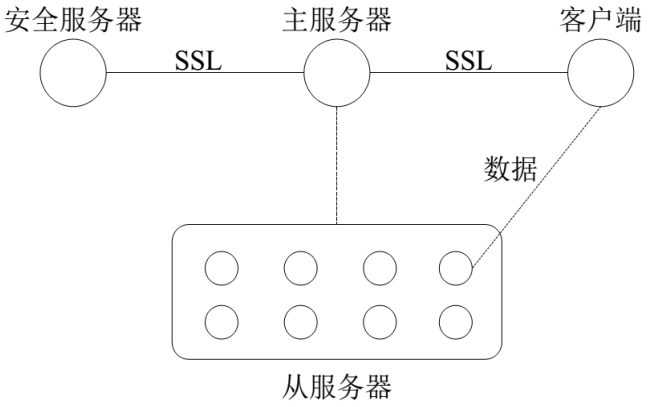


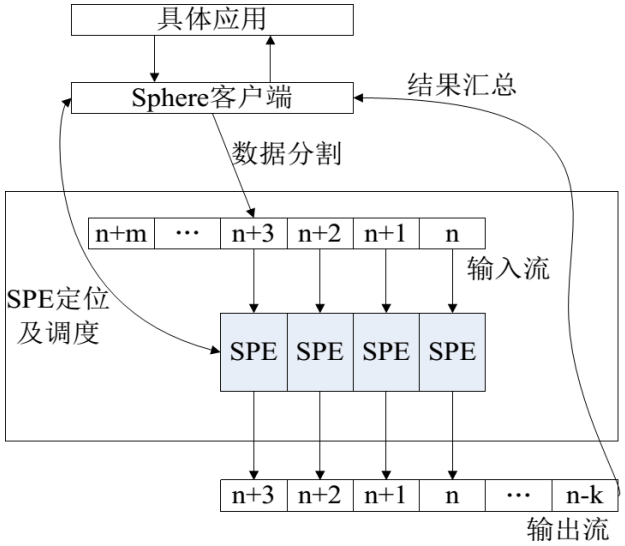
Cassandra

- 目标：可扩展且无单点故障的键值存储系统
  - **Dynamo和BigTable的结合**
- 开发者：**Facebook（2008年开源）**
- 优点：
  - 去中心化：无单点故障
  - 可扩展性强：苹果、**Netflix**、**eBay**等
  - 容错性好
  - **……**

Sector and Sphere

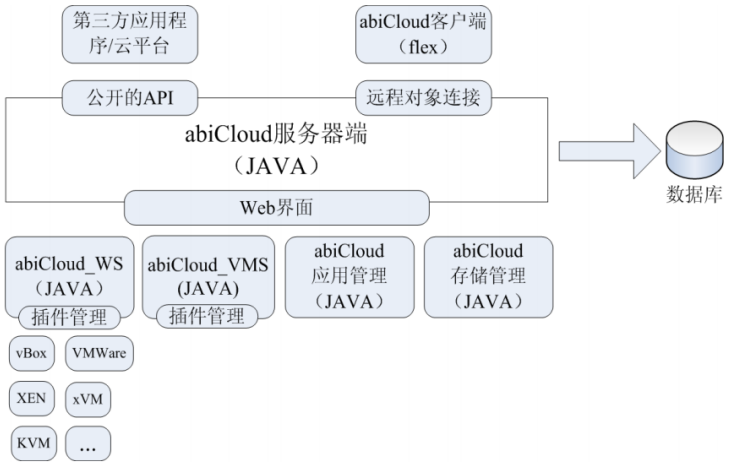
- 开发者：网格中间件**Globus**
- 实现语言：**C++**
- 组成部分：
  - **Sector**：广域网上的分布式存储系统，自动进行文件冗余存储
  - **Sphere**：建立在**Sector**之上的计算服务，为编写分布式密集型数据应用提供接口





abiquo

- 目标：在各种环境下高效地构建公有、私有或混合云
- 开发者：**abiquo**
- 组成部分：
  - **abiCloud**：创建和管理大型复杂的基础设施
  - **abiNtense**：大规模高性能计算
  - **abiData**：信息管理系统，用于搭建分析大量数据的应用，由**Hadoop Common**、**Hbase**和**Pig**开发



谢 谢