大数据架构与应用 期末复习

第 1 章 大数据概述

1. 三次信息化浪潮

信息化浪潮	发生时间	标志	解决的问题	代表企业
第一次信息化浪潮	1980 年前后	个人计算机	信息处理	Intel、AMD、IBM、苹果、微软、联想、戴尔、惠普等
第二次信息化浪潮	1995 年前后	互联网	信息传输	雅虎、谷歌、阿里巴巴、百度、腾讯等
第三次信息化浪潮	2010 年前后	大数据、云计算和物联网	信息爆炸	亚马逊、谷歌、IBM、VMware、Palantir、Hortonworks、Cloudera、阿里云等

2. 信息科技为大数据时代提供的技术支撑

（1）存储设备容量不断增加

（2）CPU 能力大幅提升

（3）网络带宽不断增加

3. 大数据的发展历程

阶段	时间	内容
第一阶段：萌芽期	20 世纪 90 年代～21 世纪初	随着数据挖掘理论和数据库技术的逐步成熟，一批商业智能工具和知识管理技术开始被应用，如数据仓库、专家系统、知识管理系统等
第二阶段：成熟期	21 世纪前 10 年	Web 2.0 应用迅猛发展，非结构化数据大量产生，传统处理方法难以应对，带动了大数据技术的快速突破，大数据解决方案逐渐走向成熟，形成了并行计算与分布式系统两大核心技术，谷歌的 GFS 和 MapReduce 等大数据技术受到追捧，Hadoop 平台开始盛行
第三阶段：大规模应用期	2010 年以后	大数据应用渗透各行各业，数据驱动决策，信息社会智能化程度大幅提高

4. 四种范式

（1）第一种范式：实验科学

（2）第二种范式：理论科学

（3）第三种范式：计算科学

（4）第四种范式：数据密集型科学

分布式存储、HDFS、并行计算、MapReduce（未知，应该是大题）

第 2 章 大数据处理架构 Hadoop

1. Hadoop 的特性

（1）高可靠性

（2）高效性

（3）高扩展性

（4）高容错性

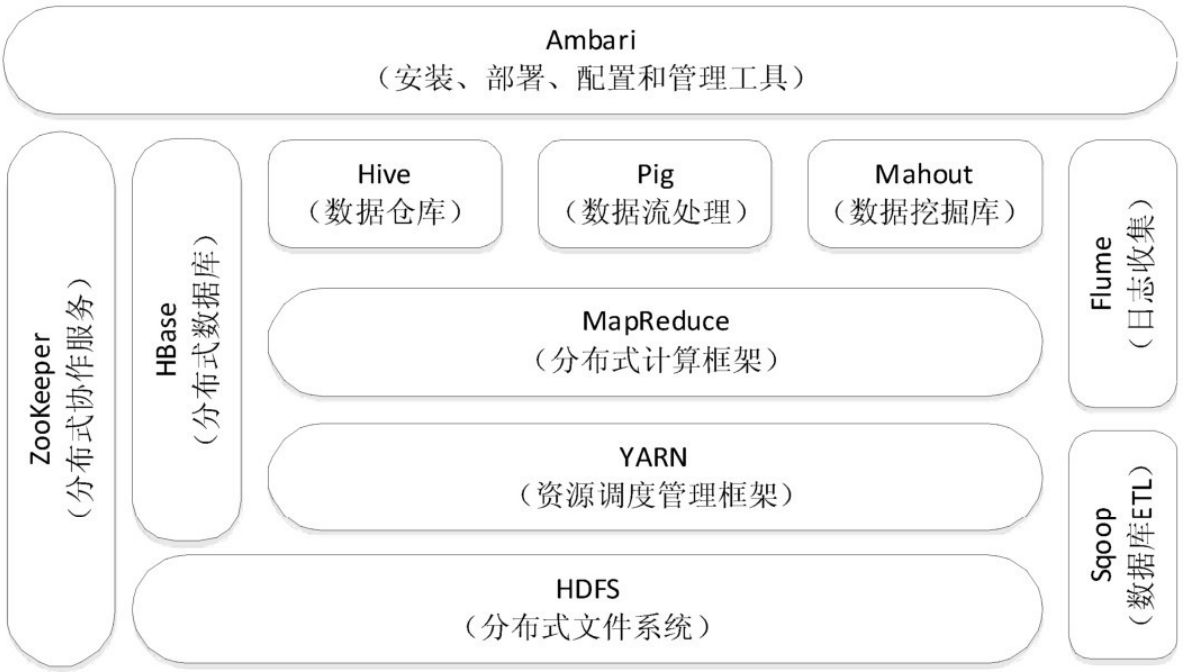
（5）成本低

（6）运行在 Linux 操作系统

（7）支持多种编程语言

2. Hadoop 生态系统

Hadoop 生态系统还包括 ZooKeeper、HBase、Hive、Pig、Mahout、 Flume、Sqoop、Ambari 等功能组件。



第 3 章 分布式文件系统 HDFS

1. 名称节点和数据节点的概念、作用，包含哪些数据结构（理解性记忆）

(1)分布式文件系统在物理结构上是由计算机集群中的多个节点构成 的，如图 3-2 所示。这些节点分为两类：一类叫“主节点”（Master Node），或者被称为“名称节点”（NameNode）；另一类叫“从节 点”（Slave Node），或者被称为“数据节点”（DataNode）。名称 节点负责文件和目录的创建、删除和重命名等，同时管理着数据节点和文件块的映射关系，因此客户端只有访问名称节点才能找到请求的 文件块所在的位置，进而到相应位置读取所需文件块。数据节点负责 数据的存储和读取，在存储时，由名称节点分配存储位置，然后由客 户端把数据直接写入相应数据节点；在读取时，客户端从名称节点获 得数据节点和文件块的映射关系，然后就可以到相应位置访问文件 块。数据节点也要根据名称节点的命令创建、删除和复制数据块

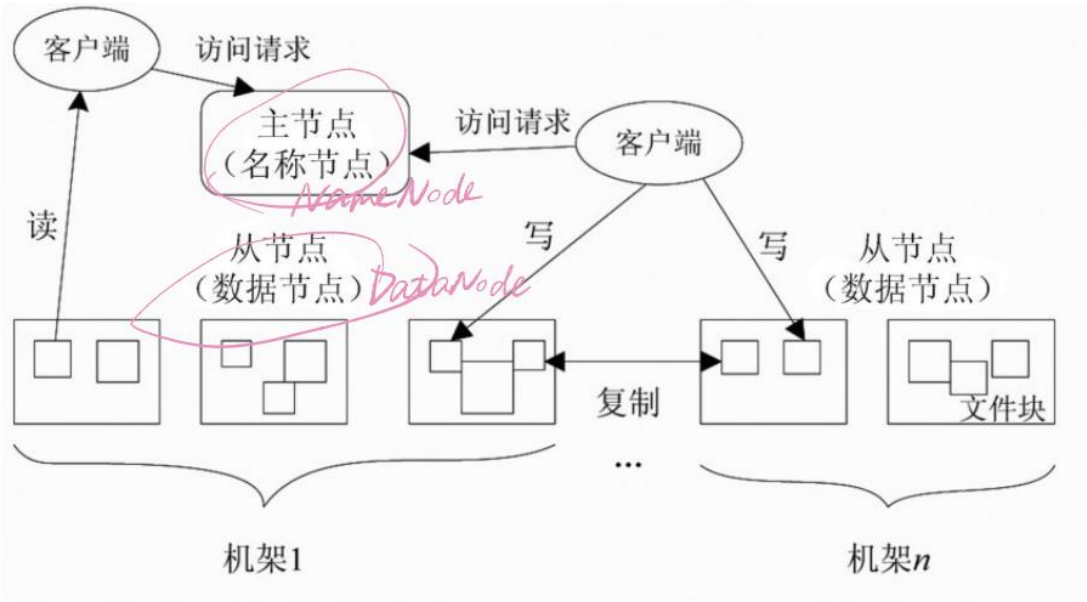


图3-2 分布式文件系统的物理结构

(2) 在 HDFS 中，名称节点负责管理分布式文件系统的命名空间（Namespace），保存了两个核心的数据结构（见图 3-3），即 FsImage 和 EditLog。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，操作日志文件 EditLog 中记录了所有针对文件的创 建、删除、重命名等操作。名称节点记录了每个文件中各个块所在的 数据节点的位置信息，但是并不持久化地存储这些信息，而是在

系统 每次启动时扫 ¾ 所有数据节点并重构，得到这些信息。

名称节点在启动时，会将 FsImage 的内容加载到内存当中，然后执 行 EditLog 文件中的各项操作，使内存中的元数据保持最新。这个操作 完成以后，就会创建一个新的 FsImage 文件和一个空的 EditLog 文 件。名称节点启动成功并进入正常运行状态以后，HDFS 中的更新操作 都会被写入 EditLog，而不是直接被写入 FsImage。这是因为对于分布 式文件系统而言，FsImage 文件通常都很庞大（一般都是 GB 级别以 上），如果所有的更新操作都直接在 FsImage 文件中进行，那么系统的 运行速度会变得非常缓慢。相对而言，EditLog 通常都要远远小于 FsImage，更新操作写入 EditLog 是非常高效的。名称节点在启动的过 程中处于 “安全模式” ，只能对外提供读操作，无法提供写操作。启 动过程结束后，系统就会退出安全模式，进入正常运行状态，对外提供读写操作

数据节点（DataNode）是分布式文件系统 HDFS 的工作节点，负责 数据的存储和读取，会根据客户端或者名称节点的调度来进行数据的 存储和检索，并且向名称节点定期发送自己所存储的块的列表信息。 每个数据节点中的数据会被保存在各自节点的本地 Linux 文件系统中

第二名称结点做了哪些工作（理解性记忆） 3.3.3

为了有效解决 EditLog 逐渐变大带来的问题，HDFS 在设计中采用了 第二名称节点（Secondary NameNode）。第二名称节点是 HDFS 架构的 一个重要组成部分，具有两个方面的功能：首先，它可以完成 EditLog 与 FsImage 的合并操作，减小 EditLog 文件大小，缩短名称节点重启时 间；其次，它可以作为名称节点的 “检查点” ，保存名称节点中的元 数据信息。

名称结点和数据结点在 HDFS 中发挥的作用、各自包含哪些数据结构

第 4 章 分布式数据库 HBase

1. HBase 和 BigTable 的底层技术对应关系

项目	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

2. HBase 与传统关系数据库的对比分析

（4.1.3 PDF P143）

- 数据类型
- 数据操作
- 存储模式
- 数据索引
- 数据维护
- 可伸缩性

3. 如何访问 HBase 的数据

（4.4.1）

行键和四维坐标

4. HBase 的三层结构

（4.4.3 PDF P160）

三级寻址（重点）

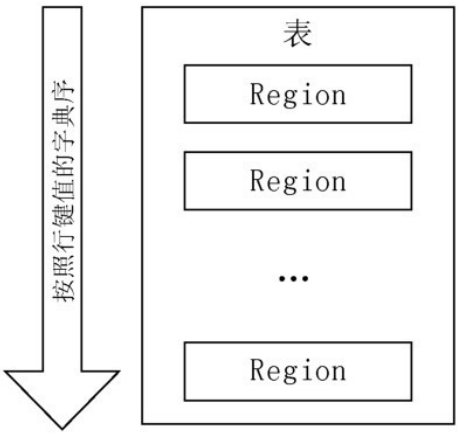
层次	名称	作用
第一层	ZooKeeper 文件	记录了-ROOT-表的位置信息
第二层	-ROOT-表	记录了.META.表的 Region 位置信息，-ROOT-表只能有一个 Region。通过-ROOT-表就可以访问.META.表中的数据
第三层	.META.表	记录了用户数据表的 Region 位置信息，.META.表可以有多个 Region，保存了 HBase 中所有用户数据表的 Region 位置信息

客户端访问用户数据之前，需要首先访问 ZooKeeper，获取-ROOT-表的位置信息，然后访问-ROOT-表，获得.META.表的信息，接着访 问.META.表，找到所需的 Region 具体位于哪个 Region 服务器，最后才 会到该 Region 服务器读取数据。

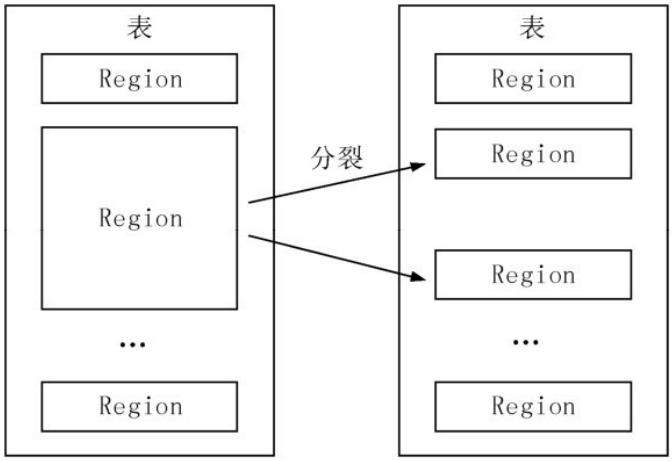
5. 表和 Region 的关系（理解）

（以下内容需要整理简化）

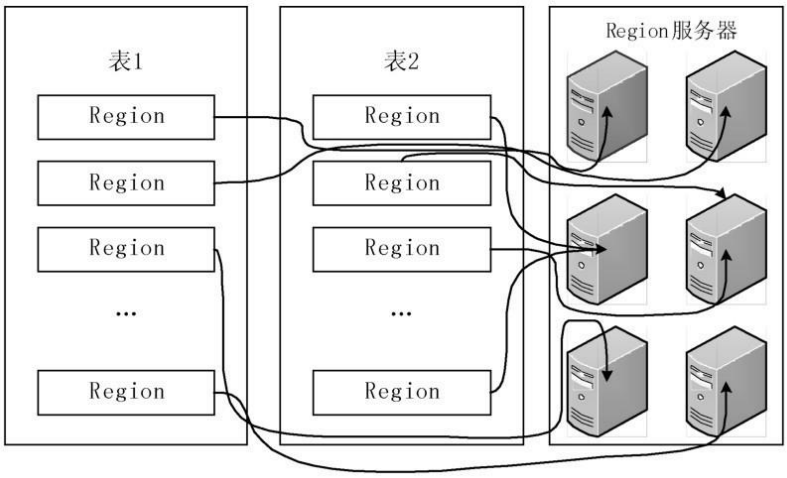
在一个 HBase 中，存储了许多表。对于每个 HBase 表而言，表中的行是根据行键的值的字典序进行维护的，表中包含的行的数量可能非常庞大，无法存储在一台机器上，需要分布存储到多台机器上。因此，需要根据行键的值对表中的行进行分区（见图 4-5）。每个行区间构成一个分区，被称为 “Region” 。Region 包含了位于某个值域区间内的所有数据，是负载均衡和数据分发的基本单位。这些 Region 会被分发到不同的 Region 服务器上。



初始时，每个表只包含一个 Region，随着数据的不断插入，Region 会持续增大。当一个 Region 中包含的行数量达到一个阈值时，就会被自动等分成两个新的 Region（见图 4-6），随着表中行的数量继续增加，就会分裂出越来越多的 Region。



每个 Region 的默认大小是 100～200 MB，是 HBase 中负载均衡和数据分发的基本单位。Master 主服务器会把不同的 Region 分配到不同的 Region 服务器上（见图 4-7），但是同一个 Region 不会被拆分到多个 Region 服务器上。每个 Region 服务器负责管理一个 Region 集合，通常在每个 Region 服务器上会放置 10～1000 个 Region。



第 5 章 NoSQL 数据库

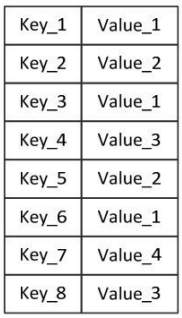
1. NoSQL 和关系数据库的差异（背诵）

对比指标	NoSQL	关系数据库	备注
数据库原理	部分支持	完全支持	关系数据库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	超大	大	关系数据库很难实现横向扩展，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	灵活	固定	关系数据库需要定义数据库模式，严格遵守数据定义和相关约束条件。 NoSQL 不存在数据库模式，可以自由、灵活地定义并存储各种不同类型的数据
查询效率	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如人意	快	关系数据库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关系数据库
一致性	弱一致性	强一致性	关系数据库严格遵守事务 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	很难实现	容易实现	任何一个关系数据库都可以很容易实现数据完整性，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 数据库无法实现
扩展性	好	一般	关系数据库很难实现横向扩展，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	很好	好	关系数据库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，关系数据库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

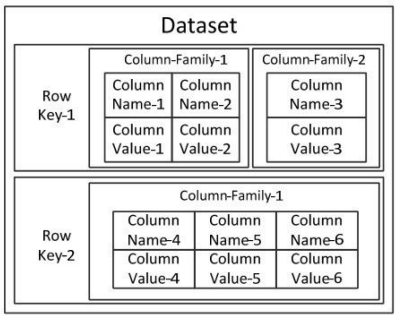
对比指标	NoSQL	关系数据库	备注
标准化	否	是	关系数据库已经标准化（SQL）。 NoSQL 还没有行业标准，不同的 NoSQL 数据库有不同的查询语言，很难规范应用程序接口
技术支持	低	高	关系数据库经过几十年的发展，已经非常成熟，Oracle 等大型厂商都可以提供很好的技术支持。 NoSQL 在技术支持方面仍然处于起步阶段，还不成熟，缺乏有力的技术支持
可维护性	复杂	复杂	关系数据库需要专门的数据库管理员（DataBase Administrator，DBA）维护 NoSQL 数据库虽然没有关系数据库复杂，但难以维护

2. NoSQL 的四大类型

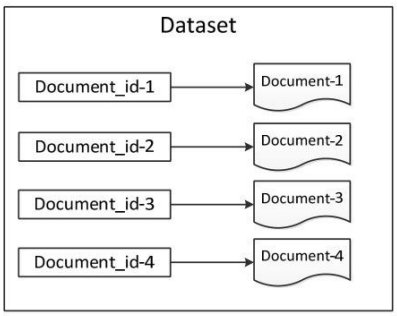
典型的 NoSQL 数据库通常包括**键值数据库、列族数据库、文档数据库**和**图数据库**。



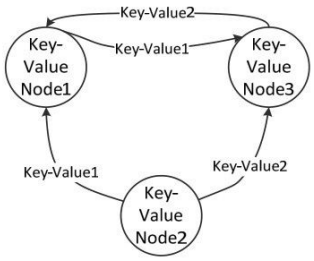
（a）键值数据库



（b）列族数据库



（c）文档数据库



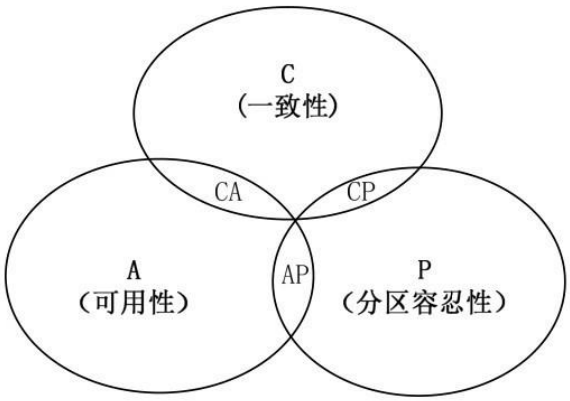
（d）图数据库

3. NoSQL 的三大基石

CAP 的含义

- **C（Consistency）：一致性**。它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。
- **A（Availability）：可用性**。它是指快速获取数据，且在确定的时间内返回操作结果。
- **P（Tolerance of Network Partition）：分区容忍性**。它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。



CAP 理论

CAP 理论告诉我们，一个分布式系统**不可能同时满足**一致性、可用性和分区容忍性这 **3 个特性**，**最多**只能**同时满足**其中 **2 个**，正所谓“鱼和熊掌不可兼得”。

如果追求一致性，就要牺牲可用性，需要处理因为系统不可用而导致的写操作失败的情况；如果要追求可用性，就要预估到可能发生数据不一致的情况，比如系统的读操作可能不能精确地读取写操作写入的最新值。

4. BASE

BASE 的基本含义

BASE 的基本含义是**基本可用（Basically Available）、软状态（Soft-state）和最终一致性（Eventual consistency）**

数据库事务的 ACID 四性

- **A（Atomicity）：原子性**。它是指事务必须是原子工作单元，对于其数据修改，要么全都执行，要么全都不执行。
- **C（Consistency）：一致性**。它是指事务在完成时，必须使所有的数据都保持一致状态。
- **I（Isolation）：隔离性**。它是指由并发事务所做的修改必须与任何其他并发事务所做的修改隔离。
- **D（Durability）：持久性**。它是指事务完成之后，它对于系统的影响是永久性的，该修改即使出现致命的系统故障也将一直保持。

第 6 章 云数据库

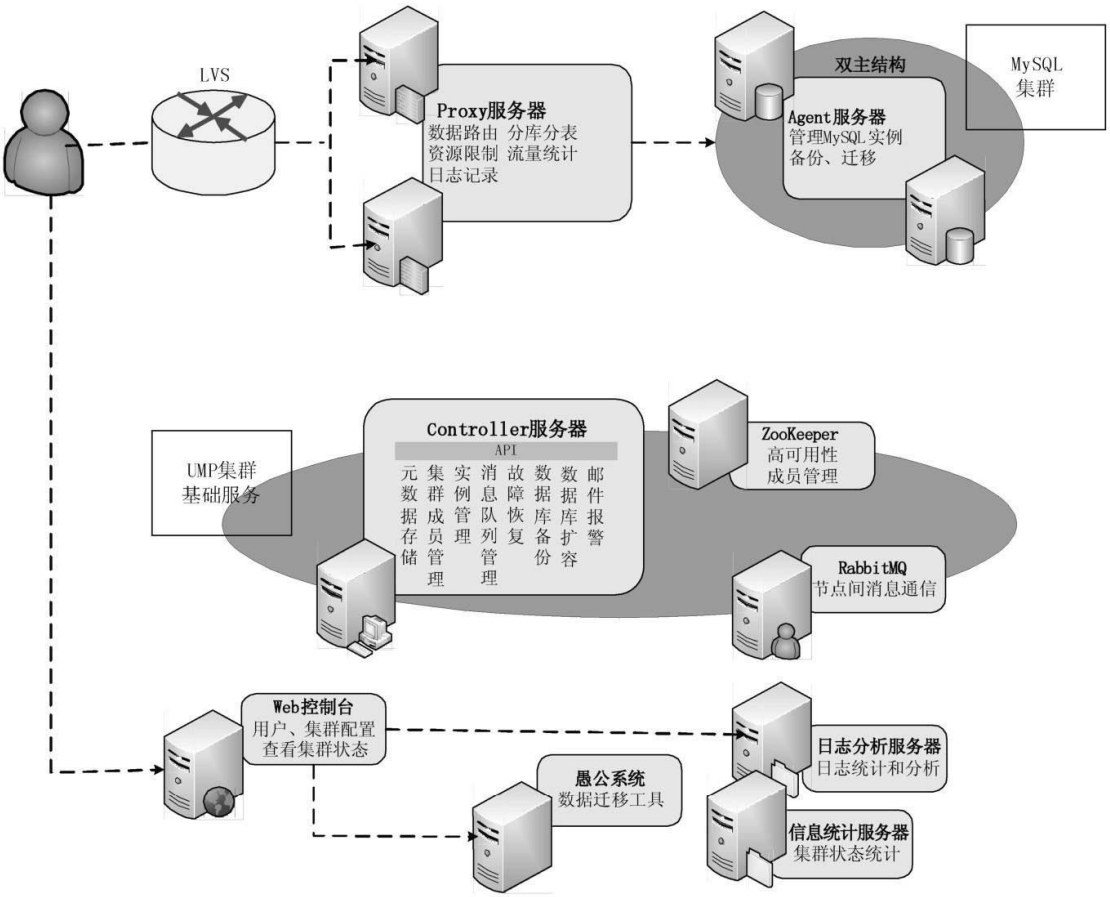
1. 云数据库和其他数据库的关系（理解）

（以下内容需要整理简化）

关系数据库采用关系数据模型，NoSQL 数据库采用非关系数据模型，二者都属于不同的数据库技术。从数据模型的角度来说，云数据 库并非一种全新的数据库技术，而是以服务的方式提供数据库功能的技术。云数据库并没有专属于自己的数据模型，云数据库所采用的数据模型可以是关系数据库所使用的关系模型（如微软的 SQL Azure 云数据库、阿里云 RDS 都采用了关系模型），也可以是 NoSQL 数据库所使用 的非关系模型（如 Amazon DynamoDB 云数据库采用的是“键值”存储）。同一个公司也可能提供采用不同数据模型的多种云数据库服务，例如百度云数据库提供了 3 种数据库服务，即分布式关系数据库服务（基于关系数据库 MySQL）、分布式非关系数据库服务（基于文档 数据库 MongoDB）、键值型非关系数据库服务（基于键值数据库 Redis）。实际上，许多公司在开发云数据库时，后端数据库都直接使 用现有的各种关系数据库或 NoSQL 数据库产品。比如腾讯云数据库采 用 MySQL 作为后端数据库，微软的 SQL Azure 云数据库采用 SQL Server 作为后端数据库。从市场的整体应用情况来看，由于 NoSQL 应用对开发 者要求较高，而 MySQL 拥有成熟的中间件、运维工具，已经形成一个良 性的生态系统等特性，因此从现阶段来看，云数据库的后端数据库以 MySQL 为主、NoSQL 为辅。

在云数据库这种 IT 服务模式出现之前，企业要使用数据库，就需要自建关系数据库或 NoSQL 数据库，它们被称为“自建数据库”。云数据库与这些“自建数据库”最本质的区别在于，云数据库是部署在云端的数据库，采用 SaaS 模式，用户可以通过网络租赁使用数据库服务，在有网络的地方都可以使用，不需要前期投入和后期维护，使用 价格比较低廉。云数据库对用户而言是完全透明的，用户根本不知道 自己的数据被保存在哪里。云数据库通常采用多租户模式，即多个租 户共用一个实例，租户的数据既有隔离又有共享，从而解决了数据存储的问题，同时降低了用户使用数据库的成本。而自建的关系数据库 和 NoSQL 数据库本身都没有采用 SaaS 模式，需要用户自己搭建 IT 基础设 施和配置数据库，成本相对而言比较高，而且需要自己进行机房维护 和数据库故障处理

2. UMP 系统架构



UMP 系统的角色

UMP 系统中的角色包括 **Controller 服务器、Web 控制台、Proxy 服务器、Agent 服务器、日志分析服务器、信息统计服务器、愚公系统**。

依赖的开源组件

依赖的开源组件包括 **Mnesia、RabbitMQ、ZooKeeper 和 LVS**。

3. UMP 系统功能

UMP 系统构建在一个大的集群之上，通过多个组件的协同作业，整个系统实现了**对用户透明的容灾、读写分离、分库分表、资源管理、资源调度、资源隔离和数据安全**等功能。

第 7 章 MapReduce

1. 词频统计实例（代码会写）

待整理 7.3 PDF P264

2. Map 和 Reduce 函数的输入输出

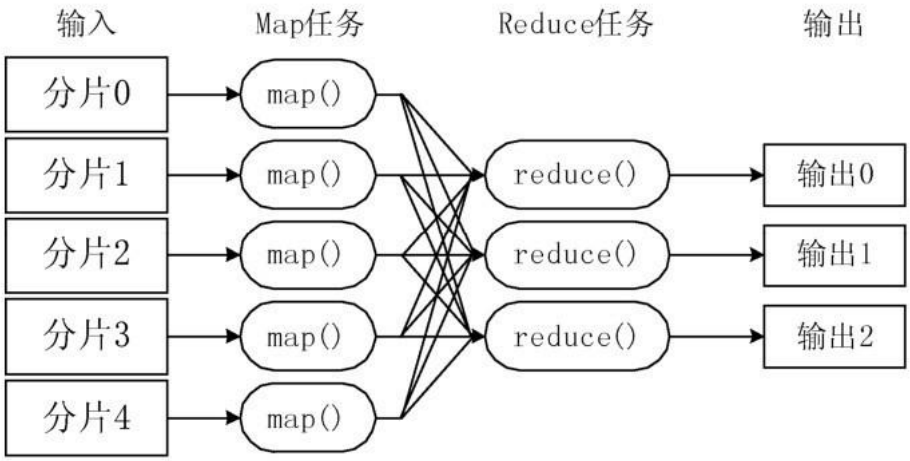
函数	输入	输出	说明
Map	$< k_1, v_1 >$	$\mathrm{List}(< k_2, v_2 >)$	(1) 将小数据集进一步解析成一批 <key, value> 对，输入 Map 函数中进行处理 (2) 每一个输入的 <k_1, v_1> 会输出一批 <k_2, v_2>，<k_2, v_2> 是计算的中间结果
Reduce	$< k_2, \mathrm{List}(v_2) >$	$< k_3, v_3 >$	输入的中间结果 <k_2, List(v_2)> 中的 List(v_2) 表示一批属于同一个 k_2 的 value

3. MapReduce 的工作流程

MapReduce 的核心思想可以用“分而治之”来概述，如图 7-1 所示，也就是把一个大的数据集拆分成多个小数据集在多台机器上并行处理。也就是说，一个大的 MapReduce 作业，首先会被拆分成许多个 Map 任务在多台机器上并行执行，每个 Map 任务通常运行在数据存储的节点上。这样计算和数据就可以放在一起运行，不需要额外的数据传输开销。当 Map 任务结束后，会生成以形式的许多中间结果。然后，这些中间结果会被分发到多个 Reduce 任务在多台机器上并行执行，具有相同 key 的会被发送到同一个 Reduce 任务，Reduce 任务会对中间结果进行汇总计算得到最后结果，并输出到分布式文件系统

需要指出的是，不同的 Map 任务之间不会进行通信，不同的 Reduce 任务之间也不会发生任何信息交换；用户不能显式地从一台机器向另一台机器发送消息，所有的数据交换都是通过 MapReduce 框架自身去实现的。

在 MapReduce 的整个执行过程中，Map 任务的输入文件、Reduce 任务的处理结果都是保存在分布式文件系统中的，而 Map 任务处理得到的中间结果保存在本地存储中（如磁盘）。另外，只有当 Map 处理全部结束后，Reduce 过程才能开始；只有 Map 才需要考虑数据局部性，实现“计算向数据靠拢”，Reduce 则无须考虑数据局部性



4. Shuffle 过程（理解）

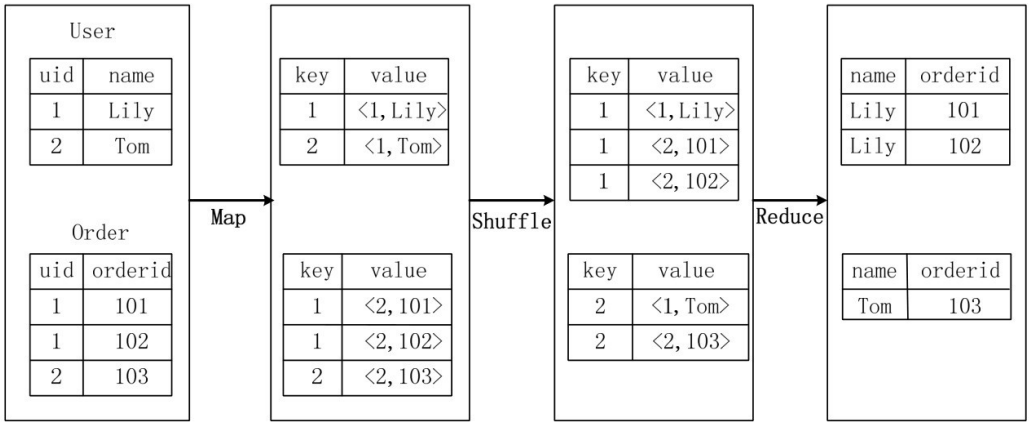
待整理 7.2.3 PDF P257

第 9 章 数据仓库 Hive

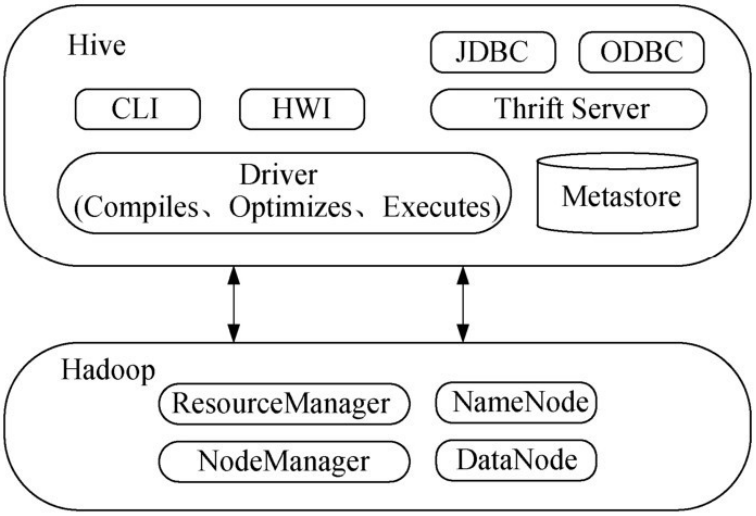
1. 连接操作转换成 MapReduce 的具体过程

9.3.1 PDF P333

join groupby

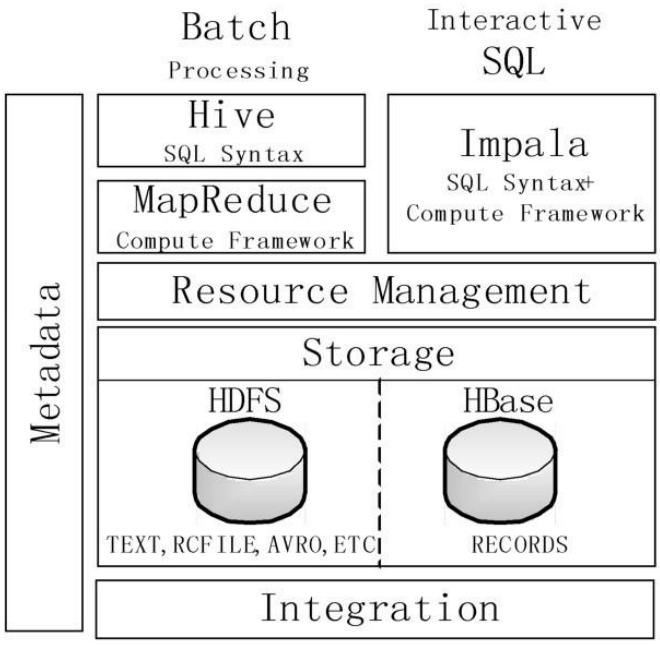


2. Hive 系统架构



- **系统架构**：用户接口模块、驱动模块以及元数据存储模块。
- **驱动模块**（Driver）包括编译器、优化器、执行器等，执行引擎可以是 MapReduce、Tez 或 Spark 等。

3. Impala 与 Hive 的比较



（以下内容需要整理简化）

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，其中，HDFS 支持存储 TEXT、RCFILE、PARQUET、AVRO、ETC 等格式的数据，HBase 存储表中记

录。

(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) 由于架构在 Hadoop 之上，Hive 也继承了其批处理的方式，在作业提交和调度的时候会涉及大量的开销，这就意味着 Hive 不能在大规模数据集上实现低延迟的快速查询。因此，Hive 比较适合进行长时间的批处理查询分析，而 Impala 适合进行实时交互式 SQL 查询。

(2) 当采用 MapReduce 作为执行引擎时，Hive 依赖于 MapReduce 计算框架，执行计划组合成管道型的 MapReduce 任务模式进行执行，Impala 则把执行计划表现为一棵完整的执行计划树，可以更自然地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，使得 Impala 更适合处理输出数据较小的查询请求，而对于大数据量的批量处理，Hive 依然是更好的选择。

总结

总之，Impala 的目的不在于替换现有的 MapReduce 工具。事实上，把 Hive 与 Impala 配合使用效果最佳，可以先使用 Hive 进行数据转换处理，再使用 Impala 对 Hive 处理后的结果数据集进行快速的数据分析。

4. Hive 的数据类型（有心的同学看一下？）

基本数据类型

类型	描述	示例
TINYINT	1 个字节（8 位）有符号整数	1
SMALLINT	2 个字节（16 位）有符号整数	1
INT	4 个字节（32 位）有符号整数	1
BIGINT	8 个字节（64 位）有符号整数	1
FLOAT	4 个字节（32 位）单精度浮点数	1.0
DOUBLE	8 个字节（64 位）双精度浮点数	1.0
BOOLEAN	布尔类型，true/false	true
STRING	字符串，可以指定字符集	"xmu"
TIMESTAMP	整数、浮点数或者字符串	1327882394（UNIX 新纪元时间）
BINARY	字节数组	[0,1,0,1,0,1,0,1]

集合数据类型

类型	描述	示例
ARRAY	一组有序字段，字段的类型必须相同	Array(1,2)
MAP	一组无序的键值对，键的类型必须是原子的，值可以是任何数据类型，同一个映射的键和值的类型必须相同	Map('a',1,'b',2)
STRUCT	一组命名的字段，字段类型可以不同	Struct('a',1,1,0)

第 10 章 Spark

1. 大数据处理的类型及其对应的 Spark 组件

处理类型	对应的 Spark 组件
复杂的批量数据处理	Spark Core 和 Spark SQL
基于历史数据的交互式查询	Spark SQL
基于实时数据流的数据处理	Spark Streaming 和 Structured Streaming

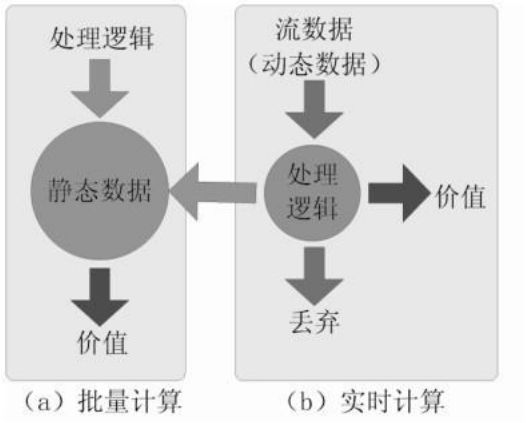
2. RDD 的概念

一个 RDD 就是一个分布式对象集合，本质上是一个只读的分区记录集合。每个 RDD 可以分成多个分区，每个分区就是一个数据集片段，并且一个 RDD 的不同分区可以被保存到集群中不同的节点上，从而可以在集群中的不同节点上进行并行计算。

RDD 提供了一种**高度受限**的共享内存模型，即 RDD 是**只读**的记录分区的集合，**不能直接修改**，只能基于稳定的物理存储中的数据集来创建 RDD.

第 11 章 流计算

批量计算和实时计算的差别



教材原文

批量计算以“静态数据”为对象，可以在很充裕的时间内对海量数据进行批量处理，计算得到有价值的信息。Hadoop 就是典型的批处理模型，由 HDFS 和 HBase 存放大量的静态数据，由 MapReduce 负责对海量数据执行批量计算。

流数据则不适合采用批量计算，因为流数据不适合用传统的关系模型建模，不能把源源不断的流数据保存到数据库中。流数据被处理后，一部分进入数据库成为静态数据，其他部分则直接被丢弃。传统的关系数据库通常用于满足信息实时交互处理需求，比如零售系统和银行系统，每次有一笔业务发生，用户通过和关系数据库系统进行交互，就可以把相应记录写入磁盘，并支持对记录进行随机读写操作。但是，关系数据库并不是为存储快速、连续到达的流数据而设计的，差别不支持连续处理，把这类数据库用于流数据处理，不仅成本高，而且效率低。

流数据必须采用实时计算，实时计算最重要的一个需求是能够实时得到计算结果，一般要求响应时间为秒级。当只需要处理少量数据时，实时计算并不是问题；但是，在大数据时代，不仅数据格式复杂、来源众多，而且数据量巨大，这就对实时计算提出了很大的挑战。因此，针对流数据的实时计算——流计算，应运而生。

个人整理

	批量计算	实时计算
数据类型	针对静态数据，对海量数据进行批处理。	处理的是流数据，需要实时响应并处理连续到达的数据。
计算模型	典型的代表是 Hadoop，使用 HDFS 和 HBase 存储静态数据，通过	需要专门针对流数据的计算模型，如流计算。

	MapReduce 进行批量计算。	
数据处理方式	在充裕的时间内处理大量数据，不追求实时性。	要求在短时间内（如秒级）得到计算结果，响应速度快。
数据存储	通常与传统关系数据库结合，用于处理静态数据。	不适用传统关系数据库，因为关系数据库不支持连续处理且在流数据处理上成本高、效率低。
应用场景	对大量静态数据进行处理和分析的场景，如离线数据分析。	需要快速响应和处理的场景，如零售系统和银行系统等。

第 12 章 Flink

Flink 的应用场景

应用类型	典型应用
事件驱动型应用	反欺诈、异常检测、基于规则的报警、业务流程监控、Web 应用（社交网络）等
数据分析应用	电信网络质量监控、移动应用中的产品更新及实验评估分析、消费者技术中的实时数据即席分析、大规模图分析等
数据流水线应用	电子商务中的实时查询索引构建、电子商务中的持续 ETL 等