问题引入

- 互联网上可访问的信息数量
- 海量的数据的有效访问
- 瓶颈---存储技术

数据库存储体系

数据存储的体系结构与计算机系统的体系结构密切相关:

- **集中式体系结构 ----> 集中式**
- **计算机的联网 ----> 客户/服务器**
- **分布计算能力 ----> 分布式**

分布式存储概念

- 集中式存储技术将数据存储在一台机器或一个节点
- 数据分散的存储
  - *充分使用网络中的每台机器上的磁盘空间*
  - *并将这些分散的存储资源构成一个虚拟的存储设备*
  - *物理上分散，逻辑上整体*

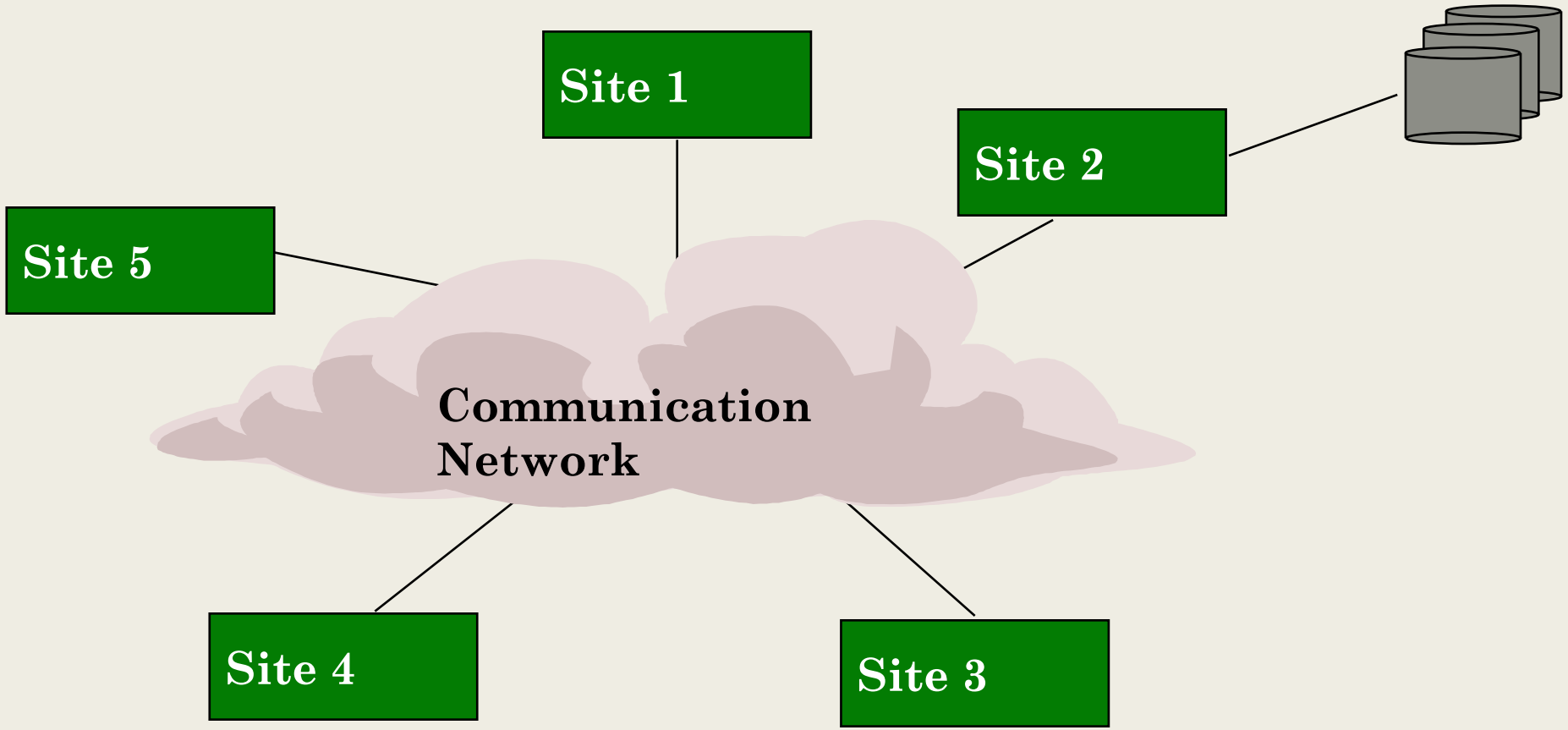
分布式存储的类型

- 结构化数据
- 非结构化数据
- 半结构化数据

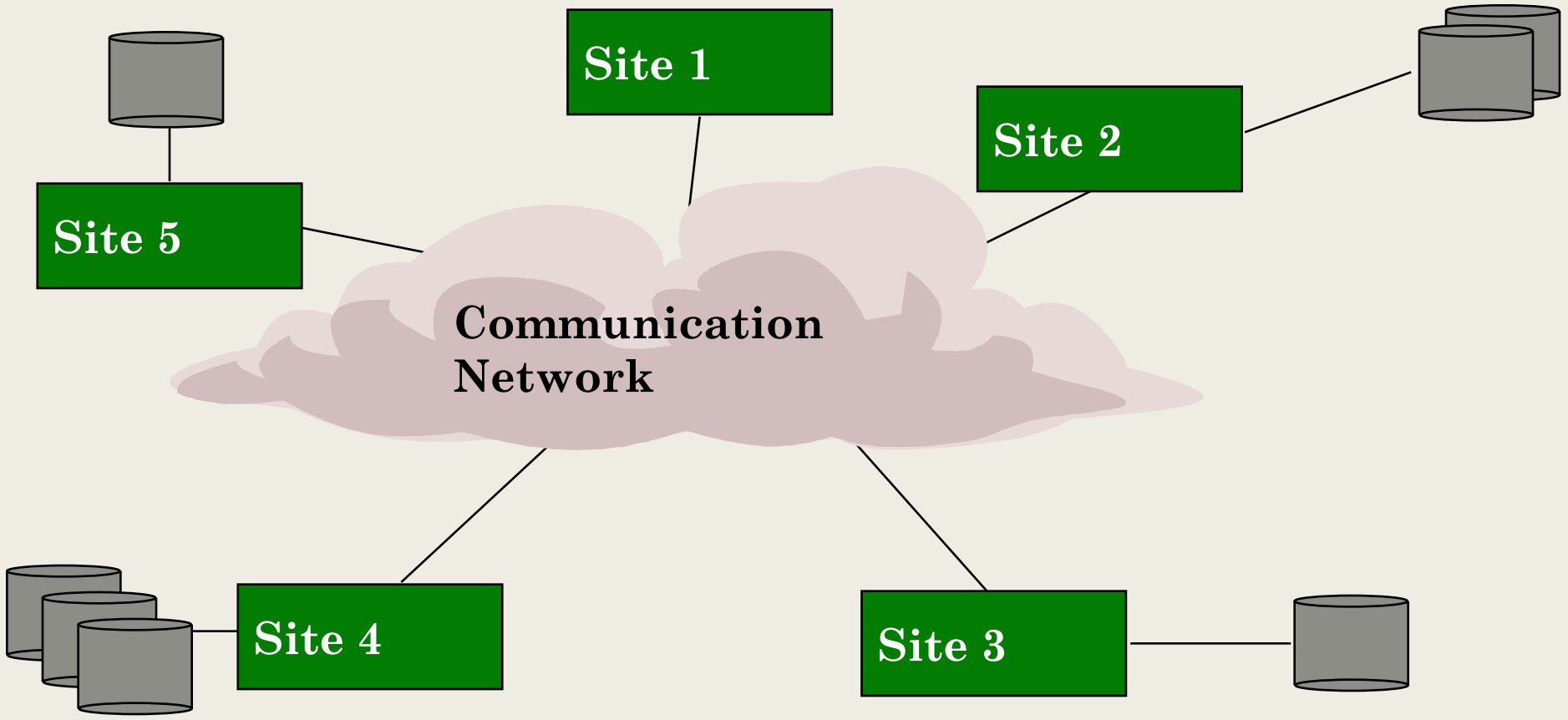
结构化数据的存储及应用

- 严格的用户定义的数据类型
  - *属性*
  - *数据类型*
  - *某种结构组合-如二维表*
- 各种关系模型数据库系统的扩展-分布式关系模型
  - *Oracle*
  - *MySql*
  - *Sql Server*
  - *DB2*

集中式



分布式 DBMS



分布式数据存储理论

在分布式数据库中存储关系r的几种方法：

- **数据复制**
- **数据分片**
- **数据复制与分片**
- **不复制也不分片**

分布式数据存储理论-复制

- **数据复制**

系统维护关系r的几个完全相同的副本（拷贝），各个副本存储在不同的节点上。

全复制：系统中的每个节点都存有关系r的一个拷贝。

主副本：指定关系r的多个副本中的一个作为主副本，从而简化副本管理。

分布式数据存储理论-分片

- **数据分片**

将关系r划分为多个片段r_1，r_2，…，r_n。这些片段中包含足够的信息，使得能够重构原始关系r

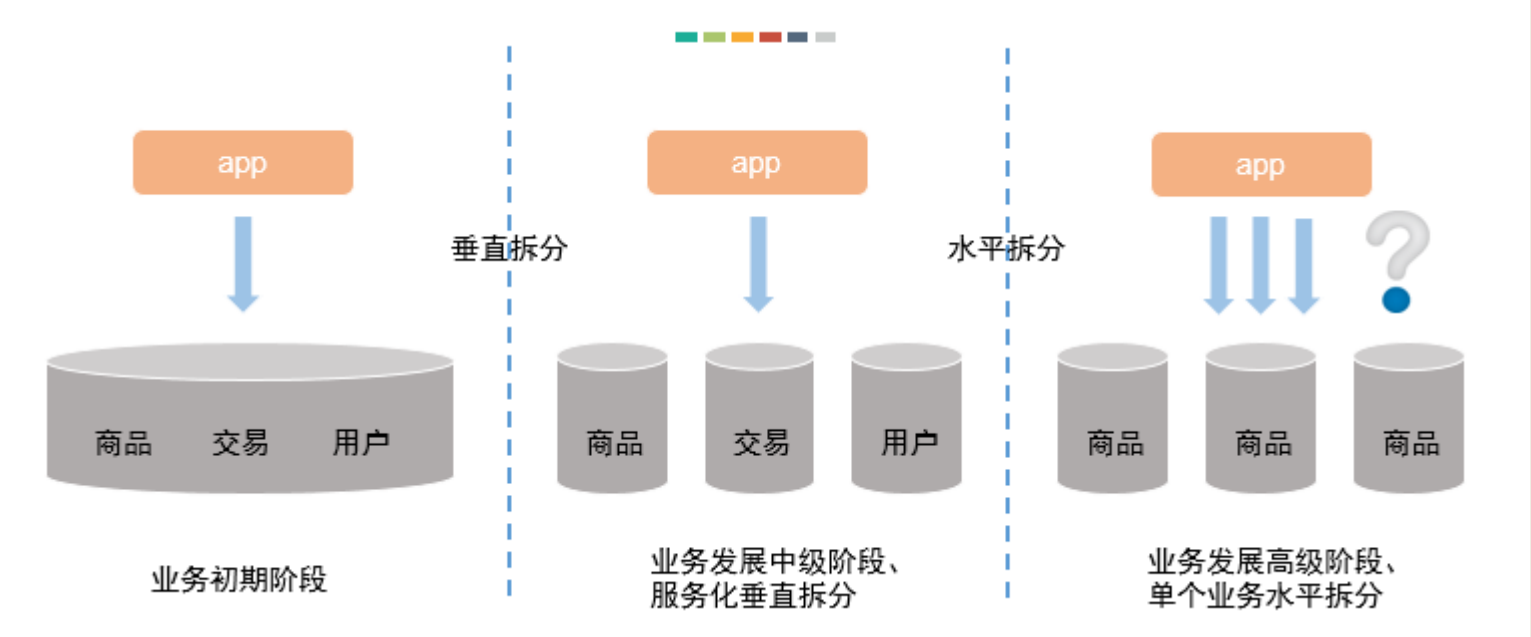
  - ***水平分片：将关系r划分为多个子集r_1，r_2，…，r_n。r的每个元组必须至少属于一个片段。***
  - ***垂直分片：将关系r（R）投影到R的属性的多个子集R_1，R_2，…，R_n。为保证关系r能被重构需要在每个R_i中都包含R的主码属性，或特殊的tuple-id属性***
  - ***混合分片：在水平分片或垂直分片的结果上再进行垂直分片或水平分片。***

高并发海量数据库应用

- 现实中的网站瓶颈
- 解决方法
  - *服务器拆分*

分布式数据库应用实践-1

- 循序渐进



分布式数据库应用实践-2

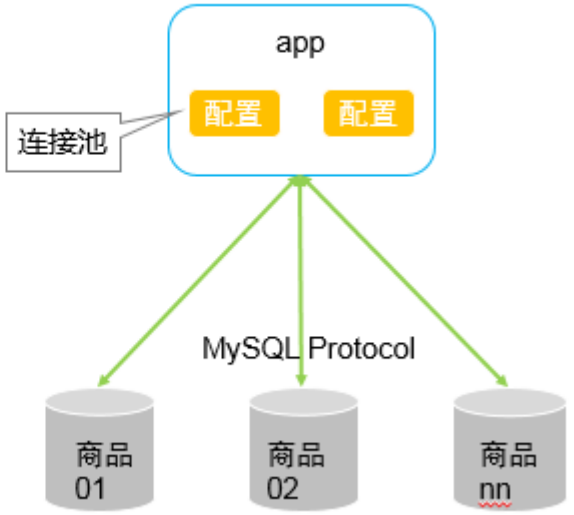
- 简单的水平拆分-客户端编程实现连接路由

无需引入额外模块，整体架构不变

程序把控力强，简单场景方便使用

对代码侵入性强★

配置管理复杂★



分布式数据库应用实践-2

- 复杂的水平拆分
- 采用中间件如:DRDS，mycat等

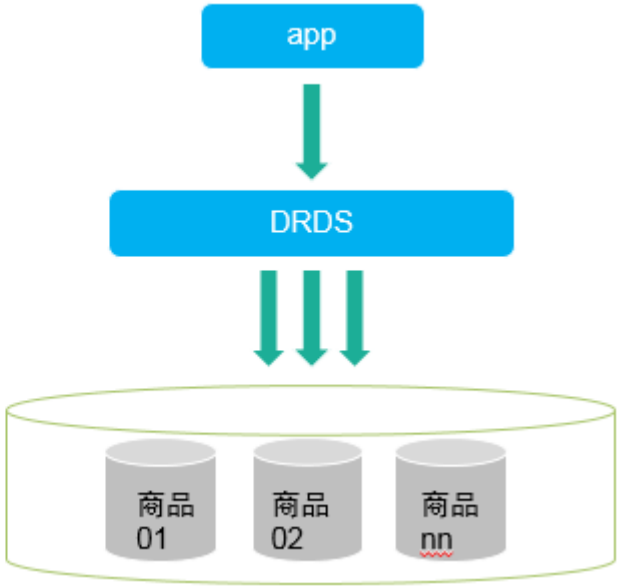
自动分库分表，对应用透明。

使用门槛极低，应用改造量小。

方便的动态水平扩容。

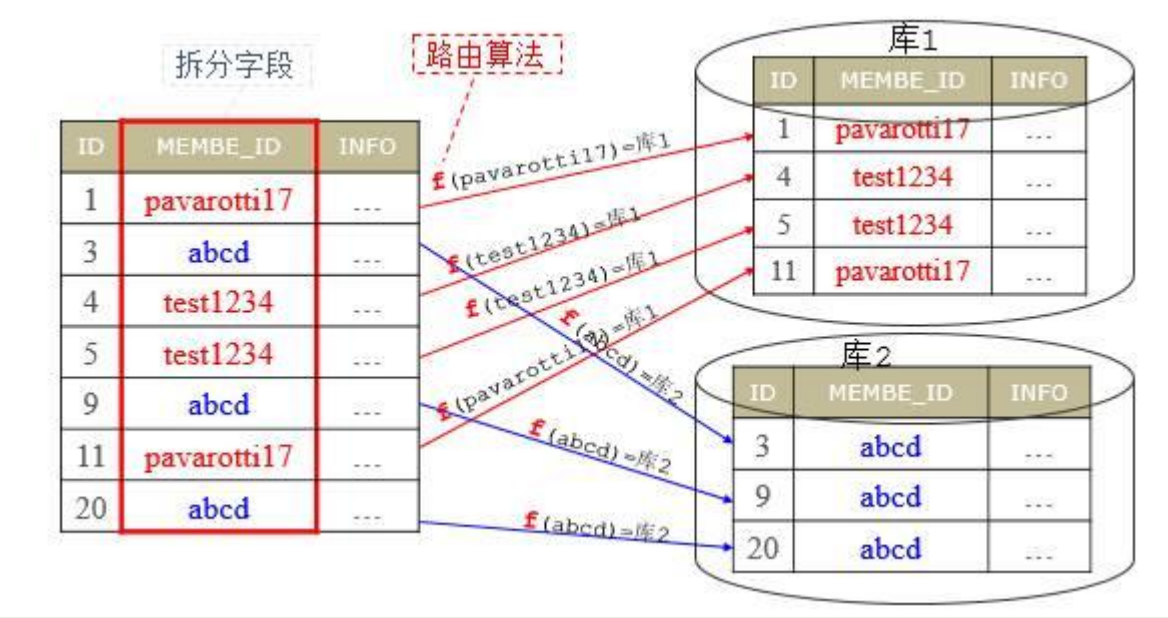
针对分布式的各种定制功能，如异构索引、小表广播等。（部分中间件产品）

最重要的是，有了数据库中间件，应用看到还是单一的数据库。★

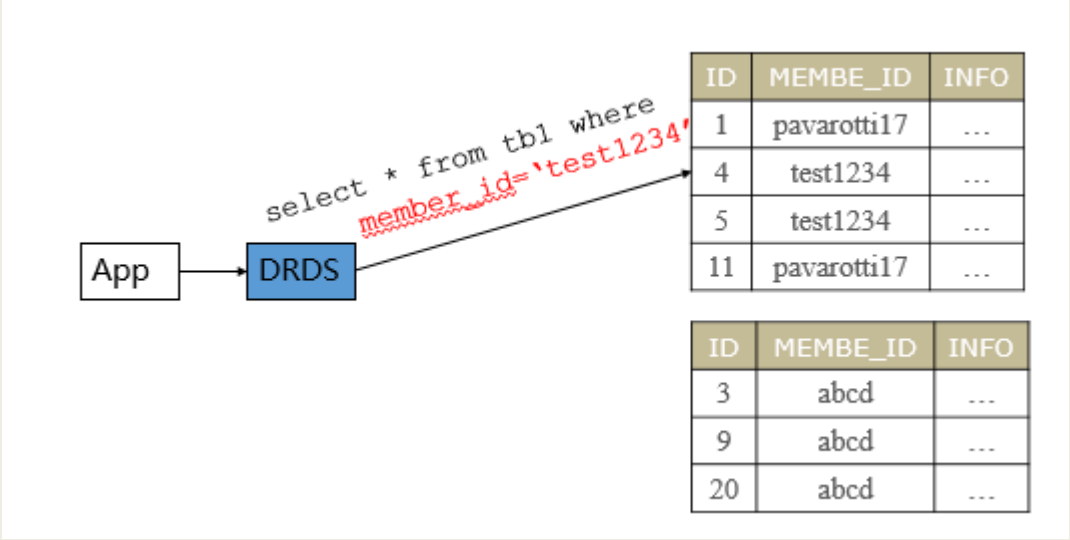


水平拆分方法

- 选择一个拆分字段
- 通过一个路由算法确定数据存放在哪个底层库



拆分表的查询


select * from tb1 where
member_id='test1234'
App
DRDS
ID
MEMBE_ID
INFO
1
pavarotti17
...
4
test1234
...
5
test1234
...
11
pavarotti17
...
ID
MEMBE_ID
INFO
3
abcd
...
9
abcd
...
20
abcd
...

非结构化数据分布式存储

- 数据格式不统一
- 数据类型多变
- 不方便用数据库二维逻辑表来表现
- 通常解决方案-分布式文件系统
  - *GFS*
  - *HDFS*

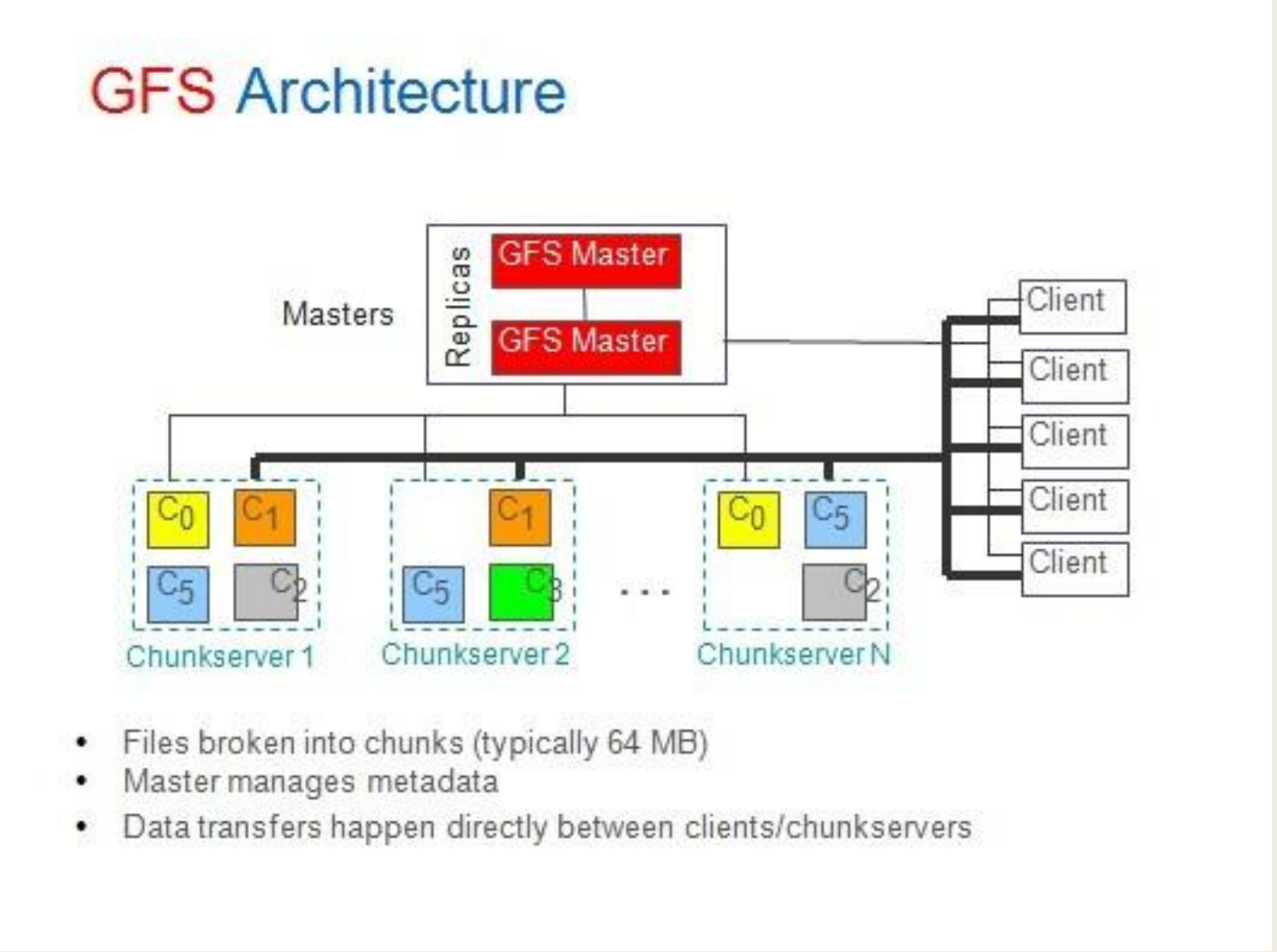
GFS-Google File System

- Google 三大论文之一：MapReduce、GFS、BigTale
- GFS是实现有史以来最强大通用机群的关键技术之一。
- 不仅仅是一个文件系统而已。它还包含数据冗余、支持低成本的数据快照，除了提供常规的创建、删除、打开、关闭、读、写文件操作，GFS还提供附加记录的操作。

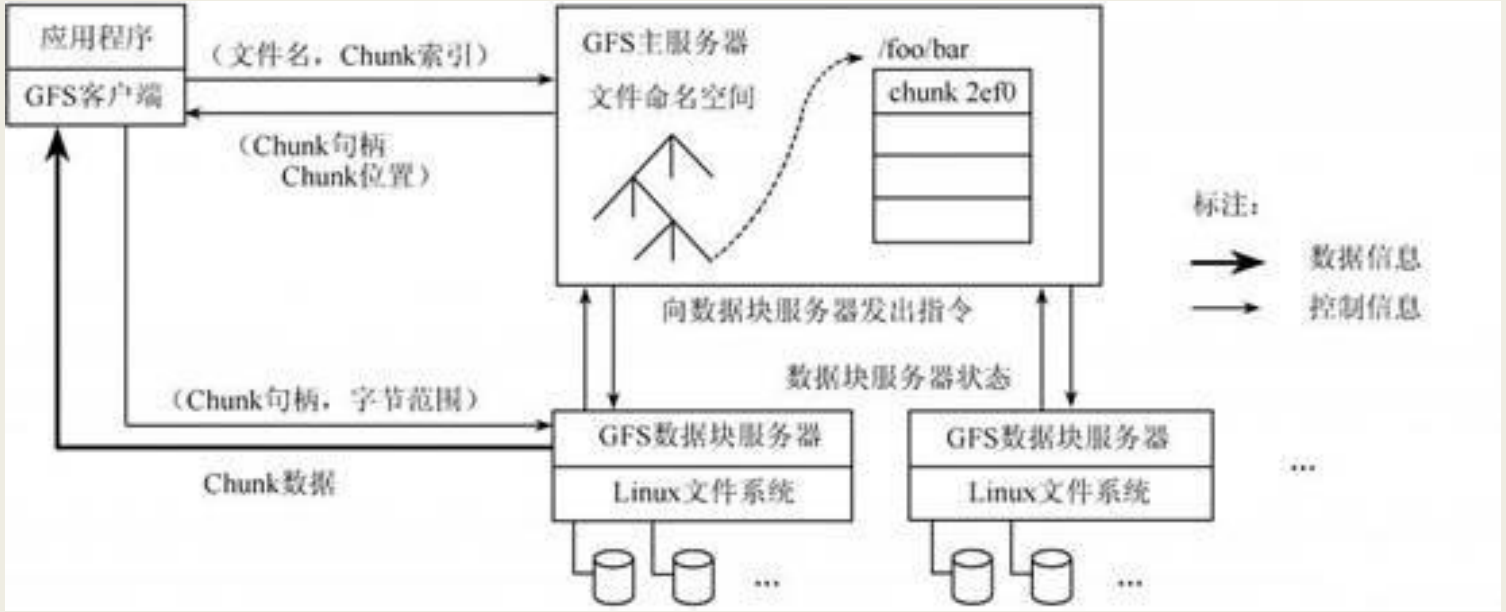
GFS 三类角色

- Client（客户端）：是GFS提供给应用程序的访问接口，它是一组专用接口，应用程序直接调用这些库函数，并与该库链接在一起。
- Master（主服务器）：是GFS的管理节点，主要存储与数据文件相关的元数据，而不是Chunk（数据块）。
- Chunk Server（数据块服务器）：负责具体的存储工作，用来存储Chunk。GFS采用副本的方式实现容错，每一个Chunk有多个存储副本（默认为三个）。 Chunk Server的个数可有有多个，它的数目直接决定了GFS的规模。

GFS架构


GFS Architecture
Masters
Replicas
GFS Master
GFS Master
Client
Client
Client
Client
Client
C0
C1
C5
C2
C1
C5
C3
C0
C5
C2
Chunkserver 1
Chunkserver 2
Chunkserver N
• Files broken into chunks (typically 64 MB)
• Master manages metadata
• Data transfers happen directly between clients/chunkservers

GFS工作数据流


应用程序
GFS客户端
（文件名，Chunk索引）
（Chunk句柄
Chunk位置）
GFS主服务器
文件命名空间
/foo/bar
chunk 2ef0
向数据块服务器发出指令
数据块服务器状态
（Chunk句柄，字节范围）
Chunk数据
GFS数据块服务器
Linux文件系统
GFS数据块服务器
Linux文件系统
...
...
...
标注：
数据信息
控制信息

半结构化数据分布式存储

- 就是介于完全结构化数据（如关系型数据库、面向对象数据库中的数据）和完全无结构的数据（如声音、图像文件等）之间的数据
- 半结构化数据模型具有一定的结构性，但较之传统的关系和面向对象的模型更为灵活。
- 半结构数据模型完全不基于传统数据库模式的严格概念。
- 不适合用传统的关系型数据库进行存储，适合存储这类数据的数据库被称作“NoSQL”数据库。

NOSQL数据库

- Non Relational Database(非关系型数据库)更为恰当
- 非关系型，分布式，轻量级
- 支持水平扩展且一般不保证遵循ACID原则。

NoSQL兴起的原因一

1、关系数据库已经无法满足Web2.0的需求。主要表现在以下几个方面：

（1）无法满足海量数据的管理需求

（2）无法满足数据高并发的需求

（3）无法满足高可扩展性和高可用性的需求

NoSQL兴起的原因二

2、“One size fits all”模式很难适用于截然不同的业务场景

•关系模型作为统一的数据模型既被用于数据分析，也被用于在线业务。但这两者一个强调高吞吐，一个强调低延时，已经演化出完全不同的架构。用同一套模型来抽象显然是不合适的

•Hadoop就是针对数据分析

•MongoDB、Redis等是针对在线业务，两者都抛弃了关系模型

NoSQL兴起的原因三

3、关系数据库的关键特性包括完善的事务机制和高效的查询机制。但是，关系数据库引以为傲的两个关键特性，到了Web2.0时代却成了鸡肋，主要表现在以下几个方面：

（1）Web2.0网站系统通常不要求严格的数据库事务

（2）Web2.0并不要求严格的读写实时性

（3）Web2.0通常不包含大量复杂的SQL查询（去结构化，存储空间换取更好的查询性能）

NoSQL与关系数据库的比较

（1）关系数据库

优势：以完善的关系代数理论作为基础，有严格的标准，支持事务ACID四性，借助索引机制可以实现高效的查询，技术成熟，有专业公司的技术支持

劣势：可扩展性较差，无法较好支持海量数据存储，数据模型过于死板、无法较好支持Web2.0应用，事务机制影响了系统的整体性能等

（2）NoSQL数据库

优势：可以支持超大规模数据存储，灵活的数据模型可以很好地支持Web2.0应用，具有强大的横向扩展能力等

劣势：缺乏数学理论基础，复杂查询性能不高，大都不能实现事务强一致性，很难实现数据完整性，技术尚不成熟，缺乏专业团队的技术支持，维护较困难等

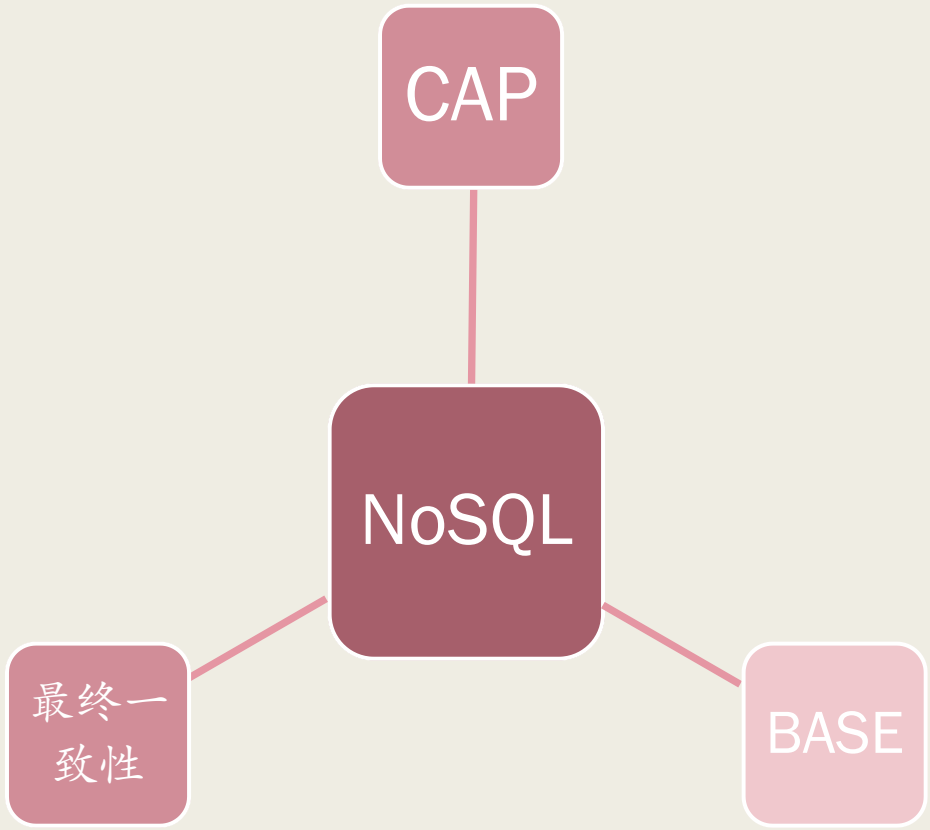
NoSQL与关系数据库的应用场景

总结

关系数据库和NoSQL数据库各有优缺点，彼此无法取代
- **关系数据库应用场景：**电信、银行等领域的关键业务系统，需要保证强事务一致性
- **NoSQL数据库应用场景：**互联网企业、传统企业的非关键业务（比如数据分析）

采用混合架构
- 案例：亚马逊公司就使用不同类型的数据库来支撑它的电子商务应用
- 对于“购物篮”这种临时性数据，采用键值存储会更加高效
- 当前的产品和订单信息则适合存放在关系数据库中
- 大量的历史订单信息则适合保存在类似MongoDB的文档数据库中

NoSQL的三大理论基石



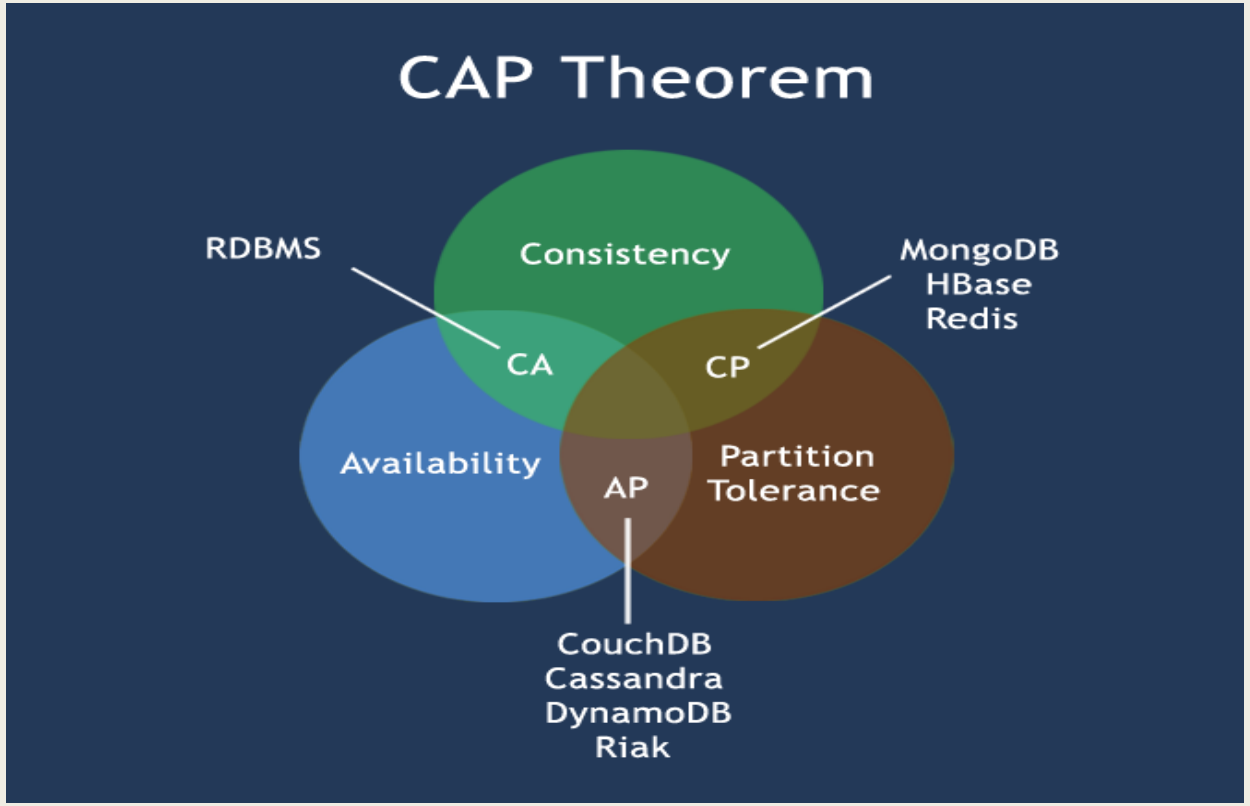
CAP

所谓的CAP指的是：

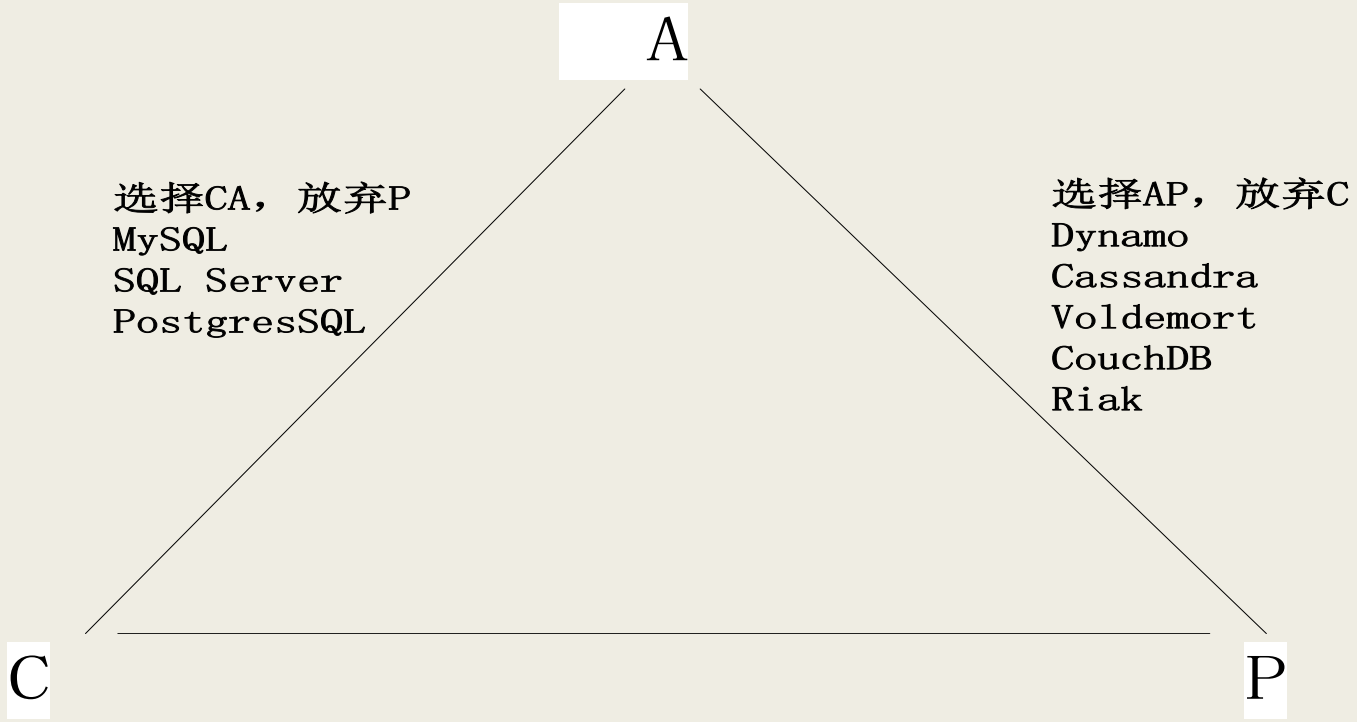
- **C（Consistency）：**一致性，是指任何一个读操作总是能够读到之前完成的写操作的结果，也就是在分布式环境中，多点的数据是一致的，或者说，所有节点在同一时间具有相同的数据
- **A:（Availability）：**可用性，是指快速获取数据，可以在确定的时间内返回操作结果，保证每个请求不管成功或者失败都有响应；
- **P（Tolerance of Network Partition）：**分区容忍性，是指当出现网络分区的情况时（即系统中的一部分节点无法和其他节点进行通信），分离的系统也能够正常运行，也就是说，系统中任意信息的丢失或失败不会影响系统的继续运作。

CAP

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三个需求，最多只能同时满足其中两个，正所谓“鱼和熊掌不可兼得”。



CAP原则的取舍


A
选择CA，放弃P
MySQL
SQL Server
PostgresSQL
选择AP，放弃C
Dynamo
Cassandra
Voldemort
CouchDB
Riak
C
P
选择CP，放弃A
Neo4J, Bigtable, MongoDB, Hbase, Hypertable, Redis

传统事务的ACID（酸）和分布式事务的BASE（碱）

ACID	BASE
原子性(Atomicity)	基本可用(Basically Available)
一致性(Consistency)	软状态/柔性事务(Soft state)
隔离性(Isolation)	最终一致性 (Eventual consistency)
持久性 (Durable)	

BASE含义

- **基本可用**（Basically Availble）

  基本可用，是指一个分布式系统的一部分发生问题变得不可用时，其他部分仍然可以正常使用，也就是允许分区失败的情形出现

- **软状态**（Soft-state）

  “软状态（soft-state）”是与“硬状态（hard-state）”相对应的一种提法。数据库保存的数据是“硬状态”时，可以保证数据一致性，即保证数据一直是正确的。“软状态”是指状态可以有一段时间不同步，具有一定的滞后性

- **最终一致性**（Eventually consistent） 允许后续的访问操作可以暂时读不到更新后的数据，但是经过一段时间之后，必须最终读到更新后的数据。最常见的实现最终一致性的系统是DNS（域名系统）。