

大型网站架构的发展

HeseyWang

About Me

- Software Engineering @ SDU
 - R&D @ Taobao.com
 - Interested in
 - Reading, Riding, Driving
 - Physics, Philosophy, Psychology
 - Anime, Cooking
 - Concurrency
 - Distribute
 - Network Communication
 - Blog: <http://hesey.net>
-

Outline

- 为什么这很重要
- 一个最简单的网站
- 架构要解决的问题
- 压力->应对
- 高性能 vs. 分布式

Outline

- 静态化
 - 缓存
 - 分布式
 - 单点
 - 分库分表
 - 读写分离
 - 异构存储
 - 消息
 - 远程调用
 - 数据中心
-

为什么这很重要

- Java
- 企业级应用（Java EE）
- Android（服务端也许还是.....）
- 我不用Java？
- It also works.

一个最简单的网站

- 想想我们的课程设计
- WebServer: Apache(Lighttpd, nginx)
- AppServer: Tomcat, JBoss, Jetty
- Language: Java(Servlet/JSP), PHP, Python, Ruby
- DataBase: MySQL, NoSQL(MongoDB, Redis, Cassandra, HBase)
- Example: Tomcat + Java + MySQL
- 够不够典型?

洪荒世界

- WebServer + DataBase

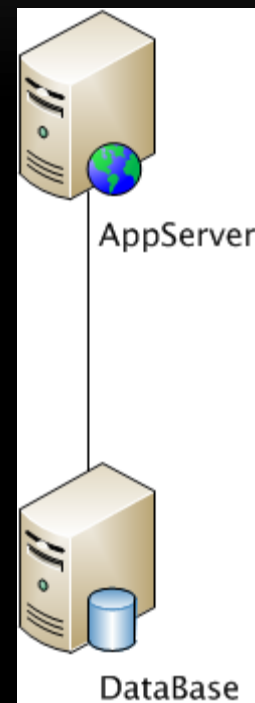


- Tomcat挂了?
- 数据库挂了?
- 系统压力密集: CPU、内存、磁盘I/O、网络
- 发现系统压力越来越大, 页面打开越来越慢



天地初开

- WebServer与DataBase分离
- 压力减小
- 去除不必要的依赖
- 针对不同类型的服务可以设置不同的操作系统参数
- 大家都有了自己的屋子，天下太平

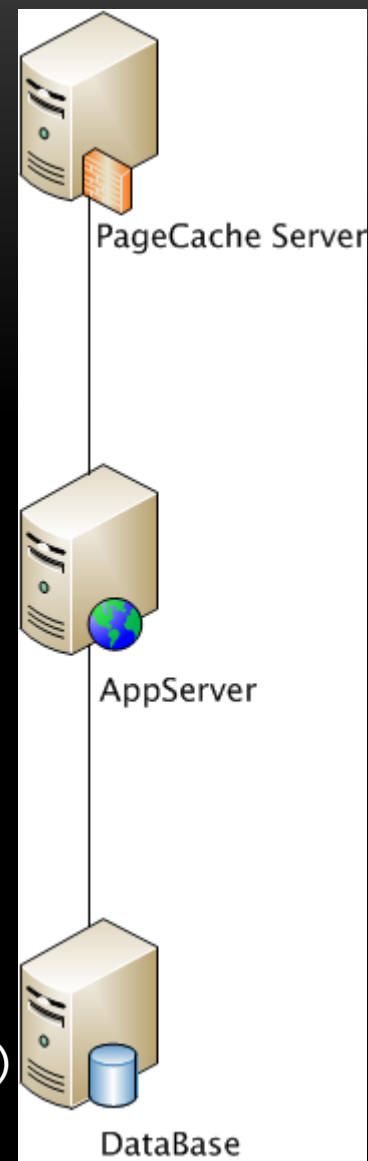


天地初开

- 访问又开始变慢
- 通常不是WebServer的瓶颈
- 排查问题：数据库连接太多，压力大（连不上，连接慢）
- 肿么办？！
- 思考
 - 为什么要连数据库
 - 变量和常量，动态和静态
 - 把静态页面提取出来

页面静态化

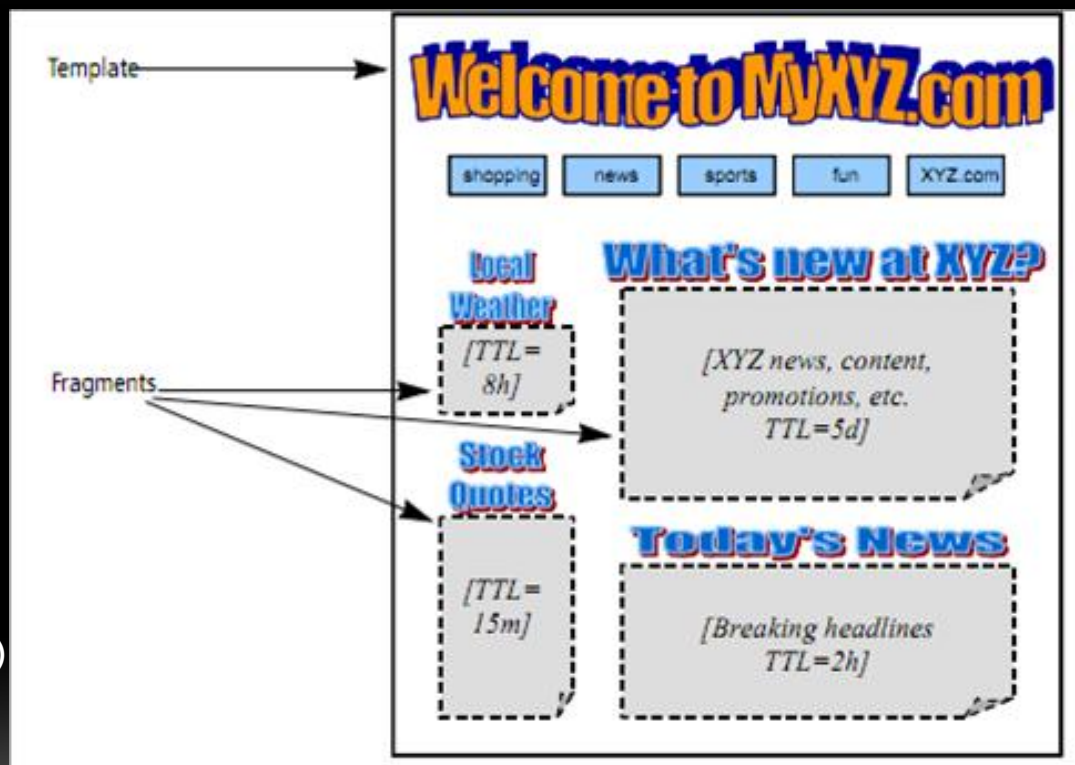
- 缓存服务器 (Squid, HAProxy)
- 作用：直接返回一个普通页面，不涉及任何和数据库的交互
- 数据库表示鸭梨不大
- 例如：首页、新闻页
- 需要关心的问题：缓存的实现、失效算法、内存
- 静态化的另一个好处：CDN(Content Distribution Network)



模块静态化

- 因为页面中有一小部分涉及动态内容，需要和数据库交互
- 无法进行页面静态化

- 模板技术
- 组件的拼接
- 就像文字 + 图片一样
- 全页面静态化 -> 部分静态化
- ESI(Edge Side Include)
- SSI(Server Side Includes)

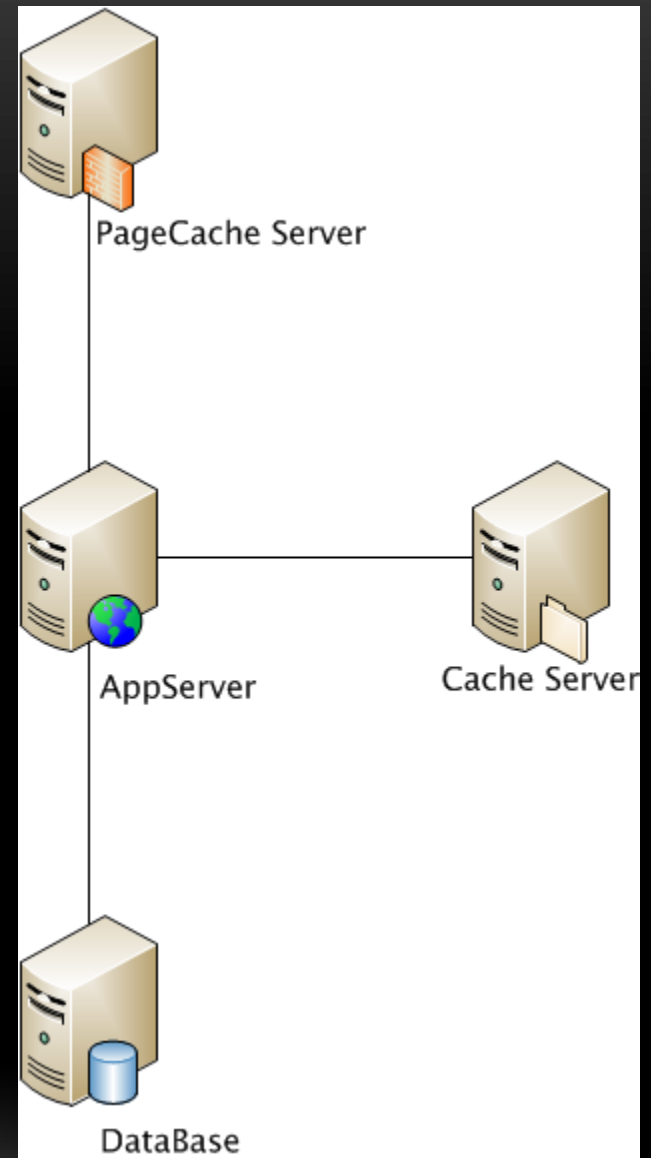


还是缓存

• L1 cache reference	0.5 ns
• Branch mispredict	5 ns
• L2 cache reference	7 ns
• Mutex lock/unlock	100 ns (25)
• Main memory reference	100 ns
• Compress 1K bytes with Zippy	10,000 ns (3,000)
• Send 2K bytes over 1 Gbps network	20,000 ns
• Read 1 MB sequentially from memory	250,000 ns
• Round trip within same datacenter	500,000 ns
• Disk seek	10,000,000 ns
• Read 1 MB sequentially from network	10,000,000 ns
• Read 1 MB sequentially from disk	30,000,000 ns (20,000,000)
• Send packet CA->Netherlands->CA	150,000,000 ns

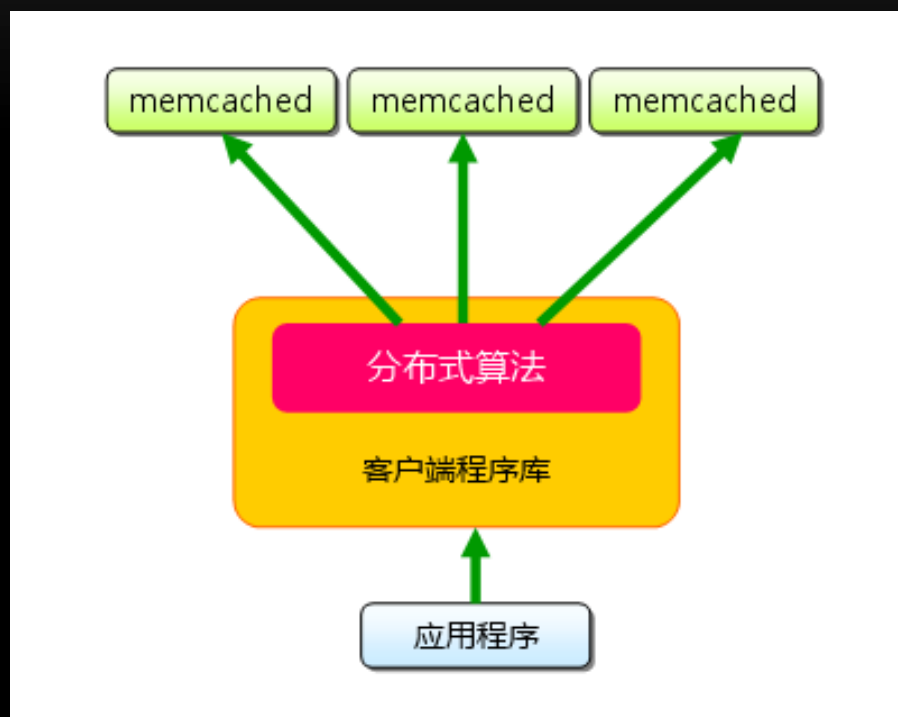
还是缓存

- WebServer层
- 页面缓存了，模块也缓存了
- 重复查询较少变动的数据？
- 用户信息、栏目结构
- 数据缓存
- 本地缓存（Map，框架），内存 vs. DB
- 分布式缓存（memcached）



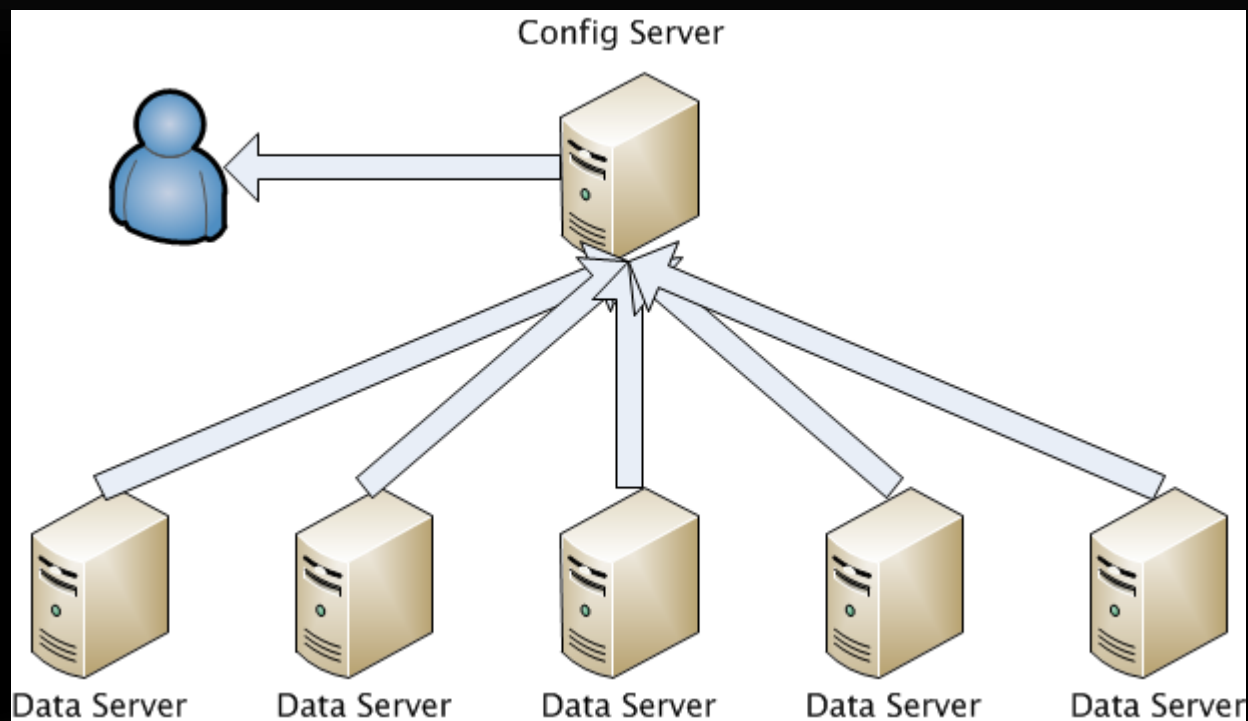
分布式缓存

- 本地缓存不够用了
- 应用服务器压力大，内存小
- memcached
- 一个没有中心节点的由客户端做Hash路由的分布式缓存
- 没有中心节点
- 客户端Hash
- 分布式



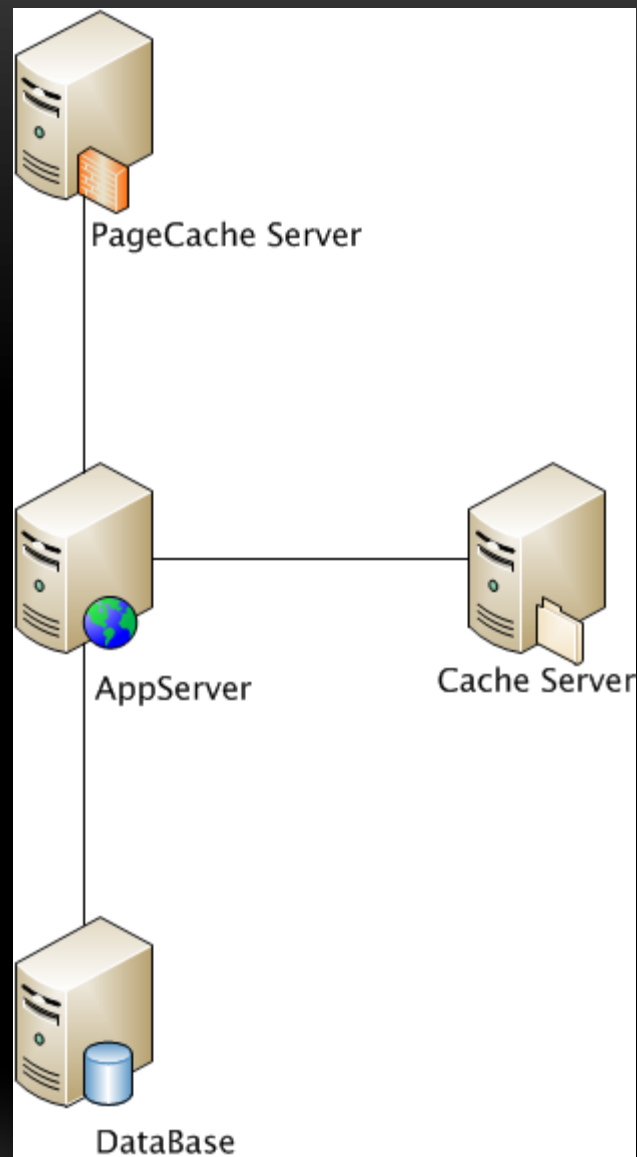
中心节点

- 分布式的含义
- 中心节点——管理者
- 有管理者一定好吗？
- 单点问题
- 没有管理者一定好吗？
- 到底是要闹哪样
- 中心节点的容灾



数据缓存

- 讲了这么多，数据缓存到底是干嘛的
- 过去的访问路径
- 浏览器 -> Tomcat -> 数据库 -> 磁盘
- 现在的访问路径
- 浏览器 -> Tomcat -> 内存直接返回
- 各位修改人人网资料的时间间隔？一周？一个月？
- 没有缓存：查询1000次数据库
- 有缓存：查询1次数据库，访问999次内存



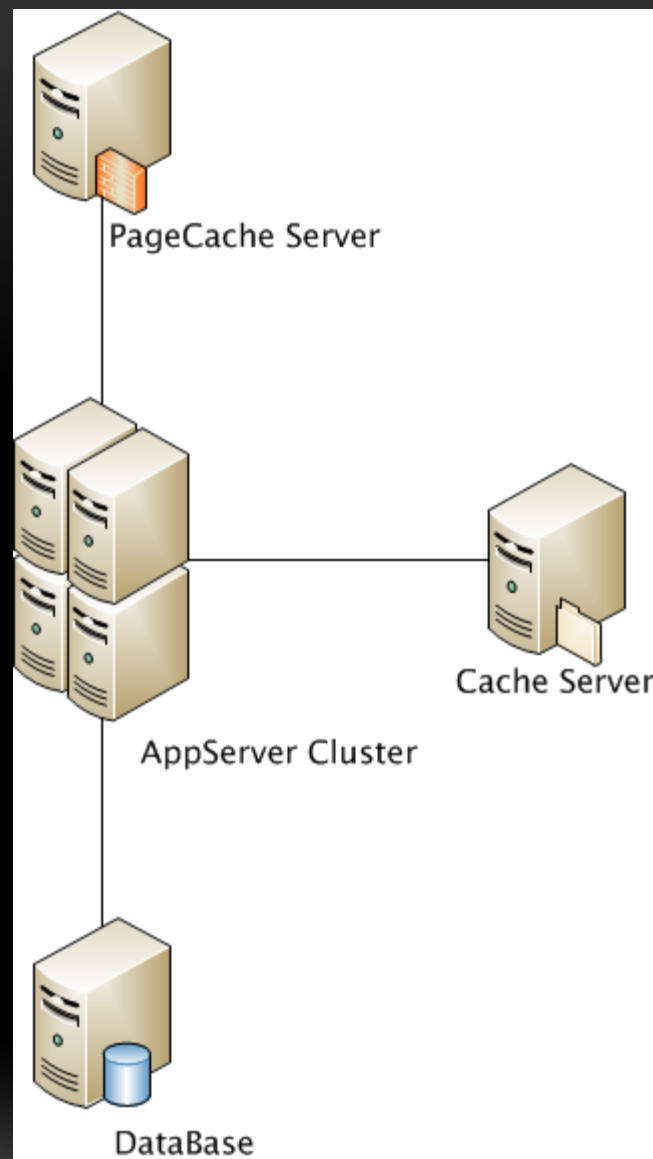
缓存的问题

- 命中率?
- 由内存容量造成
- 80%的访问都包含热点
- 例如：人人网新鲜事第一页、好友列表第一页
- 命中率低的后果
- 数据库中枪
- 缓存 \approx 无效
- 怎么办?
- 结合业务特点，合理放置缓存内容



WebServer的分布式

- 数据库比较清闲了
- WebServer压力上来了
- 加机器
- 会遇到的问题
 - 负载均衡
 - 持久化数据
 - 数据同步
 - 上传文件
- 我们的第一个集群，Cool！



数据库的分布式

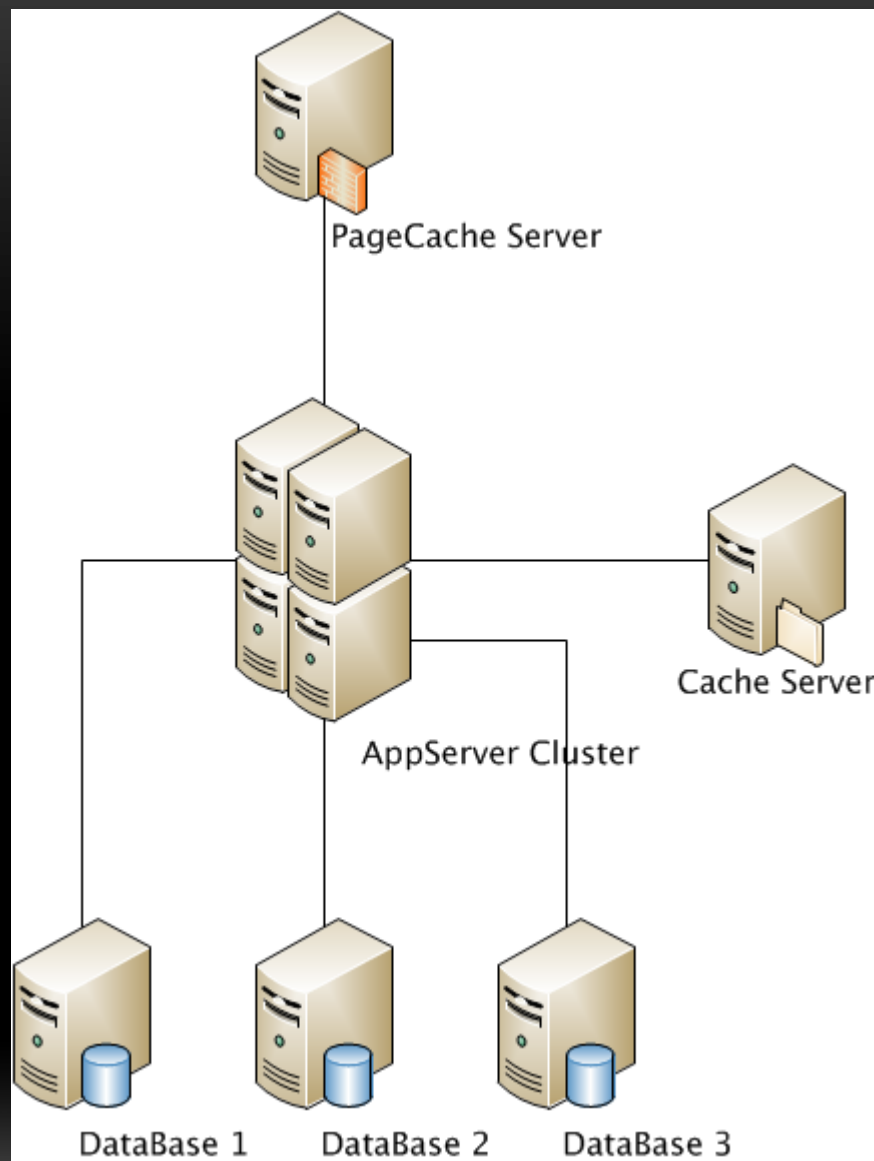
- 分布式
- 很NB的样子
- 搞定了WebServer，用同样的方式搞定数据库
- No.
- 为什么WebServer容易扩展而数据库不行？
- 数据库的分布式
- 分库和分表

数据库的分布式


- 非得分库分表吗？
- 会不会死人？
- 1、压力在查询（读）
 - 主备
 - 读写分离
- 2、压力在插入、更新、删除（写）
 - 分库和分表

分库和分表

- OK，库和表分完了
- 连接哪个库？
- 访问哪张表？
- 傻眼了
- 中间层
- 屏蔽分库分表规则
- 我们的第二个集群

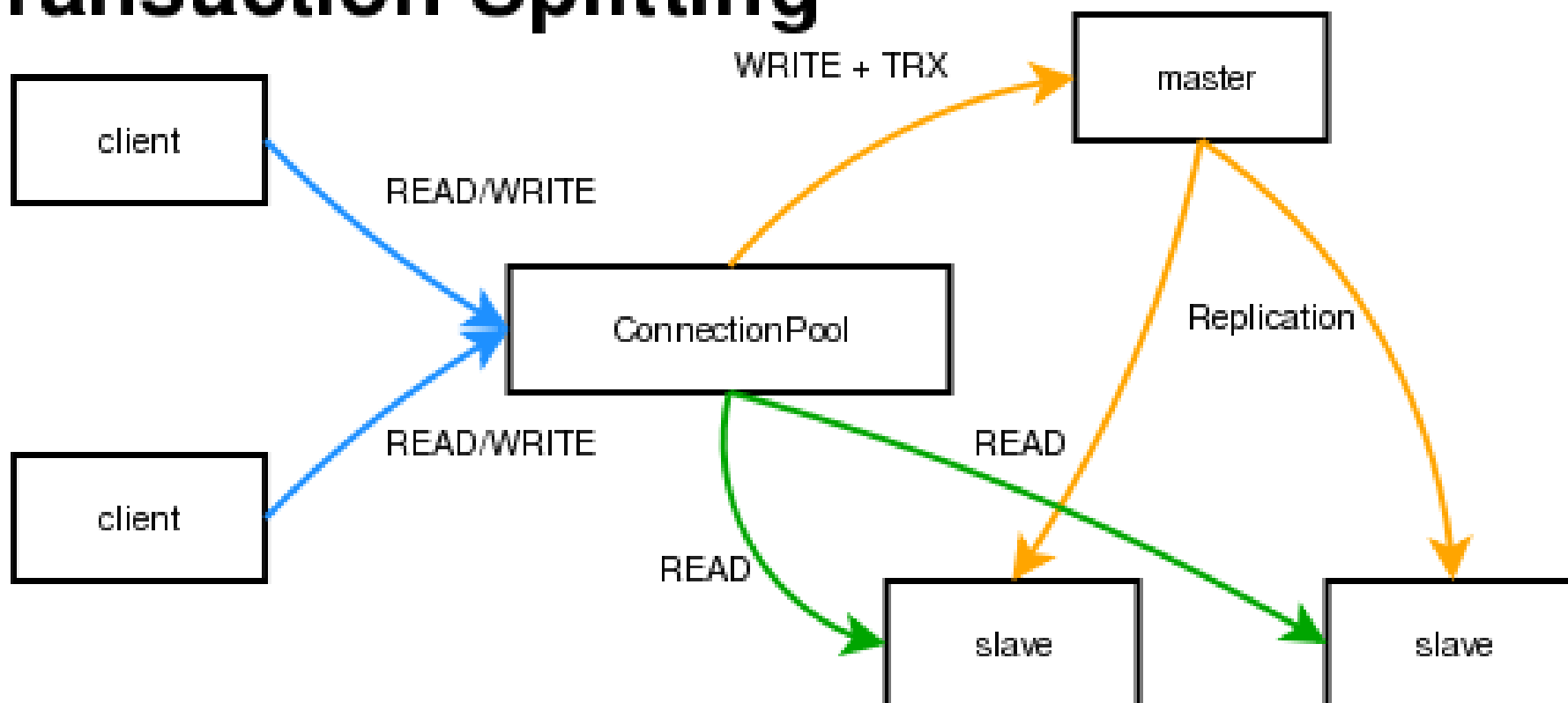


读写分离

- 访问量暴增，幸福感爆满，差点以为自己是下一个 
- 数据库压力增大
- 读写分离
- 事务操作（插入、更新、删除） — Master
- 查询 — Slave
- 主从同步

读写分离

Transaction Splitting



读写分离

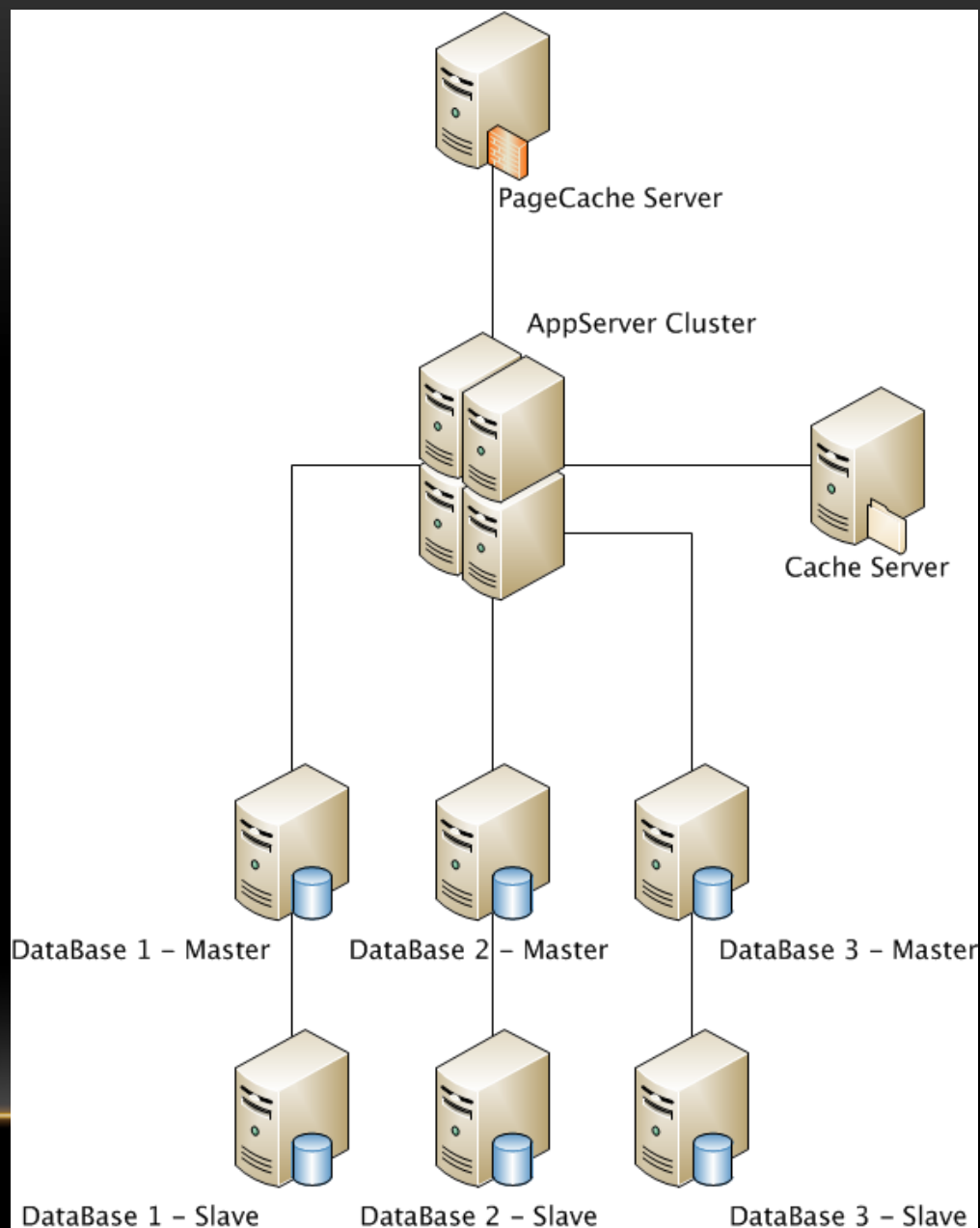
- 为什么读写分离能提高性能？

- 遇到的问题

- 同步延迟
- 实时查询

- 技术点

- 同步
- Semi-Sync
- 请求分派

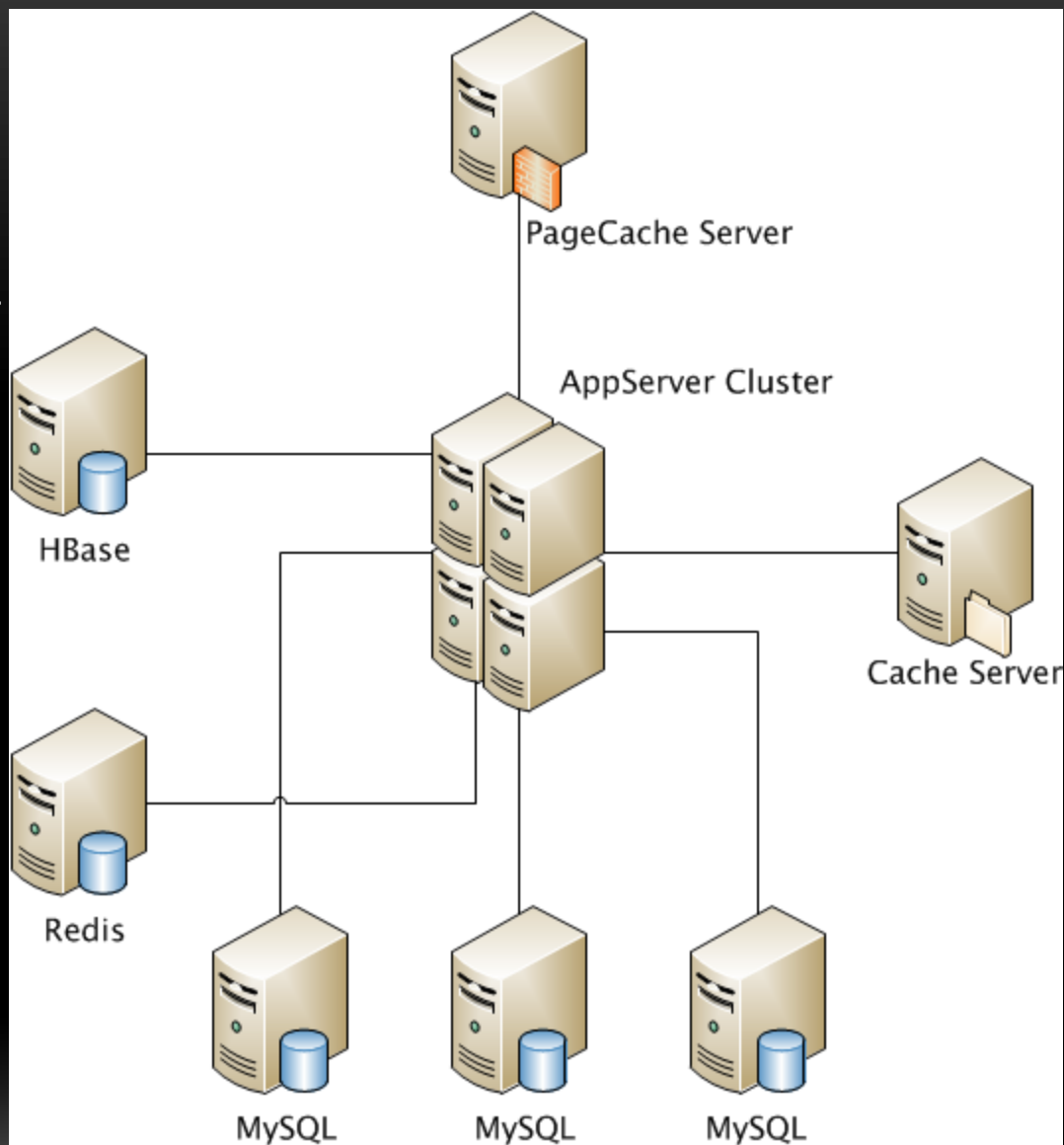


MySQL到头了

- 互联网时代，许多数据只是简单的Key-Value Pair(Map)
- 如果不需要复杂的关系查询，为什么要负担它带来的损耗呢
- SQL解析，关系模型
- 性能
- 海量数据（大块存储）
- HandlerSocket
- BigTable
- Hbase
- Redis

新时代？

- 现在我们有了各种各样的存储
- MySQL, Hbase, Redis
- 发挥各家所长
- WebServer强大的可扩展性



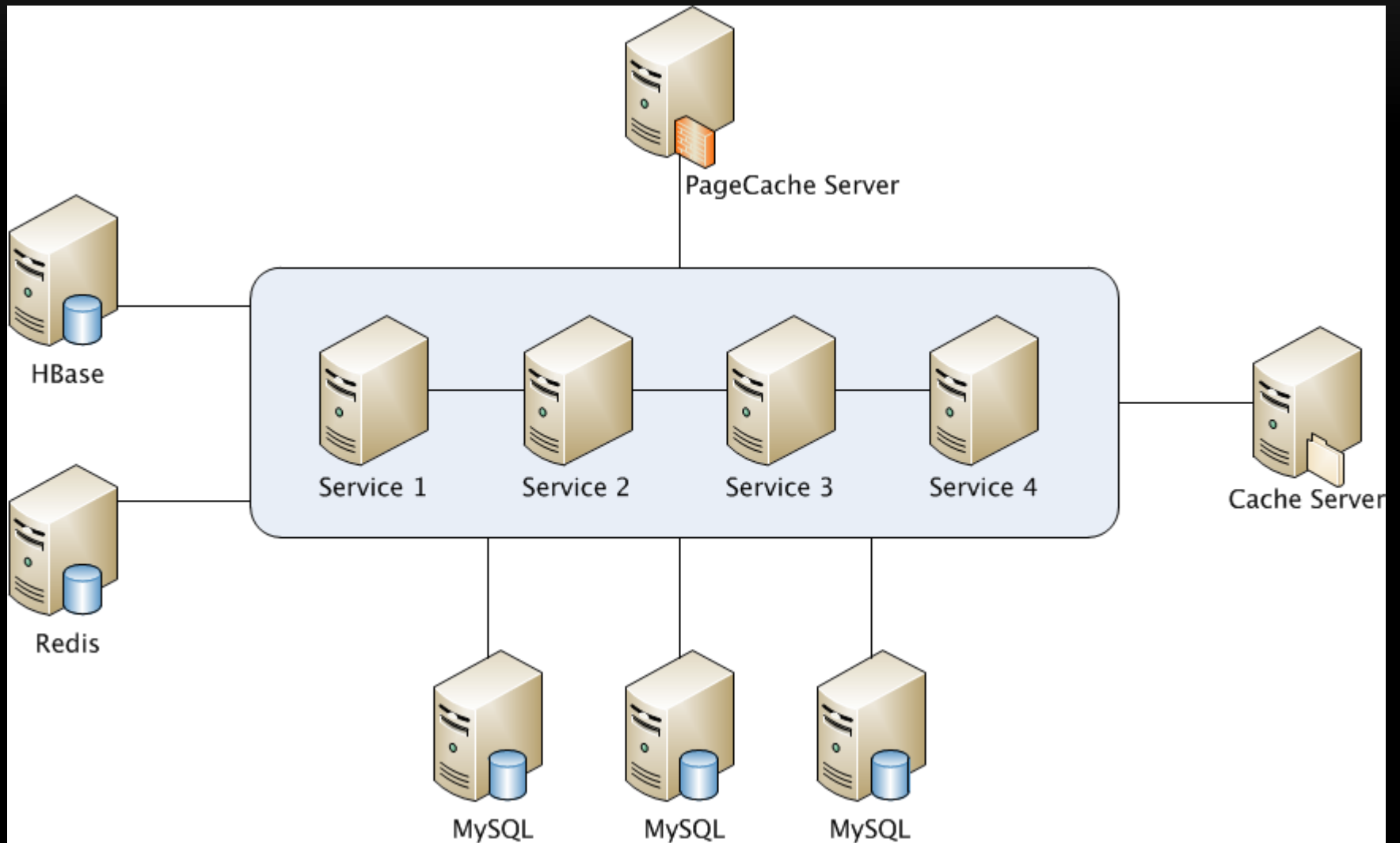
新时代？

- But?
- 之前的道路：一个应用的发展
- 模块耦合度高
- 部署
- 启动
- 排错
- 调优

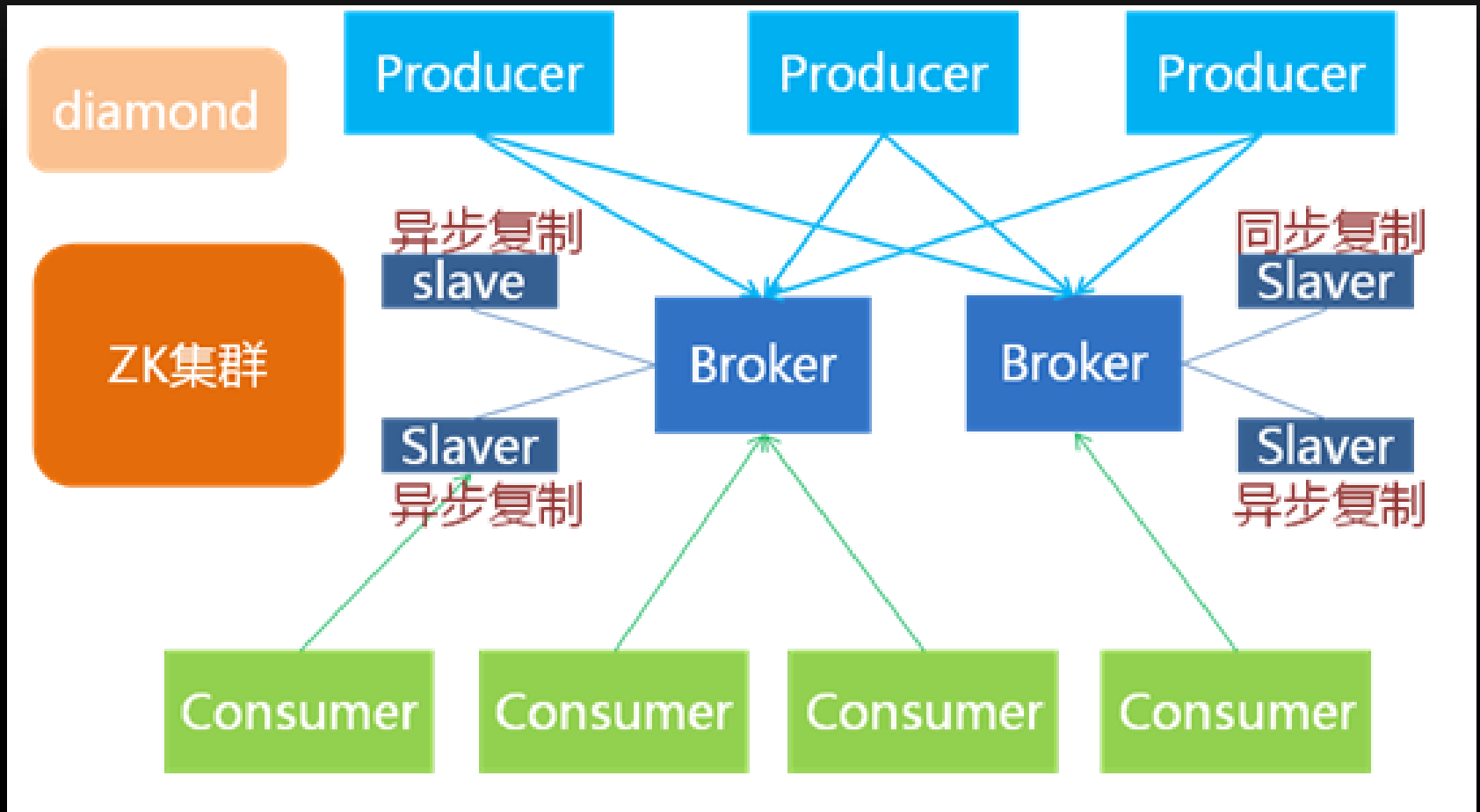
Next Generation

- 拆！
 - 服务化
 - 各司其职
 - 遇到的问题
 - 不同的应用完全在不同的服务器上
 - 通信
 - 调用
 - 共享数据
-

Next Generation



Message

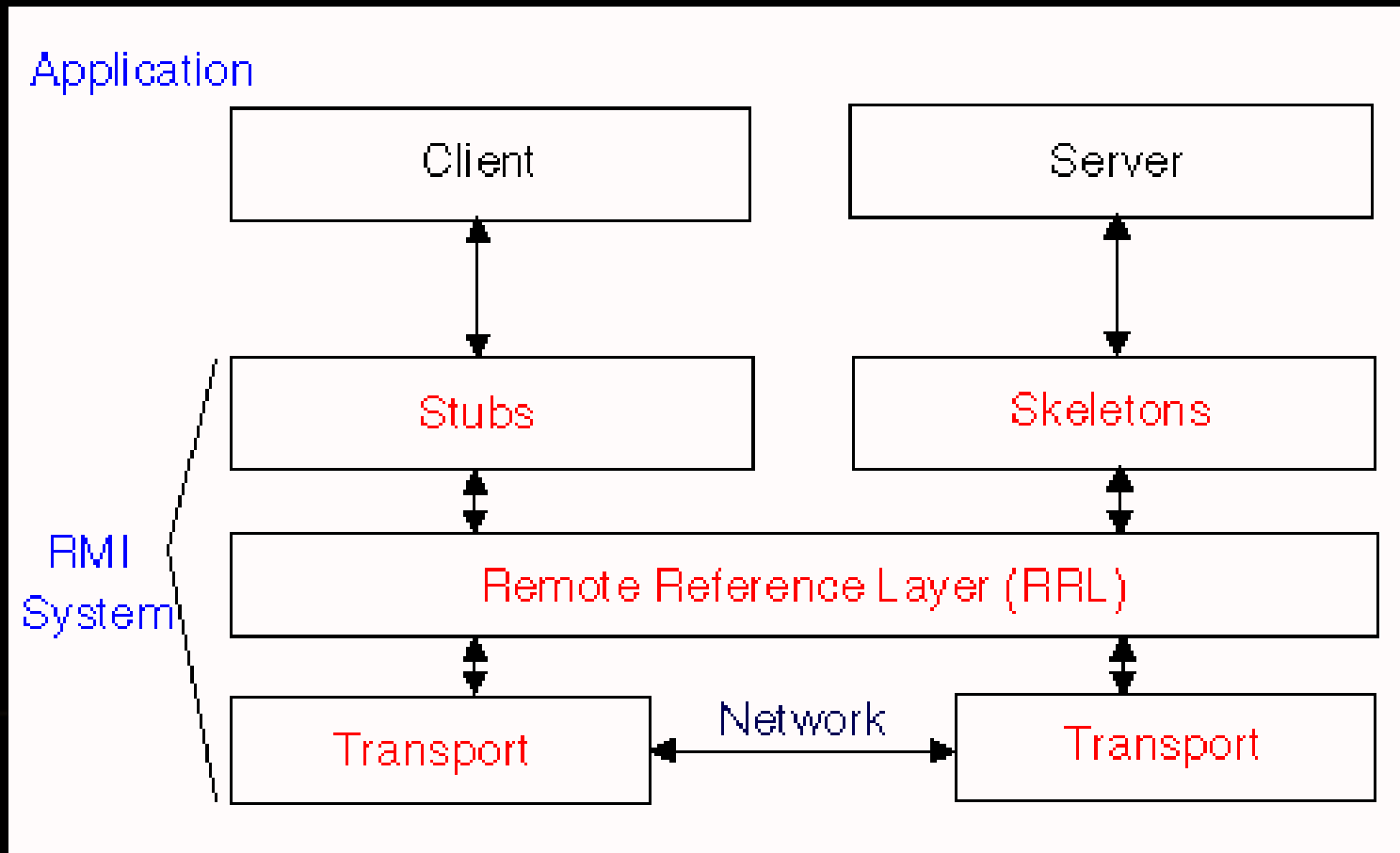


Message

- 消息中间件
- 异步
- 更贴近现实世界
- 小消息有大作用
- 系统扩展的不二法门
- 遇到的问题
 - 顺序投递
 - 重复投递
 - 事务
 - 容灾

RPC

- 远程调用
- 你可能听过：RMI/WebService

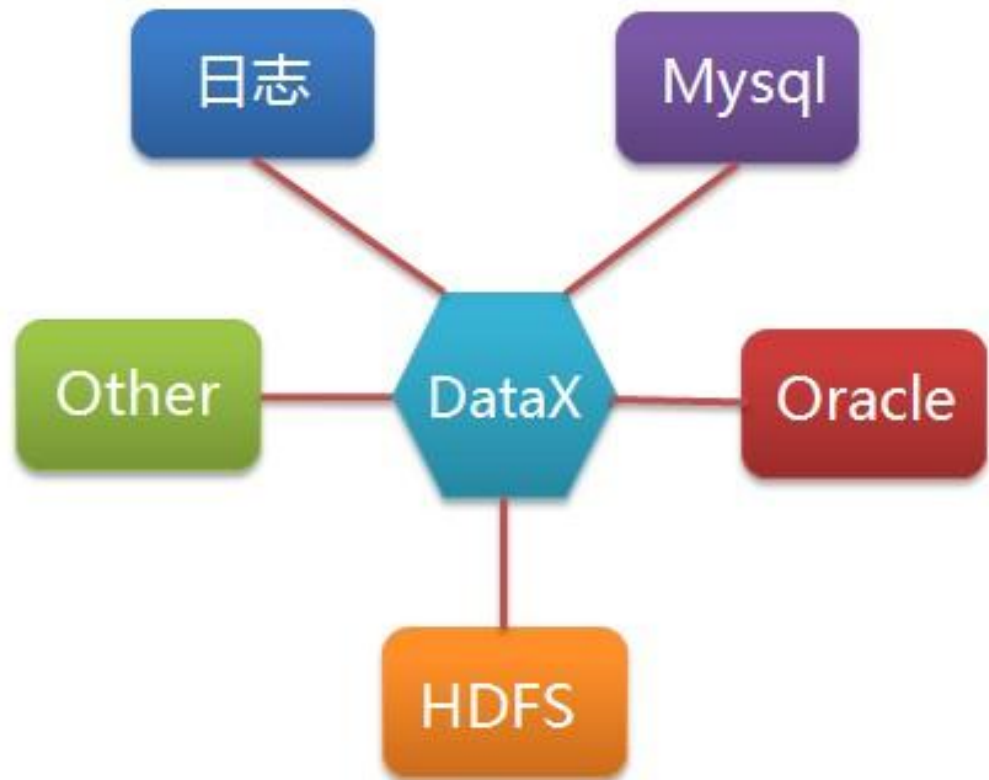


RPC

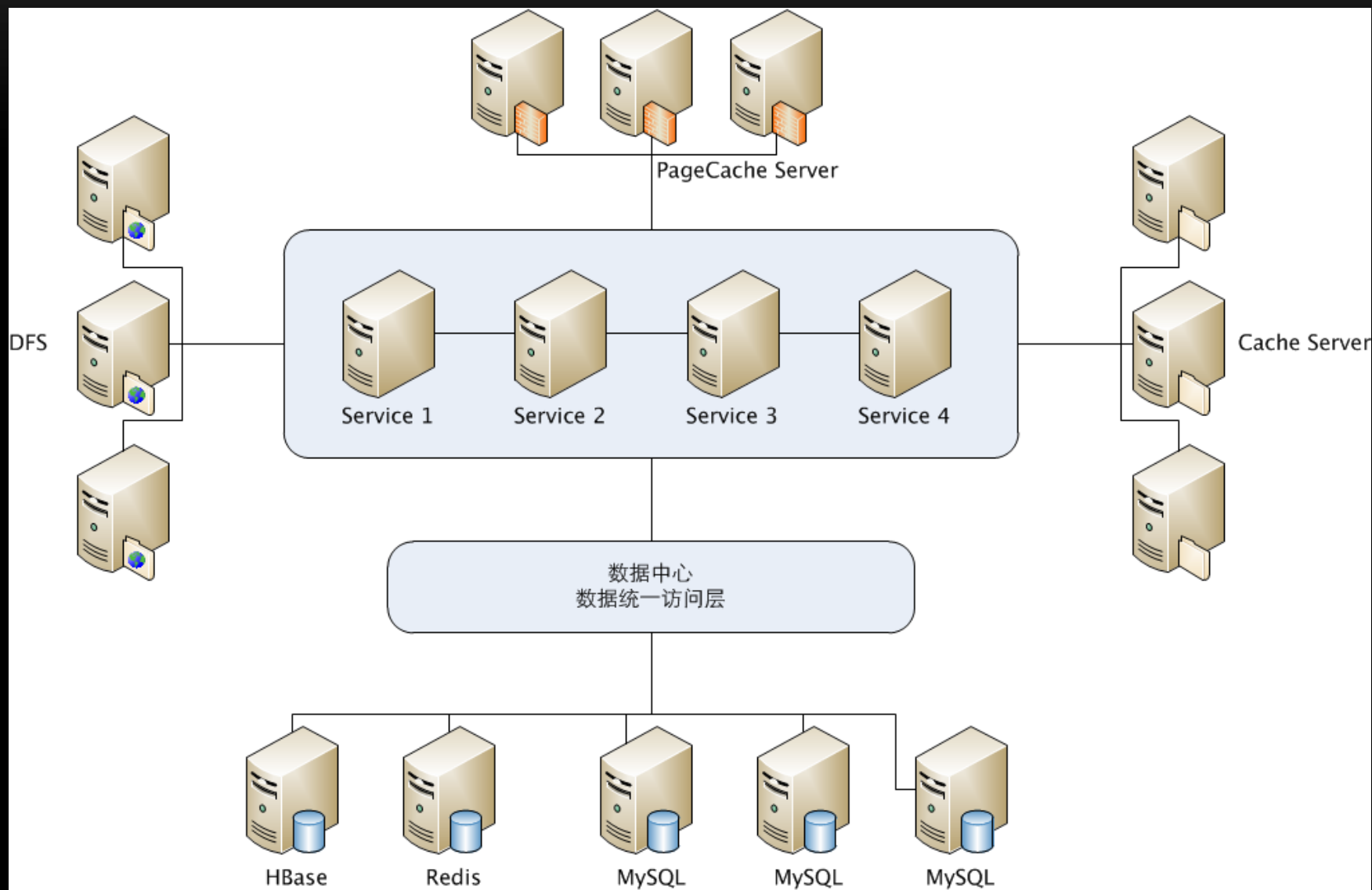
- 形式有很多种
- RMI/HTTP/TCP
- RPC的本质
- 为什么要用RPC

Data Center

- 数据中心
- 异构数据存储的统一接口
- 方便的异构回流和传输
- 服务和数据的解耦
- 便于数据分析
- Hadoop, Hive



Then...



Then...

- 一个分布式系统建成了
- 分别改进
- 有专人负责的应用
- 系统越来越复杂
- 新的挑战
- 运维

- Thank you

- Q&A