架构和设计

有什么以面向对象致富的好方法？继承！

七月在线 林应

2018年3月

主要内容

- 并发编程基础知识
- Map/Reduce与Hadoop
- 分布式系统设计简介
- 分布式存储系统设计案例
- 从数字货币谈共识机制
- 设计模式常见面试题目
- 面试经验分享

上次作业讲解

- 买卖股票的最佳时机II
  - 思路：只要第2天的价格比今天高就是买点啊，不然就是卖点，否则就卖。最后一天显然只能卖。
- 删除数字
  - 思路：对于数字abcd和ABCD，只要A > a，那么一定ABCD更大，即使bcd > BCD也没用。

并发编程基础知识

- 并发模型
  - 抢占式：总控制权在操作系统手中，操作系统会轮流询问每一个任务是否需要使用CPU，需要使用的话就让它用，不过在一定时间后，操作系统会剥夺当前任务的CPU使用权，把它排在询问队列的最后，再去询问下一个任务…
  - 协作式：一个任务得到了CPU时间，除非它自己放弃使用CPU，否则将完全霸占CPU，所以任务之间需要协作：使用一段时间CPU然后放弃，其它任务也是如此，这样才能保证系统正常运行。

并发编程基础知识

- 同步编程原语 (Windows)
  - 用户态：CriticalSection, SRWLock, InterLocked
  - 内核态：Mutex, Event, Semaphore, AsyncIO
- 优缺点
  - 用户态：速度快，无法命名，无法设置超时，容易卡死。
  - 内核态：速度慢（用户/内核态切换），可命名，可设置超时。
- 推荐参考书：Windows Via C++

并发编程基础知识

- Volatile关键字
  - 提醒编译器它后面所定义的变量随时都有可能改变，因此编译后的程序每次需要存储或读取这个变量的时候，都会直接从变量地址中读取数据。如果没有volatile关键字，则编译器可能优化读取和存储，可能暂时使用寄存器中的值，如果这个变量由别的程序更新了的话，将出现不一致现象。无锁编程
- 无锁编程
  - 利用Compare and Swap原子操作
  - 不推荐使用，可移植性和可读性都有问题。

并发编程基础知识

- 异步编程模型
  - 多线程能解决C10K问题吗？
  - 如何利用单线程支持多任务？
  - 建立正确的异步世界观
  - HTTP服务器案例

并发编程基础知识

- 高并发系统如何优化
  - 降低锁的粒度和锁的持有时间
  - 使用读写分离锁
  - 确保顺序获取锁
  - 减少对写共享资源的访问依赖
  - 同步变异步优化CPU使用率，避免过多的线程切换浪费CPU资源。

Map/Reduce简介

- 为什么要Map/Reduce？
- Map/Reduce是什么
  - Map：分解工作，把一个大任务分解成独立不相关、可以并行的子任务并完成处理。
  - Reduce：把每个子任务的处理结果汇总成最终的结果。
- 单机版矩阵乘法的例子
- 单机版topK计算的例子

Map/Reduce简介

- Hadoop与Spark的区别
  - Hadoop实质上更多是一个分布式数据基础设施：它将巨大的数据集分派到一个由普通计算机组成的集群中的多个节点进行存储，意味着您不需要购买和维护昂贵的服务器硬件。
  - Hadoop还会索引和跟踪这些数据，让大数据处理和分析效率达到前所未有的高度。
  - Spark，则是那么一个专门用来对那些分布式存储的大数据进行处理的工具，它并不会进行分布式数据的存储。

Map/Reduce简介

- 为什么Spark速度更快？
  - MapReduce是分步对数据进行处理的：从集群中读取数据，进行一次处理，将结果写到集群，从集群中读取更新后的数据，进行下一次的处理，将结果写到集群，等等…
  - Spark会在内存中以接近“实时”的时间完成所有的数据分析：“从集群中读取数据，完成所有必须的分析处理，将结果写回集群，完成。
- Hadoop可以和Spark一起工作，也可以各自独立与其他工具组合。

分布式系统设计简介

- 关于CAP问题
  - C = Consistency 一致性
  - A = Availability 可用性
  - P = Partition Tolerance 容错性
- CPA三个指标不可能被同时满足，最多同时满足其中两项。

分布式系统设计简介

CAP场景分析

- 假设我们使用一台服务器A对外提供存储服务，为了避免服务器宕机导致服务不可用，我们又部署了服务器B做为备份。每次用户往A写入数据时，A同步在B上写入备份。用户在读取数据时可以任意从A或B读取写入的历史和最新数据。
- 故障发生，A和B之间网络中断导致无法通信，用户往A写入数据时，B没有更新，针对这种情况我们策略该如何设计？

分布式系统设计简介

- CAP场景分析
  - 牺牲可用性A：A停止数据写入服务，但是依然可以读取，一致性得到保证。
  - 牺牲一致性C：A选择不把数据写入B就向用户返回操作成功。可用性得到保证。
  - 弱CPA可以满足，比如放弃强一致性而选择最终一致性。比如淘宝大促时的各种限流开关。

分布式系统设计简介

- 传统服务架构的局限性
  - 单机性能处理有上限
  - 负载分配不均衡。
  - 开发、升级成本高
  - 部署成本高
  - 维护成本高

分布式系统设计简介

- 微服务架构简介
  - 拆分服务，通过RPC通讯进行协同工作
  - 负载更均衡，服务弹性更强。
  - 理论吞吐量更高
  - 开发部署效率更高
  - 缺点
    - 通讯成本高
    - 依赖导致的部署成本
    - 服务治理和故障调试

分布式系统设计简介

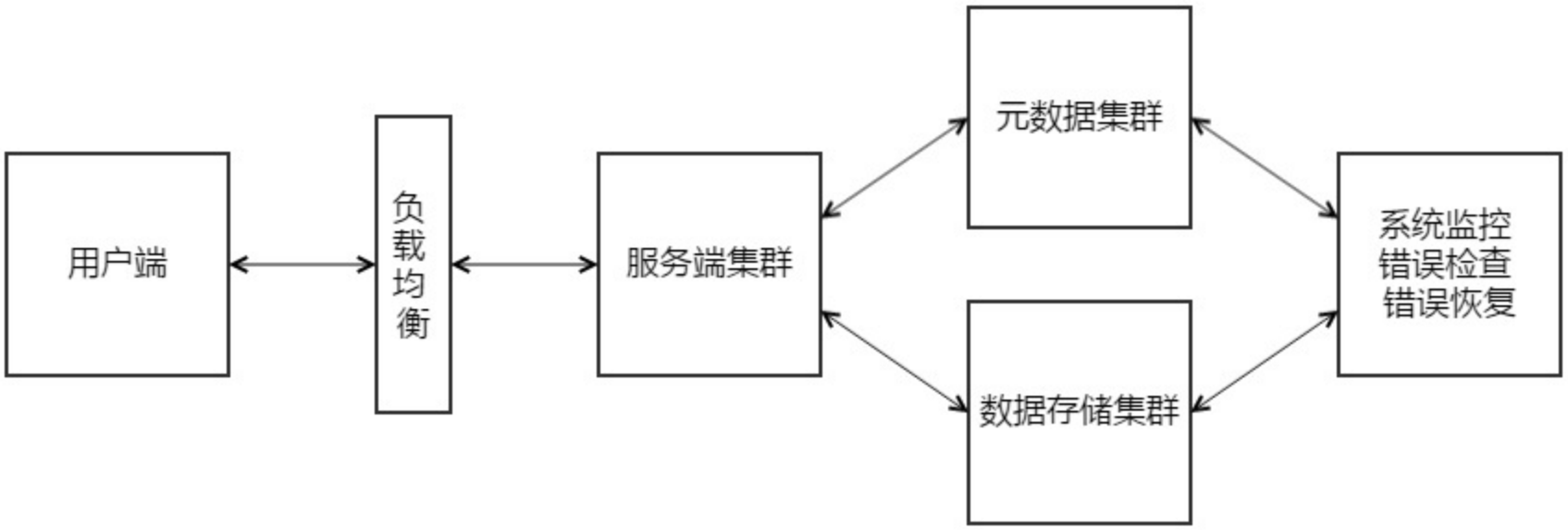
- 分布式系统常见优化方案
  - 动静分离
  - 多级访问，缓存读优化
  - 牺牲强一致性，改为最终一致性。
  - 同步变异步，通过队列传递消息。

分布式存储系统设计案例

- 常见挑战
  - 数据的高效检索
  - 数据的高效存储
  - 数据冗余与效率的平衡
  - 灾难恢复
  - 小文件存储优化

分布式存储系统设计案例

架构参考



分布式存储系统设计案例

- 解决方案
  - 元数据(meta data)通过数据库独立存储，通过元数据访问实际存储数据。
  - 增量存储与压缩合并
  - 通过专门算法降低冗余存储
  - 运维脚本实时监控扫描数据完整性
  - 小文件直接数据库存储，或者合并存储，在元素据中记载偏移量和大小信息。

从数字货币谈共识机制

- 比特币的区块链本质是分布式账本
- 分布式账本面临谁来记账的问题
- 比特币运行的环境毫无信任可言
- 比特币愿意支付什么代价
  - 计算成本
  - 不要求实时计算，效率可以牺牲。

从数字货币谈共识机制

- 比特币的区块链本质是分布式账本
- 分布式账本面临谁来记账的问题
- 比特币运行的环境毫无信任可言
- 本质是个分布式事务问题
- 比特币愿意支付什么代价
  - 计算成本
  - 不要求实时计算，效率可以牺牲。

从数字货币谈共识机制

- 共识机制的实现
  - 完成工作量证明，首先证明的拿到记账权。
  - 将交易记录打包生成新区块，同时广播告诉其它节点。同时系统对记账者发放奖励，俗称“挖矿”。
  - 其它节点收到广播后对新区块就行校验，接收后继续广播，最终全网接收，交易确认。

设计模式常见面试题目

- 如何编写一个线程安全的单例？
- 什么是IoC？IoC的优点与应用举例。
- 工厂模式应用场景
- 面向接口编程
- 开放闭合原则的应用实例
- 推荐教材：大话设计模式

面试经验分享

- 精简简历，突出重点。
- 不打无准备之仗
- 白板编程先确认题目，明确思路，先实现再优化。
- 架构问题想办法往经典套路靠拢

作业

- 阅读Windows Via C++并发编程部分内容
- 阅读大话设计模式并实践全部例子代码，了解各个设计模式实现原理和应用场景。
- 配置单机版hadoop并跑通基本的单词计数案例。参考链接：
  - https://www.cnblogs.com/vincentzh/p/5967274.html
  - https://blog.csdn.net/anneqiqi/article/details/51189513

感谢大家！

恳请大家批评指正！