第八周总结

首先是第八周的课堂笔记：

8.1 文件与硬盘I/O：如何把硬盘的读写速度提高十万倍?

1. 机械硬盘可动部分有磁头和盘片，磁头切换磁道很慢。机械硬盘读取连续的数据比离散的快得多。
2. 固态硬盘比机械硬盘快得多，并且数据连续在固态硬盘也有好处
3. B+树：每个节点是一个很大的数据块，有多个指针指向下一级，一般来说，几千万数据用这种方式存储，树的高度也只有三四级，也就是说，访问磁盘一般就三四次。不过根据索引找到数据地址后，由于目标数据分散在不同地方，读起来就很慢。导致每次查找又需要几十毫秒时间。
4. LSM树：Log Struct Merge tree。日志合并树。在内存中先合并一个树，合并到一定大小后，再刷到硬盘中去。
5. 操作系统对文件的管理：
   1. 操作系统每次至少分片一个硬盘块，即4K。
   2. 文件控制块：有个列表记录每个文件有哪些属于它的块。
6. RAID：独立硬盘冗余阵列。
   1. 业界用得较多的是RAID5：7个数据位+1个校验位（异或），校验位螺旋式分布在8个磁盘中（为了减小校验位磁盘的访问压力）
   2. RAID 6:2个校验位，比RAID 5的可靠性高，但磁盘利用率相对降低
7. 分布式文件系统hdfs：以服务器为单位记录分片
   1. Namenode：文件控制块的角色
   2. dataNodes：真正存储数据的节点
   3. 一般来说会有三备份，还会跨机架复制数据
   4. 当datanode不可用的时候，namenode根据id的记录，迅速从其他服务器找到数据并复制一个新的备份

8.2 常见数据结构与哈希表原理

1. 时间复杂度，空间复杂度
2. NP问题
3. 数组，链表，哈希表（哈希碰撞攻击，这。。。。），栈，队列（公平锁）

8.3 红黑树原理与性能特性

1. 树，二叉排序树，平衡二叉树，旋转二叉树平衡（左旋，右旋，左右旋）
2. 红黑树：增删的时候需要旋转和染色
   1. 每个节点只能是红或者黑
   2. 根节点是黑的
   3. 叶子结点是null，且是黑的
   4. 从根到叶，红色节点不会连续出现
   5. 从一个结点出发到任何能到的叶子结点，这条路径上都有相同数目的黑色节点
3. 跳表：规则简单，效率高，但是空间复杂度略高

8.4 经典算法

1. 常用算法：穷举，递归，贪心（这个感觉一点都不常用），动态规划
2. 遗传算法：染色体编码 -> 适应函数+控制参数 -> 选择算法 -> 交叉遗传。 => 这个算法应该是用于求非常高复杂度问题的近似最优解的。

8.5 网络通信基本原理与性能优化

1. 一次通信历程： 用户 -> DNS解析 (-> CDN) -> 负载均衡 -> 反向代理 -> 负载均衡 -> 网关 -> 微服务 -> 原路返回
2. OSI的七层网络协议和TCP/Ip的四层网络协议。在编码实现上，每一层好像是在直接和对应主机的相应的层通信，实际上上层通信依赖于下层通信，每一层都在添加自己的传输控制协议。
3. 网络数据包格式：
    1. 链路层协议头：本机mac，远程mac
    2. IP协议头：本机IP，远程IP
    3. TCP协议头：本机进程监听端口，远程进程监听端口
    4. HTTP协议头：方法：POST。path：xxx等
    5. 数据包：用户数据
4. 逐层看一下
    1. 物理层：负责数据物理传输。主要解决的问题是，抽象底层的逻辑，让光纤，电缆以及其他的无线通信设备对于顶层全部以0 1 的格式接口进行通信
    2. 链路层：主要是将数据封装成帧，以帧为单位进行数据传输。定义帧的大小。添加发送端和接收端的mac地址。
        1. 链路层负载均衡：负载均衡服务器修改目标mac地址，发给不同的机器，但是不修改源mac地址，因此返回包不经过负载均衡服务器直接回到访问端。注意此时负载均衡服务器和目标服务器共享相同的IP地址，但它们的mac地址不同
    3. 网络层：根据IP地址进行路由，找到目标服务器
        1. 基于IP的负载均衡：负载均衡服务器收到请求后把目标地址改成目标服务器的ip地址，把源地址改成自己的ip地址。收到目标服务器的返回后再通过修改ip地址把结果还给调用方。由于目标服务器的相应会经过负载均衡服务器，性能低于链路层负载均衡。
    4. 传输层（如TCP协议）：IP不是可靠的通讯协议，不会确保数据送达，要保证通信稳定可靠要依赖传输层协议，如TCP。当然，也有重视速度不一定可靠的，如UDP。
        1. 复习一下TCP相关的东西：有序，拥塞控制，流量控制，失败重传等；3次握手建立连接，4次挥手关闭链接
    5. 应用层（最广泛的如HTTP协议，Dubbo协议）：HTTP从1.0到1.1到2到3的过程中，每一代都在改进性能，到了HTTP3已经不再使用TCP，而是改用UDP，以防止接收结果时的阻塞等待。
8.6 非阻塞网络IO
1. 网络编程其实是利用操作系统提供的接口进行编程。操作系统已经封装了传输层及传输层以下的内容的实现。（socket）
2. 每个客户端一个socket，对于每个socket可以启动多个线程去写，以提高并发度。
3. Socket接收数据的内核过程：网络数据包 -> 网卡 -> 中断 -> 执行中断程序 -> 拷贝数据到socket接收缓冲区 -> 唤醒阻塞在这个socket上的线程。
4. 非阻塞IO
    1. Java IO：一个while true死循环，很多地方都会导致这个循环被阻塞，socket的accept等待，读数据等待，写数据等待，一个线程处理不了太多东西
    2. Java NIO：关键在于select.selectKey()，只有当事件到达的时候才去处理，一个线程通过某种select策略不停地扫描其所注册的socket，并完成数据在socket和buffer之间的搬运工作。
    3. Selector策略：epoll（有事件时放进列表），没有任何事件时，selector会阻塞
    4. 优点：可以建立大量的连接，比如nginx可以同时运行上万个连接，tomcat则不行

个人总结

第八周主要讲了三个话题：硬件存储，算法，以及网络。

1. 从硬件角度来讲，首先是硬盘本身可以通过RAID的方式来优化，使得速度成倍提高。达到瓶颈后需要使用分布式存储，比如hdfs，来支持更大容量以及更快的读写，但是这种方式消耗的资源也会多得多。
2. 算法方面，可以说是粗略讲了一下涵盖的知识点，一维的数组，链表，跳表，二维的树，以及常用的递归，动态规划等算法。这里面每一个点其实都可以有大量的展开，想要运用自如，还是需要不少的训练的。
3. 网络方面，简单介绍了七层网络架构的原理，从性能优化的角度出发，主要有各个层的负载均衡，协议的选择，以及非阻塞的异步IO等。