# 第七周学习总结

## 首先是课程笔记：

**7.1 性能测试：系统性能的主要技术指标**

1. 性能参数：性能优化的前提和基础，以及优化结果的度量和标准。
2. 性能指标主要有相应时间，并发数，吞吐量，性能计数器等。
3. 性能测试是一个总称，具体可分为性能测试，负载测试，压力测试，稳定性测试
4. TPS随着并发的增加先增加后减少，这个主要是压力过大，资源不足时，服务端的线程切换，虚拟内存切换等消耗增加，到了极限的时候系统就会开始崩溃。
5. 响应时间随着系统压力的变大也是抛物线式地增加。这也是系统资源不足所导致的线程抢占调度问题。
6. Flower异步框架下，同一个集群的不同服务之间能够做到不会相互影响。即一个服务下游rt变长导致的崩溃不会影响其他服务（这个感觉确实解决了一个很大的痛点了）

**7.2 全链路压测的挑战**

1. 全链路压测是在特定场景下，将相关链路完整地串起来同时施压，尽可能模拟出真实的用户行为，当系统整站流量都打上来的时候，必定会暴露出性能瓶颈，才能够探测出系统整体的真实处理能力，以及有指导的在大促前进行容量规划和性能优化，这便是线上实施全链路压测的目的。
2. 难点：
    1. 压测数据如何构造，并且和真实的更接近。
    2. 全链路压测直接在线上真实环境模拟，怎么保障对线上无影响?
    3. 大促活动的巨大流量怎么制作出来?
3. 数据构造：用户真实数据进行清洗，脱敏，按比例放大构造等。
4. 数据隔离：逻辑隔离（会污染线上数据）；虚拟隔离（所有写数据的地方mock，不真实去写，但是对压测结果有影响）；物理隔离（所有的存储，缓存，搜索引擎等使用专门的存储去承担压测流量）
5. 流量构造：利用CDN服务器去模拟用户的请求，用于模拟用户的位置，请求，高并发等。

**7.3 性能优化的分层思想**

1. 软件性能优化的2个原则：你不能优化一个没有测试的软件；你不能优化一个不了解的软件
2. 性能测试主要指标：响应时间，并发数，吞吐量，性能计数器
3. 性能优化的一般方法：
    1. 性能测试，获得性能指标
    2. 指标分析，发现性能与资源瓶颈
    3. 架构与代码分析，寻找性能与资源瓶颈关键所在
    4. 架构与代码及其他优化，优化关键技术点，平衡资源利用
    5. 性能测试，进入性能优化闭环
4. 系统性能优化的分层思想
    1. 机房与骨干网络性能优化（性能+可用性）：异地多活的多机房架构，专线网络与自主CDN建设

2. 服务器与硬件性能优化（需要分析网络吞吐，磁盘吞吐等）：更优的CPU，磁盘，内存，网卡，对软件的性能优化可能是数量级的，有时候远超代码和架构的性能优化

3. 操作系统性能优化（分析CPU利用率，用户态和操作系统态耗时）：系统态耗时太多就需要改操作系统的一些参数了，比如Linux的transparent huge page

4. 虚拟机性能优化：比如java，不同的GC对性能影响很大

5. 基础组件性能优化：jboss，apache，mod_jk+，DBCP，jetty等基础组件的升级也能带来几倍的性能优化和提升

6. 软件架构性能优化：缓存（读）；异步（写）；集群（负载均衡）

7. 软件代码性能优化：这里关键点我感觉是要去分析性能出现问题的时间段服务器在做啥，比如spark优化的例子，多个executor重复加载同一个17M的基础组件造成系统14s时间内啥都没做

1. 并发编程，多线程与锁
2. 资源服用，线程池与对象池
3. 异步编程，生产者，消费者
4. 数据结构，数组，链表，hash表，树

**7.4 操作系统：计算机如何处理成百上千的并发请求?**

1. 进程的运行期状态：运行；就绪；阻塞。处于运行状态的进程数目小于等于CPU的数目。阻塞状态多的时候，程序并发能力会下降，进而导致系统运行达不到我们的预期目标。
2. 一般服务端应用是一个进程多个线程，以减少切换的代价。
3. 线程栈：每个线程都有自己的栈，函数一直运行与栈顶，不同的函数有着不同的栈位置，其局部变量也得以区分。
4. Java web应用时序：物理服务器 –> 操作系统 –> JVM进程 –> Tomcat容器 –> 应用程序（每个用户请求分配一个线程）
5. 阻塞会导致高并发系统崩溃。这是由于某个线程的阻塞会在临界区阻塞其他的线程，每个线程所申请的资源也不能释放，线程本身由于总数目有限，也被浪费了。
6. 如何避免阻塞引起的崩溃：
   1. 限流：控制进入计算机请求数，从而减少创建的线程数
   2. 降级：关闭部分功能程序的执行，今早释放线程
   3. 反应式：异步；无临界区

**7.5 锁原语 CAS与各类锁**

1. CAS锁原语：比较以及设置，由底层（通常是CPU指令）保证两步连续操作不可中断
2. Java 通过CAS原语在对象头中修改Mark Word实现加锁 –> synchronize 背后有正常，偏向锁，轻量级锁，重量级锁的过程，这个过程代价会越来越大。偏向锁通过CAS自动获取，轻量级锁下面线程会有CAS自旋的行为，不阻塞，重量级锁会让当前线程和后续线程进入阻塞，由监视器唤醒，性能降低。并发越高，获得锁就越难，锁升级的可能性就会增大，性能容易下降。

**7.6 案例：异步并发式编程框架akka**

1. Akka：异步并发分布式编程框架
2. Actor：万物皆Actor，底层是一个dispatcher的线程池，这个线程池支持了数百万个actor的并发。
3. 所有actor之间的交互仅通过发消息完成
4. 垂直和水平伸缩可以完全通过部署上的配置来实现。

## 个人总结

这周主要讲了性能测试以及基于性能测试的各种优化方案。性能测试的要点在于并发，要搞清楚并发和QPS的区别，对于rt短的服务，小的并发也可以有很高的QPS，但实际上并发才是决定服务是否会跨的关键。

性能优化不能只局限于代码本身，从骨干机房，到硬件，操作系统，虚拟机，基础框架等每一个环节都有可以压榨性能的地方。