# 性能调优常规手段

**性能调优就是用更少的资源提供更好的服务，成本利益最大化。性能调优的手段并不新鲜，性能调优常规手段有：**

1. 空间换时间，内存、缓存就是典型的空间换时间的例子。利用内存缓存从磁盘上取出的数据，CPU请求数据直接从内存中获取，从而获取比从磁盘读取数据更高的效率。
2. 时间换空间，当空间成为瓶颈时，切分数据分批次处理，用更少的空间完成任务处理。上传大附件时经常用这种方式。
3. 分而治之，把任务切分，分开执行，也方便并行执行来提高效率。
4. 异步处理，业务链路上有任务时间消耗较长，可以拆分业务，减少阻塞影响。常见的异步处理机制有MQ（消息队列），目前在互联网应用中大量使用。
5. 并行，多个进程或者线程同时处理业务，缩短业务处理时间，比如我们在银行办理业务时，如果排队人数较多时，银行会加开柜台。
6. 离用户更近一点，比如CDN技术，把用户请求的静态资源放在离用户更近的地方。
7. 一切可扩展，业务模块化、服务化（同时无状态化）、良好的水平扩展能力。

**分布式架构的运用给性能带来了革命性的提升，业务流程的调整也会显著提升系统性能，单系统的调优能够压榨出更高的处理能力。单机性能分析调优可从从以下四部分入手：**

1. 性能分析方法
2. 基于单机的性能分析与调优
3. 基于业务流程优化的性能分析调优
4. 基于结构（分布式、业务拆分）的性能分析与调优

**性能分析方法**

性能分析是一个大课题，不同的架构、不同的应用场景、不同的程序语言分析的方法若有差异，抽象一下大致分为两类。

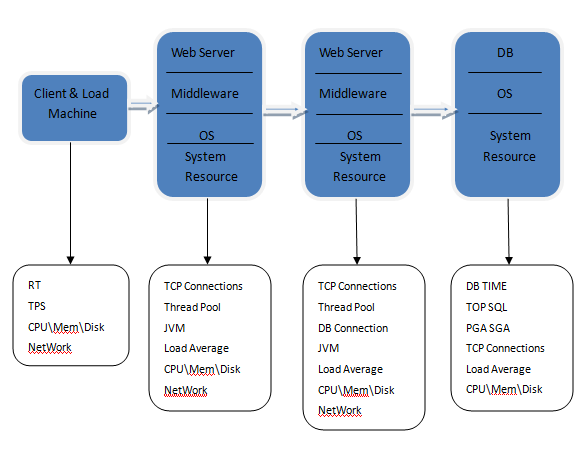
1. 自底向上：通过监控硬件及操作系统的指标（CPU、内存、磁盘、网络等硬件资源的性能指标）来分析性能问题（配置、程序等问题）。因为用户请求最终是由计算机硬件设备来完成的，做事的是CPU。
2. 自顶向下：通过生成负载来观察被测试的系统性能，比如响应时间、吞吐量；然后从请求的起点由外及里一层一层的分析，从而找到性能问题所在。

不管是自上而下还是自下而上，关键点就是生成负载、监控性能指标。好一点的方式是先用自顶向下的方式解决掉明显的性能问题，再结合自底向上的方式分析更深层次的问题。

**单机的性能分析与调优**

常见的J2EE应用架构，一般分为Web层（请求接入、负载均衡、页面渲染等）、应用层（业务逻辑实现）、持久化曾（数据记录）。

下面列出了性能测试时我们需要关注的指标。



Client：客户浏览器，比如IE、Chrome等访问Web页面。

Load Machine：是生成负载的机器，即我们的压测机器用来模拟用户负载。

Web Server：提供Web服务的服务器，即我们访问的Web页面由此服务器提供服务；一般都部署在Nginx、Apache等中间件上。

Middleware：中间件，比如Tomcat、Jboss、WebLogic等。

OS：操作系统，Windows或者Linux。

System Resource：系统资源，比如CPU、内存、磁盘、网络等。

App Server：应用服务，实现业务逻辑，比如生成订单，生成统计报表。

DB：数据库服务器，比如Oracle、Mysql、SqlServer等。

（1）

RT：响应时间，一笔业务的完成时间。

TPS：每秒完成的事物数。

CPU：CPU的性能指标，比如CPU利用率、CPU负载。

Mem：内存性能指标，比如可用物理内存、虚拟内存使用率。

Disks：Disk性能指标， 比如Disk Time、IO等待。

Network：网络指标，如带宽使用率，任务队列长度。

（2）

TCP Connections：指TCP连接数，可以用netstat命令统计得到。

Thread Pool：中间件建立的线程池，监控线程状态。

JVM：JVM性能指标，比如GC情况，Heap使用情况。

Load Average：CPU负载队列长度。

（3）

DB Connections：中间件与数据库之间建立的连接数及连接状态。

（4）

DB Time：消耗在数据库操作上的CPU时间。

TOP SQL：按内存占用由多到少排序SQL，按CPU占用由多到少排序SQL。

PGA、SGA：PGA、SGA内存使用情况。

性能分析过程：

|  |  |  |
| --- | --- | --- |
| 序号 | 步骤名称 | 说明 |
| 1 | 检查RT | 模拟用户发起负载后，采用的自顶向下的方式首先分析RT（响应时间） |
| 2 | 检查TPS | TPS大时RT小，说明性能良好 |
| 3 | 检查负载机资源 | 检查CPU使用率，CPU负载（Load Average）确认是用户CPU占用高还是系统CPU占用高 前提：确认测试脚本没有性能问题，不会造成结果统计的不准确 检查内存使用情况，确认并发内存泄漏风险，不会造成结果统计的不准确 |
|
|
| 4 | 判断负载机是否有性能问题 | 排除负载机的性能问题，确保测试结果可参考 |
|
| 5 | 检查Web服务器的资源消耗 | （1）检查CPU使用率，确认用户CPU与系统CPU占用情况 （2）检查内存使用情况 （3）检查磁盘使用情况 （4）检查占用的带宽 （5）分析Web页面响应的时间组成，确认是什么请求影响了性能 |
|
|
|
|
| 6 | 确认是否Web服务器瓶颈 | 标判断是否是Web服务器硬件性能瓶颈 |
|
| 7 | 检查中间件配置 | 确认是否是此配置问题 |
| 8 | 检查APP服务器资源消耗 | 关注CPU、内存、磁盘、IO，判断是否是App服务器硬件性能瓶颈 |
|
| 9 | 数据库服务器资源消耗分析 | （1）CPU消耗，CPU负载 （2）内存消耗 （3）IO繁忙程度 （4）数据库监控 |
|
|
|
|
| 10 | 是否是DB性能问题 | 由监控结果来判断是否是DB性能问题 |
| 11 | 是否SQL问题 | （1）定位最不合理的SQL占比 （2）索引是否正常引用 （3）检查共享SQL是否合理范围 （4）检查解析是否合理 （5）检查数据ER结构是否合理 （6）检查数据热点问题 （7）检查数据分布是否合理 （8）检查碎片整理等 |
|
|
|
|
|
|
|
| 12 | 其他 | 比如网络阻塞、磁盘IO瓶颈、热点等 |

上表列举了一种典型的分析思路，可以看到性能测试结果分析是一个考验综合知识的活动，涉及了多方面的知识，包括但不限于下面7部分：

1. 硬件知识（CPU、RAM、Disk、Net等）。
2. 系统知识（OS----Linux、Windows）。
3. 中间件知识（JVM、Tomcat、Jboss、WebLogic、WebSphere等）。
4. 数据库知识（Mysql、Sql Server、Oracle、DB2、Sysbase等）。
5. 网络知识（比如截包分析）。
6. 程序知识，比如Java程序，如何让程序更高效。
7. 架构知识，比如SSH架构。

大型系统的复杂度已经不是一个人力所能及的事情。上面提到的7个部分就可以是多个岗位（运维、程序员、架构师、DBA等），每个岗位又配置专业人员。性能分析时从他们那里获取性能指标数据，这些信息汇总后用来判断是否有性能问题。

对于性能测试工程师来说首先要做到的事情是要知道监控哪些指标？这些指标反应什么问题？什么时候去关注这些监控信息？在性能测试执行与分析时你就是总设计师，负责协调这些事项。

**程序优化**

程序调优是治本的手段，当前性能测试往往都是在SIT测试完成后进行的，性能问题暴露得太晚，这个时候去修改代码，风险较大。所以性能测试往往要提前规划，先架构后程序（先整体后个体）。

1. 系统框架选择

SSH架构是当下最流行的MVC模型。SSH架构提供了明晰的层次结构，各层协同完成业务实现，简化了程序设计过程，加快了程序交付进程。但是对大型的业务系统，特别是大数据量的分析计算过程，可以把数据处理换成在数据库中进行处理，减少网络传输，性能也会提升，所以应该不同的应用场景选择更合适的处理方式。

1. 程序优化

低效代码优化，排除架构问题，纯粹是程序逻辑及算法抵消，比如逻辑混乱、调用继承不合理、内存泄漏等。常见的解决方法如下：

1. 表单压缩，减少网络的传输量
2. 局部刷新，减少向服务器的请求
3. 仅取所需，只向服务器请求必要的内容
4. 逻辑清晰，方便维护、方便分析问题；不做错误及多余的调用，资源请求后能够释放
5. 谨慎继承，开发过程对系统架构熟悉，合理调用，减少大对象产生的可能
6. 程序算法优化，提高查询程序效率
7. 批处理，对于大数据最好做成分批处理
8. 延迟加载，对于大对象的展示可以采用延迟加载，比如分页，用到分页时再去请求
9. 防止内存泄漏
10. 减少大对象的引用
11. 防止争用死锁
12. 索引：编写合理的SQL，尽量利用索引
13. 内存分配，合理分配数据库内存，比如PGA与SGA的设置
14. 并行，使用多进程或进程来处理任务
15. 异步，比如用MQ来解耦系统之间的依赖关系，减少阻塞
16. 使用好的设计模式来优化程序，比如用回调来减少阻塞，使用监听器来阻塞依赖
17. 选择合适的IO模式，比如NIO、AIO等
18. 配置优化
19. JVM配置优化：合理的分配堆与非堆的内存，配置适合的内存回收算法，提高系统服务能力
20. 连接池：数据库连接池可以节省建立连接与关闭连接的资源消耗
21. 线程池：通过缓存线程的状态来减少新建线程与关闭线程的开销，一般是在中间件中进行配置，比如在Tomcat的server.xml文件中进行配置
22. 缓存机制：通过数据的缓存来减少磁盘的读写压力，缩小存储与CPU的效率差
23. 数据库连接池优化

数据库连接池存在的意义是让连接复用，通过建立一个数据库连接池（缓冲区）以及一套连接使用、分配、管理策略，使用的该连接池中的连接可以得到高效、安全的复用，避免了数据库连接频繁建立、关闭的开销。

连接池的主要关注的问题：

1. 连接池的配置参数。
2. 连接池配置多少连接合适
3. 监控连接池
4. 线程优化
5. 线程池优化，线程池是为了减少创建新线程和销毁线程的系统资源消耗
6. CPU处理能力
7. 内存容量
8. 系统线程数限制
9. DB优化

通常使用数据库有3个要求，性能好、数据一致性有保障、数据安全可靠；数据库优化前提也是这3个要求。

1. 优化物理结构，数据库逻辑设计与物理设计要科学高效，比如分区、索引建立、字段类型及长短、冗余设计等
2. 共享SQL、绑定变量、降低高水位
3. 查询器优化，特殊情况调整执行计划。指定的执行计划加快查找速度。比如连接查询时指定驱动表，减少表的扫描次数
4. 单条SQL优化，对单条SQL进行优化分析，比如查询条件选择索引列
5. 并行SQL，对数据量巨大的表的数据遍历，用多个线程分块处理任务。
6. 减少资源争用（锁、闩锁、缓存），可以提高IO效率减小响应时间从而提高吞吐量来缓解争用，比如用缓存；可以物理拆分把热点数据分布在不同表空间
7. 优化内存、减少物理IO访问
8. 优化IO，进行条带化、读写分离、减少热点等

注意：单系统性能分析的思路是通过现象结合监控锁定性能问题（程序、配置、IO等）

单系统性能调优的思路是减少资源占用，减少请求

**业务流程优化**

准确地说就是业务架构调整，业务架构是整个系统好坏成败的关键，对此处做调整就是推翻先前的设计，风险比较大。这点对于架构师的要求很明确。现实往往是残酷的，反过来想一下，正是因为这种矛盾的存在才导致了性能测试以及性能调优的存在。

**结构优化**

业务的增长导致性能问题推动着架构的发展，从单机到集群再到分布式结构。