**数据库连接池到底应该设置多大?**

我在研究HikariCP（一个数据库连接池）时无意间在HikariCP的Github wiki上看到了一篇文章（即前面给出的链接），这篇文章有力地消除了我一直以来的疑虑，看完之后感觉神清气爽。故在此做译文分享。

**接下来是正文**

数据库连接池的配置是开发者们常常搞出坑的地方，在配置数据库连接池时，有几个可以说是和直觉背道而驰的原则需要明确。

**1万并发用户访问**

想象你有一个网站，压力虽然还没到Facebook那个级别，但也有个1万上下的并发访问——也就是说差不多2万左右的TPS。那么这个网站的数据库连接池应该设置成多大呢？结果可能会让你惊讶，因为这个问题的正确问法是：

* “这个网站的数据库连接池应该设置成多**小**呢？”

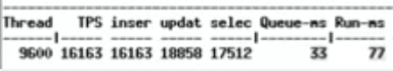
下面这个视频是Oracle Real World Performance Group发布的，请先看完：  
<http://www.dailymotion.com/video/x2s8uec>

（因为这视频是英文解说且没有字幕，我替大家做一下简单的概括：）  
视频中对Oracle数据库进行压力测试，9600并发线程进行数据库操作，每两次访问数据库的操作之间sleep 550ms，一开始设置的中间件线程池大小为2048：



初始的配置

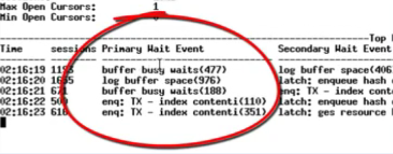
压测跑起来之后是这个样子的：



2048连接时的性能数据

每个请求要在连接池队列里等待33ms，获得连接后执行SQL需要77ms

此时数据库的等待事件是这个熊样的：



各种buffer busy waits

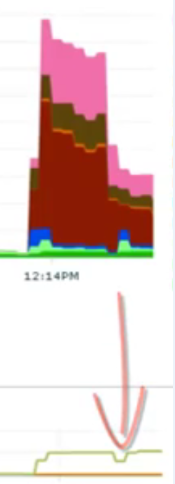
各种buffer busy waits，数据库CPU在95%左右（这张图里没截到CPU）

接下来，把中间件连接池减到1024（并发什么的都不变），性能数据变成了这样：

连接池降到1024后

连接池降到1024后

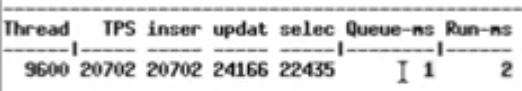
获取链接等待时长没怎么变，但是执行SQL的耗时减少了。  
下面这张图，上半部分是wait，下半部分是吞吐量



wait和吞吐量

能看到，中间件连接池从2048减半之后，吐吞量没变，但wait事件减少了一半。

接下来，把数据库连接池减到96，并发线程数仍然是9600不变。



96个连接时的性能数据

队列平均等待1ms，执行SQL平均耗时2ms。

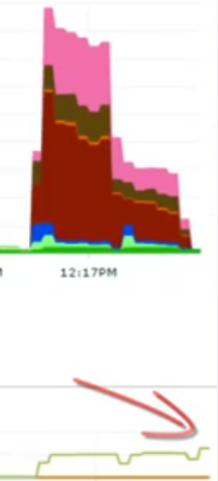


image.png

wait事件几乎没了，吞吐量上升。

**没有调整任何其他东西，仅仅只是缩小了中间件层的数据库连接池，就把请求响应时间从100ms左右缩短到了3ms。**

**But why?**

为什么nginx只用4个线程发挥出的性能就大大超越了100个进程的Apache HTTPD？回想一下计算机科学的基础知识，答案其实是很明显的。

即使是单核CPU的计算机也能“同时”运行数百个线程。但我们都[应该]知道这只不过是操作系统用时间分片玩的一个小把戏。一颗CPU核心同一时刻只能执行一个线程，然后操作系统切换上下文，核心开始执行另一个线程的代码，以此类推。给定一颗CPU核心，其顺序执行**A**和**B**永远比通过时间分片“同时”执行**A**和**B**要快，这是一条计算机科学的基本法则。一旦线程的数量超过了CPU核心的数量，再增加线程数系统就只会更慢，而不是更快。

这*几乎*就是真理了……

**有限的资源**

上面的说法只能说是接近真理，但还并没有这么简单，有一些其他的因素需要加入。当我们寻找数据库的性能瓶颈时，总是可以将其归为三类：*CPU、磁盘、网络*。把*内存*加进来也没有错，但比起*磁盘*和*网络*，内存的带宽要高出好几个数量级，所以就先不加了。

如果我们无视*磁盘*和*网络*，那么结论就非常简单。在一个8核的服务器上，设定连接/线程数为8能够提供最优的性能，再增加连接数就会因上下文切换的损耗导致性能下降。数据库通常把数据存储在磁盘上，磁盘又通常是由一些旋转着的金属碟片和一个装在步进马达上的读写头组成的。读/写头同一时刻只能出现在一个地方，然后它必须“寻址”到另外一个位置来执行另一次读写操作。所以就有了寻址的耗时，此外还有旋回耗时，读写头需要等待碟片上的目标数据“旋转到位”才能进行操作。使用缓存当然是能够提升性能的，但上述原理仍然成立。

在这一时间段（即"I/O等待"）内，线程是在“阻塞”着等待磁盘，此时操作系统可以将那个空闲的CPU核心用于服务其他线程。所以，由于线程总是在I/O上阻塞，我们可以让线程/连接数比CPU核心多一些，这样能够在同样的时间内完成更多的工作。

那么应该多多少呢？这要取决于*磁盘*。较新型的SSD不需要寻址，也没有旋转的碟片。可别想当然地认为“SSD速度更快，所以我们应该增加线程数”，恰恰相反，无需寻址和没有旋回耗时意味着**更少的阻塞**，所以更少的线程[更接近于CPU核心数]会发挥出更高的性能。**只有当阻塞创造了更多的执行机会时，更多的线程数才能发挥出更好的性能**。

*网络*和*磁盘*类似。通过以太网接口读写数据时也会形成阻塞，10G带宽会比1G带宽的阻塞少一些，1G带宽又会比100M带宽的阻塞少一些。不过网络通常是放在第三位考虑的，有些人会在性能计算中忽略它们。

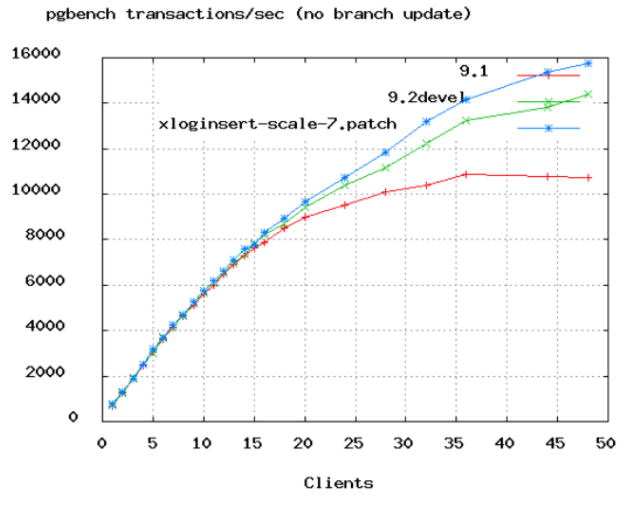


image.png

上图是PostgreSQL的benchmark数据，可以看到TPS增长率从50个连接数开始变缓。在上面Oracle的视频中，他们把连接数从2048降到了96，实际上96都太高了，除非服务器有16或32颗核心。

**计算公式**

下面的公式是由PostgreSQL提供的，不过我们认为可以广泛地应用于大多数数据库产品。你应该模拟预期的访问量，并从这一公式开始测试你的应用，寻找最合适的连接数值。

**连接数 = ((核心数 \* 2) + 有效磁盘数)**

核心数不应包含超线程(hyper thread)，即使打开了hyperthreading也是。如果活跃数据全部被缓存了，那么有效磁盘数是0，随着缓存命中率的下降，有效磁盘数逐渐趋近于实际的磁盘数。这一公式作用于SSD时的效果如何尚未有分析。

按这个公式，你的4核i7数据库服务器的连接池大小应该为((4 \* 2) + 1) = 9。取个整就算是是10吧。是不是觉得太小了？跑个性能测试试一下，我们保证它能轻松搞定3000用户以6000TPS的速率并发执行简单查询的场景。如果连接池大小超过10，你会看到响应时长开始增加，TPS开始下降。

笔者注：  
这一公式其实不仅适用于数据库连接池的计算，大部分涉及计算和I/O的程序，线程数的设置都可以参考这一公式。我之前在对一个使用Netty编写的消息收发服务进行压力测试时，最终测出的最佳线程数就刚好是CPU核心数的一倍。

**公理：你需要一个小连接池，和一个充满了等待连接的线程的队列**

如果你有10000个并发用户，设置一个10000的连接池基本等于失了智。1000仍然很恐怖。即是100也太多了。你需要一个10来个连接的小连接池，然后让剩下的业务线程都在队列里等待。连接池中的连接数量应该等于你的数据库能够有效同时进行的查询任务数（通常不会高于2\*CPU核心数）。

我们经常见到一些小规模的web应用，应付着大约十来个的并发用户，却使用着一个100连接数的连接池。这会对你的数据库造成极其不必要的负担。

**请注意**

连接池的大小最终与系统特性相关。

比如一个混合了长事务和短事务的系统，通常是任何连接池都难以进行调优的。最好的办法是创建两个连接池，一个服务于长事务，一个服务于短事务。

再例如一个系统执行一个任务队列，只允许一定数量的任务同时执行，此时并发任务数应该去适应连接池连接数，而不是反过来。