16 | 高性能和可伸缩架构：业务增长，能不能加台机器就搞定？

2020-03-27 王庆友

讲述：王庆友
时长 13:41 大小 12.54M

你好，我是王庆友，今天我来和你聊一聊如何打造高性能和可伸缩的系统。

在课程的第 11 讲，我和你介绍了，技术架构除了要保证系统的高可用，还要保证系统的高性能和可伸缩，并且能以低成本的方式落地。在实践中呢，高性能、可伸缩和低成本紧密相关，处理的手段也比较类似，这里我就放在一起来给你讲解。

在实际的工作当中，我们一般会比较关注业务功能的实现，而很少关注系统的性能，所以我们经常会面临以下这些挑战：

系统的 TPS 很低，只要流量一大，系统就挂，加机器也没用；

机器的资源利用率很低，造成资源严重浪费。

我曾经就统计过公司云服务器的资源利用率，结果让我非常意外，有相当比例的服务器，它们的 CPU 和内存平均利用率长期不到 1%，但与此同时，这些系统整体的 TPS 只有个位数。这里，你可以发现，资源利用率低和系统性能低的现象同时并存，很显然，系统没有充分利用硬件资源，它的性能有很大的优化空间。

所以今天，我就先来给你介绍一下常用的性能数据，让你建立起对性能的基本概念；然后，我会和你具体讲解实现系统高性能和可伸缩的策略，让你能在实践中灵活运用。

常用的性能数据

对于服务器来说，1ms 的时间其实不算短，它可以做很多事情，我在这里列了几个基础的性能数据，你可以把它们看做是系统性能的基线。

操作	耗时
SSD磁盘随机读取	16μs
机械硬盘随机读取	4ms
从内存顺序读取1MB数据	15μs
从SSD磁盘顺序读取1MB数据	200μs
从机械硬盘顺序读取1MB数据	2ms
网络包从加拿大到荷兰再到加拿大	150ms
网络包从深圳到上海	30ms
网络包在数据中心内部一个来回	50μs
CPU加锁	17ns
获取内存变量	100ns
内存写数据（4KB顺序写）	1600MB/s
硬盘写数据（4KB顺序写）	347MB/s
硬盘读数据（4KB顺序读，不带缓存）	45MB/s
文件打开和关闭	100W次/s
Socket打开和关闭	57W次/s
数据库访问（根据ID获取记录）	1~5ms

你可以看到，内存的数据读取是 SSD 磁盘的 10 倍，SSD 磁盘又是普通磁盘的 10 倍，一个远程调用的网络耗时是机房内部调用的 1000 倍，一个分布式缓存访问相对于数据库访问，性能也有数十倍的提升。

了解了这些常用的性能数据，你就能对性能建立一个直观的认识，有些时候，我们采取一些简单的手段就能提升系统的性能。比如说，如果磁盘的 IO 访问是瓶颈，我们只要用 SSD 磁盘来代替机械硬盘，就能够大幅度地提升系统的性能。

高性能的策略和手段

那么对于一个实际的业务系统来说，情况就会复杂很多。一个外部请求进来，需要经过内部很多的软硬件节点处理，用户请求的处理时间就等于所有节点的处理时间相加。只要某个节点性能有问题（比如数据库），或者某项资源不足（比如网络带宽），系统整体的 TPS 就上不去。这也是在实践中，很多系统 TPS 只有个位数的原因。

不同类型的节点，提升性能的方法是不一样的，概括起来，总体上可以分为三类。

加快单个请求处理

这个其实很好理解。简单来说，就是当一个外部请求进来，我们要让系统在最短的时间内完成请求的处理，这样在单位时间内，系统就可以处理更多的请求。具体的做法主要有两种：

1. **优化处理路径上每个节点的处理速度。**比如说，我们可以在代码中使用更好的算法和数据结构，来降低算法的时间和空间复杂度；可以通过索引，来优化数据库查询；也可以在高读写比的场景下，通过缓存来代替数据库访问等等。
2. **并行处理单个请求。**我们把一个请求分解为多个子请求，内部使用多个节点同时处理子请求，然后对结果进行合并。典型的例子就是 MapReduce 思想，这在大数据领域有很多实际的应用。

同时处理多个请求

当有多个外部请求进来时，系统同时使用多个节点来处理请求，每个节点分别来处理一个请求，从而提升系统单位时间内处理请求的数量。

比如说，我们可以部署多个 Web 应用实例，由负载均衡把外部请求转发到某个 Web 实例进行处理。这样，如果我们有 n 个 Web 实例，在单位时间里，系统就可以处理 n 倍数量的请求。

除此之外，在同一个节点内部，我们还可以利用多进程、多线程技术，同时处理多个请求。

请求处理异步化

系统处理请求不一定要实时同步，请求流量的高峰期时间往往很短，所以有些时候，我们可以延长系统的处理时间，只要在一个相对合理的时间内，系统能够处理完请求就可以了，这是一种异步化的处理方式。

典型的例子呢，就是通过消息系统对流量进行削峰，系统先把请求存起来，然后再在后台慢慢处理。

我们在处理核心业务时，把相对不核心的逻辑做异步化处理，也是这个思路。比如说下单时，系统实时进行扣库存、生成订单等操作，而非核心的下单送积分、下单成功发消息等操作，我们就可以做异步处理，这样就能够提升下单接口的性能。

那么，在实践中，我们应该使用哪种方式来保障系统的高性能呢？答案是，我们需要根据实际情况，把三种手段结合起来。

首先，我们要加快单个请求的处理。单节点性能提升是系统整体处理能力提升的基础，这也是我们作为技术人员的基本功。**但这里的问题是，节点的性能提升是有瓶颈的**，我们不能超越前面说的基础操作的性能。至于把请求分解为多个小请求进行并行处理，这个在很多情况下并不可行，我们知道，MapReduce 也有使用场景的限制。

对多个请求进行同时处理是应对海量请求的强有力手段，如果我们能够水平扩展每一个处理节点，这样在理论上，系统处理请求的能力无限的。**而这里的问题是**，对于无状态的计算节点，我们很容易扩展，比如说 Web 应用和服务；但对于有状态的存储节点，比如说数据库，要想水平扩展它的处理能力，我们往往要对系统做很大的改造。

至于异步化处理，在某些场景下是很好的提升系统性能的方式，我们不用增加机器，系统就能够完成请求的处理。**但问题是**，同步调用变成异步的方式，往往会导致处理结果不能实时返回，有时候会影响到用户体验，而且对程序的改造也会比较大。

所以，我们在考虑系统高性能保障的时候，首先需要考虑提升单个请求的处理速度，然后再考虑多请求的并发处理，最后通过异步化，为系统争取更长的处理时间。

具体的处理手段，根据业务场景的不同，我们需要做综合考虑，在满足业务的基础上，争取对系统改造小，总体成本低。

可伸缩的策略和手段

我们经常说，业务是可运营的，而实际上，系统也是可运营的。我们可以动态地调整系统软硬件部署，在业务高峰期增加软硬件节点，在业务低谷期减少软硬件节点，这就是系统的可伸缩能力。

系统的可伸缩也有两种实现方式。

第一个是节点级别的可伸缩

对于无状态的节点，我们直接增减节点就可以了。比如说订单服务，白天我们需要 10 台机器来提供服务，到了半夜，由于单量减少，我们就可以停掉部分机器。

如果做得好，我们还可以实现**弹性伸缩**，让系统根据硬件的负载情况，来确定机器的数量。比如说，当服务器的 CPU 或内存使用率在 10% 以下了，系统就自动减少服务实例的数量。

而对于有状态的服务，我们需要能够支持状态数据的重新分布。比如进行水平分库的时候，要从 4 个库增加到 8 个库，我们需要把原先 4 个库的数据，按照新的分库规则，重新分布到 8 个库中。如果这个调整对应用的影响小，那系统的可伸缩性就高。

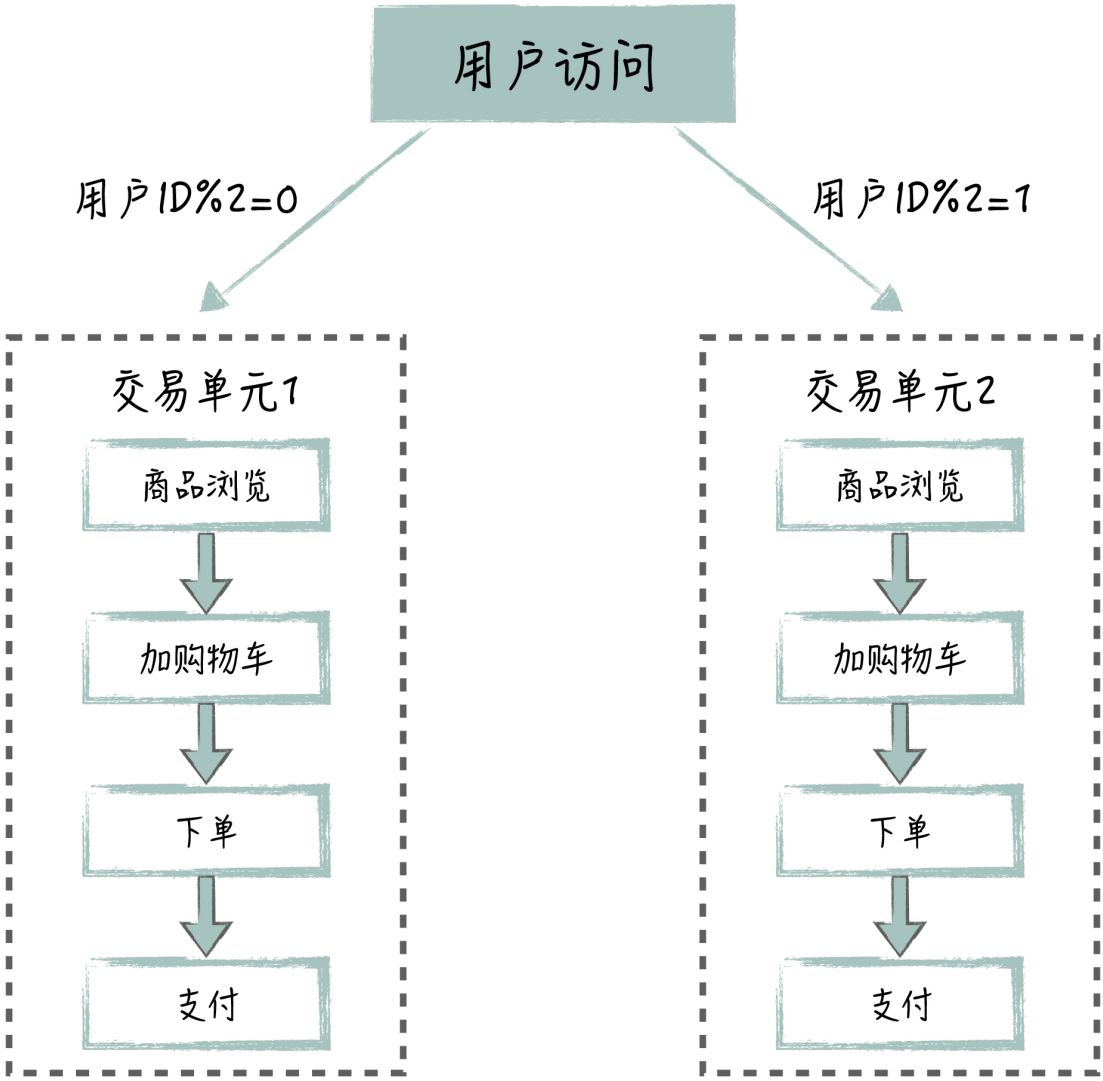
第二个是系统级别的可伸缩

我们知道，系统是一个整体，如果只是节点级别的伸缩，我们可能要对多个节点分别进行操作，而且不同节点的资源配置会相互影响，这样对各个节点的调整就非常复杂，影响了系统的可伸缩能力。**如果能实现系统端到端的伸缩，同时对多个节点进行伸缩处理，那系统的可伸缩能力就更高了。**

所以这里，我们可以把多个处理节点打包在一起，形成一个处理单元。

举个例子，针对交易场景，我们可以把商品浏览、加购物车、下单、支付这几个节点放一起，形成一个逻辑上的单元，在单元内部形成调用的闭环。具体使用的时候，我们可以按照用户维度，来划分交易单元。比如说，让交易单元 A 处理用户 ID 对 2 取模为 0 的用户下单流程，交易单元 B 处理用户 ID 对 2 取模为 1 的用户下单流程。

这样，我们对一个整体的交易单元进行扩容或者缩容，每增加一个交易单元，就意味着同时增加商品浏览、加购物车、下单、支付 4 个节点，这 4 个节点的处理能力是匹配的。你可以参考下面的这张交易单元化的示意图：



通过单元化处理，我们把相关的节点绑定在一起，同进同退，更容易实现系统的可伸缩。

而如果我们把单元扩大到系统的所有节点，这就是一个**虚拟机房**的概念。我们可以在一个物理机房部署多个虚拟机房，也可以在不同的物理机房部署多个虚拟机房，这样，部署系统就像部署一个应用一样，系统的可伸缩性自然就更好。

高性能和可伸缩架构原则

说完了高性能和可伸缩的策略，接下来，我再说下具体的架构设计原则，让你能够在实践中更好地落地。

可水平拆分和无状态

这意味着节点支持多实例部署，我们可以通过水平扩展，线性地提升节点的处理能力，保证良好的伸缩性以及低成本。

短事务和柔性事务

短事务意味着资源锁定的时间短，系统能够更好地支持并发处理；柔性事务意味着系统只需要保证状态的最终一致，这样我们就有更多的灵活手段来支持系统的高性能，比如说通过异步消息等等。

数据可缓存

缓存是系统性能优化的利器，如果数据能够缓存，我们就可以在内存里拿到数据，而不是通过磁盘 IO，这样可以大大减少数据库的压力，相对于数据库的成本，缓存的成本显然也更低。

计算可并行

如果计算可并行，我们就可以通过增加机器节点，加快单次请求的速度，提高性能。Hadoop 对大数据的处理就是一个很好的例子。

可异步处理

异步处理给系统的处理增加了弹性空间，我们可以利用更多的处理时间，来降低系统对资源的实时需求，在保证系统处理能力的同时，降低系统的成本。

虚拟化和容器化

虚拟化和容器化是指对基础资源进行了抽象，这意味着我们不需要再依赖具体的硬件，对节点的移植和扩容也就更加方便。同时，虚拟化和容器化对系统的资源切分得更细，也就说明对资源的利用率更高，系统的成本也就更低。举个例子，我们可以为单个 Docker 容器分配 0.1 个 CPU，当容器的处理能力不足时，我们可以给它分配更多的 CPU，或者增加 Docker 容器的数量，从而实现系统的弹性扩容。

实现了系统的高性能和可伸缩，就表明我们已经最大限度地利用了机器资源，那么低成本就是自然的结果了。

总结

在课程的开始，我给出了一些基础的性能指标，在具体的业务场景中，你就可以参考这些指标，来评估你当前系统的性能。

另外，我还分别针对系统的高性能和可伸缩目标，介绍了它们的实现策略和设计原则，在工作中，你可以根据具体的业务，由易到难，采取合适的手段来实现这些目标。

在实践中呢，实现高可用和可伸缩的手段也是多种多样的，接下来的课程中，我还会通过实际的案例，来具体说明实现这些目标的有效手段，帮助你更好地落地。

最后，给你留一道思考题：在工作中，你都采取过哪些手段保证了系统的高性能和可伸缩呢？

欢迎在留言区和我互动，我会第一时间给你反馈。如果觉得有收获，也欢迎你把这篇文章分享给你的朋友。感谢阅读，我们下期再见。

上一篇　15 | 高可用架构案例（三）：如何打造一体化的监控系统?

下一篇　17 | 高性能架构案例：如何设计一个秒杀系统?

精选留言 (5)

写留言

孙同学
2020-03-28

https://www.processon.com/view/link/5e51378ce4b0c037b5f9d1e3 学习整理更新，之前做过一个云端数据库和本地数据库做同步的业务，本地数据库要实时向云端传送数据，云端接口只有一个，本地调用很多，就将上传数据存储到redis中，返回上传成功结果，云端再另起脚本，用supervisor多进程消耗数据。

展开

tt
2020-03-27

之前的留言没写完不小心点击了发布，只好再写完，对不起编辑和老师了。

可水平拆分和无状态。

我们有一个内部管理系统，多个业务模块公用一套用户体系，用户的登录状态保存在sess...

展开

tt
2020-03-27

可水平拆分和无状态。对这一点，我最近可能做得有点问题。

情况是这样的，一个系统，对于用户的登录状态是保存在session中，多个业务系统公用一套用户体系

展开

特种流氓
2020-03-27

事务期间是否事务用到的资源都会被锁定

展开

作者回复: 有读锁和写锁，读锁的话大家都可以访问，写锁的话，大家对资源的修改就是互斥的。

zeor

2020-03-27

老师您好，请问怎么计算出一个系统要多少台机器，能抗住高峰期和平时，都有哪此指标和什么计算工公式?

作者回复: 没有特别好的计算方式，大致根据平时的性能进行估算，然后主要靠压测，在压测中优化。

1