二

21 高性能负载均衡：算法

21 高性能负载均衡：算法负载均衡算法数量较多，而且可以根据一些业务特性进行定制开发，抛开细节上的差异，根据算法期望达到的目的，大体上可以分为下面几类。

任务平分类：负载均衡系统将收到的任务平均分配给服务器进行处理，这里的“平均”可以是绝对数量的平均，也可以是比例或者权重上的平均。

负载均衡类：负载均衡系统根据服务器的负载来进行分配，这里的负载并不一定是通常意义上我们说的“CPU 负载”，而是系统当前的压力，可以用 CPU 负载来衡量，也可以用连接数、I/O 使用率、网卡吞吐量等来衡量系统的压力。

性能最优类：负载均衡系统根据服务器的响应时间来进行任务分配，优先将新任务分配给响应最快的服务器。

Hash 类：负载均衡系统根据任务中的某些关键信息进行 Hash 运算，将相同 Hash 值的请求分配到同一台服务器上。常见的有源地址 Hash、目标地址 Hash、session id hash、用户 ID Hash 等。

接下来我介绍一下负载均衡算法以及它们的优缺点。

轮询

负载均衡系统收到请求后，按照顺序轮流分配到服务器上。

轮询是最简单的一个策略，无须关注服务器本身的状态，例如：

某个服务器当前因为触发了程序 bug 进入了死循环导致 CPU 负载很高，负载均衡系统是不感知的，还是会继续将请求源源不断地发送给它。

集群中有新的机器是 32 核的，老的机器是 16 核的，负载均衡系统也是不关注的，新老机器分配的任务数是一样的。

需要注意的是负载均衡系统无须关注“服务器本身状态”，这里的关键词是“本身”。也就

是说，**只要服务器在运行，运行状态是不关注的。**但如果服务器直接宕机了，或者服务器和负载均衡系统断连了，这时负载均衡系统是能够感知的，也需要做出相应的处理。例如，将服务器从可分配服务器列表中删除，否则就会出现服务器都宕机了，任务还不断地分配给它，这明显是不合理的。

总而言之，“简单”是轮询算法的优点，也是它的缺点。

加权轮询

负载均衡系统根据服务器权重进行任务分配，这里的权重一般是根据硬件配置进行静态配置的，采用动态的方式计算会更加契合业务，但复杂度也会更高。

加权轮询是轮询的一种特殊形式，其主要目的就是为了**解决不同服务器处理能力有差异的问题。**例如，集群中有新的机器是 32 核的，老的机器是 16 核的，那么理论上我们可以假设新机器的处理能力是老机器的 2 倍，负载均衡系统就可以按照 2:1 的比例分配更多的任务给新机器，从而充分利用新机器的性能。

加权轮询解决了轮询算法中无法根据服务器的配置差异进行任务分配的问题，但同样存在无法根据服务器的状态差异进行任务分配的问题。

负载最低优先

负载均衡系统将任务分配给当前负载最低的服务器，这里的负载根据不同的任务类型和业务场景，可以用不同的指标来衡量。例如：

LVS 这种 4 层网络负载均衡设备，可以以“连接数”来判断服务器的状态，服务器连接数越大，表明服务器压力越大。

Nginx 这种 7 层网络负载系统，可以以“HTTP 请求数”来判断服务器状态（Nginx 内置的负载均衡算法不支持这种方式，需要进行扩展）。

如果我们自己开发负载均衡系统，可以根据业务特点来选择指标衡量系统压力。如果是 CPU 密集型，可以以“CPU 负载”来衡量系统压力；如果是 I/O 密集型，可以以“I/O 负载”来衡量系统压力。

负载最低优先的算法解决了轮询算法中无法感知服务器状态的问题，由此带来的代价是复杂度要增加很多。例如：

最少连接数优先的算法要求负载均衡系统统计每个服务器当前建立的连接，其应用场景

仅限于负载均衡接收的任何连接请求都会转发给服务器进行处理，否则如果负载均衡系统和服务器之间是固定的连接池方式，就不适合采取这种算法。例如，LVS 可以采取这种算法进行负载均衡，而一个通过连接池的方式连接 MySQL 集群的负载均衡系统就不适合采取这种算法进行负载均衡。

CPU 负载最低优先的算法要求负载均衡系统以某种方式收集每个服务器的 CPU 负载，而且要确定是以 1 分钟的负载为标准，还是以 15 分钟的负载为标准，不存在 1 分钟肯定比 15 分钟要好或者差。不同业务最优的时间间隔是不一样的，时间间隔太短容易造成频繁波动，时间间隔太长又可能造成峰值来临时响应缓慢。

负载最低优先算法基本上能够比较完美地解决轮询算法的缺点，因为采用这种算法后，负载均衡系统需要感知服务器当前的运行状态。当然，其代价是复杂度大幅上升。通俗来讲，轮询可能是 5 行代码就能实现的算法，而负载最低优先算法可能要 1000 行才能实现，甚至需要负载均衡系统和服务器都要开发代码。负载最低优先算法如果本身没有设计好，或者不适合业务的运行特点，算法本身就可能成为性能的瓶颈，或者引发很多莫名其妙的问题。所以负载最低优先算法虽然效果看起来很美好，但实际上真正应用的场景反而没有轮询（包括加权轮询）那么多。

性能最优类

负载最低优先类算法是站在服务器的角度来进行分配的，而性能最优优先类算法则是站在客户端的角度来进行分配的，优先将任务分配给处理速度最快的服务器，通过这种方式达到最快响应客户端的目的。

和负载最低优先类算法类似，性能最优优先类算法本质上也是感知了服务器的状态，只是通过响应时间这个外部标准来衡量服务器状态而已。因此性能最优优先类算法存在的问题和负载最低优先类算法类似，复杂度都很高，主要体现在：

负载均衡系统需要收集和分析每个服务器每个任务的响应时间，在大量任务处理的场景下，这种收集和统计本身也会消耗较多的性能。

为了减少这种统计上的消耗，可以采取采样的方式来统计，即不统计所有任务的响应时间，而是抽样统计部分任务的响应时间来估算整体任务的响应时间。采样统计虽然能够减少性能消耗，但使得复杂度进一步上升，因为要确定合适的**采样率**，采样率太低会导致结果不准确，采样率太高会导致性能消耗较大，找到合适的采样率也是一件复杂的事情。

无论是全部统计还是采样统计，都需要选择合适的**周期**：是 10 秒内性能最优，还是 1 分钟内性能最优，还是 5 分钟内性能最优……没有放之四海而皆准的周期，需要根据实际业务进行判断和选择，这也是一件比较复杂的事情，甚至出现系统上线后需要不断地调优才能达到最优设计。

Hash 类

负载均衡系统根据任务中的某些关键信息进行 Hash 运算，将相同 Hash 值的请求分配到同一台服务器上，这样做的目的主要是为了满足特定的业务需求。例如：

将来源于同一个源 IP 地址的任务分配给同一个服务器进行处理，适合于存在事务、会话的业务。例如，当我们通过浏览器登录网上银行时，会生成一个会话信息，这个会话是临时的，关闭浏览器后就失效。网上银行后台无须持久化会话信息，只需要在某台服务器上临时保存这个会话就可以了，但需要保证用户在会话存在期间，每次都能访问到同一个服务器，这种业务场景就可以用源地址 Hash 来实现。

将某个 ID 标识的业务分配到同一个服务器中进行处理，这里的 ID 一般是临时性数据的 ID（如 session id）。例如，上述的网上银行登录的例子，用 session id hash 同样可以实现同一个会话期间，用户每次都是访问到同一台服务器的目的。

小结

今天我为你讲了常见负载均衡算法的优缺点和应用场景，希望对你有所帮助。

这就是今天的全部内容，留一道思考题给你吧，微信抢红包的高并发架构，应该采取什么样的负载均衡算法？谈谈你的分析和理解。

上一页 下一页