21 | 高性能负载均衡：算法

2018-06-14 李运华

21 | 高性能负载均衡：算法
朗读人：黄洲君　09′03″ | 4.15M

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：

- 源地址 Hash

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。

- ID Hash

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。

欢迎你把答案写到留言区，和我一起讨论。相信经过深度思考的回答，也会让你对知识的理解更加深刻。（编辑乱入：精彩的留言有机会获得丰厚福利哦！）

精选留言

姜泮昌 16

微信抢红包架构应该至少包含两个负载均衡，一个是应用服务器的负载均衡，用于将任务请求分发到不同应用服务器，这里可以采用轮询或加速轮询的算法，因为这种速度快，适合抢红包的业务场景;另一起负载均衡是数据服务器的负载均衡，这里更适合根据红包ID进行hash负载均衡，将所有数据请求在同一台服务器上进行，防止多台服务器间的不同步问题。

2018-06-14

作者回复

基本正确，详细可以参考微信公开的红包高并发架构设计

2018-06-14

肖一林 9

看完了方乐明的对微信红包架构描述的技术文章，来回答一下问题：

有三个基本动作:发红包，抢红包，拆红包。发红包就是在数据库插一条红包库存数据，抢红包就是查询红包库存数据，拆红包就是插入一条秒杀流水并更新库存数据。

有三个难点:一是海量的并发，二是资金安全，三是良好的用户体验。资金安全决定了交易只能直接建立在数据库上，不能建立在缓存上。良好的用户体验就是要求不能出现不公平的现象，保证先抢先得，先抢的不能失败。

解决方案是:
1.分而治之，分成很多个并行的服务和数据库。相同的红包id总是分到相同的服务和数据库。所以负载均衡算法应该是hash算法
2.相同红包id的所有请求放在一个先进先出队列。然后分配一个独立的线程/进程处理。杜绝

抢锁。
3.分离过期数据，减少单表数据量

2018-06-14

作者回复

嗯，要根据具体业务来分析

2018-06-15

sunlight001 3

性能最优优先算法，抢发红包需要响应及时，该算法最优，但是复杂度比较好，需要不断调优。

2018-06-14

作者回复

没有结合业务的特点分析，详细可以参考微信公开的红包高并发架构设计

2018-06-14

Tom 2

看到评论说按红包id hash，上亿人抢同一个新年红包，应该只有一个红包id 吧?

2018-06-15

作者回复

你以为是一个红包，实际上是几千上万个红包😂

2018-06-15

赵正 Allen 2

发红包业务应该以红包的生命周期为思考的出发点。主要经历：发，抢，查询，回收（没抢完，返回给发红包的用户）。如果按这些细颗粒来看，抢，查询红包的并发要求最高，发红包次之，回收最低。
首先，发红包使用加权轮询，简单适用，成功后返回红包ID给客户端。
其次，抢红包，查询红包都带着ID给服务端，根据ID计算HASH，再利用一致性hash算法，找到最近的一个结点提供服务。
最后，回收应该由服务端定时触发，可以同样按抢红包处理。

2018-06-14

作者回复

分析到位，详细参考微信公开的技术文章，搜 微信红包高并发

2018-06-15

feifei 2

针对这个问题首先分析下抢红包的流程
1，首先一个用户在群组内发了一个红包，分成10份，并设置为随机大小。
2，假如群共有50人，都参与抢红包。
3，群里只能前10个用户可以抢到红包，并且金额随机
再来分析下业务流程
1，首先需要扣除发红包用户的红包金额。

2，按用户请求到达的先后顺序排队，进行抢红包，需要事务保证。
3，在队列中的用户抢到红包，其他用户没抢到红包。

然后再分析负载均衡的算法，主要针对抢红包的核心流程负载：
轮询，与加权轮训，都无法满足业务，此场景中存在事务，所以不合适
按机器负载，也无法满足，也是事务问题。
最后是hast，使用群id作为key进行分配，可以让同一个用户组的人负载到同一台机器，就可以完成事务。

所以我最终的负载均衡方案是选择hast算法。

我的分析是否合理？欢迎指正

2018-06-14

作者回复

红包分为发和抢，发的时候轮询就可以，抢的时候一般不是按照群id做hash，而是按照红包id做hash

2018-06-14

Leo Bright 2

一个红包对应多人去抢，服务器端要根据红包状态（是否抢完）来计算用户的结果（无法抢或抢了多少）。所以感觉最好是在同一台服务器上计算才能提高性能。那么红包和所有能够抢的用户之间应该有某个一致的关联信息（红包所在群的id），根据这个信息的负载算法负载到同一个服务器。所以我认为选hash算法合适，不知道想的对不对。

2018-06-14

作者回复

基本正确，详细可以参考微信公开的红包高并发架构设计

2018-06-14

Michael 1

老师好，这几天讲的负载均衡分类，算法，还有前面的reactor，proactor，这些东西我想再深细看一下，有没有什么相关的书籍可以推荐呢

2018-06-15

作者回复

没有专门书籍，或者说我没有看过类似书籍，都是我逐步积累的，网络编程基础看《unix网络编程 卷1》

2018-06-15

大光头 1

通过读你的这篇文章，我觉得到底采用哪种策略取决于业务，是否要求有session或者需要用户访问指定机器，如果是需要用hash，如果不是则考虑其它三种，轮巡和性能分配，采用哪

种策略取决于性能轮巡算法的复杂度和用户体验，如果复杂度高，往往采用简单轮巡更加，用户体验对响应时间跟你敏感，也需要采用性能分配。
微信抢红包，业务是有大量高并发请求，它不需要session，所以不用hash策论。由于用户体验是快速响应，所以应该采用性能响应时间策略

2018-06-14

作者回复

你前面的理解挺好，但是对于hash的理解不全面，session可以用hash，强一致性也需要用hash，单元化也需要用hash

2018-06-14

二戒 1

抢红包时可以根据红包标识hash，将同一红包路由到一台机器以减少后端逻辑复杂度，分布式一致性等难度。不过考虑企业客户发的大红包，会不会瞬间击垮一台服务器需要斟酌。红包发到哪台机器可以采取其他均衡策略。

2018-06-14

作者回复

回答正确，区分了发和抢的不同特点，详细可以参考微信公开的红包高并发架构设计

2018-06-14

孙振超 0

负载均衡算法一共十来种之多，但经过抽取共性归纳之后就变成了4类，不仅容易记忆，即使以后再新增加其他的算法，也不外乎是进行了丰富，种类并没有什么变化，归纳为4类之后遇到类似的问题也可以用这样的方式去予以解决，从中也可以看到高手不只是机械性的记忆，而是带着思考去看待问题。

具体到微信红包的算法选择上，由于并发量特别高，需要有一个简单高效的算法，因而性能优先类算法可以不做考虑。对于微信这种级别的机房，其容器化技术必然是炉火纯青，每一台vm的配置是可以完全相同的，因而也就无需采用负载均衡类算法和权重轮询算法，剩下来的就是hash类算法和简单轮询算法。对于红包业务，最主要的操作是发红包和抢红包：不管是发个人红包还是发群红包整体业务相差不大，可以采用简单轮询算法，到任何一台服务器均可。但抢个人红包和抢群红包是不同的，抢群红包是有先后顺序，当有多个人抢同一个群红包时最好是由同一个服务器进行处理，这台服务器在收到抢红包的请求后将请求放到一个队列中，而后按照先进先出的方式进行消费，可以保证先到的人能抢到红包，后到的人后抢到红包。因而对于抢红包业务最好按照红包id进行hash负载。
如果是只选择一个负载算法的话，就是hash负载，发红包按照userid进行hash负载，抢红包按照红包id进行hash负载

2018-07-22

作者回复

分析点基本到位了，详细可以参考微信公开的技术文档

2018-07-22

renwotao 0

根据作者的文章，我也注意到了业务有无状态也是选择均衡算法时要考虑的

2018-07-14

天真有邪 0

红包服务本身不存在状态，所以性能瓶颈在于如何分库，服务本身不存在性能瓶颈，按服务器好坏加权轮讯即可。由于涉及钱不考虑引入缓存增加复杂度。
分库：写入轮询即可，抢红包按照红包的主键查询即可。
1、发红包
2、抢红包

2018-07-09

作者回复

红包肯定存在状态的呀😄

2018-07-11

summer 0

微信抢红包，最好所有人都在红包所在server进行抢劫。适合ID hash这种算法

2018-06-28

作者回复

抢红包，别抢劫😂

2018-06-28

fiseasky 0

什么hash算法，轮循加权完全看不懂啊，是不是数据结构的知识，数据结构和算法全部还给老师了。

2018-06-17

作者回复

hash都不懂哇，赶紧去学😭

2018-06-18

王维 0

华仔你好，我觉得对于微信红包这种高并发的应用场景，应该采用的负载均衡算法是hash值的方式，因为
这种场景对事务的要求高，所以要保持事务的一致性不知道是不是这样，盼指教。

2018-06-16

作者回复

单点操作保证一致性，详细搜索微信红包高并发的文章

2018-06-19

张玮(大圣) 0

1. 发红包：查询类型，可采用轮询、负载最低、性能优先等

2 抢红包：更新类型，涉及到红包的分配、扣减金额到用户账户等事务性操作，如果要在负载均衡层解决，那就使用 hash 路由，如果不在负载均衡层解决，可以用分布式事务的几种解决方案

2018-06-15

作者回复

分布式事务性能扛不住，你可以搜索微信红包高并发的设计文章

2018-06-19

Tom 0

微信抢红包的特点是瞬间负载变化很大，如果用负载最低优先算法，统计状态可能要秒级别，感觉不适合。应该还是轮询算法吧，不过bug导致某台单机负载问题不知道怎么解决，靠监控系统发现然后加权呢还是其他？

2018-06-15

作者回复

详细参考微信公开的技术文章，搜 微信红包高并发

2018-06-15

星火燎原 0

抢红包肯定是选择最近这段时间窗口评论响应时间最小的服务器节点算法啦，不过腾讯这种土豪公司动态扩容节点小意思啦，也有可能简单粗暴采取随机算法呢🤔

2018-06-14

作者回复

没有结合业务的特点分析，详细可以参考微信公开的红包高并发架构设计

2018-06-14

xzyeah 0

对于微信红包场景，我觉得更加适合加权轮询类
主要原因
1，场景单一，会话无关
2，高并发，对性能要求高，加权轮询算法简单高效
3，高并发场景计算性能与负载已经没有意义，保证所有服务能均衡分配更好

以上是我的理解，如有不对请指正，谢谢！

2018-06-14

作者回复

虽然与会话无关，但是红包的一个典型特征就是红包处理逻辑是强一致性的，一个红包的处理只能在一台服务器上，如果多台服务器并发处理一个红包，锁设计会很复杂而且更低效，所以抢红包的场景和会话类似，需要用hash

发红包可以简单轮询

2018-06-14