21 | 如何进行高保真压测和服务扩容?

在后台架构中，压测非常常见，也是必须的工作。它能够帮我们发现微服务架构中的性能瓶颈，以及知道构建的微服务能承载的流量极限值。

但实际情况是，很多压测并不能发现瓶颈点和微服务所能承载的真实流量极限值。一方面是因为压测时使用的是人为构造的压测参数；另一方面有些时候压测场景是经过修改的，和实际的线上场景存在差异。

本讲将会聊聊如何构建一个更贴近真实场景的高保真压测，以及如何根据压测结果进行相对应的容量规划。

实施高保真压测

模拟参数进行压测是指人为构建符合被压测接口的一个或一组参数进行压测的方法，它存在以下几个问题。

首先，参数是模拟的，可能和线上真实环境有差异，进而导致压测数据失真。

假设我们要压测使用了本地缓存和Redis 来实现的查询用户基本信息接口，当使用一个或一组用户账号压测此接口时，压测出来的性能和 QPS 会非常好。因为，在压测的前几次调用之后，所有用户信息已经缓存到用户模块的本地缓存里，后续的所有压测请求都可以直接使用本地缓存的数据，因此性能会非常好。

而实际生产环境里，查询用户信息的接口请求里的用户账号并不是相同的，因此请求不会都命中本地缓存，所以它的性能要比压测时低。这就是产生失真的原因。

其次，即使抓取线上环境的一组参数进行压测，也不能完全代替真实环境，仍然存在失真的场景。

而高保真压测，从字面上就可以理解它的要求——使用和生产环境一模一样的用户请求进行压测。这样压测出的微服务的各项性能指标更加可信，因此可以作为限流和容量评估的参考标准。

那如何模拟线上请求呢？只要完成生产环境的流量录制，并把它用来压测即可。其实本讲介绍的高保真压测实现思路和“第 07 讲”基本类似，只是“第 07 讲”把录制的流量用来进行自动化测试回归。

基于生产环境的流量录制压测架构如下图 1所示：

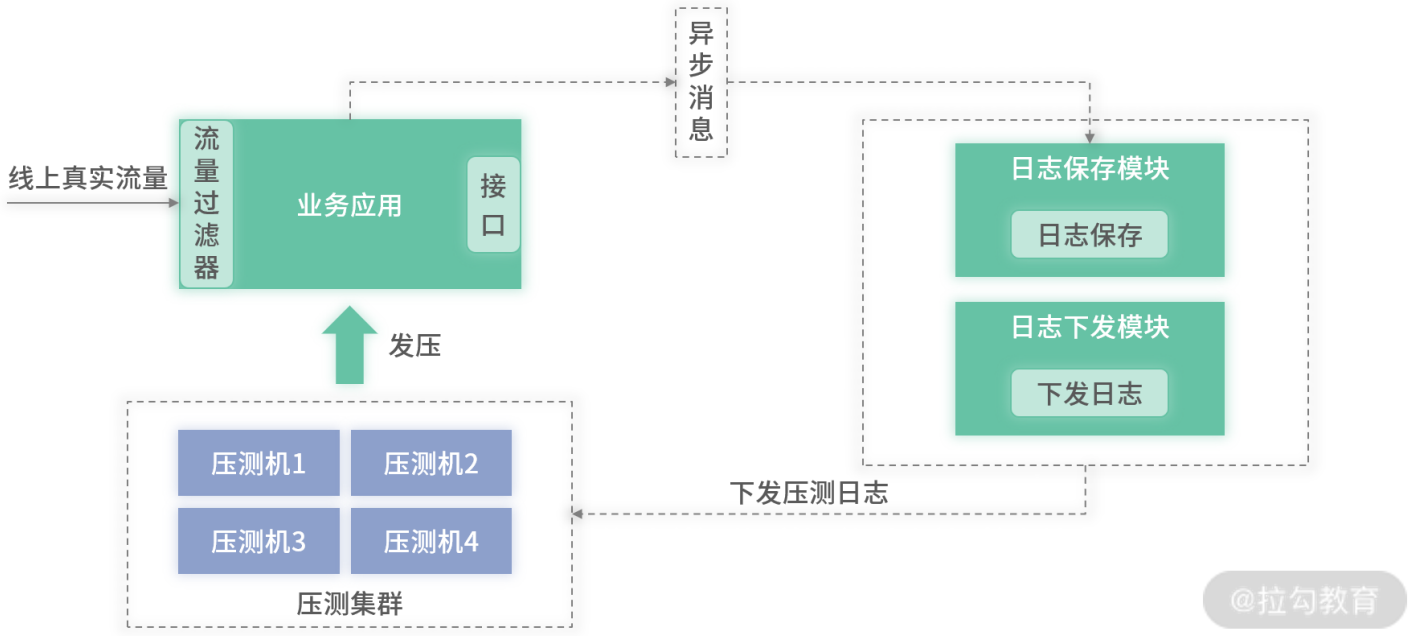

图 1：基于流量录制的压测架构图

上述架构和“第 07 讲”的自动化回归架构类似，其中包含内置于业务进程里的流量过滤器、日志保存模块、日志下发模块、压测模块以及压测管理端。

1. **流量过滤器**：和“第 07 讲”里的类似，采用 RPC 框架提供的拦截器实现，并将出入参基于 MQ 转发。
2. **日志保存模块**：根据压测设置的配置，进行压测日志的收集。压测的配置包含：当次需要压测的接口和方法、收集日志的时长和数量等。需要注意的是，相比自动化回归，此处日志收集的量应该要多很多，因为通常来说，压测需要的真实用户请求数据会更多一些。因此，日志的存储也可以升级到分布式文件系统里，如 Hadoop。
3. **日志下发模块**：主要功能是将压测的日志下发到压测机器里。为什么不是压测模块远程连接到分布式文件系统读取日志后，再进行压测呢？主要是因为考虑性能。压测希望在短时间内，给被压测应用一个洪峰流量。如果压测模块在发起压测请求前还需要调用其他远程接口获取数据，很大程度上，就实现不了这个洪峰流量了。因此，需要在压测前，将压测日志推送到压测机器上，压测模块可以读取本地磁盘的日志，性能将会有极大提升，可以短时间发起洪峰流量。
4. **压测模块**：主要的功能是读取本地日志并调用被压测机器，并将压测信息写入存储中。
5. **压测管理端**：用来设置各项压测配置以及查看压测结果值。

上述便是一个高保真压测的架构和对应模块的功能。在了解压测模块的架构后，我们再来看看压测时，被测应用需要注意的问题。

首先，最简单的被压测应用架构如下图 2 所示：

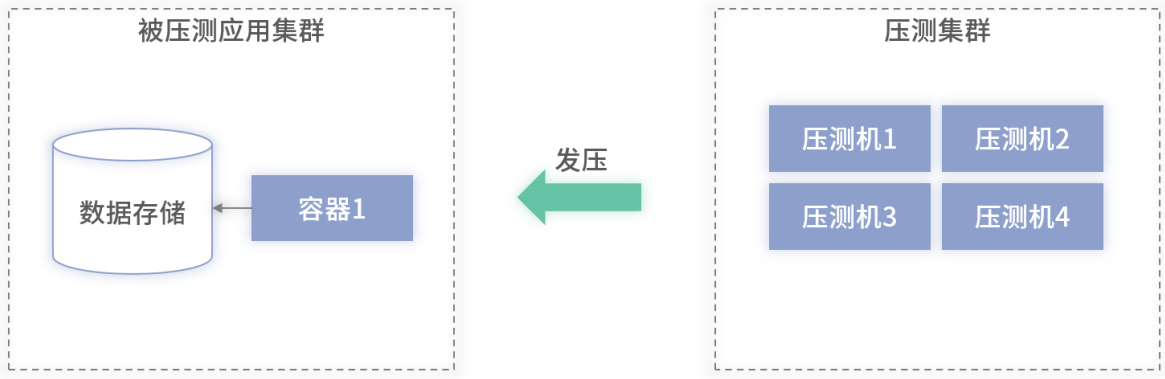

图 2：简单的被压测应用架构图

上述架构是一个非常简单的应用部署图，其中包含一个存储模块（Redis 或数据库）及一台应用机器。压测时，很大概率是应用程序所在的宿主机先达到资源的瓶颈，而不是数据存储先达到瓶颈。这可能是发生了宿主机的CPU 利用率达到 100%，或者是内存使用率满了等情况。此时，获取到的QPS 即为上述架构里单台机器能够承载的最大值。

那么，是不是拿着上述单机的最大 QPS，通过机器数量乘以单机最大 QPS，即可计算出当前线上集群能够承载的最大 QPS 呢?

答案显然是不行的，集群的QPS 并不是随着机器数量增加而线性增加的。主要原因是所有机器所处的网络是共享的、进程间的切换存在性能消耗，以及存储是共享的等因素。

在实际压测中，压测完单台机器后，可以分多次部署 2台，4 台及 8 台应用进行压测，得到对应的压测 QPS。通过不断叠加机器进行压测获得损耗比，为后续线上大规模扩容做数据准备。假如得到 1 台机器的压测 QPS 为 100，2 台的QPS为180，4 台的QPS 为 360，那么可以计为损耗比 10%。假设线上有100 台机器，由此评估线上可以支撑的 QPS为9000。

这里你可能会有疑惑，为什么不直接对线上集群进行压测，而要采用这种按比例的方式？主要有以下两个原因。

1. 线上机器并不是一成不变的。比如现在线上有100 台机器，在业务不断发展后，可能会不断扩容至 200 台或更多的机器。此时，之前直接按 100 台机器进行压测的数据就没法直接使用了。

2. 直接对线上环境进行压测会影响线上正常的业务。因为压测的应用集群和存储都是和线上共享的。如果不希望它们互相影响，那么就需要重新部署一套和线上一模一样的环境，这个资源的消耗和部署的成本会非常高。

那是不是通过上述方式压测之后，就可以根据压测值和损耗比进行压测了呢？比如，某一天运营计划做促销活动，预估带来的最大流量 QPS 为 18000/s。按上述数据，是不是扩容到 200 台机器即可了呢？

其实，并不是。此种扩容评估法做了一个假设，即服务所能够承载的 QPS 是和机器绝对线性增长的，只要机器充足，那么能够承载的 QPS 是没有上限的。但**服务依赖的存储是有上限的，微服务能够提供的极限 QPS，其实是由它本身和它依赖的存储的最小值共同决定的**。

以上述案例的压测值和损耗比为例，架构如下图 3 所示：

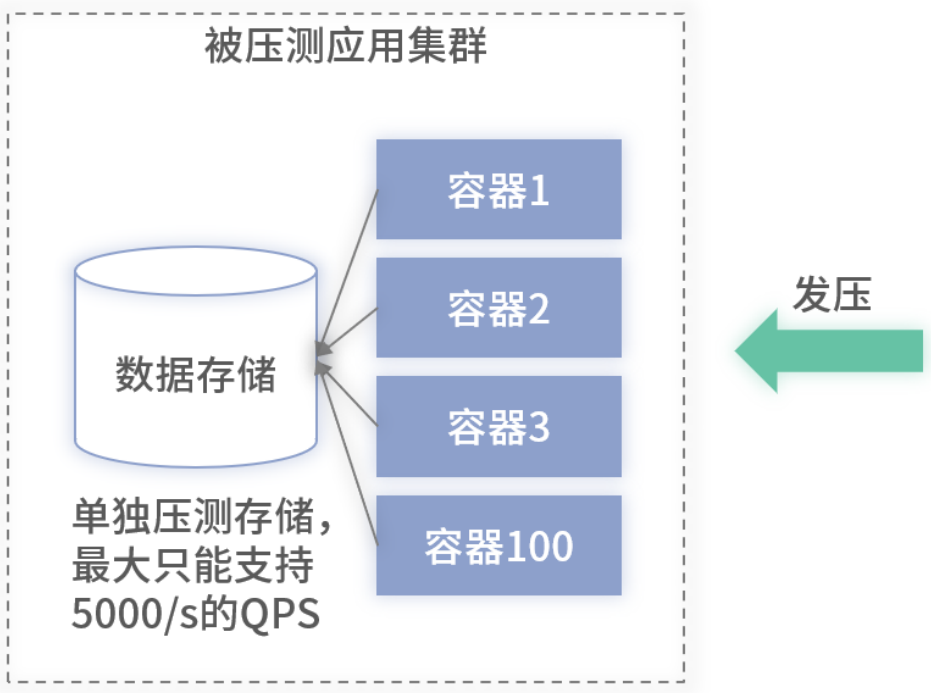

图 3：带存储的架构图

当前图示中部署了 100 台机器，理论上可以支撑 9000/s的QPS，但如果所依赖的存储只能支撑 5000/s的QPS，那么即使部署 100 台或者更多的应用机器，它能够承载的QPS 也不能线性增长，最大只能支撑到 5000/s的QPS。

因此，在实际压测中，除了寻找单机压测值和损耗比之外，还需要对微服务依赖的存储，以及除存储之外其依赖的其他微服务进行压测，寻找微服务压测中的最短板，进而确定微服务能够支撑的最大 QPS。

如何做写压测

读服务是无状态的，所以可以直接采用基于录制的压测方案进行压测。但是写服务是有状态的，因为录制的流量是用户在线上产生的真实请求，比如下单请求，如果直接使用录制的流量进行回放，可能会给客户的账号误下一笔订单，在生产环境中是不允许的。如果出现这样的线上操作，就算是线上事故了。

其实，保障相对高保真的写压测有以下两个常见的方式进行应对。

第一种是采用模拟账号进行替换或数据修改进行压测。

假设压测使用的数据是用户私有数据，比如压测发送微博的接口或提交订单的接口，就可以将录制数据里的用户账号替换为测试账号，这样压测时产生的微博和订单都隶属于测试账号，就不会对线上产生影响了。

此外，如果压测使用的数据是公有数据，比如新闻投稿接口，只要新闻投稿了所有用户都可见。对于这种公有数据的接口，可以在业务上进行处理，比如修改录制的新闻投稿数据，将所有的投稿都修改为待审核。这样压测产生的数据都处于待审核状态，在线上是不可见的，所以此种压测对线上不会产生影响。

第**二种方式是采用压测数据打标 + 影子库的方式进行特殊处理**，架构如下图 4 所示：

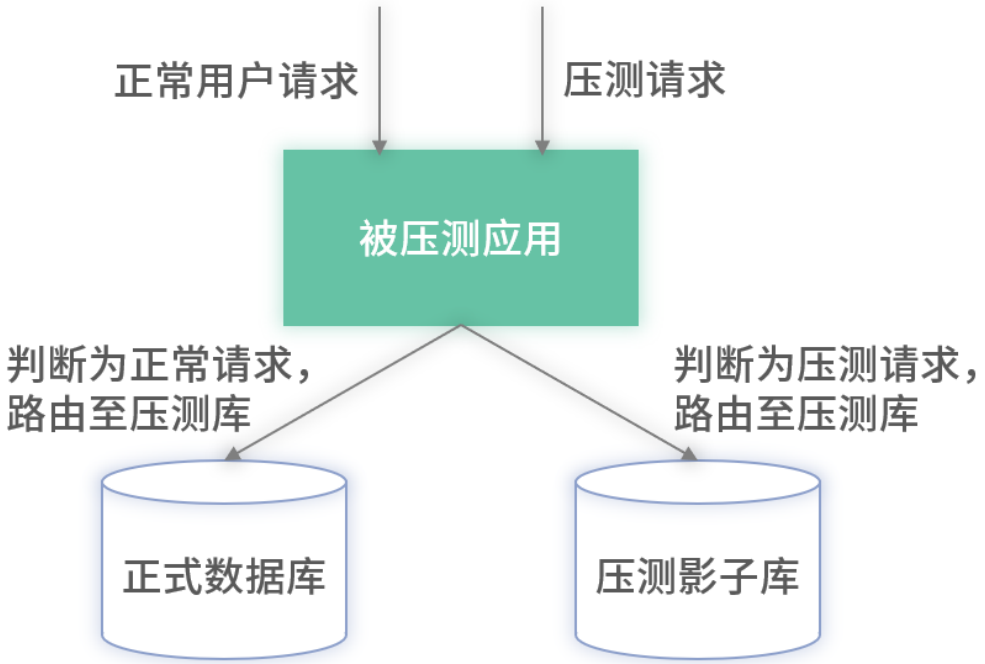

图 4：数据打标+影子库压测架构

上述架构里的数据打标和第一种里的数据修改是有区别的，它不会更改原始录制的任何数据，只是在压测的时候，对于压测模块发起的任何请求都增加一个标记，标记它为压测请求。

在业务应用模块识别此标识，如果识别出压测请求，则将压测请求的数据全部写入上图 4 中的影子库中。影子库中的数据不会暴露给外部查询以及用于进一步的生产，因此不会产生线上影响。对于微服务依赖的其他微服务提供的写接口，可以在压测时继续传递标识，被依赖的微服务也识别此标识，将压测数据写入影子库即可。通过标识传递+影子库的方式，即构建了一个线上写压测环境。

基于压测数据进行行动

压测过程中有两方面重要的数据，一个是压测过程中的各项指标数据，另一个是压测的结果即服务所能够支撑的QPS。

压测过程的各项指标数据有：压测时机器的CPU 利用率的变化、内存的变化、进程里各线程的CPU 利用率、微服务依赖的存储的CPU利用率、内存使用率等。压测过程中监控这些数据是为了发现系统瓶颈点，并快速优化，进而提升微服务能够支撑的QPS。这里简单列举一些可能存在的瓶颈点。

1. 如果压测过程中，发现被压测应用的CPU 都被某一个或某一类线程消耗，同时通过堆栈信息，确定这个或这类线程的所有 CPU 消耗都集中在一个方法里。那么极大可能，这个方法里有十分消耗 CPU 的代码，可能是一个大对象的JSON 序列化或者是一段可以优化的多层嵌套 for 循环。
2. 再比如，在只部署了一台应用机器和对应存储（MySQL）的情况下。理论上压测时，应该是单台应用机器的CPU 先达到 100%。但如果在实际压测中，是 MySQL 所在机器的CPU 先打满，那么很大概率上是被压测接口请求数据库的 SQL 是一个慢

SQL。引发这种情况的原因可能是未命中索引、一次请求的数量太多、存在 SQL 的深翻页等。此时，就需要对这些 SQL 进行调优，以便进一步提升微服务的性能。

压测的极限 QPS 除了让我们了解了微服务的最大支撑能力之外，另外一个作用就是参考此值来设置微服务的限流阈值。流量达到压测时的QPS 时，微服务的各项指标如 CPU、内存等，均已达到极限，为了保证微服务的稳定，需要将进入微服务的流量限制在压测的 QPS 之下。根据压测值设置限流时，有以下几点需要注意。

1. 上述案例中，单机压测的 QPS为100/s。但限流时，不能直接设置单机限流阈值为 100/s，因为达到此QPS 时，机器的CPU 已经达到 100% 了。正常情况下，线上机器的 CPU利用率维持在 40%~50% 是安全的，再升高就需要扩容了。因此限流时，可以将压测的 QPS 适当打折，设置压测为 QPS*40%。
2. 如果微服务提供不止一个接口，那么上述的限流阈值就还需要打折。比如微服务对外提供了两个接口，那么最简单的打折办法为：单机 QPS*40%*50%。
3. 在前述的讲解中曾提到过，微服务能够支撑的 QPS是有上限的，并不是随着机器数量无限增长的。因此，除了设置单机级别的限流之外，还需要设置微服务集群维度的限流阈值。限流阈值的设置方法可以参考上述第 1、2 点。

总结

在这一讲里，讲解了如何构建高保真的压测环境。同时，针对写服务，详细介绍了如何通过优化升级来解决“写服务有状态”这一问题。

此外，压测过程中的数据和压测结果不只是用来记录，还可以用于分析，寻找可以优化的瓶颈点。其次，需要根据压测极限值，设置微服务的限流阈值，防止流量超过压测极限值，进而将机器打挂，导致服务完全不可用。

最后，再给你留一道讨论题，你当前所在团队的压测是如何开展的？欢迎留言区留言，我们一起讨论。

这一讲就到这里，感谢你学习本次课程，接下来我们将学习22 | 重构：系统升级，如何实现不停服的数据迁移和用户切量？