京东的压测军演系统

转载

weixin_34077371 于 2017-06-21 17:19:00 发布 118 收藏 1
文章标签： git python 运维
原文链接：https://my.oschina.net/cctester/blog/994735

准备电商大促就要准备好应对高流量，全链路压测无疑是必不可少的一个环节，但是同时也涉及到很繁重繁琐的工作。京东研发设计了军演机器人，这为今年备战 618 减负不少。最初，军演机器人 ForceBot 正式立项并组建了一个虚拟的研发团队，彼时计划是基于开源的 nGrinder 项目进行二次开发，随后实现部署即可；随后在深入研发之后，又根据京东的业务场景对 nGrinder 进行了优化，以满足功能需求。

传统的压测方案

传统的系统压测基本都是部署在内网环境，和被压测的系统部署在一个局域网内，比较常用的工具有：loadrunner、jmeter、nGrinder、gatling、iperf 等等，通过这些工具，模拟生产环境中的真实业务操作，对系统进行压力负载测试，同时监控被压测的系统负载、性能指标等不同压力情况下的表现，并找出潜在的性能优化点和瓶颈，目前流行的压测工具，工作原理基本都是一致的，在压力机端通过多线程或者多进程模拟虚拟用户数并发请求，对服务端进行施压。以往在京东，备战 618 大促要提前 3 个月准备，需要建立独立系统进行线上的压力评测，这为各个性能压测团队带来了很大的工作量。

除了工作量大的问题之外，传统的压测数据与线上对比，并不准确。以前对某个系统性能的压测需要在内网线下进行多次，压测出的结果各项指标与线上相比差异太大；这是因为线下服务器配置和上下游服务质量不可能与线上一模一样，只能作为一个参考，不能视作线上系统的真实表现。压测后需要思考如何进行各系统容量规划，每次大促前备战会进行基础服务资源的分配，如果不能精确预估容量需求，会发生扩容行为的浪费。

为了更贴近线上真实结果，有些系统也会通过直接内网压测线上系统，这样做需要上下游同步协调，费事费力，这样下来至少一周时间才能搞定。使用了 ForceBot 全链路军演压测系统后，目前只需要 2 天左右就可以完成所有黄金链路系统的性能评测。

变革，制造一个军演机器人

最开始的时候京东并没有直接投入研发力量，京东 618 年中全民购物节技术执行总指挥刘海锋首先提出了 ForceBot 这个想法，京东内部则面向各研发和性能团队进行大量的调研，了解他们现在的痛点和使用的工具情况，同时谈及了 ForceBot 的想法，这个想法获得了大家的赞同并且收集了很多宝贵意见和建议。于是 ForceBot 正式立项并组建了一个虚拟的研发团队，最开始计划是基于开源的 nGrinder 项目进行二次开发，随后实现部署即可。

nGrinder 基于开源的 Java 负载测试框架 grinder 实现，并对其测试引擎做了功能提升，支持 python 脚本和 groovy 脚本；同时提供了易用的控制台功能，包括脚本管理、测试计划和压测结果的历史记录、定时执行、递增加压等功能。

根据京东的业务场景对 nGrinder 进行了优化，以满足我们的功能需求。比如：提升 Agent 压力，优化 Controller 集群模式，持久化层的改造，管理页面交互提升等。

nGrinder 能胜任单业务压测，但很难胜任全链路军演压测。分析其原因是 Controller 功能耦合过重，能管理的 Agent 数目有限。原因如下：

Controller 与 Agent 通讯是 BIO 模式, 数据传输速度不会很快；

Controller 是单点，任务下发和压测结果上报都经过 Controller，当 Agent 数量很大时，Controller 就成为瓶颈了。

也就是说问题出现在：**Controller 干的活又多又慢，整体压力提升不上去。**

尽管我们优化了 Controller 集群模式，可以同时完成多种测试场景。但是，集群之间没有协作，每个 Controller 只能单独完成一个测试场景。即 nGrinder 整体构架无法满足设想的军演规模和场景，也算是走了一些小弯路，最终在其基础上开始新的架构设计和规划，规避已知瓶颈点，着手研发全链路军演压测系统（ForceBot），为未来做好长远打算。

ForceBot

ForceBot 平台在原有功能的基础上，进行了功能模块的解耦，铲除系统瓶颈，便于支持横向扩展。

对 Controller 功能进行了拆解，职责变为单一的任务分配；

由 Task Service 负责任务下发，支持横向扩展；

由 Agent 注册心跳、拉取任务、执行任务；

由 Monitor Service 接受并转发压测数据给 Kafka；

由 Dataflow 对压测数据做流式计算，将计算结果持久化至 DB 中；

由 Git 来保存压测脚本和类库。GIT 支持分布式，增量更新和压缩

这样极大的减轻了 Controller 的负载压力，并且提升了压测数据的计算能力，还可以获取更多维度的性能指标。

在此基础上，还融合进来了集合点测试场景、参数下发、TPS 性能指标等新特性。

持续改进

ForceBot 平台在上线提供服务后，在受到好评的同时也发现了一些问题。所以我们对架构和功能实现进行了调整。对问题进行了合理化解决，重新设计了架构中的部分功能实现，并对依托 nGrinder 和 Grinder 的功能，针对京东的使用习惯和场景，进行了自研，至此，ForceBot 平台由基于 nGrinder 基础上的深度改造升级为完全自研的性能测试平台。

Git 作为先进的版本控制系统，用来构建测试脚本工程的资源库再合适不过，但是直接使用它作为资源分发服务就不太合适了。Git 的每次操作如 Clone、Pull 等都是一组交互，传输效率不高。同时一代平台使用 GitLab 构建的 Git 服务集群依赖于一个集中的 NFS 网络共享存储，这就带来性能瓶颈和单点故障的可能。

架构调整中针对 ForceBot 平台的资源分发方式进行了重新设计，借助京东基础平台自研的京东分布式文件系统（JFS）的云存储服务（JSS）进行资源的分发。

新的架构调整中，增加了 Script Package Service 为性能测试脚本提供统一的构建打包支持，并通过对 Maven 的支持，与公司内网 Maven 私有服务交互，为脚本提供更灵活可靠的依赖管理。Script Package Service 将脚本及其依赖类库、配置、数据进行打包，处理成一个统一的 Gzip 压缩文件，并上传至 JSS 以向 Agent 进行分发和归档。

平台在使用过程中，由于调试环境和实际运行环境有所差别，用户有查看 Agent 执行日志的需求。考虑到日志落地带来的磁盘 IO 对性能性能的影响，以及日志内容给平台管理带来的开销，新的架构中提升了日志的收集和处理功能。Agent 的日志不再落地本地磁盘，改为写入到内存一块固定大小的区域，最后经处理切割成一条一条的日志实体并结构化的存入 Elasticsearch 中供用户查询。

技术细节

1. 核心功能

平台核心功能在于需要向性能测试人员提供一个高效的可操作环境用于准确的描述其测试逻辑，并兼容大部门公司业务服务调用方式和场景。京东内部业务系统大部分使用 Java 语言开发，使用基础平台中间件技术部开发的 JSF、JMQ 等中间件进行服务调用，为此提供一个与 Java 语言高度兼容的脚本语言执行环境作为性能测试逻辑编写基础尤为关键。

系统选用了兼容 JSR223 规范的 Groovy 作为主要脚本语言，并效仿 Java 下著名的单元测试框架 Junit 的设计哲学设计了一套高效并友好的测试逻辑开发和执行环境。

平台核心脚本引擎为性能测试脚本设计了多种生命周期控制，以适用不同的场景，并使性能最优化。在脚本编写过程中，用户仅需要在 Groovy 脚本中使用内置的几种注解便可对脚本的执行和数据采集进行精确灵活的控制，如测试类生命周期、事物、执行权重等，大大提升脚本开发效率。

2. 容器部署

为了快速的创建测试集群，Agent 采用 Docker 容器通过镜像方式进行自动化部署。这样做好处如下：

利用镜像方式，弹性伸缩快捷；

利用 Docker 资源隔离，不影响 CDN 服务；

运行环境集成，不需要额外配置运行所需类库；

每个 Agent 的资源标准化，能启动的虚拟用户数固定，应用不需要再做资源调度；

3. 服务通信

Task Service 采用了 gRPC 与 Agent 进行通信，通过接口描述语言生成接口服务。gRPC 是基于 http2 协议，序列化使用的是 protobuf3，并在除标准单向 RPC 请求调用方式外，提供了双向流式调用，允许在其基础上进一步构建带状态的长链接调用，并允许被调用服务在会话周期内主动向调用者推送数据， 其 java 语言版采用 netty4 作为网络 IO 通讯。使用 gRPC 作为服务框架，主要原因有两点。

服务调用有可能会跨网络，可以提供 http2 协议；

服务端可以认证加密，在外网环境下，可以保证数据安全。

4. Agent 实现

Agent 采用多进行多线程的结构设计。主进程负责任务的接收、预处理和 Worker 进程的调度。将任务的控制和执行进行进程级别的分离，这样可以为测试的执行提供相对独立且高度灵活的类库环境，使不通的任务之间的类库不会产生冲突，并有益于提升程序运行效率。

Agent 与 Task Service 保持通信，向系统注册自身并获取指令。根据任务需要启动 Worker 进程执行任务，主进程负责管理 Worker 进程的生命周期。Worker 进程启动后会通过 gRPC 与主进程保持通信，获取新的变更指令，如线程数变化通知，及时进行调整。

5. 数据收集和计算

实现秒级监控。数据的收集工作由 Monitor Service 完成，也是采用 GRPC 作为服务框架。Agent 每秒上报一次数据，包括性能，JVM 等值。

Monitor Service 将数据经 Kafka 发送给 Compute Service，进行数据的流式计算后，产生 TPS，包括 TP999，TP99，TP90，TP50，MAX，MIN 等指标存储到 ES 中进行查询展示。

为了计算整体的 TPS，需要每个 Agent 把每次调用的性能数据上报，会产生大量的数据，Agent 对每秒的性能数据进行了必要的合并，组提交到监控服务以进行更有效有的传输。

新式全链路压测的打开方式

ForceBot 的工作基本原理就是站在真实用户角度出发，从公网发起流量请求，模拟数百万级用户 0 点并发抢购和浏览，由于压力机分布在全国各地，不同的运营商，所以最接近真实用户的网络环境，对于读服务通过回放线上日志形式模拟用户请求，数据更分散、更真实，避免热点数据存在，对于写服务则模拟用户加入购物车、结算、更改收货地址、使用优惠券、结算、支付等核心链路环节，每次军演都会发现新的问题和瓶颈，对后期的资源扩容、性能调优都会有针对性的备战方向，最终让研发兄弟对自己系统放心，让消费者购物无忧。

全链路压测方案最重要的部分是被压系统 **如何去适应这种压测，如何精准的识别测试流量，如何协调整个研发系统统一打标透传压测流量，并做相应的处理**，尤其对脏数据的隔离和清理尤其关键，被压测的系统要和线上环境保持一致，又要隔离这些数据，需要确保万无一失，尤其订单部分环节，出现差错是不可恢复的。

压测数据读的服务都是来自于线上服务器的日志真实数据，数据最终会由各个系统通过标记进入回收站，最终被清理。军演平台本身不会侵入到系统去清理垃圾数据，每次军演压测首先会有一个目标值，比如按照历史峰值的一个倍数去动态加压，如到了这个目标值，各系统性能表现都非常好，那么继续加压，直至有系统出现瓶颈为止。

军演机器人的过去与未来

全链路压测可以理解为网络链路 + 系统链路，网络链路是用户到机房的各个网络路由延迟环境，系统链路是各个系统之间的内部调用关系和强依赖性。 其实去年双 11，京东就采用了 ForcBot，只不过仅仅针对重要链路；因为如果没有核心系统首先参与进来，小系统是搞不起来的，因为小系统强依赖核心系统。

今年备战 618，ForceBot 技术架构没有太大调整。我们工作一部分是平台用户体验的一个优化和性能的优化，让有限的压力机产生更大的压力请求；另外一个是整合被压测的系统监控，实时展示。

京东未来希望 ForceBot 可以实现“人工智能预言”。现在还在逐步的做，我们希望未来的全链路压测引入 AI 技术，通过人工智能预言各个系统的流量值和资源分配建议，根据线上的系统军演数据预言未来的大促各系统场景，举个简单的例子，在 ForceBot 平台上录入每秒一千万并发订单场景，以现在的系统去承载，各系统是一个什么样的性能指标和瓶颈点在哪？ 这就是我们思考要做的。

转载于:https://my.oschina.net/cctester/blog/994735