基于真实业务数据的软件质量自动校验方法

田标[1]，崔伟[1]，黄慧[1]，冯小龙[2]
（1. 天翼数字生活科技有限公司，广东 广州 510630；
2. 中国矿业大学信息与控制工程学院，江苏 徐州 221116）

摘　要：软件性能校验结果有助于检验业务集群能否满足访问规模需求，但当前工具存在需要估算参数值、手工输入数据较多、性能偏低、应用复杂等不足。深入分析了JMeter并发机制设计的不足，提出了一种具有一般性的量化访问速度控制机制及基于协程的并发设计方案，给出了相应的测试即服务（TaaS）的架构设计和分布式自动性能测试方案，具有更高的并发性能，能使用真实业务数据并自动调整访问速度或调度更多服务器执行，自动汇总分析全量响应，有助于提高软件质量校验的效果及结果的直观性。

关键词：性能测试；量化速度控制；自动校验
中图分类号：TP319
文献标志码：A
doi: 10.11959/j.issn.1000−0801.2024071

An automatic verification method for software quality with real business data

TIAN Biao[1], CUI Wei[1], HUANG Hui[1], FENG Xiaolong[2]
1. E-Surfing Digital Life Technology Co., Ltd., Guangzhou 510630, China
2. School of Information and Control Engineering, China University of Mining and Technology, Xuzhou 221116, China

Abstract: Performance tests help to verify whether a business cluster can meet its access scale requirements, but current tools have problems such as evaluating proper parameters and requiring manual inputs, low performance, and difficulty in automation. A comprehensive quantitative request rate control mechanism and concurrent design approach based on coroutines were proposed, in contrast to JMeter's reliance on threads and locks. Furthermore, an architecture design for its corresponding TaaS service and a process automation scheme were introduced. This scheme automatically perceives responses, adjusts access rates accordingly, and schedules additional server executions using real business data, significantly enhancing operational efficiency and result intuitiveness.

Key words: performance test, quantitative rate control, automatic verification

收稿日期：2023−10−26；修回日期：2024−03−05

0　引言

软件性能数据描述了一定服务器和数据规模的系统在不同负载压力下的稳定性和能服务的访问量，高性能、稳定的系统可以根据实际访问量的变化及时调整系统部署规模，有助于降低IT资源成本。并发性能满足业务需要的要求，是反映业务能否在预期访问量下平稳运行的关键数据。并发性能还部分决定了用户操作效率，是影响用户留存和用户体验的重要因素。

但识别软件负载相关的性能问题及其根因一直是一个复杂且耗时的过程，工作量大且很大程度上还要依赖该领域的一些专家知识[1]，这意味着其结果数据的解读也有技术门槛。云计算、微服务和大数据技术的大量应用，增加了互联网、金融、电信等业务组成系统的规模及调用复杂性，加剧了这些问题，还会出现一些难以复现的问题，不利于更深层次问题根源的解决。软件并发性能校验技术上的其他挑战还有：常见随机生成校验请求数据的做法，在复杂接口调用过程中因参数组合及请求过滤会降低真正执行校验的访问量，也可能无法覆盖部分真实业务场景[2]；因业务需要，部分软件允许的并发访问数量要在短时间内随时间按一定规律变化，带来了细时间粒度的性能校验需求；高性能、大容量互联网业务的并发性能校验，对校验工具的性能、分布式能力提出了越来越高的要求，直接关乎资源需求、校验效率和效果；软件并发能力并非一个静止数值，受到网络、I/O、CPU、内存占用等的影响，在这样的环境下找出软件并发能力的真正上限，还需要面对怎样提高效率的挑战。目前在软件性能校验方面的研究有以下内容。

国内截至目前主要以该领域工具的应用为主，关于相关工具设计开发方案的探索有限。文献[2]指出通过回放线上业务流量并对比回归，可以避免上线后才暴露问题甚至被迫下线修复的情况，但是缺少根据网络流量获取可用数据的技术原理和要点，以及在性能测试过程中怎么保证满足一定容量要求的输出规模。文献[3-4]介绍了软件性能测试的目的、分类和衡量指标，指出在使用开源的JMeter做性能测试前需要输入线程数来模拟相同数量的用户数、指定Ramp-up期时长和循环次数，二者在当前JMeter及其用途的研究上具有一定代表性。文献[5]基于LoadRunner通过规划工作过程并为各阶段建立标准模块再应用的方式改善性能测试效率低、复杂度高的问题，但在设计、执行和结果分析上仍依赖人工操作，有一定的技术门槛，而且LoadRunner无法二次开发，其脚本的开发和维护将增加工作量。文献[6]指出LoadRunner同步压力请求的方式限制了其施压能力，它采用基于Jpcap开发的程序录制脚本，把脚本转换为处理从请求到响应解析过程的自动机，然后使用Netty的任务执行线程池，基于任务队列获取自动机实例实施性能测试，提供了恒定、递增、方波、曲波和自动摸顶5种压力模型，其爆发式和队列式的任务调度方案无法实时控制请求量，无法并行录制并直接应用网络流量，脚本及其带来的I/O和Java线程模型都接近JMeter的方案，未提供新的实质性并发请求调度和控制机制。

国外的同类研究深入到通过工具开发提高性能校验效率和发现性能瓶颈上。文献[7]介绍LoadRunner价格昂贵，存在安装和配置问题；JMeter使用简单且UI更清晰简洁，提供了丰富的结果分析选项和一些优秀插件，但部署耗时更久，文献[7]认为JMeter比任何其他同类工具都能给出更一致的结果。文献[8]提出性能测试往往采用根据以往知识、先前经验或企业政策等确定的标准数据，包括自动生成的随机数据，对于暴露问题或帮助用户发现根源不够用。文献[1]指出企业级应用性能问题的发现取决于其输入负载和用户专业知识，但即使同一个软件的不同版本的负

载量需求仍可能未知，导致预设置静态访问量的做法可能很长时间都无法找出性能问题及根源，因此基于JMeter提出了根据响应指标实时动态调整输入负载的自动化方案，降低了对用户的专业要求和资源消耗，能更快发现性能问题。文献[9]指出文献[1]的方案仅开发了基本研究原型，用户仍需要手工设置许多参数，为此文献[9]通过一个JMeter插件支持用户把性能测试过程分为2个阶段，设置总时长、各阶段时长占比、第一阶段要模拟用户数的范围和负载敏感事务量百分比，还需要设置第二阶段负载量的调整策略、调整增量、负载敏感事务量百分比、饱和点比率和采样率；然后按配置自动不断增加模拟访问量，直到达到错误率等阈值为止。文献[9]的方案仍要求用户必须为一批参数正确设置适当的参数值，设置不当仍会降低效率或得不到期望结果。

针对现有软件质量校验工具的不足和挑战，本文围绕这类工具的核心技术设计并实现了一个使用真实业务流量作为输入数据的分布式校验工具，具有不同于线程的高性能海量并发控制机制，支持灵活的压力控制，允许实时量化调整访问速度，与JMeter相比性能更高，运行时占用的资源更少，开发更简单，客户端可独立执行，不需要预先部署Java运行环境（Java runtime environment，JRE）。本文还从协议底层技术出发，提供了获取不同类型通信协议的访问流量并提供丰富应用模式的方案。该方案需要的初始化参数比文献[9]的更少，甚至不需要任何初始参数，能在性能测试过程中自动找出吞吐量上限，避免了同类工具的输出报告被误解或无法解读的情况。

本文所述方案的主要贡献如下。

（1）围绕使用真实数据高性能校验软件质量的需要，从底层技术开始实现了不同协议流量的复制、解析、处理以及按协议变速发送流量、对比验证结果的方案。

（2）允许根据实际需求使用任意压力控制模型，支持在性能测试过程的任意时刻，通过任意可积速度函数量化控制实时输出的访问量并结合积分运算精确控制总访问量，可以灵活满足实际软件校验场景的访问量或访问速度控制需求。

（3）放弃线程，基于协程设计更加轻量级、更简单但更高性能的并发控制机制，使相同环境下每秒能比线程输出更多请求，同时降低资源消耗。经验证，在4核8 GB内存的服务器上每秒可以输出超过40 000个请求；在1核2 GB内存的服务器上可以稳定地每秒输出万级请求。

（4）提供从客户端自动采集、分析验证响应并调整访问速度及其增量的方案，能自动判断并发起分布式校验直至发现软件性能上限并输出直观说明。

1 基于真实业务数据的软件质量自动校验方法设计

1.1 JMeter并发机制及其不足分析

JMeter[10]是目前应用最广泛的优秀开源性能测试工具之一，它采用Java多线程构建其访问引擎，JMeter并发机制的主要构成如图1所示。展示了其从整体到并发访问机制的主要设计。它通过一个线程模拟一个虚拟访问用户，具体虚拟用户数可以在类ThreadGroupGui的“线程数”文本框输入，然后先在类ThreadGroup的start()中通过继承AbstractThreadGroup获得设置的线程数，再循环创建指定数目的Java线程对象JMeterThread，同时根据用户设置的“Ramp-up时间”控制新创线程在正式工作前应先延迟的时长。ThreadGroup根据设置的线程循环次数和持续时间影响访问次数，但最终的实际访问次数还要结合对象服务的实时性能才能确定，然后由JMeterThread通过Sampler接口并发访问不同协议的对象服务并处理响应。为尽量接近浏览器等现代软件的通信和性能优化行为，JMeter会预先建立与对象服务之间的网络连接并缓存起来供整个过程复用。

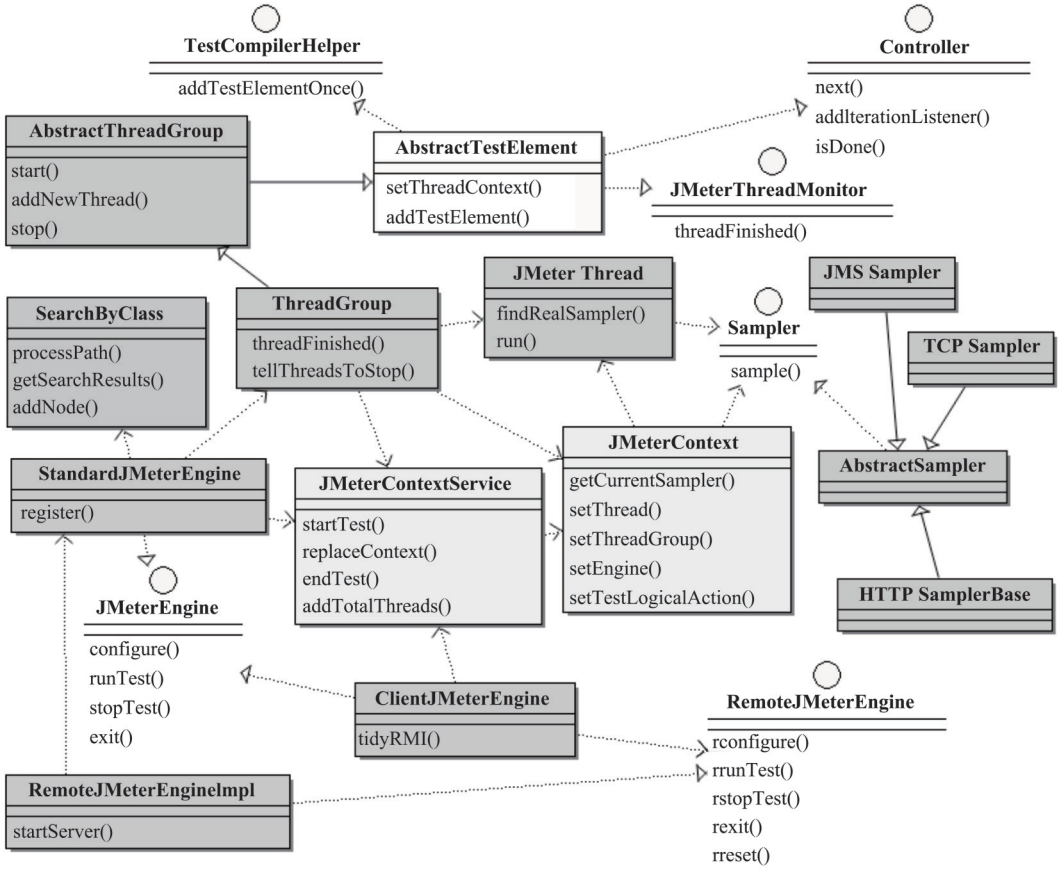


图 1　JMeter 并发机制的主要构成

常规地采用上下文对象协调管理 JMeterThread 和相关对象并在这些对象之间共享数据，有助于获得更好的性能、稳定性和可扩展性。为获得最终结果，它还通过基于多线程和 Sampler 的并发采样机制计算获得响应结果。在需要分布式测试时，基于远程方法调用（remote method invocation，RMI）协议通信。

因此，用户必须先对软件并发性能对应的虚拟用户数有初步相对准确的判断，才可能设置相对正确的“线程数”和“Ramp-up 时间”，即根据用户设置的参数创建的 Java 线程组每单位时间能输出完成的请求量必须能超过软件的并发访问量的上限，才有可能得到反映软件真实并发性能的结果，否则用户重复校验的次数越多，越浪费时间、资源。文献[11]指出锁可以对多线程程序的性能造成严重影响，考虑 Java、.Net、Rust 等不同技术平台并发机制设计和实现上的差异，在软件性能校验这样对性能要求极高的场景下，可以认为用户基于 JMeter 获得的关于这些参数设置的经验值无法直接平移应用到其他技术平台。文献[19]解决低并发问题的措施为，根据测试启动相应数量的子进程，各子进程再启动指定数量的子线程执行业务逻辑。考虑多线程并发安全和锁带来的开发和调试难题，怎样高性能地为包含性能测试在内的质量校验场景设计并发控制机制的问题便跃然纸上。

文献[12-13]对比了线程和协程的并发机制的性能差异及各自的调度器机制，发现协程的系统资源消耗比线程低，在相同配置的服务器上可以创建更多协程，可以比基于线程的方案输出更高的吞吐量，而且开发简单，可以实现无锁并发开发，能获得更高的并发性能输出[11]。所以，在相

同服务器上，可以认为采用协程而非像JMeter那样基于线程设计的并发机制，每单位时间可以完成更多性能校验请求的生成、执行，输出相同数量的性能校验请求并完成结果处理，采用协程需要的内存、CPU比采用线程时少，当需要同时调度多个服务器执行分布式性能校验时，该结论也成立。

1.2 软件访问流量复制和协议解析机制设计

性能校验场景下网络数据包的读取、协议识别、使用意味着需要处理大量不同类型的对象，需要从底层开始异步并发地处理，否则只能用于录制脚本等相对简单的操作[6]。无论在通信发起端还是接收端，通过libpcap都能实时读取任意网络设备或地址的通信数据[14]，但由于没有像传输控制协议（transmission control protocol，TCP）通信机制的保障，读取的数据包不仅无法反映实际通信顺序，还要准确组装请求及其响应并按实际协议解析后，才能继续使用。

按协议解析一个消息的数据包的过程如图2所示，主要步骤如下。

步骤1 从以太网、IP和TCP层根据网络类型解析数据包中的来源和目的IP地址、端口号、序列号（Seq）、确认号（Ack）、结束标志（FIN）、同步标志（SYN）及有效载荷（payload），记录读取数据包的时间戳、请求和响应方向。

步骤2 以源和目的端口、IP地址、Ack号构建消息ID，按是否有相同消息ID组装消息数据包，设置消息解析器。

步骤3 按协议开始函数解析判断消息开始，成功解析说明发现了所属通信协议，否则尝试按其他协议解析，成功后根据协议、消息内容设置消息是请求或响应，按消息ID关联组装请求及其响应。

步骤4 不断添加消息数据包并根据Seq号升序排序，不断计算消息长度、结束时间。

步骤5 按协议结束函数判断消息是否结束，确认结束时缓存消息再循环往复。如消息有更多数据包，则不断增加Ack号、合并数据包到消息后再检查是否结束。

消息解析器通过函数指针调用各协议数据包的开始和结束函数，识别后再调用对应的解析函数获得明文数据。结束函数按协议要求和Seq号校验数据的完整性并设置数据，保障可以确定一个完整消息的结束位置以正常解析。

由于libpcap是单线程的[14]，需要通过TCP、消息队列等异步汇聚处理消息后才能用于质量校验。结合相邻请求的时间间隔可得实际请求频率，通过后续量化控制校验请求的发送频率并验证对比响应可以把真实流量用于相同协议软件的质量校验，还可以根据目的IP地址把一个服务器的流量都集中发送给另一个服务器，所以能面向各类协议的软件逼真地模拟实际运行场景，有助于提

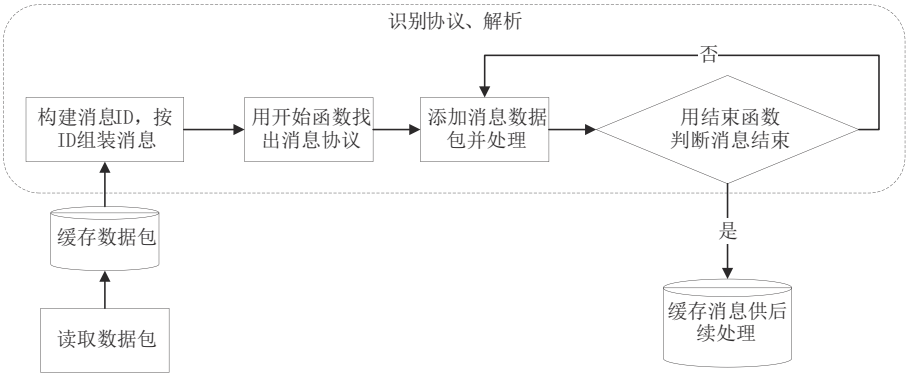


图2 按协议解析一个消息的数据包的过程

高质量校验覆盖面和重现问题。虽然JMeter[10]等也可以通过外部文件引用真实业务数据，但目前仍无法应对需要实时海量数据且具有时效性认证信息的业务场景，这些要求在工作中都普遍存在，而基于业务流量的方案能高效自动解决。

1.3　任意压力模型及访问速度控制的原理、优点

当需要一定访问量时，校验工具如果被暂停一段时间，随后对象服务可以因机器负载下降等原因承受更大访问量，这样实际结果就会偏大，反之则相反。因此除降低并发机制的损耗，还要尽量细粒度地控制访问量，使任何时刻已完成和正在执行的访问量之和尽量接近需要。

这里把对软件的瞬时访问量定义为时间t的速度函数$r(t)$。设$\forall t1$时刻开始对软件实施校验，$\forall t>t1$，理想情况下，实际已完成访问量$f(t)$应等于理论上应完成的访问量$\int_{t_1}^{t} r(t)\mathrm{d}t$，此时的访问流程控制逻辑如下。

（1）当$f(t)<\int_{t_1}^{t} r(t)\mathrm{d}t$时，立即开始下一次访问。

（2）当$f(t)\geqslant\int_{t_1}^{t} r(t)\mathrm{d}t$时，应根据当时的实际访问性能精确控制先等待一定时长再继续访问。按微分的概念，取纳秒形式的瞬时访问速度时间单位值n，在执行下一个访问前，该时刻应先等待的时长$p(t)$（纳秒）的计算式为：

$$p(t)=(f(t)+1-\int_{t_1}^{t} r(t)\mathrm{d}t)\times n/r(t) \qquad (1)$$

实际计算时取$r(t)$的反导函数替换$\int_{t_1}^{t} r(t)\mathrm{d}t$的计算，在开始每个新请求之前都执行上述流程控制逻辑，从而在整个过程的任意时刻都能准确控制实际访问量。

按这样的机制，软件并发能力越高，得到的等待时长越低、越准，访问速度的控制粒度越细，与实际工作场景越相符。软件并发能力不高时，由于按更细的时间粒度及时控制等待，不仅仍然能准确控制单位时间的访问量，还可以降低资源占用。另外，由于可选任意可积速度函数按需控制访问速度，能满足各类型软件的性能校验需求，具有充分的灵活性，可以避免像文献[6]、文献[10]那样只能提供有限的几种压力控制模型。

因此，通过不断提高访问速度，可以无限逼近软件并发能力上限，软件响应性能、错误率会快速接近允许的质量阈值；对实际访问速度或根据第1.2节的流量得到速度函数后再变换不同的速度函数，可以得到软件稳定性等运行质量指标的变化情况，提高校验效率，再结合服务器负载等监控数据就不难获得性能校验结果了。

如果在流程控制逻辑（2）处不等待而是直接并发地开始新访问，就可以简单地通过不同数量的并发访问单元快速获得校验结果，结合性能、错误率的变化情况，不难找到软件的吞吐量上限。

1.4　软件质量校验客户端工具设计

客户端工具的主要对象设计如图3所示，首先由FlowListener负责按第1.2节的方案复制、解析软件流量得到消息，Accessor通过AccessController实现第1.3节的控制逻辑并利用业务访问量消息按式（1）基于协程控制访问速度和访问量，在执行请求环节采用Target适配不同通信协议，利用一系列子类封装按对应协议基于真实流量访问软件的行为。Analyzer负责通过对比实际响应与第1.2节得到的流量响应判断请求结果是否正常[2]，可以自动发现异常响应及其具体数据，有助于提高软件及其质量校验的效率和质量，还可以再计算、分析结果并计算后续操作建议，例如，在成功率或最大性能、TP99超过一定阈值时，提示减速；否则若未指定速度则应发起分布式校验，若当前吞吐量t>单机每秒最大请求量m时，建议后续调度t/m+1台同配置服务器；如$t<m$

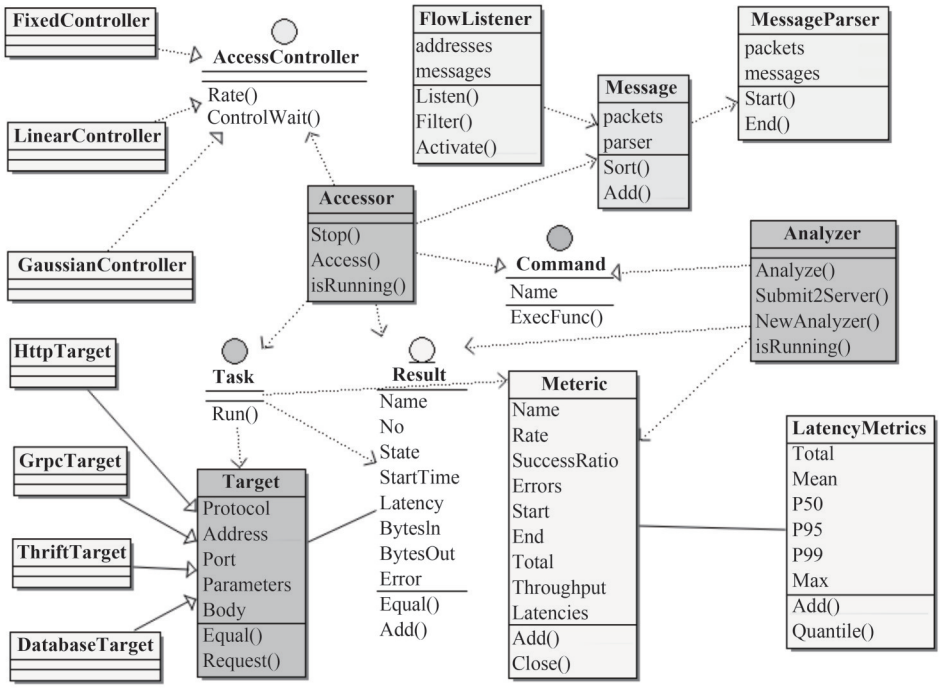


图3　客户端工具的主要对象设计

且m–t超过一定阈值则提示增速，否则投入更多服务器。

对于同一个协议在不同软件间的差异，可采用命令模式把校验过程中可能需要的连接、请求、响应解析、加/解密、编/解码等操作分别封装为对应的原子命令，各命令间利用通道、流、第三方队列等通信。

1.4.1　基于协程的高性能并发请求和响应处理机制设计

传统的并发方案主要立足线程、线程池设计，同类工具中文献[6]和文献[10]是典型例子。文献[15]通过主进程操作一系列子进程间接创建更多线程模拟用户访问以期望获得更高并发性能输出，该方案实质上仍然是多线程机制，由于进程及其调度的资源占用比线程更高，会进一步限制服务器的施压能力。综合文献[16-18]，这里组合协程、通道而不使用锁，基于内存实现各工作单元之间的并发协作，基于协程和通道实现高性能并发访问和响应分析如图4所示，这里结合文献[18]中的“固定worker工作池”模式，精简设计后采用控制协程根据访问速度要求动态创建一系列负责执行访问并搜集响应的执行协程，可以灵活支持第1.3节的访问速度量化控制机制。主要过程如下。

（1）在访问工具Accessor中按第1.3节选择适当的可积时间函数作为选定的速度模式，设置允许的总执行时长及所需参数并创建相关对象，记录速度，发起整个过程的执行。

（2）Accessor经控制协程记录开始时刻，循环顺序执行：访问控制、根据并发量需求在所有执行协程都忙时创建新执行协程、向通知通道发送消息触发实际访问。

（3）执行协程通过AccessController基于选定的速度函数控制瞬时访问速度及请求量。支持不限速访问，即每次访问结束后不等待立即开始新访问，有利于提高校验效率。

（4）主协程经通知和结果通道分析，计算响应结果并按一定编码输出。

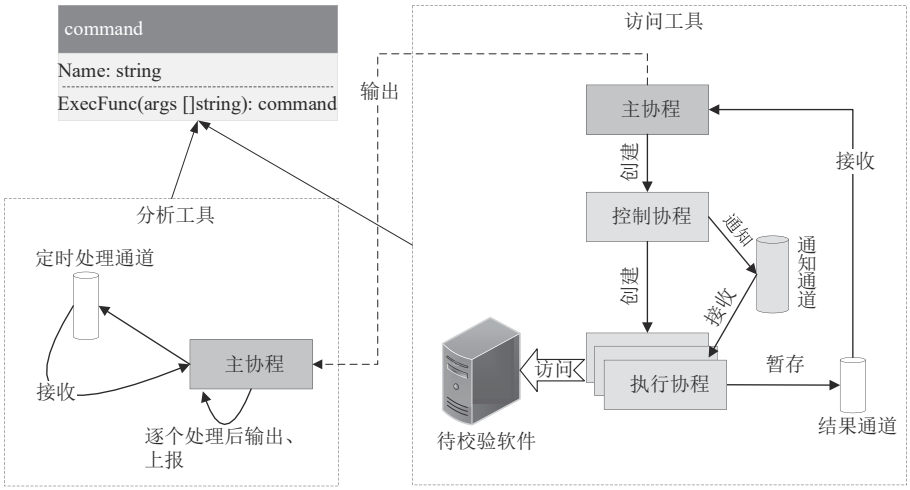


图4　基于协程和通道实现高性能并发访问和响应分析

1.4.2　分布式自动校验过程设计

鉴于分布式质量校验过程中不同客户端之间没有高并发、大量短连接和读/写操作，完全可以复用现有成熟技术尽量简单、方便地完成客户端之间的通信，不必像文献[6]、文献[10]为此开发系列功能。可以设计如图5所示的分布式性能自动校验，主要步骤如下。

步骤1　按文献[19]自动发现对象服务的技术类型、总服务实例数，基于这些数据自动设置一个初始最大协程数，使用一台服务器“不限速”访问对象软件，调整最大协程数快速找出一个最大的吞吐量。不需要用户设置任何初始参数，比文献[9]更简单。

步骤2　根据结果确定后续动作：若运行质量结果不符合要求，则降低协程数重试，若结果仍不符合要求则继续，否则返回最近一次的结果

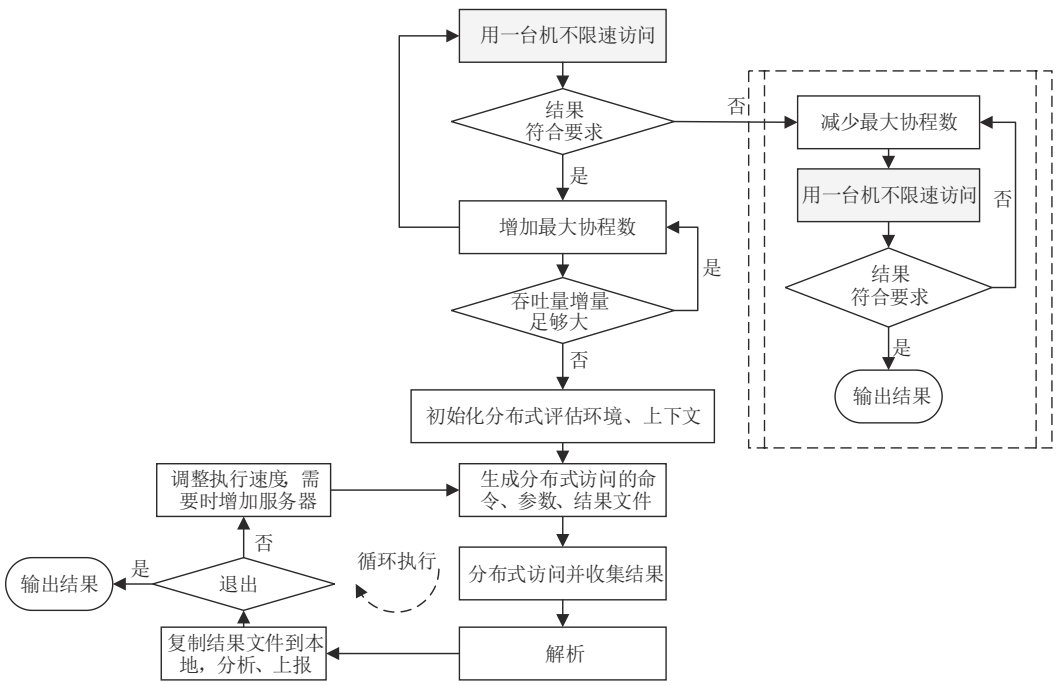


图5　分布式性能自动校验过程

并退出；若步骤1的结果符合要求，则增加协程数并重试，如果新结果符合要求但吞吐量提高不大而且最大协程数接近或超过了服务器容量，则开启分布式校验并由Analyzer计算需要的初始服务器数*Sn*。

步骤3 采用*Sn*个服务器以阶梯式速度（单个服务器时也可类似处理）或第1.3节所述适当的速度函数同时访问，初始速度为步骤2得到的吞吐量，阶梯各层级都采用固定速度并持续足够时长：初始化访问序数计数、退出标识，把步骤2的吞吐量作为各服务器的初始访问速度*r*，设置最大允许的评估次数，按结果精度要求设置每次改变速度的步长*s*。

步骤4 循环执行以下步骤得到校验结果，直到不符合质量要求或达到最大评估数后增加速度和*s*再重试。

（1）组装请求并批量执行、解析结果，得到各服务器的性能（最大、TP99，可扩充）、成功率，再与阈值比较并设置退出标识，合并各退出标识为整体退出标识，自增执行计数。

（2）从发起服务器自动登录其他服务器，搜集结果然后调用分析工具得到综合结果并上报。

（3）自动决定后续动作：若此次循环结果不符合要求，则把*r*=*r*−*s*作为新速度重试；若上次循环结果不符合要求但当前循环正常或当前循环次数超过允许值则返回当前结果并退出；如果相邻两个循环的结果都符合要求，则检查二者吞吐量变化量是否大于一定量，不大于则增加一个执行服务器重试，若大于则令*r*=*r*+*s*再重试。

综合这部分的数据可以对各类协议的不同规模软件按统一内容模板输出质量校验报告，直接自动输出：通过流量校验发现的问题、所属接口和相关数据，方便后续针对性校验并适时通知用户；符合一定运行质量标准时，软件允许的并发访问量的上限及各运行质量指标的具体表现；不同规模的服务集群分别能支撑多大用户量同时访问以及不同访问量下错误率、性能、流量等的变化趋势，直观展现软件性能瓶颈。还可以基于同类软件的校验结果绘制雷达图，直观展示软件各质量指标与同类软件的对比情况，把结果数据的解读对用户专业上的要求降到最低，不再需要通过如标准化规范约束相关工作过程[5]。

1.5 基于真实业务数据的软件质量校验服务

与传统意义上的轻量级客户端不同，校验过程中客户端之间通信有限，这里把从服务端实施调度、参数下发外的操作都放置在客户端，使之可编译为可执行文件独立运行并方便部署执行。基于真实业务数据的软件质量校验服务架构如图6所示，包括测试即服务（test as a service，TaaS）质量校验服务、客户端工具两部分。

TaaS质量校验服务主要用于业务、配置、结果的管理、分析，主要功能包括执行服务器调度、结果分析和报告输出，可以提供比JMeter报告输出更好的效果、更有价值的分析结论，还可以更方便地结合校验结果和调用关系图，针对复杂微服务提供容量分析，并协助定位性能瓶颈，尽量降低结果解读所需的用户门槛。校验过程中执行服务器和对象服务的访问量、性能及错误量等运行指标的趋势分析，同时向用户提供集中访问界面。

客户端工具根据对校验过程的划分，包括一系列命令行工具，通过命令链模式可以组合为一键评估工具，包括流量复制解析、访问和分析工具。流量工具支持读取、解析一系列协议软件的访问流量，例如，按协议解析数据库、Redis等的流量再处理转发；访问工具基于第1.3节的速度和访问量模型控制对软件的访问、通信协议适配并采集处理响应；分析工具负责响应结果的分析、对比、汇总上报，支持自动分析响应数据获得问题分类及列表、输出后续动作建议，包括自动调度其他服务器发起分布式校验。客户端可以部署于不同机房，根据服务端指令从不同地点实

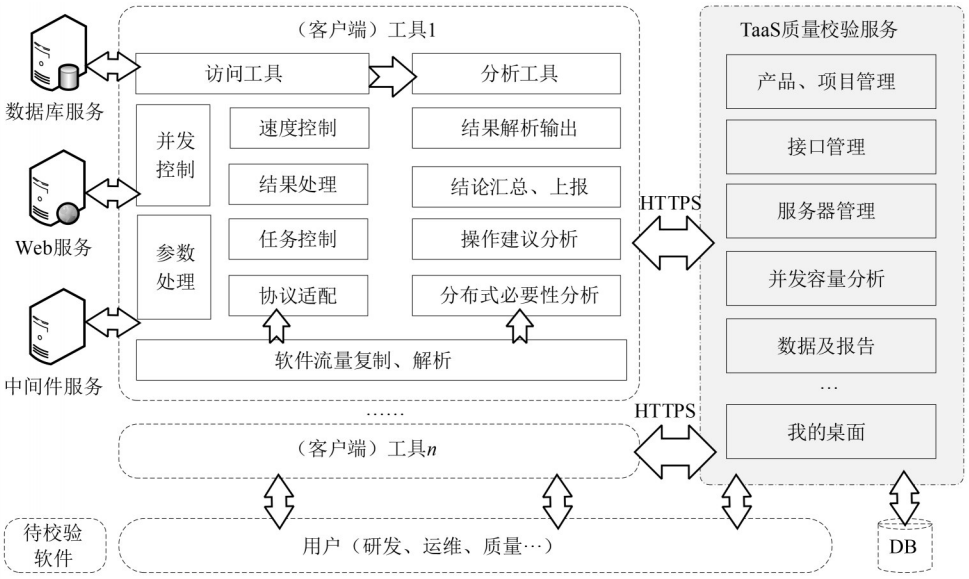


图6　基于真实业务数据的软件质量校验服务架构

施质量校验。

还可以在服务端采集分析校验过程中软件的日志和异常堆栈[20]，自动生成全量质量报告，从软件外部响应、自身错误和异常、服务器负载几个维度描述软件的并发能力，进一步可以结合软件内部各环节的运行质量定位程序漏洞或主要瓶颈，助力持续提升软件质量。

2　主要实验和应用结果

2.1　与基于线程的并发机制的对比

为尽量降低垃圾收集器对结果的影响，这里选择在相同环境下，分别采用JMeter v5.5和根据本文方案开发的工具，对一个完全做简单内存操作的Java Web应用采用JMeter的循环不间断请求的方式做性能测试，每次执行过程的持续时长为200 s，通过不断调整线程数/协程数，以请求错误率为0并且最大性能不超过1 s作为运行质量标准，找出不同配置服务器下每个方案的最大吞吐量。不同配置服务器下两种方案的应用效果对比见表1。

对比表1结果可知，此方案的吞吐量输出在各规格服务器上都优于JMeter，从性能上看更稳定；在4c8及更低配的服务器上输出的吞吐量差异超过20%，说明此方案对于服务器配置的要求比JMeter低，可以降低服务器成本；从协程/线程数和性能的变化上看，随着配置的提升，该方案的稳定性更好，更容易预测达到最大吞吐量的协程数范围；按JMeter通过线程数模拟等量用户数的理念实施性能测试，取实际在线用户数对应的线程数时得到的吞吐量容易与系统最大吞吐量有不少差异，在实验中发现这个差异可以超过1倍，对应的线程数差异更大，可以超过100倍。

关于性能测试过程中的资源消耗，以在4c16的服务器上的执行过程为例，4c16服务器上两种方案的资源消耗对比见表2，说明在这里的校验场景下JMeter的CPU占用比此方案的高，最近1 min系统负载的最大值差异超过7倍，每秒上下文切换次数的最大值超过3倍；在内存消耗上此方案不仅更低，还更稳定。

2.2　主要应用结果

对部分Redis操作读取流量、按协议解析、存储的效果如图7所示，各字段依次是客户端地址、请求内容、服务端地址、响应内容、响应长度和性能。从响应可判断对应请求是否成功，结

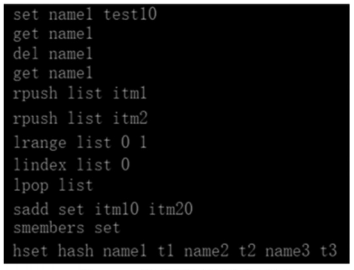
表1　不同配置服务器下两种方案的应用效果对比

服务器规格	方案类型	线程/协程数	性能/ms			吞吐量（请求数/s）	吞吐量增加
			平均	TP99	最大		
8c32	JMeter	100	2.12	12	204	45 936.14	14.03%
	此方案	1 050	16.298	40.924	118.799	52 383.22	
4c16	JMeter	160	3.71	1	548	41 559.36	8.33%
	此方案	970	16.308	40.707	233.127	45 021.72	
4c8	JMeter	350	9.8	23	1344	31 433.64	28.02%
	此方案	1 000	18.559	46.233	266.144	40 242.17	
2c4	JMeter	100	6.48	16	993	15 014.11	22.16%
	此方案	550	25.083	57.607	561.862	18 341.45	
1c2	JMeter	50	6.93	20	244	7 164.14	25.83%
	此方案	30	3.273	15.387	146.531	9 014.4	

注：服务器规格8c32指8核CPU、32 GB内存，其余类推；JMeter是线程，此方案是协程。

表2　4c16服务器上两种方案的资源消耗对比

方案类型	最大CPU占用	最大最近1 min系统负载	内存消耗/MB	每秒上下文切换最大次数/次
JMeter	71.43%	59	580~2 150.4	65 000
此方案	61.1%	7	540~720	18 000



(a) 某Redis类服务的部分操作

req_addr	req_body	resp_addr	resp_body	resp_content_length	rtt
10. :9097	set name1 test10	10. 379	OK	2	0.1593409926
10. :9097	get name1	10. 379	test10	6	0.1528660059
10. :9097	del name1	10. 379	1	1	0.1306779981
10. :9097	get name1	10. 379		0	0.1176680028
10. :9097	rpush list itm1	10. 379	1	1	0.3023290038
10. :9097	rpush list itm2	10. 379	2	1	0.2136210054
10. :9097	lrange list 0 1	10. 379	itm1 itm2	9	0.2036139965
10. :9097	lindex list 0	10. 379	itm1	4	0.1687230021
10. :9097	lpop list	10. 379	itm1	4	0.1606419981
10. :9097	sadd set itm10 itm20	10. 379	2	1	0.1871259958
10. :9097	smembers set	10. 379	itm10 itm20	11	0.2994489968
10. :9097	hset hash name1 t1 na	10. 379	3	1	21.8960342407

(b) 读取对应流量并解析保存结果到Cassandra的效果

图7　对部分Redis操作读取流量、按协议解析、存储的效果

合请求发生时间可获得请求频率，因此可以改变频率重新执行所有请求。类似地，对HTTP、JDBC、RPC等协议的软件，重放包含问题发生时间区间的访问流量有助于复现问题，重放导致软件错误和异常的流量有助于找出潜在缺陷、异常用户或安全等漏洞，把流量按软件容量以可控速度访问可用于性能测试，可以避免随机生成的参数导致部分请求被过滤而无法获得软件并发能力真实上限的问题。

一系列吞吐量在几千到几万甚至更高的互联网业务，采用JMeter利用一台4核16 GB内存的服务器做性能测试时，经常会因为JMeter的错误率高无法正常完成，需要使用更多服务器，但采用上述方案开发的工具每秒能稳定输出超过40 000个请求。在应用上曾经对多个在线业务利用有限的服务器完成了持续几周的海量访问量下的疲劳校验。

该方案的客户端可执行文件只有十几兆字节，在实际工作中应用方便，无须事先部署JRE等环境，复制客户端工具的可执行文件到执行服务器后即可开始执行；可以选用任意形式的可积时间函数控制访问速度；在服务器的正常负载范围内，访问工具基于一定速度函数的访问量输出值与理论值的误差低于0.001。

在原型调研、设计阶段能方便地使用该工具实施软件质量校验，不需要专门部署业务系统到公网云服务，在性能校验过程中能收集软件的异常、错误，轻松验证不同技术方案的差异，还可以结合业务、通信协议和技术选型上的实际需求灵活定制。分布式自动校验能大量减少手工重复操作，可协助技术人员收集业务的大量并发能力数据，确认适当的服务器部署规模。

3　结束语

针对当前软件质量校验场景下，以 JMeter 为代表的方案的不足，本文就同类软件的设计提出了一种新型方案，包括采用业务和用户无感知的方式复制流量并按协议解析，结果数据既可用于功能正确性校验，又能用于性能测试，提出具有普适性的细时间粒度访问速度量化控制机制及相应的高性能软件质量校验服务设计。相比国内外同类方案可以根据实际需要采用任意可积函数准确控制访问速度，自动调整负载输出，实现了无人值守分布式自动质量校验，可以自动发现软件性能上限。经实际应用验证，本文方案能以更少的服务器、更少的人工干预完成软件质量校验，量化评估业务服务集群的规模是否得当。未来还可以结合大语言模型使实施更简单、高效，同时向各类用户输出更有价值的质量校验结论。

参考文献：

[1] AYALA-RIVERA V, KACZMARSKI M, MURPHY J, et al. One size does not fit all: In-test workload adaptation for performance testing of enterprise applications[C]//Proceedings of the Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering. New York: ACM, 2018: 211-222.

[2] 高文辉. 基于流量回放的测试平台在线上服务压力测试上的应用[J]. IT经理世界, 2020,1(23): 62-63.
GAO W H. Application of a testing platform based on traffic replay in online service stress testing[J]. IT Manager World, 2020, 1(23): 62-63.

[3] 唐承玲, 王虎, 李光平, 等. 基于JMeter的Web性能测试研究[J]. 电脑与电信, 2021(6): 65-68, 86.
TANG C L, WANG H, LI G P, et al. Study on the web performance test based on JMeter[J]. Computer & Telecommunication, 2021(6): 65-68, 86.

[4] 陈阿妹, 陈佳丽, 陈斌仙. 基于JMeter的Web性能测试的研究[J]. 九江学院学报(自然科学版), 2016, 31(1): 70-76.
CHEN A M, CHEN J L, CHEN B X. Research on web performance testing based on JMeter[J]. Journal of Jiujiang University (Natural Science Edition), 2016, 31(1): 70-76.

[5] 韦亚军, 张文文. 一种标准化的软件性能测试流程设计与研究[J]. 计算机测量与控制, 2022, 30(8): 31-37.
WEI Y J, ZHANG W W. Design and research of a standardized software performance testing flow[J]. Computer Measurement & Control, 2022, 30(8): 31-37.

[6] 李培殿. 基于Netty框架的性能测试系统的设计与实现[D]. 北京: 北京邮电大学, 2019.
LI P D. Design and implementation of performance testing system based on netty framework[D]. Beijing: Beijing University of Posts and Telecommunications, 2019.

[7] ABBAS R, SULTAN Z, BHATTI S N. Comparative analysis of automated load testing tools: Apache JMeter, Microsoft Visual Studio (TFS), LoadRunner, Siege[C]//Proceedings of the 2017 International Conference on Communication Technologies (ComTech). Piscataway: IEEE Press, 2017: 39-44.

[8] MOLYNEAUX I. The art of application performance testing-help for programmers and quality assurance[M]. Nanjing: Southeast University Press, 2009.

[9] HUERTA-GUEVARA O, AYALA-RIVERA V, MURPHY L, et al. DYNAMOJM: A JMeter tool for performance testing using dynamic workload adaptation[M]. Cham: Springer International Publishing, 2019: 234-241.

[10] JMeter socure[EB]. 2024.

[11] 郑龙. 多线程锁同步运行时特征分析与调优机制研究[D]. 武汉: 华中科技大学, 2016.
ZHENG L. The characteristic analysis and runtime optimization of lock synchronizations in the multithreaded programs[D]. Wuhan: Huazhong University of Science and Technology, 2016.

[12] 刘书健. 基于协程的高并发的分析与研究[D]. 昆明: 昆明理工大学, 2016.
LIU S J. Analysis and research on high concurrency based on ctrip[D]. Kunming: Kunming University of Science and Technology, 2016.

[13] 丁燎原. 基于协程的Web服务器原型的设计与实现[D]. 大连: 大连理工大学,2018.
DING L Y. Design and implementation of Web server proto-

type based on coroutine[D]. Dalian: Dalian University of Technology, 2018.

[14] 折波, 王强, 董凡, 等. IP网络流量资源共享系统设计与实现[J]. 深圳大学学报(理工版), 2020, 37(S1): 118-127.
ZHE B, WANG Q, DONG F, et al. Design and implementation of IP network traffic resource sharing system[J]. Journal of Shenzhen University (Science and Engineering), 2020, 37(S1): 118-127.

[15] 阿里巴巴集团控股有限公司. 一种测试用的压力生成方法及装置: CN102955721A[P]. 2013-03-06.
Alibaba Group Holdings Limited. A pressure generation method and device for testing: CN102955721A[P]. 2013-03-06.

[16] HOARE C A R. Communicating sequential processes[J]. Communications of the ACM, 1978, 21(8): 666-677.

[17] 王明松, 于营, 秦永佩. 基于协程(goroutine)的并发编程模型及应用[J]. 电子质量, 2022(6): 95-98.
WANG M S, YU Y, QIN Y P. Concurrency programming model and application based on goroutine[J]. Electronics Quality, 2022(6): 95-98.

[18] 李文塔. Go语言核心编程[M]. 北京: 电子工业出版社, 2018.
LI W T. Go language core programming[M]. Beijing: Publishing House of Electronics Industry, 2018

[19] 天翼数字生活科技有限公司. 服务器的元数据修复方法、装置、系统和计算机设备: CN110262926B[P]. 2021-10-08.
E-Surfing Digital Life Technology Co., Ltd. Method, device, system, and computer equipment for repairing server metadata: CN110262926B[P]. 2021-10-08.

[20] 天翼数字生活科技有限公司. 数据分析方法、系统、装置、计算机设备和存储介质: ZL20191 1147117.1[P]. 2019-11(2023-09).
E-Surfing Digital Life Technology Co., Ltd. Data analysis methods, systems, devices, computer equipment, and storage media: ZL20191 1147117.1[P]. 2019-11(2023-09).

[作者简介]

田标（1978− ），男，现就职于天翼数字生活科技有限公司，主要研究方向为互联网业务及其运维、质量等工具的架构设计和开发。

崔伟（1972− ），男，天翼数字生活科技有限公司安全运行维护部总经理，主要研究方向为互联网业务运营维护、网络信息安全和软件质量管理。

黄慧（1971− ），女，天翼数字生活科技有限公司数字化质量管理中心总经理，主要研究方向为互联网业务运营维护和软件质量管理。

冯小龙（1977− ），男，博士，中国矿业大学信息与控制工程学院副教授，主要研究方向为工业信息化、自然语言处理。