# 性能测试

## 一、相关概念

性能测试方法是通过模拟生产运行的业务压力量和使用场景组合，测试系统的性能是否满足生产性能要求。通俗地说，这种方法就是要在特定的运行条件下验证系统的处理能力。

### 基准测试

准备性测试，帮助验证性能测试脚本能够在性能环境中正确执行。通常使用一个虚拟用户执行单个用例，获得最佳的用例性能效果，然后将其和后续的测试数据做对比。

### 负载测试

这种方法是对一个系统持续不段的加压，看你在什么时候已经超出“预期要求”或系统崩溃。

最典型的性能测试类型。施加足够的负载来达到预期的并发压力，达到目标后不再增加。

目的：验证可用性，并发数，吞吐率和响应时间的性能指标。

### 压力测试

压力测试方法测试系统在一定饱和状态下，例如CPU、内存在饱和使用情况下，系统能够处理的会话能力，以及系统是否稳定会出现错误。尝试探测应用或者应用的支撑基础设施某些部分的极限能力。

目的：找到被测系统的容量天花板。

会一直增加负载，直到应用的部分功能不能正常工作。

### 并发测试

并发测试通过模拟用户并发访问，测试多用户并发访问同一个应用、同一个模块或者数据记录时是否存在死锁或其者他性能问题。

### 可靠性测试

目的：为了发现那些只有长时间运行才能发现的问题。典型的例子包括内存泄漏或者某些用例没有预见地执行次数的上限。

必须有合适的监控系统。

发现问题常常表现为响应时间缓慢增加，或某时刻系统突然变得不可用。

需要将问题发生时刻的负载数据和监控数据关联后进行分析。

### 性能冒烟测试

性能冒烟测试：只测试那些被代码变更影响到的性能测试用例。

### 隔离测试

目的：定位和排查问题。

通常会对已经定位产生性能问题的用例多次重复执行。

## 二、性能指标

可用性。（可用时长）

响应时间。（用户发起请求到应用响应完全到达用户客户端所消耗的时间）

吞吐率。（某些面向应用的事件的发生速率）

资源利用率。（对某种资源理论容量的使用百分比）

### 思考时间

思考时间指的是用户在与应用进行交互时必然存在的延迟和暂停时间。

### 响应时间

从用户发起请求到应用响应完全到达用户客户端所消耗的时间。整个过程分三个部分：**呈现时间，数据传输时间和系统处理时间。**

一般测试工具都屏蔽响应的呈现过程，只是模拟多用户并发请求，计算用户得到响应的时间，不会将服务器的每个响应做客户端渲染呈现。

对于数据传输的问题，这也是我要强调的性能测试要在局域网中进行，在局域网中一般不会受到数据带宽的限制。所以，可以对数据的传输时间忽略不计。

### 吞吐率

单位时间内网络上传输的数据量，也可以指单位时间内处理客户请求数量。它是衡量网络性能的重要指标，通常情况下，吞吐率用“字节数/秒”来衡量，当然，你可以用“请求数/秒”和“页面数/秒”来衡量。

### TPS (Transaction Per second)

每秒钟系统能够处理事务或交易的数量，它是衡量系统处理能力的重要指标。

**点击率（Hit Per Second）**

点击率可以看做是TPS的一种特定情况。点击率更能体现用户端对服务器的压力。TPS更能体现服务器对客户请求的处理能力

需要注意的是，这里的点击不能简单的看作鼠标的一次“单击”操作，也许一次“单击”操作中，客户端可能向服务器发现多个HTTP请求。

### 并发用户数

虚拟并发用户数：从性能测试工具角度来看的活跃虚拟用户数。

应用并发用户数：活跃的虚拟用户，活跃指的是已经登录或者正在访问被测系统的虚拟用户。

虚拟并发用户数=应用并发用户数+退出用户数

### 服务器资源利用率

CPU利用率

内存使用率

I/O（磁盘和网络）

### RBI（rapid bottleneck identify）

是Empirix公司提出的快速识别系统性能瓶颈的方法。该方法基于以下事实。

    1. 发现的80%系统的性能瓶颈都由吞吐量制约；

    2. 并发用户数和吞吐量瓶颈之间存在一定的关联；

    3. 采用吞吐量测试可以更快速定位问题。

通过不断增加并发用户数和吞吐量观察系统的性能瓶颈。然后，从网络、数据库、应用服务器和代码本身4个环节确定系统的的性能瓶颈。

## 三、难点

性能测试的难点？

* 需求分析
* 场景设计
* 性能诊断调优
* 环境搭建和模拟

### 如何衡量可用性

可用性通常采用应用对于终端用户的可用时长来衡量。

## 四、性能测试流程

1. 获取非功能需求，确定性能目标
2. 构建性能测试环境
3. 编写性能测试脚本
4. 构建性能测试场景
5. 执行性能测试和分析
6. 测试结果分析和报告

### 问清性能需求

发布时间2017年8月11日[**虫师**](http://www.cnblogs.com/fnng/)

假如你一名新进公司的测试小菜鸟，在一次的早会中，领导突然分配给你了一个性能测试任务。我想此时的你是一脸的大写？？，怎么办？怎么办？ 这里会提供一个通用的测试思路，照着这个思路做，你就会顺利的完成性能测试。

**问清性能测试需求**

首先，要问本次性能测试的需求是什么，或者性能测试的目的是什么？ 我把性能测试按目的分以下几种。

**1）新系统能力验证**

比如，你们刚好开发了一个新系统，在上线前需要验证系统性能。这种情况比较简单；你可以有更多的自由选择测试环境、压力点和测试工具；测试策略上也比较灵活。并且如果性能测试结果没有明显的短板，也不需要进行调优。

**2）客户有明确要求**

这是一个好的结果，这说明客户对性能测试有一定的了解，知道他们需要的系统要达到一个什么样的标准。如：系统要求同时满足100用户登陆，平均每个用户登陆时间不能超过5秒。这个需求很明确，当然也不排除一些不懂装懂的用户，提一些不现实的要求。

不管怎么说，用户提要求了，这个比较容易，你可以对现系统做一次性能测试，至于，是通过优化系统还是增加硬件设备才能达到要求。就不是测试考虑的问题了。

**3）找出系统性能瓶颈**

这个需求的目的就很明确了，目的就是找出系统的性能瓶颈，进行调优或硬件扩容，所以性能测试的重点在系统的架构分析和业务分析上面。

**4）稳定性验证（强度测试）**

稳定性是系统的一个重要指标，因为系统一旦上线，就有可能会长期处在用户的访问状态，可能以前没发现的一些问题就会暴漏出来。比较典型的就是内存溢出，这种需求在测试策略上就要考虑性能测试的运行时长。

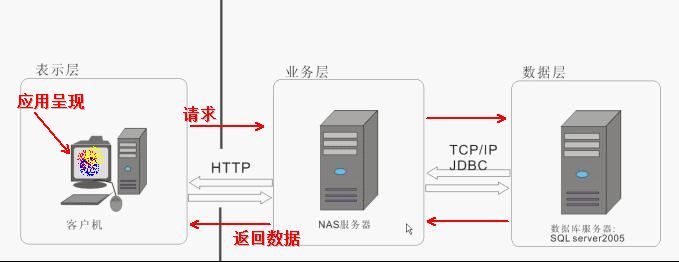
**注意：** 当拿到需求时一定要问测试的目的，一方面会显得你很专业；另一方面，我们通过测试目的可以知道后续性能测试工作的重点在哪儿？最主要的是，还可以揣摩出领导对这次测试的重视程度。^\_^!

### 了解系统架构

**发布时间 2017年8月11日**[**虫师**](http://www.cnblogs.com/fnng/)

系统架构对于测试新手来是最难的；先来了解系统所使用的技术和框架，在环境搭建阶段，你需要了解项目的部署；在性能分析与调优阶段，更要深入这些技术的细节去分析。

下面看一个基本的系统架构：



**表示层**

表示层（浏览器）通过前端技术（HTML5/JavaScript/CSS3）将系统功能和数据展示给用户，并与用户实现交互。通过TCP/HTTP协议与业务层系统通信，向应用层系统发送请求报文，并接收应用层系统返回的响应报文。

**业务逻辑层**

业务逻辑层作为中间层实现核心业务逻辑服务。

应用服务器主要运行中间件系统，中间件系统系统作为一个容器来运行各种应用软件系统。前台发来的请求报文通过中间件传递给应用程序，应用程序在处理的过程中调用数据层的数据服务器，数据服务器将查询的数据返回给应用程序，应用软件处理完成后通过中间件系统返回给客户端。

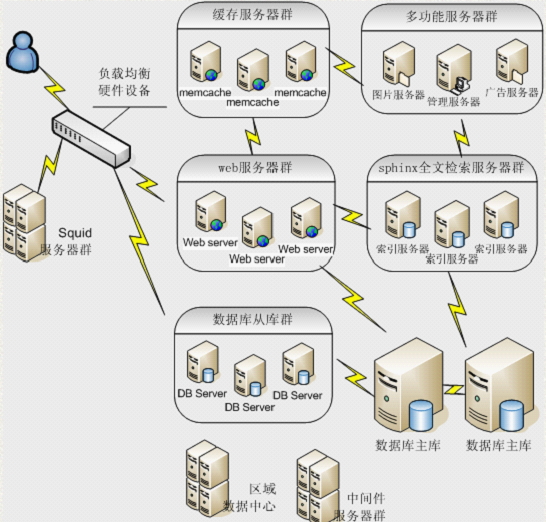
在大型的系统中，可以对应用系统进行拆分，比如拆分成交易服务，查询服务；或者通过负载均衡技术，来分散客户端发来的请求，使其能承受更大的用户访问量。

**数据层**

数据层运行在数据库主机上，负责整个系统中数据信息的存储。运行数据库服务程序，查询通过JDBC与应用程序进行通信，主要用于存储数据与提供数据查询等服务。

数据库集群技术就是对大型系统应用非常广泛的一种解决方案。

前面介绍了一般系统的架构，那么一个大型的系统在设计中使用了哪些手段或技术来提高系统的性能呢？下面通过一张图来了解一下。



常用的系统架构:

· Linux + Apache + PHP + MySQL

· Linux + Nginx + Redis +  PHP + MySQL

· Linux + Apache + Tomcat + Java+ Oracle

· Windows Server 2003/2008 + IIS + C#/ASP.NET + 数据库

· Window Server 2003/2008 + tomcat + MySql/Oracle/ + Java

· Linux + Python + uwsgi + Nginx + MySQL

### 分析测试点

**发布时间 2017年8月11日**[**虫师**](http://www.cnblogs.com/fnng/)

当你了解你的系统架构之后，接下来可以好好的分析一下性能测试点。因为这关系到你测试工具的选型。

**性能测试点的选取**

\*　发生频率非常高的（例如：某邮箱核心业务系统中的登录、收发邮件等业务，它们在每天的业务总量中占到90%以上）

\*　关键程度非常高的（产品经理认为绝对不能出现问题的，如登录等）

\*　资源占用非常严重的（导致磁盘I/O非常大的，例如某个业务进行结果提交时需要向数十个表存取数据，或者一个查询提交请求时会检索出大量的数据记录）

**对性能需求点的描述**

**准确**

如\*\*系统必须在不超过 10 秒的响应时间内，处理 20 起登录任务。再如发邮件时间最大不超过5秒以及平均时间在2秒以内。

**一致**

用户和性能测试工程师对有关术语的理解要一致，如:并发用户数、在线用户数、注册用户数:

**特定**

性能测试的需求一定是有条件的。

检查系统后台关键业务数据10G、操作数据量为20K，1500 个用户、500 个并发用户运行的负载下，连续运行12小时过程中，业务操作是否满足性能需求。

**一般性能需求描述**

1、Web首页打开速度5s以下，Web登陆速度 15s以下。

2、邮件服务支持50万个在线用户

3、计费话单成功率达到99.999%以上。

4、在100个并发用户的高峰期，邮箱的基本功能，处理能力至少达到10QPS(TPS). QPS(TPS)–每秒钟请求/事物 数量

5、系统能在高于实际系统运行压力1倍的情况下，稳定的运行12小时。

6、这个系统能否支撑200万的VU（每天登录系统的人次）VU–Virtual user(虚拟用户)

### 测试工具选型

**发布时间 2017年8月11日**[**虫师**](http://www.cnblogs.com/fnng/)

根据对系统性能测试点的分析，我们选择合适的工作。不是所有的性能需求都要用LoadRunner或Jmeter来进行的。只要能满足需求就可以了，我个人倾向于轻量级的性能工具，或自己动手写代码。

**测试工具的比较：**

软件性能测试工具分为三类，商业收费工具、开源免费工具、自主开发工具。

表：

| **–** | **商业工具** | **开源工具** | **自研工具** |
| --- | --- | --- | --- |
| 优点 | 功能强大且简单易用；提供丰富的功能，支持更多协议和并发用户数，丰富的计数器和测试结果；自动生成脚本，学习成本较低；很强的稳定性和可靠性。 | 轻量灵活容易扩展；可免费下载使用，可查看工具源代码，根须需求容易扩展。 | 贴近需求，开发出更适合需求的工具，较开源工具学习成本低。可形成特有的工具体系。 |
| 缺点 | 依赖工具本身提供的功能，较难扩展。 工具体积大，购买成本高。 | 学习成本高，不能完全满足性能测试需求，需要功能扩展。功能相对比较简陋。 | 工具开发成本高，需要专业的工具开发人员或团队，工具成本也会存在不确定。 |

通过上面的比较分析，很难判别对某类工具的取舍，各自有其明显的优势，当然也有明显的缺点，如何更好选择工具，这里就要结合实际性能的需求了。

### 测试计划

**发布时间 2017年8月11日**[**虫师**](http://www.cnblogs.com/fnng/)

#### 一．简介

  简介部分就不用过多描述了，无非项目的背景，进行此次性能测试的原因，以及性能测试覆盖的范围等等，几乎所有项目文档都在开端对项目进行简单的阐述。

#### 二．性能测试需求

寻找的被测试对象和压力点。要测试的对象不是凭空想象出来，而是经过分析与系统数据收集得到。以下取几个典型的压力点。

**登录：**对于一般的系统来说，登录是用户操作系统的前提，如果用户根本就登录不了，那么其它功能将毫无用处。例如网游戏，开新服的时候，玩家挤破了脑袋只为登录。

**查询：**查询一般比较消耗系统和数据库资源。搜索引擎的查询功能就是典型，如果你在输入框内输入内容，很久就得不到结果。我想被称为“互联网入口”的搜索引擎就不会存在。

**交易：**对于一些电子商务系统来说，交易过程的性能要求是很高的，如果交易过程消耗用户很长时间的话，用户可能会转投其它平台。当然，除了交易速度外，对交易的成功率要求也是非常高的。不然，造成的损失也是不可估量的。

被测的系统应该是最重要的最基本的功能，也是用户使用最频繁的功能。

#### 三．测试环境

 这里的测试环境主要指的软件硬件环境和网络环境。

笔者认为性能测试最好在一个独立的环境内进行，这样不会受到外界的干扰，能够保证测试的数据是独立有效的。如果现你对某个已经上线的网站进行压力测试，那么你得到的数据不是独立的，因为你在做压力测试的时候，其它散户也在访问系统。

**软件环境：**

这里的软件环境主要指项目运行的环境，比如采用什么样的操作系统、中间件、和数据库。

**硬件环境：**

这里的硬件环境除了主要包括主机内部部件，cpu、内存、磁盘以及主板、网卡等，传输介质和路由器也应该考虑在内，

**网络环境：**

网络环境除了考虑测试机与被系统服务器在一个局域网中进行，还应该保证这个网络的独立性。如果在在性能测试的过程中，其它机子也在消耗着路由器资源。那么路由器也会影响到数据库的传输速度。

#### 四．数据准备

  在很多时候，我们是要准备测试数据的，例如系统不允许相同用户的重复登录，那么必须要生成合法的用户数据。有时要对系统进行查询测试，只有在系统有一定数据量进才能验证出系统的真实性能。一个数据库中有两条数据和有两千万条数据，同相一条查询操作，对系统造成的压力是完全不一样的。

系统所需数据的分析可以参考以下方式：

历史数据分析有助于数据量级的确定。从历史数据入手，找出高峰期数据量。

从其他相似或者相同系统入手，进行数据分析，找出高峰期数据量。

无历史或者相关系统可以参考的时候，就要对系统的性能数据进行估算，包含系统容量，并发数等数据，估算以后给相关人员进行评审或者修订以后，按照大家同意的性能指标进行测试。

…………

测试数据最好和真实数据相同，如果能够获得真实系统运行3个月的数据，我们就可以在此基础上进行性能测试。

关于数据的生成，我们可以通过工具完成，如数据库数据生成工具，大小文件生成工具等。

#### 五．测试工具

  前面已经介绍如何分析需求，需求确定下来之后，我们可以考虑引入什么样的工具适合性能需求。

当然，在引入工具的时候除了考虑可以是否满足需求，还应该考虑工具的成本，这不单指工具的购买成本，还有测试人员对工具的学习成本。

关于测试工具的选择，后面会单独有一章节介绍，这里就不细说了。

如果你选择的性能测试工具不是足够的强大的话，你可能还需要其它的辅助的工具。如果Jmeter利用BadBoy来录制脚本，更能提高脚本开发效率。在压力测试的过程中也可能需要性能计数器来记录软硬件的性能。如监控服务器CPU、内存的计数器，记录中间件日志的监控中工具，监控数据库性能的监控工具等。

#### 六．测试策略

  对于一个特定的业务系统，用户一般会分散在一天的各个时间段进行访问。在不同的时间段中，用户使用业务系统的频率不同，而系统的繁忙程度不同。在一些特定的条件下，可能出现短时间内用户集中访问某个业务系统的情况。例如对于公文处理子系统而言，可能就存在短时间内大量用户查看并办理某条公文的情况。 在进行性能测试时，应当使用“考虑最坏情况的原则”。也就是应当在用户使用业务系统最频繁、对系统造成最大压力的情况下对系统的功能进行测试，判断各功能和页面是否能够满足性能的要求，系统的响应时间是否过长。

另一方面，系统性能的验证必须做到“覆盖全面”。虽然系统中各个功能的使用频率并不相同，一些功能的使用频率相对于其他功能来说比较低，但是在进行性能测试和优化时，不能忽略这些功能，编制测试用例时也不能仅仅选择最常用功能。例如可能所有的用户都会访问我的通知列表，但是一般只有5%的用户会使用通过系统设置模块查找某个用户的信息；但是在测试时，我们并不能因为查看用户信息功能的使用频率相对较少，而忽略掉这项功能的测试。所以，这里进行系统性能测试时，对于不同业务，用户的访问比例应该做一个合理分配。

在测试策略上，我们还应该考虑，同一个系统在不同硬件环境下的性能表现。从而让系统满足需求的情况下，硬件配置也能达到一个最佳的状态。过份的增加硬件来满足需求也是一种浪费。再说增加硬件设备不是能解决所有性能问题的。

#### 七．人力与时间安排

  最后一条，就是要根据项目的进度要求以及规模，来进行人力与时间的安排。对于大型的性能测试，项目前期的需求调研，环境的部署，工具的选购或开发，人员对测试工具的学习与使用，性能测试的后进行，后期数据的分析与调优。都需要人员安排的。有可以需要专业的，系统工程师、数据库工程师、软件开发工程师、网络工程师以及性能测试工程师的共同参与配合完成。不是一个性能测试人员就可以全部搞定的。

### 测试环境搭建

**发布时间 2017年8月11日**[**虫师**](http://www.cnblogs.com/fnng/)

#### 性能测试环境与功能测试环境的区别

那么性能测试环境与功能测试环境有所不同，对于一些企业为了节约资源，进行功能测试的测试环境，一台服务器可以运行多个系统，通过技术手段可以使系统之间是不会相互影响的（以前公司就是一台服务器上跑多个tomcat）。 性能测试是要对整个系统运行的软件硬件环境进行测试的，如果某环境下运行多个系统，就很难判断其中的某个环境对资源的占用情况。

#### 保证测试环境与生产环境的一致性

**1、硬件环境，包括服务器环境、与网络环境**

如服务器的型号以及是否和其它应用程序共享此服务器，是否在集群环境下，是否通过BIGIP进行负载均衡，客户使用的硬件配置情况，使用的交换机型号，网络传输速率。

**2、软件环境**

\*版本一致性

包括包括操作系统、数据库、中间件的版本，被测系统的版本。

\*配置一致性

系统（操作系统/数据库/中间件/被测试系统）参数的配置一致，这些系统参数的配置有可能对系统造成巨大的影响。所以，除了保证测试环境与真实环境所使用的软件版本一致，也要关注其参数的配置是否一致。

**3、使用场景的一致性**

* 基础数据的一致性

包括预测的业务数据量，以及数据类型的分配。很简单的一个列子，一个系统的数据库只有10条数据和一条数据库里几千万条数据，我们在对其进行性能测试时，得到的性能指标可能会有非常大的差别。

为了保证每次测试环境的更加一致性，磁盘的使用情况以及磁盘的碎片情况也会或多或少的影响的性能。

* 使用模式的一致性

尽量模拟真实场景下用户的使用情况，其实，我们在做性能测试前期的需求分析，其主要目的也就是为了更真实的模拟用户的使用情况。

#### 实施策略

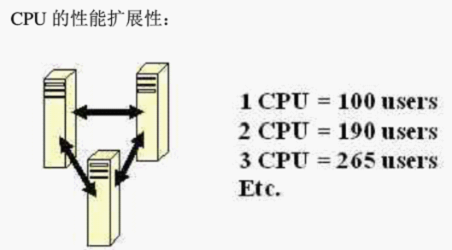
上面讲测试环境与生产环境保持一致所需要注意的内容。在实际的测试中，处于成本考虑，在很多情况下，我们很难申请到足够的且一致的资源，所以，很难搭建出与生产环境完全一致的一个测试环境。

我们一般通过两种策略来搭建性能测试环境（预估方式均有误差）

**1、通过建模的方式实现低端硬件对高端硬件的模拟**

通过配置测试来计算不同配置下的硬件性能和系统处理能力的关系，从而推导出满足系统性能的真实配置情况，这种模拟需要精确的建模，模型的采样点越多，那么得到的结果越精确，从而将在低端配置下的性能指标通过该模型转化为高端配置下的最终预计性能指标。

例如：搭建一个低端环境，首先需要对这个环境的CPU和内存进行单独的性能基准测试，同过在不同的配置的性能测试，得到一个基准信息列表，当然，在进行这个性能测试的过程中，我们要确定硬件是系统的瓶颈。如果只用一个CUP，在性能测试过程中，其使用率很低，但得到的性能数据都非常底，这起码说明CUP不是系统的瓶颈，这种情况下就无法得到想要的基准值。



如上图，在一颗CPU情况下，运行100个用户且CUP使用率接近饱和（100%）。在增加至两颗CUP的情况下，可以运行190个用户且UPU使用率接近饱和（100%），以此做记录，那么我们就可以推算出运行800个用户需要多少颗CUP。 如果你在实际应用中使用的CUP型号及其频率并非完全一样，这个时候可以使用EVEREST工具计算每种CUP的得分，对其性能进行评估。 内存也可以使用此方法进行测试推导，这里需要我们多进行试验，对硬件的性能以及对整个项目的结构都要做深入的了解，以便尽量减少误差。

**2、通过集群的方式计算**

对于较大的系统来说，单台服务器的处理能力是有限的，通常都会采用集群的方式来进行负载均衡，完成对海量请求的处理。虽然无法获得整体集群的测试环境，但是可以对集群上的一个节点进行性能测试，得出该节点的处理能力，再计算每增加一个节点的性能损失，同样也可以能过建模的方式得到大型负载均衡情况下的预计性能指标。

例如:首先在单台服务器上获得具体的性能指标，每台服务器能够承受500用户并发，平均TPS为60，响应时间为2秒，接着，添加负载均衡策略，再次测试负载策略下的数据损耗。得出数据后添加1台负载均衡服务器，测试在两台服务器下每台服务器的性能指标，以此类推，可以得到下表：

表：

| **负载服务器个数** | **并发用户数** | **平均TPS** | **响应时间** |
| --- | --- | --- | --- |
| 1 | 500 | 60 | 2 |
| 1 | 490 | 58 | 2 |
| 2 | 490 | 58 | 2 |
| 3 | 480 | 57 | 2 |

随着负载均衡服务器的添加，平均每台服务器的处理能力会逐渐稳定，从而了解在什么情况下需要多少台负载均衡服务器。

### 测试执行

**发布时间 2017年8月11日**[**虫师**](http://www.cnblogs.com/fnng/)

参考[测试计划](http://www.testclass.net/performance/test-plan/)，大概执行过程如下：

1、准备测试数据

2、使用测试工具模拟测试点，回放OK

3、根据测试策略，使用不同的虚拟用户和测试组合 运行测试。

4、监控系统CPU、内存、中间件，数据库的性能，收集数据。

5、重复3、4。

### 性能调优

**发布时间 2017年8月11日**[**虫师**](http://www.cnblogs.com/fnng/)

#### 一般系统的瓶颈

性能测试调优需要先发现瓶颈，那么系统一般会存在哪些瓶颈：

**硬件上的性能瓶颈：**

一般指的是CPU、内存、磁盘I/O 方面的问题，分为服务器硬件瓶颈、网络瓶颈（对局域网可以不考虑）、服务器操作系统瓶颈（参数配置）、中间件瓶颈（参数配置、数据库、web服务器等）、应用瓶颈（SQL 语句、数据库设计、业务逻辑、算法等）。

**应用软件上的性能瓶颈：**

一般指的是应用服务器、web 服务器等应用软件，还包括数据库系统。

例如：中间件weblogic 平台上配置的JDBC连接池的参数设置不合理，造成的瓶颈。

**应用程序上的性能瓶颈：**

一般指的是开发人员新开发出来的应用程序。

例如，程序架构规划不合理，程序本身设计有问题（串行处理、请求的处理线程不够），造成系统在大量用户访问时性能低下而造成的瓶颈。

**操作系统上的性能瓶颈：**

一般指的是windows、UNIX、Linux等操作系统。

例如，在进行性能测试，出现物理内存不足时，虚拟内存设置也不合理，虚拟内存的交换效率就会大大降低，从而导致行为的响应时间大大增加，这时认为操作系统上出现性能瓶颈。

**网络设备上的性能瓶颈：**

一般指的是防火墙、动态负载均衡器、交换机等设备。

例如，在动态负载均衡器上设置了动态分发负载的机制，当发现某个应用服务器上的硬件资源已经到达极限时，动态负载均衡器将后续的交易请求发送到其他负载较轻的应用服务器上。在测试时发现，动态负载均衡器没有起到相应的作用，这时可以认为网络瓶颈。   性能测试出现的原因及其定位十分复杂，这里只是简单介绍常见的几种瓶颈类型和特征，而性能测试所需要做的就是根据各种情况因素综合考虑，然后协助开发人员\DBA\运维人员一起定位性能瓶颈。

#### 性能调优的一般步骤

**步骤一：确定问题**

应用程序代码：在通常情况下，很多程序的性能问题都是写出来的，因此对于发现瓶颈的模块，应该首先检查一下代码。

数据库配置：经常引起整个系统运行缓慢，一些诸如oracle 的大型数据库都是需要DBA进行正确的参数调整才能投产的。

操作系统配置：不合理就可能引起系统瓶颈。

硬件设置：硬盘速度、内存大小等都是容易引起瓶颈的原因，因此这些都是分析的重点。

网络：网络负载过重导致网络冲突和网络延迟。

**步骤二：分析问题**

当确定了问题之后，我们要明确这个问题影响的是响应时间吞吐量，还是其他问题？是多数用户还是少数用户遇到了问题？如果是少数用户，这几个用户与其它用户的操作有什么不用？系统资源监控的结果是否正常？CPU的使用是否到达极限？I/O 情况如何？问题是否集中在某一类模块中？ 是客户端还是服务器出现问题？ 系统硬件配置是否够用？实际负载是否超过了系统的负载能力？ 是否未对系统进行优化？

通过这些分析及一些与系统相关的问题，可以对系统瓶颈有更深入的了解，进而分析出真正的原因。

**步骤三： 确定调整目标和解决方案**

得高系统吞吐量，缩短响应时间，更好地支持并发。

**步骤四：测试解决方案**

对通过解决方案调优后的系统进行基准测试。（基准测试是指通过设计科学的测试方法、测试工具和测试系统，实现对一类测试对象的某项性能指标进行定量的和可对比的测试）

**步骤五：分析调优结果**

系统调优是否达到或者超出了预定目标？系统是整体性能得到了改善，还是以系统某部分性能来解决其他问题。调优是否可以结束了。

最后，如果达到了预期目标，调优工作就基本可以结束了。

## ****五、性能测试设计思想****

**性能测试方案设计思路总结**

**博客：**[**http://blog.sina.com.cn/ishouke**](http://blog.sina.com.cn/ishouke)

[**一、 需求分析**](http://blog.sina.com.cn/s/blog_13cc013b50102wrh6.html#_Toc11042)

[**二、 系统分析**](http://blog.sina.com.cn/s/blog_13cc013b50102wrh6.html#_Toc16021)

[**三、 业务分析**](http://blog.sina.com.cn/s/blog_13cc013b50102wrh6.html#_Toc3523)

[**四、 用例设计**](http://blog.sina.com.cn/s/blog_13cc013b50102wrh6.html#_Toc1049)

[**五、 测试策略**](http://blog.sina.com.cn/s/blog_13cc013b50102wrh6.html#_Toc13350)

[**六、 工具选取**](http://blog.sina.com.cn/s/blog_13cc013b50102wrh6.html#_Toc28352)

[**七、 网络分析**](http://blog.sina.com.cn/s/blog_13cc013b50102wrh6.html#_Toc4834)

[**八、 硬件配置**](http://blog.sina.com.cn/s/blog_13cc013b50102wrh6.html#_Toc19990)

[**九、 性能监控**](http://blog.sina.com.cn/s/blog_13cc013b50102wrh6.html#_Toc31581)

[**十、 实施测试**](http://blog.sina.com.cn/s/blog_13cc013b50102wrh6.html#_Toc5583)

[**十一、 结果分析**](http://blog.sina.com.cn/s/blog_13cc013b50102wrh6.html#_Toc25300)

### 一、需求分析

#### 1.   测试目的

为什么测？目的在于测试软件系统相关性能能否满足业务需求。通常分以下两种情况：

1）新项目上线

2）老项目优化

如果是老项目优化，可考虑是否存有历史测试方案，如果有可以参考，或许可以省事很多。

#### 2.   测试对象

**要测啥？**

测试对象可以归结为“业务功能”。测试前，需要了解我们需要测试的业务功能（不深入细节）有哪些，比如“购买商品”、“寄送快递”。

**有没有必要测？**

需求来源哪里？，有没有数据支撑测试这个需求的必要性？

通常，可以从以下几个方面考虑：

1）是否核心功能，是否要求严格的质量

2）是否常用、高频使用的功能

3）可能占用系统较多资源的功能

4）使用人数多还是少

5）在线人数多还是少

#### 3.   拆分对象

先从业务上来分，实现这个完整的功能包含哪些流程、环节

举例：购买商品

登录->搜索商品->提交订单->支付订单->退出

#### 4.   指标分析

分析性能需求指标（如“支持300人并发登录”【并发数一般为在最大线人数的20%】）是否合理

有必要测试这个需求，考虑需求指标是否合理？有没有数据支撑？

通常，支撑数据可以从以下方面考虑：

1）采样时间段内系统使用人数

2）采样时间段内系统在线人数

3）采样时间段内系统(页面)访问量

4）采样时间段内请求数

....

常用分析思路：

**1）2/8法则**

2/8法则：80%的业务量在20%的时间里完成。这里，业务量泛指访问量，请求数，数据量等

**2）正态分布**

**3）按比例倍增**

**4）响应时间2-5-8原则**

就是说，一般情况下，当用户能够在2秒以内得到响应时，会感觉系统的响应很快；当用户在2-5秒之间得到响应时，会赶紧系统的响应速度还可以；当用户在5-8秒以内得到响应时，会赶紧系统的速度很慢，但是还可以接受；而当用户在超过8秒后仍然无法得到响应时，会感觉系统糟糕透了，或者认为系统已经失去响应。

注意：这个要根据实际情况，有些情况下时间长点也是可以接受的，好比12306

**举例：**

某公司后台监控，根据一段时间的采样数据，分析得出日高峰时段(11:00-14:00)用户下单请求数平均为1000，峰值为1500，根据这个计算并发请求数

时段：3个小时 -> 3 x 60 x 60 = 1080s

业务量：1500

吞吐量：1500 \* 80% / (1080 \* 20%) = 5.56请求数/s

假设用户下单遵循正态分布，那么并发请求数峰值会肯定大于上述估算的吞吐量

**注意:**

1、2/8原则计算的结果并非在线并发用户数，是系统要达到的处理能力（吞吐量）

2、如果要求更高系统性能，根据实际情况，也可以考虑1/9原则或其它更严格的算法

3、以上估值只是大致的估算，不是精确值

举例：

想了下，暂时没想到啥好的例子，大致就说一些涉及到数据量的性能测试，比如报表统计，或者是大数据挖掘，查询等，怎么去估算数据量？

数据生命周期：

一般来说，数据都是有一定的生命周期的，时间的选取需要结合数据周期考虑。这里假设3年后系统性能仍然需要满足业务需求。

数据增长率：

如果是老项目，可以考虑对应功能主表历史数据存放情况

这里假设按年统计，比如第一年 10000，第二年 15000，第三年 20000，第四年25000，那么我们得出，以第一年为基准，数据增长率分别为 0.5，1，1.5，每年在上一年的基础上，以5000的速度增长

预估3年后，数据增长率为 3，需要测试数据量为 （1+3）x 10000 = 40000

注意：

1、实际数据一般是没上面举例那么规律的，只能大致估算数据增长率。

2、一些大数据量的性能测试除了和数据量相关，还涉及到数据分布等，比如查询，构造数据时需要结合实际，尽量贴近实际。

3、不同业务模块，涉及表不一样，数据量要求也是不一样的，需要有区别的对待。

如果是新项目，那就比较不确定了，除非能收集相关数据。

### 二、系统分析

结合需求分析中第3点，分析系统架构。从功能实现上来看，怎么实现这个完整功能的。通常这些业务功能操作都对应着一个或多个请求(可能能是不同类型的请求，比如http, mysql等)，我们要做的是找出这些：

1）请求顺序、请求之间相互调用关系

2）数据流向，数据是怎么走的，经过哪些组件、服务器等

3）预测可能存在性能瓶颈的环节（组件、服务器等）

4）明确应用类型 IO型，还是CPU消耗性、内存消耗型-> 弄清楚重点监控对象

5）关注应用是否采用多进程、多线程架构-> 多线程容易造成线程死锁、数据库死锁，数据不一致等

6）是否使用集群/是否使用负载均衡

了解测试环境部署和生产环境部署差异，是否按1:1的比例部署

通常建议测试时先不考虑集群，采用单机测试，测试通过后再考虑使用集群，这样有个比较，比较能说明问题

参考阅读“浅谈web网站架构演变过程 ”：<http://blog.csdn.net/qiaqia609/article/details/50809383>

### 三、业务分析

1）明确要测试的功能业务中，功能业务占比，重要程度。目的在于

<1>明确重点测试对象，安排测试优先级

<2>建模，混合场景中，虚拟用户资源分配，针对不同业务功能施加不同的负载。

2）明确下“需求分析-指标分析”中相关业务功能所需基础数据及数据量问题，因为那块需求分析时可能只是大致估算下，评估指标是否合理，需要认真再分析下

### 四、用例设计

#### 1．用例设计

通常是基于场景的测试用例设计

<1> 单业务功能场景

运行测试期间，所有虚拟用户只执行同一种业务功能某个环节、操作

<2> 混合业务功能场景

运行测试期间，部分虚拟用户执行某种业务的某个环节操作，部分虚拟用户执行该业务功能的其它环节

或者

运行测试期间，部分虚拟用户执行某种业务功能，部分虚拟用户执行其它业务功能

注：这里用例没说到多少用户去跑，跑多久等，这里只是把他当作相同场景用例下的的一组组测试数据了。

#### 2．事务定义

根据用例合理的定义事务，方便分析耗时（特别是混合业务功能场景测试），进而方便分析瓶颈。

比如，购买商品，我们可以把下订单定义为一个事务，把支付也定义为一个事务。

#### 3．场景监控对象

针对每条用例，结合“系统分析”第4）点，明确可能的压力点（比如数据库、WEB服务器），需要监控的对象，比如tps，耗时，CPU，内存，I/O等

### 五、测试策略

1）先进行混合业务功能场景的测试，在考虑进行测试单业务功能场景的测试

2）负载测试 -> 压力测试-> 稳定性测试-> 强度测试

注：如果测试稳定性，时间建议至少8小时；

3）逐步加压

比如开始前5分钟，20个用户，然后每隔5分钟，增加20个用户。

好处：不仅比较真实的模拟现实环境，而且在性能指标比较模糊，且不知道服务器处理能力的情况下，可以帮我们确定一个大致基准，因为通常情况下，随着用户数的不断增加，服务器压力也会随着增加，如果服务器不够强大，那么就会出现不能及时处理请求、处理请求失败的情况下，对应的运行结果图形中，运行曲线也会出现对应的形态，比如从原本程一条稳定直线的情况，到突然极限下降、开始上下波动等，通过分析我们就能得出服务器大致处理能力，供后续测试参考。

4）单点并发

比如使用集合点，单独针对某个环节的并发测试，通常是针对某个环节的性能调优时使用。

常识：

a) 负载测试

保证系统能正常运行(通常是满足某些系统性能指标)的前提下，让被测对象承担不同的工作量，以评估被测对象的最大处理能力及存在缺陷而进行的测试

b) 压力测试

不保证系统能否正常运行的前提下，让被测对象承担不同工作量，以评估被测对象能提供的最大处理能力及存在缺陷而进行的测试

c) 稳定性测试

测试系统的长期稳定运行的能力。同疲劳强度测试的区别是，稳定性测试的压力强度较小，一般趋向于客户现场日常状态下的压力强度，当然在通过时间不能保证稳定性的状态下，需要加大压力强度来测试，此时的压力强度则会高于正常值。

d) 强度测试

通常模拟系统在较差、异常资源配置下运行，如人为降低系统工作环境所需要的资源,如网络带宽,系统内存,数据锁等等,以评估被测对象在资源不足的情况下的工作状态

注：疲劳强度测试是一类特殊的强度测试，主要测试系统长时间运行后的性能表现，例如7x24小时的压力测试。

### 六、工具选取

#### 1．协议分析

一般性能测试工具都是基于协议开发的，所以先要明确应用使用的协议

#### 2．工具选取

1）类型

开源工具、收费工具、自研工具

2）分析工具

<1> 理解工具实现原理

<2> 采用用异步还是同步

常识：

1．同步请求：发出一个调用请求，在没有得到结果之前，该调用就不返回。

2.异步请求：发出一个调用请求，在没有得到请求结果之前，该调用可立即返回。该调用请求的处理者在处理完成后通过状态、通知和回调等来通知调用者。

<3> 使用长连接还是短连接

### 七、 软件配置

#### 1．操作系统

内核版本、32 or 64位?

#### 2．应用版本

应用版本要和线上保持一致，特别是中间件、组件等的版本，因为不同版本，其性能可能不一样

#### 3．参数配置

<1> 负载均衡、反向代理参数配置

<2> Web服务器参数配置

<3> 数据库服务器参数配置

### 八、网络分析

#### 1．网络路由

通常为了排除网络型瓶颈，通常建议在局域网下进行测试。

通常，这里我的分析思路是这样的：

<1> 检查hosts文件的配置

从终端压测机(负载生成机)开始，到请求目的服务器器，机器的hosts文件配置

通常，hosts文件位于如下：

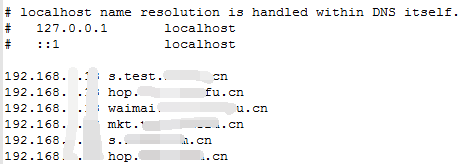
Windows：C:\Windows\System32\drivers\etc\hosts

Unix/Linux：/etc/hosts

小常识：

1、通常域名访问站点，首先要通过DNS域名服务器把网络域名（形如www.xxx.com）解析成XXX.XXX.XXX.XXX的IP地址，然后继续后续访问。

2、hosts存放了域名和ip地址的映射关系，如下

[](http://photo.blog.sina.com.cn/showpic.html#blogid=13cc013b50102wrh6&url=http://album.sina.com.cn/pic/005NDN77zy78angPYWz27)

使用hosts可以加快域名解析，在进行DNS请求以前，系统会先检查自己的hosts文件中是否有这个地址映射关系，如果有则把域名解析为映射的IP地址，不请求网络上的DNS服务器，如果没有再向已知的DNS 服务器提出域名解析。也就是说hosts的请求级别比DNS高，可加快域名解析。

<2> 检查DNS配置

不同DNS，其速度和准确率是不一样的，比如114.114.114.114速度远比8.8.8.8快，如果有用到DNS（特别是压测机），需要考虑下是否适当

<3> 确保路由正确设置

#### 2．网络带宽

如果没条件在局域网下测试，可能需要估算所需大致带宽。

如果测试时是基于UI层操作的操作，那么得估算页面平均大小，这个可以通过浏览器自带工具查看打开单个页面服务器返回的请求数据大小。如果是测试时是基于接口层的请求测试，可以通过工具查看服务器响应数据大小。

然后根据采集的页面PV峰值、请求数峰值进行计算。

假设在 PV峰值、请求数峰值 = 1000，峰值时段：8:00 - 12:00，平均页面、请求大小 200k

带宽 = 1000 x 80% / (20% x 4 x 3600s) x 200KB x /1024 x 8bit ,单位MBps

估计带宽 = tps\*请求页面大小

注意： 这里涉及到浏览器缓存等因素，估值可能不准，大致估算。

### 九、硬件配置

1) CPU

型号，频率，核数

2) 内存

3) 磁盘

不同磁盘类型，读写速率不一样

4) 网卡

不同网卡，其传输速率也不一样

注意：硬件配置最好和生产环境的配置保持一致

### 十、性能监控

略

注意：

1） 这里监控不仅仅是服务器自身性能指标监控，如cpu，还包括事务耗时监控等

2） 需要记录测试前各个性能指标数据，方便后续测试对比

### 十一、实施测试

略

### 十二、结果分析

如果是性能调优，还需同上一个版本的性能测试结果对比

略

pdf版下载：[性能测试方案设计思路总结.pdf](http://pan.baidu.com/s/1c25lgh6)

## 六、性能测试工具：Locust性能测试框架

### 1. Locust简介

Locust是使用Python语言编写实现的开源[性能](http://lovesoo.org/tag/%e6%80%a7%e8%83%bd)[测试](http://lovesoo.org/tag/%e6%b5%8b%e8%af%95)工具，简洁、轻量、高效，并发机制基于gevent协程，可以实现单机模拟生成较高的并发压力。

官网：<https://locust.io/>

主要特点如下：

1) 使用普通的Python脚本用户[测试](http://lovesoo.org/tag/%e6%b5%8b%e8%af%95)场景

2) 分布式和可扩展，支持成千上万的用户

3) 基于Web的用户界面，用户可以实时监控脚本运行状态

4) 几乎可以测试任何系统，除了web [http](http://lovesoo.org/tag/http)接口外，还可[自定义](http://lovesoo.org/tag/%e8%87%aa%e5%ae%9a%e4%b9%89)clients测试其他类型系统

### 2. 安装

使用pip或easy\_install，可以方便安装Locust

|  |  |
| --- | --- |
|  | pip install locustio |

安装完成后，可以在shell或cmd中运行[locust](http://lovesoo.org/tag/locust)命令，如查看可用的选项：

|  |  |
| --- | --- |
|  | locust --help |

**Locust主要由下面的几个库构成：**

1) gevent

gevent是一种基于协程的Python网络库，它用到Greenlet提供的，封装了libevent事件循环的高层同步API。

2) flask

Python编写的轻量级Web应用框架。

3) requests

Python Http库

4) msgpack-python

MessagePack是一种快速、紧凑的二进制序列化格式，适用于类似JSON的数据格式。msgpack-python主要提供MessagePack数据序列化及反序列化的方法。

5) six

Python2和3兼容库，用来封装Python2和Python3之间的差异性

6) pyzmq

pyzmq是zeromq(一种通信队列)的Python绑定,主要用来实现Locust的分布式模式运行

### 3. 快速入门

#### 3.1 示例

#coding:utf-8

from locust import HttpLocust, TaskSet, task

class UserBehavior(TaskSet):

    def on\_start(self):

        """ on\_start is called when a Locust start before any task is scheduled """

        self.login()

    def login(self):

        self.client.post("/login", {"username":"ellen\_key", "password":"education"})

    @task(2)

    def index(self):

        self.client.get("/")

    @task(1)

    def profile(self):

        self.client.get("/profile")

class WebsiteUser(HttpLocust):

    task\_set = UserBehavior

    host='[http://example.com](http://example.com/)'

    min\_wait = 5000

    max\_wait = 9000

上面是官方的示例demo，定义了针对[http](http://lovesoo.org/tag/http)://example.com网站的测试场景：先模拟用户登录系统，然后随机地访问index（/）和profile页面（/profile/），请求比例为2:1；并且，在测试过程中，两次请求的间隔时间为5~9秒间的随机值。

#### 3.2 运行

要运行上述[locust](http://lovesoo.org/tag/locust)脚本，如果文件名为[locust](http://lovesoo.org/tag/locust)file.py且在当前目录下，可以这样运行：

|  |  |
| --- | --- |
|  | locust |

如果locust脚本文件目录不同或名称不同，需要使用-f指定文件(--host用来指定测试主机地址)：

|  |  |
| --- | --- |
|  | locust -f locust\_files/my\_locust\_file.py --host=[http://example.com](http://example.com/) |

要运行分布在多个进程上的locust，我们需要使用--master启动主进程：

|  |  |
| --- | --- |
|  | locust -f locust\_files/my\_locust\_file.py --master --host=[http://example.com](http://example.com/) |

之后使用--slave启动任意数量的从进程：

|  |  |
| --- | --- |
|  | locust -f locust\_files/my\_locust\_file.py --slave --host=[http://example.com](http://example.com/) |

如果在多台机器上分布式运行locust，我们需要在启动从进程时指定master-host（默认为127.0.0.1）：

|  |  |
| --- | --- |
|  | locust -f locust\_files/my\_locust\_file.py --slave --master-host=192.168.0.100 --host=[http://example.com](http://example.com/) |

#### 3.3 Locust web模式

Locust默认使用该方式启动，启动后在本机打开http://localhost:8089/，可以看到Locust WEB页面，设置并发用户数及每秒请求数后即可开始[性能](http://lovesoo.org/tag/%e6%80%a7%e8%83%bd)测试。

#### 3.4 Locust no-web模式

Locust也可使用no-web模式，使用命令如下：

|  |  |
| --- | --- |
|  | locust -f locust\_files/my\_locust\_file.py --no-web --csv=locust -c10 -r2 --run-time 1h30m |

其中--no-web表示使用no-web模式，--csv表示运行结果文件名，-c 并发用户数，-r 每秒请求数，--run\_time 运行时间

### 4. locustfile详解

locustfile中测试场景均是继承自Locust和[TaskSet](http://lovesoo.org/tag/taskset)的子类，下面分别介绍Locust和[TaskSet](http://lovesoo.org/tag/taskset)两个类。

#### 4.1 Locust类

Locust类的client属性对应虚拟用户作为客户端的请求方法。在使用Locust时，需要先继承Locust类，然后在继承子类中的client属性中绑定客户端的实现类。

对于常见的HTTP(S)协议，Locust已经实现了HttpLocust类，其client属性绑定了HttpSession类，而HttpSession又继承自requests.Session。因此在测试HTTP(S)的Locust脚本中，我们可以通过client属性来使用Python requests库的所有方法，包括GET/POST/HEAD/PUT/DELETE/PATCH等，调用方式也与requests完全一致。另外，由于requests.Session的使用，因此client的方法调用之间就自动具有了状态记忆的[功能](http://lovesoo.org/tag/%e5%8a%9f%e8%83%bd)。常见的场景就是，在登录系统后可以维持登录状态的Session，从而后续HTTP请求操作都能带上登录状态。

而对于HTTP(S)以外的协议，我们同样可以使用Locust进行测试，只是需要我们自行实现客户端。在客户端的具体实现上，可通过注册事件的方式，在请求成功时触发events.request\_success，在请求失败时触发events.request\_failure即可。然后创建一个继承自Locust类的类，对其设置一个client属性并与我们实现的客户端进行绑定。主要，我们就可以像使用HttpLocust类一样，测试其它协议类型的系统了。

在Locust类中，除了client属性，还有几个属性需要关注下：

task\_set: 指向一个[TaskSet](http://lovesoo.org/tag/taskset)类，TaskSet类定义了用户的任务信息，该属性为必填；

max\_wait/min\_wait: 每个用户执行两个任务间隔时间的上下限（毫秒），具体数值在上下限中随机取值，若不指定则默认间隔时间固定为1秒；

host：被测系统的host，当在终端中启动locust时没有指定--host参数时才会用到；

weight：同时运行多个Locust类时会用到，用于控制不同类型任务的执行权重。

测试开始后，每个虚拟用户（Locust实例）的运行逻辑都会遵循如下规律：

先执行WebsiteTasks中的on\_start（只执行一次），作为初始化；

从WebsiteTasks中随机挑选（如果定义了任务间的权重关系，那么就是按照权重关系随机挑选）一个任务执行；

根据Locust类中min\_wait和max\_wait定义的间隔时间范围（如果TaskSet类中也定义了min\_wait或者max\_wait，以TaskSet中的优先），在时间范围中随机取一个值，休眠等待；

重复2~3步骤，直至测试任务终止。

#### 4.2 Taskset类

TaskSet类实现了虚拟用户所执行任务的调度算法，包括规划任务执行顺序（schedule\_task）、挑选下一个任务（execute\_next\_task）、执行任务（execute\_task）、休眠等待（wait）、中断控制（interrupt）等等。在此基础上，我们就可以在TaskSet子类中采用非常简洁的方式来描述虚拟用户的业务测试场景，对虚拟用户的所有行为（任务）进行组织和描述，并可以对不同任务的权重进行配置。

在TaskSet子类中定义任务信息时，可以采取两种方式，@task装饰器和tasks属性。

采用@task装饰器定义任务信息时，描述形式如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | from locust import TaskSet, task  class UserBehavior(TaskSet):      @task(1)      def test\_job1(self):          self.client.get('/job1')      @task(2)      def test\_job2(self):          self.client.get('/job2') |

采用tasks属性定义任务信息时，描述形式如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | from locust import TaskSet  def test\_job1(obj):      obj.client.get('/job1')  def test\_job2(obj):      obj.client.get('/job2')  class UserBehavior(TaskSet):      tasks = {test\_job1:1, test\_job2:2}      # tasks = [(test\_job1,1), (test\_job1,2)] # 两种方式等价 |

在如上两种定义任务信息的方式中，均设置了权重属性，即执行test\_job2的频率是test\_job1的两倍。

在TaskSet子类中除了定义任务信息，还有一个是经常用到的是on\_start函数。这个和LoadRunner中的vuser\_init[功能](http://lovesoo.org/tag/%e5%8a%9f%e8%83%bd)相同，在正式执行测试前执行一次，主要用于完成一些初始化的工作。例如，当测试某个搜索功能，而该搜索功能又要求必须为登录态的时候，就可以先在on\_start中进行登录操作；前面也提到，HttpLocust使用到了requests.Session，因此后续所有任务执行过程中就都具有登录状态了。

### 5. [自定义](http://lovesoo.org/tag/%e8%87%aa%e5%ae%9a%e4%b9%89)客户端测试其他系统

虽然，locust主要是为了测试HTTP而生。然而，它可以很容易地扩展到测试任何基于请求/响应的系统，只需要编写一个触发request\_success和request\_failure事件自定义客户端即可。

官网提供了详细的示例，我们简单修改下就可以用来对任意系统进行[性能](http://lovesoo.org/tag/%e6%80%a7%e8%83%bd)测试：

import time

from locust import Locust, TaskSet, events, task

import requests

class TestHttpbin(object):

    def status(self):

        try:

            r = requests.get('<http://httpbin.org/status/200>')

            status\_code = r.status\_code

            print status\_code

            assert status\_code == 200, 'Test Index Error: {0}'.format(status\_code)

        except Exception as e:

            print e

class CustomClient(object):

    def test\_custom(self):

        start\_time = time.time()

        try:

            # add your custom test function here

            TestHttpbin().status()

            name = TestHttpbin().status.\_\_name\_\_

        except Exception as e:

            total\_time = int((time.time() - start\_time) \* 1000)

            events.request\_failure.fire(request\_type="Custom", name=name, response\_time=total\_time, exception=e)

        else:

            total\_time = int((time.time() - start\_time) \* 1000)

            events.request\_success.fire(request\_type="Custom", name=name, response\_time=total\_time, response\_length=0)

class CustomLocust(Locust):

    def \_\_init\_\_(self, \*args, \*\*kwargs):

        super(CustomLocust, self).\_\_init\_\_(\*args, \*\*kwargs)

        self.client = CustomClient()

class ApiUser(CustomLocust):

    min\_wait = 100

    max\_wait = 1000

    class task\_set(TaskSet):

        @task(1)

        def test\_custom(self):

            self.client.test\_custom()

上述脚本里，我们自定义一个测试类TestHttpbin，其中status方法用来校验接口返回码；我们只需要在CustomClient类的test\_custom方法中添加自定义的测试方法TestHttpbin().status()，即可使用locust对该方法进行负载测试。

[taffy](http://lovesoo.org/tag/taffy)框架就是这样做实现了一份代码同时进行功能[自动化](http://lovesoo.org/tag/%e8%87%aa%e5%8a%a8%e5%8c%96)及性能测试。

### 6. 参考资料

1. <https://docs.locust.io/en/latest/>

2. <http://debugtalk.com/post/head-first-locust-user-guide/>