# 1.Locust介绍

An open source load testing tool.  
一个开源性能测试工具。

define user behaviour with python code, and swarm your system with millions of simultaneous users.  
使用 Python 代码来定义用户行为。用它可以模拟百万计的并发用户访问你的系统。

Locust 完全基本 Python 编程语言，采用 Pure Python 描述测试脚本，并且 HTTP 请求完全基于 Requests 库。除了 HTTP/HTTPS 协议，Locust 也可以测试其它协议的系统，只需要采用Python调用对应的库进行请求描述即可。LoadRunner 和 Jmeter 这类采用进程和线程的测试工具，都很难在单机上模拟出较高的并发压力。Locust 的并发机制摒弃了进程和线程，采用协程（gevent）的机制。协程避免了系统级资源调度，由此可以大幅提高单机的并发能力。

**注：Locust 是基于 Python 语言的一个性能测试库，如果要想使用它来做性能测试必须要先安装 Python 。**

**2.Locust安装**

通过pip命令安装，在Windows命令提示符，输入pip install locust，回车。  
检查是否安装成功，在Windows命令提示符，输入locust --help，回车。

**3.安装依赖**

打开 Locust 安装目录下的 setup.py 文件。查看安装要求：  
install\_requires=[“gevent>=1.1.2”, “flask>=0.10.1”, “requests>=2.9.1”, “msgpack-python>=0.4.2”, “six>=1.10.0”, “pyzmq==15.2.0”]  
gevent 是在 Python 中实现协程的一个第三方库。协程，又称微线程（Coroutine）。使用gevent可以获得极高的并发性能。  
flask 是 Python 的一个 Web 开发框架。  
Requests 用来做 HTTP 接口测试。  
msgpack-python 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。  
six 提供了一些简单的工具用来封装 Python2 和 Python3 之间的差异性。  
pyzmq 如果你打算运行 Locust 分布在多个进程/机器，建议你安装pyzmq。

当我们在安装 Locust 时，它会检测我们当前的 Python 环境是否已经安装了这些库，如果没有安装，它会先把这些库一一装上。并且对这些库版本有要求，有些是必须等于某版本，有些是大于某版本。我们也可以事先把这些库全部按要求装好，再安装Locust时就会快上许多。

# Locust创建性能测试

## 1.编写简单的性能测试脚本

## 创建 load\_test.py 文件，通过 Python 编写性能测试脚本。

from locust import HttpLocust, TaskSet, task

# 定义用户行为

class UserBehavior(TaskSet):  
[@task](https://github.com/task)  
def baidu\_index(self):  
self.client.get("/")

## class WebsiteUser(HttpLocust): task\_set = UserBehavior min\_wait = 3000 max\_wait = 6000

UserBehavior类继承TaskSet类，用于描述用户行为。  
baidu\_index() 方法表示一个用户为行，访问百度首页。使用[@task](https://github.com/task)装饰该方法为一个事务。client.get()用于指请求的路径“/”，因为是百度首页，所以指定为根路径。  
WebsiteUser类用于设置性能测试。  
task\_set ：指向一个定义的用户行为类。  
min\_wait ：执行事务之间用户等待时间的下界（单位：毫秒）。  
max\_wait ：执行事务之间用户等待时间的上界（单位：毫秒）。

## 2.执行性能测试

启动性能测试  
打开Windows命令提示符，输入 locust -f load\_test.py --host=https://www.baidu.com，回车。  
注：-f 指定性能测试脚本文件。  
–host 指定被测试应用的URL的地址，注意访问百度使用的HTTPS协议。

## 3.设置测试

## 通过浏览器访问：[http://localhost:8089（Locust启动网络监控器，默认为端口号为](http://localhost:8089%EF%BC%88Locust%E5%90%AF%E5%8A%A8%E7%BD%91%E7%BB%9C%E7%9B%91%E6%8E%A7%E5%99%A8%EF%BC%8C%E9%BB%98%E8%AE%A4%E4%B8%BA%E7%AB%AF%E5%8F%A3%E5%8F%B7%E4%B8%BA): 8089）

## Number of users to simulate 设置模拟用户数。 Hatch rate（users spawned/second） 每秒产生（启动）的虚拟用户数。 点击 “Start swarming” 按钮，开始运行性能测试。

性能测试参数：  
Type： 请求的类型，例如GET/POST。  
Name：请求的路径。这里为百度首页，即：<https://www.baidu.com/>  
request：当前请求的数量。  
fails：当前请求失败的数量。  
Median：中间值，单位毫秒，一半的服务器响应时间低于该值，而另一半高于该值。  
Average：平均值，单位毫秒，所有请求的平均响应时间。  
Min：请求的最小服务器响应时间，单位毫秒。  
Max：请求的最大服务器响应时间，单位毫秒。  
Content Size：单个请求的大小，单位字节。  
reqs/sec：是每秒钟请求的个数。

## charts 下面

1.吞吐量/每秒响应事务数（rps）实时统计  
2.平均响应时间/平均事务数实时统计  
3.虚拟用户数运行

# Locust参数说明

**1.命令行运行 Locust 测试**

打开Windows命令提示符，输入 locust -f load\_test.py --host=<https://www.baidu.com> --no-web -c 10 -r 2 -t 1m

F:\python1>locust -f load\_test.py --host=<https://www.baidu.com> --no-web -c 10 -r 2 -t 1m  
[2018-02-01 15:41:01,933] PCOS-1705071643/INFO/locust.main: Run time limit set to 60 seconds  
[2018-02-01 15:41:01,934] PCOS-1705071643/INFO/locust.main: Starting Locust 0.8  
[2018-02-01 15:41:01,934] PCOS-1705071643/INFO/locust.runners: Hatching and swarming 10 clients at the rate 2 clients/s...  
Name - # reqs # fails Avg Min Max | Median req/s

Total - 0 0(0.00%) 0.00

Name - # reqs # fails Avg Min Max | Median req/s

GET / - 4 0(0.00%) 147 126 157 | 150 0.00

Total - 4 0(0.00%) 0.00

**启动参数：**

–no-web 表示不使用Web界面运行测试。  
-c 设置虚拟用户数。  
-r 设置每秒启动虚拟用户数。  
-t 设置设置运行时间。

**2.打开Windows命令提示符，输入 locust --help**

| **参数** | **说明** |
| --- | --- |
| -h, –help | 查看帮助; |

-H HOST, –host=HOST | 指定被测试的主机，采用以格式：<http://10.21.32.33>;

–web-host=WEB\_HOST | 指定运行 Locust Web 页面的主机，默认为空 “。;

-P PORT, –port=PORT, –web-port=PORT | 指定 –web-host 的端口，默认是8089;

-f LOCUSTFILE, –locustfile=LOCUSTFILE | 指定运行 Locust 性能测试文件，默认为: locustfile.py;

–csv=CSVFILEBASE, –csv-base-name=CSVFILEBASE | 以CSV格式存储当前请求测试数据。;

–master | Locust 分布式模式使用，当前节点为 master 节点。;

–slave | Locust 分布式模式使用，当前节点为 slave 节点。;

–master-host=MASTER\_HOST | 分布式模式运行，设置 master 节点的主机或 IP 地址，只在与 –slave 节点一起运行时使用，默认为：127.0.0.1.;

–master-port=MASTER\_PORT | 分布式模式运行， 设置 master 节点的端口号，只在与 –slave 节点一起运行时使用，默认为：5557。注意，slave 节点也将连接到这个端口+1 上的 master 节点。;

–master-bind-host=MASTER\_BIND\_HOST | Interfaces (hostname, ip) that locust master should bind to. Only used when running with –master. Defaults to \* (all available interfaces).;

–master-bind-port=MASTER\_BIND\_PORT | Port that locust master should bind to. Only used when running with –master. Defaults to 5557. Note that Locust will also use this port + 1, so by default the master node will bind to 5557 and 5558.;

–expect-slaves=EXPECT\_SLAVES | How many slaves master should expect to connect before starting the test (only when –no-web used).;

–no-web | no-web 模式运行测试，需要 -c 和 -r 配合使用.;

-c NUM\_CLIENTS, –clients=NUM\_CLIENTS | 指定并发用户数，作用于 –no-web 模式。;

-r HATCH\_RATE, –hatch-rate=HATCH\_RATE | 指定每秒启动的用户数，作用于 –no-web 模式。;

-t RUN\_TIME, –run-time=RUN\_TIME | 设置运行时间, 例如： (300s, 20m, 3h, 1h30m). 作用于 –no-web 模式。;

-L LOGLEVEL, –loglevel=LOGLEVEL | 选择 log 级别（DEBUG/INFO/WARNING/ERROR/CRITICAL）. 默认是 INFO.;

–logfile=LOGFILE | 日志文件路径。如果没有设置，日志将去 stdout/stderr;

–print-stats | 在控制台中打印数据;

–only-summary | 只打印摘要统计;

–no-reset-stats | Do not reset statistics once hatching has been completed。;

-l, –list | 显示测试类, 配置 -f 参数使用;

–show-task-ratio | 打印 locust 测试类的任务执行比例，配合 -f 参数使用.;

–show-task-ratio-json | 以 json 格式打印 locust 测试类的任务执行比例，配合 -f 参数使用.;

-V, –version | 查看当前 Locust 工具的版本.

# Locust分布式运行

## 1.分布式运行Locust

一旦单台机器不够模拟足够多的用户时，Locust支持运行在多台机器中进行压力测试。  
为了实现这个，你应该在 master 模式中使用--master标记来启用一个 Locust 实例。这个实例将会运行你启动测试的 Locust 交互网站并查看实时统计数据。master 节点的机器自身不会模拟任何用户。相反，你必须使用 --slave 标记启动一台到多台 Locustslave 机器节点，与标记 --master-host 一起使用(指出master机器的IP/hostname)。  
常用的做法是在一台独立的机器中运行master，在slave机器中每个处理器内核运行一个slave实例。

## 注意：master 和每一台 slave 机器，在运行分布式测试时都必须要有 locust 的测试文件。

eg:  
在master模式下启动Locust:  
locust -f my\_loucstfile.py --master  
在每个slave中执行（192.168.0.14 替换为你 msater 的IP）  
locust -f my\_locustfile.py --slave --master-host=192.168.0.14

参数说明  
参数

–master

设置 Locust 为 master 模式。网页交互会在这台节点机器中运行。

–slave

设置 Locust 为 slave 模式。

–master-host=X.X.X.X

可选项，与 --slave 一起结合使用，用于设置 master 模式下的 master 机器的IP/hostname(默认设置为127.0.0.1)

–master-port=5557

可选项，与 --slave 一起结合使用，用于设置 master 模式下的 master 机器中 Locust 的端口(默认为5557)。注意，locust 将会使用这个指定的端口号，同时指定端口+1的号也会被占用。因此，5557 会被使用，Locust将会使用 5557 和 5558。

–master-bind-host=X.X.X.X`

可选项，与 --master 一起结合使用。决定在 master 模式下将会绑定什么网络接口。默认设置为\*(所有可用的接口)。

–master-bind-port=5557

可选项，与 --master 一起结合使用。决定哪个网络端口 master 模式将会监听。默认设置为 5557。注意 Locust 会使用指定的端口号，同时指定端口+1的号也会被占用。因此，5557 会被使用，Locust 将会使用 5557 和 5558。

–expect-slaves=X

在 no-web 模式下启动 master 时使用。master 将等待X连接节点在测试开始之前连接。

效果：我启动了一个 master 和两个 slave，由两个 slave 来向被测试系统发送请求。

# Locust的类和方法

**1.HttpLocust 类**

from locust import HttpLocust, TaskSet, task

class UserTask(TaskSet):

@task

def tc\_index(self):

self.client.get("/")

class UserOne(HttpLocust):  
task\_set = UserTask  
weight = 1  
min\_wait = 1000  
max\_wait = 3000  
stop\_timeout = 5  
host = "<https://www.baidu.com>"

class UserTwo(HttpLocust):  
weight = 2  
task\_set = UserTask  
host = "<https://www.baidu.com>"

每一个模拟的用户可以看做一个 HttpLocust 类的实例，HttpLocust 类有如下属性：

task\_set  
指向一个 TaskSet 类，TaskSet 类定义了每个用户的行为。

min\_wait  
用户执行任务之间等待时间的下界，单位：毫秒。如果TaskSet类中有覆盖，以TaskSet 中的定义为准。

max\_wait  
用户执行任务之间等待时间的上界，单位：毫秒。如果TaskSet类中有覆盖，以TaskSet中的定义为准。

host  
如果是 Web 服务的测试，host 相当于是提供 URL 前缀的默认值，但是如果在命令行中指定了 --host 选项，则以命令行中指定的为准。如果不是 Web 服务测试，使用默认的 None 就好。

stop\_timeout  
设置 Locust 多少秒后超时，如果为 None ,则不会超时。

weight  
一个Locust实例被挑选执行的权重，数值越大，执行频率越高。在一个 locustfile.py 文件中可以同时定义多个 HttpLocust 子类，然后分配他们的执行权重，例如：

然后在终端启动测试：  
locust -f load\_test.py UserOne UserTwo

**2.TaskSet 类**

TaskSet类定义了每个用户的任务集合，测试任务开始后，每个 Locust 用户会从 TaskSet 中随机挑选一个任务执行，然后随机等待 HttpLocust 类中定义的 min\_wait和 max\_wait 之间的一段时间，执行下一个任务。

from locust import HttpLocust, TaskSet, task

class stay(TaskSet):

def on\_start(self):

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

print("start test")

@task(3)

def readBook(self):

print('I am reading a book.')

@task(7)

def listenMusic(self):

print('I am listening to music.')

@task(1)

def logOut(self):

self.interrupt()

class UserTask(TaskSet):  
tasks = {stay:2}

@task(1)

def leave(self):

print('I don not like this page.')

class User(HttpLocust):  
task\_set = UserTask  
host = "<https://www.baidu.com>"

on\_start()：  
定义每个 Locust 用户开始做的第一件事。

[@task](https://github.com/task)  
通过[@task](https://github.com/task)()装饰的方法为一个事务。方法的参数用于指定该行为的执行权重。参数越大每次被虚拟用户执行的概率越高。如果不设置默认为1。

interrupt(reschedule=True)  
顶层的TaskSet（即被绑定到某个Locust类的task\_set的第一层TaskSet）不能调用这个方法。reschedule置为True时，从被嵌套任务出来马上选择新任务执行，如果置为False，从被嵌套任务出来后，随机等待min\_wait和max\_wait之间的一段时间，再选择新任务执行。

tasks 属性  
tasks = {stay:2} 表示每个用户执行 stay 的频率是2，也就的 UserTask 的两倍。

然后在终端启动测试：  
locust -f tc\_load\_test2.py --no-web -c 10 -r 10 -t 10s

在这个例子中虽然指定的 host ,但我们并没有调用 client() 方法发送http 请求。只是在 TaskSet 类方法中简单做了简单的打印，通过后台的输出，你可以看到每个方法被调用的频率。

## Locust设置断言

from locust import HttpLocust, TaskSet, task

class UserTask(TaskSet):

@task

def job(self):

with self.client.get('/', catch\_response = True) as response:

if response.status\_code != 200:

response.failure('Failed!')

else:

response.success()

class User(HttpLocust):  
task\_set = UserTask  
min\_wait = 1000  
max\_wait = 3000  
host = "<https://www.baidu.com>"

catch\_response = True ：布尔类型，如果设置为 True, 允许该请求被标记为失败。

通过 client.get() 方法发送请求，将整个请求的给 response， 通过 response.status\_code 得请求响应的 HTTP 状态码。如果不为 200 则通过 response.failure(‘Failed!’) 打印失败！

启动测试，运行情况：

# Locust参数化系统登录

## 以某系统登录为例，简单介绍登录用户名密码的参数化实现

from locust import HttpLocust, TaskSet, task  
from random import randint

# Web性能测试

class UserBehavior(TaskSet):

def on\_start(self):

self.login()

# 随机返回登录用户

def login\_user():

users = {"user1":123456,"user2":123123,"user3":111222}

data = randint(1, 3)

username = "user"+str(data)

password = users[username]

return username, password

@task

def login(self):

username, password = login\_user()

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

class User(HttpLocust):  
task\_set = UserTask  
min\_wait = 1000  
max\_wait = 3000  
host = "<http://www.xxx.com>"

创建 login\_user() 方法，定义登录字典 users , 通过randint 随机获取字典中的用户数据。

在 login() 登录任务中，调用 login\_user() 方法实现 随机用户的登录。

关于参数化方式很多，这里起一个抛砖引玉作用。