敏行平台压力测试报告

发布日期：2014 年 9 月

版本：V3.0

目　　录

1 引言

1.1 编写目的

本文档描述了敏行服务器进行的压力测试的方法，内容和最后的结论，该文档的作用是为实施敏行系统的用户提供一个性能上的指导，该压力测试给定了一个在特定请求量下的服务器响应指标。该响应指标可以做为系统开发和实施的基准：

1. 敏行后续的开发版本应该同该测试指标进行比较，后续发布的产品应该在主要的性能指标上不低于上一版本的测试指标。
2. 使用当前的敏行平台进行客户现场的实施，应该确保用户现场的压力水平符合当前文档所标注的水平，超过现有测试的压力水平，可能导致系统出现相应迟缓、数据丢失、系统崩溃的风险。

虽然系统可以修改系统的部署架构来增加系统的性能，但这些性能的增加往往不是线性增长的，所以，如果调整了系统的部署架构，例如分离数据库服务，都需要按照本文档给出的测试方案重新进行压力测试。

1.2 系统概述

本次压力测试使用的敏行 v2.0.2 的系统，敏行 v2.0.2 的系统在私有云环境部署分为单机版本和双机版本，我们进行测试的版本为单机版本，双机版本在理论上提供大约两倍单机性能，但本文档不包含对双机版本的测试数据。 敏行系统的后台由下列软件构成：

1. 数据库存储部分使用 Mysql 来存放用户产生的消息。本系统使用了缓存技术，因此本系统相较于传统 WEB 应用，在数据库 IO 上的性能瓶颈已经大大减少。
2. 系统的 Web 层 REST API 使用 ruby 语言实现，由于目前的 ruby 线程实现只能并行 IO，而在计算行为上无法并行，因此需要使用多个 ruby 的进程来处理

web 端的请求。

3. 为了缓存用户发送的消息，系统使用了 Redis 作为临时的数据缓存。

4. redis 也被用作业务对象之间关系的存储。

5. 系统后台数据的异步处理使用了 Sidekiq 完成。

6. 由于 Sidekiq 也是 ruby 实现的，因此如果并发消息过多，那么计算量会非常巨大，会达到 CPU 单核的计算瓶颈。

7. 如果需要更多的性能，可以启用多个 Sidekiq 实例，以便利用 CPU 并行计算提高消息处理速度。

1.3 性能设计目标

敏行系统目前按照单服务器服务 3 万人以下的用户。单服务器的配置信息为 4 核 8G 的配置。对于系统需要承载的压力来自于其他系统的经验值。根据 Twitter (http://zh.wikipedia.org/wiki/Twitter) 的历史经验，平均每天每人发送 2.5 条消息，2010 年世界杯时间，系统曾经每秒处理 3282 条消息，按照当时的用户数 7000 万推算，3 万用户每秒至少产生 2 条消息。再参考微信的数据，微信 2014 春节按照 3 亿用户计算，根据微信公布的数据，推测每个用户每分钟产生 0.03 条消息，如果根据这个数据计算，3 万用户一半活跃，系统每秒中最少要处理 8 条即时通讯消息。我们初步定义系统的性能指标为每秒 50 条消息的发送量。按照此用户规模设计系统的处理能力。

为此，我们需要将消息的发送和处理的能力作为压力测试的重点。根据压力测试的数据，评估系统的承载能力。

在压力测试上，由于涉及业务逻辑的测试，因此我们不使用主流的压力测试工具(如 Apache 的 ab)，而是自行编写 ruby 脚本发送测试请求，收集和处理必要的数据并绘制成图表。同时，我们在服务器端进行系统资源的监控。为了保证服务器的正常运行，压力测试过程中应保证服务器资源不会出现恒定的满负载状态。

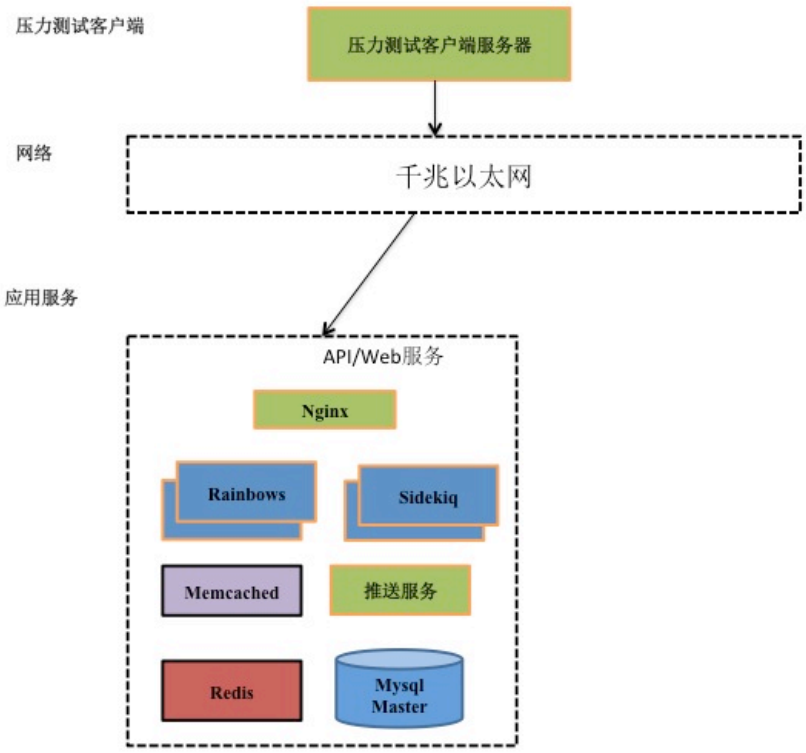
2 测试环境

2.1 软硬件环境配置

进行压力测试的服务器为单台 4 核 8G 内存的主机虚拟主机，运行在 vmware 的 esxi 的平台上。

硬件配置	应用服务器	数据库服务器	客户端
硬件配置	CPU 4Core，内存 8G，硬盘 30G	与应用服务器共用	CPU 2Core，2G
软件配置	OS:Debian 6.5 ， Rails 3.2.19，Ruby 2.1.2	Mariadb 5.5.36	ruby 2.1.2

2.2 网络拓扑结构



压力测试工具运行在一台双核的服务器上，与被测试服务器通过千兆的网络连接在一起，网络带宽不会成为系统的瓶颈。压力测试的请求、对消息的接收分别从不同的两台客户机服务器发出，确保客户端的性能和带宽不成为压力测试的瓶颈。

3 测试方法及内容

3.1 测试方法

进行压力测试需要模拟用户的登陆，发送消息，查询消息列表的操作，在测试之前需要做好下列的准备工作。

- 安装敏行平台服务器，安装测试客户端服务器
- 对服务器的参数进行调整，系统默认的 TCP 参数比较小，高并发时会拒绝服务的响应。

- 创建 2.5 万个模拟用户，将登陆的用户名和密码导出
- 使用测试脚本并发发送测试请求，记录测试结果和响应时间。
- 保证客户端不会收到连接超时、拒绝连接、服务器错误。
- 在应用服务器端记录 cpu，内存的占用信息。

每次测试后对采集数据进行分析，如果压力测试的指标不能达到响应的要求，则对系统的日志信息进行相应的分析，确认不是系统的配置问题，必要时对系统进行相应的调整，再进行相应的测试。

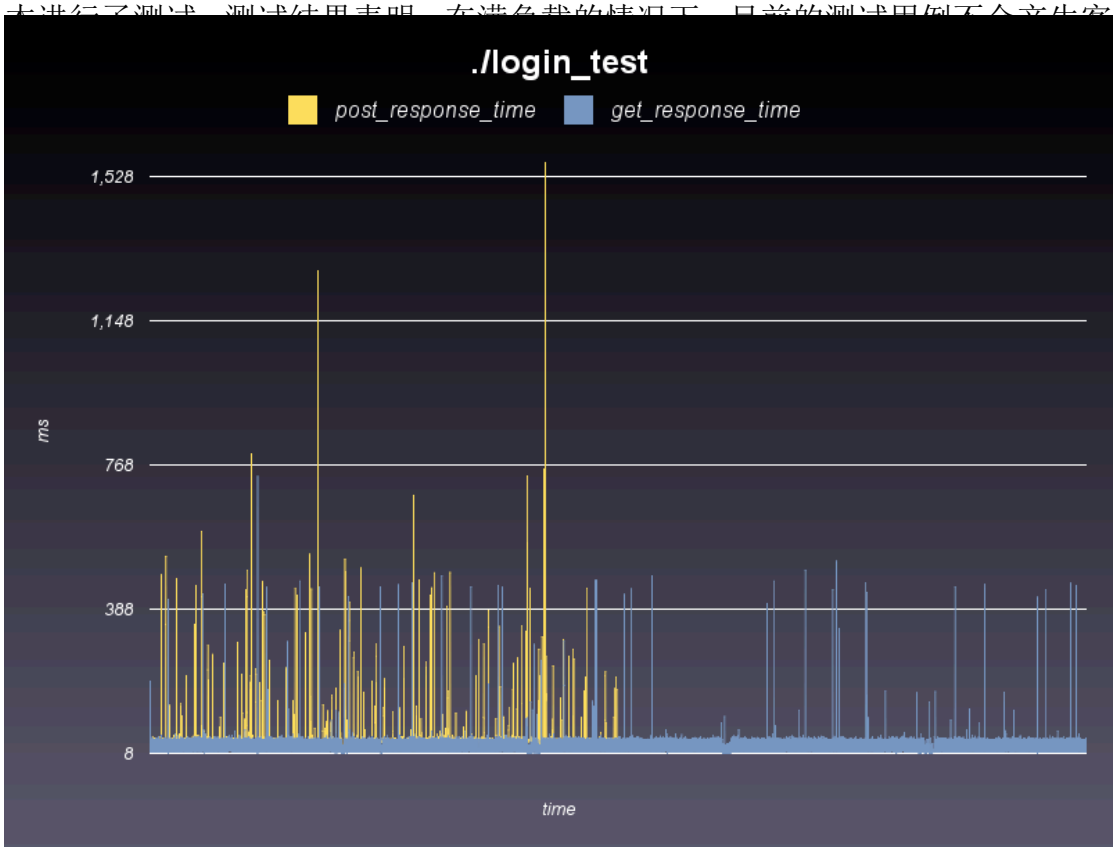
3.2 测试通过标准

敏行平台的 web 服务器端应该保证每个 HTTP 的 API 调用请求都应该有正常响应，同时在最大的并发的请求下，服务器不应该出现宕机，响应迟缓。按照现有的部署架构和系统配置，即，对于一个典型的 4CPU 服务器，在最大压力下，其处理能力应满足如下要求：

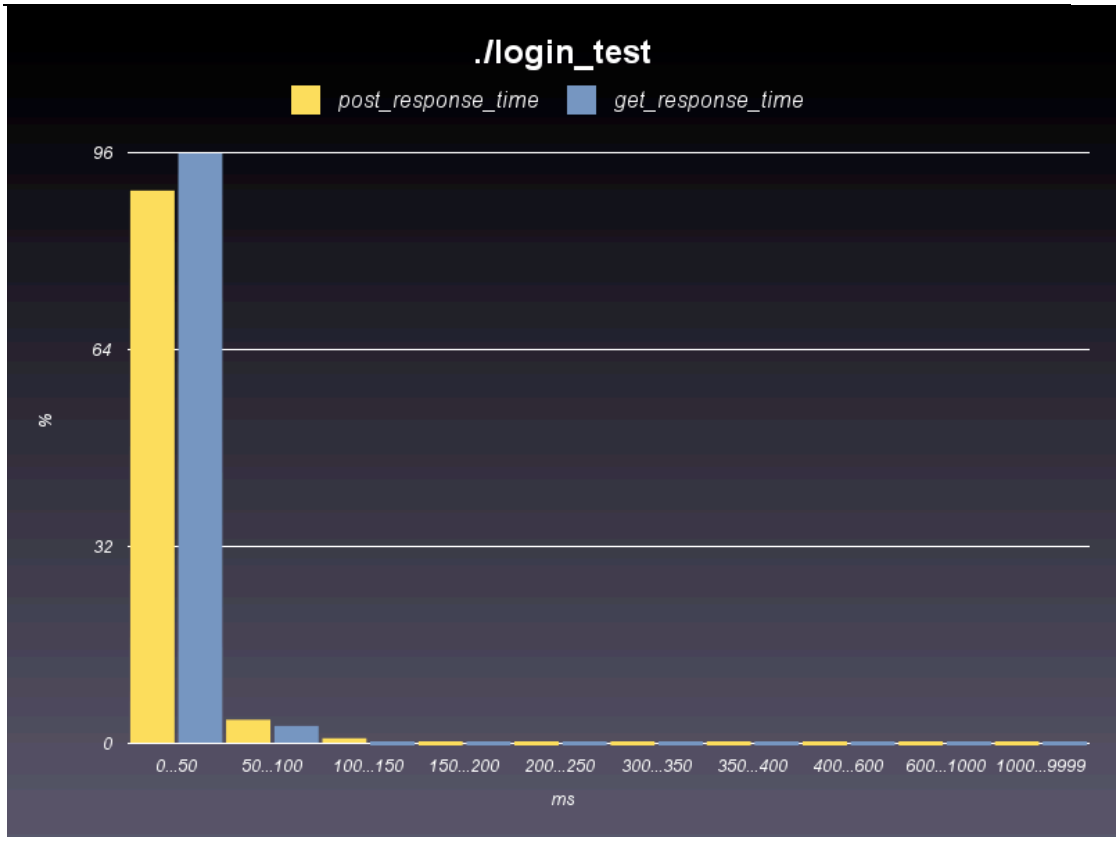
- 保证 80%的 API 请求都能在 200ms 之内得到正确的响应结果。
- 对于普通的 API，并发处理的 API 请求数不低于 30 次 / 秒。
- 服务器应至少保证每秒处理 50 条群聊请求。
- 这 50 条群聊请求，应有 80%能在 3 秒以内推送到群聊的所有用户。
- 在最大并发请求下，服务器内存不应该超过系统的全部内存的 80%。
- 对于在此压力下的少量新请求，不应有人类可感知到的延迟。
- 最大并发请求持续的时间不小于 10 分钟，在请求并发请求结束之后，系统的 CPU 占用应在 10 钟内下降到 10%以下。

4 测试

由于涉及部分业务逻辑的测试，因此我们并没有采用 ab 等流行的压力测试工具，而是使用 ruby 自行编写测试脚本。由于 MRIruby 的多线程实现尚无法执行 CPU 并行计算，只能实现 IO 的并行，因此我们在执行压力测试之前，为了验证客户端的测试脚本不会受到单核 CPU 并发数的限制，我们对这个 ruby 测试脚本进行了测试，测试结果表明，在满负载的情况下，只前的测试用例不会产生客



)



4.2 文本消息发送测试

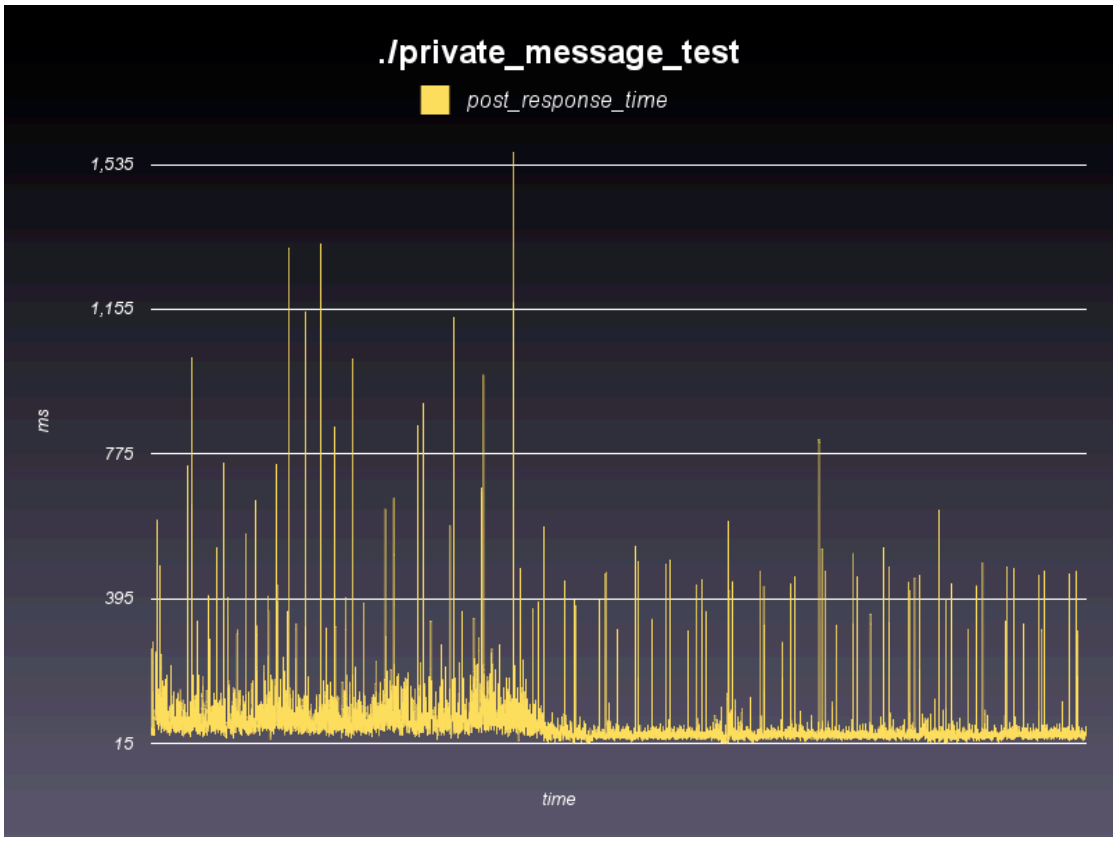
文本消息测试分为三个类型：

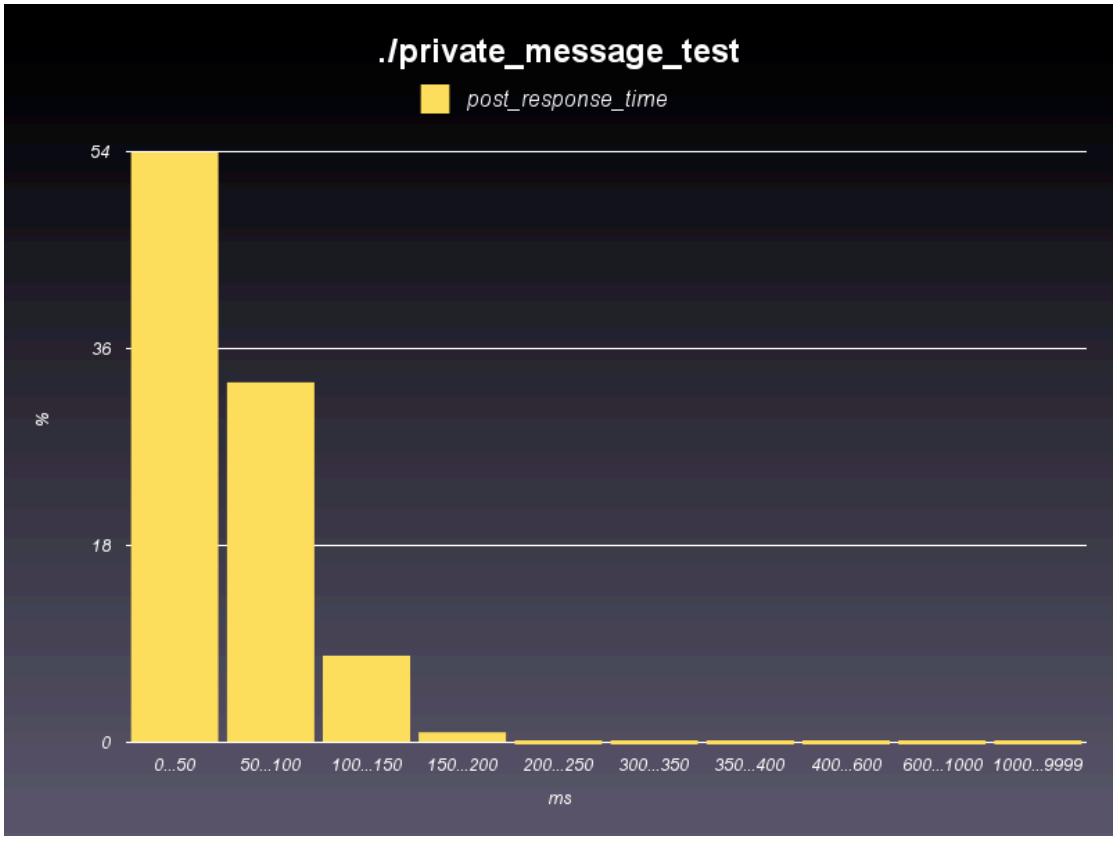
- 一对一私信聊天测试
- 正常群聊测试
- 大量建立新群聊测试

我们对每项测试，获取并记录 post 请求和 get 请求的响应时间，采集到必要的信息后做出图表，如下：

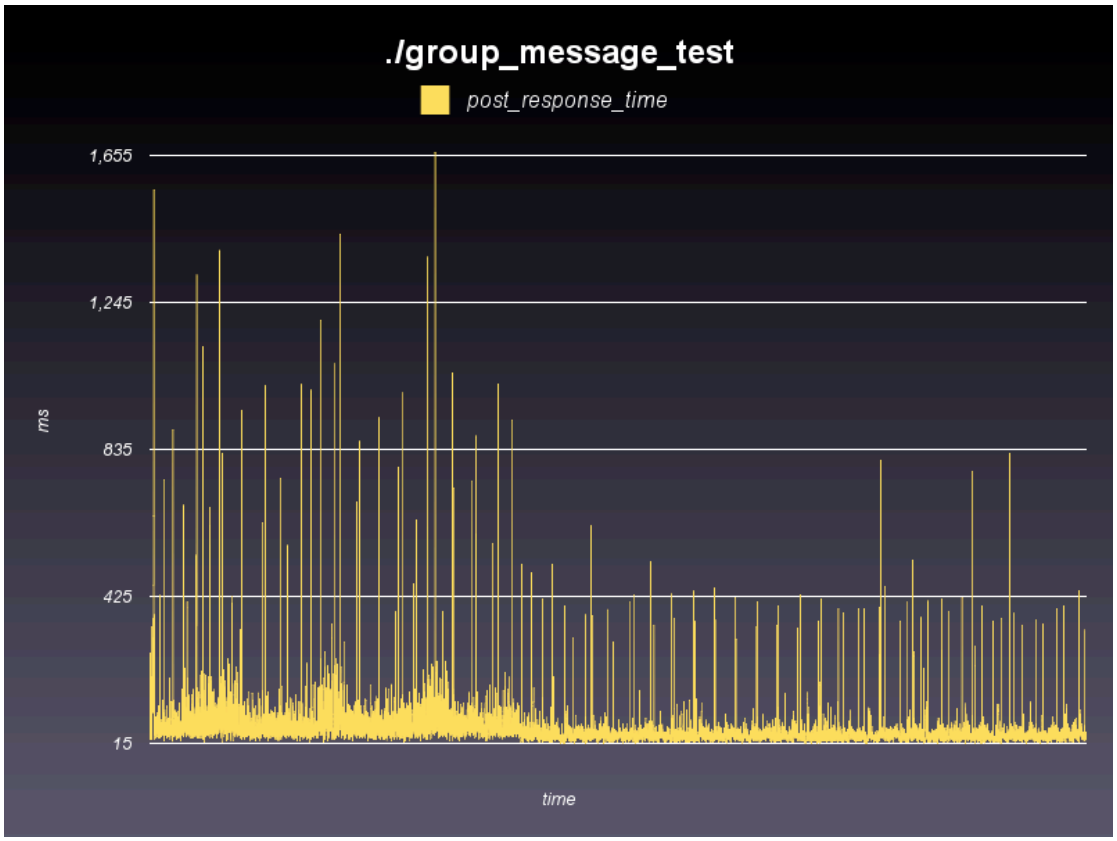
其中两张图片为一组。第一张图表对绝对数量绘图，横坐标为次数，纵坐标为毫秒;第二张图片为落在某个区间内的比率，横坐标为图中所示的左闭右开区间，纵坐标为占总数的百分比。其中，出于精确度的原因，在计算区间时，占比小于百分之一的数据会被舍弃掉。

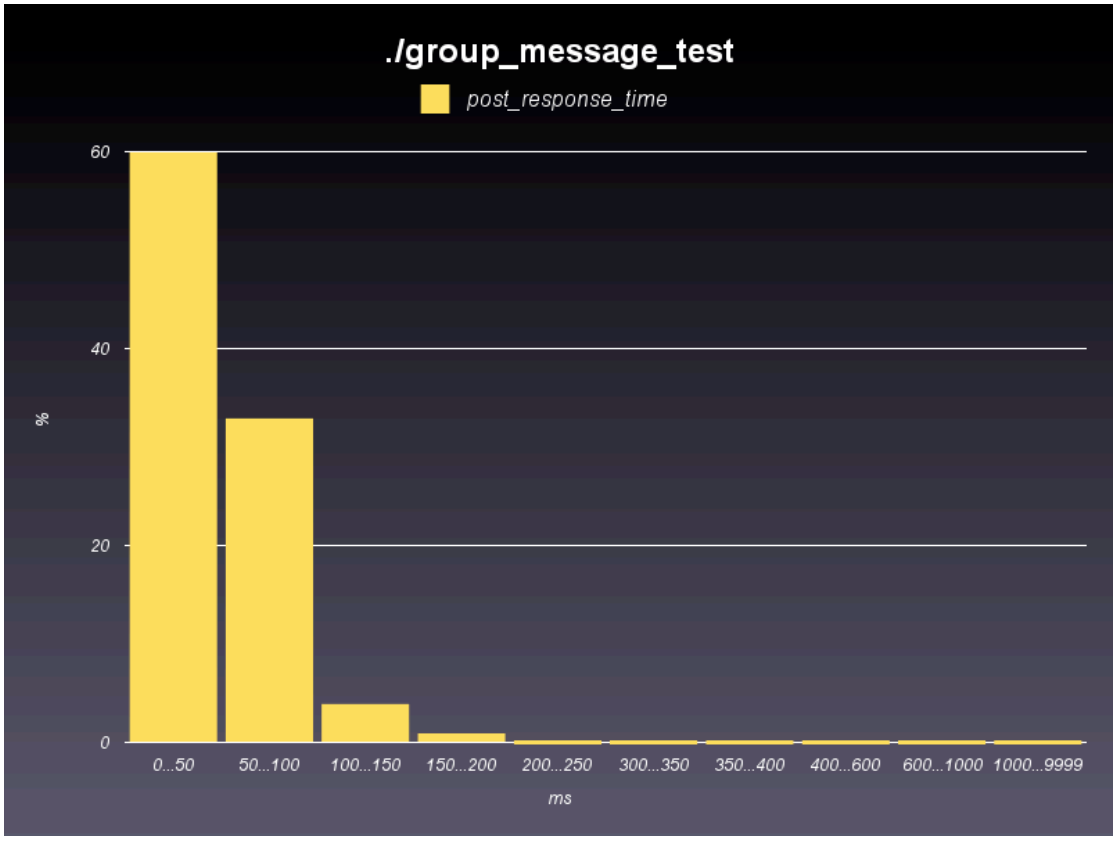
4.2.1 私信聊天测试


./private_message_test
post_response_time
1,535
1,155
775
395
15
ms
time

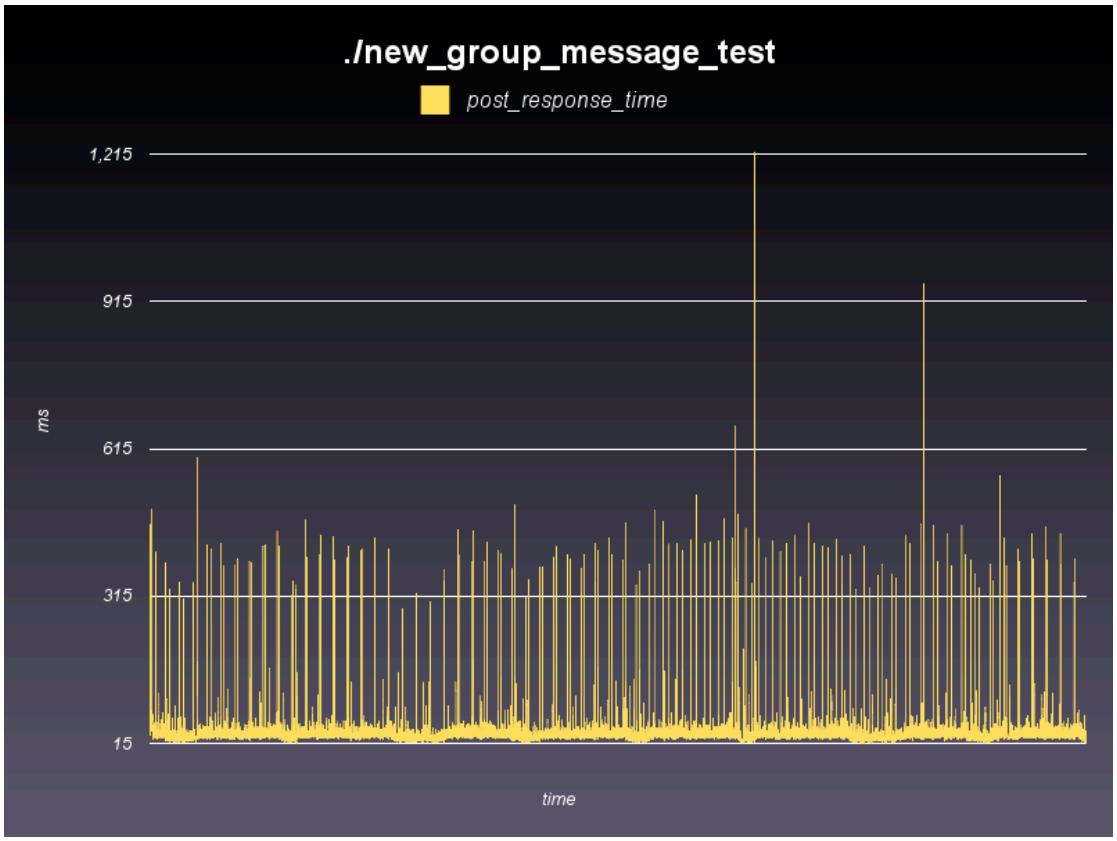

./private_message_test
post_response_time
54
36
18
0
%
0...50
50...100
100...150
150...200
200...250
300...350
350...400
400...600
600...1000
1000...9999
ms

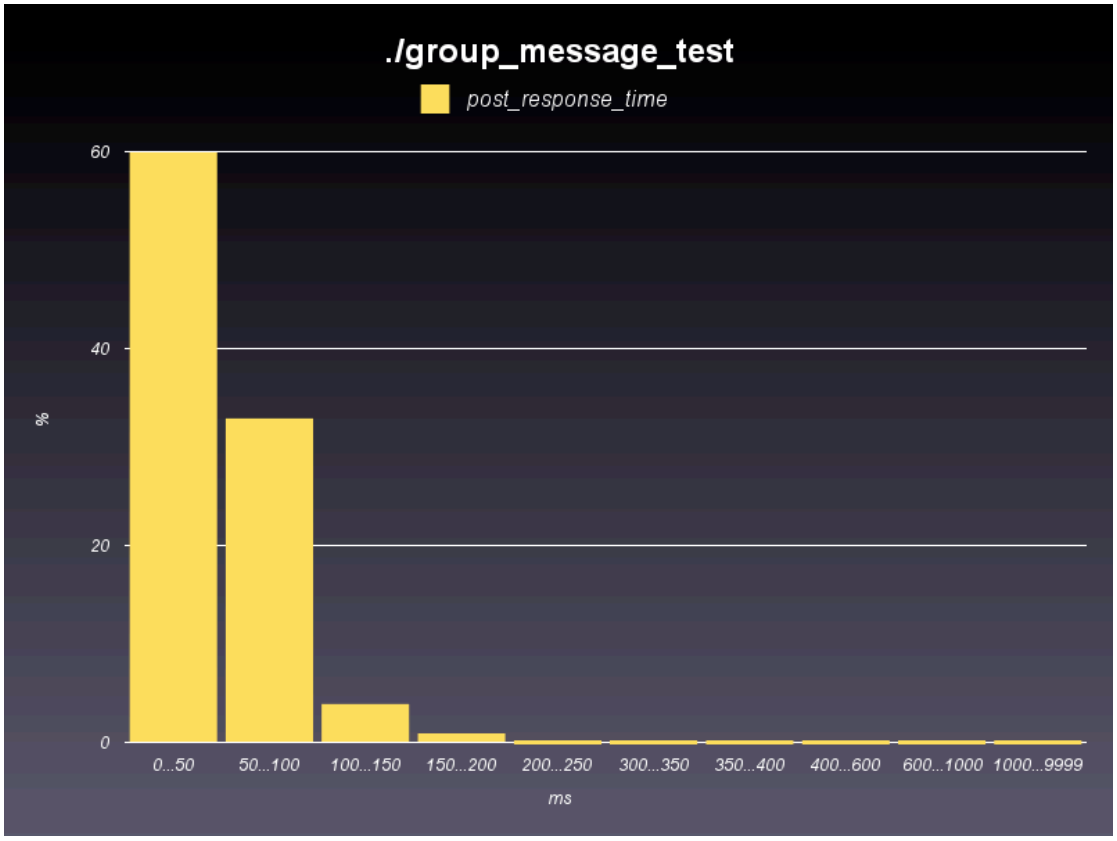
4.2.2群聊测试


./group_message_test
post_response_time
1,655
1,245
835
425
15
ms
time


./group_message_test
post_response_time
60
40
20
0
%
0...50
50...100
100...150
150...200
200...250
300...350
350...400
400...600
600...1000
1000...9999
ms

4.2.3 不断新建群聊测试





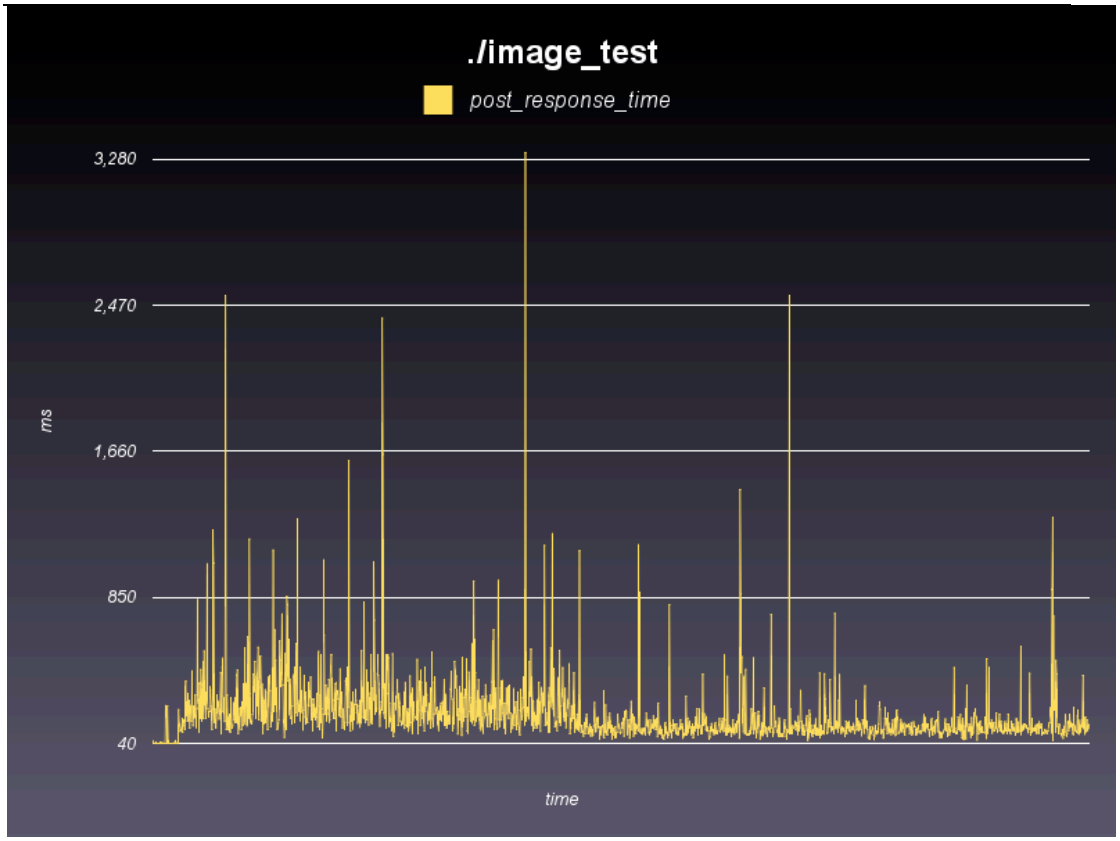
4.3图片文件上传测试

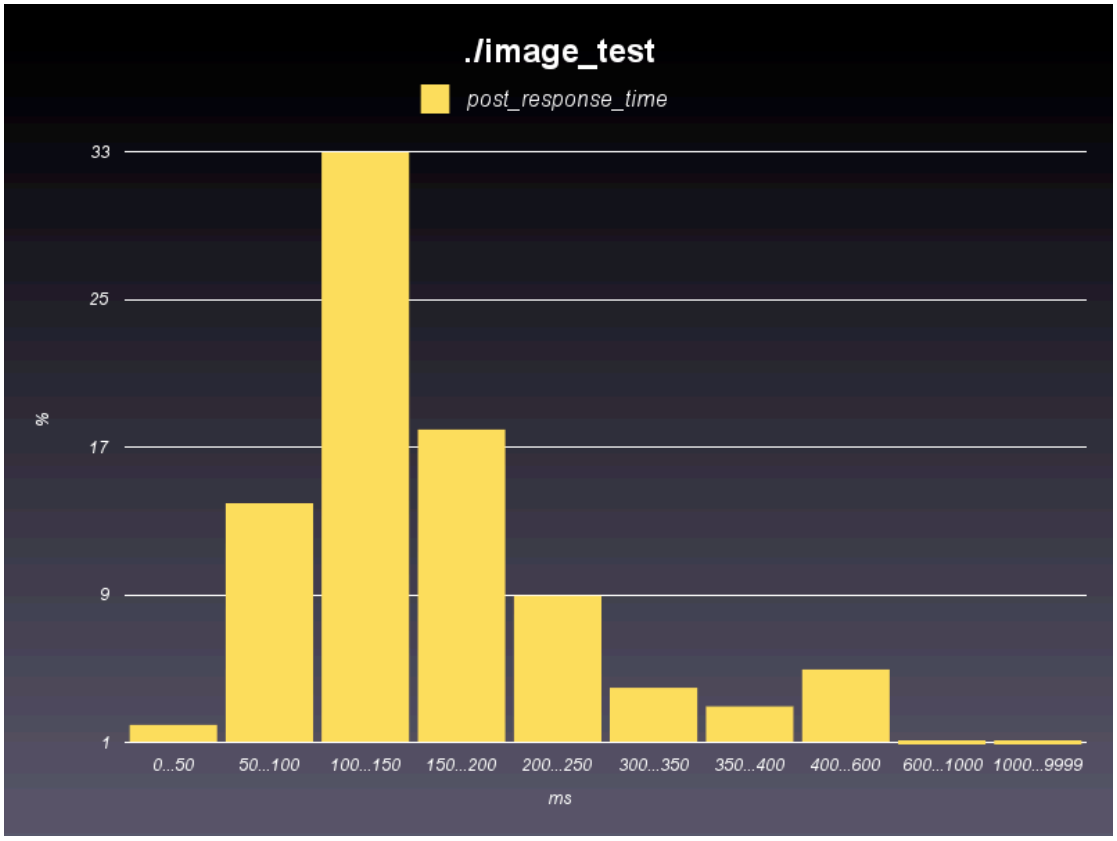
图片上传测试使用 50 个线程（用户），其中每个用户不断地从预设的图片中随机选取一张，发送到服务器，统计 API 响应时间。

考虑到用户发送图片的实际情景可能为表情、截图、照片等，我们选取一定量、不同质量的 jpg 格式小、中、大、巨大图片做模拟测试。

其文件大小和分辨率大小如下：

图片编号	分辨率	文件大小
1	180X180	60KB
2	180X180	60KB
3	180X180	60KB
4	180X180	60KB
5	180X180	60KB
6	180X180	60KB
7	250x250	25.5KB
8	1600x1200	18.5KB
9	1600x1200	456KB
10	1600x1200	454KB
11	1024x768	126KB
12	320x320	32.8KB
13	1440x1040	151KB

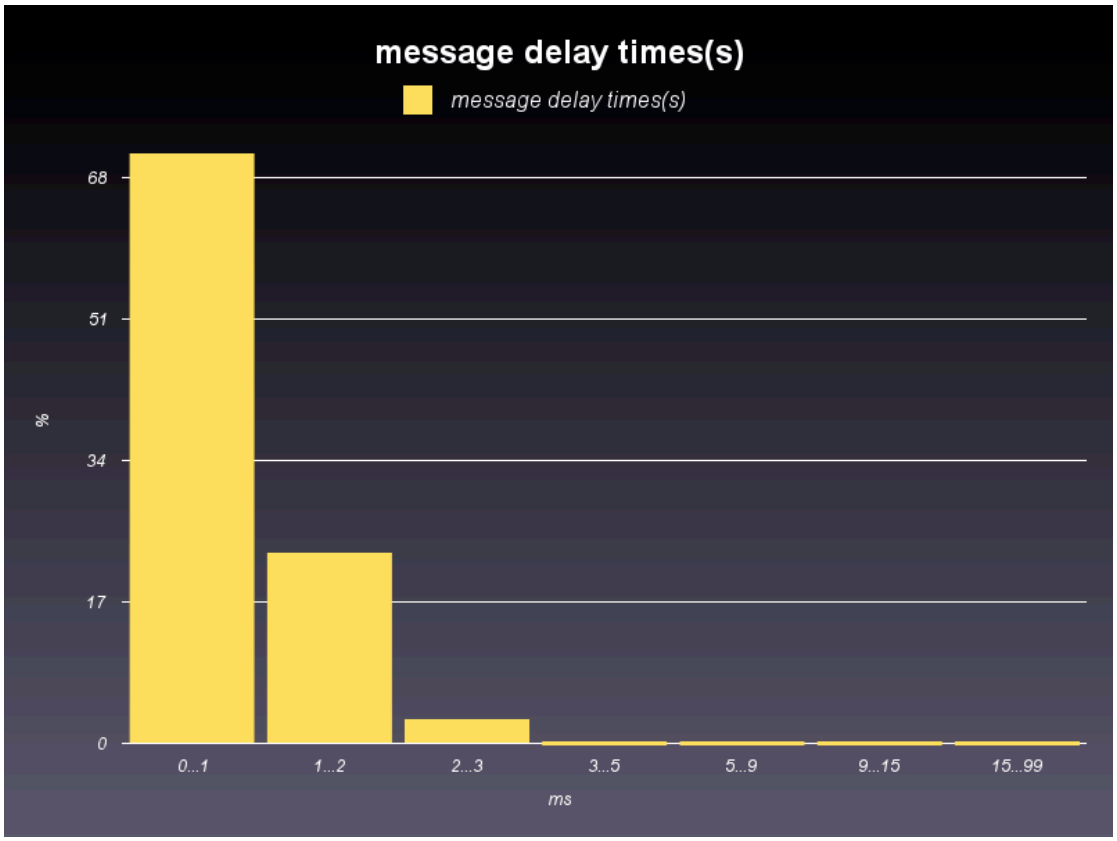

./image_test
post_response_time
3,280
2,470
1,660
850
40
ms
time


./image_test
post_response_time
33
25
17
9
1
%
0...50
50...100
100...150
150...200
200...250
300...350
350...400
400...600
600...1000
1000...9999
ms

4.4 接收推送测试

这里统计从发送文本消息，至客户端实际收到这条消息时，二者的延时。由于服务器端时间和网络的细小差距，这个时间精确到毫秒时误差较大，因此这里只精确到秒，采用舍尾法。

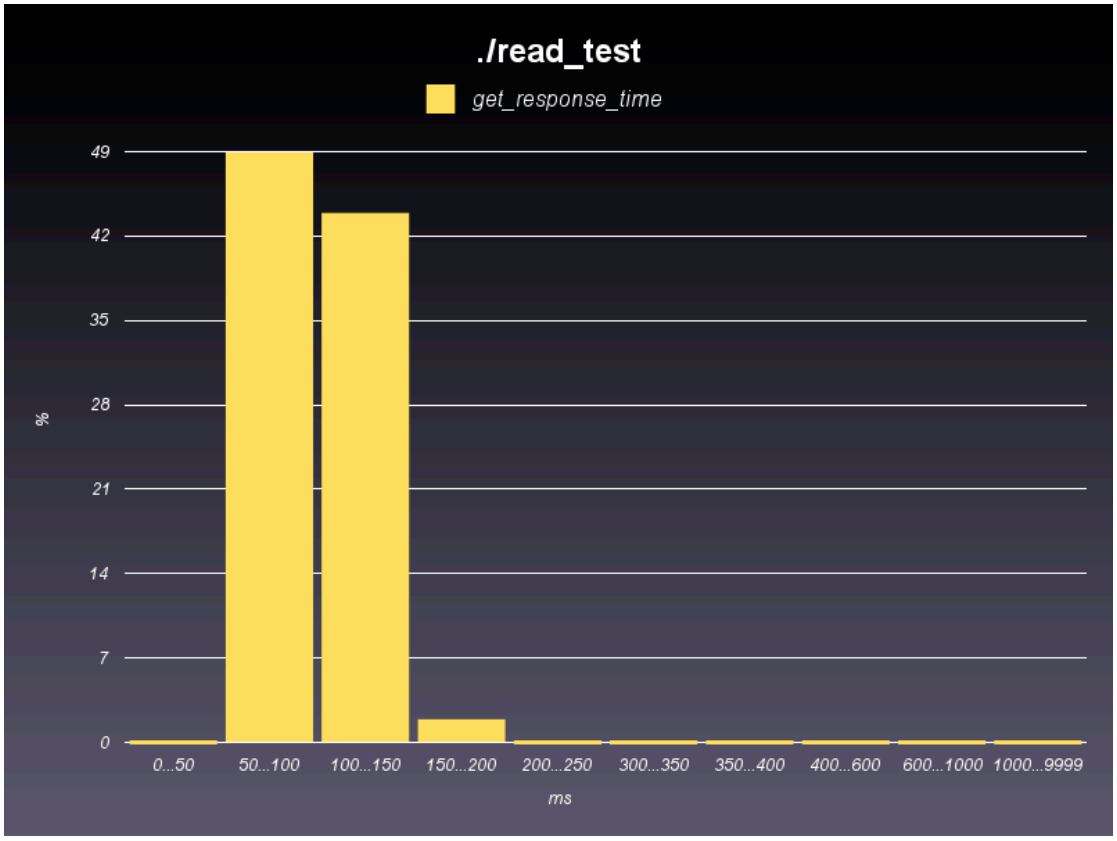
横坐标为响应时间的左闭右开区间（秒），纵坐标为落在该区间内的百分比。可见在预计的高压力下，70%的聊天消息可以在 1 秒以内接收到，20%的消息可以在一秒到两秒以内接收到。



4.5 查询工作圈消息

这里统计在满载压力时，另一客户端使用五个用户，登陆后不间断反复查看工作圈的延时。


./read_test
get_response_time
733
553
373
193
13
ms
time


./read_test
get_response_time
49
42
35
28
21
14
7
0
%
0...50
50...100
100...150
150...200
200...250
300...350
350...400
400...600
600...1000
1000...9999
ms

5 结论分析

经过测试，系统在大量的用户并发的情况下，没有出现系统崩溃，响应缓慢的问题。有少量请求响应时间较长，但其比率非常低，大多数情况下应不影响体验。

系统内存占用符合设定的目标，系统的响应时间也符合目标要求。CPU 负载较预期值为高，但没有拖慢系统响应。在反复的压力测试下，系统持续运行性良好，API 没有返回错误信息，非常稳定。