UiBot RPA 平台

性能测试报告

北京来也网络科技有限公司

目录

一、测试介绍

UiBot 产品是北京来也网络科技有限公司推出的 RPA+AI 流程自动化平台，包含创造者 Creator、劳动者 Worker、指挥官 Commander 三大模块。各司其职，为机器人的生产、执行、分配提供了良好载体。

1.1 测试目的

评估 Commander 管理系统在单节点场景下的任务调度能力，为客户实施部署与容量规则提供参考，为后续性能优化提供依据。

1.2 目标读者

售前人员、销售人员、实施人员、项目经理、目标客户、内部研发人员。

1.3 测试范围

UiBot RPA 平台包含创造者 Creator、劳动者 Worker、指挥官 Commander 三大模块，其中创造者 Creator、劳动者 Worker 属客户端，为本地运行软件，其性能与编写的流程内容紧密相关，存在较大的不确定性，故不在本次测试范围内。

本次测试范围主要针对 Commander 管理后台，Commander 同时负责组织机构、测试数据、流程、任务、计划、Creator、Worker 等的管理工作，其必须时刻与所有 Worker 保持连接，并为 Worker 提供授权、状态管理、任务下发、任务结果上报等服务，存在较大并发，是整个系统的压力与关键所在。

1.4 专业术语

术语	释义
QPS	每秒查询（请求）数(Query per second) 性能指标，评价系统性能均以每秒钟完成的技术交易的数量来衡量。系统整体处理能力取决于处理能力最低模块的 RPS 值。

RRT	平均请求响应时间(Requests Response Time) 性能指标，所有请求耗时之和 / 请求总数，有时不能很好的反映系统的波动情况。（单位是秒）
90%时间	90%的请求可以在此时间范围内完成，10%的请求会超出此时间，作为反映系统波动的补充指标
50%时间	50%的请求可以在此时间范围内完成，另外 50%的请求会超出此时间，注意与平均响应时间不是一个概念。
并发数	并发数是指系统同时能处理的请求数量，这个也是反映了系统的负载能力。
错误率	出错的请求数 / 请求总数

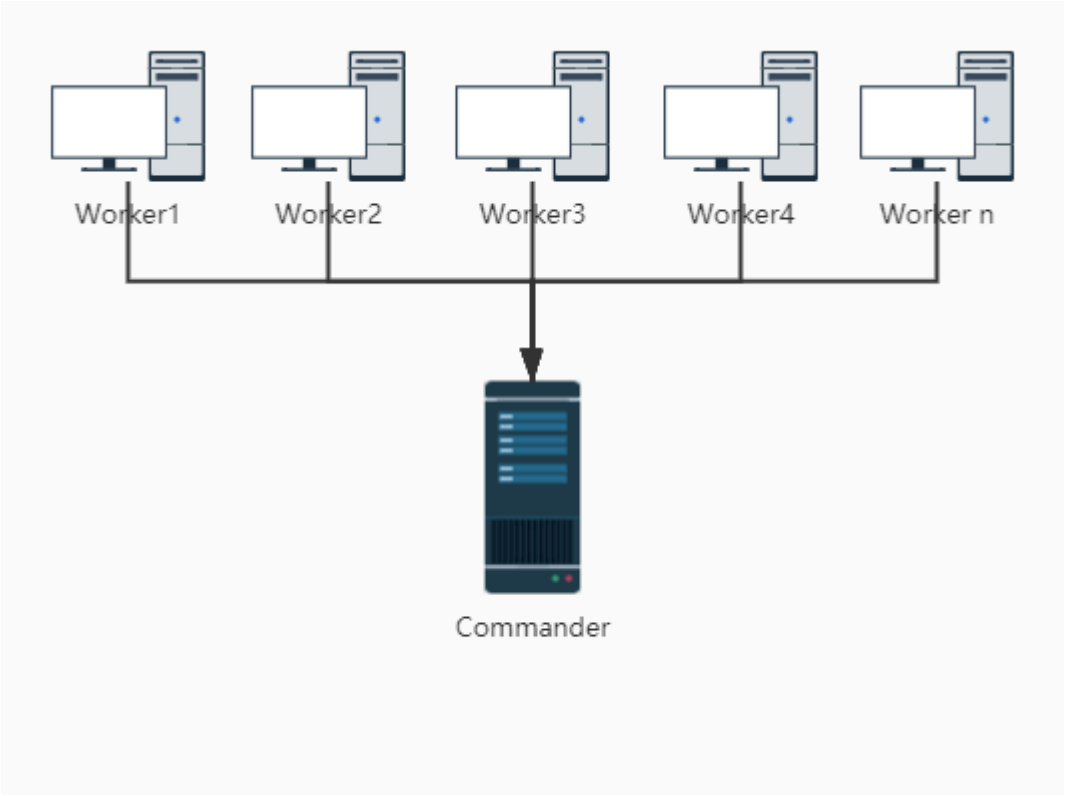
二、测试资源

2.1 测试环境

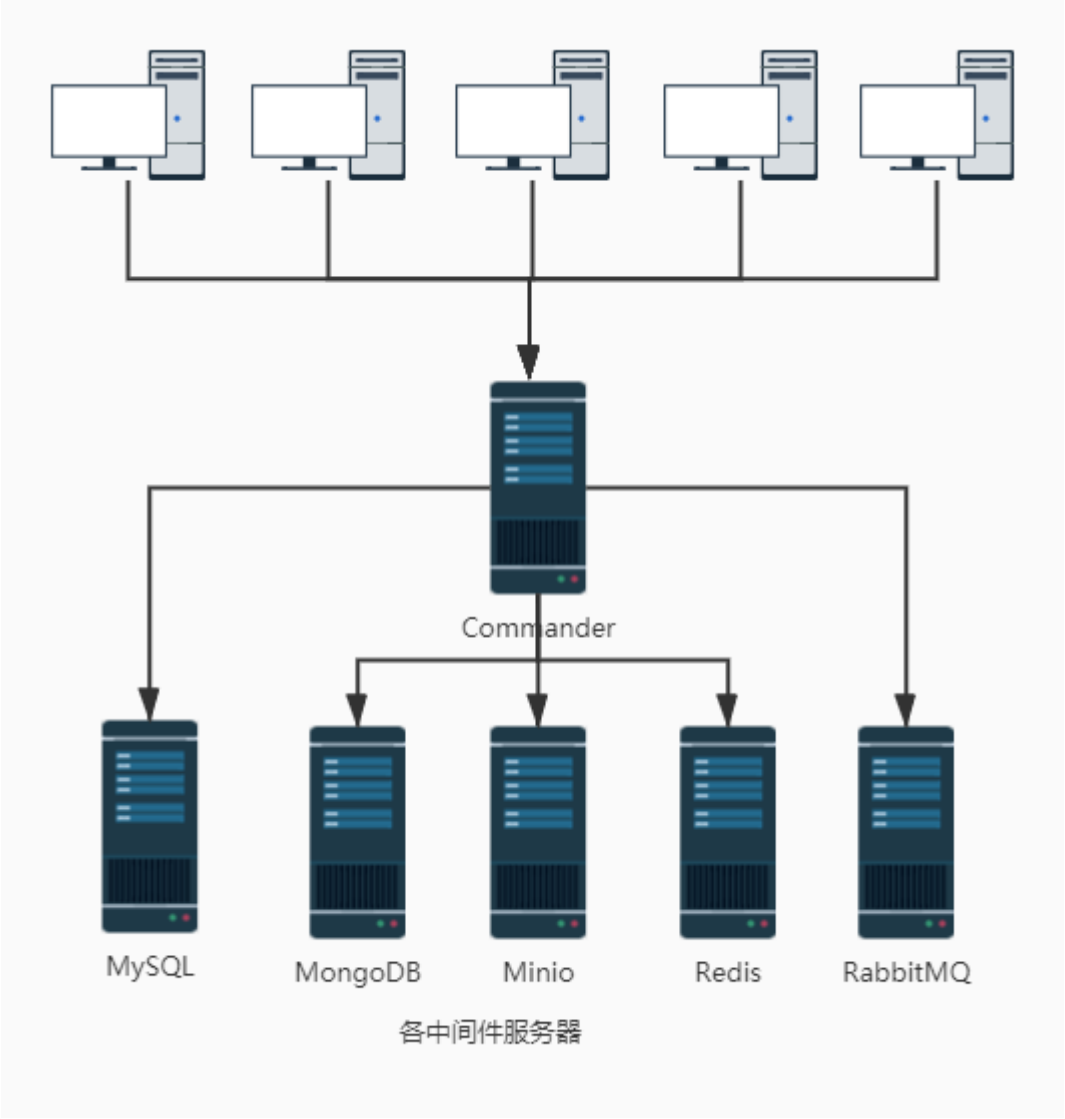
服务器地址	软硬件配置		用途
	硬件	软件	
192.168.0.25	内存：8G CPU:4核 Intel ® Xeon ® Platinum 8269CY	Centos 7 Asp.Net Core:3.1 MySQL MongoDB Redis MinIO	Commander 服务器

192.168.0.121 192.168.0.120	内存：16G CPU:4核 Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz	Windows 10 Python3.8 Locustio 0.14.5	施压客户机

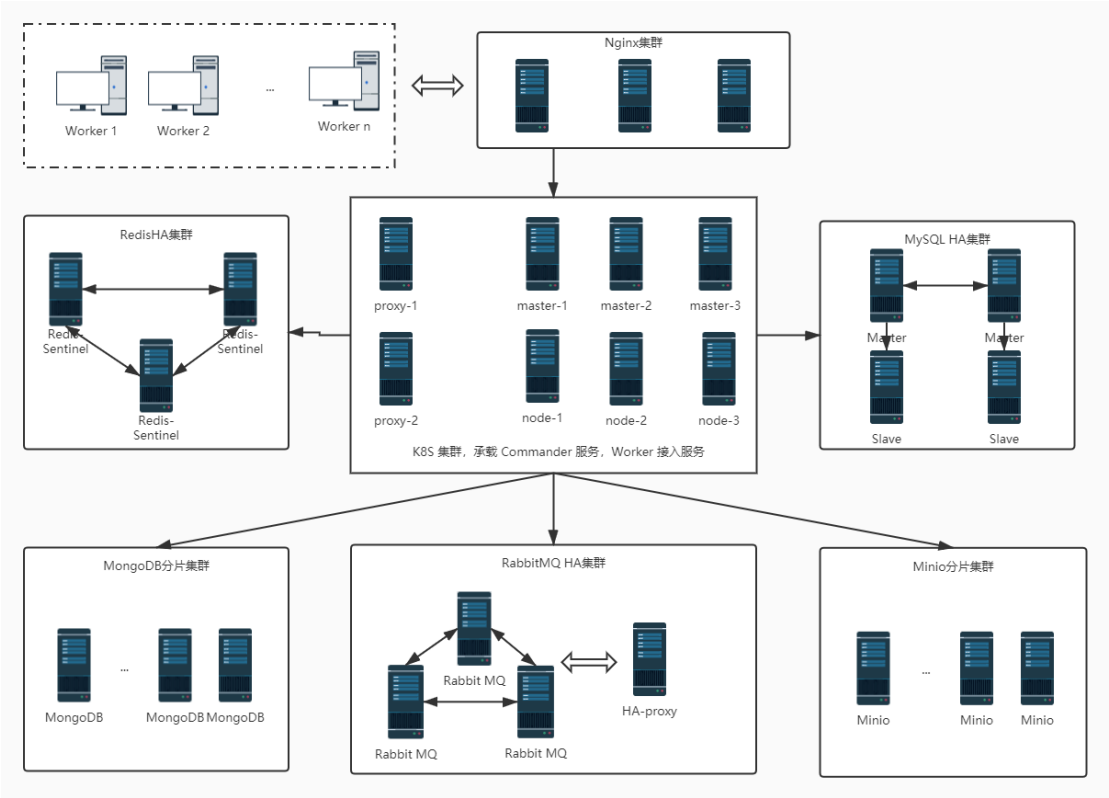
本次测试服务器采用单节点部署方案，如下图所示。其中 Commander 节点上包含 Commander 服务,Redis,MongDB,MinIO,MySQL,RabbitMQ。



另外，服务器支持多节点部署以及 Kubernetes 集群高可用性部署等方案，多节点部署如下图所示：



Kubernetes 高可用性部署方案如下图所示:



2.2 测试工具

序号	工具名称	版本	用途	备注
1	Locustio	0.14.5	开源性能测试框架	
2	Python	3.8	性能测试脚本	
3	nmon		服务器资源监控	

2.3 测试数据

在 Commander 端同一部门同一环境中新建 10 个计划，每个计划每分钟共产生 50 个任务，总计每分钟共产生 500 个新任务，除了每分钟新生成的任务外，还存在历史任务 1 万。当有 Worker 注册到此环境时，Commander 会不停的将任务派发下去，Worker 会不停的消费任务池中的任务，当所有 Worker 都启动

后，观察队列中任务的数量，如果任务一直在增加，说明此时消费能力已经饱和，可能是压力机出现了瓶颈也可能是 Commander 端单队列时的调度能力出现了瓶颈；如果队列中任务稳定在其个值，说明任务产生速度正好等于 Worker 消费速度；如果队列中的任务在不停的减少，说明 Commander 端单队列的任务调度能力大于每秒 500 个。

三、测试策略

3.1 测试场景

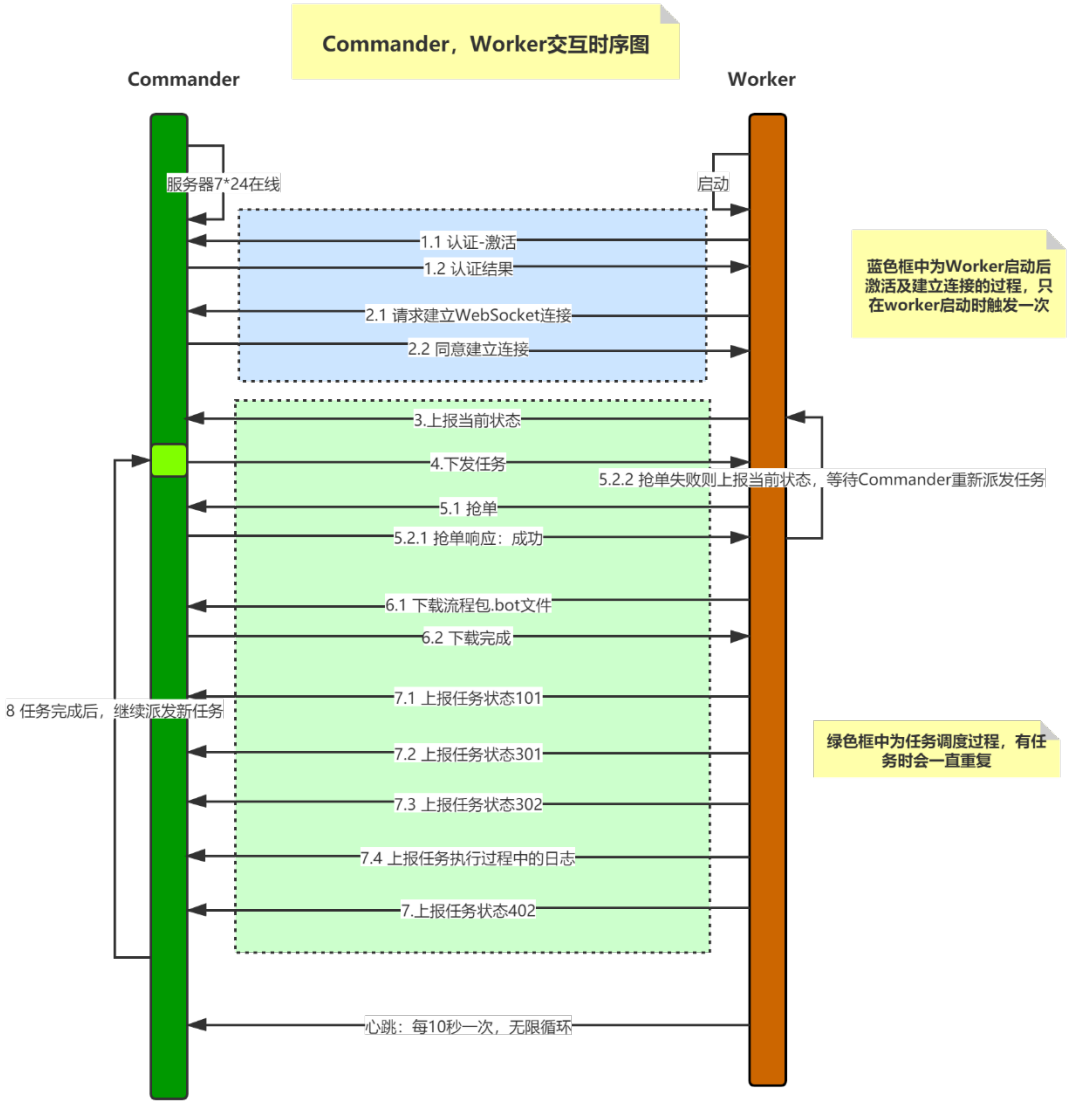
本测试选取 Commander 与 Worker 交互的调度场景为作测试场景，此场景支持多 Worker 并发，是 Commander 最大压力所在。从 Workers 启动开始，到 Commander 开始派发任务，直到所有任务调度完成，共包含以下交互：

1. http 协议: Worker 启动后进行认证激活；
2. http 协议：Worker 激活成功后主动与 Commander 建立 WebSocket 连接；
3. WebSocket 协议：连接建立后 Worker 主动向 Commander 上报当前状态；
4. WebSocket 协议：Commander 开始给 Worker 派发任务；
5. http 协议：确认接收——抢单(同一任务 Commander 会推给多个 Worker)；
6. http 协议：抢单成功后下载.bot 流程包；
7. WebSocket 协议：抢单失败后从步骤 3 开始重复执行；
8. WebSocket 协议：上报当前任务处理状态 101；
9. WebSocket 协议：上报当前任务处理状态 301；
10. WebSocket 协议：上报当前任务处理状态 302；
11. http 协议：上报流程执行过程中的日志信息；

12. WebSocket 协议：上报当前任务处理状态 402；

13. 从步骤 4 开始重复执行，直到所有任务处理完成或者退出；

整个过程中的交互如下时序图所示：



3.2 测试脚本

整个测试过程中涉及到 http 协议与 WebSocket 协议的交叉调用，考虑到复杂性与灵活性，本次采用开源测试框架 Locust 作为测试工具，手动编写 Python 程序来实现性能的压测。

编写 Python 程序，模拟 Worker 与 Commander 进行通信，采用 Locust 作为性能数据收集工具，借用其强大的并发能力，可轻松在单台施压机器上模拟上千的 Worker，以给 Commander 造成较大负载，观测并记录 Commander 端的性能表现，评估其性能指标。

3.3 施压策略

在单台施压机器上模拟 500 台 Worker，每秒钟启动 16.6 个，共计 30 秒内全部启动，启动后持续运行 30 分钟。整个过程中不设置思考时间，不设置集合点。

3.4 监控策略

Commander 服务端采用 nmon 监控系统资源的使用情况；采用 Kibana 监控日志，查看有无报错；采用 Locust 自带的性能数据收集功能监控性能指标；

Worker 客户端采用 Windows10 的任务管理器监控资源使用情况。

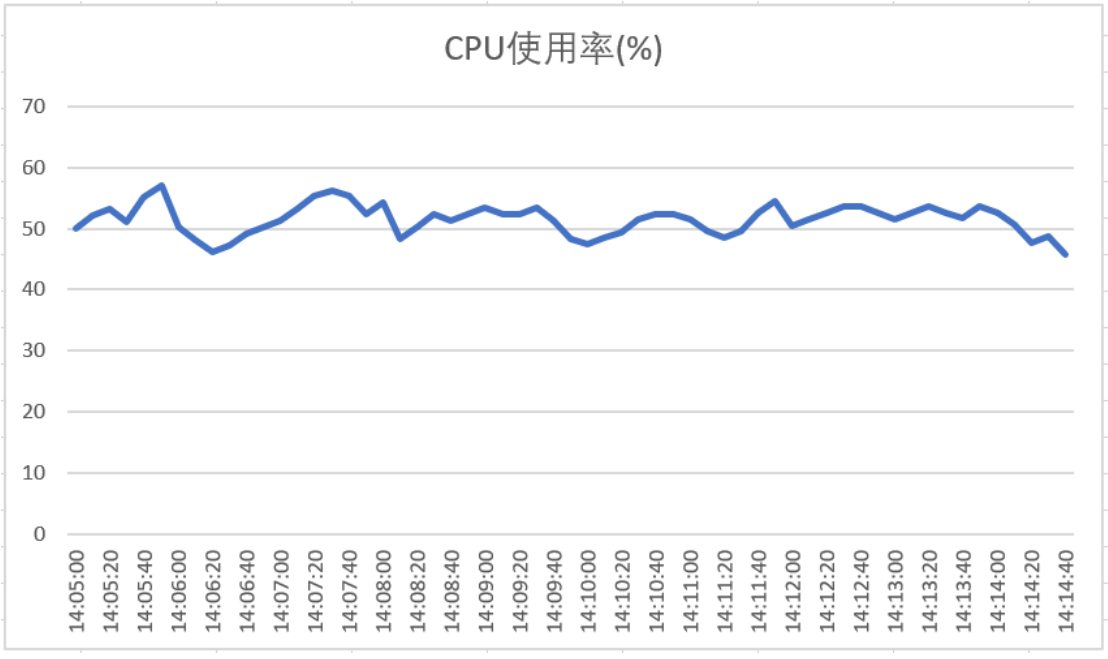
四、测试结果

4.1 性能指标

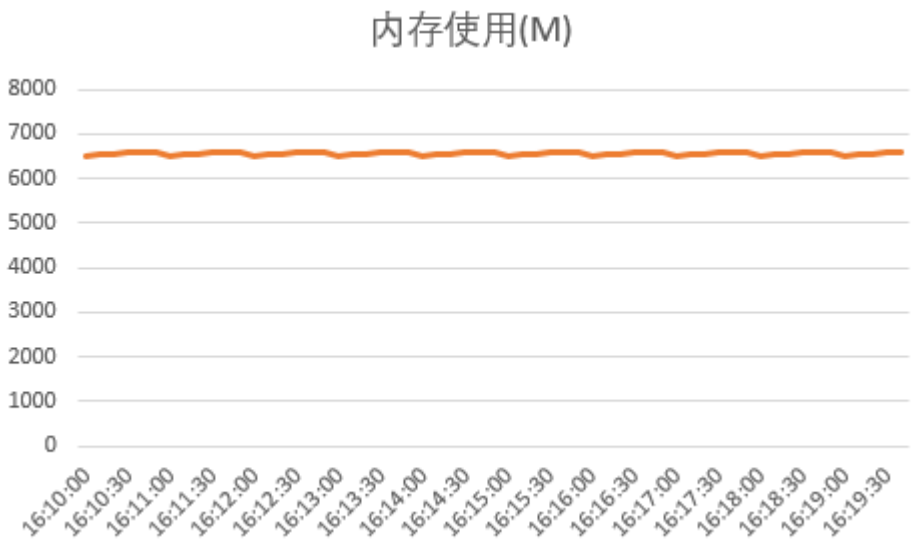
请求类型	请求名称	请求总数	失败数	50%时间 (ms)	90%时间 (ms)	平均响应时间 (ms)	最小响应时间 (ms)	最大响应时间 (ms)	平均响应大小 (bytes)
POST	worker认证	500	0	31	54	44	11	407	457
POST	websocket-创建连接	500	0	10	15	8	4	20	242
WEBSOCKET	心跳	29205	0	0	0	0	0	1	75
WEBSOCKET	上报当前任务状态	5136	0	0	0	0	0	1	77
WEBSOCKET	commander下发任务	1989	0	297	531	302	38	11021	360
POST	worker接收任务（抢单）	1985	0	424	681	459	7	901	264
GET	下载流程bot包	1984	0	50	156	70	27	201	12711
WEBSOCKET	worker上报任务执行101状态	1984	0	0	0	0	0	1	94
WEBSOCKET	worker上报任务执行301状态	1982	0	0	0	0	0	1	94
WEBSOCKET	worker上报任务执行302状态	1982	0	0	0	0	0	1	94
WEBSOCKET	worker上报任务执行402状态	1980	0	0	0	0	0	2	96
POST	worker上报任务执行日志	3618	0	22	44	29	13	229	41

4.2 服务端资源使用情况

服务器 CPU 使用率：维持在 50%左右，未到瓶颈。

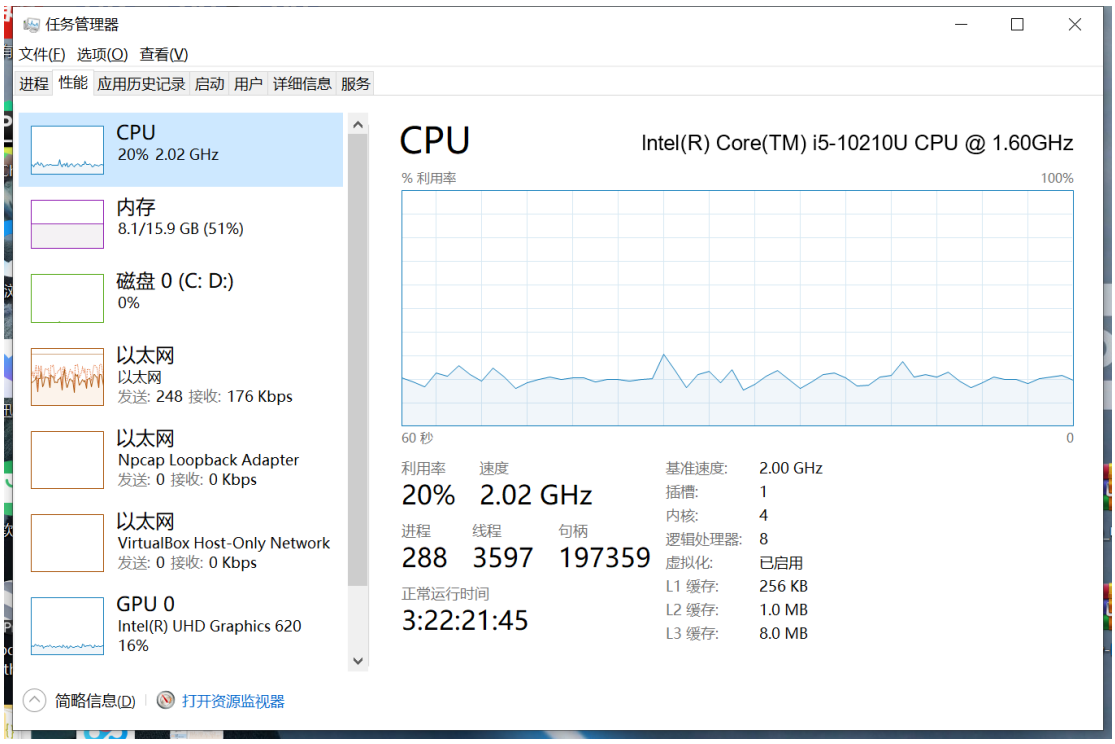


服务器内存使用率：维持在 8G 左右，内存尚有较多剩余。

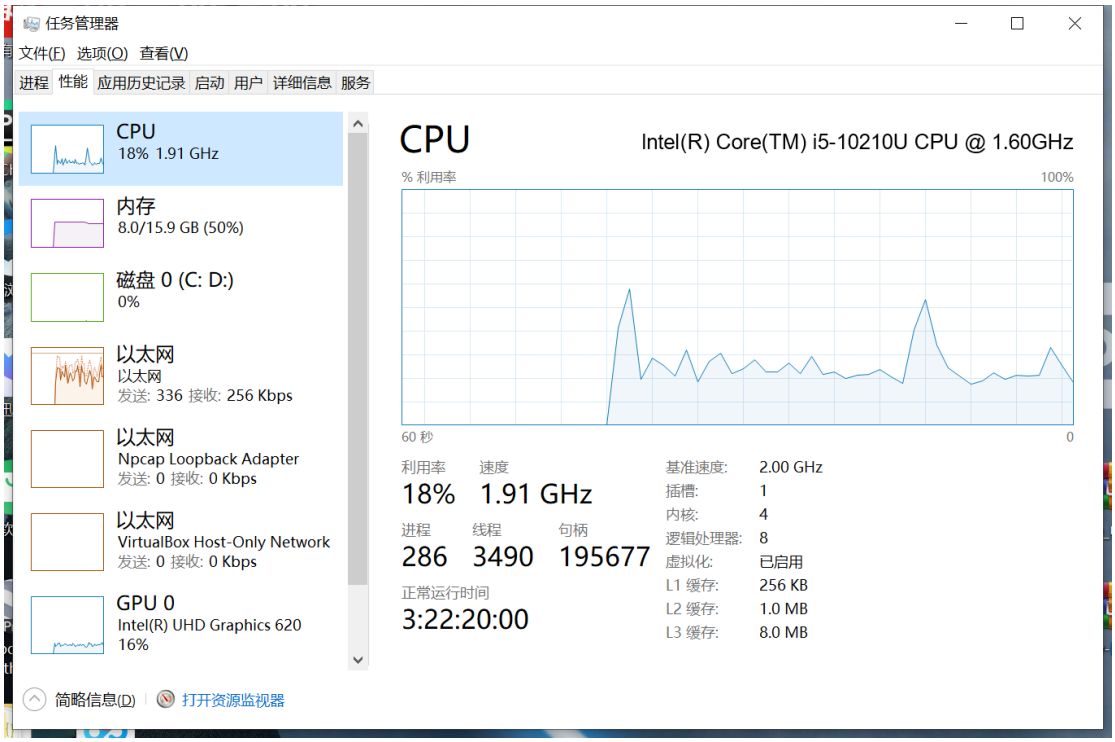


4.3 客户端资源使用情况

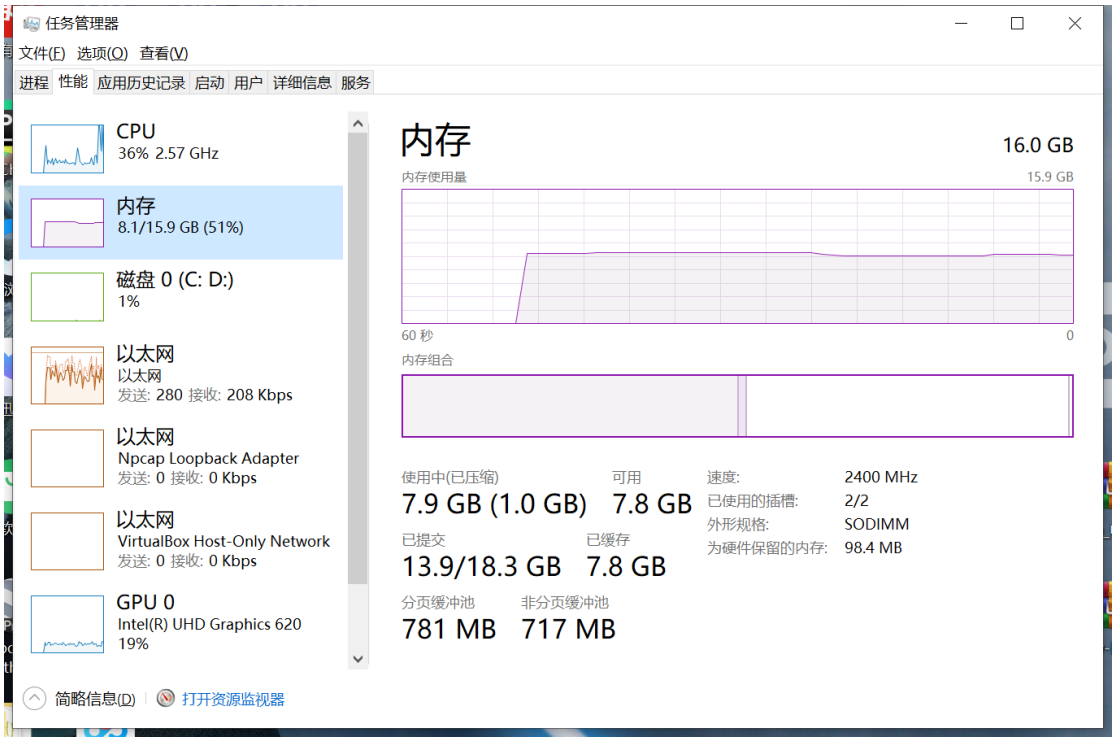
施压机 1 CPU 使用率：施压脚本启动后，客户机 CPU 使用率维持在 20%左右。



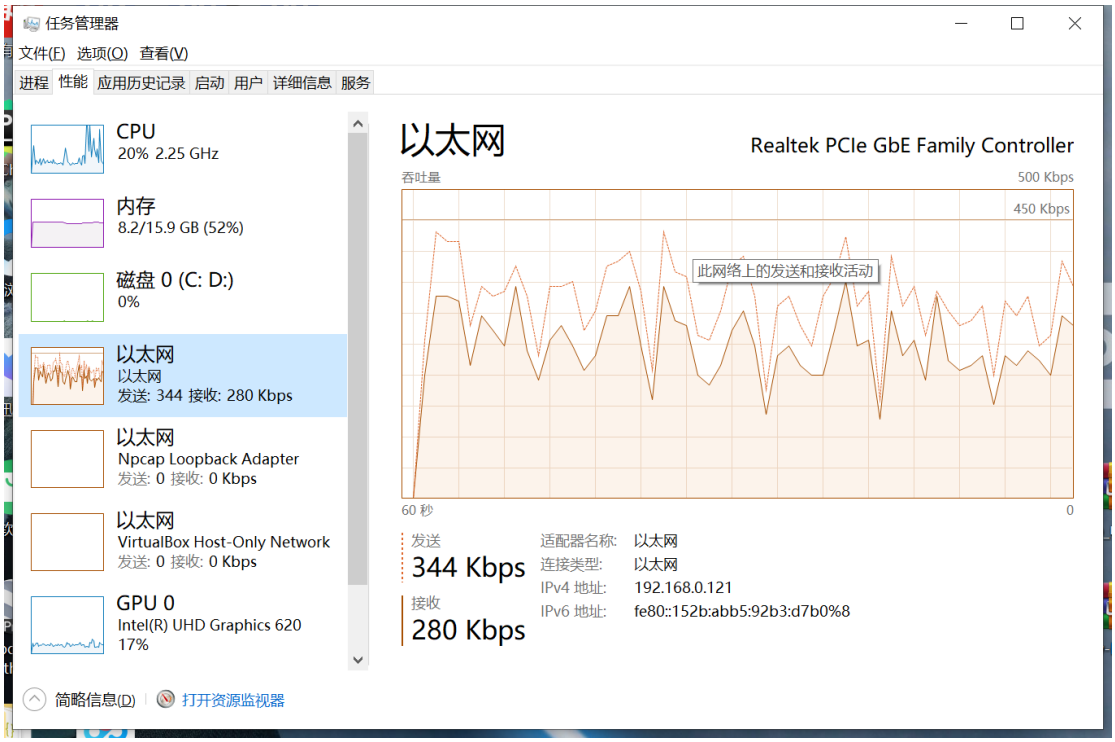
施压机 2 CPU 使用率：施压脚本启动后，客户机 CPU 使用率同样维持在 20%左右。



客户端 1 内存使用率：施压脚本启动后，客户机内存使用率维持在 50%上下。客户端 2 的内存与客户端 1 的基本一样，不再贴图。



客户端网络使用率：施压脚本启动后，客户机网络使用上行加下行维持在 650Kbps 左右，网络环境为局域网 100M 网络，尚有较大剩余。



五、测试总结

在上文所示的硬件环境与单节点部署方案下，同一环境单队列的场景下，Commander 服务器可以支持 500 台 Worker 的调度。需要继续提升 Commander 的调度能力时，可以采取拆分环境，将单环境改为多环境，将任务与 Worker 拆分到多个不同的环境中。也可以采用提升机器性能，更改部署方案等方式来提升调度能力。

名称解释

单环境：一个部门底下可以有多套环境，新建 Worker、流程、任务、计划时，必须指定其归属的环境。Commander 服务端会为每个环境启动一个队列，以支持 Worker 的调度。