南京大學

研 究 生 毕 业 论 文

(申请硕士专业学位)

论 文 题 目	面向私有区块链的自动化性能测试系统
作 者 姓 名	王佩旭
专业学位类别 (领域)	电子信息（软件工程领域）
研 究 方 向	软件工程
指 导 教 师	陈振宇 教授

2022 年 5 月 20日

学　　号： **MF20320159**

论文答辩日期： **2022 年 5 月 20 日**

指 导 教 师： **(签字)**

南京大学硕士学位论文

面向私有区块链的自动化性能测试系统

申 请 人: **王佩旭**

学　　号: **MF20320159**

专　　业: **工程硕士（软件工程领域）**

研究方向: **软件工程**

指导教师: **陈振宇 教授**

南京大学计算机软件研究所

2022 年 05 月

Automated Performance Test System For Private Blockchain

By

Peixu Wang

Supervised by

Professor Zhenyu Chen

A Thesis
Submitted to the Software Institute
and the Graduate School
of Nanjing University
in Partial Fulfillment of the Requirements
for the Degree of
Master of Engineering

Software Institute

May 2022

南京大学研究生毕业论文中文摘要首页用纸

毕业论文题目：面向私有区块链的自动化性能测试系统

工程硕士（软件工程领域）专业 2022 级硕士生姓名：王佩旭

指导教师（姓名、职称）：陈振宇 教授

摘 要

近年来区块链技术受到广泛关注，它所具有的去中心化、不可篡改、可溯源以及自动执行等特点在传统的分布式网络系统中是没有的。但是区块链的性能远没有传统的分布式系统好，而且其性能问题也一直制约着它的快速发展。区块链常部署在不稳定的网络环境中，容易发生未知的故障，且节点间通信需要大量的计算资源，这都导致了区块链的性能问题很严峻。而私有链通过牺牲一定程度的去中心化的能力大大改善了这一问题，提升了区块链系统的性能。在保证了区块链网络的不可篡改、可溯源、自动执行的特性下，其安全性、可靠性也得到了大大增强，并在很多专业领域得到应用。目前，私有链网络的种类繁多，运行原理以及各方面性能也不尽相同，如何准确评估私有链网络多方面的性能显得尤为重要。虽然现有的私有链性能测试工具可以对私有链网络的一些性能做出评估，但是其对使用者的技术要求较高。工具的使用需要复杂的配置输入和私有链网络的手动部署以及测试客户端节点的手动部署和启动，这些流程较为复杂。而且测试结果的展示也不直观，这对用户很不友好。此外，大多数测试工具使用时测试客户端节点和私有链网络节点部署在一起，没有做到与私有链网络运行环境分离的，实现可插拔的测试服务。

本文对 Ethereum、Parity、Quorum 以及 HyperLedger 私有链网络的运行原理进行了深入的研究，选取了吞吐量、交易延迟、容错性、伸缩性、以及网络节点 CPU、内存、带宽的资源消耗指标。同时结合相关性能测试方法与 SSH 免密登录技术，将测试平台服务节点、私有链网络节点以及性能测试节点进行分离，实现了可插拔的自动化私有链网络性能测试服务。根据上述的研究，本文设计并开发了自动化的性能测试系统来系统测试私有链网络。系统的整体架构分为四个模块，分别是配置管理、网络搭建、测试驱动以及结果分析。代码开发使用了 Python、C++、Shell 语言，不同的模块使用与之相合适的语言进行开发。其中测试配置管理和测试结果分析模块使用了 Python 语言进行开发，并引入了 Django 开发框架，数据的持久化使用了 MongoDB 数据库。测试链搭建模块和测试驱动模块都使用了 Shell 和 C++ 语言开发，其中 Shell 语言主要开发相应的自动化运行脚本，测试驱动的开发使用了 C++ 语言。

本文实现的系统可以对多种私有链网络的多方面的性能指标进行准确评估。本文设计了三组实验完成了对该性能测试系统的功能测试。第一组是对比分析了多种私有链网络在正常运行的过程中的吞吐量、交易延迟以及资源消耗率性能指标, 测试结果发现 Ethereum 对 CPU 资源的消耗较大，性能表现较差，而 HyperLedger 对带宽资源的消耗较大，性能表现较好；第二组是对比分析了多种私有链网络吞吐量和交易延迟性能指标受节点故障的影响程度，测试结果发现 Ethereum 和 Quorum 私有链网络的性能指标在节点故障时出现明显的变化趋势；第三组是对比分析了多种私有链网络的性能指标在节点数量不断增大时的表现，测试结果发现 Ethereum 和 HyperLedger 网络的吞吐量指标随着节点数量的增加呈现出先上升后下降的趋势，其中 Ethereum 网络的交易延迟指标呈现出先下降后上升的趋势。实验结果表明，多种私有链网络在上述一个或者多个性能指标方面存在差异，由此可见，面向私有链网络的自动化性能测试系统可以达到预期的效果。

关键词：区块链，私有链，性能指标，自动化，性能测试

南京大学研究生毕业论文英文摘要首页用纸

THESIS: Automated Performance Test System For Private Blockchain

SPECIALIZATION: Software Engineering

POSTGRADUATE: Peixu Wang

MENTOR: Professor **Zhenyu Chen**

Abstract

In recent years, blockchain technology has attracted widespread attention. It has the characteristics of decentralization, imtamability, traceability and automatic execution that are not found in traditional distributed network systems. However, the performance of blockchain is not as good as traditional distributed systems, and its performance problems have been restricting its rapid development. Blockchain is often deployed in an unstable network environment, prone to unknown failures, and communication between nodes requires a large amount of computing resources, which leads to severe performance problems of blockchain. Private chains improve the performance of blockchain systems by sacrificing some degree of decentralization. The security and reliability of the block chain network have been greatly enhanced, and it has been applied in many professional fields, under the guarantee of the immutable, traceable and automatic execution characteristics. At present, there are many kinds of private chain networks, and their operating principles and performance are not the same. How to accurately evaluate the performance of private chain networks is particularly important. Although existing private chain of performance testing tool can evaluate the performance of some private chain network, it has high technical requirements for users. Using the tool requires complex configuration inputs and manual deployment of the private chain network as well as manual deployment and startup of the test client node. And the presentation of test results is not intuitive, which is very unfriendly to users. In addition, most test tools are used when the test client node and the private chain network node are deployed together, not separated from the private chain network operating environment, to achieve pluggable test services.

This thesis conducts in-depth research on the operating principles of Ethereum, Parity, Quorum and HyperLedger private chain networks, and selects throughput, transaction delay, fault tolerance, scalability, and resource consumption indicators of network node CPU, memory and bandwidth. At the same time, the test platform service node, private chain network node and performance test node are separated by combining related performance test methods and SSH secret free login technology, and pluggable automatic private chain network performance test service is realized. Based on the above research, this thesis designs and develops an automatic performance test system to systematically test private chain network. The overall architecture of the system is divided into four modules, which are configuration management, network building, test driving and result analysis. Code development uses Python, C++, Shell language, different modules with appropriate language development. The test configuration management and test result analysis modules are developed using Python language, and Django development framework is introduced, and MongoDB database is used for data persistence. The test chain building module and test driver module are developed by Shell and C++ language, in which Shell language mainly develops the corresponding automatic running scripts, and test driver development uses C++ language.

The system implemented in this thesis can accurately evaluate various performance indexes of various private chain networks. Three groups of experiments are designed to complete the function test of the performance test system. The first group compares and analyzes the performance indicators of throughput, transaction delay and resource consumption rate of various private chain networks in the normal operation process. The test results show that Ethereum consumes more CPU resources and performs worse, while HyperLedger consumes more bandwidth resources and performs better. The second group compares and analyzes the impact of various private chain network throughput and transaction delay performance indicators on node failures. The test results show that the performance indicators of Ethereum and Quorum private chain networks show obvious change trend when nodes fail. The third group compares and analyzes the performance indicators of various private chain networks when the number of nodes increases. The test results show that the throughput indicators of Ethereum and HyperLedger networks increase first and then decrease with the increase of the number of nodes. Among them, the transaction delay indicator of Ethereum network shows a trend of first decreasing and then increasing. The experimental results show that there are differences in one or more of the performance indexes mentioned above for vari-

ous private chain networks. Therefore, the automatic performance testing system for private chain networks can achieve the expected results.

Keywords: Blockchain, Private Chain, Performance Metrics, Automation, Performance Testing

目录

表 目 录

图 目 录

第一章　引言

1.1　项目背景及意义

我们进入数字经济时代，以大数据、人工智能 AI、物联网等为代表的信息通信技术发展迅猛，其中，区块链技术因其颠覆性的创新，近年来越来越得到产业界和学术界的重视。区块是一种链式的数据结构，以它为单位来存储数据，通过严格的时间顺序和复杂的加密算法保证交易顺序以及交易的安全性，区块经过上链确认后，就很难被篡改了，因此，区块链技术给交易提供了一种安全且可验证的方式 [1]。根据特定的用户需求和具体的应用场景的不同，区块链主要分为公有区块链（简称公有链）、私有区块链（简称私有链）和联盟区块链（简称联盟链）。公有链是一种完全去中心化的区块链，任何节点都可以加入到公有链，这也为交易的速度、安全性埋下了隐患。此外，由于公有链节点众多，修改交易数据的难度很大，且要等待大部分节点都就此达成共识，才可以进行到下一步确认阶段，因此公有链的交易速度是很慢的；其次，公有链部署在互联网上，任何人或组织都可以加入到该公有链中，这无疑增加了外网攻击的风险，交易的安全性很难保证。此外，公有链采用了工作量证明的共识机制，导致大量算力的浪费 [2]。针对这些问题，私有链应用而生。私有链采用严格的许可机制，对用户规模进行控制，用户规模控制之后，达成共识速度更快，从而提升了打包交易数据的速度，提升了交易的性能；另外，私有链主要采用了非工作量证明的共识机制，节省了大量算力的浪费，但也保障了隐私和安全，于是发展迅速，遍地开花。在多个领域，私有链因其安全性在众多领域都有应用，因此，为了满足每个应用程序的需求，需要开发多个私有链框架。因此，确保区块链框架为应用程序提供安全性、访问控制和高性能是一个关键的挑战 [3]。现在业内人士也越来越关注私有链的性能问题，那么如何更好地测试私有链系统的性能，从而选择最合适的私有链平台是一个需要研究的问题。现有的区块链私有链性能测试系统的使用需要较高的技术要求，且测试配置的输入较为复杂，私有链网络和测试驱动客户端需要手动部署，测试结果的展示不直观。简单来说，主要是面向实验的，且复杂的操作流程、不友好的交互界面让人望而却步。此外，目前的私有链性能测试系统，测试客户端节点和私有链网络节点部署在一台服务器上，没有做到可插拔的测试服务。我考虑开发一个自动化的面向私有链网络的性能测试系统，实现测试和评估私有链网络的性能。

基于上述情况，本文开发了一个基于私有链网络的性能测试系统，利用私

有链的常用的性能指标，如吞吐量、交易延迟等，来研究不同种私有链网络的性能是否有显著差异。本文开发的系统可以对常见的私有链网络，如 Ethereum、Parity、Quorum 等 [4]，进行吞吐量、交易延迟、资源利用率、容错性以及伸缩性性能测试，并支持测试结果的图表化展示以及原始测试结果数据的下载。因此，本系统可以对不同的私有链网络进行系统的性能对比和评估。

本文使用了性能测试技术，同时引入了性能测试的一些重要的指标，除了一些常用的吞吐量、交易延迟、资源利用率等，还包括容错性、伸缩性等。为了对比不同种私有链网络在容错能力上是否有差异，本文模拟实际场景中节点宕机的情况，在正常性能测试过程中引入节点故障，从而测试多种私有链网络对故障节点的容错性。为了对比不同种私有链网络在伸缩性上是否有差异，本文对多种私有链网络进行节点数量逐渐增加的性能测试，从而有效测试多种私有链网络是否具有良好的伸缩性。

1.2 研究现状

随着区块链技术的不断发展，如何利用区块链技术来替代现有分布式数据库成为了热点问题。区块链技术目前还处于早期发展的阶段，虽然发展迅速，但在性能、监管等方面还存在很多问题 [5]。其中性能问题正是区块链系统面临的主要挑战之一，也是备受业界关注的问题 [6]。对此，本文利用基于私有链的性能测试系统，来评估不同的私有链网络的性能，模拟测试环境中节点发生故障以及增加节点等真实情况，分别对私有链网络的容错性和伸缩性进行测试，从而准确的获取不同私有链网络在容错性和伸缩性方面的差异，得到更加完整准确的测试结果。

性能测试技术是通过自动化的测试工具模拟多种正常和异常负载环境，来对系统的每项指标进行测试，以获取系统的性能瓶颈 [7]。性能测试的主要目的是获取在不同负载环境下的系统性能，使用户能充分了解系统能承受的负载情况 [8]。目前来看，Web 应用的性能测试工具相对成熟 [9]。举例来说，LoadRunner 测试工具，是微软发布的，主要是检测系统在负载情况下的性能情况，进行大规模的企业级测试。另外，Jmeter 也是一款性能测试工具，也可以用于接口测试，Jmeter 的操作简单并且是一款开源软件，主要用来做功能测试和性能测试（压力测试/负载测试） [10]。其次，阿里云开发的 PTS 是可以模拟海量用户的真实场景且具有分布式压测能力的 SaaS 平台，做到全方位的验证业务站点的性能、容量以及稳定性，它将压测工作简化，使测试者将更多精力关注业务和性能本身。

区块链的性能测试方法与一般的分布式系统的测试方法很像，都是通过模拟真实的环境，来给系统中的节点发送一些请求，进行测试，从而生成测试报告

的过程。DENARO G，POLINI A 等主要关注中间件的功能对分布式系统性能的影响，提出了一种新的分布式系统的性能测试方法，从架构设计中得出特定应用的测试用例，使用中间件软件来测试分布式系统的性能 [11]。DINH T，WANG J 等通过简单的 API 集成到 Blockbench，并针对基于合成智能合约的工作负载进行基准测试，且这种方式对于任何的私有链都适用 [12]。P Zheng，Z Zheng 等提出了一个基于日志方法的性能监控框架，该框架能够对区块链系统进行实时且详细的性能监控 [13]。此外，华为在 2018 年发布了 Hyperledger Caliper [1]。目前，Caliper 已经被 Hyperledger 收录为官方的区块链系统性能测试工具。2020 年，C Yuan，J Zhu 等提出了一种新的区块链系统性能测试方案 [14]。Sevindik,V. 等人公开了一种基于区块链 (分布式账本) 的无线网络测试方法。

目前，针对私有链网络的性能测试，主要是面向实验的，且操作流程复杂、耗时耗力，我开发了一个自动化的面向私有链网络的性能测试系统，通过获取私有链常用的性能指标（如吞吐量、交易延迟等），来帮助测试人员全面了解不同种私有链网络的性能是否有显著差异。

1.3 本文的主要工作

本文通过研究不同种私有链网络的多方面性能是否有显著差异，通过获取评估私有链网络常用的性能指标，如吞吐量、交易延迟时间、资源利用率等，对多种私有链的性能进行准确评估和对比分析。为了对比不同种私有链网络在容错能力上是否有差异，本文模拟实际场景中节点宕机的情况，在正常性能测试过程中引入节点故障，从而测试多种私有链网络对故障节点的容错性。同时，为了对比不同种私有链网络在伸缩性方面是否有差异，本文准备了多个服务器节点对多种私有链网络进行节点数量逐渐增加的伸缩性测试，从而有效测试多种私有链网络是否具有良好的伸缩性。

本文根据上述研究实现了面向私有链网络的自动化性能测试系统。首先，根据对本文选中的多种私有链网络本文运行原理的分析，实现了支持自动搭建私有链网络、自动启动测试客户端对待测网络施加工作负载以及自动化结果收集与分析的多种私有链性能测试系统。在该系统中，还包含了一个可视化的平台，主要用于给用户输入配置信息和结果展示。最后，在开发好的平台上针对多种私有链网络进行测试实验和结果分析。实验结果显示，系统可以对常见的私有链网络，如 Ethereum、Parity、Quorum 等，进行吞吐量、交易延迟、资源利用率、容错性以及伸缩性的自动化性能测试，并支持测试结果的图表化展示以及原始测试结果数据的下载。

[1] https://hyperledger.github.io/caliper/

1.4 本文的行文思路

论文共分为六章，行文思路为：

第一章，引言。本章主要介绍了研究课题的技术背景、研究意义和研究现状。

第二章，技术概念介绍部分。本章主要介绍了与课题相关的技术概念。

第三章，需求和架构设计部分。本章首先利用面向目标的需求工程方法定义了系统的各层次目标，并建立目标模型，然后对课题进行涉众分析和用例分析，并对功能需求与非功能需求进行详细描述。最后进行架构设计，并使用 4+1 视图的方式介绍系统设计，并画出了系统涉及到的数据持久化模型。

第四章，设计实现部分。本章主要介绍了各个模块的架构设计和核心代码，画出了各个功能模块的设计类图以及设计顺序图，对于各个模块的关键功能部分进行了相应代码实现。

第五章，实验评估分析部分。本章在面向私有链的自动化性能测试系统中设计并进行了了相关实验，对多种私有链网络进行自动化性能测试、容错性和伸缩性测试，并分析验证了测试结果。

第六章，综述和展望。本章主要对在系统开发与论文写作期间所做的研究工作进行了综述，并分析了项目中存在的不足，归纳梳理了后续项目中的研究重点。

第二章　相关概念与技术

本文中开发的系统是私有链自动化性能测试系统，它是一个以私有链网络为测试对象，对其施加工作负载并收集展示结果信息的自动化测试系统。这一章会重点介绍区块链、私有链、性能测试这三个概念和具体涉及到的技术，以及目前该技术的发展情况。

2.1　区块链

区块链是一种新型去中心化的协议，能安全地存储比特币交易数据以及其它数据，信息不可伪造和篡改，可以自动执行智能合约，无需任何中心化机构的审核 [15]。基于以上优势，近年来区块链技术和区块链的应用得到业界广泛的关注。区块链最早在中本聪的比特币白皮书上提到，但是是以工作证明链的形式存在。后来，比特币系统因其具有去中心化、无法篡改等优质的特性以及多年来的稳定运行而备受大家关注。比特币白皮书中也介绍了 P2P 网络技术、工作量证明协议等，这些都是区块链技术的基础，也是理解区块链技术的前提 [16]。区块链技术的由来也颇有意思，当开发者研究比特币的底层原理时，提取出了数据存储部分，发现其数据结构是一个一个区块首尾相连的，因此就称其为区块链。众所周知，区块链是比特币的核心创新 [16]。区块链的核心优势：区块链结构可以有效的防止双花问题的产生 [17]；修改交易记录的成本非常高；区块链实现了交易和区块两种交易记录，两种记录分别由交易者和称为矿工的单位创建的。后来在 2014 年以太坊出现，它引入了智能合约的概念，完善了区块链的概念 [18]。

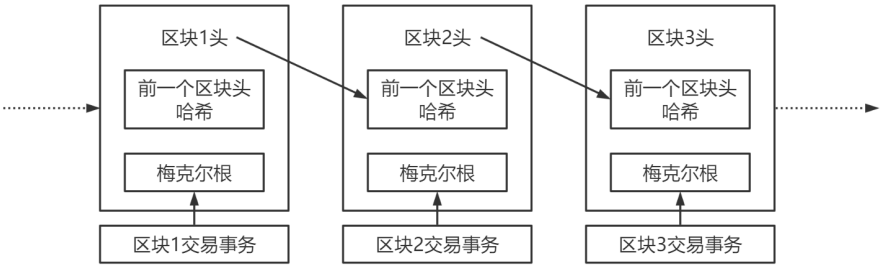

图 2.1: 区块链的结构示图

如图 2.1所示，区块链的形状类似一条链条，按照每个区块产生的时间先后顺序连接在一起。每个区块上存储了信息，如果想要修改某个区块上的信息，必

须征得半数以上节点的同意，同时要修改所有节点中的信息，然而这些节点通常掌握在不同的主体手中，因此区块链很难被篡改 [19]。

根据特定的用户需求和具体的应用场景的不同，区块链主要分为公有区块链（简称公有链）、私有区块链（简称私有链）和联盟区块链（简称联盟链）。比特币、以太坊等都属于公有链项目，它的主要优点是没有加入的限制，相对比较自由 [20]。而私有链项目有 R3 Corda [21] 和 Hyperledger Fabric [22] 等，应用场景主要是对交易效率和隐私安全比较重视的环境下。联盟链的各个节点通常由实体机构组织，加入与退出网络均需授权，去中心化的程度比私有链高。三种区块链的本质区别，在于去中心化程度。

2.1.1 共识机制

区块链技术具有去中心化的特点，与它的共识机制是密不可分的。共识机制算法，主要是基于业务需求，对区块链上各区块的先后顺序达成一致认识 [23]。

共识协议，让区块链上所有节点基于之前商量的协议来达成共识，共同作出决定，这个决定可以是一个数字或一个指令。一般的分布式系统的共识协议，通常由 Master 节点负责统一各节点的共识，这种协议在不稳定的互联网环境中，数据安全性方面有很大问题。然而，区块链一般是部署在不稳定的互联网的环境中的，这种环境下容易发生矿工自私的挖矿等的一系列作恶行为，所以，区块链的共识协议需要能识别出来，达到预防作恶行为的目的 [24]。

区块链中存在四种不同的共识机制，在不同的应用场景下，交易效率和数据的安全性各有所长。工作量证明、权益证明、再到委托权益证明和各种拜占庭容错 [25]，区块链的共识机制不断的发展，也提高了区块链的性能。

PoW，也叫工作量证明，最早是中本聪提出的。这种协议要提前商量好需要解决并验证的问题以此来达成共识，虽然它可以在不安全的环境下使用，但需要耗费大量的算力，且解决难题特别耗费时间 [26]。

BFT 也叫作拜占庭容错协议，这个协议要求区块链上的所有节点之间要通过不断的交流，相互确认一些信息，从而达成共识 [26]。基于以上分析，该共识协议的通信开销是比较大的，但好在不耗费很大的计算资源，通常也是比较常见的共识协议 [27]。所以，这种协议很适合私有链，毕竟节点数量有限 [25]。

PoS，也叫权益证明机制，类似于银行存款根据存款金额和时间计算收益，权益证明是一种根据矿工持币的数量和时间来决定挖矿成功的打包概率的共识协议。POS，对某个矿工进行交易数据的打包工作，倘若该矿工通过不正当的方式来操纵打包概率或骗取挖矿收益，该机制会将他的所有资产没收。该共识协议一般不耗费大量的计算机资源，比如网络或计算资源等，然而存在资产过度

集中的风险，导致出现垄断问题 [28]。

2.1.2 智能合约

所谓智能合约，就是一种运行在区块链上的程序。和普通程序不同的是，智能合约要保证在区块链网络的每一个节点运行的结果完全相同，这样才能使任何一个节点都可以验证区块里执行的结果是否正确，该区块是挖矿产出节点生成的。Nick Szabo 等首次提出了智能合约的概念，并阐述了智能合约是一种计算协议，自此业界开始研究智能合约，早期的的智能合约应用在简单的交易场景下 [29]。

由于智能合约的编程特性，区块链不仅支持单一的交易转账功能，还支持更为复杂的应用场景。以太坊出现后，以太坊区块链成为了第一个支持智能合约的平台，也是智能合约应用的主要场所 [30]。这也是以太坊相比于比特币的一个创新。与纸质合约不同的是，智能合约是通过代码的方式写入到区块链中，一旦满足了要求条件，合约无须人为干预将会自动的执行，这大大得降低了发生违约的概率，为交易提供了良好的保证 [31]。

2.2 私有链

2.2.1 私有链的特点

私有区块链，简称私有链，由一个统一管理者来对区块链上面的各个节点进行管理。私有链采用严格的许可机制，对用户规模进行控制，一方面，达成共识速度更快，从而提升了打包交易数据的速度，也降低通信开销，提升应用的性能；另一方面，也降低了受到恶意攻击或威胁的可能性，对私有链上的数据和隐私起到了很好的保护作用。另外，私有链主要采用了非工作量证明的共识机制，节省了大量算力的浪费，因此，私有链受到很多企业和政府的青睐。总之，私有链作为一种部分去中心化的区块链，与公有链相比具有更快的交易速度、更好的隐私保护、更高的安全性等特点。但是私有链中也有一些问题，少数节点权限过大，会导致中心化问题，与区块链的去中心化思想矛盾 [32]。

2.2.2 私有链的应用场景

虽然说私有链有很多特点和优势，但也不可以盲目的使用，必须要在有合适需求的场景再使用。要做到公私并用的话，具体选择哪一种都需要考虑项目的需求而定。私有链的应用场景通常与企业 [33] 或者政府相关，例如企业的办公流程或者政府部门的财务审批等 [34]。

2.3 性能测试

性能测试技术是利用自动化测试工具模拟各种正常、峰值和异常负载情况，从而对多种性能指标进行性能的测试 [35]。性能测试的主要目的是测试系统能否满足用户对于性能的期望 [8]。

一般来说，测试人员需要对整个软件的质量负责，性能也属于质量的一部分，需要从用户、开发、管理员等各个视角全面的考虑性能：测试人员在做性能测试时既要关注用户看到的表面的现象如响应时间、稳定性等，又需要从开发视角关注本质，比如服务器资源利用率，架构设计是否合理等，同样也需要从管理视角关注系统的容量、系统可扩展性等。

当然，有缺陷的代码设计、不关的线程同步等问题都可能会引起性能缺陷 [36]。

2.3.1 性能测试方法

性能测试根据不同的用户需求和不同的测试目的，可以分为基准测试、负载测试和压力测试，三种测试分别用在不同的环境中。

基准测试是指通过使用科学的性能测试方法和工具，对某类测试对象的性能指标进行定量测试，并对结果进行对比。该测试的主要目的是检验系统性能与相关标准的符合程度 [37]。

负载测试指对被测的系统施加压力，当性能指标超过了预先设定的指标或某种计算机资源已经用完了，从而得出测试系统的性能数据。该测试可以帮助测试人员找到系统的处理数据的极限，为进一步的系统调优提供数据支持。

压力测试，是指通过对软件系统不断得施加压力，识别出系统性能拐点，进而获得系统能够提供的最大服务级别的测试活动。该测试能够测试系统在一定饱和状态下 (CPU/内存等饱和)，系统能够处理的会话能力以及系统是否出现错误 [38]。

2.3.2 性能测试指标

性能测试指标是用来衡量系统性能的一种标准，为了对系统的评估更加全面、客观，需要我们从各个维度、多个指标来评估，主要分为宏观和微观指标 [12]。具体介绍如下：

在宏观指标方面，本系统采用吞吐量和交易延迟指标对私有链网络的事务处理能力进行评估。吞吐量主要用来评估被测试的网络的事务处理效率情况，可以理解为某段时间内，系统能处理的事务数量，在这段时间内，待测网络处理的事务量越大，表示待测网络性能越好。交易延迟用于评估待测网络的时间效率，

可以用交易请求提交到交易请求返回结果之间所间隔的时间来描述，即该时间间隔越小，交易请求正确返回得越快，表示待测网络的时间效率越高，性能越好。

本系统除了考虑吞吐量和交易延迟等测试指标外，也从整体上考虑了容错性和可伸缩性。容错性是通过节点故障时吞吐量和延迟的变化来度量。可伸缩性代表一种弹性，在系统扩展过程中，软件能够保证强劲的生命力，通过增加硬件设备就能实现整个系统处理能力的线性增长，实现高吞吐量和低延迟。可伸缩性是通过增加节点数量和并发工作负载数量时吞吐量和延迟的变化来度量。在本文实验中，我通过增加节点数量，观察吞吐量和延迟的变化情况。

在微观指标方面，本系统采用 CPU 消耗百分比、内存占用百分比以及带宽占用百分比指标来评估私有链网络的资源利用率。CPU 消耗百分比指标主要用来评估待测网络对 CPU 资源的依赖程度，如果测试过程中 CPU 使用百分比较低，则表明待测网络对于 CPU 资源的依赖程度较低，否则较高；内存占用百分比指标主要用来评估待测网络对内存资源的依赖程度，如果测试过程中内存使用百分比高居不下，那么可以认为待测网络对内存资源的依赖层度较高，否则较低。带宽占用百分比指标主要用来评估待测网络对带宽资源的依赖程度，如果测试过程中带宽使用百分比较高，或者占满整个带宽，那么可以认为待测网络对带宽资源的依赖层度较高，否则较低。

2.3.3 性能测试监控

在性能测试客户端服务器和私有链网络节点服务器运行的过程中，需要输出服务器运行的性能数据，例如 CPU、I/O 使用率以及内存、带宽的占用情况等。性能数据的收集和分析可以有效比较不同测试配置的性能测试结果，同时保障整个性能测试的正确运行。本系统在性能测试监控中使用到的工具为 nmon。nmon 是一款可以在各种 Linux 操作系统中广泛使用的监控和分析工具，该工具可以在系统运行的过程中实时、全面地捕捉资源消耗数据，这些数据将会被写到相应的文件中，之后由 nmon_analyzer 分析工具进行分析生成直观的图表。而且 nmon 可以结合 grafana 仪表盘使用，实时直观地查看所要监视的数据。

2.4 本章小结

本章主要介绍了基于私有链的性能测试系统要用到的核心概念与相关技术情况，深入介绍私有链的性能测试相关的概念，还有主要的应用场景以及整体的发展情况。同时，将现有技术和当前流行的区块链技术结合，分析这些技术与区块链的内在联系，为后面的各章节实验设计等做好铺垫。

第三章　需求与概要设计

我研究了 Ethereum [1]、Parity [2]、hyperledger [3]和 quorum [4]等开源项目，以及基于第二章对私有链的运行原理以及性能指标的介绍，本章将对我开发的基于私有链的自动化性能测试系统进行具体的需求分析与基本的概要设计工作。首先完成了对系统的目标分析并建立了目标模型；然后对系统进行涉众分析来掌握该系统的主要涉众情况；接下来是核心功能的用例描述部分，以此了解用户的基本需求情况，从而得到了功能与非功能需求；最后，对系统进行相关的概要设计，并画出系统的整体架构图、4+1 视图与模块设计图。

3.1　需求设计

3.1.1　目标分析

目标分析是在需求工程前期进行，并在非技术层次上建立目标模型定义项目前景与范围的活动。本节首先通过对系统的现状、背景和问题分析来发现高层目标，例如准确评估私有链性能、提高测试人员工作效率、提高应用的去中性化能力等，然后对高层目标的相关信息进行分析，发现其子目标，进而精化结果并完善目标模型，如图 3.1所示。

3.1.2　涉众分析

Ethereum、Parity、Hyperledger 和 Quorum 是目前比较常见的私有链平台类型，个人或企业都可以使用私有链所对应的客户端，以此来搭建私有链网络，并在此基础上开发去中性化的应用。当上层的应用发送请求到私有链网络时，用户希望私有链网络具有很高的性能，以较低的交易延迟时间、较大的吞吐量来处理用户的请求。那么，用户就需要在各种私有链中进行比较，选择更加适合上层应用的私有链网络。目前，各种私有链都已经开放源代码，除了私有链的开发者，越来越多的开源爱好者致力于解决私有链网络在应用领域中产生的性能问题。而且，私有链维护人员也会非常关注性能问题，他们希望在提升用户规模的同时不能降低用户的体验。根据上述对涉众的分析，本节将自动化性能测试系统的涉众分为三种，并介绍了他们的用户特征与期望，如表 3.1所示。

[1]https://github.com/ethereum/go-ethereum

[2]https://github.com/openethereum/parity-ethereum

[3]https://github.com/ConsensusDev/Awesome-Hyperledger-Fabric

[4]https://github.com/ConsenSys/quorum

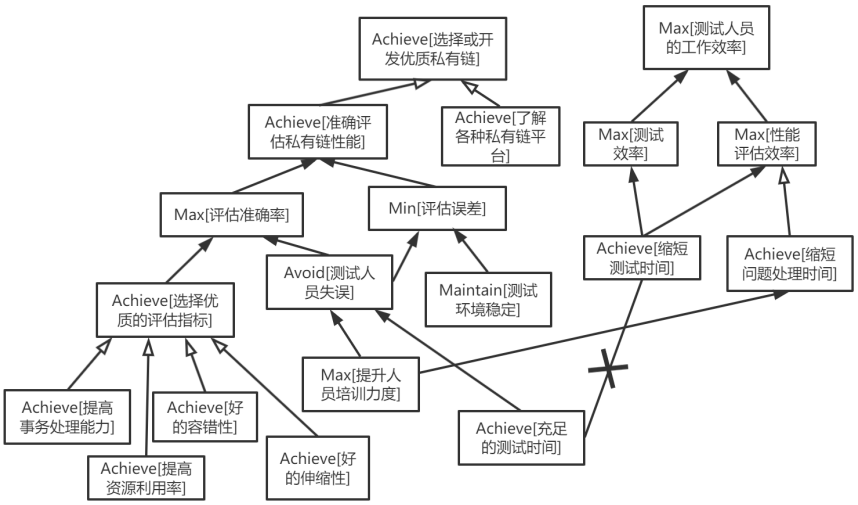

图 3.1: 目标模型图

表 3.1: 涉众的特征与期望

涉众类型	特征	期望
私有链使用者	基于私有链开发 去中心化应用	发送到链上的交易请求 能够快速并正确地被处理
私有链开发者	私有链的开发者， 熟悉私有链运行的原理	在做私有链性能优化时，希望 能直观看出性能优化的结果
私有链维护者	私有链的日常维护人员， 熟悉私有链的部署方式	测试私有链性能 是否符合用户的要求

私有链使用者。以私有链网络为数据载体搭建去中心化分布式应用的用户，了解私有链网络的使用方法。此类用户希望发送到网络上的交易请求能够快速并正确地被处理。同时，在去中心化应用开发的前期，通过对多种私有链网络中做性能分析，获取性能测试结果，了解各个私有链网络的性能特点，从而选择更加适合自身去中心化应用的私有链网络。

私有链开发者。私有链客户端代码的开发者，熟悉私有链的运行原理，他们会通过不断的性能优化和版本的迭代来提升私有链网络的运行性能。通过在本私有链自动化性能测试平台上进行相关性能测试，可以充分分析优化效果。

私有链运维者。私有链网络的运维人员，熟悉私有链网络的部署，对私有链网络有较强的性能需求，想要在增大用户规模的同时不能影响用户的体验。

3.1.3 用例分析

基于前面的分析，本小节将用例描述划分为两个部分，一个是测试任务部分，另一个是测试结果部分，从而画出了系统用例图，如图 3.2所示。

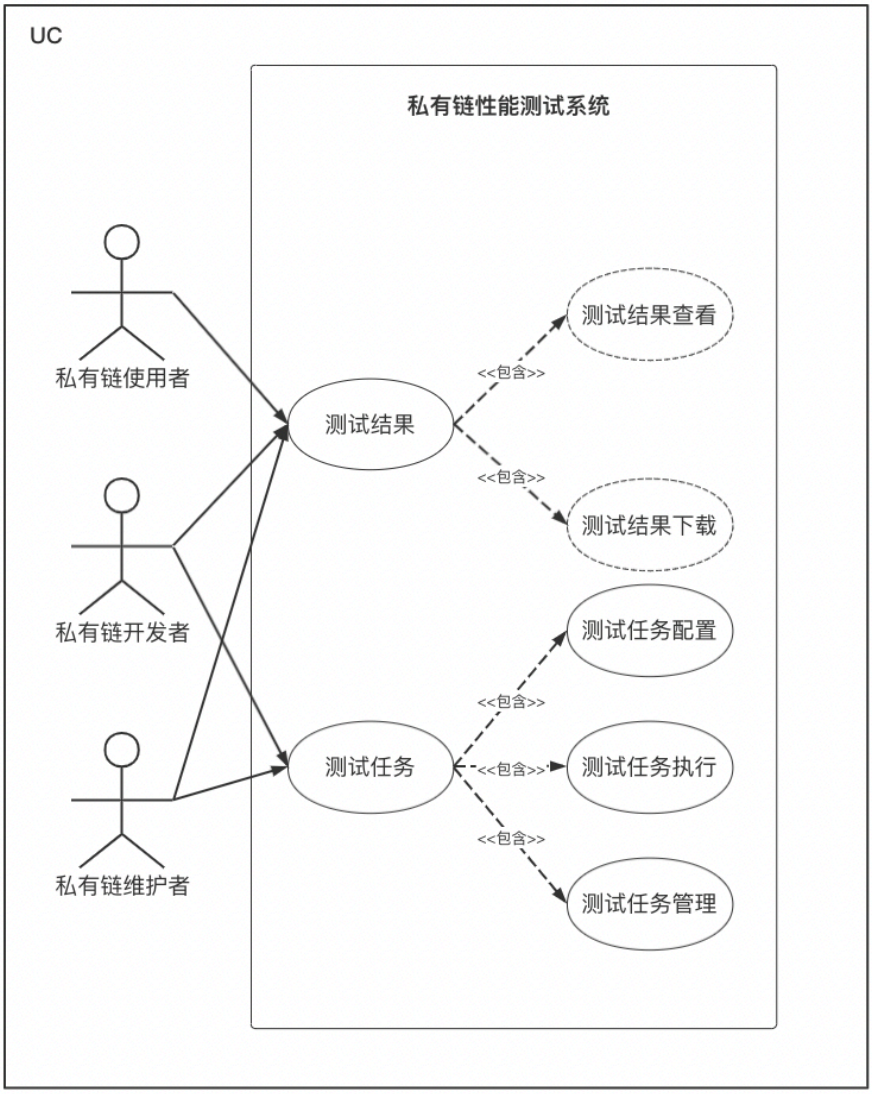

图 3.2: 系统用例图

在私有链自动化性能测试的过程中，首先，私有链的开发者或者维护者可以按照性能测试需求输入测试配置。而且不同业务背景下对私有链网络的性能测试需求一般不同，所以用户输入的测试配置信息也不一样。根据输入的测试配置信息系统开始执行自动化性能测试任务。如果有已经部署好的私有链网络系统，那么在基准配置信息中只需要指定执行测试任务客户端的 IP 地址和私有链系统中每个节点的 IP 地址，本自动化性能测试系统之后会直接进行施加负载、

收集交易结果等操作。如果没有部署好私有链网络系统，那么本自动化测试系统会先进行私有链网络的自动化搭建，然后再进行后序的步骤。最后，系统会将执行成功的测试结果持久化。同时，用户可以对测试结果或者是执行失败的测试任务进行删除操作。结合上述分析，详细的用例分析如下。

表 3.2: 测试任务配置

ID	**UC1**
名称	测试任务配置
参与者	私有链开发者、私有链维护者
触发条件者	私有链开发者、私有链维护者点击创建测试任务按钮
前置条件	用户已注册并登录系统，具有相应的操作权限
后置条件	测试配置信息持久化到数据库中
优先级	高
正常流程	1. 用户点击性能测试配置输入按钮。 2. 系统跳转到配置输入交互界面。 3. 用户逐个填写配置信息。 4. 系统对用户输入的配置进行验证。
扩展流程	3a. 用户输入不符合要求的测试配置。 1. 在不符合要求的配置信息输入框后提示错误信息。

表 3.2为测试任务配置用例的详细描述信息。该配置信息包括节点信息配置、性能测试基准配置以及工作负载配置。私有链节点信息配置为启动或者是已经启动的私有链客户端所在的 IP 地址以及所监听的端口号。性能测试客户端节点信息配置主要是指发送交易请求以及收集请求结果的性能测试客户端所在的 IP 地址，为启动客户端线程做准备。性能测试基准配置为性能测试客户端请求速率、测试持续时间以及开启多少个线程进行测试。工作负载配置是指在执行性能测试之前所需要部署的智能合约，部署完智能合约之后，可以针对智能合约中定义的方法发送交易请求。智能合约是在后端预先定义并编译好的，用户可以选择合适的智能合约对私有链进行性能测试。

表 3.3是详细的测试任务执行用例描述。在执行性能测试任务之前，系统会要求用户输入新的测试配置信息。测试任务启动后，系统会根据用户的配置信息，测试已经部署的私有链网络系统或者测试新搭建的私有链网络系统。接下来，该系统会先验证、解析前端接收到的性能测试基准配置信息，并根据配置信息按照一定的要求给待测网络施加工作负载。同时，性能测试客户端会记录下测试过程中的请求结果数据，并将原始的结果数据持久化到数据库中。测试任务全部是自动化执行的，无需用户手动部署私有链网络或者是部署启动客户端

表 3.3: 测试任务执行

ID	UC2
名称	测试任务执行
参与者	私有链开发者、私有链维护者
触发条件者	用户点击执行性能测试按钮
前置条件	用户已注册并登录本系统，具有相应的操作权限
后置条件	测试结果持久化到数据库中
优先级	高
正常流程	1. 用户填完测试配置信息，并点击开始执行的按钮，执行测试任务。 2. 系统显示测试任务的列表，并开始执行测试任务，其中正在执行的测试任务会显示在列表的最上面。 3. 用户点击测试列表，查看测试任务列表。 4. 系统按照测试任务创建的时间先后显示每一项测试任务。 5. 用户点击测试任务删除按钮，删除对应的测试任务。 6. 系统会从测试任务列表中删除对应的任务。
扩展流程	1a. 系统后端验证用户输入的配置不符合要求。 1. 系统将错误返回到前端页面，并弹框提示用户。

节点。被测试的网络需要进行足够长时间的性能测试，当测试结束后，系统会将测试结果持久化到数据库中，整个测试过程完成。

表 3.4: 测试结果查看

ID	UC3
名称	测试结果查看
参与者	私有链使用者、私有链开发者、私有链维护者
触发条件者	用户点击查看测试结果按钮
前置条件	用户已注册且登录本系统，具有相应的操作权限，并且查看的测试任务状态为已完成状态。
后置条件	无
优先级	中
正常流程	1. 用户点击查看测试任务列表按钮。 2. 系统按照测试任务创建的时间先后显示每一项测试任务。 3. 用户点击列表中某一项已经完成的测试任务的详细结果信息按钮。 4. 系统跳转到该项测试任务所对应的测试结果展示界面并展示相应的测试结果图表。
扩展流程	3a. 用户查看尚未执行成功的测试任务的详细结果。 1. 弹框提示用户测试任务尚未执行成功。

表 3.4是详细的测试结果查看用例描述。测试任务列表会列出正在执行的以

及已经执行完成的所有测试任务。主要的测试配置信息会在测试列表项中显示，用户可以点击测试任务最前面的序号链接获取更加详细直观的测试结果。当测试任务完成之后，才可以获取该结果信息，否则系统会提示错误。测试结果数据会分指标类别以图表的方式展示。性能指标数据的部分统计量会以表格的形式展示出来，包括吞吐量、交易延迟的平均值，中位数。如果测试任务为稳定性测试时，系统会展示故障前后的统计量数据。同时该数据会以柱状图的方式直观展示。动态变化的数据会通过折线图来表示，因为折线图可以形象表示出各项测试指标随着时间变化的情况。因为私有链打包区块的速度具有很强的随机性，所以直接展示原始数据可能会出现大幅度波动，因此会展示经过降噪、平滑处理的测试结果，从而更好地分析私有链网络的性能。

表 3.5: 测试结果下载

ID	**UC4**
名称	测试结果下载
参与者	私有链使用者、私有链开发者、私有链维护者
触发条件者	用户点击下载测试结果按钮
前置条件	用户已登录本系统，具有相应的操作权限，并且查看的 测试任务状态为已完成状态。
后置条件	无
优先级	低
正常流程	1. 用户查看测试任务列表。 2. 系统显示列表页面，默认按照测试任务创建的时间由新 到旧排列每一项测试任务。 3. 用户查看列表某一项测试任务的详细结果信息。 4. 系统跳转到该项测试任务所对应的测试结果展示界面。 5. 用户点击图表上的“保存图片”按钮。 6. 系统开始下载图片到本地。 7. 用户点击图表上的“数据视图”按钮。 8. 系统展示图表中显示的数据供用户查看或者复制。
扩展流程	3a. 使用者点击了未完成的测试任务的详细信息按钮。 1. 平台提示用户测试任务未完成。

表 3.5是详细的测试结果下载用例描述。在测试任务执行成功后，用户可以下载测试结果。用户不仅可以下载测试结果的展示图，也可以下载结果数据做进一步分析。

3.1.4 功能需求

通过对系统进行需求设计，系统的功能需求可以分为如下四个部分：

测试配置填写。私有链使用者、开发者以及维护者会根据各自的性能测试需求准备相应的测试配置。该配置由测试基准信息、私有链网络节点信息以及客户端节点信息组成。这些配置信息会被持久化以便后续测试任务执行部分的使用。

测试任务执行。当私有链使用者、开发者以及维护者输入配置信息后就开始执行测试任务。该部分主要包括待测网络的搭建以及客户端节点的部署和启动。

测试结果可视化。私有链使用者、私有链开发者以及私有链维护者都可以查看已经成功执行的测试任务。测试结果将以表格和图形结合的方式直观展示各方面的测试指标。而且由于私有链网络区块打包交易数据的随机性，原始数据会存在较大幅度的波动，所以原始数据会在做好降噪和平滑处理之后再展示出来。

测试任务管理。私有链使用者、私有链开发者以及私有链维护者对当前用户执行的测试任务可以进行管理操作，例如将执行失败的测试任务从测试任务管理列表中删除。该功能需要管理者具有相应的操作权限。

3.1.5 非功能需求

本系统是为了提高私有链使用者、开发者、维护者对多种私有链网络进行性能测试时的效率。其中系统需要在下面几项非功能需求上做出良好表现。首先，由于测试配置信息多而杂，需要在用户输入配置信息时做出足够的提示和说明，当用户输入不规范后需要给用户提供明确的提示信息，实现良好的人机交互。其次，由于搭建待测网络较为耗时，搭建过程中也存在很多不确定性，可用性一定要保证。最后，由于开源的代码会被不断迭代更新，系统的可维护性也同样不可忽视。

易用性。本文中开发的测试平台，充分考虑到用户的需求，对于配置信息输入界面简洁明了，性能测试执行按钮明显突出，且完全自动化测试平台，以及测试结果的图形化展示，无论对于新手用户还是专家用户，都简单易用。

可用性。由于整个待测网络和测试客户端的运行都比较复杂，对资源的消耗也较高，而且测试流程一环扣一环，如果测试失败了，那么只能重新开始测试，这就大大降低了测试的效率。因此，该系统要实现高可用。

可维护性。由于私有链客户端以及其所依赖的软件还在不断被开发人员更新，可能会出现某些函数的修改或者废弃。因此，系统需要保证可维护性，保证系统中所有功能运行正常高效。

3.2 概要设计

本节在前面需求分析的基础上，进行系统的主要概要设计。首先，对系统进行了架构设计，整体上把握了系统的层次结构。其次，利用 4+1 视图对系统进行多角度详细描述。基于此，我将系统拆为四个模块分别进行设计。

3.2.1 系统架构

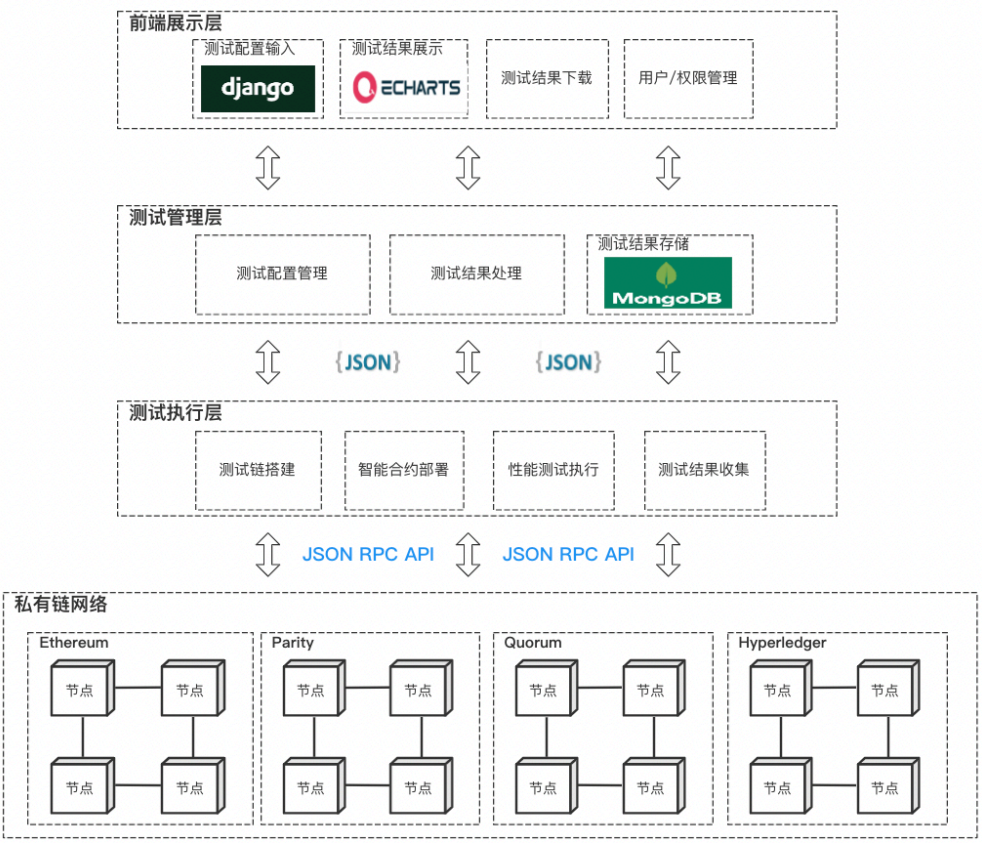

图 3.3: 系统整体框架

本文开发的自动化性能测试系统的架构设计如图 3.3所示。该自动化性能测试系统一共分为四层：前端展示层主要负责展示测试结果数据以及与用户的交互，例如填写相关的测试配置信息，用户的登录、注册以及权限管理；测试管理层会将用户输入的配置信息转换为系统可以直接使用的配置文件，并且将测试结果进行进一步处理，更加利于前端直观地展示。同时，测试管理层会将测试配置信息和测试结果信息进行持久化，以供后期的数据展示；测试执行层负责测试链网络的搭建和测试驱动客户端的启动；私有链网络层主要是运行搭建好的测试

链网络。前端部分通过 Django [5]的模板页面展示图表数据，其中在页面中渲染图表使用了 Echarts.js。后端是使用 Python 语言进行开发，引入了 Django 开发框架。测试管理层也使用 Python 语言进行开发，并使用 NumPy、Scipy 等组件处理数据并对其进行降噪处理。配置文件和测试结果数据都会以 JSON 对象进行存储，以便于字段扩展。数据存储采用 MongoDB [6]数据库，它对非结构化的数据支持较好，读取性能也比较优秀。测试执行层是整个系统的核心，测试链搭建由 SSH 免密登录、NFS 技术以及自动化 Shell 脚本实现；测试驱动程序是由 C++ 进行编写，测试请求的发送以及与私有链节点的交互主要由 RestClient 开源组件支持。

3.2.2 系统执行流程图

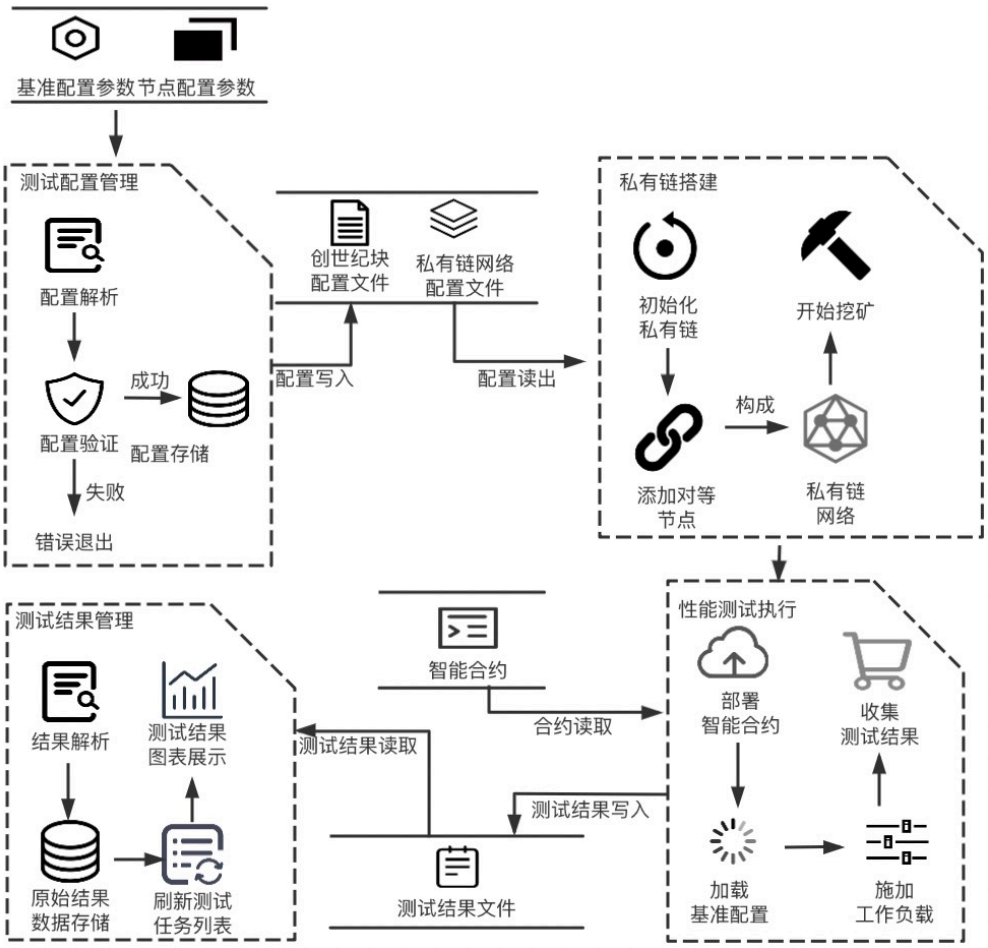

图 3.4: 系统执行流程图

[5]http://www.djangoproject.com/

[6]https://www.mongodb.com/

如图 3.4所示为本文开发的系统的执行流程图，在用户输入正确的测试配置参数后，性能测试系统将按照流程图自动化运行。当用户根据私有链性能测试需求准备好自己的基准配置参数和节点配置参数，测试配置管理服务开始对接收到的参数进行配置解析和配置验证，当配置验证成功后将配置信息存储到数据库中，否则直接报错退出当前测试。解析好的配置会写入到创世纪块配置文件和私有链网络配置文件中。私有链搭建服务开始从这些文件中读取配置数据并开始初始化私有链节点，每个节点开始互相添加对等节点从而构成私有链网络，之后开始挖矿打包交易数据。性能测试执行服务首先会读取并部署智能合约，同时加载基准配置信息并对待测网络施加工作负载以及收集测试结果数据并写入到文件中。测试结果管理模块会从测试结果文件中读取原始的日志数据，并进行结果解析，将解析完成的数据存储到数据库中，同时刷新测试任务列表，此时用户就可以在前端页面查看由测试结果生成的图表以及原始数据。

3.2.3 架构视图

本节将采用 4+1 视图详细介绍系统的架构设计。

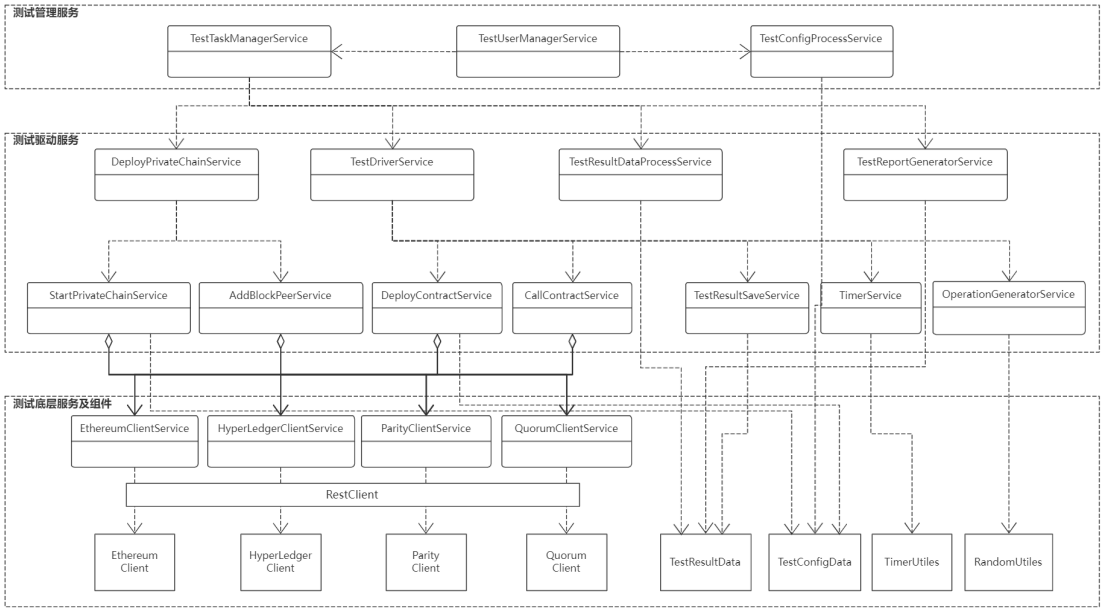

图 3.5: 逻辑视图

逻辑视图。 [39]。图 3.5展示了系统的逻辑视图。逻辑视图主要是用来描述软件的功能逻辑，即软件由哪些模块组成，每个模块包含哪些类，类与类之间的关系。

从图 3.5中可以看出自动化性能测试系统从逻辑上分为三个层次：第一个层次是测试管理层，该层会依赖下一层的服务，会对用户输入的配置信息进行解析和验证，以参数的方式传递给测试驱动层，通过调用下一层的服务实现整个测试管理任务，并且获取该任务的测试结果，以图表的方式展示出来。第二个层次是测试驱动层，如图 3.5所示包括四个服务，而且这几个服务构成了私有链性能测试的整个流程。其中私有链部署服务会进一步依赖测试驱动层的私有链启动服务和添加对等节点服务，测试驱动服务会进一步依赖于测试驱动层的智能合约部署服务、智能合约调用服务、测试结果保存服务、时间管理服务以及操作生成服务。第三个层次是测试底层服务和组件，这一层的服务是指与不同种私有链客户端的连接服务，该连接服务通过 RestClient 开源组件发送交易请求，与相应的私有链客户端进行交互。这一层还包括测试结果数据实体、测试配置信息实体以及一些封装的工具类实体。

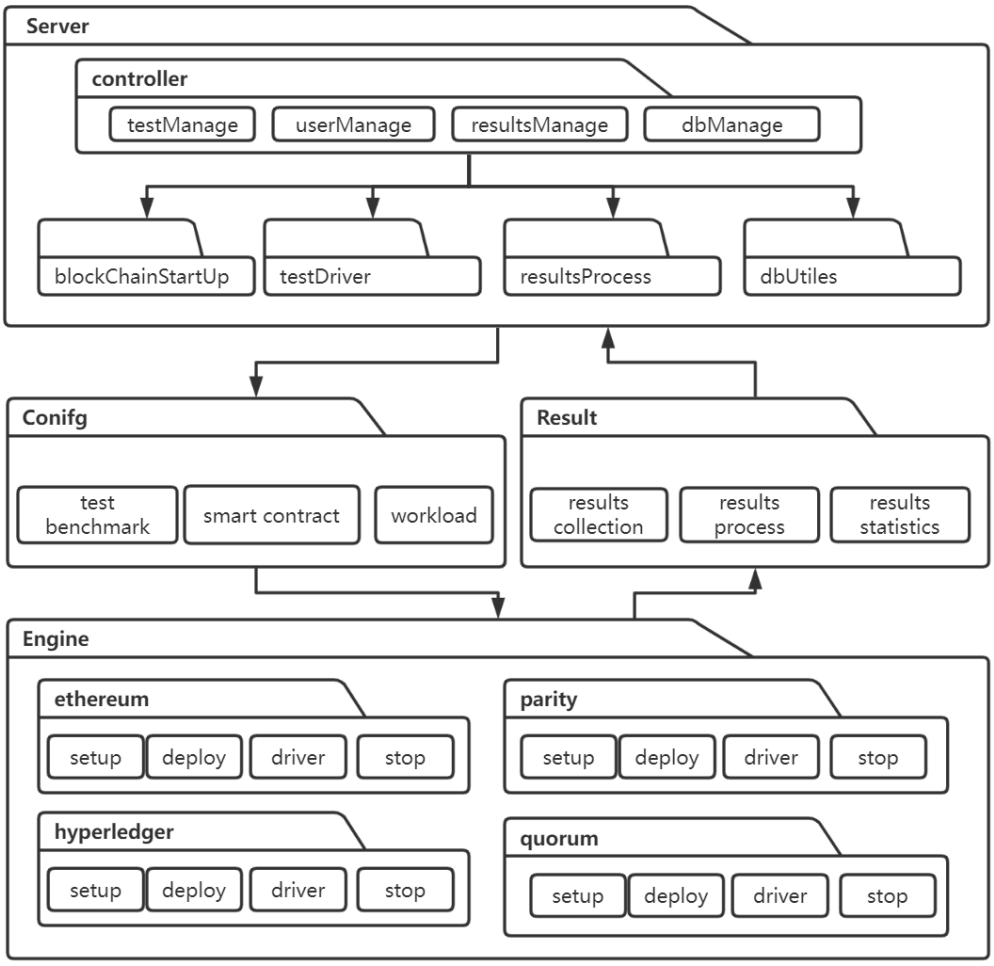

图 3.6: 开发视图

开发视图。本系统开发视图如图 3.6所示。开发视图描述了系统在开发环境下架构的层次划分以及开发包的管理。与逻辑视图类似，从不同的角度看，开发视图中的一个程序包，可能会对应逻辑视图中的一个功能模块。

在图 3.6中，开发视图分为三个层级。最上层为测试系统的服务层，包含了请求控制和业务处理，其中业务处理包括测试配置的管理、用户的管理、测试结果的管理以及数据库的操作管理。Controller 包通过调用私有链的启动包、测试驱动包、测试结果处理包以及数据库操作包完成整个私有链的自动化性能测试的整个过程。其中测试管理包含测试配置管理和测试结果管理，会调用测试配置包和测试结果处理包完成对性能测试配置的生成和结果的处理。性能测试配置包中主要包含性能基准测试文件、智能合约代码以及工作负载文件。测试结果处理包中包含结果收集、处理和统计的功能。处于最下面的一层是私有链性能测试引擎层，对于不同的私有链平台启动的方式或者是智能合约部署的方式不太一样，所以会分为不同的包进行管理。每个引擎包中都会包含启动、部署、驱动和停止这几个包，这就是从启动测试链到最终收集到数据之后停止测试链的整个过程。这四个不同的包各自完成各自的职责。

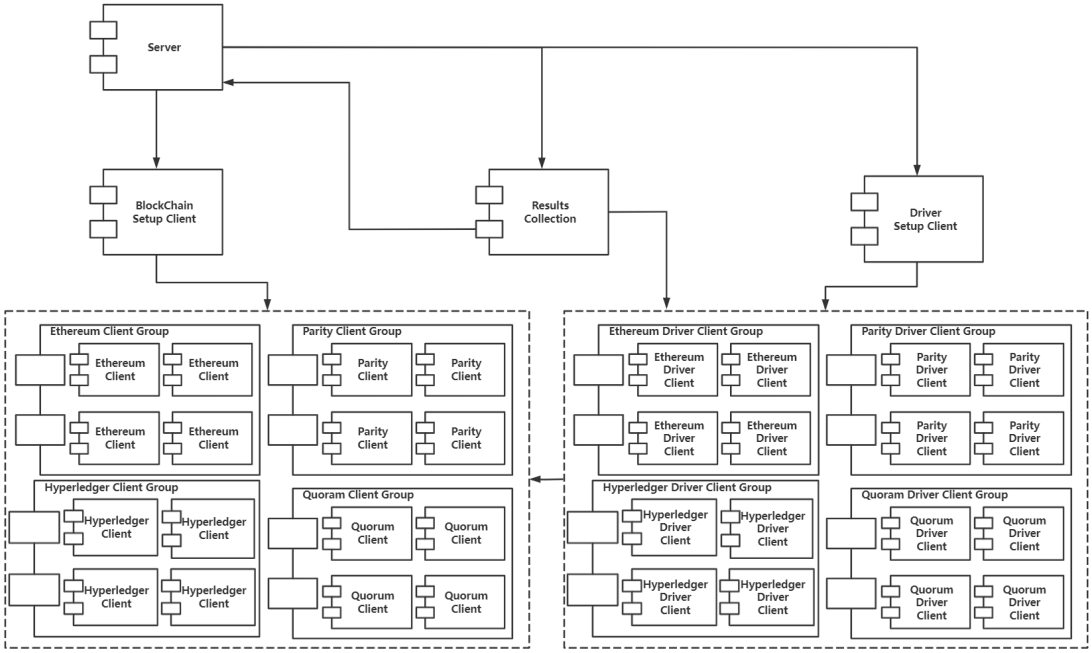

图 3.7: 进程视图

进程视图。进程视图主要关注进程、线程等运行时的概念，以及相关的并发、同步、通信等问题。

图 3.7展示了系统的进程视图。如图 3.7所示，在服务器后端主进程收到前端发送的性能测试请求后，就会开始执行性能测试任务，性能测试的配置信息会以 JSON 字符串的格式传递给私有链网络启动进程和驱动进程。测试开始后，主进程会启动搭建私有链网络、测试驱动以及结果解析这三个进程。私有链网络启动进程会根据配置通过 SSH 免密登录到各个服务器节点执行预先编写好的私有链网络的启动脚本，来启动相应的私有链网络，由于配置了多个私有链运行节点 IP，所以会产生一个私有链客户端进程组，每一个进程都会添加其他节点为自己的 Peer 节点，从而形成一个私有链网络。私有链启动进程在私有链网络形成之后其生命周期就结束了，测试客户端驱动启动进程开始通过 SSH 免密登录到客户端节点执行驱动启动的脚本，启动性能测试启动进程组。测试任务执行完成后，数据处理进程开始对结果进行解析处理，然后返回给主进程并以图表方式展示。

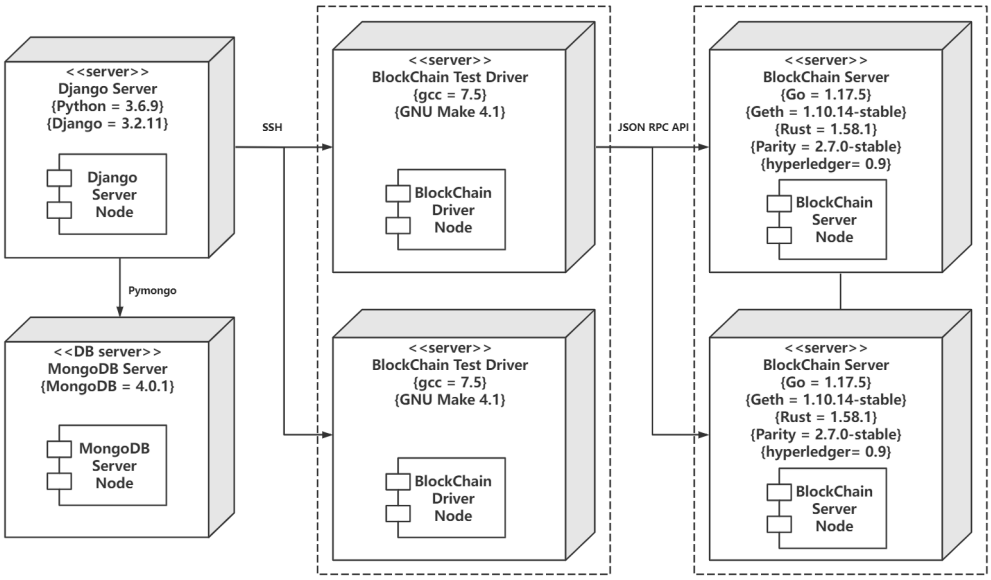

图 3.8: 物理视图

物理视图。物理视图关注的是目标程序在物理服务器上的部署方式以及在不同的服务器之间的通信方式。

图 3.8展示了该系统的物理视图。图中的方块代表物理机，从左到右为 Django、数据库、测试客户端驱动以及私有链部署节点服务器。Django 通过和 MongoDB 数据库连接存取测试配置和结果。Django 使用 SSH 免密登录到测试客户端驱动服务器和私有链节点服务器执行准备好的脚本。驱动服务器通过封装 Post 请求调用测试链的 JSON RPC API 与测试链进行数据和指令的交互。由于私有链网

络对资源的消耗比较多，而且方便后续资源消耗数量的统计，每一个部分都最好分开部署在不同的服务器上。

3.3 系统模块设计

本节将介绍系统的模块设计，系统主要划分为四个模块：测试配置、测试链搭建、测试驱动以及测试结果模块。架构设计如图 3.9所示。其中，测试配置模块会接受、处理并存储前端传递的配置信息，该模块会使用 JSON 字符串以及命令行调用的方式与其他模块通信；建立待测网络主要由第二个模块负责，该模块会初始化私有链网络、互相添加对等节点以及启动挖矿节点，如果测试后端服务与私有链网络服务部署在不同的服务器节点上时，可以通过 SSH 免密登录执行预先写好的启动脚本；测试驱动模块主要负责对已经搭好的测试链部署智能合约、施加工作负载以及收集测试结果信息，该原始结果信息会写入文件，待测试结果管理模块读取并做分析。测试结果管理模块主要是对测试驱动模块生成的测试结果文件进行解析并做持久化。

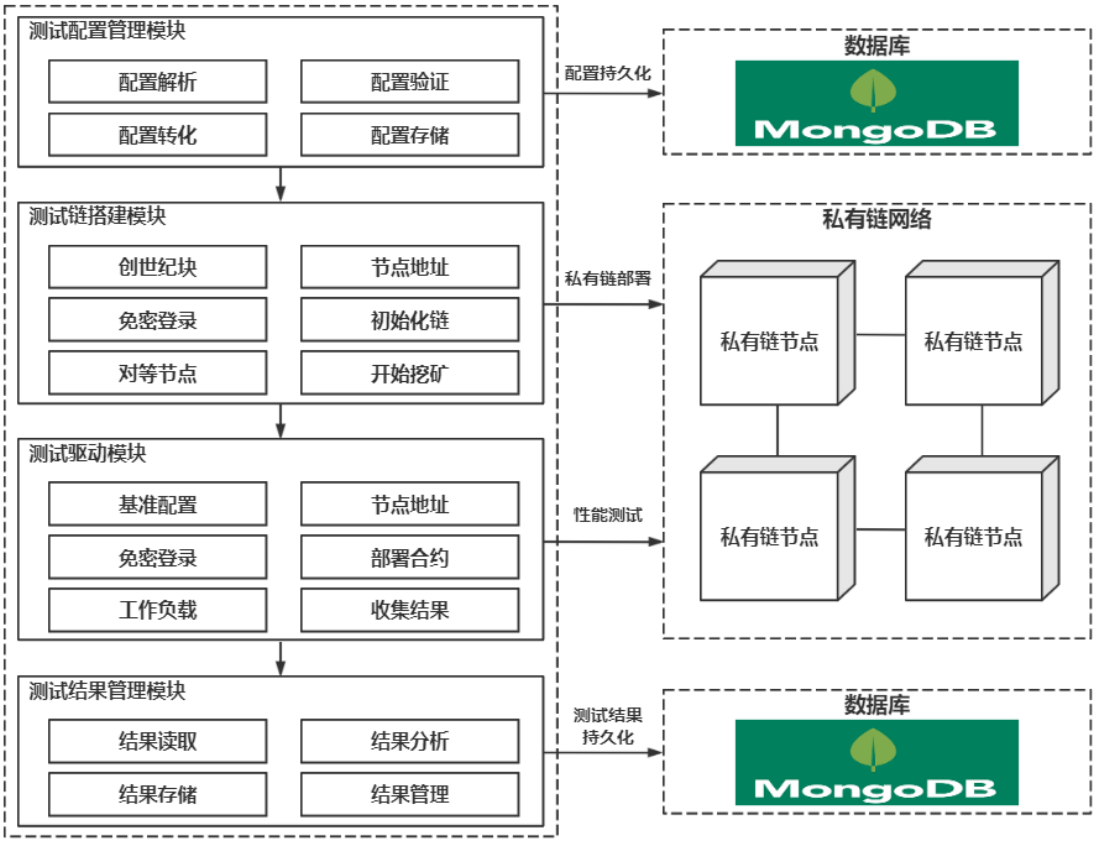

图 3.9: 系统模块架构图

3.3.1 测试配置管理模块

测试配置管理模块主要负责处理与持久化测试配置信息。如图 3.9所示，本模块主要负责解析、验证、存储用户输入的配置，并根据配置信息执行后续的测试任务。测试配置管理模块运行时首先从前端接收用户的配置信息，然后读取、验证、解析这些信息，如果发现配置信息不满足要求，则向前端发送执行失败信息，提示用户重新输入。正确的配置信息会以 JSON 对象的形式持久化到数据库中，供后续执行模块的使用。

本模块将采用 MongoDB 进行数据持久化，它本质上是一种非结构化数据库，擅长处理非结构化的数据，而且性能很强。本文选取这种数据库的主要原因是系统中需要存储的测试配置信息与测试结果数据的数据结构比较复杂，而且数据的层级关系明显，再者就是前端通过 JSON 字符串格式向后端发送数据。在测试任务完成后，测试配置信息与测试结果信息会存在数据库的同一个对象中。在测试配置管理模块中，每一个配置的 ID 都会由 Python 内置的 UUID 组件生成，而且会以参数的方式传递给之后执行的模块，之后执行的模块会通过该唯一性 ID 获取此次性能测试所需的配置进行后序任务的执行或者测试结果的保存。具体的持久化设计如表 3.6所示。

表 3.6: 测试任务字段设计

字段名	类型	描述
id	String	测试任务唯一识别 ID
user	String	测试任务创建人名称
blockChain	String	测试链的类型
smartContract	String	部署的智能合约的类型
requestsRate	Integer	每秒请求的事务的个数
threads	Integer	执行驱动程序的线程数量
duration	Integer	执行性能测试的持续时间
status	String	性能测试的状态
startTime	Integer	性能测试开始的时间
clients	Arrays	执行驱动程序的节点的 IP 地址列表
hosts	Arrays	部署测试链网络的节点的 IP 地址列表
setupPrivateChain	Boolean	是否需要搭建私有链网络
stableTest	Boolean	是否进行故障测试
results	Arrays	性能测试结果，见表 3.7

表 3.6展示了测试任务的持久化设计，其中包括私有链类型配置、智能合约配置、性能测试基准配置、驱动节点 IP 地址、部署节点 IP 地址以及测试结果等信息。目前，blockChain 字段包括 Ethereum、Parity、Quoram 和 Hyperledger 这

四种私有链类型；smartContract 字段包括 HelloWorld、YCSB 以及 SmallBank 这三种智能合约类型；status 为性能测试的状态字段，有 pending 和 finished 两种类型；clients 和 hosts 字段为 IP 地址列表，每一个 IP 地址之间由“,”隔开。另外，results 字段为结果数据，为嵌套的 JSON 格式文档，下面将会以表格的方式详细介绍这个字段。

3.3.2 测试链搭建模块

如图 3.9所示，测试链搭建模块主要用于部署一个待测私有链，其中包括使用创世纪块定义文件初始化私有链、创建新账户、获取对等节点的信息并构建私有链网络以及启动挖矿节点。这些都是通过预定义脚本的方式执行，如果测试配置管理模块和本模块部署在不同的服务器节点上，那么可以通过 SSH 免密登录执行相应的自动化脚本。私有链自动化搭建脚本如图 3.10所示。

```
#!/bin/bash
# 初始化私有链并创建新账户
geth --datadir=$ETH_DATA init $ETH_HOME/Genesis"_"$1".json"
geth --datadir=$ETH_DATA --password <(echo -n "") account new

# 启动挖矿节点并循环添加对等节点
nohup geth --datadir=$ETH_DATA --nodiscover --http --http.addr 0.0.0.0 --
http.port "8000" --http.corsdomain "*" --miner.gasprice 0 --maxpeers 32 --networkid
9119 --unlock 0 --password <(echo -n "") --mine --miner.threads 8 --allow-
insecure-unlock > $ETH_DATA/../eth_log 2>&1 &

for com in cat $ETH_HOME/addPeer.txt; do
  geth --exec $com attach ipc:$ETH_DATA/geth.ipc
done
```

图 3.10: 私有链自动化搭建脚本代码

3.3.3 测试驱动模块

如图 3.9所示，测试驱动模块主要是用于根据基准配置信息部署智能合约、对待测私有链施加工作负载以及收集性能测试的结果。这些都是通过预定义脚本的方式执行，如果测试配置管理模块和本模块部署在不同的服务器节点上，那么可以通过 SSH 免密登录执行相应的自动化脚本。相应的测试驱动启动脚本代码如图 3.11所示。

```
#!/bin/bash
# 根据基准配置启动性能测试驱动程序
LOG_DIR=$LOG_DIR/exp_$3"_"servers_$1"_"threads_$4"_"rates
mkdir -p $LOG_DIR
i=0
for host in cat $HOSTS; do
  let n=i/2
  let i=i+1
  if [[ $n -eq $2 ]]; then
    cd $EXE_HOME
    pwd
    echo $BENCHMARK
    if [ "$BENCHMARK" = "ycsb"]; then
      echo "Begin ycsb test!"
      nohup ./driver -db ethereum -threads $1 -P workloads/workloada.spec -endpoint
$host:8000 -txrate $4 -wt 60 -wl ycsb > $LOG_DIR/client_$host"_"$1 2>&1 &
    elif [ "$BENCHMARK" = "smallbank"]; then
      echo "Begin smallbank test!"
      nohup ./driver -db ethereum -ops 1000 -threads $1 -endpoint $host:8000 -txrate $4
-wt 60 -wl smallbank > $LOG_DIR/client_$host"_"$1 2>&1 &
    else
      echo "Begin helloworld test!"
      nohup ./driver -db ethereum -P workloads/workloada.spec -threads $1 -endpoint
$host:8000 -txrate $4 -wt 60 -wl donothing > $LOG_DIR/client_$host"_"$1 2>&1 &
    fi
  fi
done
```

图 3.11: 性能测试驱动启动脚本代码

3.3.4 测试结果管理模块

表 3.7展示了测试结果的持久化设计，对应测试任务设计表中的 results 字段。测试结果主要包括测试时间点、吞吐量、交易延迟、节点 CPU、内存以及带宽的资源消耗测试指标。其他如平均值、中位数等统计量可以通过数据库中处理完成的结果数据计算得到。其中，测试结果的处理方式主要包括降噪和平滑处理。

表 3.7: 测试结果字段设计

字段名	类型	描述
time	Arrays	区块打包时间点
throughput	Arrays	吞吐量
latency	Arrays	交易平均延迟
cpu	Arrays	私有链网络中所有节点的 CPU 消耗统计信息
mem	Arrays	私有链网络中所有节点的内存消耗统计信息
net	Arrays	私有链网络中所有节点的带宽消耗统计信息

3.4 本章小结

本章主要介绍了本性能测试系统的需求分析与概要设计两个阶段。在需求分析阶段，首先结合项目背景，建立了目标模型，然后对系统的涉众进行分析，并以表格的方式介绍了涉众的特征和期望，同时画出了用例图，之后又列出了系统的部分详细用例，总结了本系统的相关的功能需求和非功能需求。接下来，在系统概要设计阶段，结合 “4+1 视图” 介绍了本性能测试系统的整体架构，并详细分析了 4 种视图，包括逻辑视图、开发视图、进程视图、物理视图在本系统中的应用。最后，介绍了系统的模块设计，将该自动化性能测试系统从整体上划分为四个模块，然后详细介绍各模块的概要设计以及它们之间的关联。

第四章　详细设计与实现

本章将根据第三章进行的需求分析与概要设计，介绍系统的详细设计与实现。系统总共包括四个模块，分别是配置信息管理模块、测试链搭建模块、测试驱动模块以及结果信息管理模块。首先，本章介绍了各个模块的详细设计思路以及用到的关键技术，然后介绍并画出了各个模块的设计类图和设计顺序图。最后，根据各个模块的核心功能完成代码实现。

4.1　测试配置管理模块

4.1.1　测试配置管理模块设计

测试配置管理模块主要是负责处理并验证用户的配置输入，将用户输入的配置转化为测试链搭建模块或者是测试驱动模块所需要的配置格式，并生成相应的测试任务。在此模块中会对每一个测试任务生成唯一的 ID 号，贯穿测试的整个过程，以便于后序各个模块的测试配置信息的获取和对应测试结果的保存。测试配置信息主要包括私有链类型和智能合约类型的选择、性能测试基准配置信息以及执行性能测试和运行私有链节点的 IP 地址列表。用户通过页面输入以上配置信息，服务端读取配置信息并转换为配置文件，通过运行预先编写好的脚本代码启动测试。

该模块使用了 Python 语言进行开发，并使用了业界主流的 Django 框架，主要考虑到测试结果数据都是和时间相关的向量，并且需要进行进一步的降噪和平滑处理。Django 是一个开放源代码的 Web 应用框架，它采用了 MTV 的框架模式，即模型 M（数据存取层），视图 V（业务逻辑层）和模版 T（表现层）[40]。使用这种架构可以方便、快捷地创建高品质、易维护、数据库驱动的应用程序。而且在 Django 框架中，还包含许多功能强大的第三方插件，使得 Django 具有较强的可扩展性 [40]。

在整个私有链的自动化性能测试过程中，由于测试配置信息与测试结果数据会被用户经常查看和对比，所以，二者会整合在一起并保存在数据库中。测试配置信息会以 JSON 字符串的格式从前端传递到后端，字段之间会存在层级关系，属于非结构化的数据格式。因此，本测试系统不再考虑使用一般的关系型数据库，而是使用读写性能较号的 MongoDB 非关系型数据库。

如图 4.1所示为测试配置管理模块的类图。用户通过前端页面输入配置信息，通过 Post 请求将配置数据发送到 Django 服务器，服务器端会通过 URL 映射文

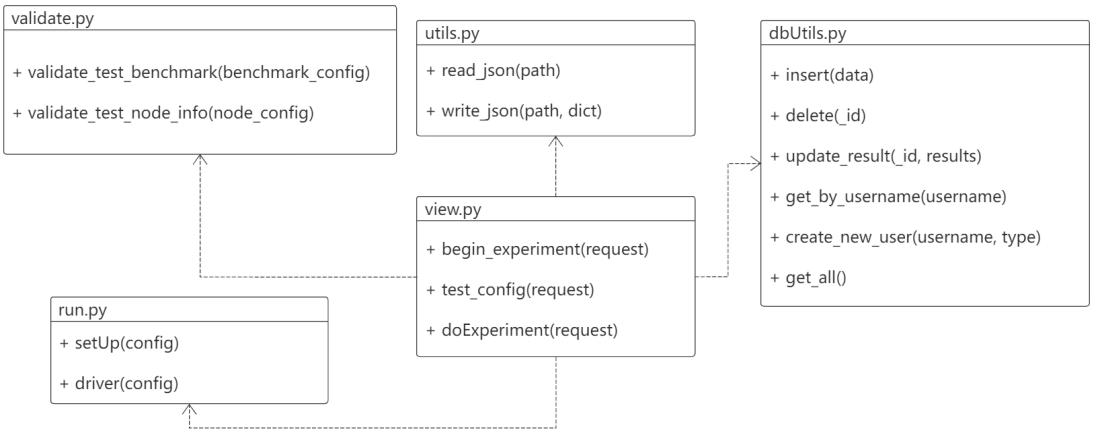

图 4.1: 测试配置管理模块类图

件 urls.py 找到 Controller 组件 view.py 中处理该请求的视图函数。在视图函数中，会调用 validate.py 组件中的 validate_test_benchmark 函数验证基准配置数据的合法性，也会调用 validate.py 组件中的 validate_test_node_info 函数验证节点配置数据的合法性，如果所有配置信息都合法，则进入后序处理环节，如果发现任何一种配置数据不符合规范，通过返回提示信息给前端让用户重新输入配置信息。在 utils.py 组件中，主要提供了对于 JSON 文件的读取和写入的函数，在 view.py 组件中，会调用 utils.py 组件中的 read_json 函数读取预先定义好的配置数据和结果数据的格式。同时，为了配置信息和结果数据的一一对应，在对配置信息进行处理时，也会通过 Python 的 UUID 组件给该配置信息生成全局唯一的 ID，以便于后序组件对于配置信息的获取或者对应测试结果的保存。最后，所有得配置信息都会持久化到 MongoDB 数据库中，dbUtils.py 组件主要用于封装对 MongoDB 数据库的增删改查操作，其通过 Python 提供的第三方组件 Pymongo 连接到 MongoDB 数据库，获取相应的集合并进行数据操作。类图中的 run.py 组件主要负责私有链节点的启动和性能测试驱动程序的启动，在所有配置信息都验证通过并持久化到数据库后，将通过 view.py 组件中的 doExperiment 函数调用该组件中的函数。

如图 4.2所示为测试配置管理模块的顺序图。用户依次输入基准测试配置信息和节点配置信息后，系统会对所有的配置数据进行验证。如果所有的配置验证通过后，系统会为该配置生成唯一的 ID 号，并且将配置数据持久化到 MongoDB 数据库中，然后通过调用 run.py 组件中的私有链节点启动函数和性能测试驱动程序启动函数开始对私有链进行性能测试。由于这个测试时间会比较长，所以系统会异步启动测试驱动程序。最后系统会返回性能测试开始执行的信息，并

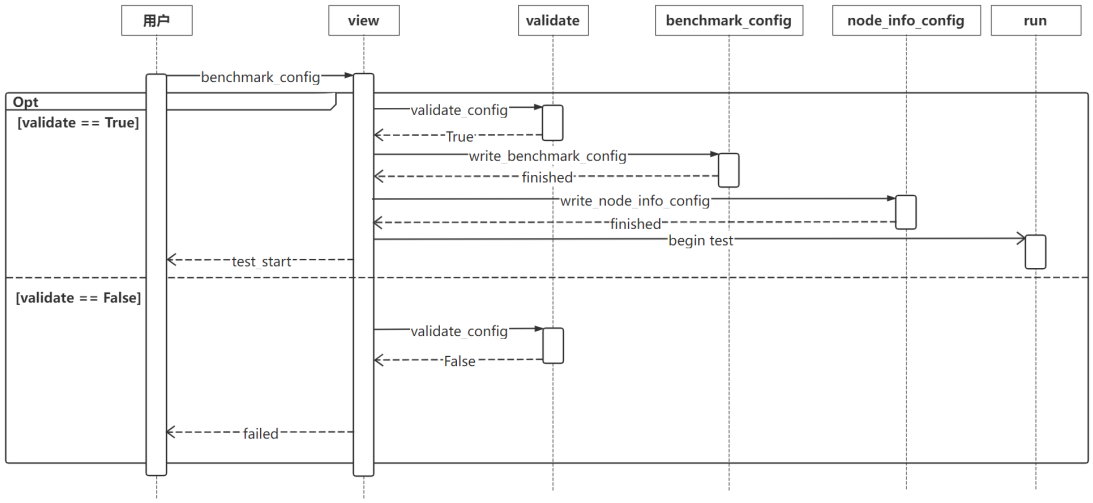

图 4.2: 测试配置管理模块顺序图

且前端页面测试任务列表中的测试任务的状态显示在测试进行中。在整个测试进程结束后，会唤醒结果处理和测试状态更新的进程，开始对收集到的结果进行处理并更新测试状态为已完成，并持久化到数据库中。如果有任何一项配置数据不符合规范，则无法进行性能测试，而且配置信息也不会持久化到数据库中，系统会返回给前端一个“用户配置信息错误”的信息，要求用户重新输入。

4.1.2 测试配置管理模块实现

测试基准配置和节点信息配置视图的代码实现如图 4.3所示。系统会从参数中获取用户输入的配置信息。在第 3 行中，系统首先会判断当前的请求是否为 POST 请求，并从请求中获取性能测试的基准配置信息，如私有链类型、智能合约的类型、请求速率、测试执行时间等等以及节点的 IP 地址信息，如客户端节点的 IP 地址和私有链部署节点的 IP 地址。然后从第 12 行开始，系统将这些信息都封装到 benchmark_config 和 node_info_config 对象中。接下来，从第 17 行到 23 行，系统开始对用户输入的配置数据进行验证，如果验证失败直接跳转到执行失败模板页面。如果所有的配置信息都验证通过，系统则会生成一个该测试任务的唯一 ID 号，该 ID 号会随着配置数据一起持久化到 MongoDB 数据库中，作为每一个 JSON 对象的 ID。持久化操作在第 27 行通过调用 dbUtils.py 组件中的相应函数实现。最后从 29 行开始，系统开启两个线程，一个线程用于调用私有链的性能测试自动化脚本开始搭建私有链并启动测试驱动程序；另一个线程用于更新性能测试的结果和状态信息，后一个线程的执行会等待前一个线程的通知，否则会挂起不继续执行。

```
@login_required
def test_config(request):
    if request.method == "POST":
        blockChainType = request.POST.get("blockchainType", None)
        smartContract = request.POST.get("smartContract", None)
        requestRate = int(request.POST.get("requestRate", None))
        threads = int(request.POST.get("threads", None))
        duration = int(request.POST.get("duration", None))
        tempClients = request.POST.get("testClients", None)
        tempHosts = request.POST.get("testHosts", None)
        ...
    benchmark_config = utils.read_json('static/json/config.json')
    benchmark_config['user'] = username
    benchmark_config['blockChainType'] = blockChainType
    benchmark_config['smartContract'] = smartContract
    ...
    if not validate.validate_benchmark_config(benchmark_config):
        return render(request, 'testConfig_py.html', {'err_msg': 'Invalid benchmark config.'})
    node_info_config = [...]
    if not validate.validate_failure_config(node_info_config):
        return render(request, 'testConfig_py.html', {'err_msg': 'Invalid node info config.'})

    id = str(uuid.uuid4()).replace("-", "")
    benchmark_config['_id'] = id
    dbUtils.insert(benchmark_config)
    event = Event()
    t1 = Thread(target=do_experiment, kwargs={"event": event, "blockChainType":blockChainType, "smartContract":smartContract, "requestRate": requestRate, "duration": duration, "tempClients": tempClients, "tempHosts": tempHosts})
    t1.start()
    t2 = Thread(target=update_status , kwargs={"event": event, "id": id, "blockChainType": blockChainType, "smartContract":smartContract, "requestRate": requestRate, "threads":threads, "tempHosts": tempHosts})
    t2.start()
    return render(request, 'testConfig_py.html', {'msg': 'All config valid, per test begin!'})
```

图 4.3: 测试配置处理函数的实现代码

如图 4.4所示的代码为执行性能测试的函数实现代码，在第 5 行中，系统开始执行提前编写好的性能测试的自动化脚本，相关的配置信息会以参数的形式

传递到后序的测试过程中。

```
@login_required
def doExperiment(event, blockChainType, smartContract, requestRate, threads, duration, tempClients, tempHosts):
    ...
    run_cmd = "python ./run.py {} {} {} {} {} {} {}"
    os.system(run_cmd.format(hn, tempHosts, tempClients, threads, requestRate, smartContract.lower(), duration))
    event.set()
```

图 4.4: 执行性能测试的函数实现代码

4.2 测试链搭建模块

4.2.1 测试链搭建模块设计

测试链搭建模块主要负责接收测试配置管理模块生成的测试配置信息、解析并完成测试链的搭建与初始化。如果待测的私有链网络已经搭建完成，则直接进入到测试驱动程序的执行。如果需要根据用户提供的节点搭建私有链网络，那么测试链搭建模块会被执行。该模块会通过远程免密登录的方式登录到私有链部署节点执行预先编写好的私有链启动的脚本。脚本中会按照顺序执行私有链的准备脚本、私有链的初始化脚本、私有链的启动启动、获取节点信息脚本以及互相添加对等节点构造私有链网络的脚本。当所有的测试链搭建脚本运行结束后，会产生一个私有链网络，节点之间的产生共识的方式会随着私有链类型的不同而有差异。虽然每个私有链节点部署在不同的机器上，但是可以通过 NFS 网络文件系统让多个不同的节点共享文件或者目录，实现不同节点测试脚本的一致性，同时也可以减少文件的冗余。

如图 4.5所示为测试链搭建模块的类图。该模块的关键功能是实现 run.py 组件的五个方法，分别是 prepare、init_all_node、start_all_node、start_all_client 和 partition。prepare 函数是用来获取和解析用户输入的基准配置信息和节点配置信息，并将节点配置信息中的节点 IP 地址写入文件中。同时，系统会根据不同的智能合约的类型生成不同的环境变量，其中包括私有链启动所需的数据目录和私有链运行所需的日志存储目录等等。init_all_node 函数主要是实现每一个私有链节点的初始化，这里主要是初始化创世纪块、初始化私有链运行所需

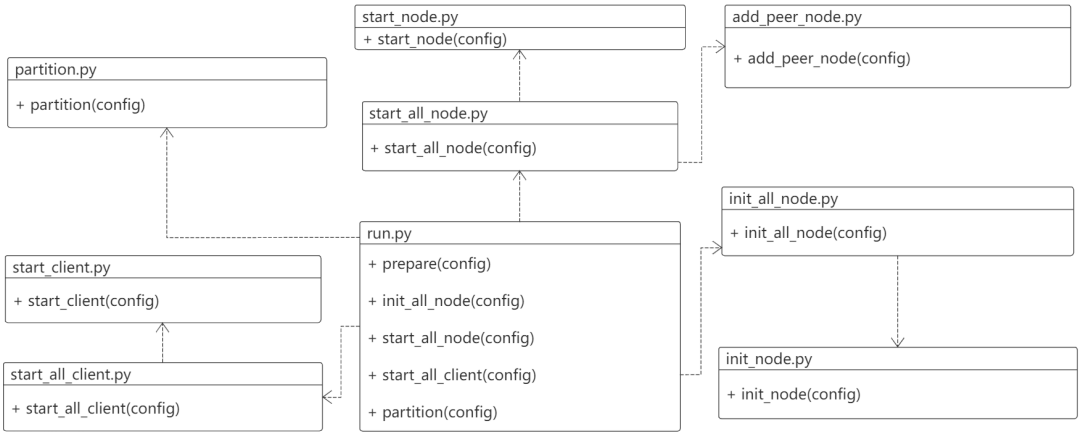

图 4.5: 测试链搭建模块类图

的数据存储目录以及初始化交易账户。由于不同的节点部署在不同的机器上，在执行私有链的初始化脚本时，需要远程免密登录到该机器。start_all_node 函数主要调用 start_all_node.py 组件实现私有链的启动。该组件中包括远程登录到相关机器执行私有链启动脚本 start_node.py，以及执行添加对等节点的脚本 add_peer_node.py。不同的私有链添加对等节点的方式会有差异，有些可能会在私有链的初始化阶段就决定那几个节点会在同一个网络中。start_all_client 函数和 start_all_node 类似，不同在于它会免密登录到客户端节点启动测试客户端，这个客户端节点的配置是由用户提供。partition 函数主要用于故意制造故障，从而测试私有链网络在故障前后的变化情况，更好看出私有链网络的容错性。

如图 4.6为测试链搭建模块的顺序图。当用户异步发送测试消息时，系统 run 组件开始执行测试任务。系统会调用 prepare 组件获取并解析用户输入的配置信息，再根据用户输入的配置信息生成相对应的测试环境变量，主要包括私有链节点运行时的数据存储目录和日志目录以及客户端节点运行时的日志目录。接下来，系统会开始初始化私有链节点，根据用户提供的私有链部署节点的 IP 地址，免密登录后执行节点初始化脚本，该脚本会根据预先设定好的创世纪块的配置生成创世纪块并在该私有链中创建新的账户。私有链初始化完成后，系统开始启动私有链，通过远程访问执行私有链启动脚本。私有链节点启动后会生成唯一的 enode 节点信息，该信息会被收集并持久化到所有节点共享的网络文件系统中，然后调用 add_peer_node 函数开始互相添加对方的节点信息，形成私有链网络。在整个私有链网络形成并进入到运行稳定阶段后，主进程通过调用 start_all_client 组件远程登录到客户端节点执行客户端启动脚本，该脚本会根据

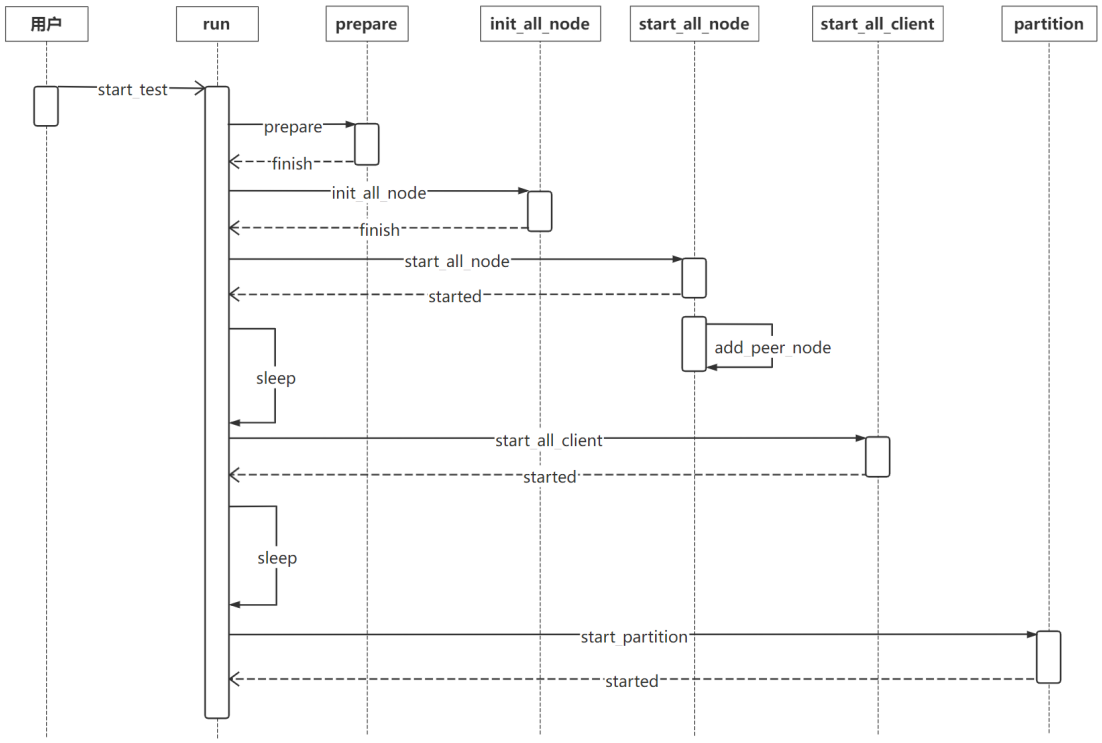

图 4.6: 测试链搭建模块顺序图

用户提供的智能合约类型、请求率、并发线程数以及测试持续时间等配置信息启动测试驱动程序。当测试驱动程序都启动后，主进程会根据用户输入的测试持续时间配置信息暂时挂起等待一段时间。在正常的测试过程结束后，系统会调用 partition 组件开始人为在测试过程过程中制造故障，进行故障测试阶段。

4.2.2 测试链搭建模块实现

如图 4.7所示的代码为测试链搭建模块 prepare 组件的实现代码。第 2-6 行获取并解析用户输入的配置信息，其中包括智能合约类型、每个线程的请求速率、并发线程数、测试持续时间等等。第 7-11 行表示系统解析用户配置的私有链节点和客户端节点 IP 地址，将字符串分割为数组，并保存到 clients 和 hosts 文件中。第 13 行开始，系统会根据不同的智能合约的类型将准备好的环境变量复制到 env.sh 文件中。环境变量主要包括自动化脚本执行的主目录、私有链节点运行时的数据存储目录和日志存储目录以及客户端节点运行时的日志存储目录。

如图 4.8所示的代码为测试链搭建模块 run 组件的实现代码。run 组件为 shell 脚本文件，参数依次为私有链节点个数、并发现线程数、客户端节点个数、每个线程的请求率以及测试持续时间。首先，该脚本会执行私有链节点初始化程序，

```
def prepare():
    benchmark = sys.argv[1]
    requestRate = int(sys.argv[2])
    threads = int(sys.argv[3])
    duration = int(sys.argv[4])
    ...
    cf = open("./clients", "w")
    cf.write(cstr.join(clients))
    ...
    hf = open("./hosts", "w")
    hf.write(hstr.join(hosts))
    ...
    if benchmark == 'ycsb':
        cp_cmd = 'cp ./env_ycsb.sh ./env.sh'
    elif benchmark == 'smallbank':
        cp_cmd = 'cp ./env_smallbank.sh ./env.sh'
    else:
        cp_cmd = 'cp ./env_helloworld.sh ./env.sh'
    os.system(cp_cmd)
```

图 4.7: 测试链搭建模块 prepare 组件的实现代码

根据预先编写好的私有链配置创建创世纪块并在链上创建新的账户。接下来，系统会开始启动所有的私有链节点，根据所有启动节点的 enode 信息并互相添加形成私有链网络。第 6-7 行定义了一个 M 变量，表示私有链网络需要预热的时间，这个时间会受到各个私有链节点的性能的影响，而且节点数量也是影响预热时间的关键因素，于是变量 M 是和节点数量存在正相关关系，所以 M 变量的确定需要进行一定数量的实验，否则，很有可能会导致私有链网络还没有进入到稳定状态就已经开始进入性能测试阶段，在很大程度上会影响自动化性能测试结果的准确性。在私有链网络预热之后，便开始启动私有链测试客户端节点，这个过程中会启动私有链测试驱动程序并部署智能合约到链上。期间，会根据用户提供的配置信息产生一定数量的线程，并对私有链施加工作负载，获取返回结果记录在客户端节点日志中。这个性能测试过程会持续用户输入的配置中的测试持续时间的一半，也就是测试时间的前半段进行正常的性能测试，后半段会进行发生节点故障之后的性能测试。在性能测试结束后，会启动 stop_all_node 组件，用于将所有节点上私有链运行进程和性能测试进程关闭，避免资源的浪费。

```
#!/bin/bash
# 参数: [私有链节点个数 并发线程数 客户端节点个数 每秒每个线程的请求率 测试持续时间]
python ./init_all_node.py $1
python ./start_all_node.py $1

let M=120+40*$1
sleep $M

python ./start_all_client.py $3 $1 $2 $4 $5 &
BACK=$!
let normal_per_test_time=$5/2
sleep $normal_per_test_time

python ./partition.py $1
wait $BACK

python ./stop_all_node.py $1
```

图 4.8: 测试链搭建模块 run 组件的实现代码

4.3 测试驱动模块

4.3.1 测试驱动模块设计

测试驱动模块为系统的核心模块，负责系统测试任务的执行。在测试链搭建模块执行完成后，便开始执行测试驱动模块。该模块主要负责按照用户设定的智能合约类型、线程数量、请求率、测试持续时间等对私有链进行智能合约的部署和性能测试，并收集测试过程中的状态数据和请求结果数据写入文件。由于在这个测试过程中会涉及到很多耗时较长的异步操作，如测试链的启动、驱动程序的启动、制造故障脚本的执行等，因此选择支持异步操作同时执行速度较快的 C++ 语言作为测试驱动模块的实现。其中利用了较为简单的异步接口 std::async，通过这个接口可以简单的创建线程并通过 std::future 中获取结果。每一种私有链都提供了成熟的 JSON API 访问或者是操作私有链上的数据，于是系统中引入 RestClient 组件构造 HTTP 请求与私有链进行交互。当客户端构造一个请求后，系统会将私有链网络返回的交易哈希值加入到等待队列中，队列中每一个元素包含 key 和 value，key 为当前交易的哈希值，value 为当前的时间。针对一个私

有链网络，该等待队列是唯一的。然后系统会间隔一定的时间根据区块号向私有链网络发送请求获取链上该区块中打包的所有的交易数据。每一个交易数据都会对应一个哈希值，这个哈希值和开始放进去的值一致，唯一识别一次交易。所有被打包的交易都可以认为是执行成功的交易，根据等待队列可以统计该间隔时间内私有链网络的吞吐量以及交易延迟，最后将成功的交易从等待队列中移除。

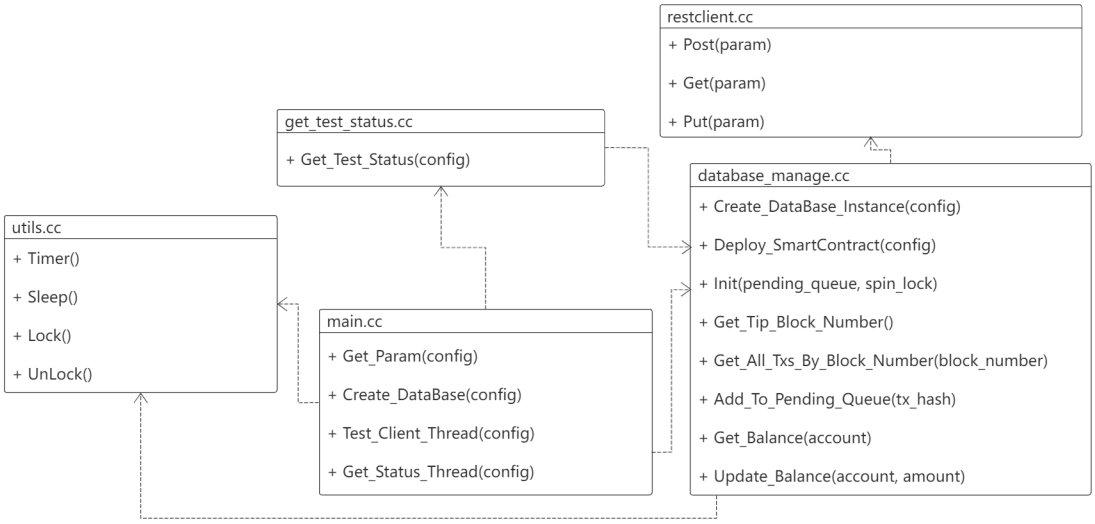

图 4.9: 测试驱动模块类图

如图 4.9所示为测试驱动模块的类图。测试驱动模块的入口为 main.cc 组件，其中包括获取用户性能参数的函数 Get_Param、创建私有链节点实例并部署智能合约的函数 Create_DataBase、启动客户端测试线程对私有链节点施加负载的函数 Test_Client_Thread 以及在测试过程中获取测试状态并统计测试结果的函数 Get_Status_Thread。其中最核心的函数为 Test_Client_Thread，它会随机产生一种交易类型并随机产生该交易类型所需的参数，调用 database_manage.cc 组件中的发送请求的方法来构造请求并发送。当请求发送后，私有链网络会返回此次交易的哈希值，database_manage.cc 组件会调用 Add_To_Pending_Queue 函数将此次交易的哈希值和交易发生的开始时间保存到等待队列 pending_queue 中。由于存在多个线程同时操作该队列，赋值的前需要先调用 utils.cc 模块获取到自旋锁，赋值完成后还需要将自旋锁释放给其他线程。database_manage.cc 组件封装了私有链实例的创建、智能合约的部署以及访问操作私有链的函数，底层会依赖 restclient.cc 组件中的 Post 等函数与私有链交互。这里将私有链网络看做是数据库，对私有链网络的请求可以理解问对数据库的请求，本质上

没有太大的区别。get_test_status.cc 组件主要用于对测试过程中等待对列 pending_queue 的管理，以及测试结果的记录。该组件会通过 database_manage.cc 组件中的 Get_Tip_Block_Number 获取到当前的私有链网络中已经确认的区块号最大的是多少，从而确认当前系统中维护的区块号是否被确认，如果当前区块号的区块在链上已经被确认了，那么可以认为该区块中的交易是成功的。于是该区块会调用 Get_All_Txs_By_Block_Number 函数将当前区块中的成功的交易全部取回到本地，并记录在这段时间内私有链网络的吞吐量和交易延迟时间，同时将成功的交易根据它的哈希值从等待队列 pending_queue 中全部删除。

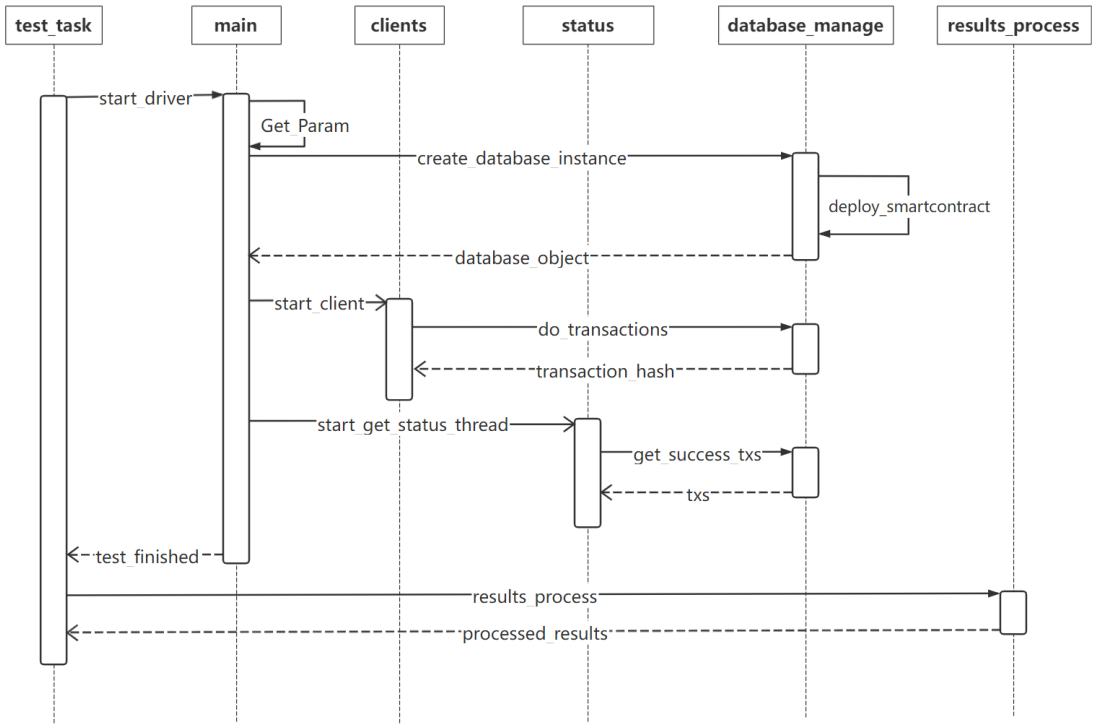

图 4.10: 测试驱动模块顺序图

如图 4.10所示为测试驱动模块的顺序图。测试任务开始会执行 main 组件中的函数。首先，系统会获取并解析传入测试驱动模块的参数，如测试链的类型、智能合约的类型、并发测试的线程数、每个线程的请求率以及测试持续的时间等。接下来，main 组件调用执行 database_manage 组件中的 create_database_instance 方法获取 database_object，该对象可以理解为私有链网络对象，封装了私有链网络相关属性和访问或者操作私有链节点的方法。其中，会执行智能合约部署函数，对象会记录并保存当前的智能合约部署的地址，以供后面操作函数使用。当智能合约部署完成后，便开始启动测试客户端程序，clients 组件会根据用户规

定的测试驱动的并发线程数量异步启动一定数量的子线程发起对私有链网络的请求。该执行过程会依赖 database_manage 组件中的 do_transactions 一系列的函数，同时每一个交易的返回值会是一个哈希值，唯一标记一次交易请求。而且，该交易会被保存到等待队列 pending_queue 中，等待队列的每一个元素由 key 和 value 构成，其中 key 为交易的哈希值，value 为交易开始的时间，这主要用于收集交易的延迟时间。当测试客户端程序启动之后，系统会异步启动获取测试状态的子线程执行测试中间状态的收集和测试中间结果的记录工作。执行的过程中会涉及到从私有链网络中获取当前已经确认的区块的所有的交易，会依赖到 database_manage 组件中的 get_success_txs 一系列的函数。获取到最大的确认区块中的所有交易数据之后，会统计私有链网络某一间隔时间中的吞吐量和所有交易的交易延迟时间，然后再将成功的交易根据它的哈希值从等待队列 pending_queue 中删除。测试驱动程序启动后会持续一段时间，这个时间由用户配置中的测试持续时间决定。最后，当测试用户规定的测试持续时间结束后，系统会杀死正在执行的驱动程序，开始调用结果处理组件 results_process 对子线程收集到的测试结果进行解析，提取并统计其中吞吐量和交易延迟相关的数据，保存到 MongoDB 数据库中。

4.3.2 测试驱动模块实现

如图 4.11所示的代码为测试驱动模块中 main 组件的实现代码。第 2 行表示系统先会读取、解析、验证性能测试所需的参数，之后会将这些参数都放在 config 配置对象中，以供后面的程序使用。第 3 行开始根据配置对象 config 使用 DataBaseFactory 工厂类创建私有链实例，该实例被看做为一个数据库，其中封装了私有链的属性和访问、操作私有链网络的函数。同时在这个过程中，用户所需执行的智能合约将部署到私有链上，私有链会返回一个调用智能合约的地址，这个地址也会保存在数据库对象中。在智能合约部署完成后，该数据库对象会初始化等待队列和自旋锁。等待队列主要用于存放未完成的交易，其中每一个元素都为 key 和 value 的键值对，key 表示交易的哈希值，value 表示该交易开始的时间。自旋锁的设计主要是为了避免在多线程操作等待队列 pending_queue 时候的线程安全问题，由于每一次对于等待队列的操作耗时极短，使用自旋锁可以有效避免操作系统做上下文切换时的性能消耗，提高整体测试的效率。当一切准备工作完成后，从第 9 行到 12 行，系统开始异步启动测试客户端线程，线程的数量由用户在基准配置中确定，每一个线程会执行对私有链网络施加负载的程序，其中的参数主要是数据库对象、执行交易的操作数量以及每秒发送请求的数量。在测试客户端线程启动后，系统紧接着会异步启动一个状态先线程，

其主要负责间隔一定的时间从私有链网络中获取私有链的状态数据，包括当前已经确认的最大的区块号以及该区块中所有被打包的交易信息，该组件所需的参数主要是数据库对象、数据库对象的名称、每次请求回去状态的间隔时间以及在测试开始时候的最大的区块号。

```
int main(const int argc, const char *argv[]) {
    Get_Param(argc, argv, config);
    DataBase *database = DataBaseFactory::Create_DataBase(config);
    ...
    database->Init(&pending_queue, &spin_lock);
    ...
    vector<future<int>> threads;
    ...
    for (int i = 0; i < threads_number; ++i) {
        threads.emplace_back(async(launch::async, Clients, database, total_operations / threads_number, txrate));
    }
    threads.emplace_back(async(launch::async, Status, database, config["dbname"], BLOCK_POLLING_INTERVAL, start_block_height));
    ...
}
```

图 4.11: 测试驱动模块中 main 组件的实现代码

如图 4.12所示的代码为测试客户端的实现代码。由于测试过程中需要模拟出交易的不确定性，系统引入了 UniformGenerator 组件制造随机参数。该组件使用的是均匀分布产生随机值，保证了所有的交易都是等概率发生的。从函数第 5 行开始，系统会根据用户指定的交易的数量随机发送交易请求，这些交易请求的执行都封装在 database 数据库对象中，其中主要包括 Amalgate：合并两个账户的余额、GetBalance：获取某一个账户的余额、UpdateBalance：更新某一个账户的余额以及 SendPayment：某一个账户给另一个账户转账等等。

如图 4.13所示为测试状态线程的实现代码。第 2 行表示状态线程首先会确定私有链区块的初始高度，这个值会被客户端维护，在不断获取私有链区块的最新高度时，会将最新高度和客户端自己维护的这个值做比对，如果该值加上确认的区块数小于最新高度，那表示当前维护的区块已经在私有链中被确认，那么其中的所有打包的交易都可以认为是执行成功的，同时客户端维护的区块高度 cur_block_height 变量会自增，来到下一个待确认的区块，如此往复不断获取

```
void Clients(DataBase* database, const int operations, const int tx_rate) {
  UniformGenerator operation_generate(1, 6);
  UniformGenerator account_generate(1, 100000);
  ...
  for (int i = 0; i < operations; ++i) {
    auto operation = operation_generate.Next();
    switch (operation) {
      case 1:
        database->Amalgate(account_generate.Next(), account_generate.Next());
        break;
      case 2:
        database->GetBalance(account_generate.Next());
        break;
      case 3:
        database->UpdateBalance(account_generate.Next(), 0);
        break;
      case 4:
        database->UpdateSaving(account_generate.Next(), 0);
        break;
      case 5:
        database->SendPayment(account_generate.Next(), account_generate.Next(), 0);
        break;
      case 6:
        database->WriteCheck(account_generate.Next(), 0);
        break;
    }
    ...
  }
}
```

图 4.12: 测试客户端的实现代码

私有链中区块的状态，也不断统计整个私有链网络的吞吐量和交易延迟。如图中第 4 行所示，confirm_duration 变量表示私有链需要等待确认的区块数，也就是在区块生成后，需要等待一定数量的区块也被确认，那么才可以认为该区块上的交易执行成功，主要避免私有链分叉所带来的数据丢失。通畅 Ethereum 需要确认的区块数为 6，其他的私有链需要确认的区块数为 1。第 8 行到第 24 行，客户端将不断请求私有链获取最新打包的区块信息，并和本地维护的区块高度做比对，如果本地区块已经在链上被确认，那么将请求私有链返回该区块中所有的交易信息。从第 12 行到 24 行，便是对私有链中最新被确认的交易信息进行统计，每一个交易都会包含一个哈希值，这个值会在客户端发送交易时暂存到请求队列中，于是通过哈希值便可知道交易的开始时间，从而统计出交易延

迟时间，保存到 latency 变量中，同样，所有成功确认的交易的数量可以认为是私有链在间隔时间内的吞吐量，该值会统计到 tx_count 变量中。当所需的数据统计完成后，还需要将已经成功的交易从等待队列 pending_queue 中删除。最后，调用 do_record 函数将这一轮的数据记录下来，以供测试结束后做结果分析。

```
void Status(DataBase* database, string dbname, double interval, int start_block_height){
    int cur_block_height = start_block_height;
    ...
    int confirm_duration = 1;
    ...
    while(true){
        ...
        int current_tip_block_height = database->get_tip_block_number();
        if (current_tip_block_height == -1)
            utils::sleep(interval);
        while (cur_block_height + confirm_duration <= current_tip_block_height) {
            vector<string> txs = database->get_all_txs_by_block_number(cur_block_height);
            cur_block_height++;
            long block_time = utils::time_now();
            spinlock.lock();
            for (string tx : txs){
                string s = tx.substr(1, tx.length() - 2);
                if (pending_queue.find(s) != pending_queue.end()){
                    tx_count++;
                    latency += (block_time - pending_queue[s]);
                    pending_queue.erase(s);
                }
            }
            spinlock.unlock();
        }
        ...
        do_record(tx_countv, latency, pending_queue.size());
        tx_count = 0;
        latency = 0;
        ...
    }
}
```

图 4.13: 测试状态线程的实现代码

4.4 测试结果管理模块

4.4.1 测试结果管理模块设计

测试结果管理模块主要负责处理性能测试任务过程中收集到的结果数据，并对测试结果进行图表展示。在所有的性能测试驱动程序都停止之后，系统会通知结果处理函数来执行预先写好的结果处理脚本程序。该脚本程序会针对测试过程中产生的日志文件进行吞吐量和交易延迟等数据的提取并汇总，产生测试结果原始数据，该数据是与执行时间相关的向量。该结果数据会根据相应的配置数据一起持久化到 MongoDB 数据库中，以便后序进行统计量的分析。测试结果管理模块会使用到 Numpy 组件对结果向量进行处理，使用 Scipy 组件对数据进行降噪和平滑处理。

如图 4.14所示为测试结果管理模块的类图。其中，view.py 组件中的 update_status 函数会在性能测试执行完成之后执行，这里会用到 Python 语言中的多线程通信机制。该函数会启动预先编写好的测试日志解析脚本，将系统所需要的吞吐量信息和交易延迟信息提取出来并持久化到数据库中。数据的操作复用 dbUtils.py 组件。detailed_result 函数是展示测试结果的详细信息，其中包含节点故障测试之前测试阶段的详细信息和节点故障之后测试阶段的详细信息，数据会以表格和折线图的方式呈现。而 PerformanceComparison 和 Scalability 函数主要用于实验结果的统计分析。其中 PerformanceComparison 为资源消耗与性能对比实验。Scalability 为伸缩性对比实验。results_process.py 组件主要用于计算实验结果的相关统计量，其中用到了 Numpy、Scipy 组件对测试结果向量进行处理。该组件可以计算整个测试过程中、节点故障之前以及节点故障之后的私有链网络的性能指标平均值、中位数等。

如图 4.15所示为所有的测试任务执行完成之后，处理和展示测试结果流程的代码设计顺序图。当测试任务执行完成之后，执行测试任务的线程会通知处于等待状态中的执行结果处理任务的线程，于是结果处理线程开始执行，读取并解析测试过程中收集到的日志，将吞吐量和交易延迟等数据解析、提取出来，并通过 dbUtils.py 组件将处理好的原始的测试结果数据和当前测试的配置信息一起放在一个对象中保存到 MongoDB 数据库。当用户想要获取详细的结果数据的时候，系统会根据测试任务的唯一 ID 从数据库中获取到对应的结果数据，并进行统计量的计算。处理完成的数据会交给 Django 的模板层进行渲染解析，最终以图表的形式呈现出来。

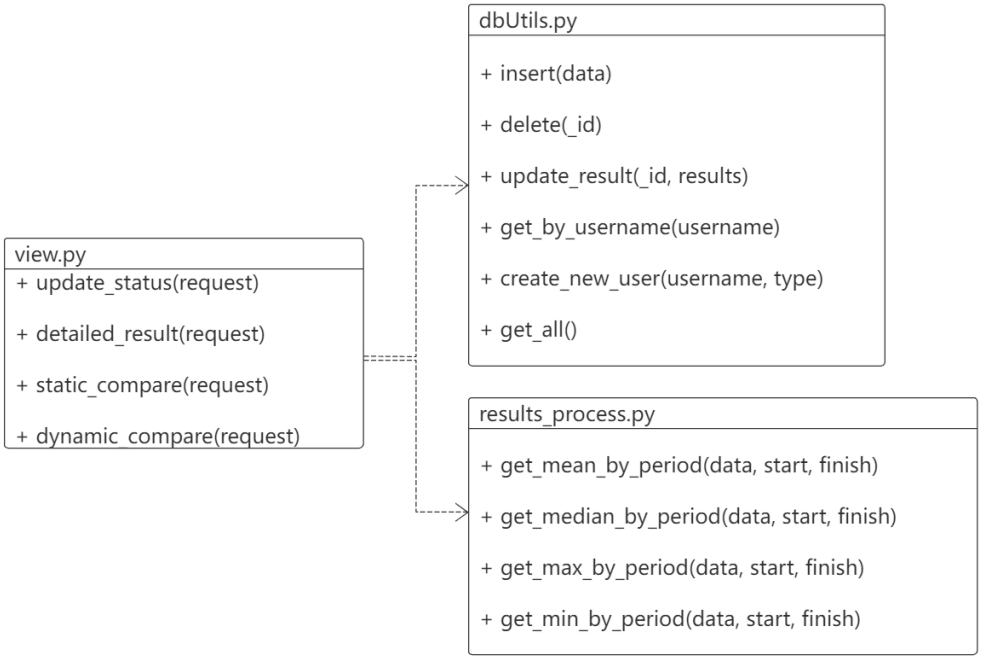

图 4.14: 测试结果管理模块类图

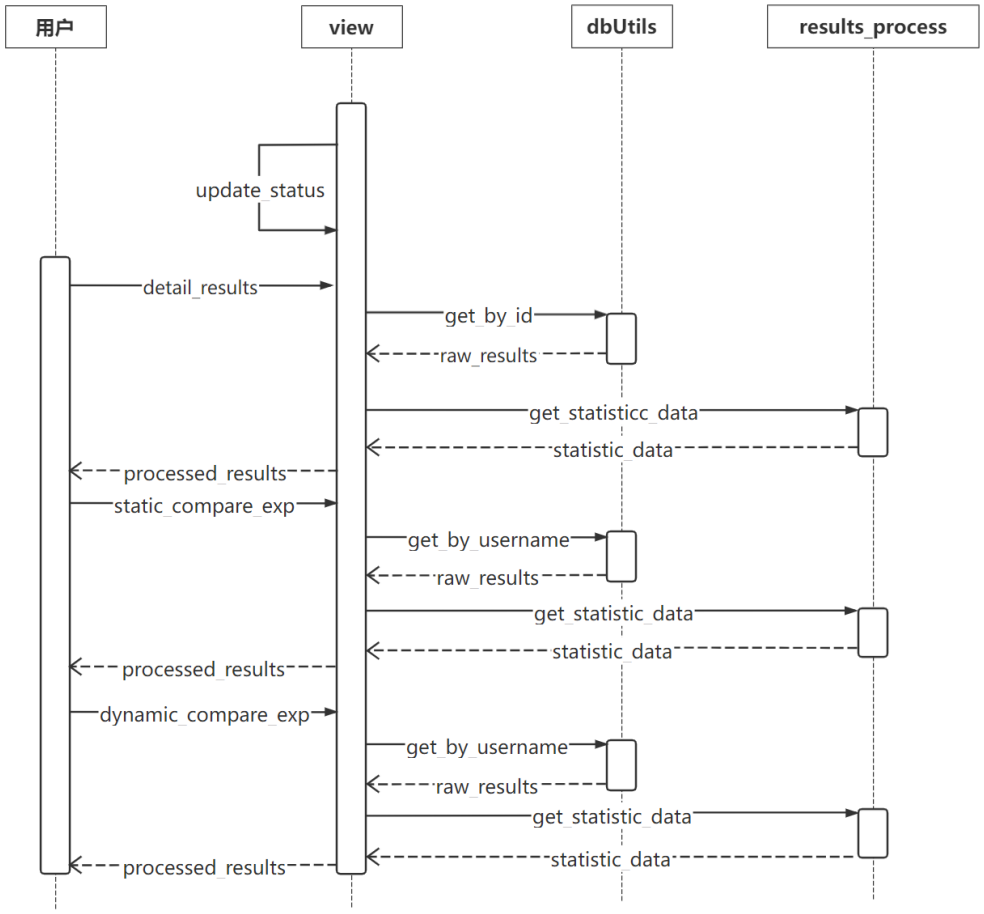

图 4.15: 测试结果管理模块顺序图

4.4.2 测试结果管理模块实现

如图 4.16所示代码为 results_process.py 组件中的数据处理函数 process_data 和统计量计算函数 get_mean_by_period 的代码实现。process_data 函数中主要实现的是对原始结果数据的处理功能。传入方法的参数 data 为待处理数据，它是一个二维向量，第一列为时间，表示测试开始后区块打包的时间点。第二列为与时间相对应的吞吐量或者是交易延迟性能指标。如果在测试过程中某一个打包时间点的被打包的交易请求数量为 0，那么此时吞吐量和交易延迟数据会出现空值。为了不影响整体的数值计算，第 5-8 行代码会先对原始数据进行数据处理，将空值以前一个邻近值代替。get_mean_by_period 函数主要实现性能指标的均值统计量的计算，传入的参数 data 为待处理的数据，和上面 process_data 函数传入的参数一样。参数 start 和 finish 表示时间的开始和结束，这里主要用于统计在测试的不同的周期下性能指标的均值，如在整个测试周期、节点故障之前的周期内以及节点故障之后的周期内等等。

```
def process_data(data):
    res = []
    tmp = np.array(data)
    x, y = tmp[:, 0], tmp[:, 1]
    for i in range(len(y)):
        if y[i] is None:
            y[i] = 0 if i == 0 else y[i - 1]
        res.append(int(y[i]))
    d = np.dstack((x, y))
    return d[0].tolist()

def get_mean_by_period(data, start, finish):
    tmp = np.array(data)
    x = tmp[:, 0]
    y = tmp[:, 1]
    print(start, finish)
    start = np.where(x >= start)[0][0]
    if finish >= len(data):
        finish = -1
    if finish != -1:
        finish = np.where(x >= finish)[0][0]
    limit = y[start:finish]
    print(limit)
    mean = np.mean(limit)
    return np.around(mean, decimals=2)
```

图 4.16: 数据处理函数与统计量计算函数代码

4.5 系统示例展示

4.5.1 系统界面截图

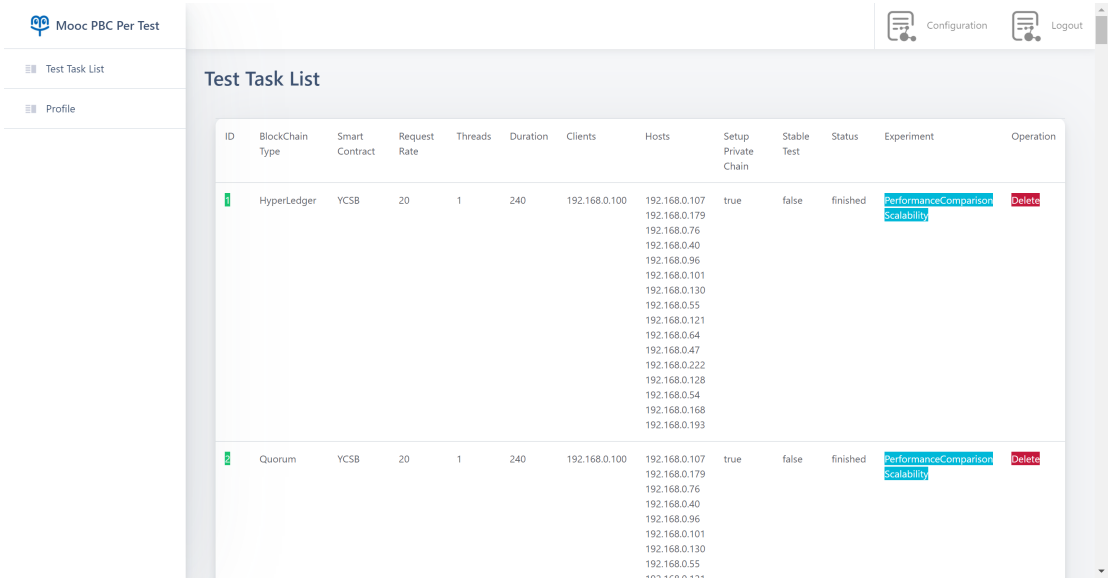

图 4.17: 测试任务界面

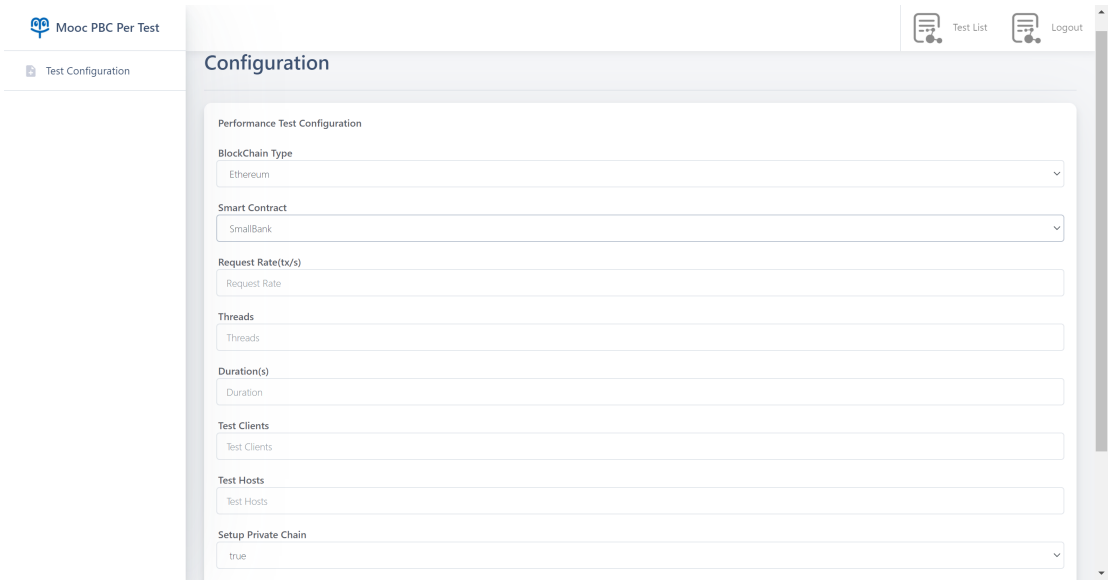

图 4.18: 测试配置界面

如图 4.17和 4.18所示为测试任务列表界面与测试配置界面。从布局看，页面分为三个部分，分别是左侧导航栏，右上方工具栏以及左下方的测试任务列表栏。在测试任务列表栏的左边，用户可以查看任务列表信息与个人信息，工具栏的按钮包括跳转测试配置界面按钮和登出系统按钮。在用户登录到系统后，主页面将会展示用户已经创建和执行的所有性能测试任务。任务列表中每一行数据对应一项测试任务，每一项都会详细列出一些基本的测试配置信息，如私有链的类型、智能合约的类型、请求速率、测试线程数、性能测试持续时间、私有链网络部署节点 IP 地址列表以及客户端节点的 IP 地址列表，还有测试的当前状态，如测试进行中状态为 pending，已完成状态为 finished。点击每个任务最前面的序号链接可以跳转到详细测试结果信息界面。在每一项的测试任务中还包含了 Experiment 实验按钮，包括 PerformanceComparison 实验和 Scalability。PerformanceComparison 为资源消耗与性能对比实验。Scalability 为伸缩性对比实验。点击右上角的配置按钮跳转至测试配置界面。配置界面包括了基准配置和节点信息配置，填写配置的操作需要用户登录后才可进行，用户正确填写完所有的测试配置信息后就可以执行测试并查看测试结果。

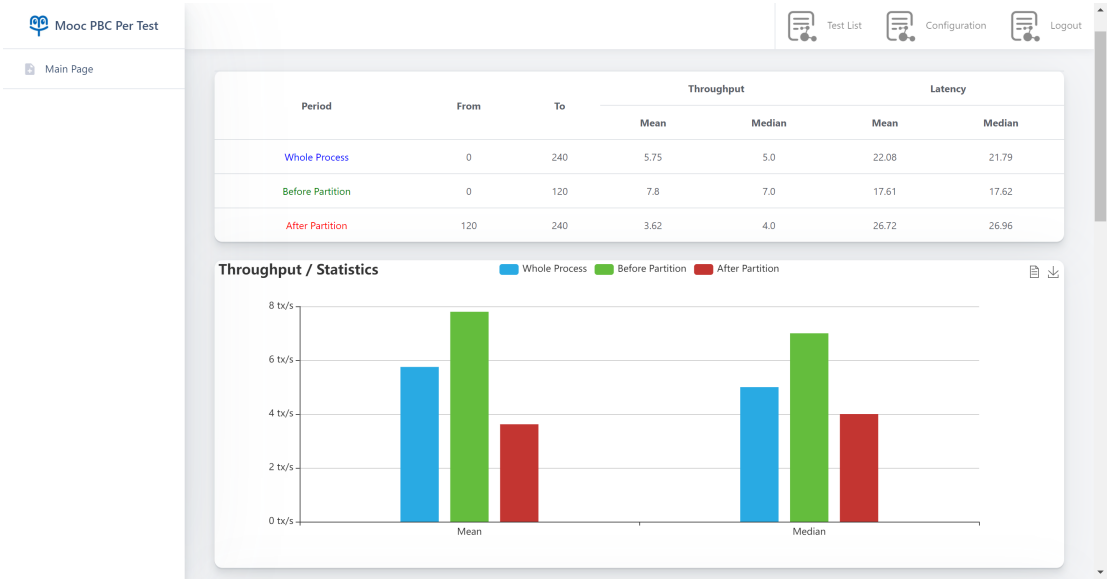

图 4.19: 测试结果详细信息界面

如图 4.19和 4.20所示为稳定性测试结果的详细信息页面。用户可以在测试配置信息中指定是否要进行私有链网络的稳定性测试。系统会以表格形式展示测试过程中吞吐量这交易延迟性能指标的平均值和中位数，如果是稳定性测试的话，一定会包括产生故障前时间段的数据和产生故障后的时间段的数据。测

图 4.20: 测试结果详细信息界面

试过程中的性能指标的动态变化数据将以折线图形式展示。图中横坐标表示时间，单位为秒，纵坐标表示相应的性能指标。图中会以红色阴影和相应的说明文字覆盖和标注产生故障后的测试时间区间，让用户直观看出节点故障对私有链网络性能指标所带来的影响。如图 4.20所示，节点故障产生后，私有链网络的吞吐量有明显的下降趋势，而交易延迟指标呈现明显的上升趋势，说明节点故障对与该私有链网络的影响较大。

4.6 系统测试

由于私有链网络的自动化性能测试流程较为复杂，而且耗时较长，对资源的消耗较大，如果进行压力测试，那么很大可能性会导致测试失败，所以本章提到的系统测试主要是对本文开发的性能测试系统进行功能测试。第五章将会设计多组实验，通过实验可以对本文实现的自动化性能测试系统进行有效的功能测试。

4.7 本章小结

本章主要介绍了系统的详细设计过程、代码的具体实现，对前面介绍的四个模块的每个模块进行详细设计和代码实现。针对每个模块，详细介绍了设计类图与设计顺序图来。然后进行详细设计，介绍关键部分的代码实现。最后展示了本系统前端的界面以及使用示例。

第五章　实验评估与分析

本文开发实现了面向私有链的自动化性能测试系统。本章将使用该系统对多种私有链进行性能对比测试、容错性测试以及伸缩性测试，并对测试结果以及测试过程中的资源消耗情况进行对比分析。

5.1　研究问题

通过对各种私有链网络的运行原理的分析，本文提出了随着私有链类型的不同，整个网络的性能、容错性以及伸缩性也会有区别。于是为了验证上述的猜想，本章提出了四个研究问题：（1）不同类型的私有链网络在资源的消耗以及消耗资源的种类是否存在显著差异；（2）不同种私有链网络是否在吞吐量和交易延迟性能指标上存在显著差异；（3）不同种类的私有链网络是否在容错性方面存在差异；（4）不同种类的私有链网络是否在伸缩性方面存在差异。同时，本章选取了四种使用较为广泛的私有链平台，它们分别是 Ethereum、Parity、Quorum 以及 Hyperledger。

针对上述的研究问题，本章提出四个研究假设：（1）不同的私有链类型对各个私有链网络中的节点资源的消耗以及消耗资源的种类存在显著影响。（2）不同种类的私有链网络在吞吐量和交易延迟性能指标上存在显著的差异。（3）不同种类的私有链网路在容错性方面存在差异。（4）不同种类的私有链类型在伸缩性方面存在差异。

5.2　实验环境

本文实验环境如表 5.1所示。由于选取的私有链类型中存在 Ethereum，它对所部署的服务器节点中的 CPU 性能有很高的的要求，同时也存在 HyperLedger 这种对所部署的节点的带宽有较高的要求，所以选择了整体性能较好的服务器。本章的实验选择了 4 核 CPU、8GB 内存、300 Mbit/s 带宽的华为云主机作为私有链节点和性能测试节点部署的服务器。其中，Ethereum 私有链网络的搭建使用了 1.10.16-stable 版本的 Geth 客户端；Parity 私有链网络的搭建使用了 2.5.13-stable 版本的 Parity 客户端；Quorum 私有链网络的搭建使用了 22.1.0 版本的 Quorum 客户端；HyperLedger 私有链网络的搭建使用了 2.4.0 版本的 HyperLedger Fabric 客户端以及 20.10.12 版本的 Docker [1]。私有链网络中的节点都启动在一个 subnet

[1]https://github.com/jenkinsci/docker

中，而且节点之间可以互相进行 ssh 免密登录执行预先编写好的自动化脚本。私有链网络中所有对等节点之间的互相发现通过静态配置实现，正常测试过程中不会有额外节点被网络中的节点发现并添加为对等节点。测试驱动程序通过 make：v4.1 调用 gcc：v7.5.0 编译为可执行的文件并执行，智能合约的编写采用 go：1.17.8 版本和 Solidity：0.4.0 版本。

表 5.1: 实验环境

参数	参数详情
云服务器	华为云服务器
操作系统	Ubuntu 18.04 64 位
CPU	4 核
CPU MHz	2200
内存	8G
带宽	300 Mbit/s
Ethereum 客户端	Geth：v1.10.16-stable
Parity 客户端	Parity：v2.5.13-stable
Quorum 客户端	Quorum：v22.1.0
HyperLedger 客户端	HyperLedger Fabric：v2.4.0
容器	Docker：v20.10.12
其他软件	gcc：v7.5.0、make：v4.1、go：v1.17.8、Solidity：v0.4.0

5.3 评价指标

本系统采用吞吐量和交易延迟指标对私有链网络的事务处理能力进行评估。吞吐量主要用来评估被测试的网络的事务处理效率情况，可以理解为某段时间内，系统能处理的事务数量，在这段时间内，待测网络处理的事务量越大，表示待测网络性能越好。交易延迟用于评估待测网络的时间效率，可以用交易请求提交到交易请求返回结果之间所间隔的时间来描述，即该时间间隔越小，交易请求正确返回得越快，表示待测网络的时间效率越高，性能越好。

本系统采用 CPU 消耗百分比、内存占用百分比以及带宽占用百分比指标来评估私有链网络的资源利用率。CPU 消耗百分比指标主要用来评估待测网络对 CPU 资源的依赖程度，如果测试过程中 CPU 使用百分比较低，则表明待测网络对于 CPU 资源的依赖程度较低，否则较高；内存占用百分比指标主要用来评估待测网络对内存资源的依赖程度，如果测试过程中内存使用百分比高居不下，那么可以认为待测网络对内存资源的依赖层度较高，否则较低。带宽占用百分比指标主要用来评估待测网络对带宽资源的依赖程度，如果测试过程中带宽使用百分比较高，或者占满整个带宽，那么可以认为待测网络对带宽资源的依赖层

度较高，否则较低。

5.4 实验设计

本章设计了三组实验：（1）不同种私有链网络在无节点故障情况下的资源消耗与性能对比实验；（2）不同种私有链网络受故障节点对性能的影响程度的对比实验；（3）不同种私有链网络在 1 个节点，2 个节点、4 个节点、8 个节点、12 个节点、16 个节点下的伸缩性对比实验。实验中客户端部署的的智能合约统一为 YCSB，测试客户端的线程数为 1，测试持续时间都为 240 秒。通过上述的实验，可以验证不同种类的私有链网络在事务处理性能、资源消耗、容错性以及伸缩性等方面的差异。

5.4.1 性能与资源消耗实验设计

本文提出了不同类型的私有链网络会在资源消耗和事务处理性能上有着显著的区别。在实验中，对私有链本身参数不做过多的调节，大部分参数都使用默认值。本系统在进行对私有链施加负载之前，都会空出一段时间，这段时间的长短会随着私有链网络类型的不同存在区别。在这段时间内，私有链网络可以进行预热，从而能更加准确地测出私有链网络在稳定状态下的事务处理性能。实验通过调节不同测试配置对不同的私有链网络进行性能和资源消耗的测试。对于每一种私有链网络，每个测试驱动线程的请求速率从 4tx/s 到 64tx/s 每隔 4tx/s 进行一次测试，共计 16 组实验。针对每一个私有链网络节点，客户端节点都会启动一个测试驱动进程对其施加负载。通过这些实验可以分析每一种私有链网络从低请求速率到高请求速率下的事务处理性能和资源消耗情况。

本节的实验设计如表 5.2所示。实验中选择的私有链的类型分别是 Ethereum、Parity、Quorum 和 HyperLedger，对于每一种私有链网络，都会配置不同的驱动线程请求速率，从 4tx/s 开始，每隔 4tx/s 开始一组实验。本性能测试系统会对实验的事务处理性能测试结果以及资源消耗测试结果进行收集并做图表显示，从而较为直观地对比不同的私有链网络的事务处理效率和资源利用率。

性能与资源消耗实验的实验结果将会通过平均值统计量和秩和检验验证方法进行评估。平均值表示的是私有链网络在整个测试阶段的平均事务处理效率以及资源利用率，可以明显看出不同种私有链网络在该方面的差异。每一种私有链网络在不同的请求速率下的平均值可以用 m_{eth}、m_{par}、m_{quo}、m_{hyp} 来表示。每一种私有链网络在不同的请求速率下所有节点的资源消耗均值同样也可以用 m_{res_eth}、m_{res_par}、m_{res_quo}、m_{res_hyp} 来表示。秩和检验用于对比两组结果数据之间存在的整体性差异性是否显著，通过比较 P 值是否小于 0.05 来判断两组数据

之间是否存在显著的整体性差异。这种检验方法不会受到数据分布方式的限制，因此适用于分布不规律的性能指标数据。使用秩和检验可以计算不同私有链网络在整个测试时间段的事务处理效率之间的数据差异以及不同种类资源的利用率的数据差异，分别用 P_{EP} 表示 Ethereum 与 Parity 之间的数据差异，用 P_{EQ} 表示 Ethereum 与 Quorum 之间的数据差异，用 P_{EH} 表示 Ethereum 与 HyperLedger 之间的数据差异，用 P_{PQ} 表示 Parity 与 Quorum 之间的数据差异，用 P_{PH} 表示 Parity 与 HyperLedger 之间的数据差异，用 P_{QH} 表示 Quorum 与 HyperLedger 之间的数据差异。通过上述的一系列 P 值可以得出这四种私有链网络的性能指标和资源消耗的指标是否存在显著差异，进而分析出不同种类的私有链网络是否会在资源消耗和事务处理性能上有着显著的区别。

表 5.2: 性能与资源消耗实验设计

实验名称	私有链类型	参数名称	参数设置			
性能与资源消耗实验	Ethereum	请求速率	4tx/s	8tx/s	12tx/s	16tx/s
	Parity		20tx/s	24tx/s	28tx/s	32tx/s
	Quorum		36tx/s	40tx/s	44tx/s	48tx/s
	HyperLedger		52tx/s	56tx/s	60tx/s	64tx/s

5.4.2 容错性实验设计

本文提出了不同类型的私有链网络会在容错性上有着显著的区别。实验中在测试持续时间过去一半时，通过远程登录私有链节点杀死相应进程的方式制造节点宕机的故障，统计发生故障后私有链网络对于事务请求的处理效率，通过故障前后网络处理效率的比较，得出不同程度的故障对于这个私有链网络的影响程度，从而判断出私有链网络的容错性。性能测试过程中的请求速率会设置为 20，因为根据第一个实验的结果，在请求速率为 20tx/s 时，Ethereum 私有链网络吞吐量较高且稳定，当增加请求速率时，会导致私有链网络压力过高而吞吐量下降。本系统在进行对私有链施加负载之前，都会空出一段时间，这段时间的长短会随着私有链网络类型的不同存在区别。在这段时间内，私有链网络可以进行预热，从而能更加准确地测出私有链网络在稳定状态下的容错性。实验中私有链网络中初始状态有 4 个节点，据统计，当 Quorum 和 HyperLedger 网络中的故障节点数量大于总节点数量的 1/3 时，整个网络会停止处理客户端发动的请求。所以对于每一种私有链网络，从这三种状态开始测试，当测试进行到一半时，通过远程登录到私有链网络，杀死其中 1/3 数量的节点，同时统计网络故障后的性能指标数据，从而进行分析故障前后的性能指标数据，得到不同私有

链网络在容错性方面是否存在显著性差异。

本节实验设计如表 5.3所示。实验中选择的私有链的类型分别是 Ethereum、Parity、Quorum 和 HyperLedger，对于每一种私有链网络，首先要在系统中的配置中选择进行故障测试。负载测试持续时间统一为 240 秒，在故障测试过程中会有两个时间段，一个时间段为故障之前，是指在私有链网络进入稳定状态后持续进行 120 秒的测试，另一种时间段为故障之后，是指在私有链网络发生节点故障后持续进行 120 秒的测试。

容错性实验的实验结果通过平均值统计量和秩和检验验证方法进行准确评估。平均值表示的是私有链网络在不同时间段的平均事务处理效率，可以明显看出数据之间的差异。故障之前和故障之后的性能平均值分别用 m_{pre_fail}，m_{after_fail} 表示。使用秩和检验可以计算故障测试实验中的故障之前时间段和故障之后时间段之间的性能指标数据差异，用 P 表示。通过上述的 P 值可以比较这两个时段内的性能指标是否存在显著差异，进而分析出不同种类的私有链网络受故障节点的影响程度。

表 5.3: 容错性实验设计

实验名称	私有链类型	故障测试	持续时间	时间段	
容错性实验	Ethereum Parity Quorum HyperLedger	true	240s	故障之前 0-120s	故障之后 120-240s

5.4.3 伸缩性实验设计

本文提出了不同类型的私有链网络会在伸缩性上存在差异。实验通过设置私有链网络中的节点的个数来观察在节点数不断增加的情况下，不同种类的私有链网络是否表现出事务处理性能的下降。本系统在进行对私有链施加负载之前，都会空出一段时间，这段时间的长短会随着私有链网络类型的不同存在区别。在这段时间内，私有链网络可以进行预热，从而能更加准确地测出私有链网络在稳定状态下的事务处理性能。本节总共分为 6 次实验，每次将私有链网络中的节点数量设置为 1、2、4、8、12、16, 每次实验进行 100 次，实验结果取平均值，从而得出不同种类的私有链网络在伸缩性方面的表现。

本节实验设计如表 5.4所示。实验中选择的私有链的类型分别是 Ethereum、Parity、Quorum 和 HyperLedger，对于每一种私有链网络，需要在系统的配置页

面配置不同个数的服务器 IP 地址，这些服务器上部署的私有链节点通过内网的连接共同组成一个私有链网络。

伸缩性实验的实验结果通过平均值和秩和检验进行评估。平均值表示的是私有链网络在不同节点数量时的平均事务处理效率，可以直观看出数据的差异。节点数量不同的私有链网络的性能平均值分别用 m_i 表示，其中 i 表示私有链网络中节点的个数。使用秩和检验可以计算不同节点数量的私有链网络之间的性能指标数据差异，用 P_{j_k} 表示，其中 j 和 k 分别表示节点数量。通过上述的一系列 P 值可以得出同一种私有链网络在节点数量不同时的性能指标是否存在显著差异，进而分析出不同种类的私有链网络的伸缩性。

表 5.4: 伸缩性实验设计

实验名称	私有链类型	参数名称	参数设置			
伸缩性实验	Ethereum Parity Quorum HyperLedger	IP 地址数组	[IP]	[IP, IP]	[IP, IP, IP, IP]	[IP, IP, ..., IP]

5.5 实验结果与分析

5.5.1 性能与资源消耗实验结果与分析

性能与资源消耗实验的实验结果会以折线图进行展示。每个图中，横坐标为客户端的请求速率，从 4tx/s 到 64tx/s，两两之间相隔 4tx/s。纵坐标为所有私有链网络在某一种请求速率下的性能指标平均值。性能与资源消耗实验的实验分析数据会以表格的方式展示，其中包括每一种私有链网络在某一种请求速率下的性能或者是资源消耗指标的平均值以及私有链网络两两之间的秩和检验值，用于验证它们两两在整个测试过程中的指标数据差异是否显著。

如图 5.1所示为不同种私有链网络在不同请求速率下吞吐量性能指标的结果。图中可以看出，当请求速率不断增大且小于 20tx/s 时，各种私有链网络的吞吐量性能指标在不断增大；当请求速率大于 20tx/s 并不断增大时，Ethereum 私有链网络随着请求速率的增大呈现下降趋势，而其他三种私有链网络依然呈现持续上升趋势。整体来看，私有链网络的吞吐量的大小关系从大到小依次是 HyperLedger、Quorum、Parity、Ethereum。

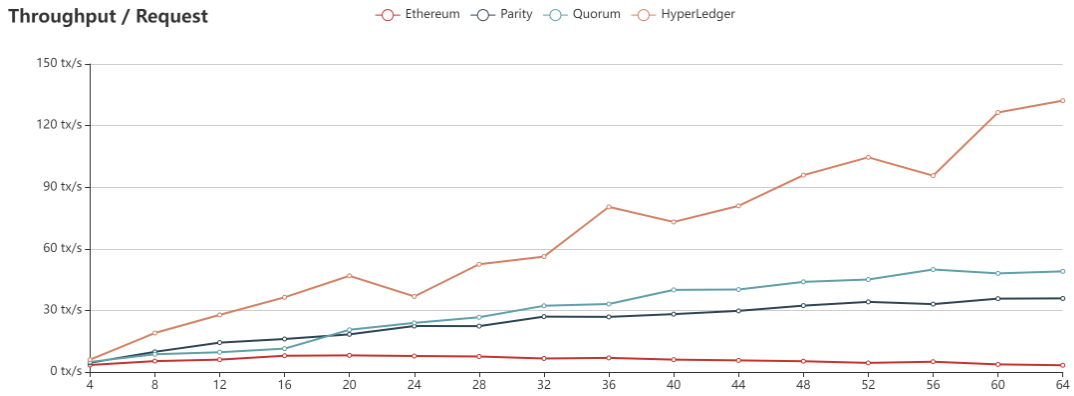

图 5.1: 吞吐量-请求速率

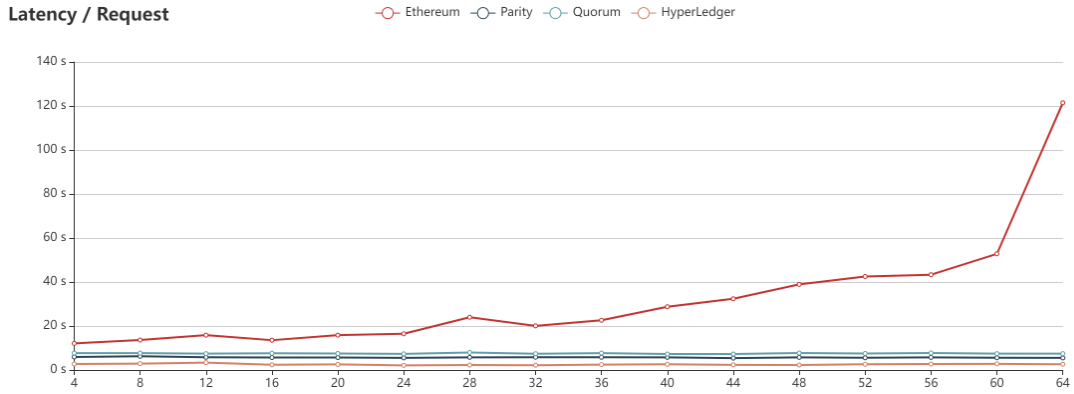

图 5.2: 交易延迟-请求速率

如图 5.2所示为不同种私有链网络在不同请求速率下交易延迟性能指标的结果。图中可以看出，当请求速率不断增大时，Parity、Quorum 和 HyperLedger 这三种私有链网络的交易延迟并没有明显的变化趋势，保持在一定范围内上下浮动；而 Ethereum 私有链网络的交易延迟呈现明显的上升趋势，而且随着请求速率的增大，上升的趋势越明显。整体来看，私有链网络的交易延迟的大小关系从大到小依次是 Ethereum、Quorum、Parity、HyperLedger。

如图 5.3所示为不同种私有链网络在不同请求速率下所有节点 CPU 资源消耗的结果。图中显示了每一次实验中所有节点 CPU 消耗百分比的平均值。从图中可以看出 Ethereum 私有链网络在整个实验的过程中每个节点的 CPU 消耗百分比持续在 95% 附近浮动，可见 Ethereum 网络比较强依赖于 CPU 资源，而其他三种私有链网络对于 CPU 资源的消耗较 Ethereum 网络而言处于一个比较低

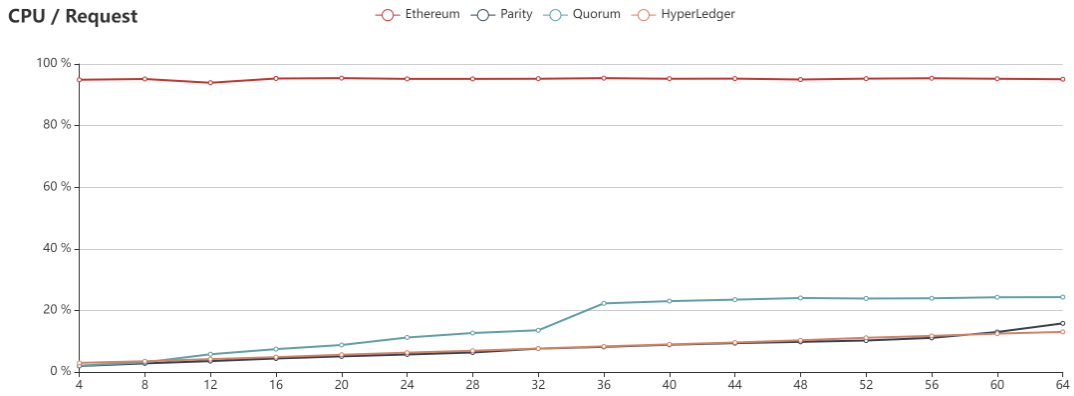

图 5.3: CPU-请求速率

的水平，尤其是 Parity 和 HyperLedger 私有链网络，虽然随着请求率的升高 CPU 资源的消耗也在增多，但是增长及其缓慢，可见实验过程中 CPU 资源对于网络的运行不存在约束作用。

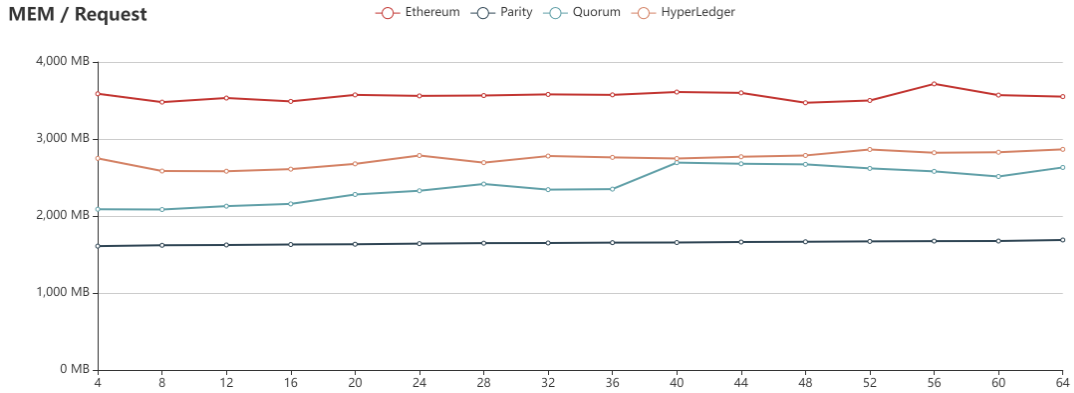

图 5.4: MEM-请求速率

如图 5.4所示为不同种私有链网络在不同请求速率下所有节点内存资源消耗的结果。图中显示了每一次实验中所有节点内存消耗的平均值。从图中可以看出，随着请求速率的不断提升，所有私有链网络在实验中的内存资源的消耗并存在明显的增长或者下降的趋势，可见网络在运行过程中对于内存资源的消耗都比较稳定。相对而言，Ethereum 对于内存资源的消耗比其他三种网络整体偏大，而 Parity 私有链网络在实验过程中对于内存资源的消耗维持在一个较低的水平，且随着请求速率的增大变化幅度不明显。

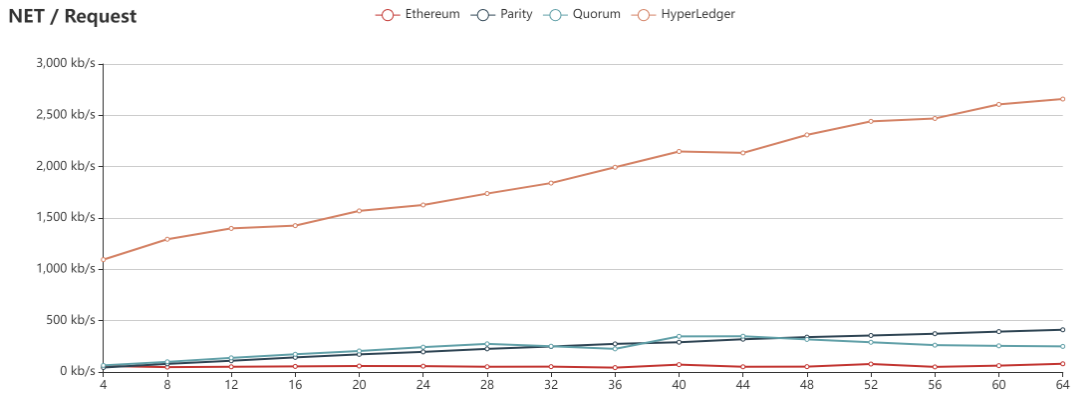

图 5.5: NET-请求速率

如图 5.5所示为不同种私有链网络在不同请求速率下所有节点带宽资源消耗的结果。图中显示了每一次实验中所有节点带宽消耗的平均值。从图中可以看出，随着请求速率的不断提升，HyperLedger 和 Parity 呈现明显的上升趋势，其中 HyperLedger 的上升趋势尤为明显，而其他两种私有链网络不存在明显的趋势，维持在某一个值附近。相对来说，Ethereum 网络对于带宽资源的消耗最低，实验过程中几乎不存在带宽的消耗。而 HyperLedger 网络在运行过程中表现出对带宽资源较强的依赖，且随着请求速率的提升带宽消耗不断增高。

表 5.5: 性能指标数据分析表

指标类型	请求速率	m_{eth}	m_{par}	m_{quo}	m_{hyp}	P_{EP}	P_{EQ}	P_{EH}	P_{PQ}	P_{PH}	P_{QH}
吞吐量 (tx/s)	4tx/s	3.38	4.47	4.9	5.98	5.7e-05	3.2e-12	3.2e-19	**0.32**	2.3e-09	2.7e-15
	8tx/s	5.31	9.83	8.64	18.99	1.6e-11	8.2e-20	2.6e-31	**0.061**	4.4e-16	1.1e-30
	16tx/s	7.93	16.09	11.36	36.36	3.3e-14	3.5e-09	4.1e-32	**0.056**	5.0e-20	1.5e-49
	32tx/s	6.59	26.98	32.24	56.2	1.4e-23	5.6e-27	1.2e-34	0.00067	1.5e-19	1.9e-15
	64tx/s	3.29	35.83	52.55	132.15	1.7e-10	1.5e-08	1.7e-11	0.00034	6.1e-27	8.6e-12
交易延迟 (s)	4tx/s	12.11	5.93	7.68	2.75	5.2e-24	2.3e-34	1.2e-41	7.1e-24	5.5e-30	1.6e-48
	8tx/s	13.62	6.25	7.69	2.88	1.3e-20	6.9e-28	2.0e-33	7.2e-16	3.5e-24	1.6e-39
	16tx/s	13.55	5.67	7.62	2.37	8.3e-25	1.0e-40	9.4e-32	4.0e-25	9.1e-26	2.1e-51
	32tx/s	20.08	5.8	7.41	2.17	7.1e-24	2.4e-28	8.2e-34	3.9e-14	2.1e-29	4.5e-37
	64tx/s	121.46	5.49	7.25	2.64	1.8e-10	1.7e-08	1.6e-11	2.5e-15	9.6e-27	6.7e-15

表 5.5为不同请求速率下私有链网络的性能指标数据分析结果。根据各项性能指标的平均值可以得出，整体来说，吞吐量会随着请求速率的增大而不断增大，但是对于 Ethereum 网络而言，吞吐量会随着请求速率的提升表现出先上升到最高点后不断下降的趋势。其中，HyperLedger 私有链网络无论在吞吐量还是交易延迟方面，表现都比其他三种网络好，平均吞吐量较高，且平均交易延迟较低。而且，Ethereum 私有链网络变现最差，整体呈现平均吞吐量较低，而交易延迟较高。

根据秩和检验的结果，在吞吐量指标上，当请求速率较小时，$P_{PQ} > 0.05$，可见此时的 Parity 和 Quorum 网络的表现没有显著差异。而在交易延迟方面 Parity 和 Quorum 网络存在显著性差异。如表所示，所有 P_{EP}、P_{EQ}、P_{EH} 的值都小于 0.05，可见 Ethereum 与其他三种私有链网络无论在吞吐量指标还是在交易延迟指标上都存在显著性差异。所有 P_{EH}、P_{PH}、P_{QH} 的值都小于 0.05，可见 HyperLedger 私有链网络与其他三种网络在吞吐量和交易延迟指标上存在显著性差异。

表 5.6: 资源消耗指标数据分析表

指标类型	请求速率	m_{res_eth}	m_{res_par}	m_{res_quo}	m_{res_hyp}	P_{EP}	P_{EQ}	P_{EH}	P_{PQ}	P_{PH}	P_{QH}
CPU (%)	4tx/s	94.87	1.94	2.07	2.96	3.9e-29	6.4e-40	6.5e-28	**0.60**	**0.51**	**0.61**
	8tx/s	95.15	2.79	3.14	3.48	1.4e-27	1.2e-38	1.4e-26	**0.52**	**0.45**	**0.60**
	16tx/s	95.31	4.39	7.44	4.85	1.7e-28	2.1e-38	1.3e-26	**0.76**	**0.72**	**0.66**
	32tx/s	95.23	7.6	13.54	7.62	5.3e-27	6.0e-38	1.5e-25	**0.66**	**0.55**	**0.68**
	64tx/s	95.06	15.8	24.33	13.04	1.7e-26	3.6e-28	1.1e-24	**0.43**	**0.38**	**0.34**
MEM (MB)	4tx/s	3587	1608	2087	2748	2.3e-26	4.5e-32	1.1e-11	3.7e-26	3.7e-20	2.5e-19
	8tx/s	3479	1619	2084	2584	1.5e-25	3.2e-30	2.0e-14	4.7e-27	2.4e-21	2.9e-21
	16tx/s	3488	1629	2157	2608	2.3e-26	3.2e-31	3.2e-16	4.7e-27	2.4e-21	1.6e-14
	32tx/s	3578	1649	2342	2778	1.5e-25	6.2e-23	5.7e-12	4.7e-27	2.4e-21	2.8e-05
	64tx/s	3550	1688	2630	2865	6.4e-26	3.3e-13	4.2e-08	5.1e-22	2.4e-21	0.00018
NET (kb/s)	4tx/s	58	43	64	1094	**0.82**	**0.72**	4.9e-07	**0.58**	4.6e-20	3.9e-10
	8tx/s	47	79	99	1293	**0.64**	**0.60**	5.0e-25	**0.79**	1.8e-18	5.2e-20
	16tx/s	54	142	172	1425	**0.68**	**0.56**	3.7e-25	**0.47**	1.7e-18	1.3e-21
	32tx/s	52	247	250	1840	**0.63**	**0.59**	6.7e-26	**0.84**	1.4e-18	8.2e-24
	64tx/s	79	411	249	2658	**0.58**	**0.43**	1.9e-25	**0.47**	1.7e-14	2.1e-19

表 5.6为不同种私有链网络在不同请求速率下资源消耗指标的数据分析结果。根据不同请求速率下各种私有链网络的资源消耗指标的平均值可以看出，CPU 和内存资源的消耗指标不会随着请求速率的增大呈现明显的趋势，整体平均值数据会持续在某一个值上下浮动。而带宽资源的消耗指标会随着请求速率的增大呈现不断增大的趋势，HyperLedger 网络在这一方面表现的尤为明显，增长速率也很大。就 CPU 资源的消耗而言，Ethereum 较其他三种网络偏大，且其对 CPU 资源具有强依赖。Parity、Quorum 以及 HyperLedger 网络对于 CPU 资源的消耗较为类似，而且根据秩和检验的结果，在 CPU 资源消耗方面，P_{PQ}、P_{PH}、P_{QH} 的值都大于 0.05，所以这三种网络在 CPU 指标上不存在显著性差异。从内存消耗指标来看，私有链网络互不相同，Ethereum 网络的内存消耗最大，Parity 网络的内存消耗最小，根据秩和检验的结果，在内存消耗方面，所有的 P 值都小于 0.05，所以私有链网络在内存消耗指标上存在显著性差异。从带宽消耗指标来看，各种私有链网络的带宽消耗平均值会随着请求速率的增大呈现不断增大的趋势，而且 HyperLedger 网络在这方面表现明显，且该网络带宽资源的消耗较大，根据秩和检验的结果，在带宽资源消耗方面，P_{EP}、P_{EQ}、P_{PQ} 的值大于 0.05，所有 Ethereum、Parity 以及 Quorum 三种私有链网络之间的带宽资源的消耗不存

在显著性差异。而 P_{EH}、P_{QH}、P_{PH} 的值都远小于 0.05，可见 HyperLedger 与其他三种私有链网络在带宽资源的消耗上存在显著性差异。

5.5.2 容错性实验结果与分析

容错性实验的实验结果会以折线图进行展示。每个图中，横坐标为时间，每一个节点表示私有链网络产生区块并打包交易的时间点，单位为秒。纵坐标为所有私有链网络在两两间隔的时间节点内的吞吐量的值，单位为 tx/s，或者是交易延迟的数值，单位为 s。容错性实验的实验分析数据会以表格的方式展示，其中包括每一种私有链网络在故障之前以及故障之后的吞吐量和交易延迟性能指标的平均值，分别用 m_{pre_fail}，m_{after_fail} 表示。同时分析计算出每一种私有链网络在这两个运行阶段下性能指标的秩和检验值，用于验证每一种网络在出现故障之前和出现故障之后的性能指标的数据差异是否显著。

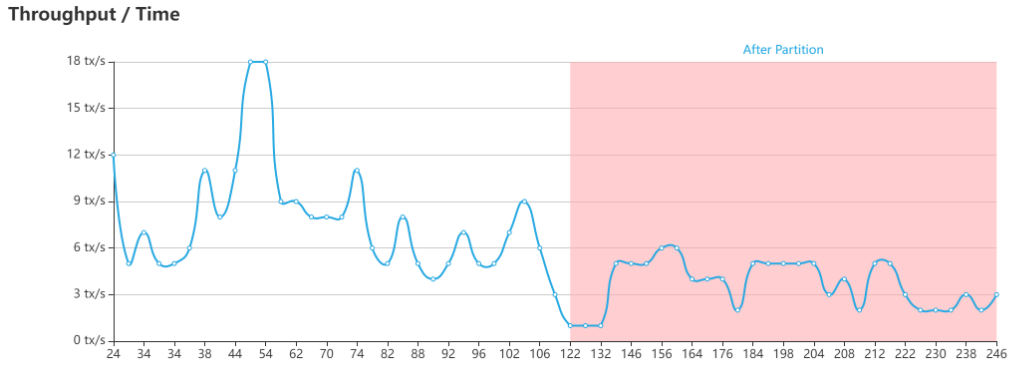

图 5.6: Ethereum 吞吐量随时间变化图

如图 5.6和图 5.7所示为 Ethereum 私有链网络在容错性实验中的性能测试结果。图中的红色背景部分为私有链网络在发生故障后的性能指标数据。图 5.6表示 Ethereum 网络随着时间的推移，吞吐量指标在不断变化。从该图中可以看出，在发生节点故障之前，Ethereum 网络的吞吐量整体较高，且经过计算，这段时间内的吞吐量均值为 7.8tx/s；当发生节点故障之后，Ethereum 网络的吞吐量整体偏低，且经过计算，发生故障之后的这段时间内吞吐量均值为 3.62tx/s，明显低于发生节点故障之前的均值。而且从图 5.7中可以看出，在发生节点故障之前的这段时间内，Ethereum 网络的交易延迟指标数据均值为 17.61s，较发生节点故障之后的交易延迟指标均值 26.72s 明显偏低。所有可以看出，Ethereum 私有链网络的性能指标数据受到故障节点的影响较大，其容错性较差。

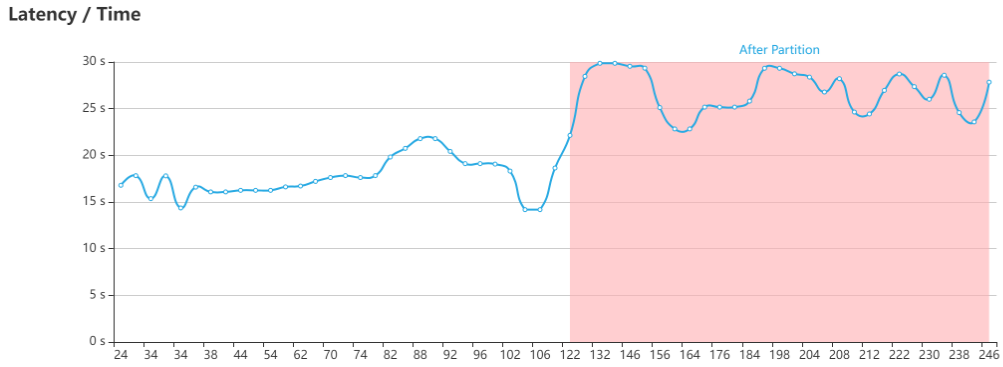

图 5.7: Ethereum 交易延迟随时间变化图

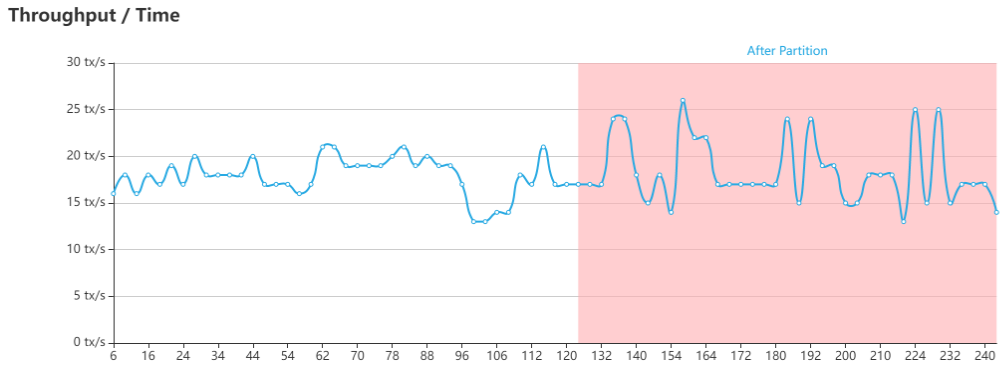

图 5.8: Parity 吞吐量随时间变化图

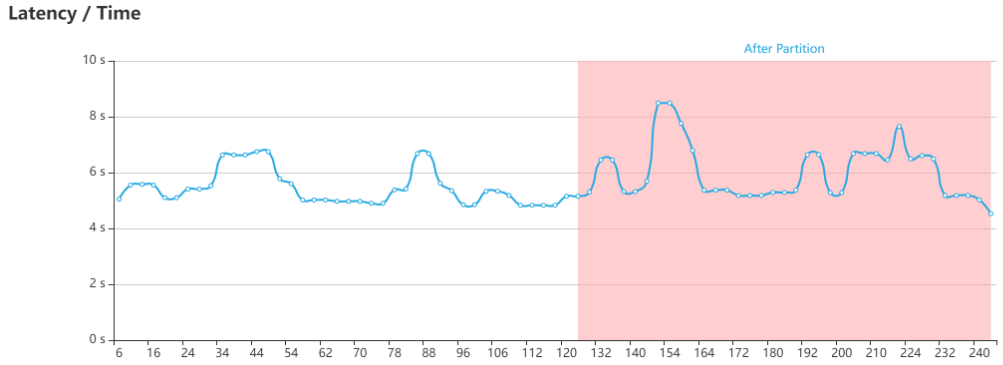

图 5.9: Parity 交易延迟随时间变化图

如图 5.8和图 5.9所示为 Parity 私有链网络在容错性实验中的性能测试结果。从该图 5.8中可以看出，在发生节点故障前后的时间段内，Parity 网络的吞吐量没有明显的变化趋势，且经过计算，发生故障前后的时间段内的吞吐量均值分别为 17.85tx/s 和 18.35tx/s，两者相差并不大。从图 5.9中可以看出，发生节点故障前后的时间段内，Parity 网络的交易延迟指标数据值同样不存在明显的变化趋势，且经过计算，这两段时间内的均值分别为 5.45s 和 5.99s，两者同样差别也不大。由此可以看出，Parity 私有链网络中的故障节点对于整个网络的性能指标数据没有明显的影响，该网络的容错性较强。

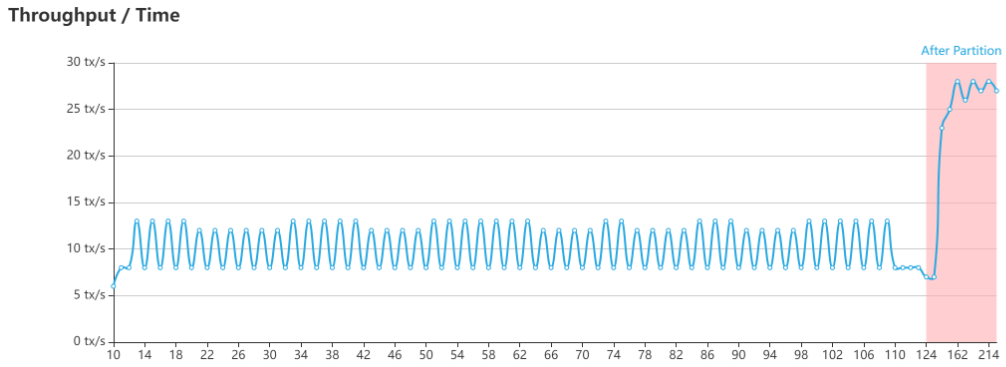

图 5.10: Quorum 吞吐量随时间变化图

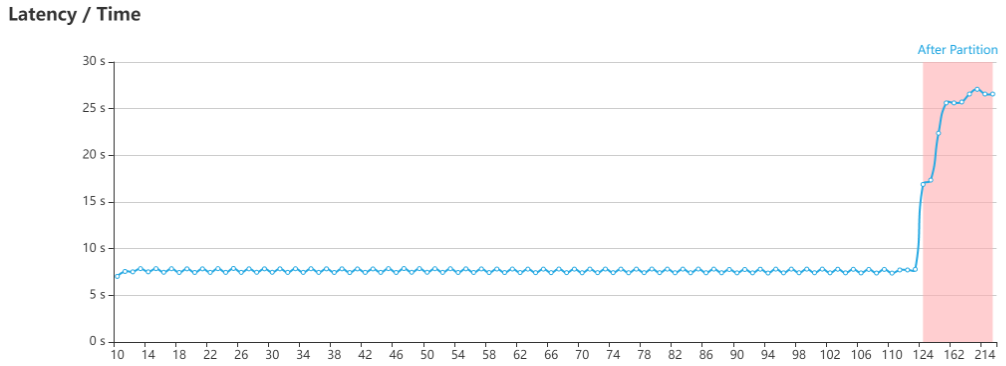

图 5.11: Quorum 交易延迟随时间变化图

如图 5.10和图 5.11所示为 Quorum 私有链网络在容错性实验中的性能测试结果。从图中可以看出，在发生节点故障之前，Quorum 网络的吞吐量和交易延

迟指标数据较为稳定，但是发生节点故障之后，该网络产生区块的速度大幅降低，这一点可以从图中红色背景的宽度可以看出。如图 5.10所示，发生节点故障之前，吞吐量指标数据整体偏低，且经过计算，其均值为 10.12tx/s；当发生节点故障后，这个网络的吞吐量急剧上升，经过计算，这段时间内的均值为 22.6tx/s，明显高于故障之前的吞吐量均值。而且从图 5.11可以看出，Quorum 网络的交易延迟指标数据在故障之前处于一个较低的值附近上下浮动，且经过计算，其均值为 7.63s，当发生节点故障后，该网络的交易延迟急剧上升，经过计算，这段时间内的均值为 24.04s，明显高于故障之前的交易延迟均值。由此可见，Quorum 私有链网络的性能指标数据受到故障节点的影响较大，其容错性较差。

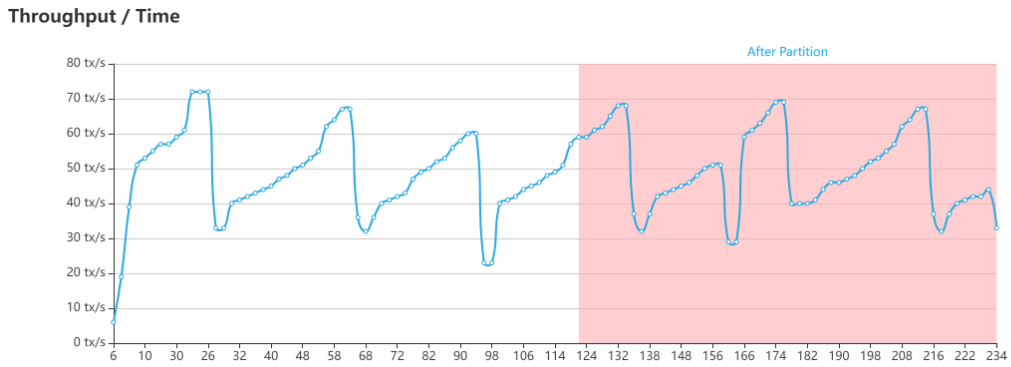

图 5.12: HyperLedger 吞吐量随时间变化图

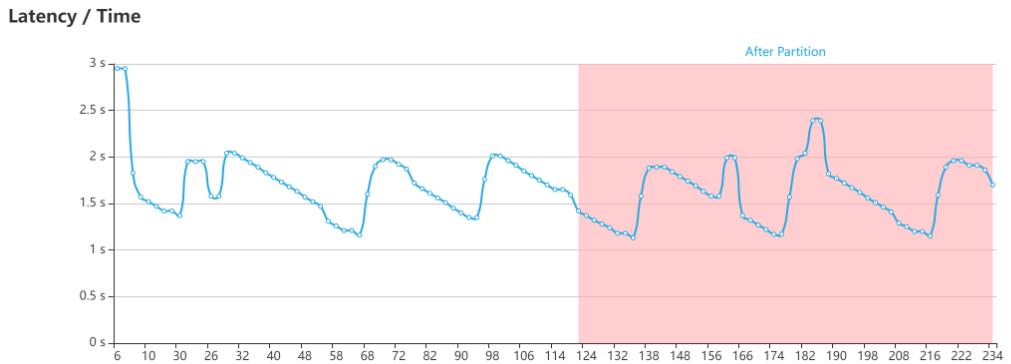

图 5.13: HyperLedger 交易延迟随时间变化图

如图 5.12和图 5.13所示为 HyperLedger 私有链网络在容错性实验中的性能

测试结果。从该图 5.12中可以看出，在发生节点故障前后的时间段内，Hyper-Ledger 网络的吞吐量没有明显的变化趋势，且经过计算，发生故障前后的时间段内的吞吐量均值分别为 47.83tx/s 和 49.63tx/s，两者相差并不大。从图 5.13中可以看出，发生节点故障前后的时间段内，HyperLedger 网络的交易延迟指标数据值同样不存在明显的变化趋势，且经过计算，这两段时间内的均值分别为 1.72s 和 1.6s，两者同样差别也不大。由此可以看出，HyperLedger 私有链网络中的故障节点对于整个网络的性能指标数据没有明显的影响，该网络的容错性较强。

表 5.7: 容错性实验性能指标数据分析表

指标类型	私有链类型	m_{pre_fail}	m_{after_fail}	P_{pre_after}
吞吐量 (tx/s)	Ethereum	7.8	3.62	1.1e-07
	Parity	17.85	18.35	**0.63**
	Quorum	10.12	22.6	0.00077
	HyperLedger	47.83	49.63	**0.69**
交易延迟 (s)	Ethereum	17.61	26.72	4.4e-11
	Parity	5.45	5.99	**0.067**
	Quorum	7.63	24.04	1.8e-07
	HyperLedger	1.72	1.6	**0.081**

表 5.7为不同种私有链网络在容错性实验中性能指标的数据分析结果。根据不同种私有链网络在故障之前和故障之后的性能表现均值，Ethereum 和 Quorum 两种网络故障前后的吞吐量以及交易延迟性能指标的均值相差较大，由此看出，故障节点对于网络的性能影响比较大。根据秩和检验的结果，在吞吐量和交易延迟这两种指标上，P_{pre_after} 值都小于 0.05，可以验证，Ethereum 和 Quorum 这两种私有链网络的性能指标受故障节点的影响较为显著，其容错性较差。与之相对的是 Parity 和 HyperLedger 这两种私有链网络，在故障前后的吞吐量以及交易延迟性能指标的均值相差不大，由此看出，故障节点对于这两种网络的性能影响较小。根据之和检验的结果，在吞吐量和交易延迟这两种指标上，P_{pre_after} 值都大于 0.05，可以验证得到，Parity 和 HyperLedger 这两种私有链网络的性能指标受到故障节点的影响不显著，其容错性较强。

5.5.3 伸缩性实验结果与分析

伸缩性实验的实验结果会以折线图进行展示。每个图中，横坐标为网络中的节点个数。纵坐标为所有私有链网络在某一节点数量下的吞吐量的值，单位为 tx/s，或者是交易延迟的时间，单位为 s。伸缩性实验的实验分析数据会以表格的方式展示，其中包括每一种私有链网络在初始节点数量为 1、2、4、8、12、

16 时的吞吐量和交易延迟性能指标的平均值，分别用 m_1、m_2、m_4、m_8、m_12、m_16 表示。同时分析计算出每一种私有链网络在这节点数量间隔情况下的性能指标的秩和检验值，用于验证每一种网络在间隔节点数量下的性能指标的数据差异是否显著，从而得出每一种网络的伸缩性结果。

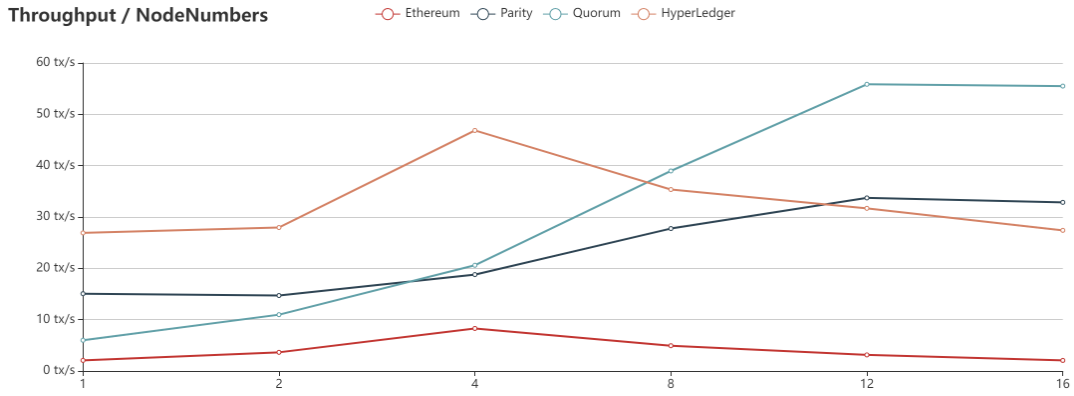

图 5.14: 吞吐量随网络中节点数量变化图

如图 5.14所示为不同种私有链网络在不同节点数量下吞吐量性能指标的测试结果。图中可以看出，当网络中的节点数量不断增大且小于 4 个时，各种私有链网络的吞吐量性能指标在不断增大；当节点个数大于 4 个并不断增大时，Ethereum 和 HyperLedger 两种私有链网络随着节点数量的增大呈现下降趋势，而其他两种私有链网络依然呈现持续上升趋势，但当网络中节点数量达到 12 个时，上升的趋势不再明显，甚至有下降的趋势。直观来看，当网络中节点数量小于 4 个时，在网络中添加节点可以一定程度增大网络的吞吐量指标；当网络中节点数量超过 4 个或者等于 4 个时，在 Ethereum 和 HyperLedger 这两种私有链网络中不断增加节点并不能起到增大吞吐量指标的效果，反而会使得这个网络的吞吐量下降；而在 Parity 和 Quorum 两种私有链网络中添加节点时，私有链网络的吞吐量性能不断提升，当达到 12 个节点时，吞吐量达到峰值，继续添加节点，这个网络的吞吐量会呈现下降的趋势。

如图 5.15所示为不同种私有链网络在不同节点数量下交易延迟时间性能指标的测试结果。图中可以看出，当网络中的节点数量不断增大且小于 4 个时，Ethereum 网络的交易延迟指标不断下降，其他三种网络的交易延迟指标趋于稳定；当节点个数大于 4 个并不断增大时，Ethereum 网络的交易延迟指标呈直线上升的趋势，可见此时节点数量的增加对于交易延迟指标的影响很大；而其他三种网络的交易延迟指标随着节点数量的增大上升缓慢，甚至趋于稳定。直观

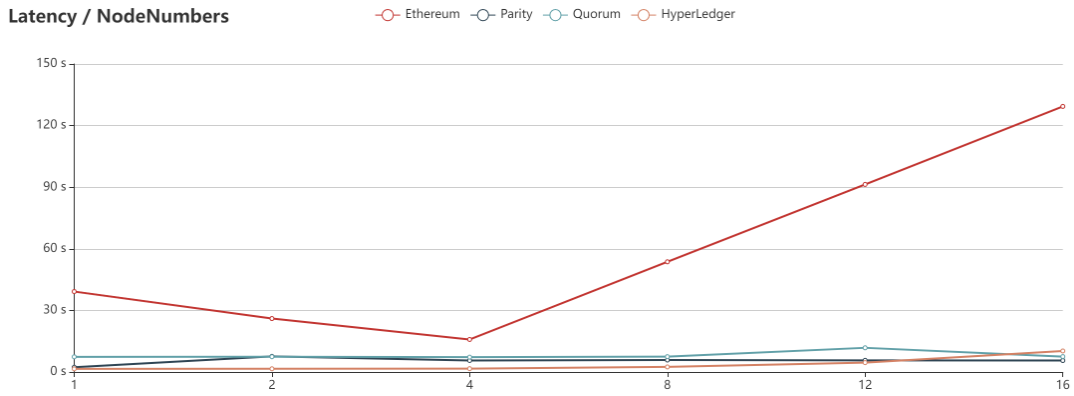

图 5.15: 交易延迟随网络中节点数量变化图

来看，对于 Ethereum 网络来说，网络中节点数量小于 4 个时，在网络中添加节点可以一定程度减小网络的交易延迟指标，从而提升性能；当网络中节点数量超过 4 个或者等于 4 个时，在 Ethereum 私有链网络中不断增加节点并不能起到减小交易延迟指标的效果，反而会使得这个网络的交易延迟时间上升，削弱整个网络的性能；而在 HyperLedger、Parity 和 Quorum 三种私有链网络中添加节点时，私有链网络的交易延迟性能增长缓慢，甚至会趋于稳定。

表 5.8: 伸缩性实验性能指标数据分析表

指标类型	私有链类型	m_1	m_2	m_4	m_8	m_{12}	m_{16}	P_{1_2}	P_{2_4}	P_{4_8}	P_{8_12}	P_{12_16}
吞吐量 (tx/s)	Ethereum	2.07	3.64	8.12	4.93	3.14	2.06	0.0063	5.5e-08	8.0e-08	3.6e-06	7.7e-08
	Parity	15.06	14.9	18.35	27.75	33.72	32.85	**0.14**	0.0029	4.3e-19	3.3e-17	**0.16**
	Quorum	5.99	9.97	20.55	38.97	55.85	54.63	6.8e-56	1.3e-49	1.2e-31	1.9e-07	0.31
	HyperLedger	26.9	27.95	46.86	36.13	31.69	27.4	**0.52**	2.6e-29	5.8e-06	0.001	**0.074**
交易延迟 (s)	Ethereum	39.17	26.03	15.84	53.67	91.36	129.35	2.9e-05	2.0e-13	9.1e-11	7.1e-09	1.1e-05
	Parity	2.32	7.62	5.67	5.82	5.66	5.58	7.1e-22	1.1e-06	**0.14**	**0.62**	**0.31**
	Quorum	7.39	7.85	7.5	7.46	9.39	7.98	**0.27**	**0.18**	**0.56**	0.0053	0.014
	HyperLedger	1.54	1.57	2.53	2.47	4.57	10.24	**0.42**	**0.061**	0.0062	**0.51**	0.023

表 5.8为不同种私有链网络在不同节点数量下性能指标的数据分析结果。根据不同节点数量下各种私有链网络的吞吐量与交易延迟性能指标的最优值直观可以看出，Ethereum 网络在节点数量为 4 个时吞吐量达到峰值，交易延时时间最短，当不断增大节点数量时，Ethereum 网络的性能指标呈现明显的下降趋势。且根据秩和检验的结果，P 值都小于 0.05，可见增大网络中节点数量网络的性能指标的变化存在显著差异。与 Ethereum 网络一样，HyperLedger 网络也在节点数量为 4 个时吞吐量到达峰值，但当节点数量不断增大时，吞吐量不会直线下滑，而是在节点数量为 12 和 16 时下降的趋势放缓，此时的吞吐量分别是 36.13tx/s

和 27.4tx/s，且根据秩和检验的结果，$P_{12\_16} > 0.05$，可见 12 个节点和 16 个节点的 HyperLedger 网络的吞吐量指标不存在显著性差异。与前两种私有链网络不同的是，Parity 和 Quorum 网络会随着节点数量的增大吞吐量指标呈现上升趋势，而交易延迟指标维持稳定，直观看来没有明显的变化趋势，且根据秩和检验的结果，两种网络的交易延迟指标的 P 值基本上都大于 0.05，所以该指标不存在显著性差异。而吞吐量指标的 P 值 P_{12_16} 之前都小于 0.05，所以 Parity 和 Quorum 网络随着节点的增大，吞吐量性能指标的提升较为显著，当节点数量达到 12 时，吞吐量性能指标达到峰值。

5.6 本章小结

本章主要介绍利用性能测试系统对多种私有链网络进行了性能测试实验，并对实验的结果进行评估与分析。针对本章开端提出的四种假设，共设计了三组实验进行验证。同时，本章介绍了实验环境、评价指标、实验的设计、实验的结果以及对实验结果数据的分析。通过数据分析可以验证，不同种私有链网络之间的性能会在一个或者几个方面存在显著的差异。

第六章　总结与展望

6.1　总结

随着区块链技术的诞生，人们认识到了其所具有的不可篡改、可溯源、智能合约以及去中心化等特性，这些特性在一般的分布式系统中是没有的。但是区块链的性能问题一直制约着它的快速发展，而私有链的出现改善了这一问题，它通过牺牲一定的去中心化的能力，保证了区块链网络的不可篡改、可溯源、智能合约以及安全可靠的特性，并应用到多个专业领域。目前来看，不管是想要开发去中性化应用的用户还是私有链的开发者、维护者，在搭建、开发或者是维护私有链网络的时候，都需要对私有链网络的性能进行准确评估。而且私有链网络是一种涉及到多个服务器节点的分布式系统，所以其容错性、伸缩性在生产环境中同样显得尤为关键。

面向私有链网络的性能测试系统主要包括用户配置信息的收集、私有链网络的搭建、测试客户端发送请求负载以及测试结果的存储与可视化。本系统会先根据用户提供的基准配置信息和节点信息配置信息做解析，如果待测链已经搭建完成，那么直接进行性能测试环节，否则系统会预先搭建配置信息中所指定的私有链网络用于性能测试，网络中节点的部署也会由用户在私有链网络节点配置信息中指定。接下来，客户端启动性能测试进程对部署、预热完成的私有链部署智能合约并发送交易请求；最后系统会解析性能测试环节中收集到的测试结果信息，并保存到 MongoDB 数据库中，以供用户查看或者分析。本系统对私有链使用者、私有链开发者、私有链维护者提供私有链网络的性能测试服务。私有链使用者可以通过本系统对比各大主流的私有链平台，选取一种适合自己去中心化应用的数据载体；私有链开发者想要优化私有链性能时通过本系统测试网络的性能是否有明显的提高；私有链维护者可以通过本系统测试私有链网络是否能够满足用户的需求。

技术层面上，系统设计结合了 Ethereum 技术、Parity 技术、Quorum 技术、HyperLedger 技术、Docker 技术、性能测试方法，将私有链网络节点和性能测试节点进行分离，实现可插拔的测试服务，在这方面，论文具有创新性。系统的开发实现采用了 Python、C++、Shell 语言相结合，不同的模块会使用适合的语言开发，论文在这方面具有一定的挑战性。其中配置信息管理模块和测试结果管理模块由 Python 语言实现，使用了业界主流的 Django 框架，测试配置信息和测试结果都保存在 MongoDB 数据库中。测试链搭建模块和测试驱动模块都使用了

Shell 和 C++ 语言开发，其中 Shell 语言主要开发相应的自动化运行脚本，测试驱动的开发还是使用了 C++ 语言。其中，在 HyperLedger 私有链网络的搭建中使用了 Docker 官方镜像搭建了 Docker 子网。

为了对比不同种私有链网络是否在吞吐量、交易延迟、资源消耗率、容错性以及伸缩性方面存在差异，本文设计了三组实验。其中第一组是对比分析了多种私有链网络在无故障运行过程中的吞吐量、交易延迟以及资源消耗率性能指标。第二组是对比分析了多种私有链网络吞吐量和交易延迟性能指标受节点故障的影响程度。第三组是对比分析了多种私有链网络的吞吐量和交易延迟性能指标在节点数量不断增大时的表现。实验结果表明，多种私有链网络在上述一个或者多个性能指标方面存在差异，由此可见，面向私有链网络的性能测试系统可以达到预期的效果。

6.2 展望

当前开发的系统存在很大可扩展的空间，以后将会从三个方面对系统进行改进：

第一，支持更多种性能测试指标。目前系统支持的性能测试指标有吞吐量、交易延迟、资源利用率、容错性以及伸缩性，这还远远不够，以后还会增加节点可靠性、基础事务执行可靠性、加密技术可靠性、智能合约可靠性等多个角度对私有链网络进行更加全面的性能测试。

第二，支持更多种私有链网络的性能测试。当前系统仅仅支持四种比较常见的私有链网络的性能测试，但是随着私有链技术的不断发展，可能会出现更多比较优秀的私有链客户端，同时也会出现更多种类的智能合约，未来系统将支持更多类型的私有链网络的测试。

第三，支持更加复杂的测试场景。本文实验中私有链网络部署的节点服务器性能较为一般，节点的个数也较少。在后续的工作中，我们将选用性能更加突出的服务器部署私有链网络，同时增加网络中节点的个数，探究私有链网络在多节点下的性能表现，从而支持更加复杂的测试场景。

参考文献

[1] 夏清, 窦文生, 郭凯文, et al. 区块链共识协议综述 [J]. 软件学报, 2021, 32(2): 277–299.

[2] CHAN K C, ZHOU X, GURURAJAN R, et al. Integration of Blockchains with Management Information Systems[J]. IEEE, 2020: 157–162.

[3] OLIVEIRA M T, CARRARA G R, FERNANDES N C, et al. Towards a Performance Evaluation of Private Blockchain Frameworks using a Realistic Workload[C] // 2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops. 2019: 180–187.

[4] TIAN Y, YUAN J, SONG H. Secure and Reliable Decentralized Truth Discovery Using Blockchain[C] // 7th IEEE Conference on Communications and Network Security, 2019, Washington, DC, USA, June 10-12, 2019. [S.l.]: IEEE, 2019: 1–8.

[5] 焦通, TONG J, 申德荣, et al. 区块链数据库: 一种可查询且防篡改的数据库 [J]. 软件学报, 2019, 30(9): 2671–2685.

[6] FAN C, GHAEMI S, KHAZAEI H, et al. Performance evaluation of blockchain systems: A systematic survey[J]. IEEE Access, 2020, 8: 126927–126950.

[7] ALKASEM A, LIU H, ZUO D. CloudPT: Performance Testing for Identifying and Detecting Bottlenecks in IaaS[M]. [S.l.]: Algorithms and Architectures for Parallel Processing, 2018: 432–452.

[8] AVRITZER A, KONDEK J, LIU D, et al. Software performance testing based on workload characterization[C] // Proceedings of the 3rd international workshop on Software and performance. 2002: 17–24.

[9] 宋巍, 张春柳, 邬斌亮. Web 系统性能测试研究与实践 [J]. 计算机应用与软件, 2015, 32(3): 4–6.

[10] HALILI E H. Apache JMeter:a practical beginner's guide to automated testing and performance measurement for your websites[M]. [S.l.]: Packt Publishing, 2008.

[11] DENARO G, POLINI A, EMMERICH W. Early performance testing of distributed software applications[J]. Acm Sigsoft Software Engineering Notes, 2004, 29(1): 94 – 103.

[12] DINH T T A, WANG J, CHEN G, et al. Blockbench: A framework for analyzing private blockchains[C] // Proceedings of the 2017 ACM international conference on management of data. 2017: 1085 – 1100.

[13] ZHENG P, ZHENG Z, LUO X, et al. A detailed and real-time performance monitoring framework for blockchain systems[C] // 2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track. 2018: 134 – 143.

[14] CHONGXUAN YUAN;JIANMING ZHU. A New Performance Testing Scheme for Blockchain System[J]. LISS2019, 2020: 757 – 773.

[15] HUNT K, ZHUANG J. Blockchain for disaster management[G] // Big Data and Blockchain for Service Operations Management. [S.l.]: Springer, 2022: 253 – 269.

[16] NAKAMOTO S. Bitcoin: A peer-to-peer electronic cash system[J]. Decentralized Business Review, 2008: 21260.

[17] CHOHAN U W. The Double Spending Problem and Cryptocurrencies[J]. SSRN Electronic Journal, 2017: 1 – 11.

[18] AMANI S, BÉGEL M, BORTIN M, et al. Towards verifying ethereum smart contract bytecode in Isabelle/HOL[C] // Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs. 2018: 66 – 77.

[19] AL MALLAH R, LÓPEZ D, FAROOQ B. Cyber-Security Risk Assessment Framework for Blockchains in Smart Mobility[J]. IEEE Open Journal of Intelligent Transportation Systems, 2021, 2: 294 – 311.

[20] KONASHEVYCH O. Constraints and benefits of the blockchain use for real estate and property rights[J]. Journal of Property Planning and Environmental Law, 2020, 12(2): 109 – 127.

[21] HAMIDA E B, BROUSMICHE K L, LEVARD H, et al. Blockchain for Enterprise: Overview, Opportunities and Challenges[J]. ICWMC, 2017: 91.

[22] ANDROULAKI E, BARGER A, BORTNIKOV V, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains[C] // Proceedings of the thirteenth EuroSys conference. 2018 : 1 – 15.

[23] PUTHAL D, MALIK N, MOHANTY S P, et al. Everything You Wanted to Know About the Blockchain: Its Promise, Components, Processes, and Problems[J]. IEEE Consumer Electronics Magazine, 2018, 7(4) : 6 – 14.

[24] CACHIN C, VUKOLIC M. Blockchain Consensus Protocols in the Wild[C] // RICHA A W. Leibniz International Proceedings in Informatics, Vol 91 : 31st International Symposium on Distributed Computing. Dagstuhl, Germany : Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017 : 1:1 – 1:16.

[25] VUKOLI M. The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication[C] // International Workshop on Open Problems in Network Security. 2016 : 112 – 125.

[26] SANKAR L S, SINDHU M, SETHUMADHAVAN M. Survey of consensus protocols on blockchain applications[C] // 2017 4th international conference on advanced computing and communication systems. 2017 : 1 – 5.

[27] BALIGA A, SOLANKI N, VEREKAR S, et al. Performance Characterization of Hyperledger Fabric[C] // 2018 Crypto Valley Conference on Blockchain Technology. 2018 : 65 – 74.

[28] KIAYIAS A, RUSSELL A, DAVID B, et al. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol[C] // Annual International Cryptology Conference. 2017 : 2 – 4.

[29] SZABO N. Formalizing and Securing Relationships on Public Networks[J]. First Monday, 1997, 2(9).

[30] WOOD G, OTHERS. Ethereum: A secure decentralised generalised transaction ledger[J]. Ethereum project yellow paper, 2014, 151(2014) : 1 – 32.

[31] 欧阳丽炜, 王帅, 袁勇, et al. 智能合约: 架构及进展 [J]. 自动化学报, 2019(3) : 13.

[32] PONGNUMKUL S, SIRIPANPORNCHANA C, THAJCHAYAPONG S. Performance analysis of private blockchain platforms in varying workloads[C] // 2017 26th International Conference on Computer Communication and Networks. 2017 : 1 – 6.

[33] MCGINN D, MCILWRAITH D, GUO Y. Towards open data blockchain analytics: a Bitcoin perspective[J]. Royal Society open science, 2018, 5(8) : 180 – 298.

[34] HOU H. The application of blockchain technology in E-government in China[C] // 2017 26th International Conference on Computer Communication and Networks. 2017 : 1 – 4.

[35] SAROJADEVI H. Performance testing: methodologies and tools[J]. Journal of Information Engineering and Applications, 2011, 1(5) : 5 – 13.

[36] WEYUKER E J, VOKOLOS F I. Experience with performance testing of software systems: issues, an approach, and case study[J]. IEEE Transactions on Software Engineering, 2000, 26(12) : 1147 – 1156.

[37] WRIGHT C P, JOUKOV N, KULKARNI D, et al. Auto-pilot: A Platform for System Software Benchmarking.[C] // USENIX Annual Technical Conference, FREENIX Track. 2005 : 175 – 188.

[38] HAN X. A Study of Performance Testing in Configurable Software Systems[J]. Journal of Software Engineering and Applications, 2021, 14(9) : 474 – 492.

[39] CHOI H, YEOM K. An approach to software architecture evaluation with the 4+1 view model of architecture[C] // Asia-pacific Software Engineering Conference. 2002 : 286 – 293.

[40] JIN-GANG Q I, TAO L I, JIN-JUN L I. Research on query pagination technology for web data based on django framework[J]. Electronic Design Engineering, 2014, 22(5) : 33 – 37.

简历与科研成果

基本情况 王佩旭，男，汉族，1994 年 4 月出生，江苏省南通市人。

教育背景

2020.9 ~ 2022.7 南京大学软件学院 硕士

2012.9 ~ 2016.7 浙江理工大学启新学院 本科

读研期间的成果（包括发表的论文及参与的专利）

致　　谢

感谢我的导师陈振宇教授为我指明了项目研究的方向，在我的系统开发和论文写作过程中给我的悉心指导，而且在项目的设计和开发过程中给了我很多督促和帮助，在论文写作方面也给出了很多建设性的意见。

感谢王兴亚博士后。王老师在区块链研究方面有着丰富的经验，在我的研究过程中，王老师悉心指导，为我的项目研究提供极大的帮助。

感谢黄勇先生。黄勇先生在软件开发方面有着丰富的实践经验，在我项目开发过程中给我提出了很多宝贵的意见。

感谢软件学院所有的老师、同学们。感谢老师们的悉心教导，让我的专业知识和技能有了质的飞跃。感谢同学们在学习过程中的陪伴。

感谢一直支持我的家人们。很快我就要步入社会了，我会尽我最大的努力报答你们养育之恩。

《学位论文出版授权书》

本人完全同意《中国优秀博硕士学位论文全文数据库出版章程》（以下简称“章程”），愿意将本人的学位论文提交“中国学术期刊（光盘版）电子杂志社”在《中国博士学位论文全文数据库》、《中国优秀硕士学位论文全文数据库》中全文发表。《中国博士学位论文全文数据库》、《中国优秀硕士学位论文全文数据库》可以以电子、网络及其他数字媒体形式公开出版，并同意编入《中国知识资源总库》，在《中国博硕士学位论文评价数据库》中使用和在互联网上传播，同意按“章程”规定享受相关权益。

作者签名：王佩旭

2022 年 5 月 26 日

论文题名	面向私有区块链的自动化性能测试系统				
研究生学号	MF20320159	所在院系	软件学院	学位年度	2022
论文级别	☐学术学位硕士 ☑专业学位硕士 ☐学术学位博士 ☐专业学位博士 (请在方框内画钩)				
作者 Email	13773607064@163.com				
导师姓名	陈振宇 教授				

论文涉密情况：

☑ 不保密

☐ 保密，保密期（_____年_____月_____日 至 _____年_____月_____日）