学校代码：10246
学　　号：17212040263

硕 士 学 位 论 文

（专业学位）

一种基于状态反馈负载均衡策略和以太坊的私募股权交易平台设计与实现

A Design and implementation of a private equity trading platform based on state feedback load balancing strategy and Ethereum

院　　系：　计算机科学技术学院
专业学位类别(领域)：　计算机技术
姓　　名：　赵晓峰
指 导 教 师：　卢暾 教授
完 成 日 期：　　年 月 日

指导小组成员名单

顾　宁　　教　授
张　亮　　教　授
卢　暾　　教　授
丁向华　　副教授

目 录

摘 要

区块链技术近年来发展迅速，相关应用如雨后春笋般蓬勃发展。区块链应用生态圈逐渐形成，绝大多数的应用通过接入区块链公共服务机构如以太坊(Ethereum)、超级账本(HyperLedger)等加入区块链网络。应用的接入依赖于单一的节点导致了整个应用的流量压力全部集中到一个 Geth 客户端上，使得基于以太坊的区块链应用的并发性能较低。同时，由于以太坊技术多应用于数字信息交易领域，对数据安全要求较高，因此单 Geth 客户端的架构在容灾和可靠性方面亦存在不足。

针对上述问题，结合区块链与相关业务场景的特点，设计和实现高性能以太坊服务系统极为必要。因此，本文结合微服务思想设计了以太坊后台服务架构，基于 Docker 构建了多 Geth 客户端的以太坊服务平台，基于 Geth 客户端在 Docker 容器中的运行状态设计了反馈负载均衡策略，并最终实现了基于状态反馈负载均衡策略和微服务的私募股权交易平台。本文主要工作如下：

1. 提出一种基于微服务思想的私募股权交易平台后台服务设计方案。根据私募股权业务中读写业务对以太坊 Geth 客户端服务能力影响的不同设计了读写分离封装子服务的应用架构，有效地提高了私募股权交易平台核心业务的并发性能。

2. 基于以太坊自身特点提出了一种状态反馈负载均衡策略，该策略考虑区块链节点运行状态建立状态空间，对状态空间中的节点进行负载分配实现系统的可靠性和高性能。并将该负载均衡策略应用于基于容器化技术思想构建了一种灵活的底层区块链服务平台。该平台将以太坊客户端运行于容器中作为单一节点，结合负载均衡策略实现可靠性和高性能。

3. 实现了面向私募股权的区块链服务系统，包括账户管理、转账管理和合约发布等业务功能，以及状态监控、网关管理等后台模块功能。结合区块链的智能合约等技术特点实现了私募股权交易自动化结算与科学管理功能。

关键字：以太坊、高并发、微服务、负载均衡、私募股权
中图分类号：TP3

Abstract

As Blockchain has developed rapidly in recent years, related applications have mushroomed as well. The blockchain application ecosystem is gradually forming, and most applications join the Blockchain network by accessing Blockchain public service agencies such as Ethereum and HyperLedger. The access of these applications relies on a single client node, which causes the traffic pressure of the entire application to be concentrated on a Geth client, making the Ethereum-based blockchain application's concurrent performance poor. At the same time, because blockchain technology is mostly used in the field of digital information transactions, and requires high data security, the architecture of a single Geth client also has shortcomings in disaster tolerance and reliability.

In view of the above problems, it is extremely necessary to design and implement a high-performance Ethereum service system based on the characteristics of the blockchain and related business scenarios. Therefore, this research designs the Ethereum back-end service architecture based on the idea of the microservices, builds an Ethereum service platform with multiple Geth clients based on Docker, designs a feedback load balancing strategy based on the running state of the Geth client in the Docker container, and finally implements A blockchain application platform for private equity transactions. The main work of this research is as follows:

1. Propose a design scheme of a private equity-oriented blockchain service platform based on the idea of microservices. Design a flexible application architecture based on business features and technical traits, which can meet business requirements with minimal resource consumption, and quickly expand the system at the moment of need.

2. Based on the characteristics of Ethereum, a state feedback load balancing strategy is proposed. This strategy considers the operating status of the blockchain nodes to establish a state space, and performs load distribution on nodes in the state space to achieve system reliability and high performance. This load balancing strategy is applied to build a flexible underlying blockchain service platform based on the idea of containerization technology. The platform will run the Ethereum client in the container as a single node, and combine the load balancing strategy to achieve reliability and high performance.

3. Implemented a private equity-oriented blockchain service system, including business functions such as account management, transfer management, and contract

issuance, as well as background module functions such as status monitoring and gateway management. Combined with the technical characteristics of the smart contracts of blockchain, it has realized the functions of automatic settlement and scientific management of private equity transactions.

Keywords: Ethereum, high concurrency, micro-services, load balancing, private equity
Chinese Library Classification: TP3

第一章 绪论

1.1 研究背景与研究意义

私募股权(Private Equity) 由美国投资家本杰明格雷厄姆创立，是一种非上市企业或单位进行的非公开项目招募活动[1]，此类招募活动不限于资金募集、项目招标等。私募股权源于金融领域，首先被用于面向特定对象释放企业私有股权以达到募集资金的作用[3]。私募股权市场广大，据二八定律知 20%的人持有 80%的社会财富，而 20%的人持有的财富中，有 80%属于私募产品[4]。随着经济与时代的发展，私募股权的范围越来越广，私募股权所募集的目标不再局限于资金，劳动力、资源、技术等等均成为募集对象[4]。私募股权交易需要考虑多方面因素，如数字资产的安全性、交易客户的隐私以及交易快速性。因此，构建针对私募股权业务场景的交易平台极为重要。作为复旦大学的科研合作单位，上海计算机软件技术开发中心建立了服务于尚信资本的私募股权交易平台。

私募股权交易平台是面向私募股权活动的中间交易平台，用户可以发布需求信息或者对特定需求进行投标。该平台极大得降低了私募股权活动中合作方的交易难度，同时由于平台的监管提高了合作方之间的信任度[6]。私募股权平台的构建需要考虑用户隐私、用户发布的项目的信息安全性、用户之间股权活动交易等方面问题[7]。同时，由于私募股权项目需求等信息的价值很高，私募股权平台需要考虑自身的权益问题，即不能使用户可以在利用平台提供的信息后越过平台直接进行合作。基于以上问题考虑，采用区块链技术作为底层技术。

区块链(BlockChain)是由中本聪于 2008 年提出的革命性技术[8]。该技术结合了加密、分布式存储和等技术。它可以保障私募股权交易中数字资产的安全性，同时在区块中记录了数字资产的流通过程，有效地保障了数字资产相关方地权益。近年来，区块链技术在数字资产交易领域等得到了极大的应用[9]。

目前，区块链技术在构建交易平台中的应用方法，主要采用接入区块链服务平台如以太坊(Ethereum)、超级账本(Hyperledger)等[10]。以太坊是由 Vitalik Buteri 于 2013 年 12 月提出的可编程区块链，该平台除了可基于内置的以太币(ether)实现数字货币交易，还提供了图灵完备的编程语 Solidity 以编写智能合约，首次在区块链领域提出了智能合约的概念[11]。目前，私募股权交易平台的构建中对以太坊的接入包括以下三个部分：

1）以太坊 Geth 客户端。Geth 客户端是以太坊网络提供的交互工具，应用

通过以太坊 Geth 客户端成为网络中的节点参与账户管理、合约管理和数据存取等操作。

2）智能合约。智能合约是一段运行在 EVM 中的程序，通过智能合约编写代币实现私募股权交易平台中的数字资产的价值兑换和交易。

3）后台服务模块。后台服务模块实现和封装了私募股权平台中的核心逻辑，即项目和需求的发布以及购买等操作，同时包括对以太坊网络的操作接口，方便用户的使用和私募股权平台管理员的管理。

其中，数字资产交易过程是整个平台最核心的部分，然而在实际运行过程中，经常出现交易效率低下以及交易失败的问题，主要有以下两方面原因：

1）以太坊 Geth 客户端。单一的 Geth 客户端服务能力有限，具体在于共识过程所需算力造成资源损耗，当整个应用的并发流量到达后 Geth 客户端需要同步完成请求，导致完成任务时间较长。

2）后台服务模块。私募股权交易平台核心业务在于数字资产的存取和交易。一般而言，从以太坊网络中读取信息的操作频率远大于写入信息的操作频率。另外，因为写入信息的操作最终需要将信息打包到区块，这一步耗时较长。因此，后台服务架构和任务分配模型会影响任务整体的执行效率。

以太坊在私募股权交易平台中的并发性、稳定性和安全性等问题是区块链应用过程中普遍存在的问题。同时，对于数字资产交易领域，应用平台的安全性、稳定性以及交易并发量都是非常重要的问题。针对私募股权交易平台，当前的后台服务模块未考虑交易平台中读写操作中，读操作占比较大，写操作占比较少的特点；同时，未考虑读写操作花费时间差别大，对资源需求不同的情况。另外，以太坊服务平台中单 Geth 客户端需要承受整个应用的流量压力，不仅会造成处理多并发请求较慢的问题，还可能导致因为网络故障或者服务器故障等原因造成的服务崩溃。

综合上述需求，本文提出了一个基于状态反馈负载均衡策略和以太坊的私募股权交易平台。该平台融合容器化技术、微服务思想以及云计算领域中的基于状态反馈的负载均衡模型，不仅极大的提高了区块链技术作为底层技术的稳定性、安全性和并发性能，同时使得区块链技术的应用更加便利和灵活。该服务系统中的业务设计是面向私募股权领域，基于区块链的智能合约设计代币作为平台通用的股权激励，将项目信息打包永久性存储到区块链中。同时，通过事件记录区块链网络中的交易过程。该系统的应用架构采用微服务应用架构，根据区块链读写数据特点构建相应的服务提供者。底层区块链节点与单个的服务均运行于容器中，提高系统的整体可靠性和系统的延展性。在底层区块链服务平台与后台服务之间，

设计和应用基于状态反馈的负载均衡策略，提高系统的并发性能。

1.2 国内外研究现状

私募股权交易可理解为数字资产信息交易的一种，因此其涉及第三方信任机制[1]。即交易双方互不信任，但由于共同信任第三方，因此他们之间依然可以完成交易。因此，在私募股权交易中，构建第三方交易平台非常重要。然而，第三方信任机制过于依赖第三方，当对平台信任度不够的情况下，交易便无法顺利完成。区块链技术的去中心化机制解决了第三方信任机制问题，信任机制由简单的第三方证明转移到工作量证明 PoW(Proof of work)、股东权益证明 PoS(Proof of States)等[8]。魏生等人在构建私募股权交易平台的过程中引入了区块链技术[12][13]，但他们在系统中使用区块链的方式仅将关键信息存储到区块链上，未考虑服务性能。

目前，针对企业级应用的并发性能的提升方法，主要有缓存、并发以及后台服务架构等方法，后台服务架构采用分布式架构能有效提升应用的并发能力。近年来，最流行的分布式服务架构主要为 SOA 架构和微服务架构。SOA 架构在各个领域有广泛的应用，郑福生等人基于 SOA 架构构建了面向电力通信系统的远程监测诊断系统[14]，Avila K 等人基于 SOA 构建了面向物联网领域的家庭健康检测系统[15]。但以上对 SOA 架构应用的案例说明了 SOA 架构面向的应用具有业务体系多且杂的情况，针对私募股权交易平台而言，其业务的划分粒度较大。微服务技术自 2010 年开始兴起的分布式应用架构技术，相对于 SOA 架构，该架构业务划分粒度更小，结合容器技术后在敏捷开发、自动化运维等领域应用更加广泛，其各个子服务独立开发部署的架构优点不仅可以提高项目开发效率，同时可以提高系统的扩展性和运行稳定性[12][13]。微服务技术在应用开发领域取得了巨大的成功，J Lawson 等人提出了一种基于微服务的通信系统[14]；A Balalaie 的研究说明微服务技术可实现自动化运维且是一种原生的云服务架构[15]。而微服务技术往往与容器化技术结合使用，J Stubbs 等人提出了一种使用 Docker 容器的分布式系统架构[16],M Amaral 评估了结合容器技术的微服务架构的表现，认为微服务结合容器技术能发挥其最大的性能优势[17]。

负载均衡发展历史悠久，有非常多经典的负载均衡算法如加权、轮询、最少连接等[21]。该类负载均衡最大的缺陷在于忽略了服务节点的特点以及当前状态。采用了将任务流进行分组，并且考虑对其进行优先级划分的方式得到了较高的负载效率[22]。负载均衡的目标在于使得系统稳定且资源消耗最少得情况下得到最少得任务完成时间[23]。等人提出的相空分析方法首先建立以服务器参数为坐标轴的相位空间，然后将服务节点的参数归一化投射到该相位空间中[24]。等人提

出了基于相位空间的负载均衡方法，充分考虑了节点的状态进行负载和调度[25]。

由上述可知，提升应用的并发性能的方法多种多样，具体方法的选择需要根据应用业务的特点，对上述方法进行选取、组合和改进。私募股权交易平台核心业务发布项目、浏览项目和购买项目可以分为读和写两种类型，这两种类型的任务的出现频率和对以太坊服务平台的性能影响均不同。但是这些方法，对业务的拆分并未考虑业务的性能特点和频率特点，并不符合私募股权平台的业务场景。因此，本文考虑私募股权平台业务自身频率特点和对以太坊服务平台的性能影响特点对服务进行读写拆分，提出面向私募股权的微服务后台服务系统，并将其应用到私募股权交易平台中。

1.3 主要工作和创新点

本文主要工作是分析了构建私募股权交易平台面临的困难，提出了以以太坊技术为底层核心技术的解决方案，并实现了面向私募股权的以太坊服务系统。在设计和实现系统的过程中，本文充分考虑私募股权交易中交易频率高，读写任务频率不同等特点，通过优化后台服务模块的系统架构，以及设计适合本系统的基于服务节点 CPU 和内存占比状态的反馈负载均衡策略，提高系统的可靠性、安全性和延展性，并通过性能对比实验对系统进行了评估和验证。

本文基于微服务思想对私募股权平台中的业务进行读写划分，并分类封装于不同的子服务，同时设计和实现了基于状态反馈负载均衡策略。最后本文实现了私募股权交易平台，并通过实验验证了系统的可用性、并发性等性能。本文研究的主要工作有以下几点：

1. 提出一种基于微服务思想的面向私募股权的以太坊服务平台设计方案。根据私募股权业务中读写任务频率不同，对资源消耗不同的业务特点和以太坊服务方式的技术特点设计灵活的应用架构，有效地提高了私募股权平台中以太坊管理相关业务的并发性能、系统稳定性和延展性。

2. 基于以太坊(Ethereum)的单 Geth 客户端服务的特点提出了一种基于状态反馈的负载均衡策略，该策略考虑区块链节点运行状态建立状态空间，对状态空间中的节点进行负载分配实现系统的可靠性和高性能。并将该负载均衡策略应用于基于容器化技术思想构建了一种灵活的底层以太坊服务平台。该平台将以太坊客户端运行于容器中作为单一节点，结合负载均衡策略实现高并发性。

3. 实现了基于以太坊和微服务的私募股权交易平台，包括账户管理、发布项目、购买项目等业务功能。并结合基于微服务的后台服务模块和基于状态反馈的

负载均衡策略，提高了系统的稳定性、并发性以及延展性能。

1.4 篇章结构

本文将围绕面向私募股权的以太坊服务系统设计与性能优化等方面的研究与实现安排六个章节展开介绍。

第一章，绪论。本章介绍本文的研究背景、研究目的与意义，简要介绍了私募股权相关背景和以太坊技术在应用中服务并发能力较差稳定性较弱的现状，并简述了本文的主要工作与创新点。

第二章，相关工作与技术背景。本章主要介绍本文提出的区块链服务系统涉及的相关技术，从以太坊技术、微服务架构和负载均衡技术三个方面介绍了其技术原理以及应用方式。

第三章，介绍了考虑运行 Geth 客户端的容器的状态建立的状态反馈负载均衡策略，包括负载均衡依据、任务调度模型、考虑私募股权交易业务的任务优先级模型和任务分配算法描述等。然后本章对该负载均衡策略进行了模拟实验，验证其并发能力相对未使用负载均衡的服务系统的更强。

第四章，基微服务的私募股权后台服务架构设计与实现。本章描述了私募股权平台的后台服务架构，重点介绍了多 Geth 客户端的以太坊服务平台以及读写子服务的设计与实现。

第五章，私募股权交易平台的实现与评估。，本章介绍了基于前两章内容的面向私募股权的区块链服务系统，并设计实验对系统进行了功能说明与性能评估。

第六章，总结与展望。本章总结本文总体的研究内容和方法与技术的局限性展望未来工作，并对下一步工作做出规划。

第二章 相关工作及技术背景

本章首先介绍了以太坊以及一般企业级应用使用以太坊的方法和架构，然后介绍了业内流行的微服务架构解决方案及其特点。最后，本章介绍了云计算负载均衡策略应包括的各个模块，基于以上知识设计面向私募股权的以太坊服务平台。

2.1 以太坊

以太坊系统是一种无需中央管理和协调机制的分布式系统，该系统可以运行智能合约和小型应用，其创造者的初衷是开发一套可以自治的“世界计算机”[15]。它在比特币的基础上更进一步，不仅可以在全网节点上验证和存储交易数据，还可以在全网所有节点中运行智能合约代码，节点使用 EVM(Ethereum Virtual Machine)运行智能合约[32]。以太坊同时具有分布式数据存储和计算的能力[33]。

以太坊的应用方式主要分为两类：1、使用 Metamask 等浏览器插件工具连接以太坊网络；2、基于官方提供的 Geth 客户端连接以太坊网络。上述连接的以太坊网络可以是以太坊主网，也可以是 Rinkeby、Kovan 等测试网络。对于第二种应用方式，也可以基于 Geth 客户端构建自己的以太坊网络—私链，其中 Geth 客户端作为以太坊网络中的单个节点。

2.1.1 Geth 客户端

以太坊的使用是通过以太坊客户端(Ethereum client, Geth)实现，用户通过以太坊客户端连接以太坊网络中的其他节点，并参与区块同步、挖矿、交易验证的工作[33]。以太坊上的所有节点地位相同，同时没有中心协同和管理节点。在成为以太坊节点后可以进行连接以太坊网络、查看以太坊区块链、发布交易和智能合约、运行智能合约以及挖矿等工作[33]。

Geth 客户端是使用 Go 语言编写的实现了 Ethereum 协议的客户端软件，它是目前用户最多，使用最广泛的以太坊客户端。如图 2.1 的以太坊应用架构中可以看出 Geth 客户端与以太坊网络的关系，Geth 客户端对外部应用而言是应用与以太坊网络交互的工具和接口；对以太坊网络而言，Geth 客户端所运行的节点是组成以太坊网络的节点。换言之，对于以太坊网络而言，Geth 客户端是网络中对数据进行挖矿和保存到区块的矿工。

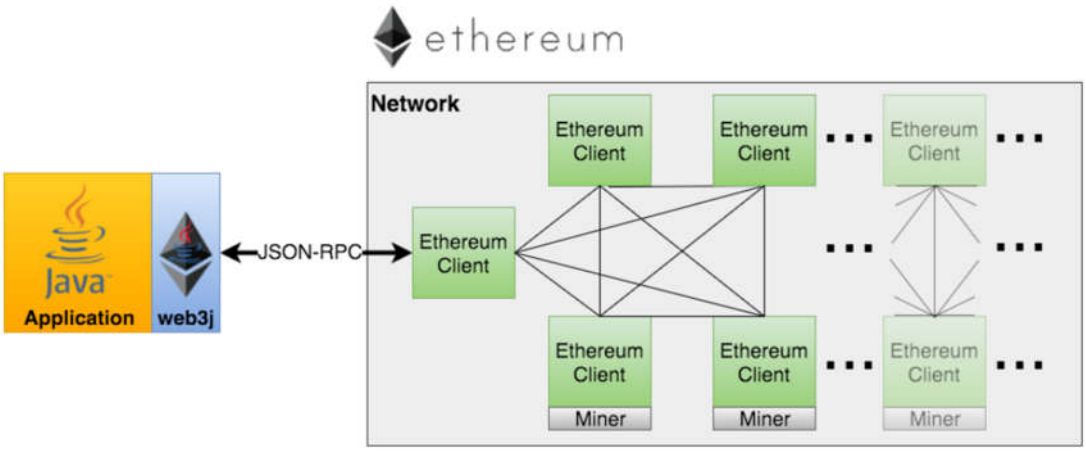

图 2.1 Geth 客户端形成的以太坊网络

2.1.2 web3j

web3j 是一个高度模块化、反应式、类型安全的 Java 和 Android 库，用于处理应用并与以太坊网络上的 Geth 客户端(节点)的交互问题。如图 2.1 所示，web3j 通过 JSON-RPC 的方式与 Geth 客户端进行交互，实现应用与以太坊网络中的数据通信。web3j 库仅提供了基本操作接口，在实际应用过程中，还需要根据应用的实际业务特点，定制智能合约运行到以太坊虚拟机(EVM)中，然后对其进行调用处理。

web3j 的使用一般是作为一个工具模块集成到应用中，这使得业务耦合度升高、延展性降低等问题。对于私募股权交易平台，该模块与应用的集成使得应用无法按照读写业务的实际需要进行扩展。因此，考虑基于微服务思想，按照读写业务的分类集成子服务，提高服务效率。

2.2 微服务架构

微服务是一种架构概念，旨在通过将功能分解到各个离散的服务中以实现对解决方案的解耦[17]。它的主要作用是将功能分解到离散的各个服务当中，从而降低系统的耦合性，并提供更加灵活的服务支持。围绕业务领域组件来创建微型应用，这些应用可独立地进行开发、管理和迭代[18]。在分散的组件中使用云架构和平台式部署、管理和服务功能，使产品交付变得更加简单[19]。

2.2.1 微服务应用架构

微服务架构与传统的单体应用架构相比开发效率更高，减少了开发中的相互

等待和冲突；维护更简单，代码耦合度极大的降低使得微型应用更加简单；更加灵活，构建时间更短；稳定性增强，不会因为一个小问题导致整个系统崩溃；扩展性增强，可以根据需求，对特定的功能组件进行扩展以提高系统的并发性。

微服务架构的具体实践中需要解决如下四个问题：

1）首先是客户端如何访问服务。

对此一般采用在服务和 UI 之间设置名为 API Gateway 的代理机构，由该机构提供服务的统一入口，聚合后台服务，节省访问流量并提供过滤、流量控制等 API 管理功能。

2）是服务间的通信方式。

服务之间的调用分为同步调用和异步调用，同步调用一致性强，但性能较差，如 REST API 的方式基于 HTTP 实现，在调用的过程中会有大量的多余通信消耗。异步调用往往通过消息队列实现，此方式在分布式系统中应用广泛，它既能降低系统耦合，又能实现调用的缓冲，有效的防止了系统因流量过大而崩溃的情况，不过异步调用也会导致一致性的减弱。

3）服务的实现，即服务如何管理，服务之间如何管理。

一般有两类做法，基于客户端的服务注册与发现和基于服务端的服务注册与发现如图 2.5 所示。当服务上线时，服务提供者将自己的服务信息注册到 ZK（或类似框架），并通过心跳维持长链接，实时更新链接信息。服务调用者通过 ZK 寻址，根据可定制算法，找到一个服务，还可以将服务信息缓存在本地以提高性能。当服务下线时，ZK 会发通知给服务客户端。

4）服务崩溃时如何处理。

对于此问题一般有限流、熔断、负载均衡等方式处理。其中设计合适的负载均衡算法不仅可以应对服务崩溃，可以提高系统的并发性能。

综上所述，一个完成的微服务应用除业务组件外，还包括负责处理客户端访问服务的 API 网关；负责管理服务注册与发现中心；服务与服务之间的通信和熔断保护机构。

2.2.2 微服务解决方案

微服务解决方案是业内实现微服务架构的具体实施方案，本小结介绍业内最流行的微服务解决方案 Spring Cloud Alibaba，并介绍针对微服务应用架构中需要

解决的四个问题的部件。

微服务解决方案众多，其中由阿里巴巴开源的组件和云产品组成的 Spring Cloud Alibaba 尤其突出[39]。2018 年 10 月 31 日，Spring Cloud Alibaba 正式入驻了 Spring Cloud 官方孵化器，并在 Maven 中央库发布了第一个版本。Spring Cloud Alibaba 致力于提供微服务开发的一站式解决方案。此项目包含开发分布式应用微服务的必需组件，方便开发者通过 Spring Cloud 编程模型轻松使用这些组件来开发分布式应用服务。该解决方案提供了一系列组件来解决以上提到的微服务应用遇到的四个问题[40]。

针对微服务架构中需要解决的四个问题，Spring Cloud Alibaba 提供了以下组件解决：

1）Spring Cloud Gateway

Spring Cloud Gateway 是 Spring 官方基于 Spring 5.0，Spring Boot 2.0 和 Project Reactor 等技术开发的网关，Spring Cloud Gateway 旨在为微服务架构提供一种简单而有效的统一的 API 路由管理方式。Spring Cloud Gateway 不仅提供统一的路由方式，并且基于 Filter 链的方式提供了网关基本的功能，例如：安全，监控/埋点，和限流等。

2）Nacos

服务注册中心，它是服务，其实例及元数据的数据库。服务实例在启动时注册到服务注册表，并在关闭时注销。服务和路由器的客户端查询服务注册表以查找服务的可用实例。服务注册中心可能会调用服务实例的健康检查 API 来验证它是否能够处理请求。Nacos 致力于帮助您发现、配置和管理微服务。Nacos 提供了一组简单易用的特性集，帮助您快速实现动态服务发现、服务配置、服务元数据及流量管理。

3）RocketMQ

RocketMQ 是一个消息中间件, 消息队列作为高并发系统的核心组件之一，能够帮助业务系统解构提升开发效率和系统稳定性。消息中间件中有两个角色：消息生产者和消息消费者。RocketMQ 里同样有这两个概念，消息生产者负责创建消息并发送到 RocketMQ 服务器，RocketMQ 服务器会将消息持久化到磁盘，消息消费者从 RocketMQ 服务器拉取消息并提交给应用消费。

综合以上三种组件以及服务之间 REST API 的通信方式，Spring Cloud Alibaba 可以作为私募股权交易平台的基础解决方案，在其基础上定制化读写服务提供者与服务消费者，实现后台服务功能。

2.3 状态反馈负载均衡策略

在微服务等分布式应用架构中，为提高系统的可靠性、容错性和并发性，往往存在多种相同的服务节点共同对外提供服务。任务在这些服务节点中的分配方式，即为负载均衡技术[24]。负载均衡策略包含任务负载任务的划分方式、任务的调度以及服务节点的架构等方面，其目的在于减少总的任务完成时间，提高服务资源的利用率以及保证良好的系统负载均衡度[25]。经典的负载均衡策略包括轮询、加权、最少连接等，此类负载均衡算法最大的问题在于未考虑到服务节点的状态[29]。在云计算领域，状态反馈负载均衡算法充分考虑服务节点的状态，根据节点状态以及任务类型进行负载分配，可以实现更好的负载效果[28]。

状态反馈负载均衡策略的设计包含四个方面，首先是负载均衡依据，其次是任务调度模型，第三是构建任务优先级模型，最后设计任务分配模型。

在负载均衡依据中，选取对服务节点状态影响较大的参数如内存占用率、CPU 占用率的归一化建立服务节点状态空间，然后将服务节点的状态向量投射到该状态空间中。若服务节点在状态空间中的位置距离原点较近，说明其资源消耗度较小；如果位置距离原点较远，则说明该服务节点的资源消耗度较高，处于资源占用状态。

2.4 小结

本章首先介绍了以太坊相关知识，包括官方提供的用于交互和管理 Geth 客户端，以及常见应用使用以太坊的方式。在此基础上加以改进，构建多 Geth 客户端的以太坊服务平台。本章还介绍了一般的微服务应用架构的必备组件和业内最流行的微服务实践方案 Spring Cloud Alibaba 及其重要组件。最后，本章介绍了状态反馈负载均衡策略指定过程中需要考虑的因素。

第三章 基于状态反馈负载均衡策略

本章首先介绍基于状态反馈的负载均衡策略，其中包括负载均衡依据、任务调度模型、任务优先级模型和任务分配模型。

3.1 基于状态反馈的负载均衡策略设计

3.1.1 负载均衡依据

以太坊的 Geth 客户端在 Docker 中的运行时，容器的工作状态可以表示为一个参数向量，如 a = (a1, a2, …,an)。当任务分发到该容器执行时，容器的状态会改变，因此，通过参数向量的变化可以看出容器当前的服务能力和与工作状态。在 Geth 客户端运行的容器中，CPU 和内存的占比受服务任务的影响较大，因此本文选取容器的这两个状态参数建立 2 维平面，然后将 Geth 服务节点按照对应参数映射到该平面。

假定 Geth 服务节点的数量为 n，在任务分配调度时考虑 CPU 和内存的状态两个因素。CPU 和内存在对 Geth 所服务的任务的重要性不同，因此假设它们在调度系统中的权重为 w1 和 w2，显然 w1+w2=1。则 Geth 服务节点在 2 维平面的点集为

$$U=\{x_{i1}, x_{i2}\}, 1 \le i \le n, 0 \le x_{i1} \le w_1, 0 \le x_{i2} \le w2 \qquad (3.1)$$

Geth 服务容器在二维平面投影的重心位置反映了容器集群的整体负载，如果服务节点的位置均聚集在重心附近，则说明该系统的负载均衡性良好；否则，则说明负载均衡效果差，因此服务节点的分散投影说明了负载并不均衡。

综上所述，可以根据重心定义一个直观的负载均衡参数作为负载均衡依据，定义其为健康参数 LBH。假设 Geth 服务节点的数量为 n，其在 2 维状态平面投影的重心为 G(X1, X2)，则各服务节点的投影到重心的举例平均值为

$$\frac{\sum_{i=1}^{n}\sqrt{(x_{i1}-X_1)^2+(x_{i2}-X_2)^2}}{n} \qquad (3.2)$$

由上式可知，当 Geth 服务节点中未执行任务的节点与因为执行任务导致节点满载的节点各占 50%时，以太坊服务平台的整体负载均衡性最差。Geth 服务

节点的 CPU 和内存两参数在二维平面中的重心位置为$\left(\frac{w_1}{2},\frac{w_2}{2}\right)$，各服务节点到重心位置的距离的最大值为$\frac{\sqrt{{w_1}^2+{w_2}^2}}{2}$。考虑各服务节点到重心距离的平均值，作归一化处理后即为负载均衡健康参数 LBH

$$LBH=\frac{\sum_{i=1}^{n}\sqrt{(x_{i1}-X_1)^2+(x_{i2}-X_2)^2}}{n\frac{\sqrt{{w_1}^2+{w_2}^2}}{2}} \tag{3.3}$$

负载均衡健康参数是衡量以太坊服务平台中的 Geth 服务节点的负载均衡健康指标，通过该参数可以量化服务平台的负载均衡状况。理想情况下，当所有的服务节点的负载情况相同时，它们在二维平面的投影集中到重心位置，此时健康参数 LBH = 0。因此，LBH 的理论取值范围是[0, 1]，该健康参数越小，服务平台的负载均衡性越好。

3.1.2 任务调度模型

本文设计的调度模型首先假定任务为将 m 个任务调度分配到 n 各服务节点，其中 n<m；基于私募股权交易的应用业务场景，调度任务可根据任务类型分类为读写两种任务，对于写任务，可进一步划分。由于读任务仅通过服务节点中的客户端调用相应的 API，故资源消耗较少，而写任务需要等待数据打包、验证等操作，因此耗时更长，资源消耗也更多。假设任务集合为$T(n)=(t_1,t_2,...,t_n)$，其中t_i为第 i 个子任务。任务t_i由参数表示即

$$t_i=(t_{i\_\cos t},t_{i\_m},t_{i\_memory},t_{i\_cpu}) \tag{3.4}$$

在上述公式中，$t_{i_\cos t}$是任务完成所期望的花费，t_{i_m}代表任务的大小，t_{i_memory}和t_{i_cpu}分别代表完成任务所需的内存和 CPU 占用的需要。

m 个容器资源可以表示成$CON(m)=(con_1,con_2,...,con_m)$，$con_j$表示第 j 个容器，其属性向量为

$$con_j = (con_{j\_cost}, con_{j\_power}, con_{j\_memory}, con_{j\_cpu}) \tag{3.5}$$

公式 3.5 中，con_{j_cost} 是容器处理任务的花销，con_{j_power} 是容器当前的资源能力，con_{j_memory} 和 con_{j_cpu} 是容器当前剩余内存和 CPU 的占比。

私募股权平台的任务在 Geth 服务节点上运行的时间可以表示为

$$ET_{ij} = \frac{t_{i\_m}}{con_{j\_power}} \tag{3.6}$$

任务 t_i 在服务节点 con_j 上的期望完成时间为

$$RT_{ij} = s_j + ET_{ij} \tag{3.7}$$

其中 s_j 为任务开始时间，任务完成的总时间可以表示为

$$RT = \sum RT_{ij} \tag{3.8}$$

本调度模型的目标是让系统承受尽量大的并发，函数与约束条件为 min(RT)；

约束条件为：

$$\begin{cases} t_{i\_memory} <= con_{j\_memory} & i = 1,2,\cdots,n \\ t_{i\_iomemory} <= con_{j\_iomemory} & j = 1,2,\cdots,m \end{cases} \tag{3.9}$$

即仅在约束条件满足的时候才能将任务分配给相应的节点。

3.1.3 任务优先级模型

在区块链环境下，任务可以分为读任务 RT(Read Task)和写任务(Write Task)，读任务不消耗容器资源因此优先使用第二区域的容器，在第二区域容器均无空闲时选择第三区域的容器。对于写任务，可以分为发布合约任务 DT(Deploy Task)，发布合约任务较少，操作账户任务 MAT(Manage Accounts Task)次之，调用合约任务 MCT(Manage Contract Task)较多。本文根据任务出现的频率对其先后执行顺序，优先执行处理出现次数较少的任务。

任务优先级模型算法具体描述如下：

读任务与写任务分开执行；

读写任务可能在第二区域冲突，此时因为读任务耗时短且不消耗系统资源，优先执行；

读任务按时间先后顺序执行；

写任务分为发布合约任务，操作账户任务和调用合约任务，其在实际业务场景中出现频次依次增加，同时重要性会减小，因此优先级为降序排列；

具体任务优先级图示如下：

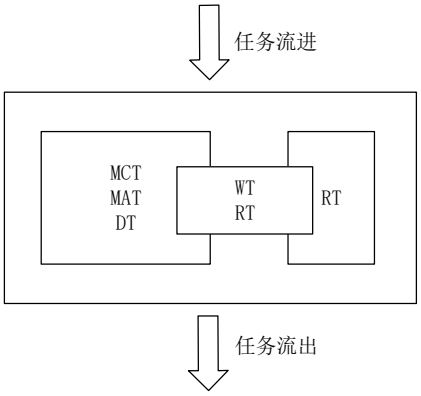

图 3.4 任务流优先级图示

3.1.4 任务分配模型与算法描述

任务的优先级决定了任务的调度顺序，在本文中，我们根据容器的健康参数将容器分为三个区域，由此，我们可以将读任务交给第二区域的容器处理，写任务交给第一区域的容器处理。第三区域的容器，进行定时回收重启处理。在各个区域内部，本文结合区块链场景下的任务的性质以及资源的使用和分配情况，构建动态优先级。

因为容器的状态影响其执行写操作，因此我们将容器按状态分为两类，具体映射到状态平面空间为 I 区、II 区。以公式(3)得到的数值 $D\mathrm{is}_{ave}$ 为分界线。

我们将容器状态平面分为两个部分，I 区代表安全区，II 区代表非安全读区。

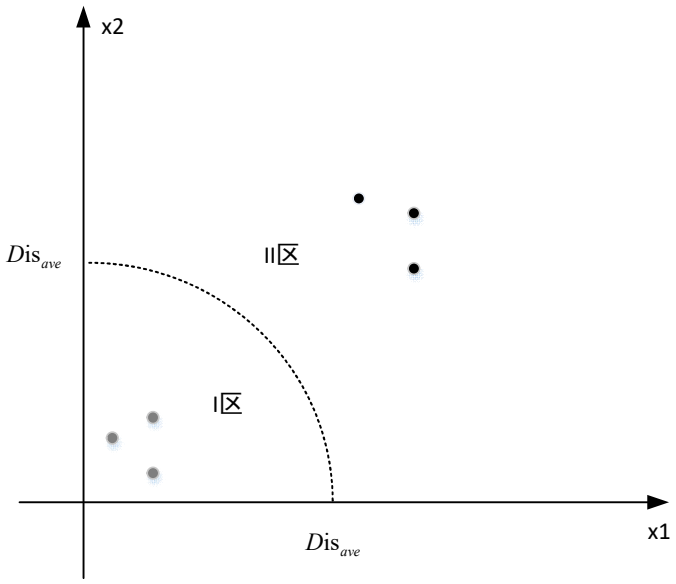

图 3.5 容器状态平面分区示意图

任务分配模型描述：

对于写任务，优先分配到 I 区，当 I 区没有节点时分配到 II 区；

对于读任务，优先分配到 II 区，当 II 区没有节点时分配到 I 区；

本文设计的面向区块链微服务化场景下的任务调度算法描述如下：

将任务根据任务的读写性质进行分类并且按时间排序；

将任务依次动态的分配到适合的区域中的容器中；

更新容器的状态信息，动态将任务分配到其中；

将第三分区中的容器依次进行重启操作，回收资源；

检查健康状态参数，若健康状态参数为负数，则暂停如任务分配，等待知道健康状态为正数则继续分配任务；

返回流程开始下一个任务的调度；

3.2 实验

3.3 本章小结

本章首先介绍了考虑 Geth 服务容器状态的负载均衡策略的设计，具体包括负载均衡依据、任务调度模型、任务优先级模型和任务分配算法。然后实验(未确定)。

第四章 基于微服务的私募股权后台服务架构设计与实现

为了将基于状态反馈的负载均衡策略应用到私募股权交易平台，需要分析私募股权交易平台的核心业务特点，同时考虑以太坊的服务场景设计相应的后台服务架构。本章首先介绍私募股权交易平台的核心业务及其特点，然后设计多 Geth 客户端的以太坊服务平台，最后基于微服务设计后台服务组件。

4.1 私募股权交易核心业务

私募股权交易平台所交易内容为数字资产，该数字资产可以简化理解为项目信息的出售和购买。本小结将介绍私募股权交易平台的核心业务流程。

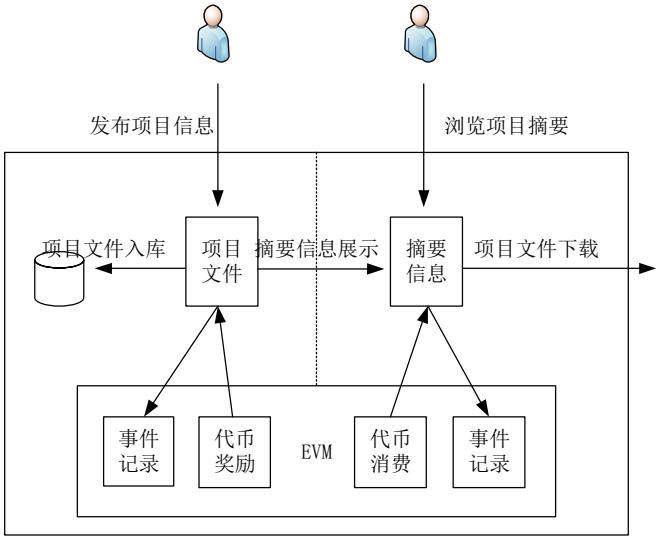

图 4.1 私募股权业务架构图

如图 4.1 所示，私募股权的业务场景下，用户主要分为两类，一类用户发布项目信息招募项目合作人；另一类用户通过浏览平台提供的项目摘要选取目标项目，在对项目感兴趣的情况下使用本平台代币购买项目。其中，发布项目信息可以得到相应的代币奖励，本平台所有交易均通过智能合约代币，项目发布、浏览项目和下载文件等操作均通过智能合约中的事件机制进行记录。

该业务场景流程图如图 4.2 所示，主要分为以下三种类型：

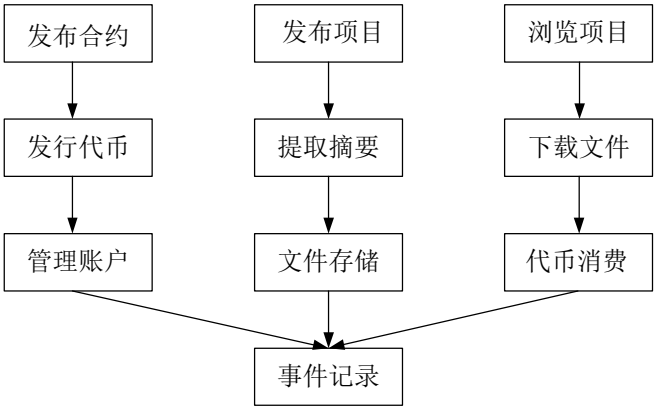

图 4.2 私募股权核心业务流程图

1）管理流程。

管理流程主要包括三个核心步骤，首先是发布智能合约，让实现编写好的智能合约运行于 EVM 中；其次是发行本系统的通用型代币；最后是管理账户。其中管理账户具体包括创建账户、给账户发币、账户转账等操作。

2）发布流程。

发布流程是在本平台发布项目的用户逻辑流程，主要包括三个核心步骤，用户发布项目到平台；后台提取项目摘要展示给需要购买项目的用户；项目文件存储到文件系统中。

3）购买流程。

购买流程针对的是购买项目的用户。此类用户首先在平台浏览需要购买的项目摘要，通过摘要判断需要购买后下载项目文件，同时消耗自己的代币进行购买。代币的来源通过充值后由智能合约发布。

4.2 私募股权平台后台服务架构

私募股权交易平台的后台服务架构主要分为两个部分，首先是底层的多 Geth 以太坊服务平台，其次是基于微服务思想与 web3j 工具的后台服务架构。

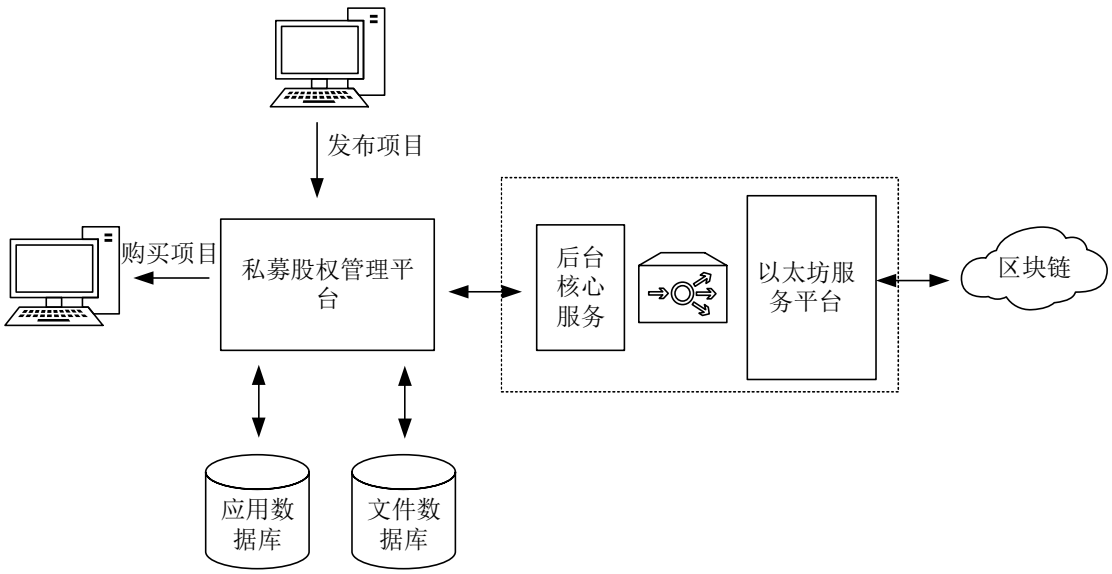

图 4.3 私募股权平台整体架构图

如图 4.3 为私募股权交易平台整体架构图，整体应用架构包括数据存储、前端交互、以太坊后台服务和底层的以太坊服务平台。本文工作主要集中于以太坊服务平台与后台核心服务模块。

4.2.1 系统设计目标及原则

以太坊服务平台的系统设计应该遵循微服务架构设计原则，具体如下：

1）扩展性与伸缩性

扩展性是指系统在业务需求的需要增加的情况下能够方便快速扩展的能力；伸缩性是指指不断向集群中添加服务器来缓解不断上升的用户并发访问压力和不断增长的数据存储需求。扩展性的实现有两种方式，首先是使用消息队列进行解耦，在服务之间传递通信；其次是将业务进行拆分复用，在本架构中即将业务拆分为读写业务。

2）易维护性

本设计涉及大量的分布式组件，系统的维护工作应该尽可能地科学和简单，避免大量地维护工作带来地开销和负担。

3）高性能和资源利用率

系统的响应速度吞吐量等指标应满足高性能要求。同时，在满足系统性能要求的情况下尽可能少使用资源，使得资源地利用率达到最高。

4.2.2 多 Geth 客户端以太坊服务平台

多 Geth 客户端以太坊服务平台的构建基于 Docker 容器，通过容器编排工具 docker-compose 进行编排管理。如图 4.4 所示为该服务平台的整体架构，该服务平台的中的节点从功能上可以分为三类：状态监控、以太坊服务节点、负载均衡模块，所有的 Geth 客户端使用的账户数据包括区块数据统一管理。

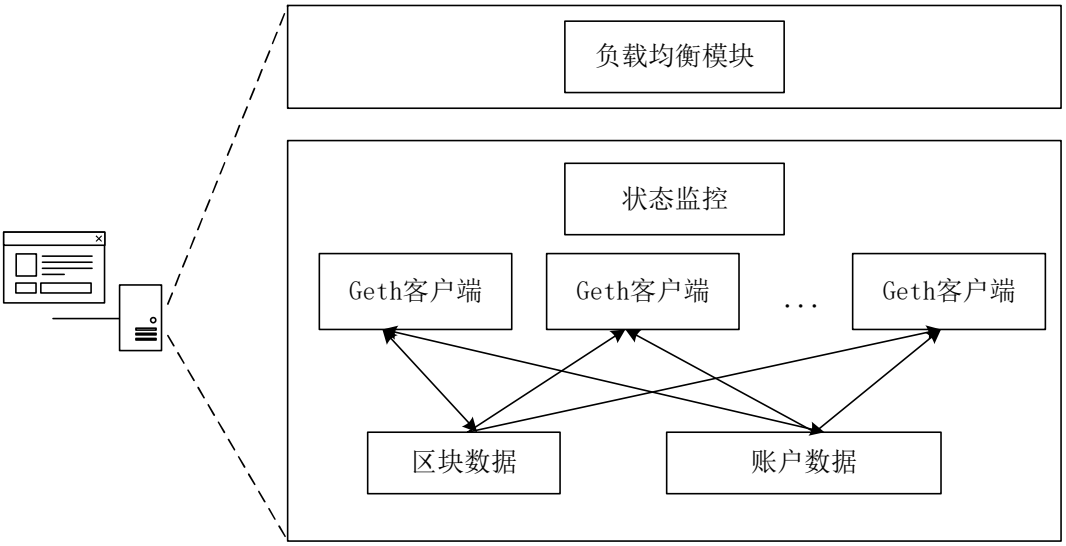

图 4.4 以太坊服务平台架构图

下面分层对以太坊服务平台的设计进行说明：

1） 数据层。

数据层的数据主要包括两部分：账户数据——以太坊中的账户的私钥及地址数据和区块数据。使得所有 Geth 客户端使用同一份数据的方式在于构建以太坊服务平台时将容器内的数据存储位置统一映射到一个固定位置。

2） Geth 客户端。

Geth 客户端是以太坊官方提供的与以太坊网络进行交互的工具，将该客户端运行于 Docker 容器之内，作为网络中的节点使用。该类节点负责打包数据、创建账户等以太坊功能性操作，可以手动扩展该类节点的数量。

3） 状态监控。

状态监控容器节点基于 eth-natstat 镜像构建，通过该模块观察节点的运行状态以及以太坊服务平台中的区块数量等信息。

4） 负载均衡。

负载均衡模块基于 SpringBoot 框架构建并运行于 Docker 容器中，该模块用于根据容器状态管理 Geth 客户端的上线与资源回收，实现第三章的负载均衡策略，提供任务的具体分配功能。

4.2.3 基于微服务的后台核心组件

私募股权平台的以太坊交互模块的构建基于 Spring Cloud Alibaba 实践方案。如图 4.5 所示为该后台模块的整体架构，从微服务角度出发该后台服务系统具有统一的接口管理中心—接口网关、服务管理机构—服务注册中心以及用户任务消息缓存的消息队列模块。从业务角度出发，根据私募股权交易平台的业务特点划分读写两类业务并分别封装成独立的子服务。纵向划分为服务提供者与服务消费者，在提供者与消费者之间使用熔断机制保障单个服务的失败不会导致整体应用的崩溃。

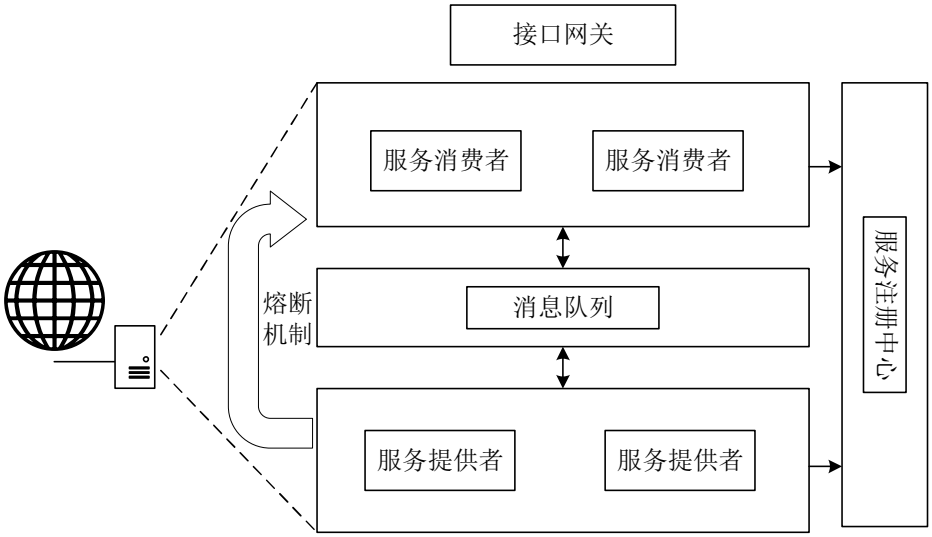

图 4.5 私募股权交易平台核心业务后台架构

系统功能角度说明：

1）服务注册中心。

服务注册中心是系统管理各个微型服务的模块。在本系统中，一个服务随时可能下线，也可能应对临时访问压力增加新的服务节点。以及服务与服务之间的感知，均有此模块实现。

2） 接口网关。

接口网关对后台服务的接口进行管理，并对外提供统一的调用接口。接口在

调用不受后台单个服务的上线或下线影响。接口网关的作用可以总结为：提供统一的服务入口，让微服务对前台透明；聚合后台的服务，节省流量同时提升性能；提供安全，过滤，流量控制等 API 管理功能。

3）消息队列。

消息队列是服务与服务之间的异步调用方式，它既能减低调用服务之间的耦合，又能成为调用之间的缓冲，确保消息积压不会冲垮被调用方，同时能保证调用方的服务体验，继续干自己该干的活，不至于被后台性能拖慢。

4）服务消费者。

服务消费者是轻量的，根据业务特点分别封装了读写相关操作，在接口网关对服务消费者的调用的时候即可实现第一级负载调度，本级负载调度可采用轮询或加权等方式，对本系统性能影响较小。

5）服务提供者。

服务提供者重量的耗时的，提供直接操作以太坊客户端的功能。在服务消费者和服务提供者之间使用消息队列进行解耦和做任务缓冲。服务者和提供者均由服务注册中心进行统一管理。

系统业务角度说明：

1）写服务

写服务是私募股权核心业务中需要将信息写入区块的业务，具体包括发布智能合约、发布项目、购买项目、发布需求和回答需求，另外还包括对账户管理与交易相关的操作，具体包括为某账户添加代币、代币转移和代币删除操作。

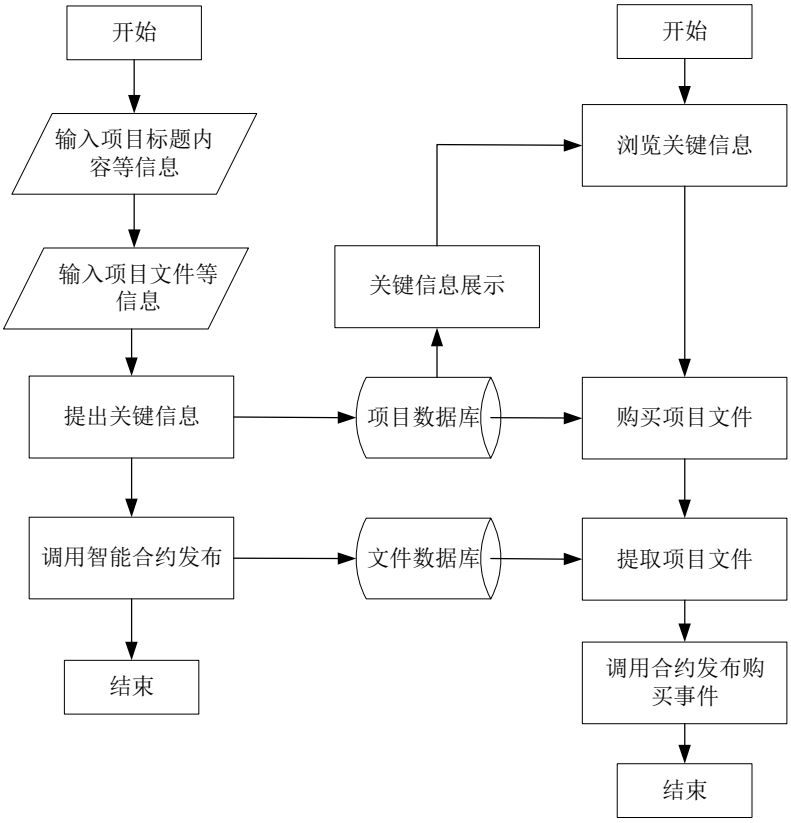

图 4.6 写服务核心流程

写服务的核心流程如图 4.6 所示，其中仅说明了项目发布与购买流程，需求的发布与回答流程相似，故不作赘述。

2）读服务

读服务是私募股权业务中需要从以太坊中读取信息的服务集合，具体包括创建账户、解锁账户、获取所有账户、通过 hash 值获取交易和区块信息、获取某账户的代币值、获取网络中的区块数量和获取 gas 价格等。

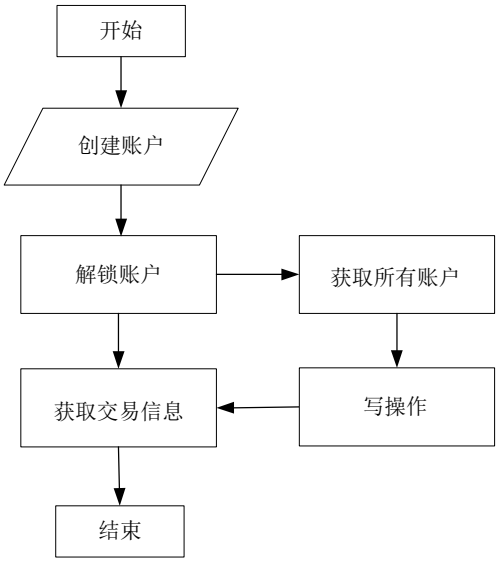

图 4.7 读服务核心流程

读服务与管理服务的核心流程如图 4.7 所示，其中写操作包括写服务中的发布项目、发布需求等操作。

4.3 私募股权后台服务实现

基于前面两节对私募股权交易平台的后台服务的设计，本节介绍私募股权交易平台后台核心服务的具体实现。具体从两个方面介绍，首先介绍多 Geth 客户端以太坊平台的实现，然后介绍私募股权后台服务的实现，此处仅介绍读写子服务的具体实现。

4.3.1 多 Geth 客户端以太坊服务平台实现

多 Geth 客户端以太坊服务平台的基于 Docker 实现，在此基础上使用 docker-compose 作为容器编排工具。服务节点按照功能分为普通的 Geth 服务节点与状态监控节点。下面分别进行介绍：

1）Geth 服务节点

该类节点基于 ethereum/client-go 镜像构建。首先，基于 ethereum/client-go 编写 Dockerfile 文件用于构建镜像，然后再 docker-compose.yml 中构建相应的 Geth 服务镜像。

代码 4.1：Geth 服务节点构建

```
1.  ethminer:
2.      build: monitored-geth-client
```

```
3.      restart: failure # 容器失败时重启
4.      links:
5.        – netstats # 状态监控节点
6.      entrypoint: /root/start.sh
7.      volumes: # 数据卷
8.        - ./files/password:/root/files/password:ro
9.        - ./files/genesis.json:/root/files/genesis.json:ro
10.       - ./files/keystore:/root/.ethereum/devchain/keystore:rw
11.     command: '--datadir=~/.ethereum/devchain --rpcapi
    "db,personal,eth,net,web3" --rpccorsdomain="*" --networkid=1618 --rpc
    --rpcaddr="0.0.0.0" # 容器启动命令
```

代码 4.1 为 geth 服务节点的 docker-compose 配置文件， 其中 build 对应的是构建镜像的 Dockerfile 文件所在位置，netstats 为状态监控节点。再数据卷映射中，将创世区块以及私钥位置映射到一个统一的位置，保障各容器使用同一份数据。Command 是容器的启动命令，其中包括设置数据存放位置“--datadir”，对外开通的 rpc 接口“--rpcapi”等信息。

2）状态监控节点

该节点基于开源项目 Ethereum Network Stats 构建，该开源项目广泛应用于以太坊网络监控。与 Geth 服务节点类似，首先编写 Dockerfile，然后构建镜像。

代码 4.2：Geth 服务节点构建

```
1.  netstats:
2.      build: eth-netstats
3.      restart: always
4.      container_name: netstats
5.      environment:
6.        - WS_SECRET=eth-net-stats-secret
7.      ports:
8.        - "3000:3000"
```

代码 4.2 为监控节点 netstat 的 docker-compose.yml 配置文件，其中 eth-netstats 是 Dockerfile 所在位置。对外开放接口为 3000，当节点成功运行后可以于该地址观察到以太坊网络的工作情况，具体界面如图 4.8 所示：

图 4.8 以太坊服务平台状态监控界面

以太坊服务平台模块还包括负载均衡模块，即将读写任务合理分配到不同的 Geth 节点的模块。负载均衡模块针对主要实现两个功能，首先是读取容器的状态数据，然后是通过对容器的状态计算得到下一个任务访问的 Geth 客户端；针对任务，该模块从 Rocket 中读取任务组后判断任务的类型，然后根据第三章的状态反馈负载均衡策略执行任务。主要函数核心代码如下所示：

代码：负载均衡

输入：

服务节点的状态数组：*gethStats*；

任务数组：*tasks*；

输出：

下一个任务：*targetTask*；

目标服务节点：*targetGeth*；

核心代码：

1. LoadTaskToGeth(GethStat[] gethStats, Task[] tasks)

2. int lbh = 0; //初始化健康参数 lbh

3. List<GethStat> gethPool; //初始化可用服务节点池

***4.** for gethStat in gethStats*

***5.** //x1 代表 geth 的内存占用率，x2 代表 cpu 占用率*

```
6.      //Disave 的定义见 3.2 节
7.      lbh =lbh + Disave – Math.sqrt(gethStat.x1^2 + gethStat.x2^2)
8.      if Disave – Math.sqrt(gethStat.x1^2 + gethStat.x2^2) < 0
9.      then gethPool.add(gethStat)
10. if lbh < 0 then goto 3 //如果健康指数小于 0 持续回收服务节点
11. List<Task> taskQueue //初始化任务执行序列
12. for task in tasks
13.    if task.type == w
14.        then taskQueue.add(task)
15.    else
16.        taskQueue.addFirst(task)
17. return gethPool.get(0), taskQueue.get(0)
```

4.3.2 私募股权后台服务实现

私募股权交易平台的核心后台服务主要是以太坊相交互的模块，包括读写两个模块。该后台服务整体架构基于微服务实现，以 Spring Cloud Alibaba 为实践架构。Spring Cloud Alibaba 提供了 API Gateway 网关、nacos 服务注册中心、RocketMQ 消息队列等模块，此类模块的构建没有创新之处，因此本小结仅介绍跟业务相关的读写子服务。

4.3.2.1 写子服务实现

写子服务分为主要分为两个部分实现，第一部分是写子服务消费者，该部分接受业务平台发送的 HTTP 应用请求，然后进行简单的请求校验后发送到消息队列 RocketMQ；第二部分是写子服务提供者，该部分通过使用 web3j 服务实现与 Geth 客户端的直接交互。以上两部分的实现均基于 SpringBoot 框架，其基本架构为 MVC 三层架构的变形，即采用 Controller 对外提供接口，使用 Service 实现具体服务内容。区分服务者与消费者的原因在于业务的纵向拆分后可以加入熔断机制，保证系统不会因为某个组件的失效导致大面积崩溃。写服务提供者直接与

以太坊服务平台交互，如图 4.9 为写服务提供者的架构及封装的方法。

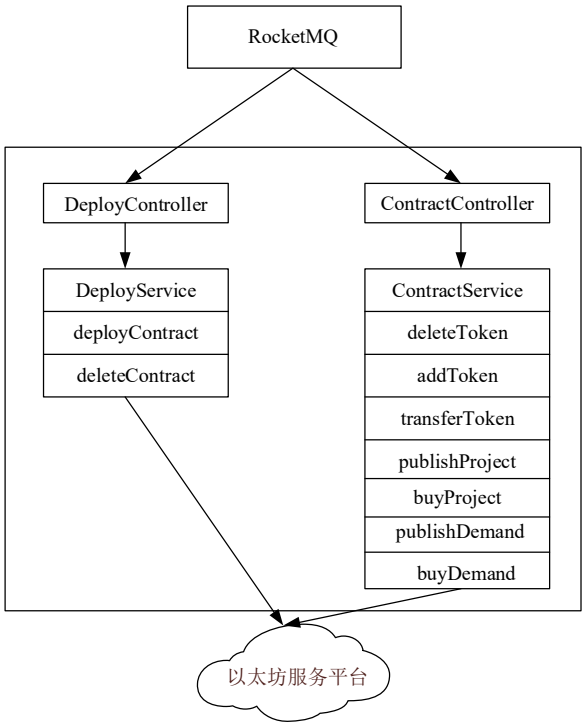

图 4.9 写服务提供者架构

在写服务提供者中，Controller 通过调用 Service 对外提供服务，Service 有两个。DeployService 用于发布和删除智能合约，ContractService 用于实现私募股权平台的核心业务如发布项目、购买项目等操作。

4.3.2.2 读子服务实现

读子服务的实现同样分为两部分：读服务消费者与读服务提供者，与写服务不同的是读服务封装的具体操作有所不同，具体实现如图 4.10 所示：

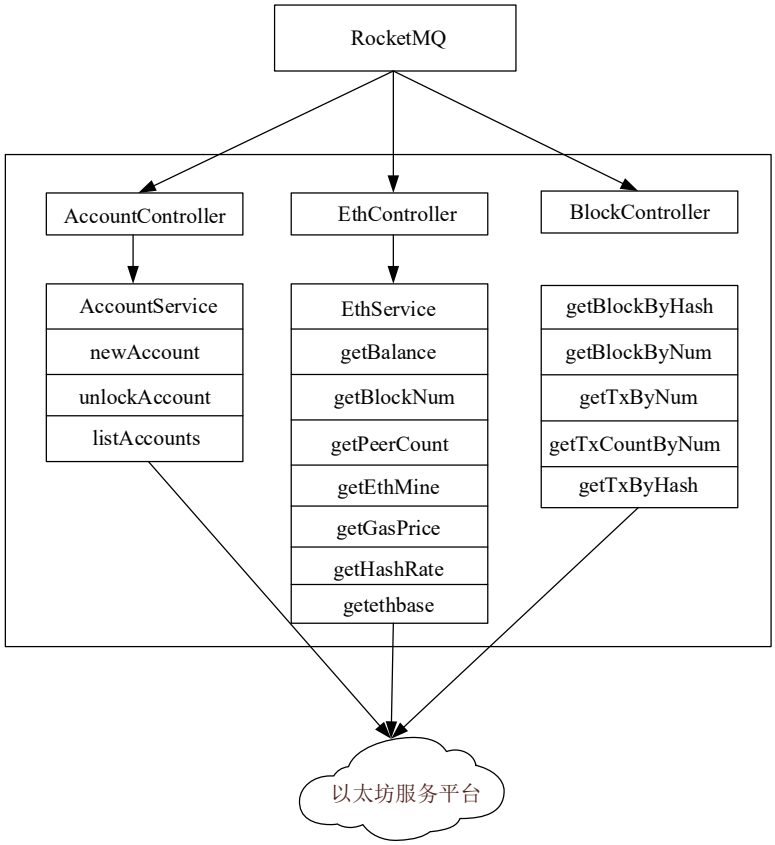

图 4.10 读服务提供者架构

读服务提供者的实现分为三个 Controller 分别调用 Service。AccountService 主要负责管理账户等操作，封装了创建账户、解锁账户和列举全部账户的操作。EthService 负责获取以太坊信息，如账户余额、区块数量、矿工数量等操作。BlockService 负责获取交易与区块的内容，封装了通过 hash 获取区块内容、通过区块号获取区块内容和获取交易信息等操作。

4.4 本章小结

本章首先介绍了私募股权交易平台的核心业务，然后考虑其对以太坊 Geth 节点的读写操作会导致其服务能力下降设计了多 Geth 客户端的以太坊服务平台，同时根据其业务特点设计了读写分离的子服务系统，并最终介绍了以上两部分中各个模块的实现。

第五章 私募股权交易平台实现与评估

前两章介绍了基于状态反馈的负载均衡策略设计与基于微服务的私募股权后台服务系统设计，基于前两章的工作，本章进一步实现私募股权交易平台，并对该交易平台的系统性能进行了评估。

5.1 系统功能设计与实现

5.1.1 项目与需求发布功能

项目和需求发布功能是私募股权交易平台的核心功能之一，项目是指用于用户浏览与购买的数字资产信息，需求是指用户发布的私募股权招募信息。项目的发布如图 5.1 所示，用户可以填写基本项目信息、上传项目文件最终实现发布。当前端发布后，后台将此项目发布作为智能合约事件记录到区块中，同时返回该项目的地址信息。

图 5.1 项目发布流程

需求的发布与项目发布有所不同，如图 5.2 所示，需求的发布需要填写的内容包括需求的描述、详细补充说明以及最重要的需求失效时间和悬赏金额。

图 5.2 需求发布流程

项目和需求的发布使得用户可以在该平台展示和出售用于交易的私募股权信息，当一部分用户发布项目项目与需求后，另一部分用户则可以浏览与购买项目信息包括回答需求。

5.1.2 项目浏览购买与需求浏览回答功能

项目的浏览与购买功能是私募股权交易的核心，其中项目的浏览需要显示的内容包括项目的摘要信息以及项目在以太坊中的地址信息。如图 5.3 为项目浏览与购买页面，用户通过查看摘要信息确定自己是否需要购买该项目的详细文件信息，然后对项目进行购买。项目的购买通过智能合约事件记录到区块中，花费的是本合约平台代币。

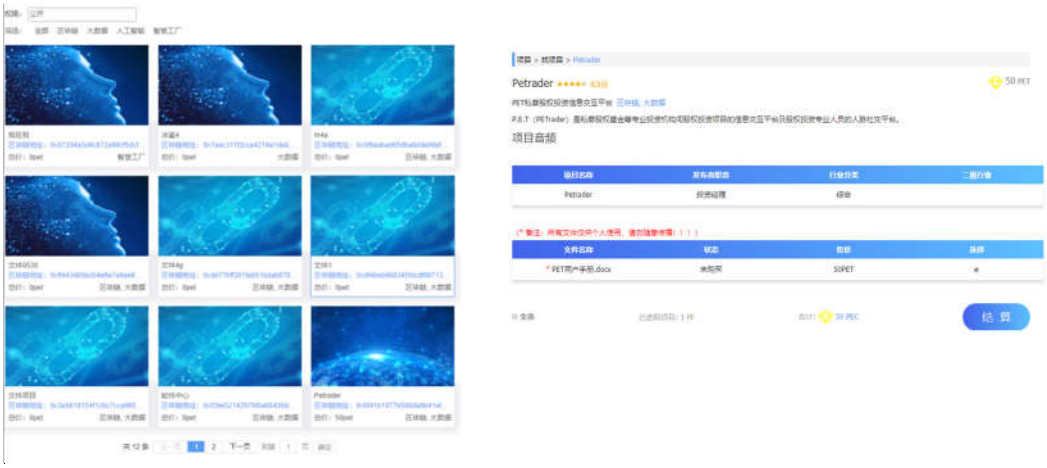

图 5.3 项目浏览与购买页面

5.1.3 账户管理功能

账户管理功能主要包括三个部分，分别是人脉管理、账户充值和账单。

图 5.4 人脉管理界面

如图 5.4 为人脉管理界面，用户可以在此搜索浏览私募股权交易平台中的其他用户并发送好友请求，当对方同意添加请求后成为好友。在项目和需求发布时可以设置相应的隐私级别，如“仅好友可见”，则该项目和需求的可见方仅为好友。

图 5.5 账户充值界面

如图 5.5 为账户充值界面，用户可以在此界面进行充值。充值即通过人民币兑换本系统通用代币 petToken。本系统所有的项目购买均通过 petToken。该充值界面实现了微信和支付宝两种支付方式。

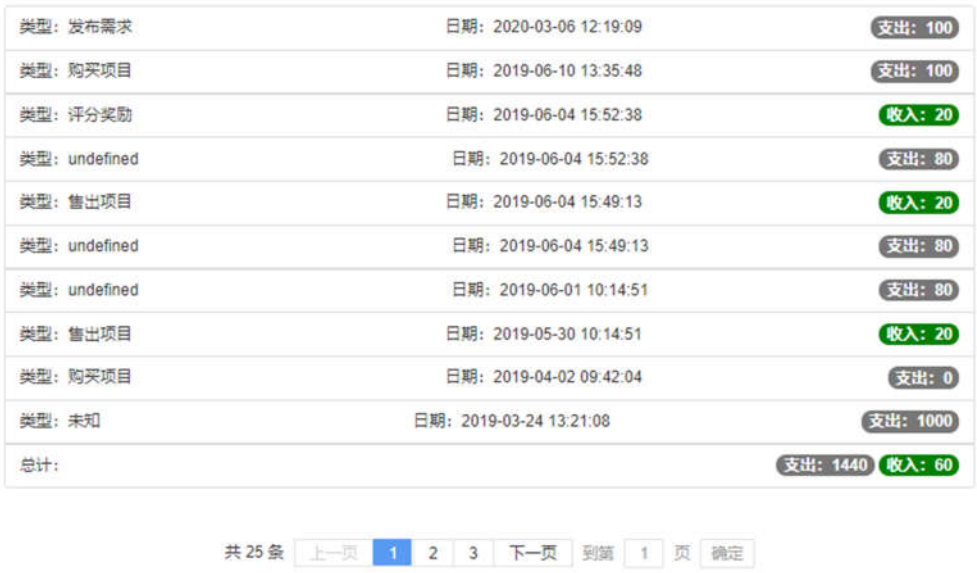

图 5.6 账户账单页面

如图 5.6 为账户账单页面，用户可以在此页面查看自己的历史账单。该账单包括充值兑换的收入、购买项目的支出、出售项目收入、招募需求的支出和回答需求的收入。

5.2 系统性能评估

5.2.1 对照架构

本文基于微服务思想构建读写分离的以太坊后台服务系统，底层基于 Docker 容器构建多 Geth 客户端的以太坊服务平台，并基于状态空间反馈设计合理的负载均衡策略将任务合理分发到对应客户端。为了验证本系统在性能上的进步，本文对照单体应用架构的以太坊服务系统做对照实验。

单体应用在具有不同的逻辑模块化架构，但应用程序被作为一个单体进行打包和部署，系统使用 Java 实现，基于 SpringBoot 框架构建并编译成 WAR 包直接运行于应用服务器 Tomcat 中。底层客户端的调用同样使用单客户端模型，因此无需负载均衡策略。

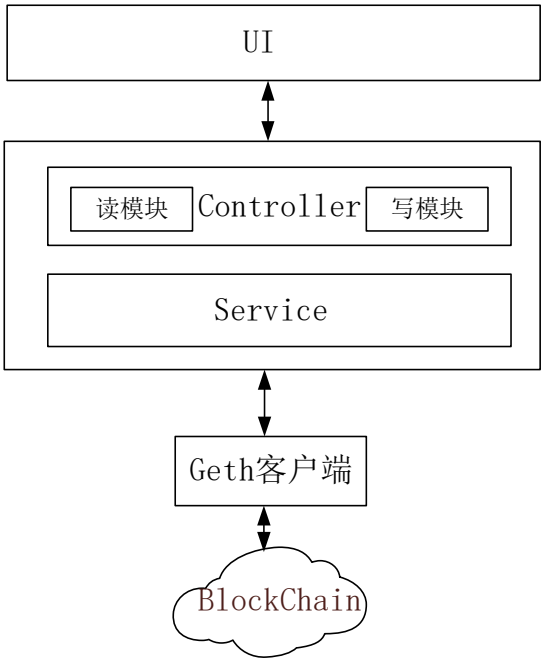

图 5.9 对照实验中的单体应用架构

如图所示为对照实验的单体应用架构，该应用内部同样区分为读和写逻辑模块，但是在实际的应用架构中是作为一个整体部署的。底层的 Geth 客户端的服务也是单个的 Geth 客户端对整个应用提供服务。

5.2.2 实验设计

目前面向私募股权平台的版权交易平台运行良好，本部分测试内容主要针对性能测试。功能测试即是否可以完成创建账户，转账，发布智能合约和挖矿等已于上一小结测试。性能测试通过与单客户端与单体区块链服务后台系统构成的区块链服务平台做并发性能对比。

本文测试环境为基于 VMware station pro 构建的虚拟机。VMware station pro 搭建于单机电脑，其配置为：

系统 Windows 10 专业版 64 位；

处理器 Intel(R) Core(TM) i7-5600U @2.6GHz；

内存 12GB；

VMware station pro 相关配置为：

版本 12.5.7 build-5813279；

系统 Ubuntu 16.04；

内存 2G；

CPU 2 核；

硬盘 20G；

容器 Docker 19.03.4；

为模拟真实环境的请求情况，本测试涉及的区块链系统操作主要分为四类，分别是连接管理、账户管理、交易管理与合约发布。四类操作选取典型操作代表做并发性测试，值得注意的是在我们无法准确的预测在真实的业务场景中各种操作的具体数量，因此本文仅将各种操作按照大致比例进行混合操作，具体比例见表 5.1。

功能分类	详细功能	所占比例
连接管理	获取某账户币值	20%
	获取当前主账户	5%
	获取本客户端所有账户	5%

	获取区块数量	10%
账户管理	创建新账户	5%
	展示当前所有账户	5%
	解锁账户	5%
交易管理	发送交易	30%
	获取交易哈希	5%
	通过交易哈希获取交易内容	5%
合约发布	发布智能合约	5%

表 5.1 并发测试数据

性能测试的维度包括两个方面，首先是并发性能，其次是资源消耗情况。并发性能通过完成固定数量的请求的时间展现，资源的消耗通过对 CPU 和内存的占用率的变化来展现。因区块链底层服务平台是多客户端架构，为了测试多客户端在提供服务方面的能力，这里将采用的测试分别采用 3 客户端、5 客户端和 8 客户端三种形式进行测试。对于并发性方面的测试，我们分别在 200、600、1000 和 1200 并发量的情况下测试响应时间(单位 s)。

5.2.3 实验结果

依照实验设计，本文通过模拟请求测试并发性，并同时检测各个容器的内存和 CPU 的占用情况。如图 5. 为并发测试结果。

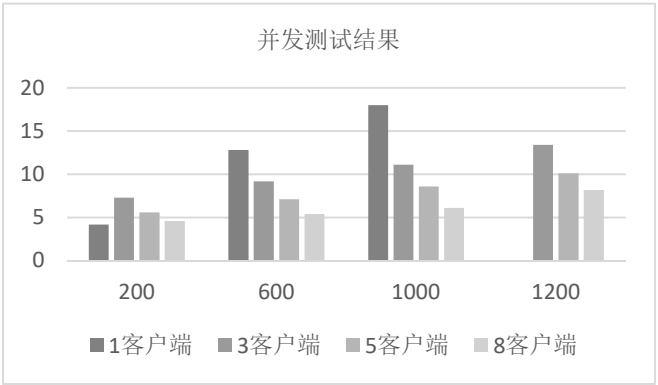

图 5.10 并发测试结果

从上图中可以看出，在 200 并发量的时候，单客户端的完成速度高于多客户

端，这是因为在多客户端之间的切换需要消耗时间。因此在低并发情况下，单客户端架构性能优于多客户端架构。在并发量到 600 及往上的时候，多客户端架构的优越性逐渐体现出来，多客户端的任务完成时间小于单客户端的任务完成时间，而且随着客户端的数量的增加，任务完成的时间变得更短。在并发量达到 1200 的时候，单客户端因为阻塞问题导致无法正常完成功能。但是多客户端架构依然在没有影响的情况下完成任务。

5.3 本章小结

本文首先介绍了本文基于反馈负载均衡策略的以太坊服务系统的主要功能，以及服务场景。随后利用 Java 后台性能测试工具，对比单体架构，评估了基于反馈负载策略的以太坊服务系统的性能。性能的评估角度从并发性压测维度出发，结果符合预期，验证了系统的性能提升以及稳定性。最后，该系统被应用于私募股权应用平台，验证了系统的实践性和应用性。

第六章 总结与展望

6.1 工作总结

随着区块链技术的发展，越来越多的区块链应用如雨后春笋般应运而生。由于区块链技术涉及相关技术点繁杂，其应用方式往往基于以太坊(Ethereum)等平台，而以太坊在应用过程中存在部署开发效率低、数据安全性保障低等问题。上海软件技术中心开发了基于以太坊的私募股权管理平台，其中以太坊服务模块存在效率低下，安全性低下的情况。且在项目开发和运维过程中存在开发效率低下，复用性较低的问题。对于以上问题，本文结合微服务思想设计和实现了以太坊服务系统，该系统提高了私募股权管理平台中区块链相关操作的效率和安全性，且提高了相关模块的运维和迁移效率。综上，本文研究内容主要包括以上部分：

首先，本文介绍了区块链应用的相关工作和国内外研究现状，总结了当前以太坊在对外提供服务时出现效率和扩展性低下的问题，无法满足在并发能力较高的情况。基于相关研究现状以及一般应用场景下的问题，本文设计和实现了一种高效稳定的以太坊服务系统。

其次，本文将以太坊客户端运行于 Docker 容器中，制作了相应的镜像，使得以太坊环境的迁移效率得到了极大的提升。本文结合微服务思想，设计和实现了以太坊服务提供者和服务调用者等子服务，并将该类子服务运行于 Docker 容器中。

然后，对运行在 Docker 中的以太坊客户端的调用，根据容器的状态设计了反馈负载均衡算法，使得系统整体的服务性能达到最高。

最后，详细充分的分析了私募股权平台应用场景下的业务特点，对与以太坊相关的业务做了横向的读写拆分并封装为相应的子服务。在该基础上结合反馈负载均衡算法设计了相应的业务调度模型。并最终实现了基于负载均衡算法的以太坊服务系统，并对其进行了功能和性能两方面的实验验证。

6.2 工作展望

本文的主要工作是研究以太坊在服务过程中出现的性能低下和数据相关操作安全性低下的问题，针对以太坊在容器中的运行状态设计了高效的负载均衡算法，并根据私募股权管理应用场景下的具体业务设计和实现了基于反馈负载均衡

策略的以太坊服务系统。本文的系统设计基于微服务思想，采用基于 Java 的 Spring Cloud Alibaba 解决方案。本文设计的相关方案以及系统实现仅适用以太坊在构建联盟链的应用场景中。为进一步提高该系统的服务效率，安全性等服务能力，可以考虑从系统架构以及负载均衡算法两个方面对系统进行优化：

微服务架构方面的改进意见：

首先，本文以太坊服务系统实现是基于 Spring Cloud Alibaba 解决方案，该方案中各子系统的实现基于 SpringBoot，而 Java 的运行需要 JVM 的支持，因此较为笨重，其开发效率与运行效率相对较低，可以考虑采用 Google 于 2009 年发布的 Golang 语言重构微服务系统中的各个子模块。Golang 是编译型语言，它更适合多处理器的分布式系统，且其协程特性使得并发性能较 Java 更好。

其次，本文微服务系统中的服务管理采用的方式是 Docker Compose，它在容器编排方面的灵活性不够。可以考虑引入 Kubernetes 提供容器的部署、编排和维护。Kubernetes 会自动去新建、监控和重启服务，管理员可以加载一个微服务，让规划器来找到合适的位置，同时，Kubernetes 也系统提升工具以及人性化方面，让用户能够方便的部署自己的应用。

最后，云计算时代已经到来，可以考虑将该服务系统进行上云处理，以实现真正的 BaaS(BlockChain As A Service)。

负载均衡算法优化：

本文主要考虑私募股权管理业务所面临的业务场景，因此，本负载均衡算法考虑容器的状态较少，换言之，本文所建立的容器状态向量空间的维度较低。因此，可以在深入研究以太坊客户端的运行特性和 Docker 容器的特性后，针对不同的应用场景设计更加高效的反馈负载均衡算法。随着云计算时代的到来，各种高效的负载均衡算法蓬勃发展，本文所设计的反馈负载均衡算法也应该与时俱进，且该领域潜力巨大，一定可以在本文所设计架构下提高以太坊服务能力。

参考文献

[1] Nadauld T D, Sensoy B A, Vorkink K, et al. The liquidity cost of private equity investments: Evidence from secondary market transactions[J]. Journal of Financial Economics, 2019, 132(3): 158-181.

[2] Antoni M, Maug E, Obernberger S. Private equity and human capital risk[J]. Journal of Financial Economics, 2019.

[3] Bernstein S, Lerner J, Sorensen M, et al. Private equity and industry performance[J]. Management Science, 2017, 63(4): 1198-1213.

[4] Faccio M, HSU H C. Politically connected private equity and employment[J]. The Journal of Finance, 2017, 72(2): 539-574.

[5] Ang A, Chen B, Goetzmann W N, et al. Estimating private equity returns from limited partner cash flows[J]. The Journal of Finance, 2018, 73(4): 1751-1783.

[6] Van Alstyne M W, Parker G G, Choudary S P. Pipelines, platforms, and the new rules of strategy[J]. Harvard business review, 2016, 94(4): 54-62.

[7] Bansraj D, Smit H T J, Volosovych V. Can Private Equity Funds Act as Strategic Buyers? Evidence from Buy-and-Build Strategies[R]. Working paper, www. ssrn. com, 2019.

[8] Nakamoto S. Bitcoin: a peer-to-peer electronic cash system, October 2008[J]. Cited on, 2019: 53.

[9] Zheng Z, Xie S, Dai H N, et al. Blockchain challenges and opportunities: A survey[J]. International Journal of Web and Grid Services, 2018, 14(4): 352-375.

[10] Cao C, Yan J, Li M X. How to Understand the Role of Trusted Third Party in the Process of Establishing Trust for E-Commerce?[J]. 2019.

[11] Sherman A T, Javani F, Zhang H, et al. On the origins and variations of blockchain technologies[J]. IEEE Security & Privacy, 2019, 17(1): 72-77.

[12] 魏生, 戴科冕. 基于区块链技术的私募股权众筹平台变革及展望[J]. 广东工业大学学报, 2019, 36(02): 37-46.

[13] 安立. 区块链在私募股权交易领域的应用[J]. 上海金融学院学报, 2017 (2017 年 02): 47-51.

[14] 邵奇峰, 金澈清, 张召, 等. 区块链技术: 架构及进展[J]. 计算机学报, 2018, 41(5): 969-988.

[15] 郑福生, 陈芳, 陈彦宇, 等. 一种基于 SOA 的电力通信网健康度评估系统设计[J]. 电力信息与通信技术, 2017, 15(7): 25-30.

[16] Avila K, Sanmartin P, Jabba D, et al. Applications Based on Service-Oriented Architecture

(SOA) in the Field of Home Healthcare[J]. Sensors, 2017, 17(8): 1703.

[17] Buterin V. A next-generation smart contract and decentralized application platform. White Paper, 2014.

[18] Namiot D, Sneps-Sneppe M. On micro-services architecture[J]. International Journal of Open Information Technologies, 2014, 2(9): 24-27.

[19] Dragoni N, Giallorenzo S, Lafuente A L, et al. Microservices: yesterday, today, and tomorrow[M]//Present and ulterior software engineering. Springer, Cham, 2017: 195-216.

[20] Lawson J, Wolthius J. System and method for providing a micro-services communication platform: U.S. Patent 9,363,301[P]. 2016-6-7.

[21] Balalaie A, Heydarnoori A, Jamshidi P. Microservices architecture enables devops: Migration to a cloud-native architecture[J]. Ieee Software, 2016, 33(3): 42-52.

[22] Stubbs J, Moreira W, Dooley R. Distributed systems of microservices using docker and serfnode[C]//2015 7th International Workshop on Science Gateways. IEEE, 2015: 34-39.

[23] Amaral M, Polo J, Carrera D, et al. Performance evaluation of microservices architectures using containers[C]//2015 IEEE 14th International Symposium on Network Computing and Applications. IEEE, 2015: 27-34.

[24] Bui T. Analysis of docker security[J]. arXiv preprint arXiv:1501.02967, 2015.

[25] Anderson C. Docker [software engineering][J]. IEEE Software, 2015, 32(3): 102-c3.

[26] Peng H, Han W, Yao J, et al. The Realization of Load Balancing Algorithm in Cloud Computing[C]//Proceedings of the 2nd International Conference on Computer Science and Application Engineering. ACM, 2018: 140.

[27] Revah Y, Melman D, Mizrahi T, et al. Method and apparatus for load balancing in network switches: U.S. Patent 9,876,719[P]. 2018-1-23.

[28] Sharma M, Kini S, Tuli S A, et al. Adaptive load balancing for single active redundancy using EVPN designated forwarder election: U.S. Patent Application 10/050,809[P]. 2018-8-14.

[29] Arumugam M, Verzunov S, Kamath S, et al. Auto discovery and configuration of services in a load balancing appliance: U.S. Patent Application 10/101,981[P]. 2018-10-16.

[30] Kansal N J, Chana I. An empirical evaluation of energy-aware load balancing technique for cloud data center[J]. Cluster Computing, 2018, 21(2): 1311-1329.

[31] Soo W K, Ling T C, Maw A H, et al. Survey on load-balancing methods in 802.11 infrastructure mode wireless networks for improving quality of service[J]. ACM Computing Surveys (CSUR), 2018, 51(2): 34.

[32] Nofer M, Gomber P, Hinz O, et al. Blockchain[J]. Business & Information Systems Engineering, 2017, 59(3): 183-187.

[33] Crosby M, Pattanayak P, Verma S, et al. Blockchain technology: Beyond bitcoin[J]. Applied

Innovation, 2016, 2(6-10): 71.

[34] Mell P, Dray J, Shook J. Smart Contract Federated Identity Management without Third Party Authentication Services[J]. arXiv preprint arXiv:1906.11057, 2019.

[35] Dannen C. Introducing Ethereum and Solidity[M]. Berkeley: Apress, 2017.

[36] Leidner J L, Nugent T, Chadwick S. Systems and methods for smart contract intervention: U.S. Patent Application 16/133,932[P]. 2019-1-17.

[37] Mahajan A. A DApp for e-voting using Blockchain-enabled Smart Contracts[D]. , 2019.

[38] Cito J, Ferme V, Gall H C. Using docker containers to improve reproducibility in software and web engineering research[C]//International Conference on Web Engineering. Springer, Cham, 2016: 609-612.

[39] Rad B B, Bhatti H J, Ahmadi M. An introduction to docker and analysis of its performance[J]. International Journal of Computer Science and Network Security (IJCSNS), 2017, 17(3): 228.

[40] Chung M T, Quang-Hung N, Nguyen M T, et al. Using docker in high performance computing applications[C]//2016 IEEE Sixth International Conference on Communications and Electronics (ICCE). IEEE, 2016: 52-57.

[41] Leyi G O U, Qing C, Liang J, et al. Technical Research and Application Analysis of Microservice Architecture[J]. DEStech Transactions on Computer Science and Engineering, 2019 (iccis).

[42] Sharma S. Mastering Microservices with Java: Build Enterprise Microservices with Spring Boot 2.0, Spring Cloud, and Angular[M]. Packt Publishing Ltd, 2019.

发表论文和科研情况说明

攻读硕士期间发表的学术论文

[1] 赵晓峰，张绍华，卢暾，戴炳荣，李超. 面向私募股权的区块链服务系统设计，计算机应用与软件(全国中文核心期刊，已录用).

攻读硕士期间申请的发明专利

[1] 卢暾，赵晓峰，张绍华，顾宁，戴炳荣，李超. 基于以太坊的高性能区块链服务系统，专利号：2019112702245480

攻读硕士期间参与的科研项目

[1] 上海区块链产业研究与评估验证平台研发应用　编号 18DZ1112100，上海市 2018 年度科技创新行动计划

致 谢

时光荏苒，岁月如梭，三年的研究生生涯转瞬即逝。回首过去的三年，感慨良多，收获也很丰厚，最幸运的是遇到一群优秀的良师益友，将这段经历书写成了最难忘的人生篇章。依然记得，2017 年在研究生复试面试上顾老师和蔼的询问是否愿意加入信息协同与计算实验室的大家庭，卢老师鼓励地表示“应该早点与我联系”。怀着对复旦研究生生活的向往和对老师们的知遇之恩的感激，我在暑假便开始了实验室的生活。实验室为我们配备了充足的设备，提供了舒适的工作和学习环境以及老师的指导和同学们的同窗陪伴，无论多久，都会是记忆里最美好的印记。

首先，我要感谢卢暾副教授，卢暾老师是我的导师。卢老师从研一开始便为我提供了阅读论文等学习技巧方面的指导，还在研二期间我的亲人因病住院期间为其捐款和鼓励我。卢暾老师对事业的严谨的态度和对学生温暖的关怀是将永远指引我前进的明灯。

其次，我要感谢顾宁教授对我的指导和帮助，所谓“高山仰止，景行行止”，顾宁老师是信息协同领域首屈一指的专家，其严谨的工作态度和求真务实的科研态度深深的影响着我。

我要感谢丁向华副教授和张鹏老师，丁老师在学术工作中严谨认真，求真务实，给我带来了深刻的影响。张鹏老师在我 2017 年入学时是实验室的在读博士生，在其博士毕业后接着在实验室开展科研工作，在论文选题和完成论文期间给予了我非常大的帮助。

特别地，我要感谢张绍华老师以及上海软件技术中心的各位领导。在研二期间，我在上海软件中心开展区块链相关科研工作，张绍华老师给予了生活上和工作上非常大的帮助。在毕业论文的选题和写作过程中，张老师给出了非常多的真知灼见，对我的整体工作起到了非常重要的指导作用。

最后，我要感谢实验室的同学们。在过去的三年中，无论是科研、技术和生活方面，你们的陪伴和鼓励都是我保持对生活的热情的最原始的动力。

无论人生的路通往何方，你们对我的帮助都是我最珍贵的宝藏，感谢你们！

复旦大学
学位论文独创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。论文中除特别标注的内容外，不包含任何其他个人或机构已经发表或撰写过的研究成果。对本研究做出重要贡献的个人和集体，均已在论文中作了明确的声明并表示了谢意。本声明的法律结果由本人承担。

作者签名：_____ 日期：_____

复旦大学
学位论文使用授权声明

本人完全了解复旦大学有关收藏和利用博士、硕士学位论文的规定，即：学校有权收藏、使用并向国家有关部门或机构送交论文的印刷本和电子版本；允许论文被查阅和借阅；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其它复制手段保存论文。涉密学位论文在解密后遵守此规定。

作者签名：_____ 导师签名：_____ 日期：_____