树状区块链性能测试与基于 Rust 的重写

摘　要

2008 年，区块链技术横空出世，并迅速以其去中心化、不可篡改等特点，迅速博得了大量行业的青睐。目前，区块链技术已经在数字货币、车联网等领域取得了亮眼的成绩。然而，传统区块链采用单链结构，每个区块仅存储上一个区块的哈希信息，在需要应对大吞吐量的工况下，容易因链条过长导致性能下降；此外，在应用于车联网这一场景下时，由于区块并未按照地理位置存储，而车辆节点需要关心的信息大多来自临近区域的区块数据，故可能需要耗费诸多不必要的查询开销。

为解决上述两个问题，“树状区块链”应运而生。在树状区块链中，区块被分为创世块、分支区块和叶子区块三种。叶子区块和传统的区块链并无太大差异；分支区块则负责将数个叶子区块组织起来，按照叶子区块所代表的地理位置，结合 GeoHash 编码技术形成多叉树树结构；创世块和分支区块类似，但它没有父链指针。由于树状结构相比单链结构的深度更小，且采用了与地理位置相关的 GeoHash 进行分支构造，故树状区块链有望为上述两个问题提供合理的解决方案。

本课题首先研究树状区块链相较传统单链结构区块链的优势与局限性；其次，对实验室已有工作——基于区块链的出租车调度系统进行复现，验证其可用性；对于树状区块链引入的特色功能——跨子链转账，设计系统的性能测试，并建立简单的数学模型，供开发者在不同应用场景下选择树状区块链和传统单链结构区块链参考。在上述工作的基础上，以出租车调度系统为背景，测试树状区块链和传统链式区块链在运行该调度系统时的性能表现差异。最后，提出一种使用 Rust 编程语言重写树状区块链的可行方案，讨论将现有树状区块链的开发平台由以太坊开发平台迁移至更加优秀 Substrate 开发框架的优势及可行性，并进行部分树状区块链的功能特性的重写工作加以佐证。

关键词：区块链; 车联网; 测试

The Testing of Tree-Like Blockchain and Rust-Based Rewriting

Abstract

In 2008, the blockchain technology came into the world, and quickly with its decentralized, tamper-free characteristics, quickly won the favor of a large number of industries. At present, blockchain technology has made remarkable achievements in digital currency, Internet of vehicles and other fields. However, the traditional blockchain adopts the single-chain structure, and each block only stores the hash information of the previous block. Under the working condition of large throughput, the performance is likely to be degraded due to the long chain. In addition, when applied to the scenario of the Internet of vehicles, because the block is not stored according to the geographical location, and most of the information that the vehicle node needs to care about is from the block data in the adjacent area, it may consume a lot of unnecessary query costs.

To solve the above two problems, ”tree-like blockchain” came into being. In a tree-like blockchain, blocks are divided into genesis blocks, branch blocks and leaf blocks. The leaf block is not very different from the traditional blockchain, while the branch block is responsible for organizing several leaf blocks, according to the geographical location of the leaf block, combined with GeoHash coding technology to form a tree structure similar to dictionary tree. The major difference between the branch block and the genesis block is that a genesis block does not have the so-called ”parent block pointer”. Due to the smaller depth of the tree structure compared to the single chain structure and the use of GeoHash for branch construction, it is expected to provide a reasonable solution to the above two problems.

This thesis first studies the advantages and limitations of tree-like blockchain compared to traditional single-chain structured blockchain; secondly, it reproduces the existing work in the laboratory - a taxi dispatch system based on blockchain, to verify its usability; for the special function introduced by the tree-like blockchain - cross-sub-chain transfer, design system performance testing, and establish a simple mathematical model to provide suggestions for developers to choose between tree-like blockchain and traditional

single-chain structured blockchain in different application scenarios. On the basis of the above work, taking the taxi dispatch system as the background, test the performance difference between tree-like blockchain and traditional chain-like blockchain when running this dispatch system. Finally, propose a feasible plan to rewrite tree-like blockchain using Rust programming language, discuss the advantages and feasibility of migrating the existing tree-like blockchain development platform from Ethereum development platform to a better Substrate development framework, and partially rewrite some functional features of tree-like blockchain to provide evidence.

Key Words: Blockchain; Internet of Vehicle; Testing

目 录

第 1 章 绪论

1.1 研究背景

区块链是一种分布式的共享账本，允许数个参与方一同共享数据[1]。区块链技术所拥有的去中心化、透明性和安全性等优势，令这一新兴的概念迅速为各行各业接受：中国人民银行数字货币研究所正在积极探索区块链技术在低并发、低敏感的资产确权、交易转让、账本核对等场景下的应用[2]；区块链透明化的特点和极高的安全性也引起了地产行业的注意[3]；物联网、车联网等领域也在积极拥抱区块链[4]。可以预见，区块链技术在未来将吸引更多行业加入，以其去中心化、不可篡改等特性造福人类社会。

树状区块链，是实验室正在开发并已趋于完善的改良型区块链[5]。其基本思想大致为：将区块分为创世块、分支区块和叶子区块三种；结合 GeoHash 编码技术，不再采用传统区块链的单链结构，形成类似于字典树的树状结构；同时，为了满足快速查询的需要，在区块中增添了一些辅助数据结构。经过以上改良，树状区块链可以在对地理位置敏感、且网络结构变化较频繁的应用场景中，发挥相较传统区块链更好的理论性能。

车联网技术（Internet of Vehicle）属于物联网技术的范畴，其思想乃是在车辆上搭载接入网络的设备，旨在实现不同车辆之间的相互通信；不仅如此，车联网技术也容许车辆与行人、路边基站等交通参与方和交通基础设施通信，实现实时的车况检测、路况查询与收集等功能，对于提升车主用车体验、乘客出行体验有强大的潜力[6]。2022 年 12 月 8 日，公安部发布的数据显示全国机动车保有量到达 4.15 亿辆，机动车驾驶员人数超过 5 亿位！随着如何安全有效地管理如此庞大的保有量带来的海量数据这一巨大挑战变得日益严峻，人们纷纷将目光转移到了区块链上[7][8]。然而，传统区块链在处理车联网场景下的具体事务时，往往存在诸如此类的一些弊端：

- 车辆作为区块链网络的参与者（节点）时，其地理位置可能发生很大变化，致使网络结构需要频繁更新；
- 传统区块链采用单链结构，在区块链上执行查询的时间复杂度较高；
- 在车联网系统中，车辆应该关心的信息大部分来自于其所在位置的临近街区，

而传统区块链并不以地理位置索引区块及其交易，故一次查询可能会获得较多无用信息等。

由于树状结构相比单链结构的深度更小，且采用了与地理位置相关的 GeoHash 编码[9]进行分支构造，故有效降低了查询开销，令基于位置的信息查询能够更加“有的放矢”，有望为运用区块链技术处理车联网问题提供合理可行的解决方案。

目前，实验室已有的树状区块链采用以太坊（Ethereum）实现。以太坊是一个开源的区块链计算平台，允许开发者进行去中心化应用程序（DApp）的开发和部署[10]。其支持以 Solidity 编程语言编写运行在在以太坊虚拟机上的脚本程序（智能合约 Smart Contract），极大拓宽了区块链的功能适用性。

Substrate 是一套开源、模块化的区块链开发框架[11]，允许开发者自由地使用官方预定义的各种组件，或自行实现的组件构建个性化的区块链，并于其上使用基于 Rust 的 ink! 编程语言进行智能合约开发。与以太坊比较，Substrate 具有包括而不仅限于如下优势：

- 支持模块化设计，程序员可以轻松增删模块，构建更加贴近实际需求的区块链
- 采用 Rust 编程语言作为其底层实现，速度更快，效率更高
- 支持编译为大多数现代浏览器支持的 WebAssembly 二进制，提供了更好的跨平台兼容性
- 为多链结构提供了较好的支持

本文将在车联网的应用场景下，以实验室已有工作——出租车调度系统为例，探究树状区块链在不同工况下的性能表现，验证其拥有相较传统单链区块链更好的性能表现。本文还将积极尝试为将树状区块链从以太坊平台转向更优秀的 Substrate 平台。鉴于毕设时间之限制，本文仅讨论区域索引区块链的部分特性的迁移工作，此举旨在验证两平台在功能上的相似性，进而确保未来树状区块链的底层功能迁移工作的可行性。

1.2 相关技术调研

1.2.1 区块链技术概述

2008 年，一位自称为中本聪的人发布了名为《Bitcoin: A Peer-to-Peer Electronic Cash System》的论文[12]，宣告了区块链技术的诞生。区块链，乃是一个分布式的账本；区块链网络不存在所谓的“中心服务器”，每台参与构成区块链网络的计算机（又被称为“节点”）均持有一份该账本的副本。每一笔交易，都将记录在名为“区块”的数据结构中；随着区块的不断产生，它们将形成一条单向链状结构，且区块上存储的数据将不可再被修改。通过称为共识算法的机制，各节点能够就区块链的当前状态达成一致，并在链上数据发生变化时及时追踪并更新到自身存储的账本中[13]。不仅如此，若某个节点尝试擅自修改自身所持有的账本，其行为会被共识算法拒绝，从而规避了恶意篡改链上数据的风险[14]。上述区块链的优势，令区块链这一新兴的概念迅速吸引了各行各业的眼球。可以预见，区块链技术在未来将吸引更多行业加入，以其去中心化、不可篡改等特性造福人类社会。

1.2.2 智能合约概述

智能合约（Smart Contract）这一概念由 Nick Szabo 于 1994 年提出。在比特币的支付模型中，仅存在一个简单的堆栈计算机。由于其可用的操作方法并非图灵完备，只能执行比较简单的操作，从而限制了区块链的应用场景[15]。智能合约的出现打破了这一局面。在 Szabo 于 1996 年撰写的《Smart Contracts: Building Blocks for Digital Markets》一文[16]中，Nick 设想智能合约就是运行在区块链上的一段程序，当满足某种条件时，相应代码将自动被执行，而该过程人类无需也无法介入。智能合约一定程度上避免了交易双方抵赖的问题，并且其图灵完备的特性也令区块链技术在不同应用场景下的适应性大大增加了。

1.2.3 区域索引区块链和树状区块链概述

周畅设计的区域索引区块链，实现了区块链地理信息索引方法，能够依据地理位置信息快速查询特定位置的交易[5]。相较以太坊官方实现的传统区块链而言，区域索引区块链在区块头中加入了区域状态树的树根哈希，以支持基于位置的快速信息查询；同时，为追踪每个账户的包括位置信息的完整状态，在账户状态数据结构中还加入了当前的地理位置字段、和账户位置树这一数据结构。不仅如此，在记录交

易、收据时，均会记录发起动作的地理位置信息，并以此更新发起人的账户的当前位置及账户位置树。文献[5]中指出，采用 3 到 6 位 GeoHash 编码时，区域索引区块链相较传统的无索引区块链，执行相同查询的耗时仅为后者的 5.3%。

区域索引区块链仍然保留了传统区块链的单链结构，当区块数量很大时，其查询效率仍然会随之降低。因此，周畅进一步提出并设计了基于区域索引区块链改良的树状区域索引区块链（下简称树状区块链），如图1-1。树状区块链按照 Geohash 编码的前缀，表达父链和子链的关系，并划分区域子链。树状区块链的区块也分为三种：叶子区块、分支区块和创世块。叶子区块和传统的区块链并无太大差异，叶子区块及其后续区块均采用传统的单链结构加以组织；分支区块则负责将数个叶子区块组织起来，按照叶子区块所代表的地理位置，创世块和分支区块十分相似，但它没有父链指针。上述树状区块链的设计，进一步缩短了区块链网络中的单链长度，进而提升了查询效率。

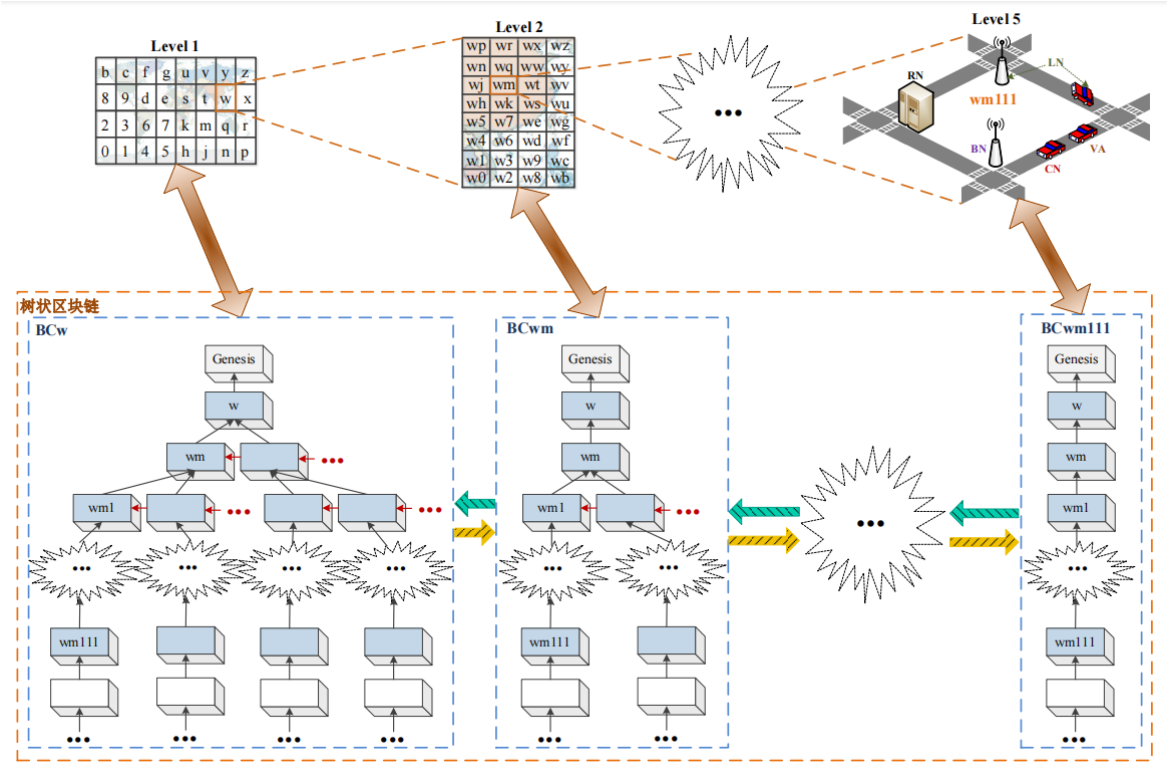

图 1-1 树状区块链示意图

1.2.4 以太坊和 Substrate

中本聪提出的比特币区块链系统支持简单的基于栈的编程语言，由于其并非图灵完备（例如不支持循环等结构），程序员仅能在其上进行受限的操作，这很大程度上限制了区块链技术的应用场景。以太坊[10]技术的诞生，旨在提出一种建立分布式应用程序（又名去中心化应用程序，DApps）的新方法，其提供名为以太坊虚拟机[17]，采用图灵完备的编程语言，令程序员得以编写更复杂更强大的智能合约，进而根据实际需求灵活定制状态转换函数等功属性。

以太坊技术的成熟引领人们从 Web 2.0 时代[18]走进了 Web 3.0 时代[19]。然而，随着以太坊在各行各业使用愈加广泛，其局限性亦逐渐显露。简而言之，以太坊为保证其功能全面性，其官方实现非常复杂且可扩展性不足，导致程序体积巨大、运行效率低下、且核心定制变得尤为繁琐。Substrate 提出了一种模块化的区块链构建框架，允许程序员自行选用充分必要的模块构建更贴近实际需求，拥有更少多余功能的区块链。此外，Substrate 选用 Rust 编程语言[20]作为其底层实现，令区块链在 WebAssembly 上运行成为可能。由于现代浏览器普遍支持 WebAssembly，这意味着区块链的运行环境终于从以太坊虚拟机（EVM）中解放出来，可以在浏览器中直接运行，不仅程序运行速度更快，且跨平台兼容性大大提升。

1.3 本文研究内容及贡献

本文将在车联网这一应用场景下，首先基于区域索引区块链，复现实验室已有工作——出租车调度系统的有关工作，证明该系统的可用性。此后，设计并进行树状区域索引区块链在单父链双子链的网络结构下的跨链转账实验，探究树状区块链在不同转账请求压力下的性能表现，验证其功能可用性，并测试跨链操作带来的额外时间开销。作为测试实验的收尾，本文将设计实验，令出租车调度系统分别在网络结构不同的树状区块链上运行，收集并可视化实验数据，评估树状区块链在不同工况的实际应用场景下的性能表现。本文还将对树状区块链从以太坊平台转向更优秀的 Substrate 平台做出积极探索。鉴于毕设时间之限制，本文仅讨论区域索引区块链的部分功能特性的迁移工作，此举旨在验证两平台在功能上相似，进而证明未来树状区块链的底层功能迁移工作的可行性。

本文结构如下。第二章将基于区域索引区块链，复现实验室已有工作——出租车调度系统，证明该系统的可用性，记录步骤操作，汇总复现过程中遇到的问题，并

尝试给出解决方案。第三章将设计并进行树状区域索引区块链在单父链双子链的网络结构下的跨链转账实验，记录其对 10 个、20 个、40 个、80 个、120 个、160 个、200 个账号执行子链之间的跨链转账的耗时、吞吐量等数据，并进行可视化，为开发者在树状区块链和单链结构区块链之间的选择提供一些建议。第四章以第二章中复现的基于区块链的出租车调度系统为背景，探究拥有四条叶子区块链的树状区块链网络在运行该系统时的性能表现，并与区域索引区块链运行相同系统的测试数据进行比较，将数据可视化，验证树状区块链在地理位置敏感型应用场景中的性能优势。第五章中，本文提出了使用 Rust 编程语言，基于 Substrate 框架重写树状区块链的改进方法，定位并分析了区域索引区块链、树状区块链与传统区块链之间的不同，并以此为基础，探索区域索引区块链的部分功能特性从以太坊向 Substrate 的迁移的可行性，最后为 Substrate 官方实现引入账户地理位置这一树状区块链的功能特性，验证了该重写方案的可行性与合理性。

第 2 章 基于区域索引区块链的出租车调度系统复现

2.1 基于区块链的出租车调度系统简介

基于区域索引区块链的出租车调度系统，以董斌的区块链地图存储作为基础，由成佳壮首先提出并实现路径规划和司乘匹配等算法，并随后经过万琦玲完善，并初步形成了复现手册；它是一套采用实验室已有成果——区域索引区块链作为服务器后端、Vue 2.JS 作为前端的出租车调度系统。在万琦玲设计的前端页面上，用户可以通过实时地图信息，直观地获得自己的位置，以及附近在线的乘客、司机等信息。用户角色分为乘客和司机两大类，前者可以进行提交乘车请求、确认上车和付款等操作，而后者可以选择是否接受派来的订单，确认到达乘客上车地点和确认到达乘客下车地点等操作。

本文将基于上述实验室已有工作，进行该基于区域索引区块链的出租车调度系统的复现工作，并详细讲解在复现过程中遇到的问题，及其解决方案。此外，该部分工作还补充了初版复现手册的缺漏，修正了其中的错误，重构了已有的脚本代码，提升了可扩展性和鲁棒性，形成了新版的复现手册，方便后人参考[1]。

2.2 环境配置

本章的复现工作将在表 2-1 所示的环境中进行。

表 2-1 复现环境

中央处理器	Intel Core i5-12500H
图形处理器	Intel Iris Xe 80EU
内存	24GB
操作系统	Ubuntu 22.04.1 LTS
虚拟机	VMWare Workstation Pro 17

将 Ubuntu 虚拟机环境配置妥当之后，还需安装 node.js、npm、web3.js 等 JavaScript 库。同时，将区域索引区块链的二进制可执行文件（代码仓库中的 geth1 二进制可执

[1] https://github.com/Endericedragon/ReproducingBlockchain

行文件）存放到/usr/local/bin 文件夹下备用。

2.3 复现步骤

详细的复现过程已记录于代码仓库的《8 重做调度系统复现实验.md》文档中，本文将只进行简单介绍。需要指明的是，原复现手册中存在许多前置实验，本文略过了这些前置实验，仅详细介绍最后有关调度系统复现的实验。

2.3.1 建立区域索引区块链

启动终端，切换至创世配置文件 genesis.json 所在的目录，随后键入如下指令并执行：

代码 2.1: 初始化区块链

```
geth1 --identity "MyEth" --rpc --rpcaddr 127.0.0.1  --rpcport "8545"
   --rpccorsdomain "*" --datadir gethdata --port "30303" --nodiscover
   --rpcapi "eth,net,personal,web3,admin" --networkid 91036 init
   genesis.json
```

对于其中的关键参数选项解释如下：

- rpcaddr、rpcport：RPC 端口的地址及端口号。外部程序可以使用该端口接入区块链，进而借助 JSON-RPC API 或者 web3.js 库和区块链进行交互。
- datadir：该选项指定链上数据在本地永久存储的位置。
- rpcapi：使用 RPC 端口与链交互时，仅能使用该选项指定的数个功能。本文将使用 eth，net，personal，web3，admin 这 5 个功能模块，它们涵盖了挖矿、发现节点、账号管理等功能，足以满足复现实验要求。
- init genesis.json：指定使用名为 genesis.json 的创世配置文件进行初始化。

执行结束后，再执行以下命令，即可启动该区块链：

代码 2.2: 启动区块链

```
geth1 --datadir ./gethdata --networkid 91036 --port 30303 --rpc --rpcaddr 127.0.0.1 --rpcport 8545 --rpcapi 'personal,net,eth,web3,admin' --rpccorsdomain='*' --ws --wsaddr='localhost' --wsport 8546 --wsorigins='*' --wsapi 'personal,net,eth,web3,admin' --nodiscover --allow-insecure-unlock --dev.period 1 --syncmode='full' console
```

注意其中的 syncmode 参数选项，其值为'full'。此时，该区块链将仅使用区域索引方法进行加速，而不会使用任何树状区块链的功能特性。

启动区块链后，终端中将出现 JavaScript 控制台，可以使用 JavaScript 编程语言的一个子集和链进行交互[21]。此时可以创建测试账户，充当司机和乘客；并且，每次启动区块链，都需要解锁这些账户，否则将无法进行智能合约部署等工作。本文创建了 8 个测试账户，分别扮演司乘角色，进行后续的实验。

创建账号后，还需前往创世配置文件中，为创建的账号赋予初始余额。赋予余额后，为强制 geth1 重新加载创世配置文件，需要手动删除存储链上数据的 gethdata 目录中的 geth 目录，并随后重新初始化并启动区块链。此时，geth1 将从创世块配置信息中加载账户余额，确保后续实验步骤可以正常开展。至此，区域索引区块链已经建立完成。

2.3.2 部署合约

实验室已有成果——出租车调度系统由两份智能合约构成，其文件名和其功能如下表所示：

表 2-2 智能合约功能概述

文件名	功能
StoreMap.sol	存储详细的地图数据，提供数据查询的结构及 A-Star 寻路等算法的实现
StoreTraffic.sol	司机和乘客的信息管理，提供司乘位置的改查、基于 Geohash 的距离计算等服务

将两份合约部署到区块链上，即完成树状区块链的部署工作。首先，使用

Remix Desktop 编译合约，记录编译结果中的 ABI 字段并进行进行压缩转义，同时记录 bytecode（下称字节码）字段。然后，将二者复制入附录 B 中的代码模板（其中的标识符 Contract 和 contractInstance 可以任意修改），将两份编辑好的模板先后复制入正在运行 geth1 的 JavaScript 命令行后并点按回车后，合约部署的请求就已经提交至交易池。开始挖矿并密切观察控制台输出，直至观察到形似以下格式的输出。此即为两份合约各自的合约地址，其顺序与提交合约部署申请的顺序一致：

```
null [object Object]
Contract mined! Address: 0x23b98f92ceac005e570b6768da377b3abd11012e
[object Object]
null [object Object]
Contract mined! Address: 0xfa6b8f0b92b323c28557faf69da028e33856f6ca
[object Object]
```

笔者的部署顺序是：先部署 StoreMap，再部署 StoreTraffic。因此，地图存储合约的地址即为 0x23b98f92ceac005e570b6768da377b3abd11012e，而调度合约地址为 0xfa6b8f0b92b323c28557faf69da028e33856f6ca。

2.3.3 上传地图数据

成佳壮完成了上传地图所需的 JavaScript 脚本。使用时，首先需要前往该脚本源代码（对应代码仓库中的`uploadmap_cjz_3.js`），修改其中关于 StoreMap 合约地址的字符串变量的值、执行数据提交的账号的公钥地址，以及要上传的地图文件。令区块链开始挖矿后，运行该脚本，将看到终端不断输出信息，直至输出“地图数据上传完成”字样，结束挖矿。此时，所有地图数据就已全部上传到区块链上了。

2.3.4 配置前端并进行可用性测试

出租车调度系统的前端由 Vue 2 写成，在启动之前尚需一些配置方可正常运行。首先，需要修改`investigation-cjzhuang2020/cjz_underg_2021_09`目录下的一系列文件，各个文件的文件名及其内容如下表所示：

完成以上配置后，即可在区块链中开始挖矿，启动`vehicle_test.py`，程序将打开浏览器并自动操作，可观察到如下图的字样：

此时启动`passenger_test.py`，程序将启动另一个浏览器窗口操作，此时需要在司机端浏览器窗口选择接客与否：

表 2-3 前端配置文件功能

文件名	功能
passengerAccounts.py	存储扮演乘客角色的账号的公钥
vehicleAccounts.py	存储扮演司机角色的账号的公钥
mapContract.js	关于地图智能合约的调用，需要修改合约地址，与 2.2.2 节记录的 StoreMap 合约地址一致
trafficContract.js	关于路径规划合约的调用，需要修改合约地址，与 2.2.2 节记录的 StoreMap 合约地址一致
passengers.js	存储每一位乘客的起始位置、上车位置和目的地位置
vehicles.js	存储每一位司机的初始位置

图 2-1 司机的初始化

图 2-2 司机手动接客

按下 pickUp 按钮，程序将继续自动运行，直至看到如下输出，调度成功：

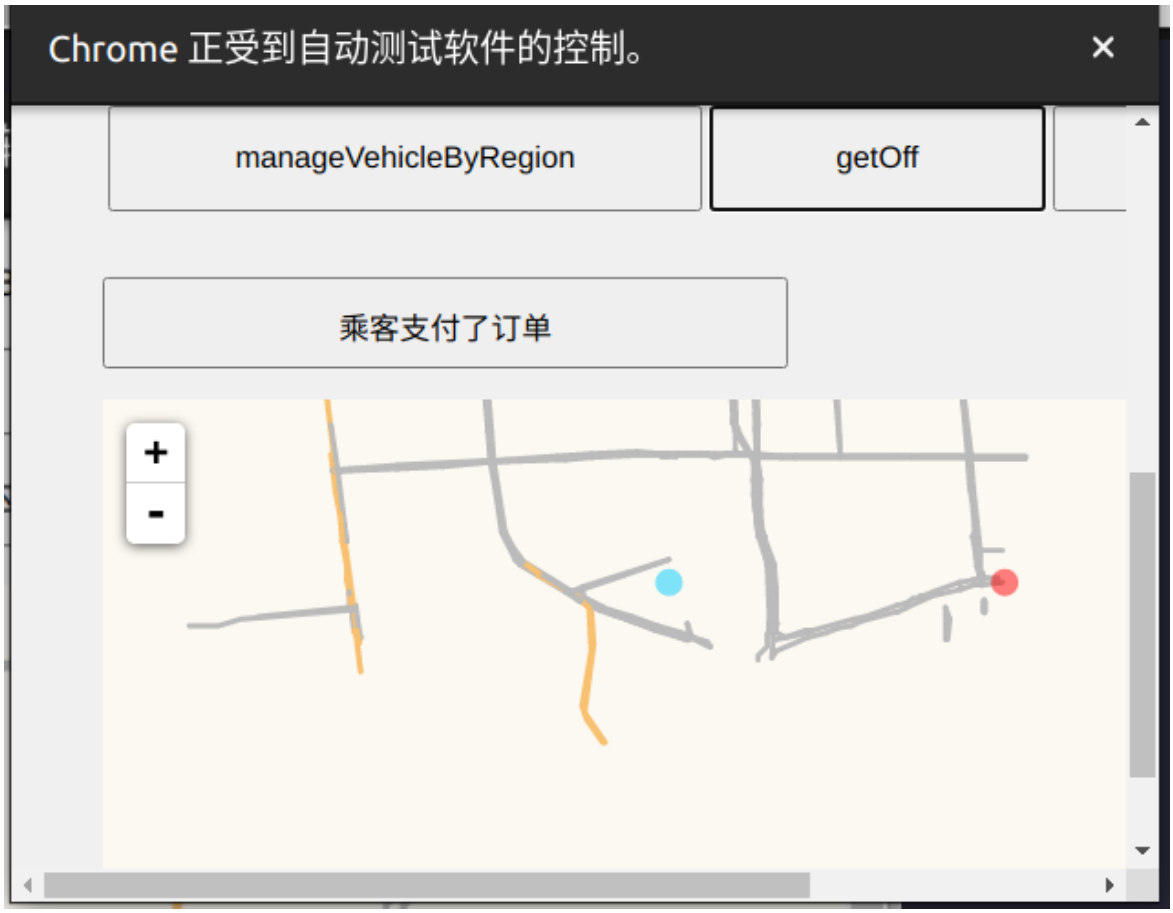

图 2-3 乘客抵达并支付

至此，原复现手册中的所有预期结果均已达到，调度系统复现实验全部完成。

2.4 问题及解决方案

笔者在进行出租车调度系统复现实验及其前置时，遇到许多原复现手册中未提及或未强调的注意事项；原版代码中，亦存在已废弃的语法和有待改进的设计。本节将陈述笔者遇到的问题及其解决方法。

2.4.1 创世配置文件指代不明

原复现手册中，并未指明在进行复现实验时使用的创世配置文件。经过笔者测试，附录 A 中的创世配置文件可用于所有出租车调度系统复现实验的全部环节。笔者已将其补充到代码仓库和新版复现手册中。

2.4.2 节点无法加入区块链网络

笔者在尝试建立双节点区块链网络时发现，无论如何使用`admin.addPeer()`函数，尝试将第二个节点连接到第一个节点，都无法成功，观察`net.peerCount`的值始终为 0，代表着第二个节点并未发现第一个节点。

实际上，节点与区块链网络中的其他节点使用的创世配置文件完全相同，是该节点加入此区块链网络的必要条件。因此，在配置第二个节点时，必须保证两个节点的创世配置文件完全相同。

需要注意的是，创世配置文件设定了各个账户的初始信息，例如余额和初始位置等等。因此，两个节点的创世配置文件完全相同，意味着两个节点中需要存在完全一致的账号。这要求程序员将第一个节点的 keystore 文件夹复制到第二个节点的相同的相对路径下。

综合以上两点，可以得出结论：当节点无法加入区块链网络时，需要检查节点的创世配置文件是否相同，以及节点的 keystore 文件夹内的内容是否一致。同时满足以上两点条件，节点才能成功加入区块链网络。

2.4.3 Selenium 库的废弃语法

Selenium 是一个用于自动化测试网页的工具组，其可以操控浏览器，模拟人类的操作，自动找到页面中的元素并与其进行交互。自发布以来，Selenium 已经发布了数个版本，其 API 亦有小幅改动。笔者在运行引入了 Selenium 的代码时，遇到了形如`selenium has no attribute named find_element_by_id`的错误信息。这是因为 Selenium 已经更改了定位元素的接口。在笔者使用的 Selenium 版本中，可使用形似`find_element(by=By.id, value=...)`的 API 代替上述废弃 API。同理，`find_elelements_by_class`也需要更换为`find_element(by=By.CLASS_NAME, value=...)`。具体的 API 使用方法，请参见 Selenium 的官方网站[2]。

2.4.4 合约相关错误

在浏览器的前端系统中进行操作时，按下 F12 打开开发者选项窗口，可能会见到有关汽油费的错误提示（Out of Gas）。经过排查，这是合约的编写、上传、调用的过程出现了错误。由于问题隐蔽、报错信息不能直接反应实际问题点，笔者花费了

[2]https://selenium.dev

较多时间调试合约相关之错误。现将部署合约以及与合约交互的正确操作流程陈述如下。

2.4.4.1 编译合约

笔者使用 Remix Desktop 进行合约编译。编译完成后，切换到编译选项界面，点击“Compilation Details”按钮，即可观察编译结果的详细信息，如下图所示：

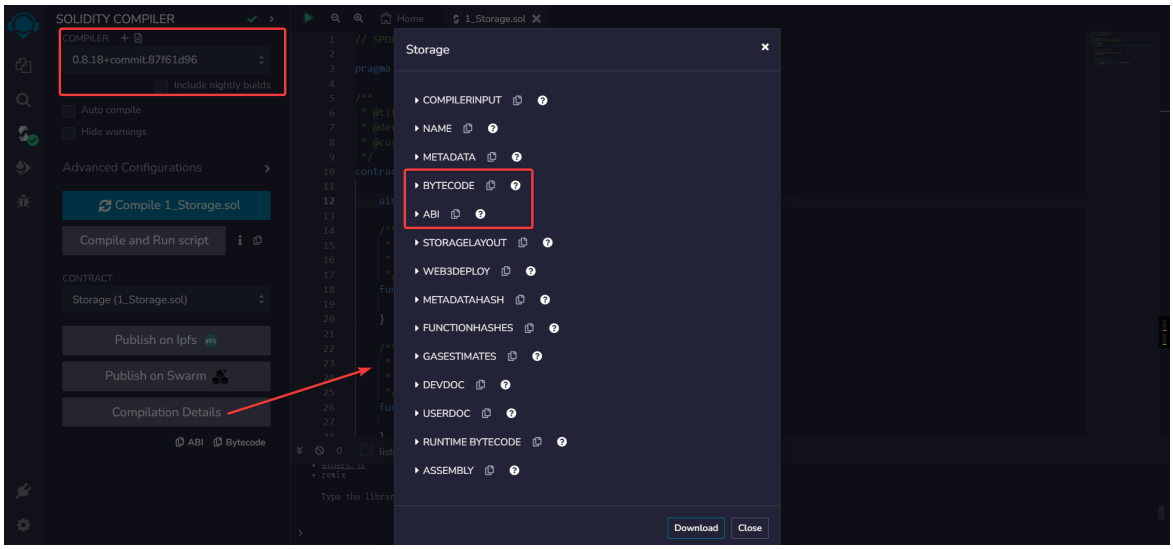

图 2-4 Remix IDE 编译选项界面

图中，更靠近中央部分的红框即为部署合约时需要使用的应用程序二进制接口（ABI）和以太坊虚拟机字节码（bytecode）信息，需要妥善记录保存。另外，在合约源代码所在的目录下的 artifacts 目录中，有一份与合约同名的 json 文件，该文件亦需要妥善保存备用。

2.4.4.2 部署合约

本文使用附录 B 中的合约部署模板进行合约部署工作，具体步骤如下：

1. 将代码模板中的“经过压缩转义后的 ABI”替换为经过压缩转义处理（即：去除多余空格，且为双引号等特殊字符进行转义处理）的上一步骤中获得之应用程序二进制接口

2. 将“获得的字节码字符串”替换为上一步骤中获得之以太坊虚拟机字节码

3. 修改变量名 Contract、contractInstance 为合适的名字，避免多份合约重名

4. 将经过以上修改的模板代码复制到正在运行 geth1 的控制台中，单击回车提交部署请求

5. 使用`miner.start()`开始挖矿，合约地址将随后在终端中显示

需要注意的是，在模板中，可见一名为 position 的字段。若该字段指代的 Geohash 在区块链的管辖范围之外（即：区块链管辖的 Geohash 并非 position 字段的前缀），将获得 out of the blockchain 错误，无法继续部署。因此，每次部署前，均需检查 position 字段指示的范围是否在区块链管辖范围以内。

2.4.4.3 合约交互

部署完成后的合约可以使用合约地址与其建立交互渠道，并调用合约内定义的方法。在 web3.js 库的辅助下，与合约进行交互仅需短短数行代码。最小示例如下：

代码 2.3: 合约交互

```
// 使用web3.js库
const Web3 = require('web3');
// 使用WebSocket协议连接到区块链，本例中端口号为8546
let web3 = new Web3(new Web3.providers.WebsocketProvider("ws://127.0.0.1:8546"));
// 部署合约时获得的合约地址
let contractAddress = '0x23b98f92ceac005e570b6768da377b3abd11012e';
// 编译合约时保存的应用程序二进制接口信息
let contractAbi = JSON.parse(fs.readFileSync('./contractAbi.json', 'utf-8'));
// 合约实例
let contractInstance = new web3.eth.Contract(mapContractAbi, mapContractAddress);
// 调用合约中定义的example方法
```

```
12    contractInstance.methods.example().then((res) => {/* 处理返回值res
      */});
```

在进行实验时，若出现难以排查的错误，应首先考虑合约相关错误。综合上述部署步骤，合约相关错误可以从以下几点进行排查：

- 从编译详情处获得的各项数据是否正确？例如，以太坊虚拟机字节码是否与应用程序二进制接口出自同一次编译过程？
- 部署合约时，是否正确配置各选项，例如 position 字段？
- 部署完成之后，是否正确记录合约地址？
- 进行合约交互前，是否确认区块链允许 WebSocket 协议连接？部分功能，例如订阅事件，仅能在 WebSocket 连接下进行。若无，则这些功能可能失效，与合约的交互可能失败。

2.5 本章小结

本章介绍了使用区域索引区块链进行基于区块链的出租车调度系统的复现实验之相关工作。首先，简要介绍该调度系统的功能及其组成部分；其次，介绍实验进行的环境配置和包括建立区块链、部署合约、上传地图、前端调试和使用等复现步骤，并展示了复现工作的运行结果，证明了复现工作的正确性；最后，介绍了在进行复现实验时遇到的典型疑难问题，对这些问题进行了产生原因的剖析，给出了解决方法和排查思路。

第 3 章 基于树状区块链的跨子链转账测试

3.1 树状区块链的跨子链转账

树状区块链，乃是以区域索引区块链为基础，旨在解决传统区块链单链结构面对大量区块时产生高昂性能代价的痛点。其借助 Geohash 技术，编码地理位置，并借此对区块链进行划分，形成类似于字典树的树状结构。

然而，这样的结构带来了一个问题：若某个账户的位置发生了较大改变，以至于它离开了目前所在的子链的管辖范围（即账户所在子链表示的 Geohash 范围不再是账户实际位置的 Geohash 编码表示的前缀），那么账户在新地理位置上发送的所有交易将失败，因为在逻辑上账户并不由管辖新地理位置所在片区的区块链直接管辖，后者可能根本没有关于该账号的任何记录，或者其上与该账户关联的信息并非切实。

对此，树状区块链的解决方案是：账户需要向一个特殊的管理账号发送一种特殊的交易，该交易及其携带的信息足以令管理账号从账户原先所在的区块链中，将账户的余额信息转移到新区块链上。此后，该账户由新区块链接手管辖，可以正常进行诸如发送交易等的区块链交互操作。由于上述转账过程与同一区块链内两不同账户间的转账不同，乃是横跨两个区块链的、同一账户之间的转账，故称该特殊转账过程为“跨子链转账”。

显然，在尝试解决单链结构的效率问题的同时，树状区块链引入了跨子链转账所需的额外时间开销。跨子链转账操作的效率，将对用户的使用体验有较大的影响。因此，本章将围绕跨子链转账这一主题进行探究。首先，设计并进行数组跨子链转账测试实验；其中，以较小规模的 10 账号跨子链转账实验测试跨子链转账的正确性；验证正确性后，再开展更大规模的跨子链转账测试，检验跨子链转账操作在不同压力情况下的运行效率波动；最后，给出一个简单的数学模型，探究树状区块链跨子链转账代价与其链上数据查询复杂度降低带来的性能优势基本持平时的临界条件，进而给出适用树状区块链代替传统单链结构区块链的应用场景建议。

3.2 测试设计思路

跨子链转账测试实验将在如图 3-1 所示的区块链网络中进行。

该区块链网络共由三个链组成，分别以链 w1，链 w11 和链 w12 指代。链 w1 为

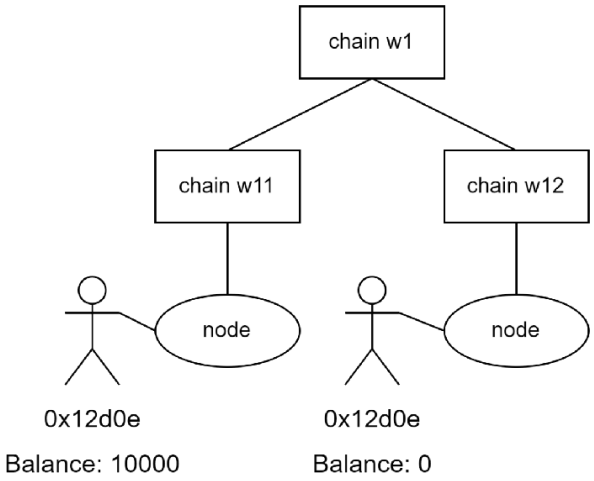

图 3-1 双子链树状区块链结网络结构

树状区块链中的分支区块链，其管辖的地理位置范围为 Geohash 编码为 w1 的所有区域；由链 w1 负责串联的子链 w11、子链 w12 均为叶子区块链，分别管辖 Geohash 编码前缀为 w11、w12 的所有区域，其下分别运行有一个节点，可以同传统区块链节点一样进行挖矿、交易等操作。初始时，两子链中存在相同的数个账号（类似图3-1中，位于不同子链下的同公钥账户 0x12d0e），但 w11 中的账号均有 10000 单位的余额，w12 中的账号的余额均为 0，模拟账号所有者进行了大范围物理位置变动后，尚未转移余额给管辖新地理位置的子链的情况。跨子链转账开始后，待所有链 w11 中的账号同时发起转账申请后，树状区块链将开始串行地遍历所有子链 w11 中的账号，并将它们在链 w11 的余额全部转移至链 w12 的对应账号之下。待转账完成之后，检查链 w12 中各账户的余额，即可证明跨子链转账的功能正确性。

在此过程中，监控程序将记录所有账号发起转账申请的时间戳、以及转账完成时的时间戳。通过分析以上数据，即能得到每个账号从串行处理开始至转账完成所花费的总时间，进而在初始待转账账号数量不同时，得出树状区块链在不同工况下的跨子链转账效率。

本测试将从 10 个账号的小规模跨子链转账测试开始，测试在小压力工况下跨子链转账功能是否运转正常；在此基础上，继续开展 20 账号，40 账号，80 账号，120 账号，160 账号及 200 账号跨子链转账实验，探究面对不同压力工况时树状区块链的性能表现变化。

3.3 环境配置

跨子链转账测试的测试环境如表 3-1 所示。

表 3-1 跨子链转账测试环境

中央处理器	Intel Core i5-12500H
图形处理器	Intel Iris Xe 80EU
内存	24GB
操作系统	Ubuntu 22.04.2 LTS
虚拟机	VMWare Workstation Pro 17

另外，还需安装`make`程序包，并下载树状区块链的源代码[1]。

3.4 测试步骤

3.4.1 编译并配置树状区块链

下载源代码后，在源代码目录启动终端，输入`make geth`，等待编译完成，即能在`build/bin`目录中发现 geth 二进制可执行文件。将其重命名为 geth-tree，并复制到`/usr/local/bin`目录中。

3.4.2 进行测试

代码仓库[2]给出了构建图 3-1 所示的区块链网络的数据文件和脚本，并附有一份实验手册。本节将简要介绍测试步骤。

1. 确认 w1、w11 和 w12 的数据目录中不存在`gethdata/geth`子目录。若存在，则需要将其删除；

2. 在代码仓库根目录中启动终端，运行`sh w1_init.sh`指令，启动分支区块链 w1，并留意形如 INFO [04-10|20:36:48.833] Started P2P networking self=”enode://.....” 的输出，记录该双引号包裹的字符串；

[1]https://github.com/xyongcn/BlockChain2017/tree/master/src/go-ethereum1.9.12-modify/go-ethereum

[2]https://gitee.com/endericedragon/transfer-2leaf

3. 在链 w11 和链 w12 的预加载脚本中，替换admin.addPeer()方法的参数为上一步中记录的字符串

4. 另启动两个终端，分别执行启动链 w11 和链 w12 的脚本，若观察到形如

```
INFO [04-10|20:54:02.335] Block synchronisation started
---k:aaaaaaaaaaaaw1,v.RegionId:w1,v.Number:1---
---parent.Number:0, branchb.RegionId:w1,ptd:131072---
!!commitBranchBlock[aaaaaaaaaaaaw1][1]--[td:262144]success!!
```

字样，即为子区块链和父区块链同步成功，可以进行下一步骤的实验；

5. 另启动一个终端，运行branchnode-remastered.js脚本。该脚本将监视父区块链 w1 的日志内容，并根据之运行相应代码，完成转账等操作；

6. 在链 w11 和链 w12 中启动挖矿后，启动一个终端，执行node transfer_test_step1.js指令。该指令将从链 w11 的矿工账号，分别为链上的其他账号转账 10000 单位的资产作为初始资金，在接下来的步骤中，这些资产将被转移到链 w12 中的同名账号上；

7. 确认初始资金全部到位后，执行node transfer_test_step2.js，终端中将出现与挖矿提示line:--handler-TX_request--不同的输出，等待，直至终端仅输出挖矿提示；

8. 执行node query_transfer_time_w11_w12.js，脚本将访问链 w11 和链 w12 上的所有区块，统计其中包含的交易及其详细信息，生成测试结果报告。

3.5 测试结果分析

使用eth.getBalance(accountAddress)函数，可以在 Geth 的 JavaScript 控制台中年轻松地查看给定账号的余额，进而检查转账是否成功。本章所有测试均已确认在跨子链转账结束后，原链 w11 中各账户的余额均为 0，而链 w12 中的对应账户余额为 10000 单位，从而证明了跨子链转账功能的正确性。

由于测试设计为将账户余额从链 w11 转账到链 w12 中，故在生成的测试报告中，仅需关注tx_request_w12.txt和tx_result_w11.txt文件即可。前者记录了发起转账申请的时间戳，由于所有账号乃是同时发起转账申请，故所有条目的时间戳均相同；后者记录了各账户在链 w12 真正收到资产的时间戳。根据后者（即tx_result_w11.txt文件），可以轻松重建树状区块链串行处理所有账户的顺序，以及处理各个账户分别花费的时间。

3.5.1 小规模跨子链转账测试结果分析

本文以 10 账号参与的跨子链转账测试结果作为小规模跨子链转账测试的测试结果。根据测试报告内容，所有 10 个账号均在时间戳 1683465655 发起转账申请，但在链 w12 上收到资产的时间戳不同，数据如下：

表 3-2 10 账号测试结果

账号公钥地址	收到资产的时间戳
0x4461e120a1bcbdc9e08730f59c7e169bac5de38f	1683465667
0x59cadf05182c56784b60960159c0fb4d16860d10	1683465680
0x8ed2d00a4ee496e51fab00ddc7561f85186e2a9c	1683465688
0x95fcbbba05858b53b829361a052450179d7a62ca	1683465694
0xcada164cb319316a133741dbaa1b40fcc8caec52	1683465703
0x1daf02e444bec7fc7fdbbac7704c57d001b19648	1683465711
0x023bc9309e89678b5de3ea084a5a91cc0679dd39	1683465720
0x4d326e5422c48ca1db8695bb59c9a58005a3fb44	1683465724
0x12d0e4381ef94a70a49252e35b9a65fadd3872b9	1683465735
0x0b424be2eb61a4fa045161198754613a93845857	1683465743
0xf41384cb20cd007daea6b0d7eefa3942ac44a3d1	1683465750

结合发起转账请求之时间戳，可以计算得到树状区块链为每个账户办理转账所花费的时间。笔者使用形如附录 C 的 Python 代码进行可视化，如图 3-2 所示：

根据图 3-2 计算，得到如下测试数据：

观察图 3-2 不难发现，虽然 10 个账号的转账申请确实是同时发起的，但由于以太坊仅能串行处理的特性，导致从第一份转账请求发起，到最后一次转账交易完成

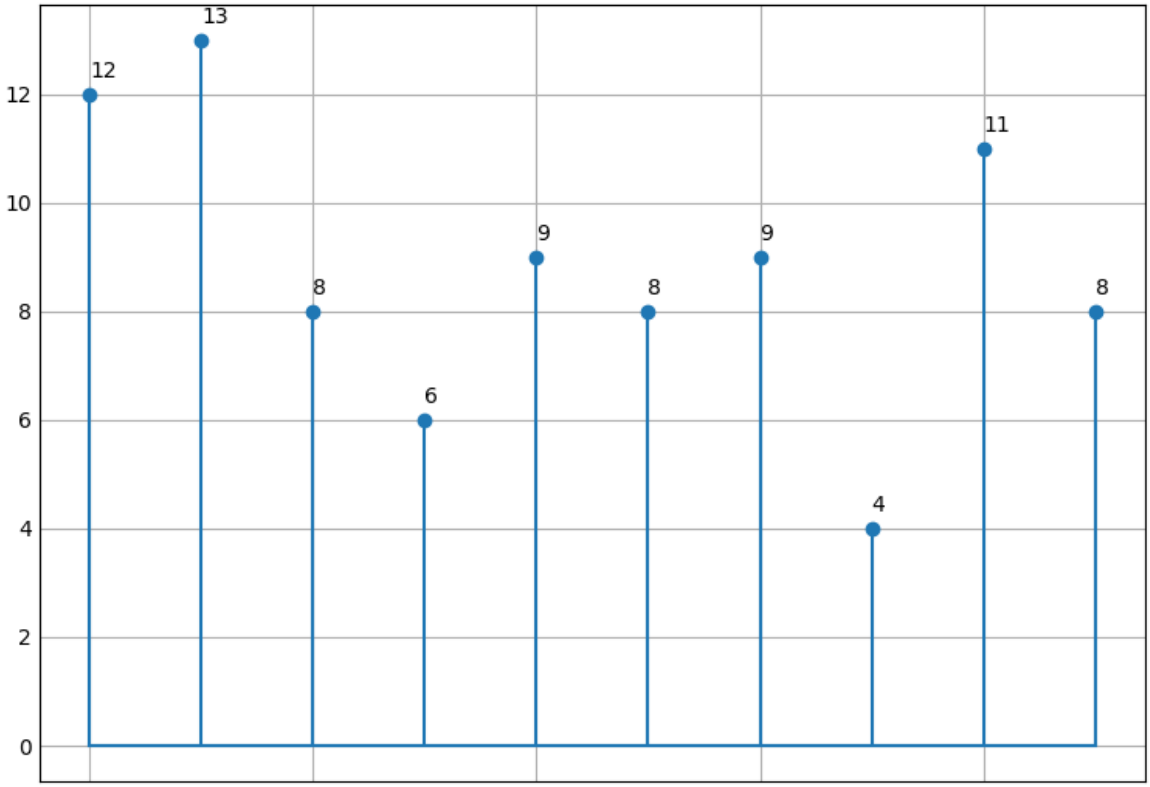

图 3-2 10 账号跨子链转账测试的可视化

表 3-3 10 账号测试数据

总耗时（秒）	95
最快转账处理速度（秒）	4
最慢转账处理速度（秒）	13
平均转账处理速度（秒）	8.8000
转账处理速度方差（秒 ^{2}）	6.5600

并写入区块为止的总耗时较为夸张。同时，参与测试的 10 个账号之转账处理时间方差较大，最快转账处理速度较最慢处理速度的差距足有 9 秒。

3.5.2 大规模跨子链转账测试结果分析

随着请求转账的账号数量增加，其转账处理效率是否会随之变化？本文继续以类似的方法，开展了 20 账号、40 账号、80 账号、120 账号、160 账号、200 账号的跨子链转账测试。由于篇幅限制，本节将不展示可视化图示，仅展示经过计算得出的统计数据。

表 3-4 测试数据汇总

	20 账号	40 账号	80 账号	120 账号	160 账号	200 账号
总耗时（秒）	163	319	654	978	1363	1621
最快转账处理速度（秒）	4	2	1	2	1	1
最慢转账处理速度（秒）	16	17	21	17	27	21
平均转账处理速度（秒）	9.0556	8.6216	8.9589	8.5789	8.5188	8.7622
转账处理速度方差（秒 ^{2}）	8.0525	11.0460	19.0805	8.3666	25.9446	12.8407

观察表 3-4 的数据，可以提出以下猜想：

- 转账过程耗时和参与转账的账号存在线性关系；
- 虽然转账处理速度方差较大，但平均处理速度较为稳定，大约在 8.5 秒到 9.1 秒之间。

其中，第二个猜想可以从表 3-4 中较直观地得到，故本文将重点讨论第一个猜想的验证。

3.5.2.1 验证转账耗时和账号数量的线性关系

本文使用最小二乘法进行线性拟合计算。记账号数量为 x，假设 $time_\theta(x)$ 为 x 个账号跨子链转账需要花费的时间，那么后者可以写作：

$$time_\theta(x) = \begin{bmatrix} 1 & x \end{bmatrix} \cdot \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

根据最小二乘法：

$$\vec{\theta} = (\vec{X}^T\vec{X})^{-1}\vec{X}^T \cdot \vec{Y}$$

其中，$\vec{X}$，$\vec{Y}$ 是分别为样本的输入向量和输出向量。根据以上算法，可得本例中的各个向量为：

$$\vec{X} = \begin{bmatrix} 1 & 20 \\ 1 & 40 \\ 1 & 80 \\ 1 & 120 \\ 1 & 160 \\ 1 & 200 \end{bmatrix} \quad \vec{Y} = \begin{bmatrix} 163 \\ 319 \\ 654 \\ 978 \\ 1363 \\ 1621 \end{bmatrix}$$

带入最小二乘法公式进行计算，可得：

$$\theta = \begin{bmatrix} -4.68767123 \\ 8.26794521 \end{bmatrix}$$

$$R^2 = 1 - \frac{\sum_i (y_i - \overline{y})^2}{\sum_i (y_i - times_{theta}(x_i))^2} = 0.9982362552443657$$

其中，x_i，y_i 代表 $\vec{Y}$ 的各个水平分量，$\overline{y}$ 代表 $\vec{Y}$ 的算术平均值。

拟合度 R^2 非常接近 1，这正证明了自变量——账号数量 x 和因变量——跨子链转账耗时 $time_theta$ 确实可以在统计学上认为存在线性关系 $time_\theta(x) = -4.68767123 + 8.26794521 \cdot x$，进而验证了以太坊顺序串行处理收到的所有交易的特点。

3.6 基于测试结果进行数学建模

传统区块链单链结构的查询效率低，但胜在无需额外操作维护正确的账户状态；而树状结构的查询效率高，但引入了跨子链转账的额外开销。根据使用场景的不同，用户需要在上述两种区块链实现中恰当进行选择，方能在特定的应用场合获得更好的使用体验。本节将基于已收集的测试数据，建立简单的数学模型，探讨在不同场景下使用树状区块链和传统区块链的理论性能差异，为用户在两种区块链实现之间的选择提供建议。

3.6.1 静态查询复杂度分析

静态查询复杂度分析，约束所有账号的地理位置不发生变化，因此不会考虑跨子链查询的情况。本小节将基于以上约束，讨论单链结构区块链和树状区块链查询信息的复杂度情况。

在单链结构区块链下，所有链上信息均存储在同一条链中。若进行一次查询，最坏情况下需要遍历整条链才能获得结果。记总区块数为 n，则该过程的平均时间复杂度为 $O(n)$。

在树状区块链下，情况较为复杂，为简单起见，本节讨论一条父链，x 条子链组成的双层树状区块链，并假设所有区块均匀地分布在子链中。仍记总区块数为 n，那么每条子链中包含 $\frac{n}{x}$ 个区块。此时若进行一次查询，最坏情况下仅需遍历一条子链即可，平均时间复杂度为 $O(\frac{n}{x})$。

注意到当分母 x 越大，查询的时间复杂度越低。这是因为，在区块均匀分布的前提下，数状结构的深度随分支的增加而减少。结合树状区块链分支的实际意义，可以得到以下结论：

- 当链上交易发生的地理位置跨度较大，且在各地理位置范围分布较均匀时，适合使用树状区块链而非单链结构区块链；
- 当链上交易发生的地理位置非常集中时，树状区块链的表现和单链结构区块链接近，故无需使用树状区块链。

3.6.2 动态查询复杂度分析

动态查询复杂度分析，假设节点在树状区块链的各子链间周期性移动，从而将跨子链这一开销纳入考量。本小节中，将针对节点中的一个账号进行分析。该账号行为描述如下：处于某一条子链中时，进行数笔交易，这些交易分别记录在不同的区块中。随后，账号立即转移前往下一个子链，此时若进行查询操作，必须先进行跨子链转账，将该账户之余额转移到新子链上，然后再进行查询。

在单链结构区块链下，情况与静态查询情况类似，进行一次查询，最坏情况下仅需遍历一条子链即可，平均时间复杂度为 $O(n)$。

在树状区块链下，情况更为复杂。记 $lcm(c,q)$ 为账号跨子链的间隔之间产生的区块数量 q 和查询间隔之间产生的区块数 c 的最小公倍数，那么在产生 $lcm(c,q)$ 个区

块的过程中，将发生 $\frac{lcm(c,q)}{c}$ 次查询，且账号将进行 $\frac{lcm(c,q)}{q}$ 次跨子链移动。那么 $\frac{lcm(c,q)}{c}$ 次查询的总时间复杂度为 $O(q \times \frac{lcm(c,q)}{c}) + \frac{lcm(c,q)}{q} \times T_{cross}$，其中 T_{cross} 为单个账户跨子链转账的时间开销。那么，平均到每一次查询的时间开销即为 $(O(q \times \frac{lcm(c,q)}{c}) + \frac{lcm(c,q)}{q} \times T_{cross}) \times \frac{c}{lcm(c,q)} = O(q) + \frac{c}{q} \times T_{cross}$。

令 $lcm(c,q) = n$，比较两区块链实现在生成相同区块数量下的复杂度表现，列不等式：

$$O(lcm(c,q)) \geq O(q) + \frac{c}{q} \times T_{cross}$$

由于在遍历长为 $length$ 的区块链链条时，可以认为找到目标区块的期望复杂度为 $\frac{length}{2}$；同时，经过实测，T_{cross} 的值可以取表 3-4 中跨子链转账测试的平均处理速度的平均值（经过计算，大约 8.7493 秒），故可以改写上述不等式为：

$$\frac{lcm(c,q)}{2} \geq \frac{q}{2} + \frac{c}{q} \times 8.7493$$

当满足上述不等式时，树状区块链更有可能提供相较传统区块链更优越的性能。

观察树状区块链的动态查询复杂度表达式可知，当查询非常频繁，即 c 的值变小时，复杂度将相应降低。

注意到，q 的增大在 $O(q)$ 中，对时间复杂度起到增加作用，却在 $\frac{c}{q} \times acc \times T_{cross}$ 中起到减少时间复杂度的作用。这是因为，增大 q 相当于容许子链拥有更长的长度，账号所有者停留在同一子链中的时间延长，也就相应减少了跨子链转账操作的次数。因此，即便是在树状区块链中，用户也需要合理设计子链的管辖范围，将节点跨子链的频率控制在合理的区间内。

综上所述，在下列场景中，选择树状区块链将比选择单链结构区块链更优：

- 账号的地理位置变化频率适中；
- 数据查询请求量较大。

3.7 本章小结

本章介绍了树状区块链为保证子链间数据一致性而生的新概念——跨子链转账操作，介绍了该操作引入的额外时间复杂度。随后，设计并进行了一系列测试，按照

压力自小到大的顺序逐步验证了跨子链转账操作的结果正确性、具体测量了该操作带来的额外时间开销，将其可视化并统计整理，进行统计学分析。在分析时，发现并合理利用一些最优化方法验证了参与转账的账户数量与转账耗时的线性关系，证明跨子链转账这一操作为串行处理。最后，分别在静态和动态的场景下建立简单的数学模型，从理论与实际测量数据结合的角度比较了传统单链结构区块链和树状区块链的性能表现，并就在何场景下使用何者给出了一些建议。

第 4 章 以出租车调度系统为背景的树状区块链测试

4.1 树状区块链上的出租车调度系统测试简介

树状区块链将单链结构转化为树状结构的改进，提升了区块链技术在地理位置有关的应用场景下的理论效率。但实验室尚未对树状区块链在实际应用环境中的表现进行测试。因此，测试树状区块链在实际应用中的真实性能，证明其相较传统单链结构区块链具有可观性能提升的工作迫在眉睫。本章将以实验室已有工作——基于区块链的出租车调度系统为例，使用 geth1 和 geth-tree 分别构建不同拓扑结构的区块链并于其上运行该调度系统，统计在一定压力负载下司机侧和乘客侧各关键节点的时间戳并将结果可视化，以测试树状区块链在实际应用场景中的性能表现。同时，重构已有的脚本代码，增强其可扩展性和易用性，方面后人进行本测试实验的复现。

4.2 测试设计思路

本测试分为基于树状区块链 geth-tree 的测试和基于区域索引区块链 geth1 的测试两部分，用以对比两种区块链实现在同一套系统下的性能表现差异。

两部分实验中均在真实世界地图中 Geohash 编码前缀为 wx4e 的区域下进行。树状区块链部分的实验在该区域下的细分区域 wx4en、wx4ep、wx4eq 和 wx4er 区域下进行。每个区域中，均存在 16 位司机和 32 位乘客，所有司机的初始位置均相同，所有乘客的出发地点和目的地也相同。以上地点的选点工作基于蒙思洁完成的真实地图信息提取与筛选工作进行，已提前确保选择的路线可以在真实世界地图上导航成功。

本测试使用 JavaScript 脚本模拟司乘交互行为。司机模拟脚本负责读取司机的公钥地址、初始位置，并将其上传到链上；随后，司机将监听一系列合约中定义的事件，例如乘车请求事件、乘客上车事件和付款事件等，并作出接单、导航、设置车辆状态等响应。乘客模拟脚本负责读取乘客的公钥地址、起始点位置和目的地位置，并将其上传到链上；每隔一段时间（在树状区块链中为 3 秒，在区域索引区块链中为前者的 $\frac{1}{4}$），将有一名乘客发射乘车请求事件。发射乘车请求事件后乘客将在临近区域周边搜索与自身曼哈顿距离最近的车辆并尝试占有。若车辆已被占用，则等待一段时间（等待时间为冲突次数的一次函数）后再次重复上述步骤，直至车辆分配成

功。乘客端、司机端的模拟脚本运行逻辑可分别用图4-1、图4-2所示之流程图表示。

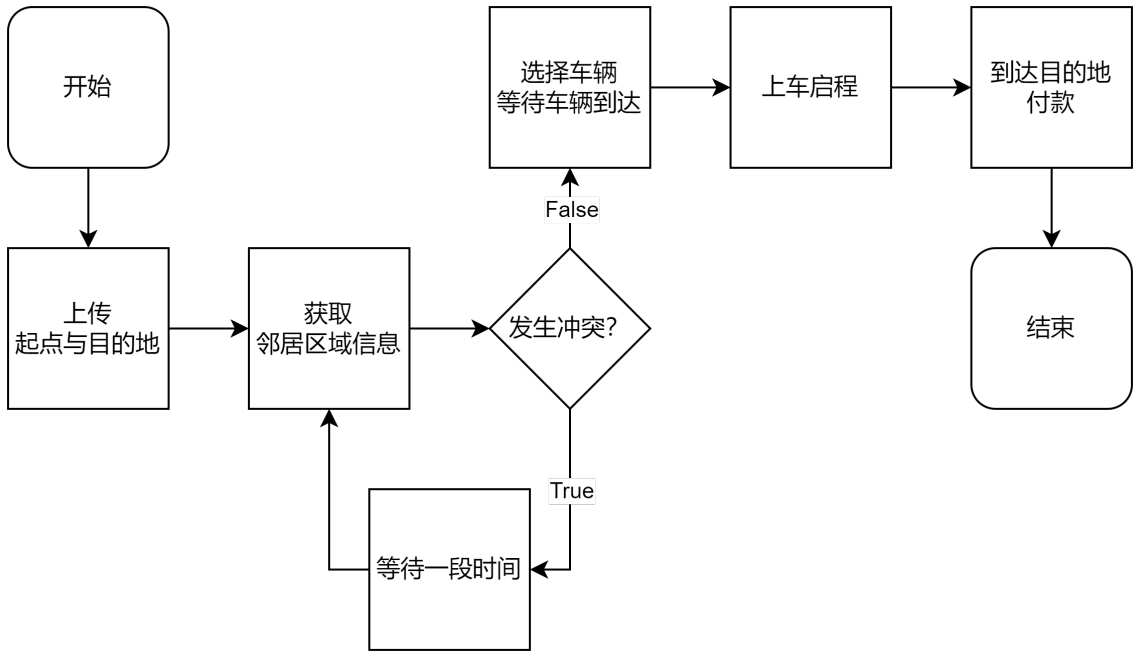

图 4-1 乘客端模拟脚本运行逻辑

进行树状区块链部分的实验时，首先搭建树状结构，在每个子链上分别部署合约。待合约部署完毕后，所有子链同时挖矿，并同时运行模拟司乘交互行为的 JavaScript 脚本，该脚本读取对应子链管辖的细分区域内的司乘信息，并记录司乘双方各自在调度过程关键节点的时间戳。

在区域索引区块链测试部分中，测试步骤大致相同，但运行司乘交互模拟脚本时，应令其读取所有四个细分区域内所司乘的信息，模拟不进行区域细分，使用单链结构区块链运行出租车调度系统的应用场景。

4.3 测试环境

基于树状区块链的调度系统测试在表4-1环境中进行。由于运行四子链的性能开销较大，笔者使用 Windows 下的 Linux 子系统（WSL 2）代替虚拟机，以提升开发体验。

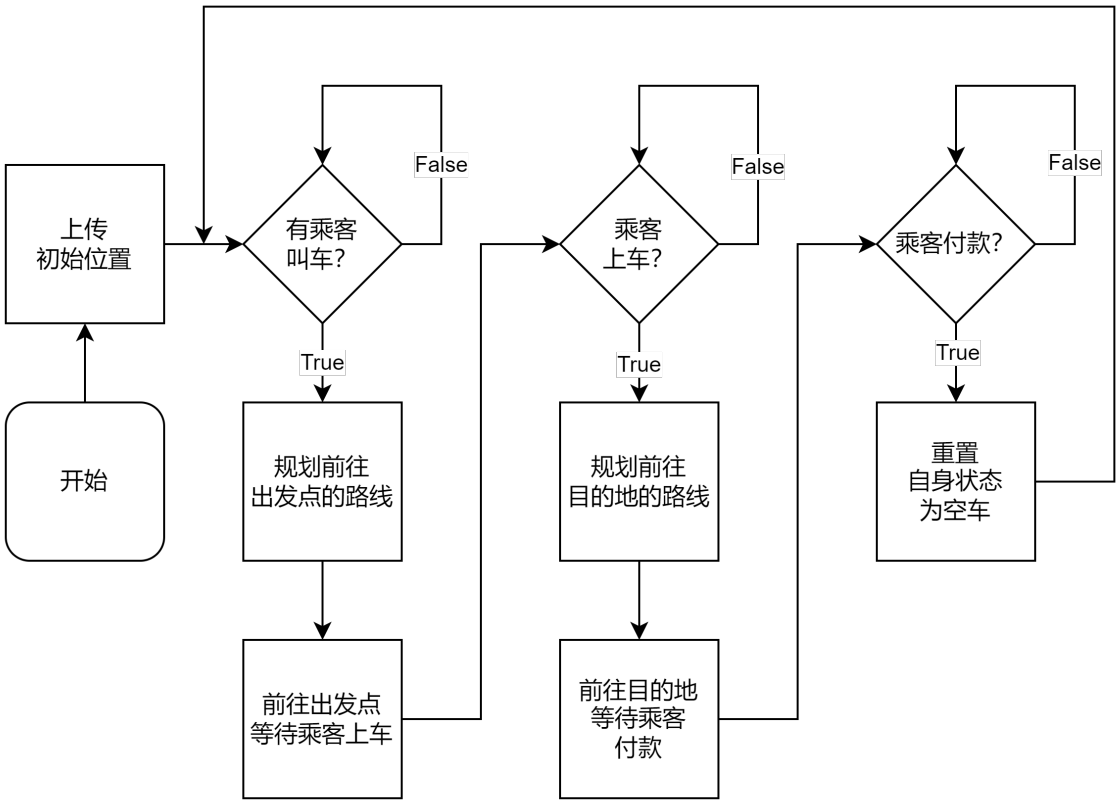

图 4-2 司机端模拟脚本运行逻辑

表 4-1 树状区块链调度系统测试环境

中央处理器	Intel Core i5-12500H
图形处理器	Intel Iris Xe 80EU
内存	24GB
操作系统	Ubuntu 22.04.2 LTS
虚拟机	Windows Subsystem Linus 2

4.4 准备数据

4.4.1 确定司乘位置信息

本节将分别在各树状区块链子链的管辖区域内，为接下来的测试选择一条能够导航成功的路线，其始端和终端分别作为乘客的起点与目的地；司机的初始位置，则设置为乘客的目的地，即导航路线的终端。经过挑选和在调度系统中的验证工作后，笔者最终选择表4-2中所示之点位，作为本章测试的测试数据集。

表 4-2 测试数据集选点

区域 Geohash 前缀	起点	目的地
wx4en	wx4enscgue5	wx4enrq9mm9
wx4ep	wx4epb8scg1	wx4ep8e5gw0
wx4eq	wx4eq7rgmxk	wx4eqt6u0vu
wx4er	wx4erd4xkyz	wx4erw9rmze

4.4.2 划分账号扮演的角色

基于区块链的出租车调度系统中，账号可以扮演司机或乘客角色。本测试共有192 个账号参与其中，且司机角色与乘客角色的数量之比为 1 : 2。不仅如此，测试还需保证在四条子链中包含相同数量的司机账号和乘客账号，以维护子链之间的测试公平性。经过计算，本测试将如表4-3所示，对链上创建的共计 192 个账号（表中以`eth.accounts`指代）进行划分，该划分方案恰好满足所有子链包含相同数量的乘客账号和司机账号的测试需求。

表 4-3 测试账号角色划分

区域 Geohash 前缀	司机账号	乘客账号
wx4en	`eth.accounts.slice(0, 16)`	`eth.accounts.slice(16, 48)`
wx4ep	`eth.accounts.slice(48, 64)`	`eth.accounts.slice(64, 96)`
wx4eq	`eth.accounts.slice(96, 112)`	`eth.accounts.slice(112, 144)`
wx4er	`eth.accounts.slice(144, 160)`	`eth.accounts.slice(160, 192)`

上述划分方案及各账号相关的位置信息将以 JSON 格式存储到文件，供模拟运行脚本读取之用。

4.5 进行模拟运行测试

本节的详细测试步骤已记录于在线代码仓库[1]，故本节仅简要介绍大致测试方法。

1. 准备司乘数据，使用 JSON 格式存储每位司乘的信息，并将其均匀分作 4 份，作为树状区块链子链的测试数据；同时，准备一套存储完整司乘信息的文件，作为区域索引区块链的测试数据；

2. 建立有四个子链的树状区块链网络，确保四个子链中均拥有相同的 192 个账号，并已经为它们执行解锁操作；

3. 在四子链上分别部署合约，并使用得到的合约地址更新各脚本中存储的合约地址，随后上传地图；

4. 依次启动子链挖矿，注意分配 CPU 核心数量相同以控制变量，随后选择冲突等待时间和冲突次数的一次函数为 $t_{waiting} = 10000 + 4000 \times count$，其中 $count$ 为冲突次数，编辑模拟司机与乘客交互行为的脚本后启动之，等待测试结束；

5. 模拟脚本运行结束后，将生成测试报告，可对其进行数据处理和可视化；

6. 建立仅包含一个节点的区域索引区块链网络，于其上进行相似的测试。

4.6 乘客端测试数据分析

本节采用 Python 3 编程语言的 MatPlotLib 库作为数据可视化工具，从乘客端方面对收集的测试数据进行分析。

本测试统计三个指标：乘车请求提交耗时、车辆分配耗时和到达并付款耗时。上述三个指标与图 4-1 中的对应关系如下：

- 乘车请求提交耗时：从“开始”开始计时，直至第一次执行“发生冲突?”检查的耗时；

[1] https://gitcode.net/qq_39710999/taxi-4-leaves

- 车辆分配耗时：从第一次执行“发生冲突？”检查开始，直至运行至“选择车辆等待车辆到达”的耗时；
- 到达并付款耗时：从“选择车辆等待车辆到达”开始计时，直至“结束”的耗时。

应用上述指标规则，对乘客端测试数据进行计算和可视化处理，其可视化结果如图4-3所示所示。

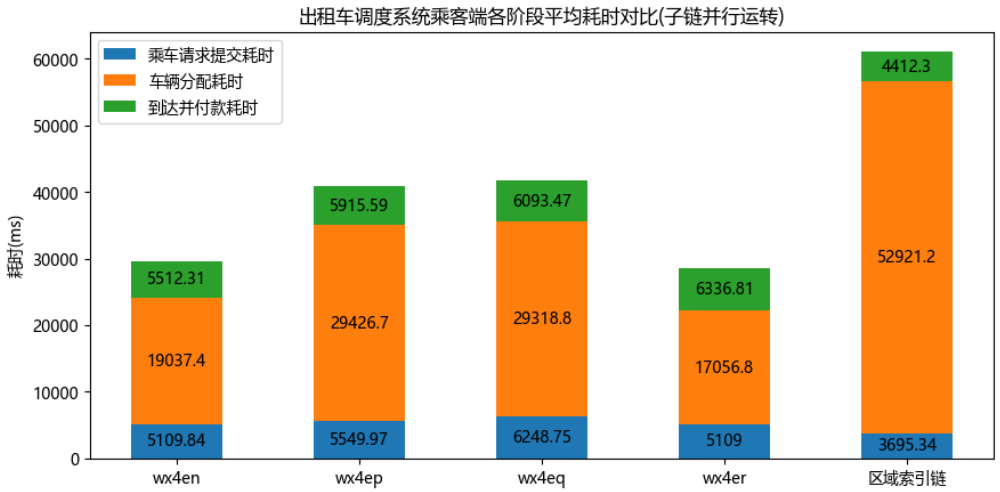

图 4-3 乘客端各阶段平均耗时对比 (子链并行运转)

分析图4-3可知，树状区块链在为乘客分配车辆时，展现出了更为优越的性能。在该阶段中，树状区块链最快用时约 17056.8 毫秒即完成车辆分配，四子链平均车辆分配时间为 23709.9 毫秒；相比之下，区域索引区块链共耗时 52921.2 毫秒完成车辆分配，耗时增长至前者的 2.23 倍。

进一步分析测试报告发现，区域索引区块链上发生的冲突次数远远大于树状区块链中的各条子链上的冲突次数，是拖慢调度系统在其上运行速度的主要原因。测试报告指出，在区域索引区块链上，一共发生了 266 次这样的冲突事件；作为对比，冲突事件在 wx4en 子链、wx4ep 子链、wx4eq 子链和 wx4er 子链中的发生次数分别仅有 18、31、33、12 次，远低于区域索引区块链上的冲突次数。

4.6.1 减少无关变量后的补充测试

注意到，乘客方提交乘车请求、到达目的地并支付路费时，其处理速度相较区域索引区块链并未展现出更高的效率，耗时甚至不降反升。由于测试时四条子链并行运行，且笔者观察到测试时计算机中央处理器负载极高，一度达到满载，故做出如下合理推测：四条子链并行的测试方法极有可能受到测试机器性能上限之影响，从而引入无关变量，令测试数据失真。

为缓解测试机器性能上限对测试结果的影响，笔者补充进行了四次树状区块链实验，每次实验中，仅有一条子链单独运行模拟运行测试，其余三条子链空闲无负载。使用新方法重新测试后，子链的性能表现有了极大的提升，如图4-4所示。

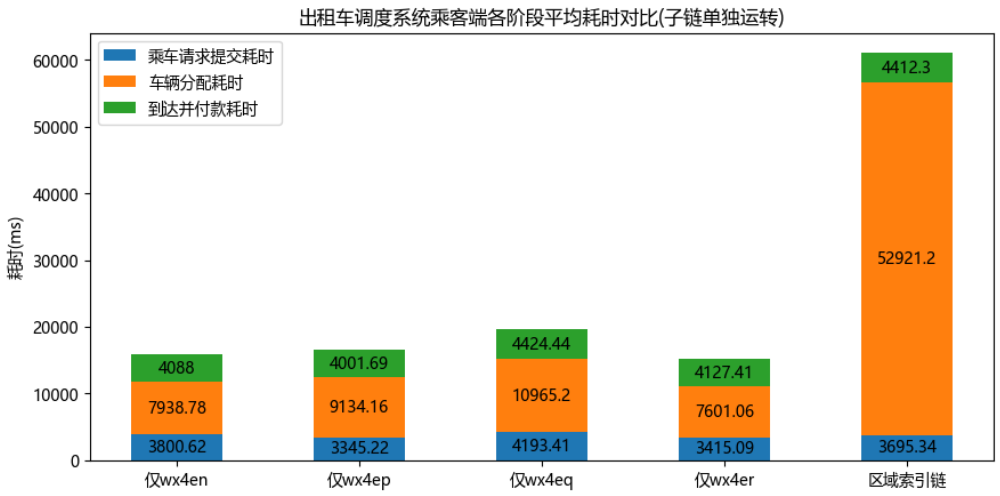

图 4-4 乘客端各阶段平均耗时对比 (子链单独运转)

在各个子链独立运行的场景下，测试机器的性能上限带来的影响进一步减弱，此时树状区块链爆发出了远超区域索引区块链的性能表现。在乘车请求提交平均耗时方面，树状区块链与区域索引区块链在该阶段的平均耗时基本一致；在车辆分配耗时方面，四条子链的平均耗时为 8909.8 毫秒，相较四子链并行运行的工况减少了 62.4%，相比区域索引区块链，其时间开销更是缩短到原来的 16.8%。在到达并付款耗时方面，由于调度系统运行至该阶段时，链上存在的区块数量已较有规模，树状区块链将区块分散至子链以换取更高的处理速度的优势得以提现，此时，四条子链的平均时间消耗为 4160.4 毫秒，以 251.9 毫秒的优势领先区域索引区块链。

4.7 司机端测试数据分析

本节使用与上一节相同的工具，对司机端方面进行测试数据分析。

本测试统计三个指标：接单并导航至上车点耗时、导航至目的地耗时和确认乘客下车并付款耗时耗时。上述三个指标与图4-1中的对应关系如下：

- 接单并导航至上车点耗时：从“有乘客叫车”检查为真开始，至“前往出发点等待乘客上车”的耗时；
- 导航至目的地耗时：从“乘客上车”检查为真开始，至“前往目的地等待乘客付款”的耗时；
- 确认乘客下车并付款耗时：从“乘客付款”检查为真开始，至“重置自身状态为空车”为止。

应用上述指标规则，对司机端测试数据进行统计和可视化处理。此处沿用了乘客端测试的方法，同时测试了在四条子链并行运转和分别运转的工况下调度系统的运行情况。经过处理和汇总的数据可视化结果如图4-5所示所示。

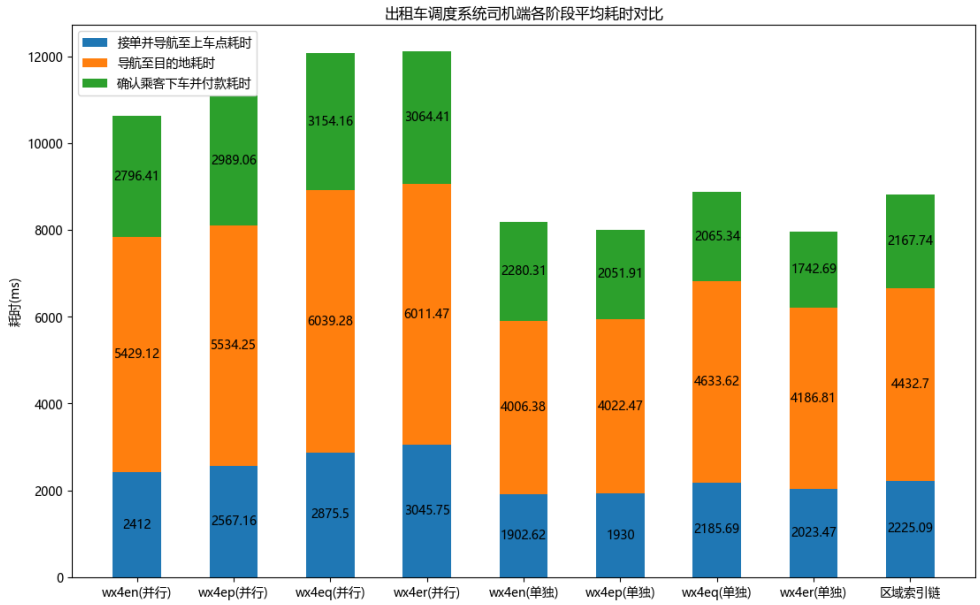

图 4-5 出租车调度系统司机端各阶段平均耗时对比

由图4-2可知，司机端不存在冲突的情况（即：系统不允许同一时间有两位司机接受同一位乘客的订单），加之并行运行为测试计算机带来的较大负担，在四子链并行运行的工况下，树状区块链的性能表现大幅落后于区域索引区块链。然而，在四子链分别单独运行，测试计算机性能影响因素被减弱的工况下，树状区块链仍然以567.7 毫秒的平均周转耗时优势超越了单链结构的区域索引区块链。测试证明，树状区块链相较传统区块链的性能优势并非仅来自调度系统运行时产生的诸如冲突等特殊情况，其树状结构的物理设计功不可没。然而，四条子链同时运行产生的性能开销，有可能制约树状区块链的综合性能，令区块链网络的运行效率产生不可忽视的下跌。

4.8 本章小结

本章介绍了树状区块链在基于区块链的出租车调度系统上进行的各项测试及其数据分析。首先，阐述了设计测试的思路及大致方法。本章测试在采用经过分块处理的真实世界地图的基础上，采用脚本模拟司机与乘客的交互行为，在具有四个叶子链的树状区块链系统上部署出租车调度系统合约并运行模拟脚本，以收集实验数据。接下来，展示了实验环境配置，及较具体的数据准备、模拟测试的步骤方法。最后两节分别从乘客端角度和司机端角度出发，选取了不同的性能指标进行统计分析，并将结果进行可视化。针对树状区块链优于传统区块链的情况，给出了直观的展示；针对树状区块链不及传统区块链的情况，则给出了测试平台性能上限的猜想，同时设计进行补充实验加以验证该猜想，最终获得了树状区块链在实际应用场景的不同工况下的性能表现，证明了它相较传统的单链结构区块链的性能优越性。同时，证明了测试平台性能上限对树状区块链运行的影响，解释了树状区块链提供更好性能的代价之一——对系统资源的占用情况较为严重。

第 5 章 使用 Rust 重写树状区块链

5.1 从以太坊到 Substrate——基于 Rust 重写的改进思路简析

树状区块链是在以太坊官方客户端 Go-Ethereum 的源代码上修改而来，因此，虽然在结构上做出了很大的调整，它也继承了许多 Go-Ethereum 的特点，例如共识算法和 EVM 虚拟机等，其性能表现也依然受制于 Go-Ethereum。从整体上评估，本文第三章通过统计学方法，验证了以太坊顺序串行执行交易的特点，这样的执行策略使得以太坊在面对高并发请求时的处理效率不尽如人意；从局部评估，研究[22]表明，以太坊所使用的共识算法之一——基于工作量的证明（Proof of Work），其性能表现已落后其他更先进的算法。然而，以太坊并未在源代码层面留有太多的可扩展空间，这也意味着许多诸如更换共识算法，修改交易执行逻辑等的自定义修改在实践时困难重重，限制了在以太坊平台改良优化的空间。因此，另寻一全新的平台，使用全新的工具重写树状区块链，方能绕过以太坊开发平台的短板，有效提升树状区块链的性能表现。在本章中，笔者将选择基于 Rust 编程语言的 Substrate 区块链框架作为目标开发平台，介绍使用 Rust 编程语言重写树状区块链的相关内容。

本章首先对 Substrate 区块链开发框架进行评估，解释为何笔者选择它作为重写工作的目标平台；接下来，更为详细地介绍树状区块链的结构，估算重写工作的大致工作量，确定模块划分，并从中选择本章计划完成的部分——账户地理位置信息，进行编程实现和验证性演示。

5.2 Substrate 框架的选择与评估

Substrate[11]由 Parity Technologies 推出，是一套基于 Rust 编程语言开发的开源的区块链开发框架。在 Substrate 诞生前，人们花费了大量的精力，试图设计一个支持多链结构的新型区块链。然而，所有研究结果最终导向了同一个结论：当下做出的深思熟虑的选择很可能成为未来的绊脚石。这是因为随着时间的推移，区块链依赖的某些特定的技术或假设，可能会阻碍并最终扼杀创新[23]。因此，以太坊创始人之一 Gavin Wood 成立了 Parity 技术公司，力图改写这一局面。他们的处女座——以太坊客户端 Parity，在相同的硬件配置环境下展现出了远胜 Go-Ethereum 的性能表现，提升幅度达到了可观的 89.8%[24]；在后续开发 Parity 自研的区块链网络 Polkadot 时，

Gavin 意识到，仅需将 Polkadot 网络进行抽象，剥离部分细节，即能获得一个可扩展性极强，适用范围更广的区块链框架。在 2018 年，Polkadot 和用于开发它的区块链框架终于被分离开，成为两个独立的项目，而后者，即是本章讨论的主角——Substrate。

具体地，Substrate 在设计时严格遵循了三点原则：

- 将 Rust 编程语言作为代码库的核心编程语言。Rust 是一种系统编程语言，它具有高性能、安全和并发性的优势[20]。通过引入所有权、借用和生命周期等概念，Rust 可以在编译时检查并确保内存安全。Rust 还支持函数式、并发和泛型编程的特性，使得代码更加灵活和可复用。虽然 Rust 语言的学习曲线较为陡峭，但其极快的速度，极具辨识度的内存管理方式，灵活的抽象能力，以及可编译为 WebAssembly 的特点使它成为操作系统、嵌入式系统、网络编程等领域极受欢迎的选择[25]；

- 将 WebAssembly 作为应用程序逻辑的执行环境。WebAssembly 是一种新型代码，由万维网联盟创建，可从 Rust、C、C++ 等语言编译获得，且受到多种 JavaScript 引擎的广泛支持，具有良好的兼容性[26]。Substrate 的易升级性也建立于 WebAssembly 的基础之上：它将区块链的具体业务逻辑编译为 WebAssembly 字节码，并存储于区块链的数据存储区中，用户可以像发起普通交易一样发起一个申请修改链上存储的 WebAssembly 字节码的交易，从而便利地更新升级区块链系统；

- 广泛使用分层抽象、泛型实现和灵活的 API 作为主要的编码实践，并将库分离为不同的体系结构组件。在核心功能方面，Substrate 官方提供了许多不同的实现，例如数据库层的 RocksDB 和 ParityDB，共识层的 AURA 引擎和 Grandpa 引擎等，可以任由开发者选择；在应用功能方面，Substrate 允许开发者调用官方已开发妥当的模块 pallet 为他们的区块链添加自定义功能，例如保存并处理账号信息的 balances 模块，和管理智能合约的 contracts 模块；不仅如此，Substrate 也提供了这些模块的实现源代码，开发者可以自行下载并进行修改后引入区块链，实现功能的定制化。这一设计原则，赋予了 Substrate 极好的可扩展性，便于开发人员依据实际需要进行功能增删和优化改进等操作。

此外，Substrate 对多链结构的区块链网络提供了良好的支持。基于 Substrate 开发的 Polkadot 网络，引入了平行链和中继链的概念。中继链是波卡网络的核心，它

连接、验证平行链，并路由平行链之间的消息，实现了不同区块链之间的互操作性和共享安全性；而平行链是可随开发者目的不同而自定义的定制链，它通过 collator 打包交易信息，并提交给中继链进行有效性验证。平行链可以并发处理交易，提高网络的扩展性和效率。

综上所述，使用 Rust 编程语言，在 Substrate 区块链开发框架内对树状区块链进行重写，将至少可以获得以下以太坊开发平台难以提供的优势：

- 使用效率更高，平台兼容性更好的 WebAssembly 虚拟机，使树状区块链可以在诸如嵌入式设备这样性能一般的计算机上运行；
- 使用高性能、安全和并发性的 Rust 编程语言进行开发，可避免在开发过程中引入内存泄漏等不易察觉的缺陷，令树状区块链运行更快速稳定；
- 允许开发者使用官方提供的模块和自行实现的模块为区块链增添新功能，令自定义链上功能的过程更加简单方便；
- 提供了更好的多链支持，存在 Polkadot 的平行链、中继链等设计概念作为参考，有利于树状区块链多链结构的构建。

5.3 树状区块链实现简析

由于区域索引区块链是树状区块链的前身，故本节的实现分析将以区域索引区块链作为起点。

区域索引区块链在保留以太坊官方实现的单链结构和共识机制等结构的基础上，新增了支持快速检索地理位置相关数据的区域状态树和支持历史交易地理位置查询的账户位置树。账户位置树和区域状态树均为键值对结构，前者以时间为键，以 Geohash 编码的地理位置字符串为值，其根哈希存储在区块头中；后者以 Geohash 编码的地理位置字符串为键，以当前区块链管辖范围内的账户信息、交易信息及收据信息作为值，其根哈希存储在账户状态中。此外，区域索引区块链还为表示交易、收据和账户状态的数据结构添加了 Geohash 编码的地理位置字段，如图5-1所示。

树状区块链在在区域索引区块链上，使用 Geohash 编码进行层级划分，将原有的单链结构划分为形似字典树一般的树状结构。为实现该目的，根据链上存储的数据不同，区块链被分为保存所有分支区块和普通区块的完整树状结构的全区块链、向下

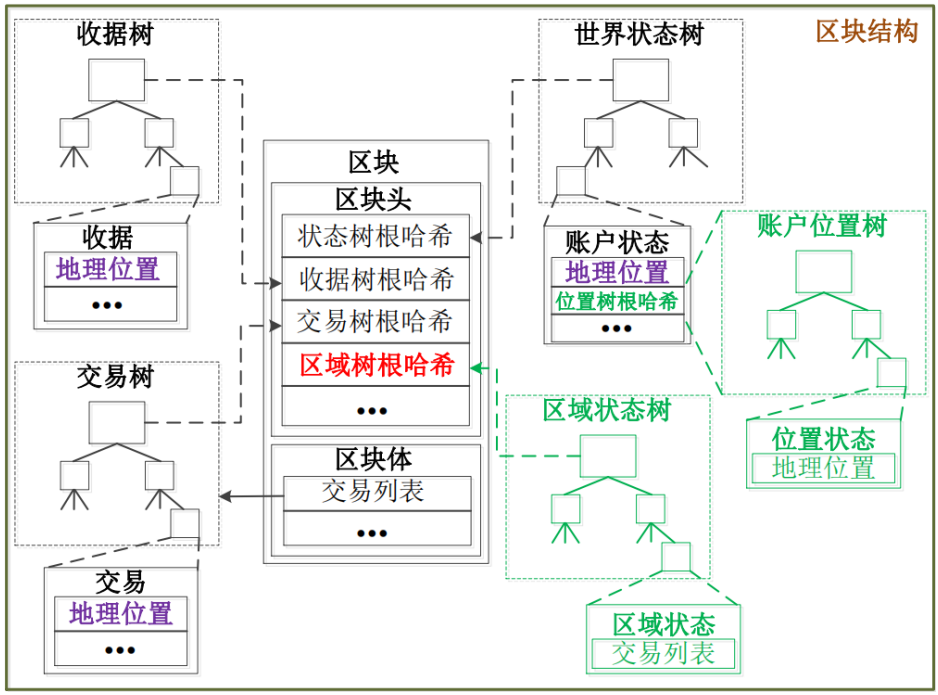

图 5-1 区域索引区块链区块结构

保存所有子区块并向上保存所有以自身为终点的父区块的分支区块链和去除分叉的叶子区块链三种，区块亦被分为三种类型：创世块、分支区块和普通区块。在图5-2所示的树状区块链区块结构中，“区块类型”字段将被用于区分上述三种类型的区块。除此之外，树状区块链的区块头中亦增添了链 ID 和平行链哈希字段。链 ID 用于消除各子链之间因区块序号相同引起的冲突，而平行链哈希为分支区块独有，其指向分支区块位于同一父分支区块下的其他兄弟分支区块，令基于地理位置的查询更加灵活高效。

树状区块链结构中，各个叶子区块链相互独立，其内部的区块数据和区域状态互不影响。为了维护分支区域和全区域内的区域状态数据，需要构建逻辑单链以实现完整区域内区块数据和区域状态的汇总。故，树状区块链提供了区域信息汇总的算法，以实现在上层分支区块链中分支节点对所有子链的汇总功能。同时，它也提供了跨子链转账的功能，并配套实现了所需的共识机制。

综上，将树状区块链使用 Rust 编程语言重写，大致需要经过以下步骤：

首先，在 Substrate 区块链开发平台上，实现区域索引区块链的功能特性。包括定位收据树、交易树，世界状态树等结构的实现代码，在其中加入合适的字段存储地理位置；设计区域状态树，并修改区块头结构以存储区域状态树根哈希。待这些

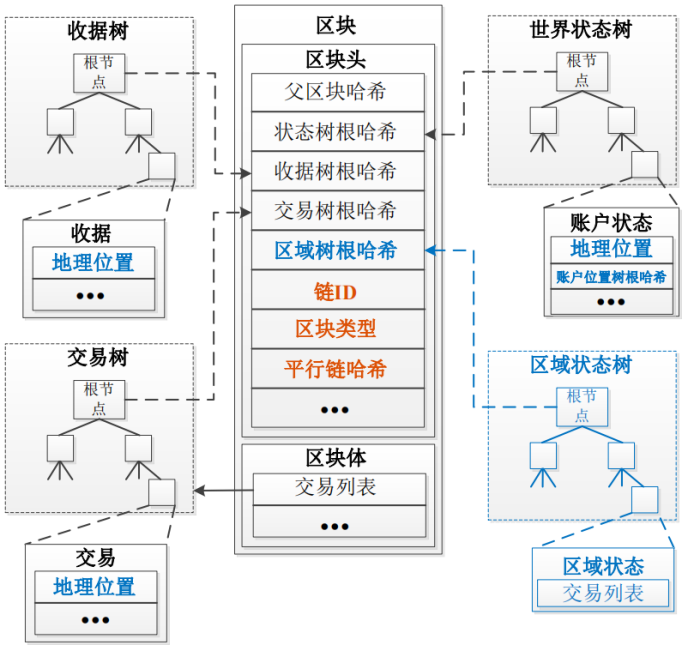

图 5-2 树状区块链区块结构

数据结构全部配置妥当之后，实现基于地理位置区域的快速查询算法。至该阶段完成，区域索引区块链应当已完全迁移至 Substrate 框架，表现出比 Substrate 官方实现的节点模板更好的性能。

在上述工作基础上，进一步引入树状区块链的功能特性，包括引入区别区块类型的字段，实现分支区块链汇总区域信息的算法和跨子链转账功能，以及该过程所需的对共识机制等的必要改动。

受项目时间所限，本文选择以账户状态作为切入点，为其引入地理位置字段，实现区域索引区块链的账户地理信息存储功能，以证明将树状区块链使用 Rust 重写的可行性及合理性。

5.4 Substrate 节点架构

由于本章工作尝试将树状区块链上的账户地理位置信息引入 Substrate 区块链框架中，故有必要对 Substrate 中的节点概念进行调研。

在一个去中心化的区块链网络中，每一个节点都同时充当了客户端和服务器的作用，因为它既可以从网络中请求数据，也可以向网络提供数据。Substrate 沿用了这一思想，并将其贯彻到了架构设计中。

如图 5-1 所示，Substrate 节点可以认为由两个部分组成：客户端外部节点以及

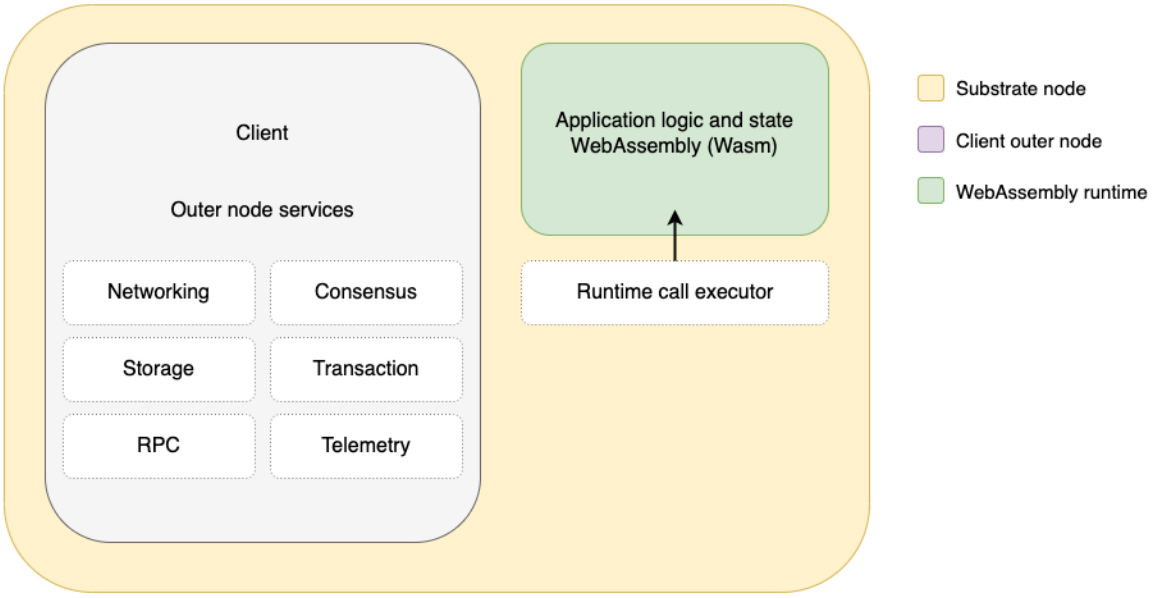

图 5-3 Substrate 节点架构简图

WebAssembly 运行时环境。

5.4.1 客户端外部节点

客户端外部节点负责运行时外部发生的活动。例如，外部节点负责发现对等节点、管理交易池、与其他节点通信以达成共识，以及响应来自外部的 RPC 调用或浏览器请求。

外层节点处理的一些最重要的活动包括以下几种：

- 存储: 使用简单高效的键值对存储层，保存 Substrate 区块链不断变化的状态；
- 点对点网络: 使用 libp2p 等方式与其他的网络参与者通信；
- 共识: 与其他网络参与者通信，确保他们对区块链的状态达成共识；
- RPC API：接收入站的 HTTP 和 WebSocket 请求，以便区块链用户与网络交互；
- 维护节点度量: 通过内嵌的 Prometheus 服务器收集并提供节点度量相关的信息；
- 执行环境: 为运行时选择要使用的执行环境（浏览器中的 WebAssembly 或本地的 Rust 环境）然后将调用分派给所选的环境。

执行这些任务通常需要外部节点查询运行时以获取信息或向运行时提供信息。这种通信通过调用专门的 runtime APIs 来处理。

5.4.2 WebAssembly 运行时环境

WebAssembly 运行时环境确定交易是否有效，并负责处理区块链的状态转换函数的更改。因为运行时能执行它接收到的函数，所以它可以控制如何将交易包含在区块中，以及如何将区块返回到外部节点以传播或导入到其他节点。本质上，运行时负责处理区块链上发生的所有事情，它也是构建 Substrate 区块链节点的核心组件。

与外部节点向运行时提供信息的方式类似，运行时使用专门的 host function 与外部通信。

5.5 Substrate 节点模板介绍

Substrate 官方提供了一份开源的节点模板[1]。节点模板中已包含较完整的运行时环境，且允许开发者自由增删模块，实现功能定制。此外，Substrate 也提供了教程文档[2]，辅助开发者学习使用节点模板。本节将演示在 Linux 环境下编译节点模板、与节点模板交互、以及为节点模板引入新模块的方法。

5.5.1 环境配置

本章的所有工作在表 5-1 所示的环境中进行。为提高开发效率，笔者继续使用第四章中搭建的 WSL 2 环境进行本章的工作。

表 5-1 Substrate 相关工作环境

中央处理器	Intel Core i5-12500H
图形处理器	Intel Iris Xe 80EU
内存	24GB
操作系统	Ubuntu 22.04.2 LTS
虚拟机	Windows Subsystem Linux 2

同时，使用节点模板要求安装 git、make、clang、curl 并配置 Rust 开发环境[3]。

[1]https://github.com/substrate-developer-hub/substrate-node-template

[2]https://docs.substrate.io/tutorials/

[3]https://docs.substrate.io/install/linux/

使用如下指令安装包含 git、make、clang、curl 工具包：

代码 5.1: 安装工具包

```
sudo apt install build-essential
```

使用如下指令安装 Rust 工具链：

代码 5.2: 安装 Rust 工具链

```
sudo apt install --assume-yes git clang curl libssl-dev protobuf-
    compiler
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
source \$HOME/.cargo/env  # 刷新环境变量

rustup default stable
rustup update  # 更新rust版本为最新的稳定版

# 安装rust nightly的2023-03-21前发布的最新版
rustup update nightly-2023-03-21
rustup target add wasm32-unknown-unknown --toolchain nightly
    -2023-03-21
```

5.5.2 使用节点模板进行开发

本小节主要讨论编译节点模板，及使用 Web UI 与其交互的方法。

下载节点模板源代码后，观察`runtime/Cargo.toml`。该文件记录了节点模板使用的运行时环境的各项配置。在`dependencies`字段下，写有该运行时包含的各项模块名，及其版本、源代码来源等信息。关于增添新模块和修改官方提供的模块的方法，请见下一小节的介绍。

在源代码根目录中打开终端，执行`cargo build --release`命令，稍事等待至编译完成后，在终端运行`./target/release/node-template --dev`，即能在开发模式下启动节点模板编译而成的节点。

使用 Polkadot JS APP 连接到该区块链[4]，在页面导航栏中依次选择“账户”-“账

[4]https://polkadot.js.org/apps/?rpc=ws%3A%2F%2F127.0.0.1%3A9944#/explorer

户”，既能看到节点中预定义的账户的各项信息。在导航栏中依次选择“开发者”-“交易”，既能利用 WebAssembly 运行时环境中引入的模块进行各项操作。如图 5-2 所示，选择 ALICE 账户后，依次选择 balances 模块，选择 transfer 方法，在 Id: AccoundId 处选择 FERDIE，value 处填写 10，随后点击“提交交易”-“签名并提交”，即可提交一个转账交易，将 10 单位代币从 ALICE 账号余额中转移到 FERDIE 账号余额中。

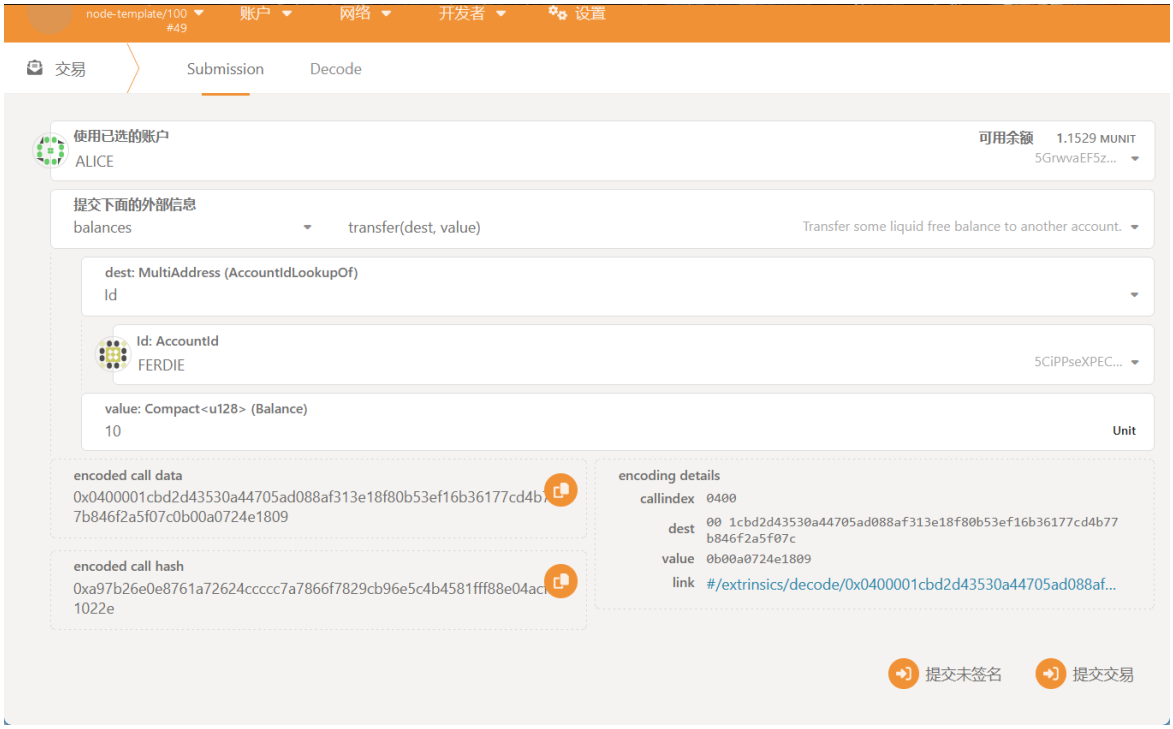

图 5-4 转账交易示例

5.6 为账户加入地理位置属性

树状区块链中，为记录每个账号所处的地理位置，向账号结构中加入了地理位置这一字段，以字符串形式存储账号的 Geohash 编码。本小结将演示如何在 Substrate 节点模板上实现该效果。

5.6.1 更改 balances 模块的引用源

模块 balances 负责记录账号信息，并提供转账等与账号信息有关的功能。为实现账户信息的增添，必须对 Substrate 官方实现的 balances 模块源代码进行一定修改。

访问 substrate 代码仓库[5]，切换至`polkadot-0.9.40`分支以和节点模板使用的模块所处的分支一致。下载该分支的源代码并将`frame/balances`目录复制到节点模板的`pallets`目录中。

由于下载获得的 balances 模块源代码使用相对路径引用其他模块，但节点模板目录中并不存在这些路径，故需要将`pallets/balances/Cargo.toml`配置文件中对其他模块的引用修改为从 Github 上直接引用。举例而言，对于该相对路径引用：

```
sp-std = { version = "5.0.0", default-features = false, path = "../../primitives/std" }
```

修改为从 Github 上直接引用：

```
sp-std = { version = "5.0.0", default-features = false, git = "https://github.com/paritytech/substrate.git", branch = "polkadot-v0.9.40" }
```

将所有模块的引用源更改完毕后，保存并关闭`pallets/balances/Cargo.toml`，打开`runtime/Cargo.toml`，将其中对模块`pallet_balances`的引用从 Github 上引用改为从本地的`pallets/balances`路径引用，修改方法与前文之所述类似，此处不再赘述。

最后，修改节点模板根目录中的`Cargo.toml`：

```
[workspace]
members = [
    "node",
    "pallets/template",
    "pallets/balances",  # <-- 新增这行
    "runtime",
]
[profile.release]
panic = "unwind"
```

保存并关闭即可。

[5]https://github.com/Endericedragon/substrate/tree/polkadot-v0.9.40

5.6.2 修改 balances 模块的源代码

定义“账户”这一概念的结构体是位于 pallets/balances/src/lib.rs 源代码的 AccountData 结构体。笔者在这个结构体中，加入了一个 position 字段，其类型为笔者自定义的 GeoHash 类型。增添的代码如下：

代码 5.3: 为 balances 模块新增代码

```
1    const GEOHASH_LENGTH: usize = 14;
2
3    #[derive(Encode, Decode, Clone, PartialEq, Eq, RuntimeDebug,
     MaxEncodedLen, TypeInfo)]
4    pub struct Geohash([u8; GEOHASH_LENGTH]);
5    impl Default for Geohash {
6        fn default() -> Self {
7            Geohash([0; GEOHASH_LENGTH])
8        }
9    }
10   impl Geohash {
11       pub fn from(geohash: Vec<u8>) -> Self {
12           assert!(geohash.len() <= GEOHASH_LENGTH);
13           let mut position: [u8; GEOHASH_LENGTH] = [0;
     GEOHASH_LENGTH];
14           for (i, cc) in geohash.iter().enumerate() {
15               position[i] = *cc;
16           }
17           Geohash(position)
18       }
19   }
20
21   // -- snip --
22
```

```
    pub struct AccountData<Balance> {
        // --snip --

        // The position of this account, encoded in geohash
        pub position: Geohash,
    }
```

每个模块均可包含一些方法，开发者可调用这些方法修改链上存储的数据，与模块交互。本小节为 balances 模块新增两个方法：set_position()方法和transfer_with_position()方法。

首先，为 balances 模块新增一个set_position()方法，允许账户为自己设置地理位置：

```
    #[pallet::call]  // <-- 在pallets/balances/src/lib.rs中全局搜索这行即可找到
    impl<T: Config<I>, I: 'static> Pallet<T, I> {
        // -- snip --

        // Set position for oneself
        #[pallet::call_index(6)]
        #[pallet::weight(0)]
        pub fn set_position(origin: OriginFor<T>, new_position: Vec<u8>) -> DispatchResult {
            let sender = ensure_signed(origin)?;
            Self::try_mutate_account(
                &sender,
                |target, _| {
                    target.position = Geohash::from(new_position);
                    Ok(())
                }
            )
```

```
        }
    }
```

其次，添加一个transfer_with_position()方法，令转账发起人提供自身位置信息，用以更新它的position字段：

```
    #[pallet::call_index(7)]
    #[pallet::weight(T::WeightInfo::transfer())]
    pub fn transfer_with_position(
        origin: OriginFor<T>,
        dest: AccountIdLookupOf<T>,
        #[pallet::compact] value: T::Balance,
        position: Vec<u8>
    ) -> DispatchResultWithPostInfo {
        let transactor = ensure_signed(origin)?;
        let dest = T::Lookup::lookup(dest)?;
        <Self as Currency<_>>::transfer(
            &transactor,
            &dest,
            value,
            ExistenceRequirement::AllowDeath,
        )?;
        Self::try_mutate_account(
            &transactor,
            |target, _| {
                target.position = Geohash::from(position);
                Ok(().into())
            }
        )
    }
```

至此，对代码的修改暂告一段落。以上修改已形成文档[6]，以便后人阅读参考。

5.6.3 编译并测试修改结果

在节点模板根目录中运行`cargo build --release`命令进行编译，确认编译过程无任何错误并结束后，使用 5.3 节的方法启动测试链并将 Web UI 连接至测试链上。在导航栏上点击“开发者”-“交易”，选择 ALICE 账号并按图 5-3 所示填写交易的各项参数，以调用 balances 模块的`set_position()`方法。提交设置位置的交易图中的 wx4111b 即为目标账户新地理位置的 Geohash 编码。需要注意的是，5.3.2 小节的代码中设置了该字段的限长`GEOHASH_LENGTH`为 14，故提交的字符串长度必须小于该限长。

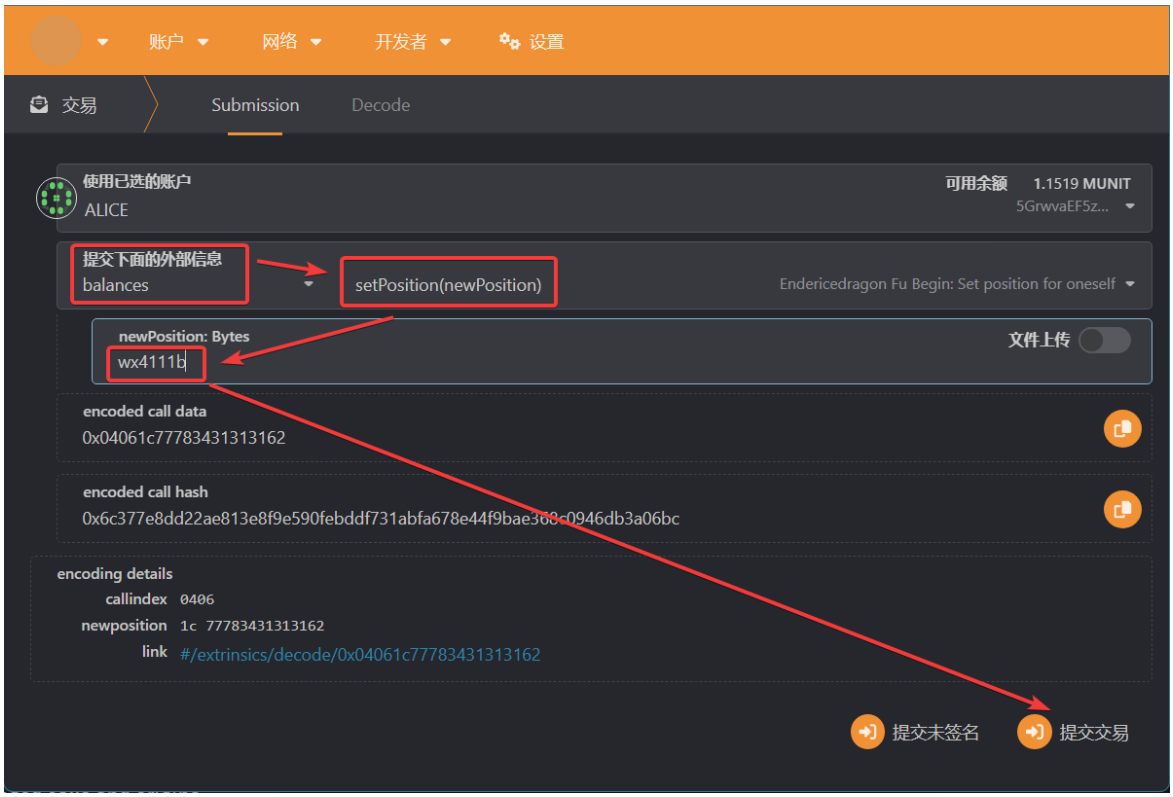

图 5-5 发起设置位置交易

待签名并提交该交易后，点击导航栏上的“开发者”-“链状态”，按图 5-4 所示填写查询信息。点击界面右侧的加号，查询结果将立刻显示在界面下方，可以看到

[6]https://github.com/Endericedragon/substrate-node-template/blob/main/%E4%BF%AE%E6%94%B9%E8%AE%B0%E5%BD%95.md

账户信息中出现了 position 字段，其值为 0x77783431313162000000000000000。此即为提交的字符串的 16 进制编码，按照两位 16 进制编码对应一个 ASCII 字符的规则进行转码后，还原其记录的字符串信息即为wx4111b，与上一步骤中提交的交易参数一致。

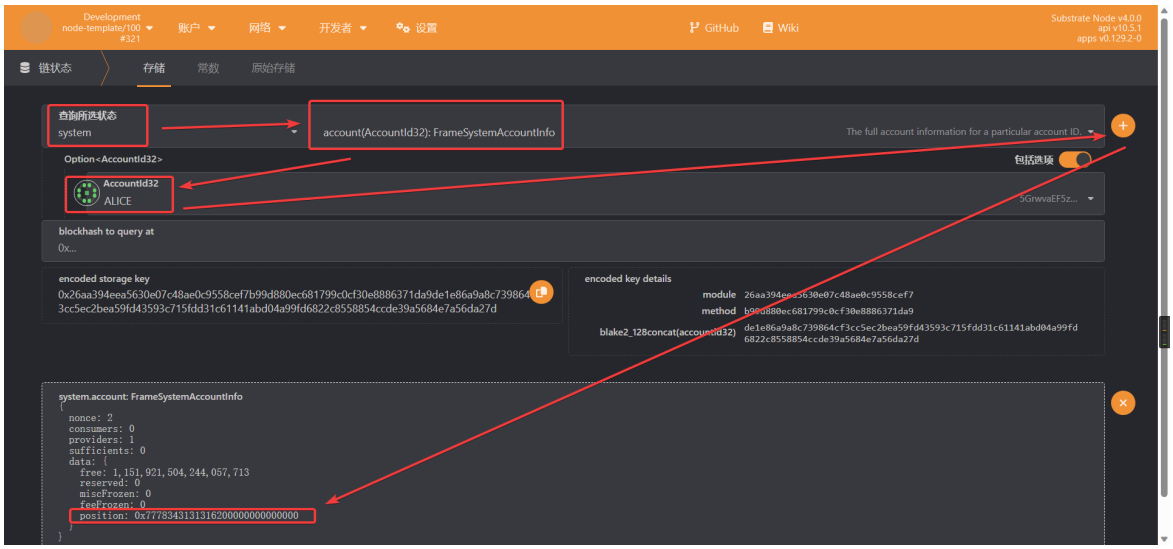

图 5-6 查询账户信息

使用相似的方法，调用 balances 模块的transfer_with_position()方法，令账号 ALICE 在 wx4111c 位置处发起转账，如图 5-5 所示。

随后，查询转账发起人 ALICE 的位置信息，获得图 5-6 所示的输出。其中，position 字段的值 0x77783431313163 即为 ALICE 的新位置 Geohash 编码 wx4111c 的 16 进制表示。

至此，账户信息中已包含地理位置字段。通过完成树状区块链部分特性在 Substrate 框架中的实现，本节工作证明了 Substrate 具有优秀的可扩展性，也为树状区块链的改进工作指明了方向。

5.7 本章小结

本章首先比较了 Substrate 相较以太坊的优势并介绍了 Substrate 的特点，阐述了树状区块链的改进思路，即将其从以太坊平台迁移至扩展性更强，综合性能更佳的 Substrate 开发框架。其次，介绍了 Substrate 的节点架构，其可分为客户端外部节点

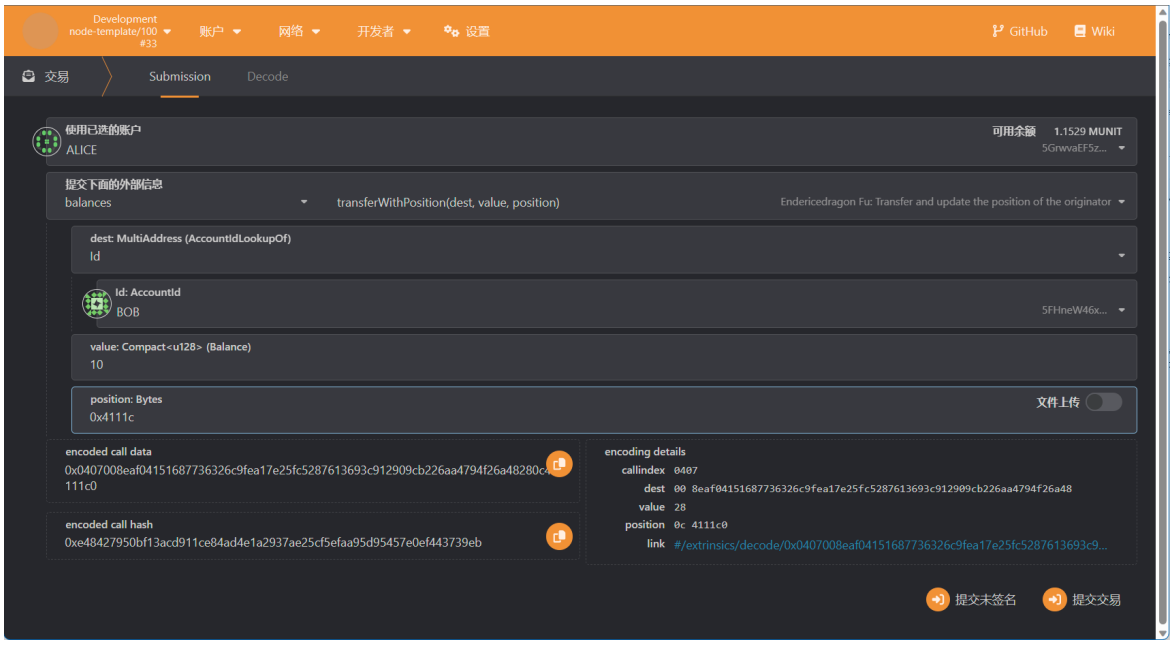

图 5-7 调用带位置信息的转账方法

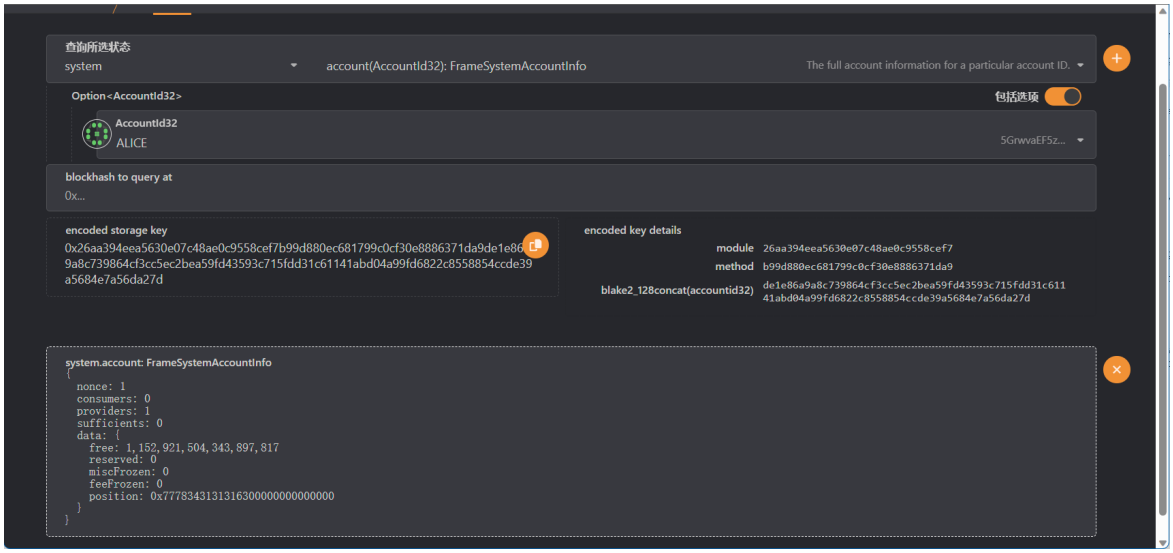

图 5-8 转账发起人位置已更新

和 WebAssembly 运行时环境两大板块，分别负责不同类型的事务处理。最后，介绍了使用节点模板进行运行时开发的方法，并成功将树状区块链的部分功能特性迁移到节点模板上，证明了 Substrate 具有强大的可扩展性，未来可以作为理想的树状区块链所依托的基础环境。

结　论

区块链技术以其透明性、去中心化、不可篡改等特点，在众多领域取得了瞩目的成就。然而，传统区块链的单链结构面对地理信息敏感、区块数量巨大的应用场景，无法提供令人满意的性能表现。针对该问题，实验室提出了树状区块链作为一种可行的解决办法，其采用树状结构组织多条子链，打破了单链结构的桎梏，辅之多种支持快速查询的数据结构，大大增强了区块链技术在上述几种应用场景中的适用性。

本文以基于区块链的出租车调度系统作为应用背景，针对树状区块链进行了相关调研、测试与部分重写工作。

首先，在树状区块链的前身——区域索引区块链上，复现了出租车调度系统相关工作，证明了该系统的可用性，同时指出并修复了实验室原有工作的疏漏，最后将以上工作归纳为详尽的手册，为树状区块链上运行出租车调度系统的测试奠定了基础。

其次，本文介绍了树状区块链的跨子链转账功能，详细解释了提出该功能的动机、该功能的作用和代价，接着设计了不同规模的测试，考察了树状区块链执行跨子链转账功能在不同负载情况下的执行效率，最后对测试结果进行分析、可视化，并建立简单的数学模型，为开发者在不同场景下选择不同区块链实现提供了一些建议。

在上述工作基础上，本文设计并进行了基于区块链的出租车调度系统在不同结构区块链上运行的对比测试。在确定了测试所使用的数据集、以及测试参与方与链上合约组成的调度系统的交互流程和方法后，进行数据准备并随后开展测试。测试结束后，以乘客端视角和司机端视角，对测试数据进行处理、汇总和可视化，针对树状区块链和区域索引区块链的长处短板进行分析，最后完成了补充实验的设计及进行，证明分析得出的猜想。

最后，本文介绍了使用 Rust 重写树状区块链的合理性，并移植了部分树状区块链的特性到基于 Rust 实现的区块链框架——Substrate 中，证明了该重写方案的可行性。通过分析以太坊的一系列不足之处，笔者强调了继续在以太坊平台上进行树状区块链开发的局限性，同时将 Substrate 区块链开发框架与其对比，介绍其突出优势，阐明了使用 Rust 重写树状区块链这一迁移方案的合理性。随后，本文对树状区块链当前在 Golang 上的实现进行了分析，评估了完成该迁移工作的大致工作量及大致工

作划分；为证明该移植方案的确实存在可行性，本文选择了账户地理位置信息这一树状区块链特性，添加到 Substrate 的官方节点模板中，并成功进行了验证性演示。

虽然本文基本完成了有关树状区块链的系统性测试，提出了使用 Rust 编程语言重写树状区块链的可行方案并进行了部分验证，但笔者认为该项目仍有可拓展的研究内容，例如：

- 第三章中，基于测试结果建立的模型较为简单；在进行动态查询复杂度分析时，仅考察了深度为 2 的树状区块链网络。若能针对更复杂的网络拓扑结进行分析，构建的数学模型将更能贴近树状区块链在实际场景中的工作情况；
- 第四章中，四条子链并行运行所获得的结果并不理想。虽然笔者提出了测试计算机的性能上限影响了树状区块链的性能发挥的猜想，并设计补充实验加以验证，但若能在性能更好的测试平台上进行该实验，获得的数据才能更有力地证明这一猜想；
- 第五章中，已针对账户地理位置信息完成了一部分树状区块链的特性移植。可以继续进行特性移植，将树状区块链的完整特性移植到 Substrate 平台，完善该项工作。

参考文献

[1] 蔡晓晴, 邓尧, 张亮, 等. 区块链原理及其核心技术[J]. 计算机学报, 2021, 44(1): 84-131.
[2] 马梅若. 数研所推出贸易金融区块链平台提升贸易融资效率助力经济快速发展[EB/OL]. 中国金融新闻网. (2020-09-10) [2023-04-24]. https://www.financialnews.com.cn/if/if/202009/t20200910_200546.html.
[3] Madhura K, Mahalakshmi R. Usage of block chain in real estate business for transparency and improved security[C]//. 2022: 1-10.
[4] Campanile L, Iacono M, Marulli F, et al. Designing a GDPR compliant blockchain-based IoV distributed information tracking system[J]. Information Processing & Management, 2021, 58(3): 102511.
[5] Zhou C, Lu H, Xiang Y, et al. Geohash-Based Rapid Query Method of Regional Transactions in Blockchain for Internet of Vehicles[J]. Sensors [Online]. Available: https://doi. org/10. 3390/s22228885, 2022.
[6] Xia Z, Wu J, Wu L, et al. A Comprehensive Survey of the Key Technologies and Challenges Surrounding Vehicular Ad Hoc Networks[J]. ACM Trans. Intell. Syst. Technol., 2021, 12(4).
[7] Aung N, Kechadi T, Zhu T, et al. Blockchain Application on the Internet of Vehicles (IoV)[C]//2022 IEEE 7th International Conference on Intelligent Transportation Engineering (ICITE). IEEE, 2022.
[8] 杨柳青. 区块链技术在车联网中的应用研究[J]. 中国新通信, 2018, 20(6): 110.
[9] G N. Geohash tips&tricks[EB/OL]. 2013 [2023-05-17]. http://Geohash.org/site/tips.html.
[10] Buterin V. 以太坊白皮书[EB/OL]. 2014 [2023-05-02]. https://ethereum.org/zh/whitepaper/.
[11] Technologies P. Home | Substrate[EB/OL]. 2023 [2023-05-02]. https://substrate.io/.
[12] Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System[EB/OL]. 2008 [2023-05-02]. https://assets.pubpub.org/d8wct41f/31611263538139.pdf.
[13] Lashkari B, Musilek P. A Comprehensive Review of Blockchain Consensus Mechanisms[J]. IEEE Access, 2021, 9: 43620-43652.
[14] 孙士奇. 区块链技术的发展及应用[J]. 信息系统工程, 2018, 88(10): 85-86.
[15] 朱岩, 王巧石, 秦博涵, 等. 区块链技术及其研究进展[J]. 工程科学学报, 2019, 41(190326-0004): 1361.
[16] Szabo N. Smart Contracts: Building Blocks for Digital Markets[EB/OL]. 1996 [2023-04-24]. https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html.
[17] Ma F, Ren M, Fu Y, et al. Security reinforcement for Ethereum virtual machine[J]. Information Processing & Management, 2021, 58(4): 102565.
[18] O'Reilly T. What is Web 2.0?[M]. 美国: O'Reilly Media, 2009.
[19] Hendler J. Web 3.0 Emerging[J]. Computer, 2009, 42(1): 111-113.
[20] Teams R. Rust 程序设计语言[EB/OL]. 2009 [2023-05-18]. https://www.rust-lang.org/zh-CN/.
[21] Go-ethereum Authors T. JavaScript Console[EB/OL]. 2023 [2023-05-02]. https://geth.ethereum.org/docs/interacting-with-geth/javascript-console.
[22] 庄海燕. 基于私有区块链的共识算法性能分析[J]. 滨州学院学报, 2020(02): 63-68.
[23] Technologies P. Substrate Documentation[EB/OL]. 2023 [2023-05-02]. https://docs.substrate.io/.
[24] Rouhani S, Deters R. Performance analysis of ethereum transactions in private blockchain[C]//

2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS). 2017: 70-74.
[25] Borgsmüller N. The Rust programming language for embedded software development[D]. Technische Hochschule Ingolstadt, 2021.
[26] Haas A, Rossberg A, Schuff D L, et al. Bringing the Web up to Speed with WebAssembly[J]. SIGPLAN Not., 2017, 52(6): 185-200.

附　录

附录 A 创世配置文件

代码 A.4: 创世配置文件

```
{
    "config": {
        "chainId": 666,
        "homesteadBlock": 0,
        "eip150Block": 0,
        "eip150Hash": "0
x0000000000000000000000000000000000000000000000000000000000000000",
        "eip155Block": 0,
        "eip158Block": 0,
        "byzantiumBlock": 0,
        "constantinopleBlock": 0,
        "petersburgBlock": 0,
        "istanbulBlock": 0,
        "ethash": {}
    },
    "nonce": "0x0",
    "timestamp": "0x5ddf8f3e",
    "extraData": "0
x0000000000000000000000000000000000000000000000000000000000000000",
    "gasLimit": "0xffffffff",
    "difficulty": "0x20000",
    "mixHash": "0
x0000000000000000000000000000000000000000000000000000000000000000",
    "coinbase": "0x0000000000000000000000000000000000000000",
```

```
22        "alloc": {},
23        "number": "0x0",
24        "gasUsed": "0x0",
25        "parentHash": "0
    x0000000000000000000000000000000000000000000000000000000000000000"
26    }
27
```

一些关键字段的解释如下：

- gasLimit：区块链为防止恶意参与者不停发送交易耗尽服务器资源，往往都对交易进行“收费”。gasLimit 字段限制一次交易的最大花费。为保证实验成功，故此处设置得较大。
- difficulty：挖矿难度。难度越低，越容易挖到符合要求的新区块，出块速度也越高。
- alloc：记录链上部分账户的余额等信息。由于该创世配置文件乃是为尚未准备账户的全新区块链所准备，故此项为一个空白的 JavaScript 对象。

附录 B 合约部署的代码模板

代码 B.5: 合约部署代码模板

```
1     abi = JSON.parse("经过压缩转义后的ABI")
2     bytecode = "获得的字节码字符串"
3
4     Contract = web3.eth.contract(abi);
5     web3.eth.estimateGas({data: bytecode})
6     contractInstance = Contract.new({
7         from: web3.eth.accounts[0],
8         data: bytecode,
9         gas: '3000000',
10        position:"w2511111111111",
```

```
        txtime:277001
    },function (e, contract){
        console.log(e, contract);
        if(!e){
            if(!contract.address) {
                console.log("Contract transaction send:
TransactionHash: " + contract.transactionHash + " waiting to be
mined...");
            } else {
                console.log("Contract mined! Address: " + contract.
address);
                console.log(contract);
            }
        }
 });

```

附录 C 跨链转账测试的数据可视化代码

代码 C.6: 跨链转账测试的数据可视化

```
import matplotlib.pyplot as plt
from matplotlib.axes._axes import Axes
from matplotlib.figure import Figure
import os

plt.style.use('_mpl-gallery')

dir_prefix: str = input("Directory name: ")

candidates: list[str] = list()
```

```
    for each in os.listdir():
        if os.path.isdir(each) and each.startswith(dir_prefix):
            candidates.append(each)

    if len(candidates) > 1:
        print("Choose one:")
        for (i, each) in enumerate(candidates):
            print(f"({i}) {each}")

        choice: int = int(input("Choice: "))
        dir_name: str = candidates[choice]
    else:
        dir_name: str = candidates[0]

    # make data
    with open(os.path.join(dir_name, "tx_request_w12.txt"), "r",
encoding="utf-8") as f:
        start_time = int(f.readline().split('\t')[1])

    times: list[int] = [start_time]
    with open(os.path.join(dir_name, "tx_result_w11.txt"), "r",
encoding="utf-8") as f:
        times += [int(each.split('\t')[1]) for each in f.readlines()]
    times.sort()

    y: list[int] = list(filter(lambda x: x != 0 and x < 55, [times[i]
- times[i - 1] for i in range(1, len(times))]))
    x: range = range(len(y))

```

```
    # plot
    fig, ax = plt.subplots()
    fig: Figure
    ax: Axes

    stem = ax.stem(x, y)

    for (i, j) in zip(x, y):
        plt.text(i, j + 0.3, str(j))

    print(f"total time consume = {times[-1] - times[0]}")

    print(f"fastest = {min(y)}")
    print(f"variance = {max(y)}")

    average: float = sum(y) / len(y)
    print(f"average = {average:.04f}")

    variance: float = sum((average - each) ** 2 for each in y) / len(y
)
    print(f"variance = {variance:.04f}")

    plt.show()


```