Санкт-Петербургский государственный электротехнический университет

“ЛЭТИ” им. В. И. Ульянова (Ленина)

(СПбГЭТУ “ЛЭТИ”)

Направление подготовки: 09.03.01 – “Информатика и вычислительная техника”
Профиль: Организация и программирование вычислительных и информационных систем

Факультет компьютерных технологий и информатики
Кафедра вычислительной техники

К защите допустить:
Заведующий кафедрой
д. т. н., профессор _____ М. С. Куприянов

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Тема: “Масштабируемые распределенные структуры данных и примитивы синхронизации”

Студент _____ Д. А. Королев

Руководитель
к. т. н., доцент _____ А. А. Пазников

Консультант по разработке
и стандартизации программных
средств к. э. н., доцент _____ М. А. Косухина

Консультант от кафедры
к. т. н., доцент, с. н. с. _____ И. С. Зуев

Санкт-Петербург
2023 г.

Санкт-Петербургский государственный электротехнический университет
“ЛЭТИ” им. В. И. Ульянова (Ленина)
(СПбГЭТУ “ЛЭТИ”)

Направление: 09.03.01 – “Информатика и вычислительная техника”
Профиль Организация и программирование вычислительных и информационных систем
Факультет компьютерных технологий и информатики
Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой ВТ
д. т. н., профессор
(М. С. Куприянов)
“___” _____ 2023г.

ЗАДАНИЕ
на выпускную квалификационную работу

Студент Д. А. Королев Группа № **9306**

1.Тема. Масштабируемые распределенные структуры данных и Примитивы синхронизации
(утверждена приказом № _____ от _____)

Место выполнения ВКР: СПбГЭТУ «ЛЭТИ»

2.Объект исследования: моделирование инвестиционного портфеля и расчет рисков.
3.Цель: разработка распределенной системы для моделирования оптимального инвестиционного портфеля и расчета рисков.
4.Исходные данные: техническое задание, документация используемых библиотек и инструментов.
5.Содержание:
- Распределенные структуры данных в инвестиционных задачах.
- Разработка и реализация распределенной системы.
- Экономическое обоснование разработки распределенной системы.

6.Технические требования:
- Система должна иметь возможность получить данные о состоянии активов.
- Система должна иметь возможность получать данные о доступных к торговле активах.
- Система должна поддерживать один из методов моделирования оптимального инвестиционного портфеля.
- Система должна обеспечивать возможность расчета рисков на основе имеющихся данных
- Система должна иметь возможность работать в распределенной среде.

- Система должна обеспечивать согласованность и целостность данных.

7.Дополнительные разделы: Разработка и стандартизация программных средств.

8.Результаты:

- Выбраны технологии и язык реализации распределенной системы.
- Разработаны архитектура и структура распределенной системы.
- Спроектированы и реализованы на языке Java модули распределенной системы.
- Реализованы алгоритмы и сущности, удовлетворяющие требованиям распределенной системы для моделирования оптимального инвестиционного портфеля.

Дата выдачи задания
«_____» _____ 2023 г.

Дата представления ВКР к защите
«_____» _____ 2023 г.

Руководитель
к. т. н., доцент _____ А. А. Пазников

Студент _____ Д. А. Королев

Санкт-Петербургский государственный электротехнический университет “ЛЭТИ” им. В. И. Ульянова (Ленина)
(СПбГЭТУ “ЛЭТИ”)

Направление: 09.03.01 “Информатика и вычислительная техника”
Профиль: Организация и программирование вычислительных и информационных систем
Факультет компьютерных технологий и информатики
Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой ВТ
д. т. н., профессор
(М. С. Куприянов)
“___” _____ 2023г.

КАЛЕНДАРНЫЙ ПЛАН

выполнения выпускной квалификационной работы

Тема. **Масштабируемые распределенные структуры данных и примитивы синхронизации**

(утверждена приказом № _____ от _____)

Студент Д. А. Королев Группа № **9306**

№ этапа	Наименование работ	Срок выполнения
1	Обзор технического задания	14.03 – 17.03
2	Составление диаграмм сущностей и блок-схем логики методов	17.03 – 20.03
3	Написание кода программы	20.03 – 15.04
4	Тестирование на корректность работы программы	15.04 – 20.04
5	Исправление недочетов в коде программы	20.04 – 5.05
6	Экономическое обоснование проекта	5.05 – 6.05
7	Оформление пояснительной записки	6.05 – 29.05

Руководитель
к. т. н., доцент _____ А. А. Пазников
Студент _____ Д. А. Королев

РЕФЕРАТ

Пояснительная записка содержит: … с., 6 рис., 3 табл., 1 приложение, 42 источника литературы.

Цель выпускной квалификационной работы: реализовать распределенную вычислительную систему, предназначенную для моделирования оптимального инвестиционного портфеля.

В выпускной квалификационной работе приводится обзор структуры распределенной системы, анализ проектируемых компонентов и их функций, реализация распределенной системы, тестирование и демонстрация работы получившейся системы.

Результаты работы: программное обеспечение, реализующее все необходимые функции для моделирования оптимального инвестиционного портфеля в распределенной среде.

Область применения результатов: грамотное управление финансовыми инструментами в финансовых организациях, банковских системах, и фондах. А также помощь частным инвесторам в управлении активами.

Выводы: в результате выполнения ВКР получено программное обеспечение, реализующее распределенную систему с использованием распределенных структур данных и примитивов синхронизации, для моделирования оптимального инвестиционного портфеля.

ABSTRACT

The aim of the graduation thesis is to implement a distributed computing system designed for modeling an optimal investment portfolio.

The thesis provides an overview of structure of distributed system, an analysis of the designed components and their functions, the implementation of the distributed system, and demonstration of the resulting system.

The result of the work includes software that implements all the necessary functions for modeling an optimal investment portfolio in distributed environment.

Application area of the results is effective management of financial instruments in financial organizations, banking systems, and funds. Additionally, providing assistance to individual investors in asset management.

In conclusion, the thesis resulted in software that implements a distributed system using distributed data structures and synchronization primitives for modeling an optimal investment portfolio.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

Инвестиционный портфель – совокупность финансовых активов, таких как акции, облигации, фонды и другие инструменты, в которые инвестор распределяет свои средства для достижения определенных инвестиционных целей.

Актив (финансовый инструмент) – финансовый инструмент, который может быть куплен, продан или торговаться на финансовом рынке. Примеры активов включают акции, облигации, товары, валюты и другие финансовые инструменты.

Оптимизация инвестиционного портфеля – процесс выбора оптимального сочетания активов в инвестиционном портфеле с целью достижения оптимального соотношения доходности и риска.

Оптимальный инвестиционный портфель – то сочетание активов в инвестиционном портфеле, которое предлагает наилучшее соотношение риска и доходности.

Риск – в контексте инвестиций, это возможность потери средств или непредвиденного отклонения от ожидаемых результатов.

API - Application Programing Interface, набор определенных правил и протоколов, которые позволяют программам взаимодействовать между собой.

ВВЕДЕНИЕ

В настоящее время распределенные системы и распределенные структуры данных стали неотъемлемой частью многих приложений, включая финансовые системы, обработку данных и машинное обучение. Одним из главных преимуществ использования распределенных систем является возможность масштабирования приложений на несколько узлов, что позволяет увеличить производительность и обеспечить более надежную работу.

В этом проекте будет рассматриваться проблема моделирования инвестиционного портфеля в распределенной системе. В этом контексте распределенные структуры данных и примитивы синхронизации играют важную роль в обеспечении эффективной и безопасной работы системы.

Актуальность темы обусловлена необходимостью разработки масштабируемых распределенных систем, которые могут эффективно обрабатывать большие объемы данных в режиме реального времени. Это особенно важно в финансовых системах, где высокая производительность и надежность являются критически важными. Разработка масштабируемой распределенной системы для моделирования инвестиционного портфеля может привести к улучшению точности прогнозирования, а также уменьшению затрат на управление рисками.

Целью проекта является разработка и анализ масштабируемой распределенной системы для моделирования оптимального инвестиционного портфеля. Основные задачи включают разработку распределенной структуры данных, обеспечение эффективной синхронизации данных между узлами системы, оптимизацию производительности системы и эффективное моделирование оптимального инвестиционного портфеля.

Однако, реализация таких систем сопряжена с рядом проблем и вызовов, которые нужно решить для обеспечения их эффективной работы. Одной из

таких проблем является управление распределенными структурами данных и примитивами синхронизации, которые используются для обмена данными между узлами и обеспечения координации выполнения задач.

Объектом данного проекта является распределенная вычислительная система для моделирования оптимального инвестиционного портфеля.

Предметом является язык программирования Java [1], подходящий для высоконагруженных систем.

В данном проекте особое внимание уделяется исследованию масштабируемости распределенных структур данных и примитивов синхронизации, а также моделированию инвестиционного портфеля в распределенной среде. Его цель состоит в том, чтобы исследовать различные подходы к управлению данными в распределенных системах и определить наиболее эффективные и масштабируемые методы для работы с инвестиционным портфелем.

Для достижения цели были поставлены следующие задачи:

- Провести обзор предметной области распределенных структур данных и примитивов синхронизации.
- Проанализировать и сравнить существующие методы управления данными в распределенных системах.
- Разработать алгоритм моделирования инвестиционного портфеля в распределенной среде.
- Реализовать разработанный алгоритм и провести экспериментальное исследования эффективности работы алгоритма на различном количестве узлов.
- Провести анализ полученных результатов и сделать выводы о применимости и эффективности предложенного подхода.

Данная работа имеет практическую значимость для области финансовых технологий, а также для области разработки и управления распределенными системами. Полученные результаты могут быть использованы для создания

новых распределенных систем, которые используются для обработки данных в режиме реального времени, а также для улучшения существующих систем, которые используются в финансовой индустрии.

Обзор распределенных структур данных и примитивов синхронизации, описание и сравнение доступных технологий приведены в первом разделе. Во втором разделе приведены требования к системе, выбор используемых технологий, разработка и реализация системы. В третьем разделе приводится экономическое обоснование выпускной работы. Заключение включает основные выводы по работе.

1 Распределенные структуры данных в инвестиционных задачах

Современные вычислительные системы часто используют распределенную архитектуру для решения сложных задач, которые требуют высокой производительности и масштабируемости. Распределенные системы состоят из нескольких компьютеров, которые работают в согласованном режиме, обмениваясь данными и ресурсами.

Одной из ключевых проблем в распределенных системах является управление данными. Для обеспечения масштабируемости и эффективности работы системы необходимо использовать распределенные структуры данных и примитивы синхронизации. Распределенные структуры данных позволяют эффективно хранить и обрабатывать большие объемы данных в распределенной среде. Примитивы синхронизации обеспечивают согласованность данных и их целостность в условиях конкуренции за ресурсы.

1.1 Распределенные структуры данных

Распределенная структура данных представляет собой совокупность логически связанных элементов данных, которые распределены по нескольким узлам в распределенной среде. Распределенные структуры данных используются для управления и обработки больших объемов информации в распределенных системах, где данные хранятся и обрабатываются на разных узлах в сети. Распределенная структура данных обеспечивает эффективный доступ к данным, возможность масштабирования системы и повышение надежности и доступности данных в распределенной среде. Она может быть реализована с помощью различных протоколов, таких как клиент-сервер, пиринговая сеть, протоколы совместного доступа к файлам и других.

Существует множество различных типов распределенных структур данных, каждая из которых имеет свои особенности и способы реализации и

предназначена для решения определенных задач. Некоторые из основных типов включают:

- Распределенные массивы [2]: данные хранятся на нескольких узлах и могут быть обработаны параллельно, что ускоряет вычисления. Это полезно, например, для анализа больших объемов данных.
- Распределенные хеш-таблицы [3]: данные хранятся на нескольких узлах и могут быть быстро найдены по ключу. Они используются, например, для кеширования данных и поиска информации.
- Распределенные очереди [4]: данные хранятся в очереди на нескольких узлах, и могут быть обработаны по мере их поступления. Это полезно, например, для обработки потоков данных.
- Распределенные графы [5]: данные представляются в виде графов, где узлы являются объектами, а связи между ними – отношениями. Они используются, например, для анализа социальных сетей и связей между объектами.

Инвестиционные компании могут использовать распределенные системы и структуры данных для улучшения производительности и надежности своих инвестиционных решений, а также для обеспечения масштабируемости своих приложений. Следует разобраться, для чего могут быть использованы распределенные структуры данных в инвестиционных задачах.

Распределенные структуры данных могут быть использованы в различных инвестиционных задачах, таких как:

- Анализ больших объемов финансовых данных: с помощью распределенных массивов можно анализировать большие объемы финансовых данных, таких как цены акций и объемы торгов.
- Распределенное хранение и обработка портфелей: с помощью распределенных хеш-таблиц можно хранить и обрабатывать портфели

инвесторов, что позволяет быстро получать информацию о состоянии портфеля и принимать инвестиционные решения.

- Обработка потоков данных: с помощью распределенных очередей можно обрабатывать потоки финансовых данных в режиме реального времени, что позволяет оперативно реагировать на изменения на рынке и принимать соответствующие решения.

1.1.1 Способы организации распределенных структур данных

В распределенных системах узлы общаются между собой с помощью различных технологий и протоколов, в зависимости от требований и характеристик системы. Ниже приведено несколько популярных технологий:

- Обмен сообщениями [6] – это один из наиболее распространенных способов коммуникации между узлами в распределенной системе. Узлы отправляют друг другу сообщения через определенные протоколы и каналы связи.
- Remote Procedure Call (RPC) [7] – эта технология позволяет узлам вызывать удаленные процедуры и функции на других узлах в сети. Узлы обмениваются запросами и ответами для выполнения операций и передачи данных.
- Использование RESTful API [8] – распределенные системы могут использовать RESTful API (Representational State Transfer) для обмена данными между узлами. Это основано на принципах HTTP-протокола, где узлы могут отправлять HTTP-запросы для получения или изменения данных на других узлах.
- Peer-to-Peer (P2P) коммуникация [9] – в некоторых случаях узлы могут использовать прямое соединение друг с другом, обмениваясь данными напрямую без централизованного посредника.

- Информационные агенты [10] – могут быть использованы для обмена информацией и координации между узлами. Они могут собирать, анализировать и распространять данные в системе, осуществляя связь и взаимодействие между узлами.

Конкретный выбор технологии коммуникации зависит от требований распределенной системы, ее характеристик и ожидаемых возможностей.

1.2 Примитивы синхронизации

Примитивы синхронизации – это механизмы, используемые для согласования доступа к общим ресурсам в многопоточной среде, чтобы избежать состояний гонки и других проблем, связанных с параллельным доступом к данным. Распределенные системы также требуют использования примитивов синхронизации для обеспечения согласованного доступа к общим ресурсам в разных узлах системы.

Основные типы примитивов синхронизации включают в себя:

- Мьютексы [11]: примитивы синхронизации, используемые для ограничения доступа к общим ресурсам только одним потоком в определенный момент времени.
- Семафоры [12]: примитивы синхронизации, позволяющие ограничить количество потоков, имеющих доступ к общим ресурсам, в зависимости от значения счетчика.
- Мониторы [13]: примитивы синхронизации, позволяющие гарантировать, что только один поток может одновременно выполнять критическую секцию кода.
- Блокировки [14]: примитивы синхронизации, позволяющие блокировать доступ к общим ресурсам в случае, если они уже используются другим потоком.

Примитивы синхронизации могут использоваться в распределенных системах для обеспечения согласованного доступа к общим ресурсам, таким как распределенные структуры данных и файловые системы. Например, мьютексы могут быть использованы для ограничения доступа к общим данным только одним узлом в определенный момент времени. Мониторы и блокировки могут использоваться для гарантирования, что только один узел имеет доступ к критической секции кода в определенный момент времени. В целом, применение примитивов синхронизации в распределенных системах позволяет обеспечить целостность и согласованность данных, а также управлять доступом к общим ресурсам для эффективной работы системы.

Существуют и более сложные примитивы синхронизации, например, атомарная рассылка [15]. Это механизм в распределенных системах, который обеспечивает доставку сообщений к участникам системы в определенном порядке. Он гарантирует, что все участники получат одинаковую последовательность сообщений и применят их в одном и том же порядке.

Атомарная рассылка обеспечивает согласованность данных в распределенной среде. Когда узлы системы совместно работают над общими данными, атомарная рассылка гарантирует, что изменения данных будут применены в одинаковом порядке на всех узлах. Это важно для поддержания целостности и согласованности данных в распределенной системе.

Технологии, такие как информационные агенты или система обмена сообщениями, могут быть использованы для реализации атомарной рассылки в распределенных системах. Они обеспечивают надежную доставку и применение сообщений между узлами системы с согласованным порядком. Это позволяет узлам синхронизировать свои операции и поддерживать согласованность данных в распределенной структуре.

Также стоит ввести понятие консенсуса в распределенных системах. Консенсус – это понятие, которое относится к проблеме достижения согласия среди участников распределенной системы относительно определенного

значения или результата. В распределенных системах, где участвуют несколько узлов, каждый узел может иметь свою собственную локальную информацию или взаимодействовать с внешней средой. Однако важно, чтобы все участники достигли единого мнения о некоторых важных аспектах системы.

Проблема консенсуса возникает в ситуациях, когда участники распределенной системы сталкиваются с неопределенностью, конфликтами или асинхронностью в своих мнениях или действиях. Цель состоит в том, чтобы достичь согласованности, даже при возможных отказах участников или сбоях в сети.

Протоколы консенсуса обычно включают механизмы для предложения значений, голосования, выбора лидера (координатора) и согласования решения среди участников системы. Цель состоит в том, чтобы достичь единого и согласованного решения, которое будет принято большинством участников или согласно определенным правилам.

Решение проблемы консенсуса имеет важное значение для распределенных систем, так как позволяет достичь единого мнения или результата, обеспечивает надежность и целостность данных, а также предоставляет гарантии согласованного выполнения операция в распределенной среде.

1.3 Алгоритмы оптимизации портфеля и расчета рисков

Конструирование и управление инвестиционным портфелем является сложной задачей, которая требует учета множества факторов, включая цель инвестирования, ожидаемый доход, риск и т. д. Для достижения наилучших результатов в управлении портфелем инвесторы часто используют алгоритмы оптимизации портфеля и методы моделирования рисков портфеля.

Алгоритмы оптимизации портфеля используются для выбора наилучшего набора активов в инвестиционном портфеле на основе заданных

ограничений, таких как минимальный и максимальный процент вложений в каждый актив, минимальный ожидаемый доход и максимальный риск портфеля. Эти алгоритмы помогают инвесторам создать портфель, который максимизирует доходы и минимизирует риски.

Методы моделирования рисков портфеля используются для оценки возможных потерь инвестиционного портфеля в будущем. Они могут помочь инвесторам понять, какие риски могут быть связаны с инвестированием в конкретные активы и как эти риски могут повлиять на доходность портфеля. Эти методы могут также помочь инвесторам принимать более обоснованные решения о распределении своих инвестиций и управлении рисками.

Общая цель использования алгоритмов оптимизации портфеля и методов моделирования рисков – это создание оптимального инвестиционного портфеля, который максимизируем доход и минимизирует риски на основе инвестиционных целей и ограничений инвестора.

1.3.1 Обзор методов оптимизации инвестиционного портфеля

Важной темой в инвестиционном управлении является правильный выбор оптимизационного алгоритма, поскольку правильный выбор оптимального портфеля может привести к максимизации ожидаемой доходности и минимизации риска. Существует множество методов оптимизации портфеля, каждый из которых может быть применен в разных ситуациях и с разными целями.

Одним из первых и наиболее известных методов оптимизации портфеля является метод Марковица [16], предложенный в 1952 году Гарри Марковицем. Суть метода заключается в поиске наилучшего соотношения доходности и риска портфеля, оптимизируя их совместно. Метод Марковица использует матрицу ковариаций активов для оценки рисков портфеля. Он определяет эффективный фронт – границу всех возможных комбинаций

портфелей с минимальным уровнем риска для заданной доходности или максимальной доходности для заданного уровня риска.

Еще один метод оптимизации портфеля – метод Монте-Карло [17]. Он основывается на моделировании случайных событий и их влиянии на цены активов. Метод заключается в том, что он создает несколько случайных портфелей, оценивает их доходности и риски, а затем выбирает наилучший портфель. Метод Монте-Карло может использоваться для оценки вероятностей выпадения определенных значений доходности и рисков портфеля.

Метод максимальной полезности [18] – это метод, основанный на теории ожидаемой полезности, разработанной в 1940-х годах. Этот метод учитывает предпочтения инвестора и целевую функцию, которую он хочет максимизировать. Он определяет оптимальный портфель, который удовлетворяет требованиям инвестора по доходности, риску и предпочтениям.

Основным преимуществом метода Марковица является то, что он учитывает ковариацию между активами, что позволяет снизить риски портфеля. Метод Монте-Карло позволяет получить множество возможных вариантов портфеля и их оценок, что может быть полезно для инвестора при принятии решений. Метод максимальной полезности позволяет учитывать индивидуальные предпочтения инвестора, что может быть полезно для определения оптимального портфеля.

Однако все эти методы имеют и недостатки. Метод Марковица, например, требует точного определения ковариации между активами, что может быть сложно и неточно. Кроме того, метод Марковица предполагает, что доходность активов имеет нормальное распределение, что может быть не всегда верным.

Метод Монте-Карло также имеет недостатки, такие как высокая вычислительная сложность, особенно при моделировании большого

количества активов. Кроме того, этот метод не учитывает изменения на рынке и не может обеспечить точную оценку риска и доходности портфеля.

Метод максимальной полезности может быть непригодным для инвесторов, не имеющих ясных целевых функций, или не знающих своих индивидуальных предпочтений. Кроме того, этот метод может не учитывать некоторые факторы, такие как налоги или ликвидность активов.

Все методы оптимизации портфеля имеют свои преимущества и недостатки, и их выбор зависит от индивидуальных потребностей и целей инвестора. Поэтому важно выбрать метод, который наилучшим образом соответствует требованиям инвестора и обеспечивает наилучший результат.

1.3.2 Обзор методов моделирования рисков инвестиционного портфеля

Конечная цель любого инвестора – получить прибыль, но не менее важно снизить риски, и моделирование рисков портфеля – это один из способов достижения этой цели. Наиболее распространенными методами моделирования рисков портфеля являются Value at Risk, Expected Shortfall и Conditional Value at Risk.

Value at Risk (VaR) [19] – это метод, который определяет максимальную возможную потерю портфеля на определенном уровне вероятности за заданный временной период. Он использует статистические методы для вычисления потенциальной потери портфеля, которая может возникнуть при неблагоприятных условиях. В качестве преимущества VaR можно назвать то, что он позволяет инвесторам оценивать потери, связанные с различными вариантами портфелей и принимать решения на основе этих данных. Однако, VaR не учитывает распределение потерь, которое может быть не нормальным, и не показывает вероятность того, что потери превысят VaR.

Expected Shortfall (ES) [20] – это метод, который измеряет среднюю потерю портфеля в случае, если VaR был превышен. Это означает, что ES

учитывает не только вероятности возникновения убытков, но и силу этих убытков. Он позволяет оценивать потери более точно, чем VaR, так как он учитывает не только вероятность возникновения потерь, но и их масштаб. Недостатком ES является то, что его расчет может быть сложнее, чем VaR.

Conditional Value at Risk (CVaR) [21] – это метод, который учитывает не только вероятность возникновения потерь, но и силу этих потерь. Он описывает среднюю потерю портфеля в тех случаях, когда потери превышают определенный порог. Этот метод обеспечивает более точную оценку рисков, чем VaR, и учитывает все потери, а не только те, которые находятся за порогом VaR. Преимуществом CVaR является его более точное описание рисков, чем VaR и ES. Недостатком CVaR является его вычислительная сложность.

В целом моделирование рисков является важным инструментом для инвесторов, которые хотят снизить риски и улучшить результативность своих портфелей. Каждый из трех методов – VaR, ES и CVaR – имеет свои преимущества и недостатки, и выбор конкретного метода зависит от целей инвестора, его инвестиционной стратегии и доступности данных для расчетов. Некоторые инвесторы могут использовать комбинацию различных методов, чтобы получить наиболее точное представление о рисках своих портфелей. Важно также помнить, что любое моделирование рисков связано с определенной степенью неопределенности и возможностью непредвиденных событий, поэтому инвесторы должны быть готовы к неожиданным изменениям и принимать соответствующие меры для снижения рисков.

2 Разработка и реализация распределенной системы

Разработка и реализация распределенной системы для моделирования оптимального инвестиционного портфеля и расчета рисков является важным шагом в создании эффективной инвестиционной стратегии. Цель этого раздела – определить требования к системе, необходимые для ее успешной разработки и реализации, а также разработка и реализация самой системы.

2.1 Определение требований к системе

В этом разделе будут определены требования к системе, чтобы обеспечить ее эффективную работу. Это позволит создать систему, которая сможет своевременно оценивать риски портфеля и помогать инвесторам принимать обоснованные решения на основе полученных результатов.

Для того, чтобы система моделирования оптимального инвестиционного портфеля и расчета рисков работала корректно и точно, необходимо определить, какие данные будут использоваться в процессе ее работы. Например, это могут быть данные о финансовых инструментах, такие как цены акций, курс валют, ставки процентов и т. д. Кроме того, могут потребоваться данные о биржевых индексах, торговых объемах, доходности инвестиций и прочие данные.

Также важно учесть требования к масштабируемости, надежности и производительности системы. Масштабируемость означает возможность системы поддерживать работу с большим количеством данных и пользователей. Надежность подразумевает, что система должна работать стабильно без сбоев, а также обладать механизмом защиты от ошибок. Производительность системы является важным фактором, который напрямую влияет на скорость работы и время отклика при выполнении задач.

Все действия относительно ценных бумаг инвестор совершает на бирже. Это специальная площадка, на которой люди заключают сделки и

обмениваются деньгами, товарами, ценными бумагами или производными контрактами. Напрямую человек не может взаимодействовать с биржей, для этого существуют брокеры – профессиональные участники рынка ценных бумаг, которые являются посредниками между инвесторами и биржей. В России существуют две биржи: Московская и Санкт-Петербургская. На каждой из них предоставляется доступ к разному числу активов. Самым популярным брокером в России, который предоставляет доступ к этим и многим другим зарубежным биржам, является «Тинькофф Инвестиции» [22]. При разработке системы будут использоваться данные об активах, торги которых проходят на Московской [23] и Санкт-Петербургской [24] биржах. Ограничение только российскими биржами обуславливается риском торговли на иностранных биржах ввиду множества экономических санкций, наложенных на Россию.

Определим следующие требования для разрабатываемой системы моделирования оптимального инвестиционного портфеля и расчета рисков:

- Система должна иметь возможность получить данные о доступных к торговле активах, расчет которых происходит в рублях.
- Система должна иметь возможность получать данные о состоянии активов, таких как акции, облигации, валюты и ценные металлы.
- Система должна поддерживать один из методов моделирования оптимального инвестиционного портфеля.
- Система должна обеспечивать возможность расчета рисков на основе имеющихся данных, используя один из методов расчета рисков.
- Система должна иметь возможность распараллеливать вычисления и работать в распределенной среде, чтобы обеспечить масштабируемость.
- Система должна обеспечивать согласованность и целостность данных между узлами.

- Система должна иметь гибкую архитектуру, позволяющую легко добавлять новые методы моделирования и алгоритмы расчета рисков.

2.2 Выбор технологий и инструментов

Существует множество инструментов для работы с распределенными вычислениями и синхронизацией данных, которые могут быть полезны при разработке и реализации системы моделирования оптимального инвестиционного портфеля и расчета рисков. Для реализации системы был выбран язык программирования Java ввиду своей эффективности при обработке больших объемов данных.

2.2.1 Выбор инструментов для реализации распределенной структуры данных

Для языка Java существует достаточное количество инструментов для работы с распределенными системами. Среди них:

- Apache Hadloop [25] – это фреймворк для распределенной обработки больших объемов данных. Он предоставляет инструменты для хранения, обработки и анализа данных на кластерах серверов. Hadloop использует MapReduce алгоритм для параллельной обработки данных, а также имеет свою собственную файловую систему HDFS, что обеспечивает высокую производительность.
- Apache Spark [26] – это фреймворк для обработки и анализа больших объемов данных. Spark также использует параллельную обработку данных, но в отличие от Hadloop, он работает в памяти, что обеспечивает высокую производительность.
- Apache Cassandra [27] – это распределенная база данных, которая обеспечивает высокую доступность и масштабируемость данных. Cassandra использует согласованную хеш-таблицу для хранения данных, что позволяет

обеспечить быстрый доступ к данным и предотвратить их потерю при отказах узлов.

- Apache Kafka [28]– это распределенная система обмена сообщениями, которая позволяет передавать потоковые данные между различными приложениями. Kafka обеспечивает надежную доставку сообщений, а также гарантирует сохранение данных при отказах узлов.
- Apache ZooKeeper [29]– это распределенная служба координации, которая обеспечивает надежность и согласованность в распределенных приложениях. ZooKeeper предоставляет механизмы управления доступом к данным и обеспечения консистентности при изменении состояния приложения.
- Hazelcast [30]– это инструмент для создания распределенных систем в Java. Hazelcast предоставляет механизмы для распределения данных и вычислений, кэширования, управления жизненным циклом объектов и т. д.

При разработке и реализации системы моделирования оптимального инвестиционного портфеля необходимо учитывать требования к масштабируемости, надежности и производительности. Для этого можно использовать распределенные системы, такие как Hadloop и Spark, которые обеспечивают высокую производительность и масштабируемость. Также необходимо учитывать требования к синхронизации данных и координации приложения, для чего могут быть использованы инструменты, такие как Kafka и ZooKeeper.

Для реализации распределенной системы была выбрана библиотека Hazelcast. Она предоставляет гибкую настройку распределенной структуры данных, а также множество удобных примитивов синхронизации.

2.2.2 Выбор инструментов для работы с финансовыми данными

Когда дело доходит до работы с финансовыми данными, моделирования портфеля и расчета рисков на языке Java, есть множество библиотек, которые

можно использовать. Некоторые из них являются платными, другие – бесплатными, и каждая библиотека предоставляет свои уникальные возможности и функции. Ниже перечислены некоторые из наиболее популярных библиотек для работы с финансовыми данными:

- Apache Commons Math [31] – библиотека, которая содержит математические функции, которые могут быть использованы в финансовых приложениях, таких как расчеты процентных ставок, моделирование портфелей, вычисление статистических значений и многое другое. Она также включает в себя реализации алгоритмов линейной алгебры, оптимизации, интерполяции и численного интегрирования.
- JQuantLib [32]– это порт библиотеки QuantLib [33] на Java. QuantLib – это библиотека на языке C++, которая используется для финансового моделирования и вычисления ценных бумаг. JQuantLib предоставляет возможность эту библиотеку на языке Java, а также добавляет некоторые дополнительные функции.
- Smile [34]– это Java библиотека, которая содержит реализации различных методов моделирования и анализа рисков. Она включает в себя алгоритмы для построения моделей временных рядов, оценки параметров моделей, симуляции и т. д.
- QuantTools [35]– это Java библиотека, которая содержит реализации алгоритмов для моделирования и анализа финансовых данных. Она включает в себя алгоритмы расчета стоимости опционов, оценки портфелей, анализа временных рядов и много другое.

Для работы с финансовыми данными было принято решение написать собственный модуль, реализующий все необходимые функции. Это сделано для того, чтобы меньше нагружать вычислительную систему сложными библиотеками, использующими собственные типы данных, не поддерживаемыми Hazelcast.

2.2.3 Выбор инструментов для получения данных об активах

При работе с финансовыми данными получение актуальной информации об активах является критически важным шагом. Однако, в зависимости от используемого источника данных, набор доступной информации и ее точность может существенно отличаться. Поэтому выбор подходящего API для получения данных об активах является важным этапом при разработке системы моделирования портфеля и расчета рисков. Ниже представлены некоторые из наиболее распространенных API для получения финансовых данных на языке Java:

- Yahoo Finance API [36]– обеспечивает доступ к данным о котировках акций, фондовых индексах и валютах.
- Alpha Vantage API [37]– предоставляет данные о котировках акций, фондовых индексах и криптовалютах.
- Кроме того, многие брокерские компании предоставляют свои API для получения данных о котировках акций, такие как: TD Ameritrade API, Interactive Brokers API, Tinkoff Invest API [38] и другие.

Перед использованием любого из этих API, необходимо внимательно изучить документацию и правила использования, чтобы убедиться в соответствии с авторскими правами, законами и политиками.

Для разрабатываемой системы был выбран API от брокера «Тинькофф Инвестиции»: Tinkoff Invest API. Это наилучший выбор, учитывающий изложенные требования к системе. Плюсом этого инструмента является возможность подключения своего профиля, путем чего можно реализовать алгоритм автоматической торговли

2.3 Разработка системы

Конструкция надежной и эффективной системы для моделирования портфеля и расчета рисков требует тщательного планирования,

проектирования и разработки. В этом разделе будут определены структура системы и ее компоненты, разработаны алгоритмы моделирования портфеля и расчета рисков, а также разработаны протоколы синхронизации данных и обработки ошибок, необходимые для достижения оптимальной производительности, масштабируемости и надежности системы. Важными элементами в этом процессе является настройка и интеграция в систему выбранных инструментов и библиотек.

2.3.1 Определение структуры системы и ее компонентов

В данном разделе будут описаны архитектура системы, ее компоненты и их взаимодействие. Кроме того, будут определены функциональные требования к каждому компоненту. Описав структуру системы и ее компонентов, можно будет перейти к разработке алгоритмов моделирования портфеля и расчета рисков, а также протоколов синхронизации данных и обработки ошибок.

Структура системы должна быть построена на основе классической клиент-серверной архитектуры, где клиентом выступает пользовательский интерфейс, а сервером – основная часть системы, отвечающая за обработку запросов пользователя и предоставление ему необходимой информации.

Клиент-серверная архитектура является широко распространенной в современных информационных системах. Она основана на разделении функций между клиентом (клиентским приложением) и сервером. Клиент предоставляет пользовательский интерфейс и отправляет пользовательские запросы, а сервер обеспечивает доступ к данным и выполняет вычисления.

В такой архитектуре клиент и сервер будут работать на разных устройствах и взаимодействовать между собой посредством сетевых протоколов. Это позволит достичь высокой масштабируемости, гибкости и надежности системы.

Сервер может представлять собой не одну вычислительную машину, а несколько. В такой связке каждая вычислительная машина будет являться узлом, на котором будет выполняться часть вычислений. В разрабатываемой системе должно быть минимум три вычислительных машины:

- Компьютер клиента, средствами которого будет осуществляться взаимодействие клиента с сервером. Его роль будет выполнять стационарный компьютер, подключенный к интернету.
- Центральный узел – часть серверного кластера, которая управляет общими данными, синхронизацией и постановкой задач остальным узлам. Его роль будет выполнять ноутбук, подключенный к интернету.
- Вычислительный узел – часть серверного кластера, выполняющая вычисления, требовательные к ресурсам. Его роль будет выполнять арендованный выделенный сервер.
- Дополнительными вычислительными мощностями могут выступать так же арендованные выделенные сервера. Они будут являться частью серверного кластера и выступать в роли вычислительных узлов.

Внутри серверного кластера должны находиться следующие компоненты системы:

- Компонент получения данных – отвечает за получение данных об активах через Tinkoff Invest API и сохранение этих данных в базу данных.
- Компонент моделирования портфеля – отвечает за создание портфеля и расчет его характеристик на основе имеющихся данных об активах.
- Компонент расчета рисков – отвечает за расчет рисков портфеля.
- Компонент оптимизации портфеля – отвечает за нахождение оптимального инвестиционного портфеля.
- Компонент синхронизации данных – отвечает за обновление данных об активах в базе данных и передачу этих данных между компонентами системы.

- Компонент хранения данных – две распределенные структуры. Первая хранит общие входные данные, вторая – результат оптимизации.

На рисунке 2.1 представлена схема взаимодействия компонентов системы между собой.

Рисунок 2.1 – Схема взаимодействия компонентов системы.

2.3.2 Разработка алгоритма получения данных

Получение данных об активах является важной задачей для системы моделирования оптимального инвестиционного портфеля и расчета рисков. В этом разделе будут описаны основные принципы алгоритма получения данных об активах с использованием Tinkoff Invest API.

Для начала разработки нужно изучить возможности Tinkoff Invest API. Данный набор средств разработки программного обеспечения предоставляет возможность получения доступных для торговли активов, исторических данных по выбранным активам, множество дополнительной информации по каждому запрашиваемому активу, включающее в себя расчетную валюту, страну регистрации, биржу, на которой проходят торги данного актива, и т. д.

Хранение больших объемов данных в финансовых системах связано с большим количеством дублирующейся информации. К сожалению, Tinkoff Invest API при определенных запросах передает и эти дубликаты. Значит необходимо реализовать дополнительную фильтрацию данных, получаемых через Tinkoff Invest API.

После очистки данных от пропусков и дубликатов необходимо привести все данные к единому виду. Для работы алгоритмов оптимизации инвестиционного портфеля данные об активах должны иметь равные временные интервалы. Например, были получены исторические данные по двум активам, у первого актива история изменения цены содержит 100 записей, а у второго – 75. В этом случае увеличить количество записей второго актива невозможно, значит нужно уменьшить количество записей первого актива, чтобы привести количество записей к равному количеству 75. Удалять следует наиболее старые записи, чтобы синхронизировать ход цен активов относительно времени.

Схема алгоритма получения данных представлена на рисунке 2.2.

Рисунок 2.2 – Схема алгоритма получения данных.

2.3.3 Разработка алгоритма оптимизации инвестиционного портфеля

Данный раздел имеет ключевое значение в разработке финансовой системы. Здесь определяется основной алгоритм, используемый для оптимизации инвестиционного портфеля в рамках системы. Алгоритм должен быть точным и эффективным, чтобы обеспечить быстрый и точный расчет.

Существует несколько подходов к оптимизации портфеля. Один из самых распространенных подходов – моделирование портфеля на основе сценариев. В этом подходе портфель моделируется на основе различных возможных сценариев, которые могут возникнуть на рынке. Эти сценарии могут быть основаны на исторических данных, экономических прогнозах и других факторах.

Другой подход – оптимизация портфеля на основе вероятностного анализа. В этом подходе портфель моделируется на основе вероятностных распределений доходности активов, а также на основе корреляций между различными активами. Этот подход используется для более точного расчета рисков, но требует большего количества вычислительных ресурсов.

Частью второго подхода являются стохастические методы, они используют вероятностные модели и методы анализа для изучения случайных процессов и событий. Одним из таких методов является метод Монте-Карло. Этот метод использует случайные числа для описания и моделирования систем, в которых происходят случайные события. Он основан на идее повторения экспериментов (симуляций), используя случайный набор значений для каждого параметра модели, чтобы получить аппроксимацию вероятностных распределений выходных данных. Таким образом, метод Монте-Карло позволяет оценить риски и вероятности различных событий в сложных финансовых системах. Он и будет реализован в разрабатываемой системе.

2.3.4 Разработка алгоритма расчета рисков

При выборе оптимального портфеля должны учитываться характеристики портфеля, такие как общая портфельная доходность и метрика риска Value at Risk. VaR показывает, какой максимальный убыток может понести инвестор при заданном уровне вероятности потери. Для этого портфелю присваивается уровень доверия. Например, уровень доверия 95%, тогда с вероятностью 95% максимальный убыток портфеля не будет превышать VaR, и с вероятностью 5% убыток будет больше VaR. VaR рассчитывается по формуле $VaR = cost_p * trust * \sigma_p$, где $cost_p$ – общая стоимость портфеля, trust – уровень доверия и σ_p – стандартное отклонение потенциальной потери портфеля. σ_p можно рассчитать по формуле

$$\sigma_p = \sqrt{\sum_{i=1}^{n} w_i^2 \sigma_i^2 + 2 \sum_{i=1}^{n} \sum_{j=i+1}^{n-1} w_i w_j cov(i,j)},$$

где w_i – вес i-го актива в портфеле, σ_i – стандартное отклонение доходности i-го актива, а cov(i, j) – ковариация между доходностями активов i и j. Схема алгоритма оптимизации представлена на рисунке 2.3.

Рисунок 2.3 – Схема алгоритма Монте-Карло.

2.3.5 Разработка протокола синхронизации данных

Разработка протоколов синхронизации данных является важным компонентом распределенной системы управления инвестиционным портфелем, позволяющим обеспечить корректную и своевременную передачу данных между различными компонентами системы.

Для обеспечения правильной синхронизации данных должно быть разработано два протокола синхронизации. Первый протокол должен синхронизировать общую работу вычислительных узлов путем передачи данных о текущем состоянии узла. Это необходимо для обеспечения равного распределения нагрузки между вычислительными узлами. Второй протокол должен обеспечивать управление доступом к базе данных.

Оба протокола строятся на основе общих принципов, таких как:

- Установка соединения между узлами – узлы должны установить соединение между собой для передачи информации.
- Определение формата сообщений – узлы должны договориться о формате сообщений, которые будут передаваться между ними.
- Определение типов сообщений – узлы должны определить типы сообщений, которые будут передаваться между ними. Должно быть два типа сообщений: о состоянии узла и о доступе к базе данных.
- Определение протокола обмена сообщениями – узлы должны определить протокол обмена сообщениями. Например, это может быть простой протокол «запрос-ответ», где один узел отправляет запрос, а другой отвечает на него.
- Определение протокола подтверждения – узлы должны определить протокол подтверждения для гарантии того, что сообщения были успешно доставлены.

Для протокола, обеспечивающего управление доступом к базе данных, существует еще один принцип – определение протокола блокировки данных.

Для обеспечения целостности данных, узлы должны определить протокол блокировки данных. Например, если один узел получил доступ на запись, то другие узлы должны временно блокировать доступ к тем же данным.

2.4 Реализация системы

Данный раздел представляет обзор и подробное описание технической реализации разработанной распределенной системы оптимизации инвестиционного портфеля. Здесь будут рассмотрены архитектурные решения, выбранные технологии и инструменты, а также основные этапы разработки, включая получение данных об активах, моделирование портфеля, оптимизацию и анализ рисков.

Перед началом реализации системы были проведены исследования и анализ требований, которые определили функциональность и особенность системы. В ходе разработки было уделено особое внимание выбору эффективных и масштабируемых технологий, которые позволили реализовать систему, способную обрабатывать большие объемы данных и предоставлять точные результаты оптимизации портфеля.

В целом, реализация распределенной системы оптимизации инвестиционного портфеля представляет собой комплексную и технически сложную задачу, которая требует интеграции различных компонентов и применения передовых технологий.

2.4.1 Общая архитектура системы

Общая архитектура системы представляет собой три вычислительных узла, связанных между собой через интернет. Программное обеспечение узлов идентично, за исключением центрального узла. На центральном узле производится скачивание, сортировка, удаление дубликатов и пропусков инвестиционных данных. После получения инвестиционных данных

происходит инициализация распределенной структуры данных на всех узлах, в которую загружается информация по инвестиционным инструментам. Каждый узел выполняет обработку данных, не зависимо от остальных узлов, после чего передает оптимизированный инвестиционный портфель на центральный узел. На рисунке 2.4 представлена схема взаимодействия узлов системы между собой.

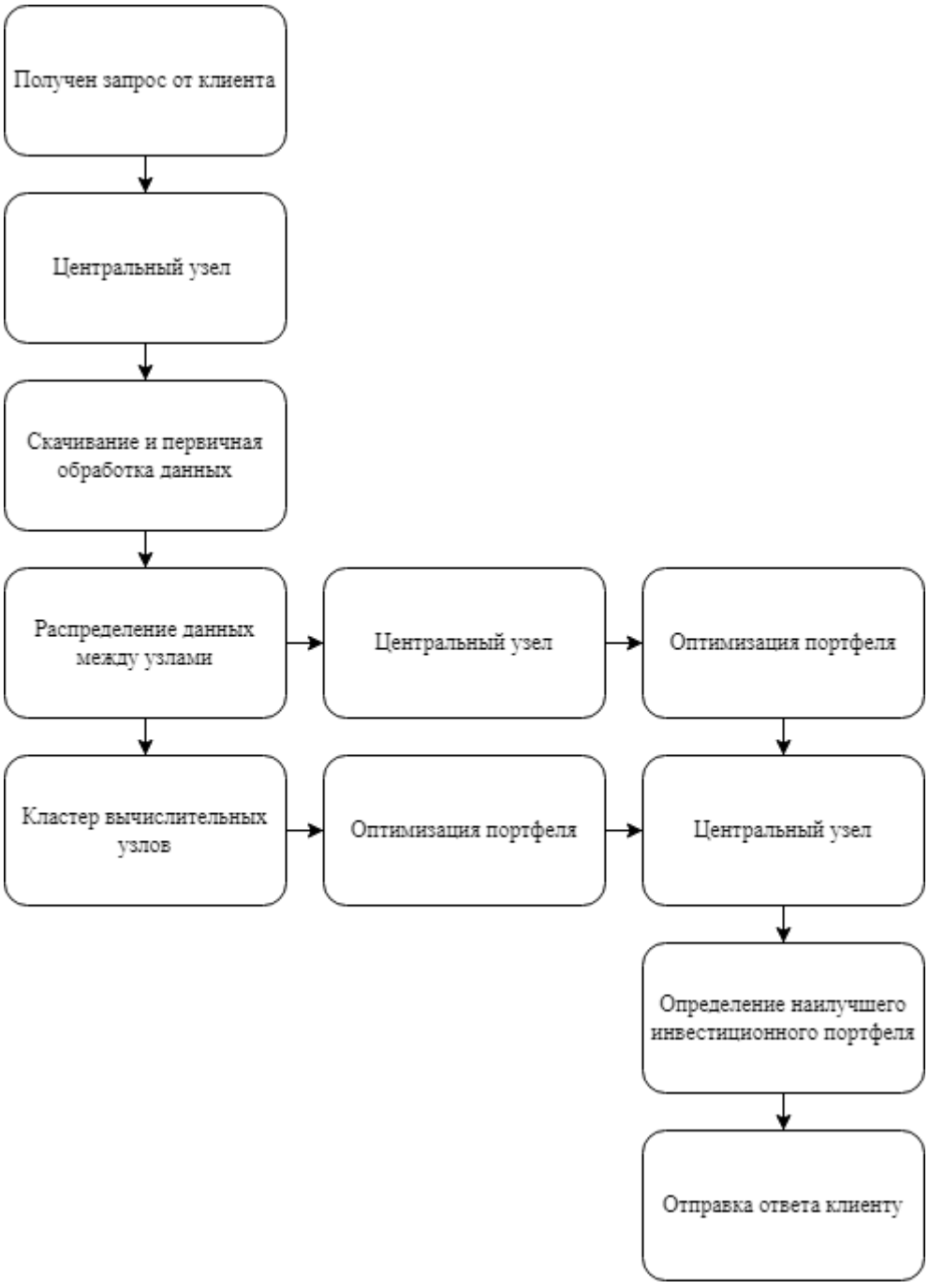

Рисунок 2.4 – Схема взаимодействия узлов системы.

Распределение данных между узлами подразумевает выгрузку данных о состоянии все активов в распределенную структуру данных, хранящую общие для всех исходные данные.

2.4.2 Получение данных об активах

Для получения данных был использован инструмент, предоставляемый брокером «Тинькофф Инвестиции», Tinkoff Invest API. Данный инструмент предоставляет возможность получения данных об активе за определенный промежуток времени. Чтобы узнать историю изменения цены конкретного актива, нужно знать его уникальный идентификатор, называемый FIGI. Чтобы не скачивать ненужные данные, была проведена предварительная фильтрация запрашиваемых инструментов. Для начала нужно узнать FIGI всех инструментов, доступных для торговли, после чего отсеять инструменты, торгующиеся в иностранной валюте и торгующиеся на иностранных биржах. После проделанной фильтрации можно скачивать инвестиционные данные. Код получения доступных инструментов будет выглядеть следующим образом:

```
//Все инструменты
var instruments = api.getInstrumentsService();
//Акции
var shares = instruments.getAllSharesSync();
//Облигации
var bonds = instruments.getAllBondsSync();
//Валюты и металлы
var currencies = instruments.getAllCurrenciesSync();
```

А код получения исторических цен для одного актива так:

```
//Получение тикера инструмента
String ticker = info.substring(info.indexOf(":") + 1, info.
     indexOf(","));
```

```
//Получение FIGI инструмента
String figi = info.substring(info.lastIndexOf(":") +
     1, info.length());
String path = stringPath;
path = path + ticker + "-" + figi + ".txt";
deleteFile(path);
//Получение исторических цен
var candlesDay = api.getMarketDataService().getCandlesSync(figi,
     from, to, CandleInterval.CANDLE_INTERVAL_DAY);
//Сохранение данных во временный файл
for (HistoricCandle candle : candlesDay) {
     printCandleInFile(candle, path);
}
```

Для получения данных о всех активах необходимо выполнить этот код для каждого актива.

2.4.3 Моделирование портфеля

Моделирование портфеля является важным аспектом разработки системы оптимизации инвестиционного портфеля. Оно позволяет оценить поведение портфеля на основе различных факторов, таких как цены активов, волатильность, корреляции и других параметров.

Для моделирования портфеля были реализованы алгоритмы, которые позволяют создать виртуальный портфель и оценить его производительность и риски.

Основными параметрами модели портфеля были приняты потенциальная доходность, риск, уровень доверия и список финансовых инструментов, которые находятся в портфеле. Для удобства был реализован алгоритм моделирования финансового инструмента, параметрами которого являются:

- Тикер – общепринятое название данного актива.

- Средняя доходность актива за рассматриваемый промежуток времени.
- Процентная доходность актива – отношение разницы цен актива в начале и в конце временного интервала к первой цене актива в рассматриваемом интервале.
- Стандартное отклонение доходности – математическое значение, показывающее колебание доходности актива ото дня ко дню.
- История изменения цены – массив цен рассматриваемого актива.
- Вес актива – доля содержания данного инструмента в инвестиционном портфеле.

2.4.4 Расчет рисков

Расчет рисков является важным компонентом оптимизации инвестиционного портфеля. Он позволяет оценить потенциальные потери и степень неопределенности, связанные с инвестициями в портфель.

Метрикой риска была выбрана Value at Risk (VaR), она показывает потенциальные потери портфеля при определенном уровне доверия к портфелю. Так, например, при уровне доверия портфелю 95% вероятность потенциальной потери больше VaR будет 5%, а с 95% вероятностью потери будут меньше. Алгоритм расчета VaR приведен в разделе 2.3.4 Разработка алгоритма расчета рисков. Программный код данного алгоритма будет выглядеть следующим образом:

```
private float calculateValueAtRisk(){
    return (1 - trust) * calculateSigmaLoss();
}
```

При этом уровень доверия задается пользователем вручную (по умолчанию задан 95%). А код расчета стандартного отклонения доходности портфеля выглядит так:

```
private float calculateSigmaLoss(){
```

```
        float varianceSummary = 0;
        float covarianceSummary = 0;
        //Расчет дисперсии
        for(FinancialInstrument instrument: instruments){
            varianceSummary += Math.pow(instrument.
                getWeight(),2) * Math.pow(
                    instrument.getSigmaProfit(), 2);
        }
        //Расчет корреляции
        for(int i = 0; i < instruments.size() - 1; i++){
            for(int j = i + 1; j < instruments.size(); j++){
                covarianceSummary +=
                    instruments.get(i).getWeight() *
                        instruments.get(j).getWeight() *
                            covarianceAB(
                                instruments.get(i),
                                    instruments.get(j));
            }
        }
        //Расчет стандартного отклонения доходности портфеля
        return (float)Math.sqrt(varianceSummary + 2 *
            covarianceSummary);
}
```

Данный код выполняется для каждого сгенерированного портфеля. Это одна из самых нагруженных частей системы ввиду большого количества вычислений внутри функции и внутри вызываемых ею функций.

2.4.5 Оптимизация портфеля

Конечным результатом проделанной работы должен быть оптимизированный инвестиционный портфель. Целью оптимизации – найти наилучшую комбинацию активов в портфеле с учетом заданных ограничений и целей инвестора.

В данной работе оптимизация была реализована с помощью метода Монте-Карло. Оптимизационный модуль генерирует большое количество случайных инвестиционных портфелей, среде которых выбирается наилучший. В данном модуле были реализованы методы генерации случайных финансовых инструментов и случайных значений весов каждого инструмента.

После генерации определенного количества портфелей необходимо найти наилучший. В данном случае оптимизация сводится к нахождению портфеля с наибольшей доходностью и наименьшим значением Value at Risk. Это можно сделать путем сортировки всех полученных портфелей по одному ключевому значению, являющимся комбинацией доходности и риска портфеля.

Код генерации случайных активов выглядит следующим образом:

```
private List<Path> randomizeInstruments(){
     instrumentsCount = (minInstrumentsCount + (int)(Math.
          random() * 1000) % 101 - minInstrumentsCount + 1);
     List<Path> choosedTemp = new ArrayList<Path>();
     List<Integer> choosedInstrumentsIndex =
          new ArrayList<Integer>();
     for(int i = 0; i < instrumentsCount; i++){
          int newIndex = (int)(Math.random() * 1000) %
               availableFiles.size();
          while(choosedInstrumentsIndex.contains(newIndex)){
               newIndex = (int)(Math.random() * 1000) %
                    availableFiles.size();
          }
          choosedInstrumentsIndex.add(newIndex);
          choosedTemp.add(availableFiles.get(newIndex));
     }
     return choosedTemp;
}
```

Вместе с этим код генерации случайных весов для инструментов:

```
public void randomizeWeights(Portfolio portfolio){
```

```
        int fractionSummary = 0;
        List<Integer> fraction = new ArrayList<Integer>();
        int maxFraction = (1 + (int)(Math.random() * 100)) * 100;
        int minFraction = portfolio.getInstruments().size() *
             maxFraction / 100;
        for(int i = 0; i < portfolio.getInstruments().size(); i++){
             fraction.add((int)(minFraction + (Math.random() *
                  10000) % (minFraction - maxFraction + 1)));
             fractionSummary += fraction.get(fraction.size()-1);
        }
        for(int i = 0; i < portfolio.getInstruments().size(); i++){
             try {
                  float weight = (float) fraction.get(i) /
                       fractionSummary;
                  portfolio.getInstruments().get(i).
                       setWeight(BigDecimal.
                            valueOf(weight).setScale(3,
                                 RoundingMode.HALF_UP).
                                      floatValue());
             }
             catch (NumberFormatException e){
                  System.out.println(e.getMessage());
             }
        }
        portfolio.updatePortfolio();
}
```

2.4.6 Распределенная структура данных

Для хранения данных обо всех инвестиционных инструментах и результатах оптимизации были реализованы две распределенные структуры данных. Первая представляет собой распределенный список, хранящий финансовые инструменты, класс которого содержит данные, необходимые для всех расчетов, такие как: история изменения цены, временные интервалы и

доходность данного актива. Вторая структура данных содержит лучшие результаты оптимизации от каждого узла и представляет собой распределенный список портфелей.

Распределенные структуры данных были реализованы с помощью библиотеки Hazelcast. Она предоставляет широкий спектр возможностей для работы с распределенными системами. Распределенный список расширяет базовый класс списка Java, добавляя возможность управления распределенной структурой данных, включая добавление новых узлов хранения данных через ip-адрес узла.

Для синхронизации распределенной структуры данных, хранящей результаты оптимизации, используются семафоры. Они позволяют достичь согласия данных на разных узлах и предотвратить одновременную перезапись данных несколькими процессами. Схема алгоритма работы структуры данных, хранящей результаты оптимизации каждого узла, показана на рисунке 2.5.

Рисунок 2.5 – Схема алгоритма распределенной структуры данных.

2.5 Анализ производительности и масштабируемости

В данном разделе будут описаны следующие аспекты:

- Производительность системы – скорость выполнения операций и запросов в системе, количество обработанных операций в единицу времени.
- Масштабируемость системы – оценка способности системы эффективно масштабироваться при увеличении нагрузки или добавлении дополнительных ресурсов.
- Результат оптимизации – анализ эффективности полученного оптимизированного инвестиционного портфеля.

2.5.1 Производительность системы

В данном разделе были проведены тесты для определения производительности системы в различных условиях. Ниже представлены основные результаты тестирования:

- Время отклика системы – в среднем, система обрабатывает запросы с временем отклика в диапазоне от 30 до 70 секунд. Это время включает в себя отправку запроса, обработку на сервере и получение ответа.
- Распределение нагрузки – во время тестирования была оценена способность системы распределять нагрузку между узлами. Показательный пример – при увеличении числа итераций система успешно распределяет нагрузку между узлами и поддерживает стабильную производительность.
- Отказоустойчивость – во время тестирования была проведена проверка отказоустойчивости системы. При сбое одного из узлов, система автоматически переключается на другие доступные узлы и продолжает обработку запросов.

Важно отметить, что время отклика системы зависит от количества доступных вычислительных узлов, но не сильно, так как в конечном итоге время отклика зависит от количества заданных итераций при оптимизации портфеля. На рисунке 2.6 представлен график сравнения времени отклика системы с одним узлом, двумя и системы с тремя доступными узлами.

Как можно заметить, в время отклика системы сильно зависит от количества доступных узлов. Среднее время выполнения запроса оптимизации на 1000 итераций на одном узле составляет 166 секунд, на двух узлах – 106 секунд, на трех узлах – 57 секунд. Это объясняется тем, что для повышения эффективности работы системы общее количество итераций делится между всеми узлами. То есть при количестве итераций 1000 и доступных трех узлах каждый узел выполняет в среднем 333 итерации.

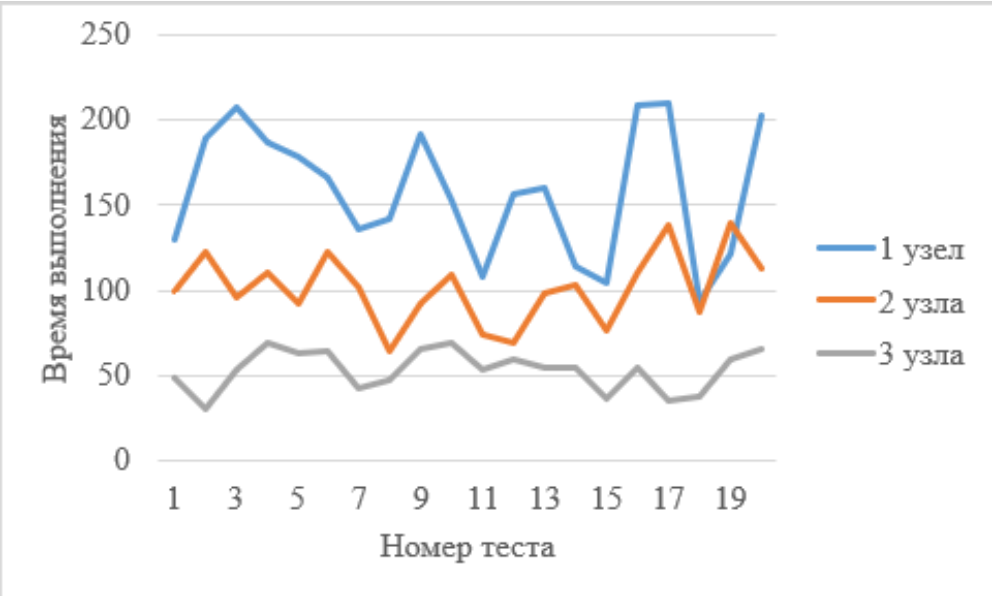

Рисунок 2.6 – Сравнение производительности 1 узла, 2 узлов и 3 узлов.

Также следует измерить среднюю производительность узлов в системе, то есть определить, какое количество итераций в секунду выполняют узлы. Результаты приведены на рисунке 2.7.

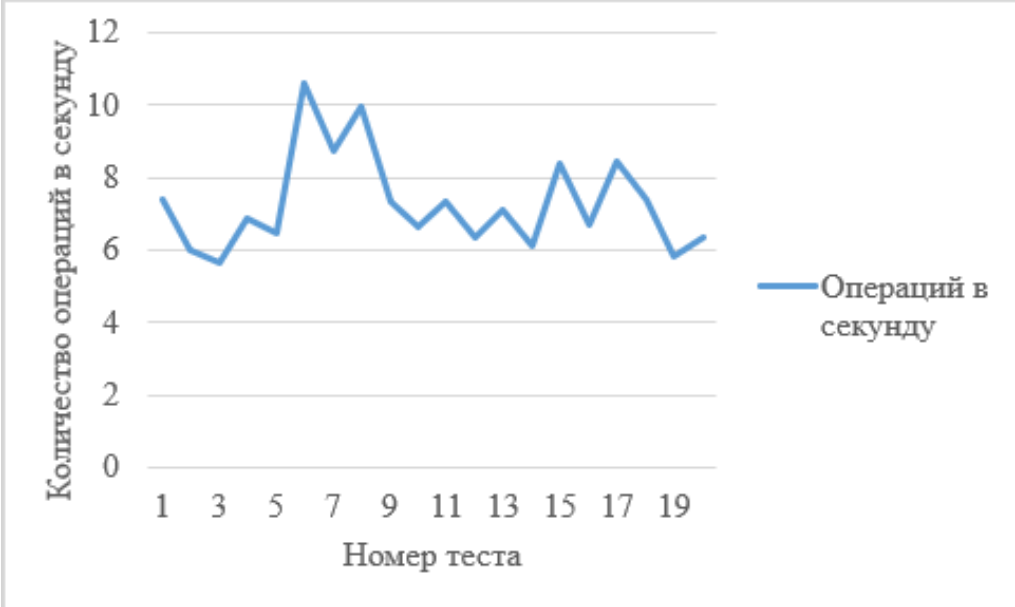

Рисунок 2.7 – Среднее количество итераций в секунду.

Как можно заметить, производительность колеблется от теста к тесту, это объясняется непостоянным количеством активов в генерируемых портфелях. Чем меньше размера портфеля, тем быстрее он генерируется и тем выше средняя производительность. За 20 тестов среднее выполняемое количество итераций равняется 7.

Также следует отследить зависимость времени выполнения оптимизации от количества итераций, результат исследования приведен на рисунке 2.8.

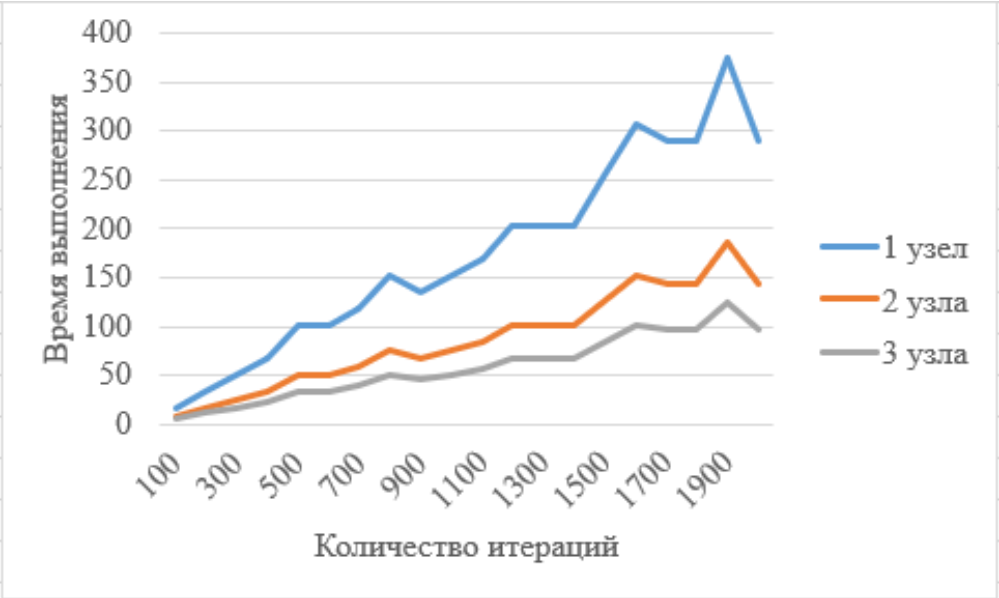

Рисунок 2.8 – Зависимость времени выполнения от итераций.

Очевидно, что с увеличением количества итераций возрастает и время выполнения оптимизации, как и ее качество.

Приведенные результаты позволяют сделать вывод о том, что система обладает приемлемой производительностью. Однако, для более точной оценки производительности системы стоит провести дополнительные тесты и анализировать результаты в реальных условиях использования системы. Это позволит выявить потенциально слабые места и предпринять соответствующие меры для их устранения и улучшения производительности.

2.5.2 Масштабируемость системы

Данный раздел представляет собой оценку способности системы масштабироваться с ростом объема данных. Здесь был проведен ряд тестов для определения масштабируемости системы. Ниже представлены основные результаты тестирования:

- Горизонтальное масштабирование [39]– увеличение количества узлов в системе способствует более высокой производительности оптимизации инвестиционного портфеля. С увеличением количества доступных узлов время выполнения оптимизации сокращается.
- Вертикальное масштабирование [40]– система поддерживание вертикальное масштабирование, то есть возможность увеличения ресурсов на каждом узле. Увеличение вычислительной мощности, объема памяти или пропускной способности сети на отдельных узлах системы приводит к улучшению производительности и способности обрабатывать более высокие нагрузки.
- Автоматическое распределение данных – система использует механизм автоматического распределения данных между узлами. При добавлении новых узлов в систему, данные автоматически реплицируются и распределяются между узлами, что обеспечивает балансировку нагрузки и сохранность данных.

Приведенные результаты подтверждают, что система обладает высокой масштабируемостью, что является важным фактором для успешной реализации распределенной системы. Масштабируемость системы позволяет ей эффективно обрабатывать большие объемы данных и поддерживать высокую производительность.

2.6 Пример работы программы

Для удобства пользования и тестирования был разработан простой интерфейс пользователя. Для примера работы программы были заданы следующие параметры оптимизации инвестиционного портфеля: количество итераций – 1000, минимальное количество активов в портфеле – 3, максимальное количество активов в портфеле – 30. Результат работы программы приведен на рисунке 2.9.

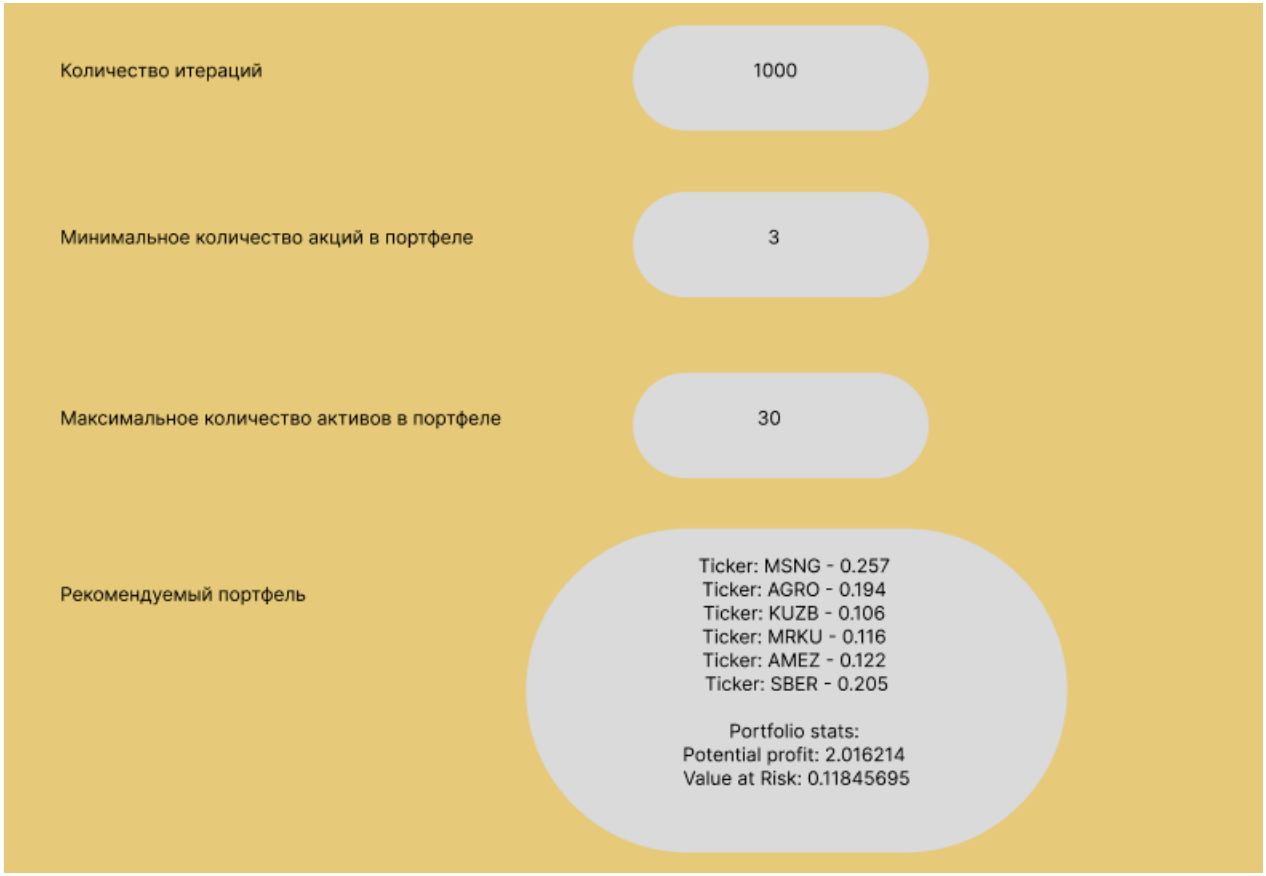

Рисунок 2.9 – Результат работы программы.

2.7 Оценка качества оптимизации инвестиционного портфеля

В конце следует оценить, на сколько хорошо получившаяся система оптимизирует инвестиционный портфель. Для оценки можно воспользоваться любым доступным сервисом, предоставляющим историю котировок финансовых инструментов, например, TradingView [41].

Оценка должна производится со стороны процентной прибыли, которую можно было бы получить при покупке инструментов, указанных на рисунке 2.6. Предполагается, что покупка производится 26.04.2023, так как для тестирования эффективности оптимизации портфеля анализировались цены активов за последнюю тысячу торговых дней до 26.04.2023. Оптимизация предполагает долгосрочное инвестирование средств от года и больше. Но даже так можно оценить промежуточный результат. Предполагается, что продажа активов производится 6.06.2023. С 26.04.2023 до 6.06.2023 прошло 30

торговых дней. В ходе оценивания были получены результаты, приведенные в таблице 2.1.

Таблица 2.1 – Оценка эффективности оптимизации портфеля.

Общее название актива	Доля актива в портфеле	Цена на 26.06.2023, руб.	Цена на 6.06.2023, руб.	Доходность, %
Мосэнерго	0,257	2,507	2,870	0,037
Русагро	0,194	8,270	9,620	0,032
Банк Кузнецкий	0,106	0,043	0,045	0,005
МРСК Урала	0,116	0,222	0,303	0,042
Ашинский метзавод	0,122	91,500	66,970	-0,033
Сбербанк	0,205	235,210	240,910	0,005
Общая доходность, %				0,088

3 Экономическое обоснование разработки распределенной системы

В данном разделе описывается процесс планирования разработки распределенной системы для оптимизации инвестиционного портфеля, а также расчёт стоимости разработки и определение кода разрабатываемого приложения.

3.1 Разработка плана проекта

Описываемый проект можно разделить на следующие частные работы:

1) Составление списка реализуемых функций системы;
2) Определение необходимых принципов работы системы;
3) Тестирование возможных методов оптимизации;
4) Реализация функций системы с учётом выбранного метода оптимизации;

a) Реализация основы системы (внутренних методов и распределенной структуры данных);

b) Реализация настроечных и инициализирующих функций;

c) Реализация функций оптимизации портфеля;

d) Реализация функций расчета рисков;

e) Реализация функций управления распределенной структурой данных;

f) Реализация математических функций;

5) Тестирование реализованных функций;

a) Тестирование настроечных и инициализирующих функций;

b) Тестирование функций оптимизации портфеля;

c) Тестирование функций расчета рисков;

d) Тестирование функций управления распределенной структурой данных;

e) Тестирование математических функций;

f) Совместное тестирование реализованных модулей;

6) Исправление ошибок, выявленных в ходе тестирования.

В соответствии со списком частных работ и планируемыми сроками их выполнения была составлена диаграмма Ганта [42], представленная на рисунке 3.1.

Таблица 3.1 – диаграмма Ганта

	Исполнитель	17.04	18.04	19.04	20.04	21.04	24.04	25.04	26.04	27.04	28.04	5.05	8.05	9.05	10.05	11.05
Разработчик	Разработчик															
Разработчик	Разработчик															
Разработчик	Разработчик															
Разработчик	Разработчик															
Разработчик	Разработчик															
Разработчик	Разработчик															
Разработчик	Разработчик															
Разработчик	Разработчик															
Разработчик	Разработчик															
Тестировщик	Тестировщик															
Тестировщик	Тестировщик															
Тестировщик	Тестировщик															
Тестировщик	Тестировщик															
Тестировщик	Тестировщик															
Тестировщик	Тестировщик															
Разработчик	Разработчик															

Продолжение таблицы 3.1.

Работы	Начало	Конец	5.04	6.04	7.04	10.04	11.04	12.04	13.04	14.04
1	5.04	5.04								
2	6.04	7.04								
3	10.04	11.04								
4a	12.04	13.04								
4b	14.04	14.04								
4c	17.04	18.04								
4d	19.04	19.04								
4e	20.04	21.04								
4f	24.04	24.04								
5a	20.04	20.04								
5b	21.04	24.04								
5c	25.04	25.04								
5d	26.04	27.04								
5e	28.04	28.04								
5f	5.05	8.05								
6	9.05	11.05								

3.2 Расчет цены проекта

Одним из важнейших вопросов перед началом разработки проекта является расчёт его себестоимости и цены. На этапе заключения договора на разработку сложно провести детальный расчёт себестоимости, убедительный для заказчика. В этих условиях может быть применена методика укрупнённого расчёта, представленная в данном разделе.

3.2.1 Расчет себестоимости проекта

Значение среднемесячной зарплаты разработчика (Zзп) примем за 50000 руб./мес. $Zзп = 30000$ руб./мес.

Дипломный проект выполняется в период Апрель-Июнь 2023 года. Среднемесячное число рабочих дней (Tср) в этот период:

$$\text{Тср} = \frac{(\text{Тапрель}+\text{Тмай}+\text{Тиюнь})}{3} = \frac{(20 + 23+22)}{3} = \ 21{,}6 \text{ дня}.$$

Тарифная дневная ставка разработчика (Zд):

$$Z\text{д} = \frac{Z\text{зп}}{\text{Тср}} = \frac{50000}{21{,}6} = 2314{,}81 \text{ руб./день}.$$

Процент страховых взносов, исчисляемых от фонда заработной платы (Ф), определим согласно действующему законодательству:

- Страховые взносы в ФНС на обязательное пенсионное страхование – 22%.
- Страховые взносы в ФНС на обязательное социальное страхование на случай временной нетрудоспособности и в связи с материнством – 2,9%.
- Страховые взносы в ФНС на обязательное медицинское страхование – 5,1%.
- Страховые взносы в ФСС на обязательное социальное страхование от несчастных случаев на производстве и профессиональных заболеваний – 0,2% (IT-индустрия считается наименее опасным видом бизнеса).

Итоговый процент страховых взносов является суммой вышеперечисленных процентов страховых взносов.

$$\Ф = 22 + 2{,}9 + 5{,}1 + 0{,}2 = \ 30{,}2\%.$$

Величина страховых взносов на работника в день (Ссд):

$$\text{Ссд} = Z\text{д} * \Ф = 2314{,}81 * 30{,}2\% = 699{,}07 \text{ руб./день}.$$

Дневная оплата работника с учётом страховых взносов (Zдс):

$$Z\text{дс} = Z\text{д} + \text{Ссд} = 2314{,}81 + 699{,}07 = 3013{,}88 \text{ руб./день}.$$

Процент накладных расходов (Н) примем за усреднённое значение процента накладных расходов в сфере IT – 65%.

$$\text{Н} = 65\%.$$

Накладные расходы на одного работника в день (Снр):

$$\text{Снр} = Z\text{д} * \text{Н} = 2314{,}81 * 65\% = 1504{,}62 \text{ руб./день}.$$

Стоимость человеко/дня (Сч/д):

$$\text{Сч/д} \ = \ Z\text{дс} + \text{Снр} = 3013{,}88 + 1504{,}62 = 4518{,}5 \text{ руб./день}.$$

Средний процент прибыли (П) возьмём как средний в сфере IT – 17%.

$$\text{П} = 17\%.$$

Дневная прибыль на одного работника (Спрд):

$$\text{Спрд} = \frac{\text{Сч}}{\text{д}} * \text{П} = 4518{,}5 * 17\% = 768{,}14 \text{ руб./день}.$$

Дневная ставка специалиста (без учёта НДС) (Сдсс):

$$\text{Сдсс} = \text{Сч/д} + \text{Спрд} = 4518{,}5 + 768{,}14 = 5286{,}64 \text{ руб./день}.$$

Ставку налога на добавленную стоимость (НДС) определим согласно действующему законодательству – 20%.

$$\text{НДС} = 20\%.$$

Величина налога на добавленную стоимость в расчёте на день (Сндс):

$$\text{Сндс} = \text{Сдсс} * \text{НДС} = 5286{,}64 * 20\% = 1057{,}33 \text{ руб./день}.$$

Полная дневная стоимость работы специалиста (Сполн):

$$\text{Сполн} = \text{Сдсс} + \text{Сндс} = 5286{,}64 + 1057{,}33 = 6343{,}97 \text{ руб./день}.$$

Сведём рассчитанные значения в таблицу 3.2.

Также добавим в таблицу 3.2 данные для остальных участников проекта – руководителя ВКР и консультанта по доп. разделу.

Таблица 3.2 – Расчёт себестоимости рабочего дня

Статья расходов	Затраты на разработчика	Затраты на тестировщика	Затраты на консультанта по доп. разделу	Затраты на руководителя ВКР
Среднемесячная зарплата разработчика, руб./месяц	50000	60000	67000	67000
Среднемесячное число рабочих дней, дней/месяц	21,6	21,6	21,6	21,6
Расчёт стоимости дневной работы специалиста				
Тарифная дневная ставка разработчика, руб./день	2314,81	2777,77	3101,85	3101,85
% страховых взносов	30,2	30,2	30,2	30,2

Продолжение таблицы 3.2.

Величина страховых взносов, руб./день	699,07	838,88	936,75	936,75
Дневная оплата работника со страховыми взносами, руб./день	3013,88	3616,65	4038,6	4038,6
% накладных расходов	65	65	42	42
Накладные расходы, руб./день	1504,62	1805,55	1302,77	1302,77
Стоимость человеко/дня, руб./день	4518,5	5422,2	5341,37	5341,37
% прибыли	17	17	17	17
Дневная прибыль, руб./день	768,14	921,77	908,03	908,03
Дневная ставка специалиста (без НДС), руб./день	5286,64	6341,97	6249,4	6249,4
Ставка НДС	20	20	20	20
Величина НДС, руб./день	1057,33	1268,79	1249,88	1249,88
Полная дневная стоимость работы специалиста	6343,97	7610,76	7499,28	7499,28

Рассчитав полную дневную стоимость работы участвующих специалистов, рассчитаем общую себестоимость разработки проекта (Спр):

Сполн.разр – полная дневная стоимость работы разработчика.

Сполн.рук – полная дневная стоимость работы руководителя ВКР.

Сполн.конс – полная дневная стоимость работы консультанта по доп. разделу.

Сполн.тест – полная дневная стоимость работы тестировщика.

Кзагр.разр – коэффициент загрузки разработчика.

Кзагр.рук – коэффициент загрузки руководителя ВКР.

Кзагр.конс – коэффициент загрузки консультанта по доп. разделу.

Кзагр.тест – коэффициент загрузки тестировщика.

Д – количество рабочих дней, отведённых на разработку проекта.

Др – количество рабочих дней разработчика.

Дт – количество рабочих дней тестировщика.

Спр = (Сполн.разр * Кзагр.разр) * Др +

+ (Сполн.рук * Кзагр.рук + Сполн.конс * Кзагр.конс) * Д +

+ (Сполн.тест * Кзагр.тест) * Дт = 6343,97 * 1 * 17 +

(7499,28 * 0,05 + 7499,28 * 0,03)* 20 +

+ 7610,76 * 1 * 9 = 107847,49 +

11998,84 + 68496,84 = 188343,17руб.

3.2.2 Расчет цены предлагаемого продукта

Данный проект подразумевает вывод на рынок нового программного средства. Цена в таком случае может быть определена из соотношения:

$$С_{прогр} = С_{пр} + С_{изгот}(1 + П)(1 + НДС),$$

где $С_{прогр}$ – цена программы, созданной в рамках проекта, $С_{изгот}$ – затраты на изготовление копий. Затраты на изготовление копии можно принять за $С_{пр}$.

$$С_{прогр} = 188343,17 + 188343,17(1 + 0,17)(1 + 0,2) = 452776,98 \text{ руб}$$

3.3 Определение классификационного кода разрабатываемого программного средства

Согласно классификации ОКПД 2, проект, разрабатываемый в рамках данной ВКР, можно отнести к следующему коду:

62.01.11.000 «Услуги по проектированию и разработке информационных технологий для прикладных задач и тестированию программного обеспечения».

ЗАКЛЮЧЕНИЕ

Целью данной работы являлась разработка распределенной вычислительной системы для моделирования оптимального инвестиционного портфеля. Полученное программное обеспечение отвечает основным критериям разработки распределенных систем: эффективность вычислений, масштабируемость, надежность и безопасность, и имеет гибкую архитектуру, разделенную на модули, что позволяет легко дополнять и развивать систему.

Разработанная распределенная система может помочь инвесторам принимать наиболее взвешенные решения при работе с финансовыми инструментами и привлечь внимание инвесторов к правильному управлению рисками при торговле на финансовых рынках.

Таким образом, полученное программное обеспечение может оказать положительное влияние на капитал частных инвесторов и больших инвестиционных компаний, минимизируя риски и максимизируя доходность.

Дальнейшее развитие продукта предполагает:

- Добавление дополнительных алгоритмов оптимизации портфеля, подходящих для решения специализированных задач.
- Реализация распределенной базы данных для хранения большего количества данных.
- Расширение списка финансовых инструментов и расширение анализируемых временных интервалов.
- Реализация дополнительных возможностей, благоприятно влияющих на доходность инвесторов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Официальный сайт языка программирования Java [Электронный ресурс] // Java.com: [сайт]. URL: https://www.java.com/. (Дата обращения: 20.04.2023).

2. Distributed Array [Электронный ресурс]// Sciencedirect.com: [сайт]. URL: https://www.sciencedirect.com/topics/computer-science/distributed-array. (Дата обращения: 20.04.2023).

3. Распределенные хэш-таблицы [Электронный ресурс]// Medium.com: [сайт]. URL: https://medium.com/dtechlog/технологии-распределённые-хэш-таблицы-565f77b610d1. (Дата обращения: 20.04.2023).

4. Архитектура распределенной очереди [Электронный ресурс]// Habr.com: [сайт]. URL: https://habr.com/ru/companies/vk/articles/573212/.(Дата обращения: 20.04.2023).

5. Распределенные графы [Электронный ресурс]// Youtube.com: [сайт]. URL: https://www.youtube.com/watch?v=Jka920VLh8M. (Дата обращения: 20.04.2023).

6. Интеграция распределенных систем через обмен сообщениями [Электронный ресурс]// Youtube.com: [сайт]. URL: https://www.youtube.com/watch?v=yYoiGX2c_tQ. (Дата обращения: 21.04.2023).

7. RPC и способы его мониторинга [Электронный ресурс]// Habr.com: [сайт]. URL: https://habr.com/ru/companies/rvision/articles/716656/. (Дата обращения: 21.04.2023).

8. Введение в REST API [Электронный ресурс]// Habr.com: [сайт]. URL: https://habr.com/ru/articles/483202/.(Дата обращения: 21.04.2023).

9. P2P протоколы обмена данными [Электронный ресурс]// Habr.com: [сайт]. URL: https://habr.com/ru/articles/596983/.(Дата обращения: 21.04.2023).

10. Агенты и актёры [Электронный ресурс]// Habr.com: [сайт]. URL: https://habr.com/ru/companies/host-tracker/articles/234623/.(Дата обращения: 21.04.2023).

11. Мьютекс [Электронный ресурс]//Wikipedia.org: [сайт]. URL: https://ru.wikipedia.org/wiki/Мьютекс. (Дата обращения: 22.04.2023).

12. Семафор [Электронный ресурс]// Wikipedia.org: [сайт]. URL: https://ru.wikipedia.org/wiki/Семафор_(программирование). (Дата обращения: 22.04.2023).

13. Монитор [Электронный ресурс]// Wikipedia.org: [сайт]. URL: https://ru.wikipedia.org/wiki/Монитор_(синхронизация). (Дата обращения: 22.04.2023).

14. Блокировка [Электронный ресурс]// Wikipedia.com: [сайт]. URL: https://ru.wikipedia.org/wiki/Спин-блокировка. (Дата обращения: 22.04.2023).

15. Atomic broadcast [Электронный ресурс]// Wikibrief.org: [сайт]. URL: https://ru.wikibrief.org/wiki/Atomic_broadcast. (Дата обращения: 22.04.2023).

16. Теория портфеля Марковица [Электронный ресурс]// Fin-plan.org: [сайт]. URL: https://fin-plan.org/blog/investitsii/teoriya-portfelya-markovitsa/.(Дата обращения: 23.04.2023).

17. Метод Монте-Карло [Электронный ресурс]// Habr.com: [сайт]. URL: https://habr.com/ru/articles/274975/.(Дата обращения: 23.04.2023).

18. Функция полезности [Электронный ресурс]// Scienceforum.ru: [сайт]. URL: https://scienceforum.ru/2017/article/2017032860. (Дата обращения: 23.04.2023).

19. Explanation Of VaR Calculations [Электронный ресурс]// Medium.com: [сайт]. URL: https://medium.com/fintechexplained/market-risk-explanation-of-var-calculations-e31734f26d93. (Дата обращения: 24.04.2023).

20. Expected Shortfall (ES) [Электронный ресурс]// Medium.com: [сайт]. URL: https://irmindiaaffiliate.medium.com/measuring-risk-expected-shortfall-es-or-value-at-risk-var-2dd03ad59b59. (Дата обращения: 24.04.2023).

21. Using Conditional Value at Risk [Электронный ресурс]// Medium.com: [сайт]. URL: https://medium.com/@mandyphg/using-conditional-value-at-risk-cvar-in-quantitative-risk-management-qrm-for-market-risk-844431d152cf. (Дата обращения: 24.04.2023).

22. Тинькофф Инвестиции [Электронный ресурс]// Tinkoff.ru: [сайт]. URL: https://www.tinkoff.ru/invest/.(Дата обращения: 24.04.2023).

23. Московская биржа [Электронный ресурс]// Moex.com: [сайт]. URL: https://www.moex.com/.(Дата обращения: 24.04.2023).

24. СПБ Биржа [Электронный ресурс]// Spbexchange.ru: [сайт]. URL: https://spbexchange.ru/.(Дата обращения: 24.04.2023).

25. Apache Hadloop [Электронный ресурс]// Hadloop.apache.org: [сайт]. URL: https://hadoop.apache.org/.(Дата обращения: 27.04.2023).

26. Apache Spark [Электронный ресурс]// Spark.apache.org: [сайт]. URL: https://spark.apache.org/.(Дата обращения: 27.04.2023).

27. Apache Cassandra [Электронный ресурс]// Cassandra.apache.org: [сайт]. URL: https://cassandra.apache.org/_/index.html. (Дата обращения: 27.04.2023).

28. Apache Kafka [Электронный ресурс]// kafka.apache.org: [сайт]. URL: https://kafka.apache.org/.(Дата обращения: 27.04.2023).

29. Apache ZooKeeper [Электронный ресурс]// Zookeeper.apache.org: [сайт]. URL: https://zookeeper.apache.org/.(Дата обращения: 27.04.2023).

30. Hazelcast [Электронный ресурс]// Hazelcast.com: [сайт]. URL: https://hazelcast.com/.(Дата обращения: 27.04.2023).

31. Apache Commons Math [Электронный ресурс]// Commons.apache.org: [сайт]. URL: https://commons.apache.org/proper/commons-math/.(Дата обращения: 27.04.2023).

32. JQuantLib [Электронный ресурс]// Jquantlib.com: [сайт]. URL: http://www.jquantlib.com/en/latest/.(Дата обращения: 28.04.2023).

33. QuantLib [Электронный ресурс]// Quantlib.org: [сайт]. URL: https://www.quantlib.org/.(Дата обращения: 28.04.2023).

34. Smile [Электронный ресурс]// Haifengl.github.io: [сайт]. URL: https://haifengl.github.io/.(Дата обращения: 28.04.2023).

35. QuantTools [Электронный ресурс]// Quanttools.bitbucket.io: [сайт]. URL: https://quanttools.bitbucket.io/_site/index.html. (Дата обращения: 28.04.2023).

36. Yahoo Finance API [Электронный ресурс]// Yo-sarawut.gitbook.io: [сайт]. URL: https://yo-sarawut.gitbook.io/tutorials/tutorials/finance/yahoo-finance-api. (Дата обращения: 28.04.2023).

37. Alpha Vantage API [Электронный ресурс]// Alphavantage.co: [сайт]. URL: https://www.alphavantage.co/.(Дата обращения: 28.04.2023).

38. Tinkoff Invest API [Электронный ресурс]// Tinkoff.github.io: [сайт]. URL: https://tinkoff.github.io/investAPI/.(Дата обращения: 30.04.2023).

39. Горизонтальное масштабирование [Электронный ресурс]// Habr.com: [сайт]. URL: https://habr.com/ru/companies/oleg-bunin/articles/319526/.(Дата обращения: 30.04.2023).

40. Балансировка нагрузки [Электронный ресурс]// 1cloud.ru: [сайт]. URL: https://1cloud.ru/blog/informatsionnyie-sistemyi-balansirovka-dlya-proizvoditelnosti. (Дата обращения: 5.05.2023).

41. TradingView [Электронный ресурс]// Tradingview.com: [сайт]. URL: https://ru.tradingview.com/chart/.(Дата обращения: 5.05.2023).

42. Диаграмма Ганта [Электронный ресурс]// Wikipedia.org: [сайт]. URL: https://ru.wikipedia.org/wiki/Диаграмма_Ганта(Дата обращения: 5.05.2023).

ПРИЛОЖЕНИЕ А

ФРАГМЕНТЫ ИСХОДНОГО КОДА

Файл DistributedResultList.java. Реализует распределенную структуру данных, хранящую результат.

```
public class ResultDistributedList {
//Конфигурация распределенной структуры данных
    private Config config;
    private HazelcastInstance server;
//Структура, хранящая результат оптимизации
    private IList<Portfolio> resultsList;
        public ResultDistributedList(){
        config = new Config();
//Добавление остальных узлов кроме текущего
        config.getNetworkConfig().getJoin().getTcpIpConfig().
addMember("Deleted");
        config.getNetworkConfig().getJoin().getTcpIpConfig().
addMember("Deleted");
//Индикатор подключения узлов
        server = Hazelcast.newHazelcastInstance(config);
        server.getClientService().addClientListener(
new ClientListener() {
            @Override
            public void clientConnected(Client client) {
                System.out.println(client.getClientType() + ", "
+
client.getUuid() + " is connected.");
            }
            @Override
            public void clientDisconnected(Client client) {
                System.out.println(client.getClientType() + ", "
+
client.getUuid() + " is disconnected.");
            }
```

```
        });
        resultsList = server.getList("Result");
    }
//Функция добавления результатов
    public void addPortfolio(Portfolio portfolio){
        resultsList.add(portfolio);
    }
//Функция получения результатов
    public IList<Portfolio> getResults(){
        return resultsList;
    }
}
```

Файл FinancialInstrument.java. Реализует класс финансового инструмента.

```
public class FinancialInstrument {
    private String ticker;           //Тикер актива
    private float averageProfit;     //Средняя доходность
    private float profit;            //Процентная доходность
    private List<Float> price;      //История цен
//Стандартное отклонение доходности
    private float sigmaProfit;
    private float weight;           //Вес актива в портфеле
    public FinancialInstrument(String _ticker,
          List<Float> _price){
        ticker = _ticker;
        price = _price;
        averageProfit = calculateAverageProfit();
        profit = calculateProfitPercentage();
        sigmaProfit = calculateSigmaProfit();
        weight = 0.01f;
    }
//Функция расчета средней доходности
    private float calculateAverageProfit(){
        float profitSummary = 0;
```

```
        for(int i = 1; i < price.size(); i++){
            profitSummary += price.get(i) - price.get(i - 1);
        }
        return profitSummary / (price.size() - 1);
    }
//Функция расчета процентного дохода
    private float calculateProfitPercentage(){
        return (price.get(price.size() - 1) - price.get(0)) /
               price.get(0);
    }
//Функция расчета стандартного отклонения доходности
    private float calculateSigmaProfit(){
        float tempSubstract;
        float variance = 0;
        for(int i = 1; i < price.size(); i++){
            tempSubstract = price.get(i) - price.get(i - 1);
            variance += Math.pow(averageProfit - tempSubstract,
                    2);
        }
        variance = variance / (price.size() - 1);
        return (float)Math.sqrt(variance);
    }
    public void setWeight(float _weight){
        weight = _weight;
    }
    public float getProfit() {
        return profit;
    }
    public float getWeight() {
        return weight;
    }
    public List<Float> getPrice(){
        return price;
    }
```

```
public float getAverageProfit() {
    return averageProfit;
}
public float getSigmaProfit(){
    return sigmaProfit;
}
public String getTicker(){
    return ticker;
}}
```

Файл Portfolio.java. Реализует класс инвестиционного портфеля.

```
public class Portfolio {
    private List<FinancialInstrument> instruments;
    private float trust;       //Уровень доверия
    private float valueAtRisk;
    private float profit;
    private float fullWeight;
    public Portfolio(){
        instruments = new ArrayList<FinancialInstrument>();
        trust =  0.95f;
        profit = 0;
        valueAtRisk = 0;
        fullWeight = 0;
    }
//Функция добавления финансового инструмента в портфель
    public void addInstrument(FinancialInstrument
           instrument){
        instruments.add(instrument);
        profit = calculateProfit();
        valueAtRisk = calculateValueAtRisk();
        fullWeight += instrument.getWeight();
    }
//Функция расчета потенциальной доходности портфеля
    private float calculateProfit(){
        float profitSummary = 0;
```

```
        for(FinancialInstrument instrument: instruments){
            profitSummary += instrument.getProfit() *
                instrument.getWeight();
        }
        return profitSummary;
    }
//Функция расчета риска портфеля
    private float calculateValueAtRisk(){
        return (1 - trust) * calculateSigmaLoss();
    }
//Функция расчета стандартного отклонения доходности портфеля
    private float calculateSigmaLoss(){
        float varianceSummary = 0;
        float covarianceSummary = 0;
        for(FinancialInstrument instrument: instruments){
            varianceSummary += Math.pow(instrument.
                getWeight(),2) * Math.pow(instrument
                    getSigmaProfit(), 2);
        }
        for(int i = 0; i < instruments.size() - 1; i++){
            for(int j = i + 1; j < instruments.size(); j++){
                covarianceSummary += instruments.get(i).
                    getWeight() * instruments.get(j).
                        getWeight() *
                       covarianceAB(instruments.get(i),
                            instruments.get(j));
            }
        }
        return (float)Math.sqrt(varianceSummary + 2 *
                covarianceSummary);
    }
//Функция расчета корреляции двух активов в портфеле
    private float covarianceAB(FinancialInstrument A,
           FinancialInstrument B){
```

```
        return covarianceRatioAB(A, B) * A.getSigmaProfit() *
B.getSigmaProfit();
    }
//Функция расчета коэффициента корреляции двух активов
    private float covarianceRatioAB(FinancialInstrument A,
           FinancialInstrument B){
        float numerator = 0;
        float denominatorA = 0;
        float denominatorB = 0;
        List<Float> priceA = A.getPrice();
        List<Float> priceB = B.getPrice();
        for(int i = 0; i < priceA.size(); i++){
            numerator += (priceA.get(i) - A.
                getAverageProfit()) * (priceB.get(i) - B.
                     getAverageProfit());
            denominatorA += Math.pow(priceA.get(i) - A.
                getAverageProfit(), 2);
            denominatorB += Math.pow(priceA.get(i) - A.
                getAverageProfit(), 2);
        }
        return numerator / (denominatorA * denominatorB);
    }
//Функция обновления портфеля
    public void updatePortfolio(){
        profit = calculateProfit();
        valueAtRisk = calculateValueAtRisk();
        fullWeight = 0;
        for(FinancialInstrument instrument: instruments){
            fullWeight += instrument.getWeight();
        }
    }
    public List<FinancialInstrument> getInstruments(){
        return instruments;
    }
```

```
        public float getValueAtRisk() {
            return valueAtRisk;
        }
        public float getProfit() {
            return profit;
        }
        public float getFullWeight() {
            return fullWeight;
        }
    }
```

Файл PortfolioRandomizer.java. Реализует генерацию случайных инвестиционных портфелей.

```
public class PortfolioRandomizer {
    List<Portfolio> portfoliosList;
    int portfoliosCount;
    Loader loader;
    public PortfolioRandomizer(int minInstrumentsCount,
       int portfoliosCount){
        this.portfoliosCount = portfoliosCount;
        portfoliosList = new ArrayList<Portfolio>();
        loader = new Loader(minInstrumentsCount);
    }
//Функция генерации случайных инвестиционных портфелей
    public void randomizePortfolios(){
        for(int i = 0; i < portfoliosCount; i++){
            Portfolio portfolio = new Portfolio();
            loadOnePortfolio(portfolio);
            portfoliosList.add(portfolio);
        }
    }
//Функция добавления случайных активов и их весов в портфели
    private void loadOnePortfolio(Portfolio portfolio){
        loader.loadDataInPortfolio(portfolio);
        loader.randomizeWeights(portfolio);
```

```
    }
    public List<Portfolio> getPortfolios(){
        return this.portfoliosList;
    }
}
```

Файл Optimizer.java. Реализует поиск наилучших инвестиционных портфелей из предложенных.

```
public class Optimizer{
    private int bestPortfoliosCount;
    public Optimizer(int bestPortfoliosCount){
        this.bestPortfoliosCount = bestPortfoliosCount;
    }
    public List<Portfolio> optimizer(List<Portfolio>
          portfolioList){
        List<Portfolio> bestPortfolios = new
              ArrayList<Portfolio>();
        sortPortfolioList(portfolioList);
        for(int i = 0; i < bestPortfoliosCount; i++){
            bestPortfolios.add(portfolioList.get(i));
        }
        return bestPortfolios;
    }
//Функция нахождения лучших портфелей из сгенерированных
    public void sortPortfolioList(List<Portfolio> portfolioList){
        Comparator<Portfolio> portfolioComparator = new
              Comparator<Portfolio>() {
            @Override
            public int compare(Portfolio o1, Portfolio o2) {
                if((o1.getProfit() - o1.getValueAtRisk()) >
                    (o2.getProfit() - o2.getValueAtRisk())){
                        return -1;
                     }
                else if((o1.getProfit() - o1.getValueAtRisk()) <
                    (o2.getProfit() - o2.getValueAtRisk())){
```

```
                        return 1;
                    }
                    else return 0;
                }
            };
            portfolioList.sort(portfolioComparator);
        }
    }
```