

Пенза 2020

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 М. А. Митрохин
«08» ноября 2018 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

1. Студент Тугарев Иван Дмитриевич гр. 18ВВм1 факультета вычислительной техники
2. Тема работы «Реализация распределенных алгоритмов на вычислительной сети».
Тема утверждена приказом ПГУ № 1039/о от «30» сентября 2019 г.
3. Руководитель работы д. т. н., профессор каф. ВТ ПГУ Зинкин С. А.
4. Задание на работу (назначение разработки, исходные данные и т.п.)
Назначение разработки: передача подзадач для реализации распределённого алгоритма внутри локальной вычислительной сети.
5. Перечень подлежащих разработке вопросов.
Необходимо построить программу таким образом, чтобы любой узел мог получить сведения о текущей конфигурации сети под настроенный алгоритм, получить соответствующие указания для включения в цепочку выполнения распределённого алгоритма, а также обеспечить проверку выполнения алгоритма внутри сети.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области	17	22.03.2020	
2.	Изучение технологий разработки приложений с распределёнными вычислениями	15	08.04.2020	
3.	Разработка модели выполнения распределения подзадач узлам в распределённом алгоритме	15	23.04.2020	
4.	Разработка программы выполнения распределённого алгоритма	15	29.04.2020	
5.	Отладка и тестирование	15	29.04.2020	
6.	Составление выводов	11	18.05.2020	
7.	Оформление выпускной работы	12	10.06.2020	

Дата выдачи задания «26» сентября 2019 г.

Руководитель ВКР


(подпись, дата)

26.09.19 Зинкин С. А.

Задание к исполнению принял студент


(подпись, дата)

Тугарев И. Д.

Работу к защите допустить
Декан факультета


(подпись, дата)

Фионова Л. Р.

РЕФЕРАТ

Пояснительная записка содержит 97 листов, 25 рисунков, 2 таблицы, 25 источников, 3 приложения.

ЛОКАЛЬНАЯ ВЫЧИСЛИТЕЛЬНАЯ СЕТЬ, КЛИЕНТ-СЕРВЕРНЫЕ ПРИЛОЖЕНИЯ, БЕССЕРВЕРНЫЕ ВЫЧИСЛЕНИЯ, РАСПРЕДЕЛЁННЫМ ВЫЧИСЛЕНИЯ

Объект исследования является локальная вычислительная сеть.

Цель работы — разработка программы получения узлом локальной вычислительной сети сведений о текущей конфигурации сети под настроенный алгоритм, получить соответствующие указания для включения в цепочку выполнения распределённого алгоритма, а также обеспечить проверку выполнения алгоритма внутри сети.

Программа серверной части должна получать данные о реализуемом распределённом алгоритме из файла. После этого ищутся активные узлы-приложения, которые добавляются в список. Узлы приложения создаются в приложении клиента и, подключаясь к приложению, получают данные о выполняемой задаче автоматически. При подключении необходимого количества узлов к программе сервера начинается процесс выполнения распределённого алгоритма подключёнными узлами приложений. По завершении своей работы узлы передают выполненный результат на сервер.

В результате исследования была разработана программа распределения подзадач в локальной вычислительной среде.

Данное приложение является минимальной моделью бессерверной системы на локальной вычислительной сети и для эксплуатации на производстве разработанной системы необходимо дальнейшее развитие модели развёртки распределённого алгоритма на вычислительную сеть.

СОДЕРЖАНИЕ

Перечень сокращений и обозначений	6
Введение	7
1 Организация распределённых вычислений	9
1.1 Понятие вычислительной сети	9
1.1.1 Назначение вычислительных сетей. Основные определения и термины	9
1.1.2. Режимы работы и предоставляемые услуги.....	13
1.1.3 Классификация вычислительных сетей	14
1.2 Распределённые вычислительные системы	20
1.2.1 Введение в распределённые вычислительные системы	20
1.2.2 Промежуточное программное обеспечение	22
1.2.3 Терминология РВС	24
1.2.4 Классификация РВС	25
1.2.5 Связь в РВС.....	26
1.3 Бессерверные вычисления	27
1.3.1 Основные понятия	27
2 РЕАЛИЗАЦИЯ КЛИЕНТ-СЕРВЕРНОГО ПРИЛОЖЕНИЯ	31
2.1 Определение инструментов разработки.....	31
2.1.1 Выбор языка программирования и среды разработки ПО	31
2.1.2 Выбор вида хранимых данных и инструментов для их хранения	33
2.2 Структура данных в приложении	33
2.2.1 Описание функциональных единиц, используемых в алгоритме	33
2.2.2 Примеры моделей распределённого алгоритма.....	34
2.2.3 Формальное представление распределённого алгоритма в бессерверной системе	35
2.3 Структура приложений	36
2.3.1 Структура клиентского приложения	36
2.3.2 Структура серверного приложения	37
2.4 Выбор архитектуры приложения и формирование поведения системы	38

2.5 Описание процесса разработки системы	39
2.5.1 Описание разработки клиентского приложения	39
2.5.2 Описание разработки серверного приложения	42
3 ОПИСАНИЕ ПРОГРАММЫ	44
3.1 Описание клиентского приложения	44
3.2 Описание серверного приложения	46
3.3 Отладка и тестирование системы	47
3.4 Возможные пути последующих исследований	49
Заключение	50
Список использованных источников	51
Приложение А листинги Сервера	53
Приложение А.1 Файл main.cpp	53
Приложение А.2 Файл trackerserver.h	53
Приложение А.3 Файл trackerserver.cpp	55
Приложение А.4 Файл trackerworker.h	58
Приложение А.5 Файл trackerworker.cpp	59
Приложение Б Листинги программы Клиента	62
Приложение Б.1 Файл main.cpp	62
Приложение Б.2 Файл mainwindow.h	62
Приложение Б.3 Файл mainwindow.cpp	64
Приложение Б.4 Файл nodemanager.h	66
Приложение Б.5 Файл nodemanager.cpp	67
Приложение Б.6 Файл netnodeinterface.h	68
Приложение Б.7 Файл networknode.h	69
Приложение Б.8 Файл networknode.cpp	70
Приложение Б.9 Файл nodetask.h	77
Приложение Б.10 Файл nodetask.cpp	78
Приложение Б.11 Файл nodeserver.h	80
Приложение Б.12 Файл nodeserver.cpp	81
Приложение Б.13 Файл nodeserverworker.h	83

Приложение Б.14 Файл <code>nodeserverworker.cpp</code>	84
Приложение В Презентация	86

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей работе применяют следующие сокращения и обозначения:

ЛВС – локальная вычислительная сеть.

РВС – распределённая вычислительная система.

OSI – базовая эталонная модель взаимодействия открытых систем. (Open Systems Interconnections basic reference model)

FaaS – модель «функция как сервис». (Function as a Service)

ПО – программное обеспечение

ППО – промежуточное программное обеспечение

DNS – система доменных имен (Domain Name System)

ОС – операционная система

КС – компьютерная сеть

ИВС – информационно-вычислительной системы

АСУ – автоматизированная система управления

P2P – Одноранговая сеть(peer-to-peer)

ВВЕДЕНИЕ

С каждым годом увеличивается потребность в компьютерном вычислении. В течение последних десятилетий планомерно повышению потребностей также росли и вычислительные мощности устройств, предназначенных для этой цели. Однако темпы роста характеристик вычислительных систем падает с каждым годом. Вследствие чего одним лишь улучшением производительности отдельной машины выполнять имеющиеся вычислительные задачи в заданный срок стало невозможным.

В связи с этим были приняты решения увеличивать количество вычислительных модулей в системе. Это можно увидеть в увеличении числа ядер процессора на одном кристалле, возможности установить множество устройств на одной плате, так и путём соединения нескольких вычислительных систем в одну сеть.

Выполнение задач на наборе компьютеров в одной сети позволяет создавать вычислительные системы, способные обработать большие наборы данных, при этом позволяя сократить расходы на приобретение вычислительных ресурсов, а также данный способ увеличения вычислительных мощностей хорошо масштабируем в отличие от решений, выполненных на одном физическом устройстве.

В связи с удобством развертывания локальных сетей было придумано множество способов взаимодействия узлов сети между собой, тем не менее проблема развёртки распределённого приложения на ЛВС и быстрая смена её конфигурации всё ещё остаётся актуальной.

Целью данной выпускной квалификационной работы является разработка программы получения узлом локальной вычислительной сети сведений о текущей конфигурации сети под настроенный алгоритм, получить соответствующие указания для включения в цепочку выполнения распределённого алгоритма, а также обеспечить проверку выполнения алгоритма внутри сети.

Для достижения указанной цели были поставлены и решены следующие задачи:

- изучение как выполняются распределённые вычисления;
- изучение технологии бессерверных вычислений;

Объект исследования — локальная вычислительная сеть, состоящая из набора компьютеров.

Предметом исследования являются модели и алгоритмы развёртывания распределённых вычислительных систем на сеть.

1 ОРГАНИЗАЦИЯ РАСПРЕДЕЛЁННЫХ ВЫЧИСЛЕНИЙ

1.1 Понятие вычислительной сети

1.1.1 Назначение вычислительных сетей. Основные определения и термины

По мере развития и увеличения объемов баз данных, усложнения средств программного обеспечения, в том числе и прикладного, возникла насущная необходимость в соединении отдельных компьютеров между собой. Такое связывание компьютеров, позволяющее объединить их ресурсы – процессоры, память (внутреннюю и внешнюю, включая жесткие диски, разнообразные внешние устройства – принтеры, факс-аппараты, модемы и др.), каналы связи, и представляет собой вычислительную сеть, в которой каждый компьютер может передать другому компьютеру, подключенному к сети, любой набор данных. Таким образом, любая компьютерная система, состоящая из нескольких компьютеров, наверняка перерастет в более сложную систему, которая потребует высокоскоростного обмена данными между компьютерами с сервисными возможностями. Такой обмен не может быть организован при помощи стандартных простых средств операционных систем (ОС) и прикладных программ, а требует организации принципиально новой информационной структуры – вычислительной сети. [2]

Сеть – это совокупность объектов, образуемых устройствами передачи и обработки данных. Существуют понятия коммуникационная сеть и информационная сеть. Коммуникационная сеть предназначена для передачи данных, также выполнения задачи с их преобразованием. Коммуникационные сети различаются по типу используемых физических средств соединения. Прием и передачу звука, сигнала, текста, знака, письменного изображения по кабельной, проводной, магнитной, оптической, радио и другим сетям принято называть телекоммуникацией. Телекоммуникация – форма связи, способ передачи информации на большие расстояния. Система технических средств, с

помощью которой осуществляется телекоммуникация, называется сетью телекоммуникаций. Информационная сеть предназначена для хранения информации и состоит из информационных систем. На базе коммуникационной сети может быть построена группа информационных сетей. Под информационной системой следует понимать систему, которая является поставщиком или потребителем информации, или объект, способный осуществлять хранение, обработку или передачу информации. В состав информационной системы входят компьютеры, программы, пользователи и другие составляющие, предназначенные для процесса обработки и передачи данных. В дальнейшем информационная система, предназначенная для решения задач пользователя, будет называться рабочей станцией. Рабочая станция в сети отличается от обычного персонального компьютера наличием сетевой карты, канала для передачи данных и сетевого программного обеспечения. Компьютерная сеть (КС) состоит из информационных систем и каналов связи и представляет собой магистральные информационные структуры, состоящие из физического и логического уровней или составляющих, основное назначение которых – обмен информацией.

Физический уровень представлен компонентами сети, обеспечивающими физическое соединение между компьютерами. К ним зачастую относят среду передачи данных, проводную или беспроводную, узловые элементы, такие как маршрутизаторы и пр., конечные элементы/разъёмы/заглушки и т.п.

На рисунке 1 изображена схема физического уровня сети.



Рисунок 1 -- Схема физического уровня сети

Логический уровень – это разнообразное программное обеспечение, предоставляющее возможность использования имеющихся в наличии физических компонент сети. Среди всего многообразия ПО можно выделить несколько типов: драйверы и фоновые процессы сетевых протоколов операционных систем, серверные или клиентские приложения системы. На рисунке 2 изображена схема логического уровня вычислительной сети.



Рисунок 2 -- Схема логического уровня сети

Под каналом связи следует понимать путь или средство, по которому передаются сигналы. Средство передачи сигналов называют абонентским, или физическим, каналом. Каналы связи (data link) создаются по линиям связи при

помощи сетевого оборудования и физических средств связи. Физические средства связи построены на основе витых пар, коаксиальных кабелей, оптических каналов или эфира. Между взаимодействующими информационными системами через физические каналы коммуникационной сети и узлы коммутации устанавливаются логические каналы.

Логический канал – это путь для передачи данных от одной системы к другой, который прокладывается по маршруту в одном или нескольких физических каналах, его можно охарактеризовать как маршрут, проложенный через физические каналы и узлы коммутации. Информация в сети передается блоками данных по процедурам обмена между объектами. Эти процедуры называют протоколами передачи данных. Протокол – это совокупность правил, устанавливающих формат и процедуры обмена информацией между двумя или несколькими устройствами. Загрузка сети характеризуется параметром, называемым трафиком. Трафик (traffic) – это поток сообщений в сети передачи данных. Под ним понимают количественное измерение в выбранных точках сети числа проходящих блоков данных и их длины, выраженное в битах в секунду. Существенное влияние на характеристику сети оказывает метод доступа. Метод доступа – это способ определения того, какая из рабочих станций сможет следующей использовать канал связи и как управлять доступом к каналу связи (кабелю). В сети все рабочие станции физически соединены между собой каналами связи по определенной структуре, называемой топологией. Топология – это описание физических соединений в сети, указывающее, какие рабочие станции могут связываться между собой. Тип топологии определяет производительность, работоспособность и надежность эксплуатации рабочих станций, а также время обращения к файловому серверу. В зависимости от топологии сети используется тот или иной метод доступа. Состав основных элементов в сети зависит от ее архитектуры. Архитектура – это концепция, определяющая взаимосвязь, структуру и функции взаимодействия рабочих станций в сети. Она предусматривает логическую, функциональную и физическую организации

технических и программных средств сети. Архитектура определяет принципы построения и функционирования аппаратного и программного обеспечения элементов сети.

В основном выделяют три вида архитектур: архитектура терминал – главный компьютер, архитектура клиент – сервер и одноранговая архитектура. Современные сети можно классифицировать по различным признакам: удаленности компьютеров, топологии, назначению, перечню предоставляемых услуг, принципам управления (централизованные и децентрализованные), методам коммутации, методам доступа, видам среды передачи, скоростям передачи данных и т. д. Все эти понятия будут рассмотрены более подробно при дальнейшем изучении.

1.1.2. Режимы работы и предоставляемые услуги

Режимы работы:

- обмен данными между абонентскими системами;
- запрос и выдача информации;
- сбор данных;
- пакетная обработка по запросам удаленных пользователей;
- диалоговый режим.

Предоставляемые услуги:

- телекоммуникационные услуги – электронная почта, телеконференции и телесеминары, электронные доски объявлений, передача больших массивов – файлов, размножение сообщений и передача их по заранее подготовленному списку и др.;
- информационные услуги – поиск информации по вопросам, интересующим абонентов;
- консультационные услуги – консультации по информационному и программному обеспечению сети, консультации по технологии использования общесетевых ресурсов и др.;

- технические услуги – установка программного обеспечения, установка и тестирование модемов и др.;
- рекламные услуги – размещение рекламы в электронных конференциях и семинарах, на электронных досках объявлений.

1.1.3 Классификация вычислительных сетей

На рисунке 3 указана классификация вычислительных сетей.



Рисунок 3 -- Классификация компьютерных сетей

1. По функциональному принципу:

- вычислительные сети – сети, ориентированные на обработку данных, большая часть работ связана с вычислительными процедурами;
- информационные сети – преобладающим является информационно-справочная работа.

2. По информационному принципу сети в зависимости от организации передачи делят на несколько групп:

- сети с селекцией информации – информация как бы вбрасывается в сеть вся, но извлечь ее может только тот абонентский пункт, которому она предназначена;

- сети с маршрутизацией информации – устанавливается маршрут от источника к получателю, и информация идет только по этому маршруту. Информация, таким образом, доступна только получателю. Сети с маршрутизацией, в свою очередь, делятся на три группы:

- сети с коммутацией каналов;
- сети с коммутацией сообщений;
- сети с коммутацией пакетов.

3. По принципу структурной классификации сети делят на абонентские и ассоциативные системы. На рисунке 4 изображена структурная классификация сети.



Рисунок 4 -- Структурная классификация сети

Абонентские системы в зависимости от выполняемых функций подразделяются на четыре вида: рабочие, терминальные, административные и смешанные. Рабочая система предназначена для предоставления пользователю информационно-вычислительных ресурсов: банка данных, результатов обработки задач по подсистемам АСУ и т.д. Терминальная система

предоставляет абонентам (пользователям) информационно-вычислительной системы (ИВС) через один или несколько терминалов информационно-вычислительные ресурсы рабочих систем (часто функции рабочей и терминальной систем совмещены). Административной называется система, на которую возлагаются функции управления всей либо какой-нибудь частью ИВС. Смешанной система называется в том случае, если она выполняет функции двух, а иногда даже трёх рассмотренных выше видов абонентских систем. Ассоциативные системы в зависимости от выполняемых функций подразделяются на два вида: межсетевые и сетевые. Межсетевой называется ассоциативная система, предназначенная для обеспечения взаимодействия двух либо более ИВС. Ассоциативная система, которая связывает абонентские системы внутри одной сети, получила название сетевой.

4. По территориальному признаку

Основным признаком отличия является классификация КС по размерам. В зависимости от протяжённости КС принято делить на три вида: локальные, региональные и глобальные. На рисунке 5 изображена классификация сетей по территориальному признаку.

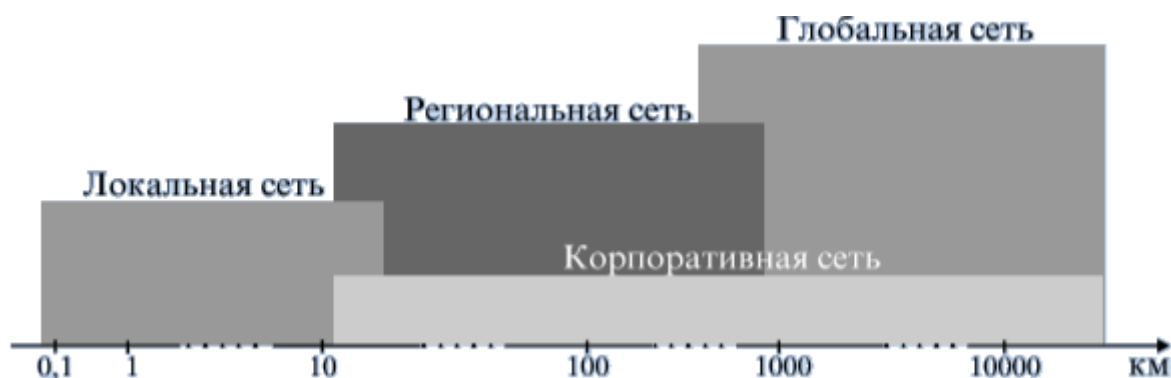


Рисунок 5 -- Классификация сетей по территориальному признаку

Локальной называется сеть, абоненты которой находятся на небольшом расстоянии друг от друга. Протяженность локальной вычислительной сети ограничивается несколькими километрами. Обычно ЛВС охватывают одно либо несколько расположенных рядом зданий. Именно на базе локальной ИВС разрабатываются современные АСУ фирмы, банка, вуза и т.д.

Региональная компьютерная сеть связывает абонентов, расположенных на значительном (от 10 до 1000 км) расстоянии друг от друга. Она может включать абонентов района, города, области и даже небольшой страны. Глобальная компьютерная сеть является третьим видом ИВС, она объединяет абонентов, расположенных на территории большой страны, разных стран и даже континентов. Построение этой сети возможно с помощью спутников. В настоящий момент имеется несколько глобальных компьютерных сетей и их протоколов, например, RelCom, CompuServ, Internet и т.д. Большинство таких сетей имеют тысячи серверов, десятки и сотни тысяч пользователей и носят статус международных, так как связывают компьютерные системы различных стран и континентов. В последнее время для характеристики ИВС всё чаще стали использовать понятие корпоративные компьютерные сети. Эти сети объединяют ряд предприятий одной фирмы, в зависимости от взаиморасположения предприятий они могут быть региональными или глобальными. АС в таких сетях взаимодействуют на базе различных территориальных сетей связи, в которых используются телефонные линии связи, радиосвязь, системы спутниковой связи. Корпоративные компьютерные сети – техническая база корпорации. Им принадлежит ведущая роль в реализации задач планирования, организации и осуществления производственно-хозяйственной деятельности корпорации.

5. По условиям доступа

Все сети, в том числе и глобальные, делят на коммерческие, в которых доступ и услуги сервисных служб платные, и некоммерческие – т.е. «условно бесплатные». Условно означает, что какую-то плату за подключение и использование сетевых служб, а также эксплуатацию систем связи пользователь все-таки вносит, но она несоизмеримо меньше, нежели в коммерческих системах, однако и уровень сервиса соответственный. Коммерческие сети поддерживаются профессиональными организациями, цель которых – предоставление сетевых услуг и высококачественного коммерческого сетевого сервиса. Некоммерческие, как правило,

поддерживаются на добровольных началах образовательными и информационными структурами и организациями общественного характера, не имеют четкой организации, единого управления, целенаправленного структурирования и стратегии развития.

6. По топологии сети делятся на:

- широковещательные – сети с передачей информации всем её участникам;
- сети с последовательной передачей – когда информация либо напрямую отправляется получателю, либо проходит несколько последовательных сегментов сети и доступна только конкретному получателю.

К широковещательным структурам относятся общая шина, «звезда» с пассивным центром, дерево. На рисунке 6 изображены широковещательные конфигурации сетей.

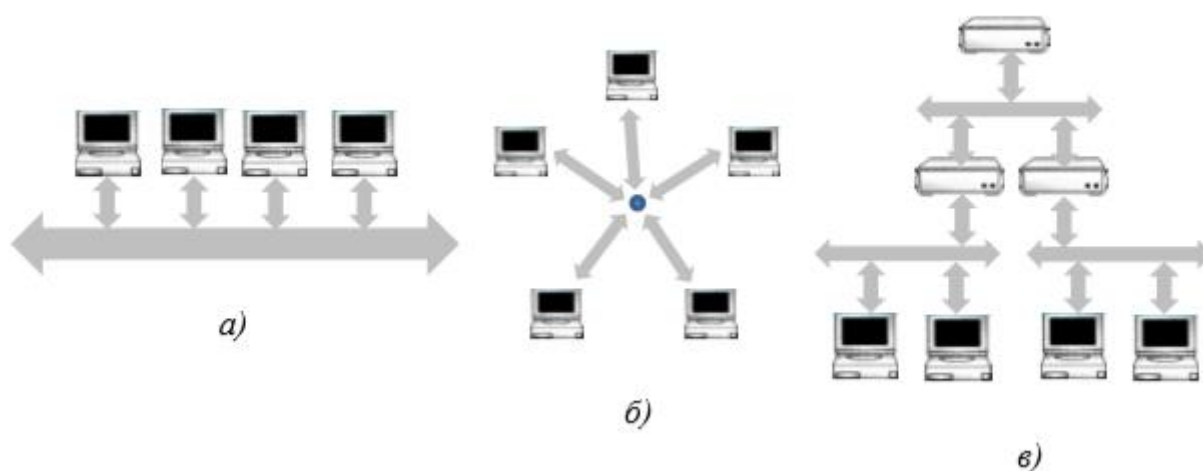


Рисунок 6 – Широковещательные конфигурации сетей: а – общая шина; б – «звезда» с пассивным центром; в – дерево

В таких структурах в каждый момент времени активна одна станция. Сообщения доступны всем, но принимает их только та, которой они предназначены. В последовательных конфигурациях, характерных для сетей с маршрутизацией информации, данные передаются последовательно от одной сети к соседней, причем на различных участках сети могут использоваться разные виды физической передающей среды. К последовательным

конфигурациям относятся произвольная (ячеистая), полносвязная, «кольцо», «цепочка», «звезда» с интеллектуальным центром. На рисунке 7 изображена последовательные конфигурации сетей.

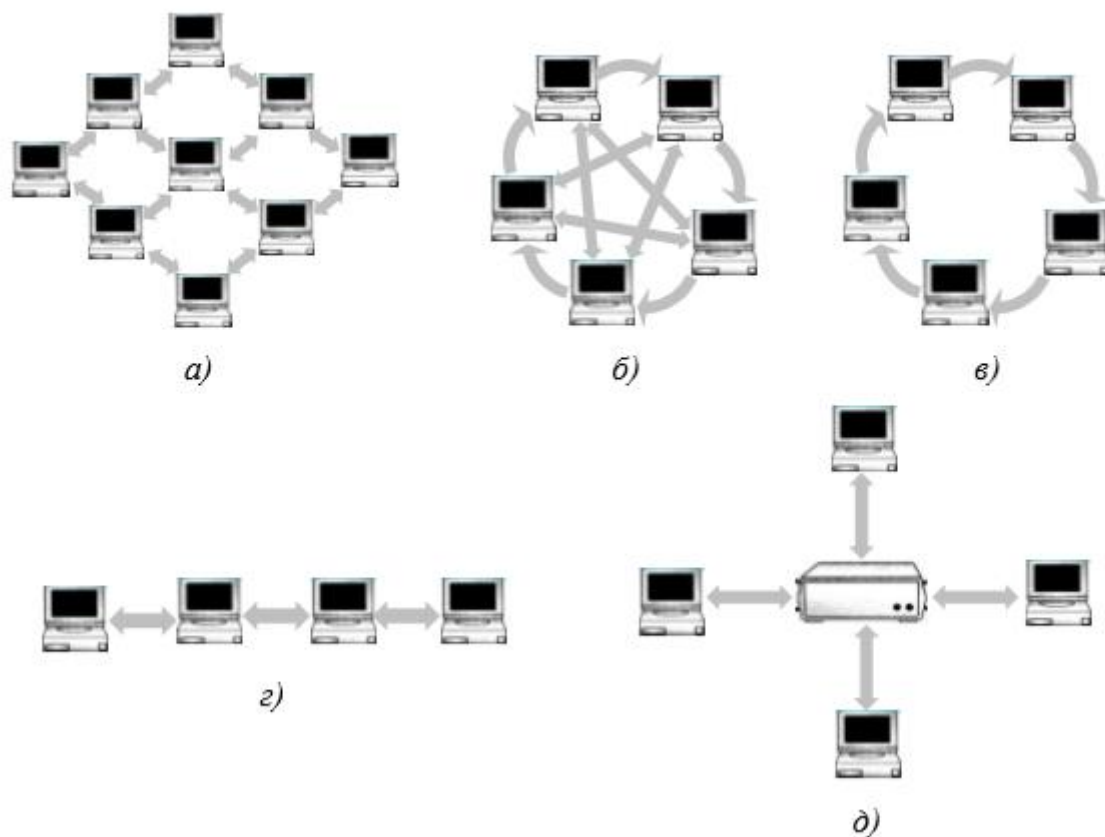


Рисунок 7 - Последовательные конфигурации сетей: а – ячеистая, б – полносвязная, в – «кольцо»; г – «цепочка»; д – «звезда» с интеллектуальным центром

7. По виду обмена все сети делятся на четыре группы в зависимости от организации передачи по каналу:

- 1) симплексные сети – обмен осуществляется только по одному из направлений;
- 2) полудуплексные сети – обмен может осуществляться в 2 направления, но в один момент времени возможна передача только одному направлению;
- 3) дуплексные сети – одновременная передача по обоим направлениям канала;
- 4) широкополосные системы – для передачи сигналов в обоих направлениях используется несущая частота.

1.2 Распределённые вычислительные системы

1.2.1 Введение в распределенные вычислительные системы

Область распределенных вычислительных систем в настоящее время характеризуется быстрыми темпами изменения идеологий и подходов. За короткую историю существования систем такого типа появилось множество различных парадигм реализации распределенных вычислений, набравших большой вес и общее признание, но практически исчезнувших впоследствии под давлением более новых и модных подходов. Однако когда технология исчезает из виду, очень часто она появляется вновь под новым именем. В результате происходит непрерывное перемешивание базовых концепций с новейшими подходами к разработке. В середине 1990-х существовало два основных подхода к разработке распределенных вычислительных систем. С одной стороны, концепция Веб представляла собой ориентированное на человека распределенное информационное пространство. Между тем системы распределенных объектов росли с точки зрения предоставляемых возможностей, но становились все более тяжелыми в плане разработки и использования. Сразу после начала нового тысячелетия произошел взрыв развития новых методов и промежуточного программного обеспечения для распределенных вычислительных систем, включая технологии одноранговых сетей (peer-to-peer или P2P) и грид-технологии. Применение P2P позволило множеству пользователей, которые раньше были простыми потребителями информации, поучаствовать в предоставлении контента. С другой стороны, применение технологии грид позволило интегрировать крупные комплексы обработки и хранения данных, обеспечивая их доступность для различных правительственных и научных пользователей. Концепция грид-вычислений была ориентирована на построение инфраструктуры, обеспечивающей «вычисления по требованию», аналогично тому, как мы сейчас получаем доступ к коммунальным услугам, например, к электричеству.

В то же время, бизнес-сообщество занялось разработкой следующего поколения спецификаций, призванных решить проблемы ранних стандартов распределенных объектных технологий посредством Веб-сервисов и сервисориентированной архитектуры. Слияние бизнес подхода к предоставлению вычислительных ресурсов в виде сервисов и концепции грид-вычислений привело к появлению в конце 2010-х новой концепции получившей название Облачных вычислений. Далее в этой главе мы попытаемся дать определение распределенных вычислительных систем (РВС), рассмотрим их основы и базовые понятия.

Формального определения распределенной вычислительной системы в настоящее время не существует. Из множества различных определений, можно выделить ироничное определение Лесли Лампорта¹ – Распределенной вычислительной системой можно назвать такую систему, в которой отказ компьютера, о существовании которого вы даже не подозревали, может сделать ваш собственный компьютер непригодным к использованию.

Это определение он дал в мае 1987 года, в своем письме коллегам по поводу очередного отключения электроэнергии в машинном зале. Эндрю Таненбаум, в своем фундаментальном труде «Распределённые системы. Принципы и парадигмы» предложил следующее (чуть более строгое) определение, которого мы будем придерживаться в рамках данной книги:

«Распределенная вычислительная система (РВС) – это набор соединенных каналами связи независимых компьютеров, которые с точки зрения пользователя некоторого программного обеспечения выглядят единым целым».

В этом определении фиксируются два существенных момента: автономность узлов РВС и представление системы пользователем, как единой структуры. При этом, основным связующим звеном распределенных вычислительных систем является программное обеспечение.

1.2.2 Промежуточное программное обеспечение

Распределенная вычислительная система представляет собой программноаппаратный комплекс, ориентированный на решение определенных задач. С одной стороны, каждый вычислительный узел является автономным элементом. С другой стороны, программная составляющая РВС должна обеспечивать пользователям видимость работы с единой вычислительной системой. В связи с этим выделяют следующие важные характеристики РВС: – возможность работы с различными типами устройств: – с различными поставщиками устройств; – с различными операционными системами, – с различными аппаратными платформами. [3]

Вычислительные среды, состоящие из множества вычислительных систем на базе разных программно-аппаратных платформ, называются гетерогенными; – возможность простого расширения и масштабирования; – перманентная (постоянная) доступность ресурсов (даже если некоторые элементы РВС некоторое время могут находиться вне доступа); – сокрытие особенностей коммуникации от пользователей.

Для обеспечения работы гетерогенного оборудования РВС в виде единого целого, стек программного обеспечения (ПО) обычно разбивают на два слоя. На верхнем слое располагаются распределенные приложения, отвечающие за решение определенных прикладных задач средствами РВС. Их функциональные возможности базируются на нижнем слое – промежуточном программном обеспечении (ППО). ППО взаимодействует с системным ПО и сетевым уровнем, для обеспечения прозрачности работы приложений в РВС.

На рисунке 8 изображены слои программного обеспечения в РВС

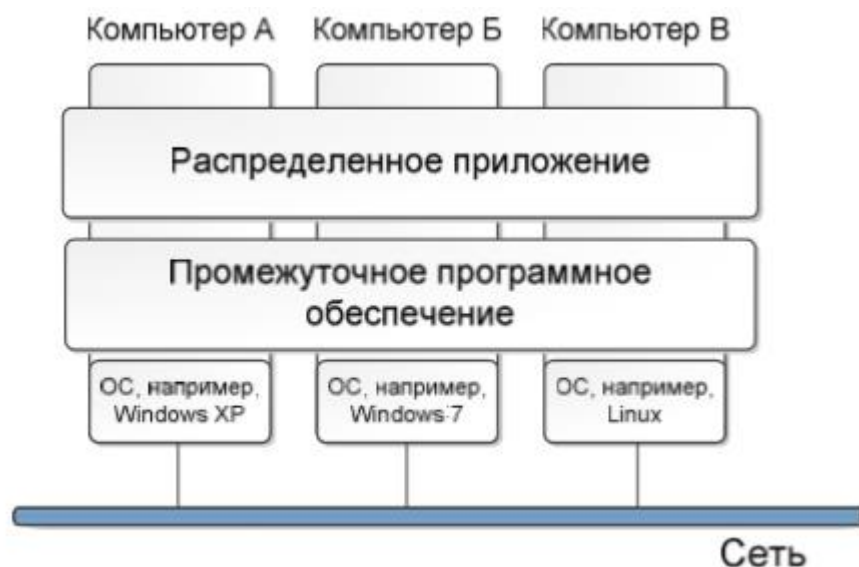


Рисунок 8 -- Слои программного обеспечения в РВС

Для того чтобы РВС могла быть представлена пользователю как единая система, применяют следующие типы прозрачности в РВС:

- прозрачный доступ к ресурсам – от пользователей должна быть скрыта разница в представлении данных и в способах доступа к ресурсам РВС;
- прозрачное местоположение ресурсов – место физического расположения требуемого ресурса должно быть несущественно для пользователя;
- репликация – сокрытие от пользователя того, что в реальности существует более одной копии используемых ресурсов;
- параллельный доступ – возможность совместного (одновременного) использования одного и того же ресурса различными пользователями независимо друг от друга. При этом факт совместного использования ресурса должен оставаться скрытым от пользователя;
- прозрачность отказов – отказ (отключение) каких-либо ресурсов РВС не должен оказывать влияния на работу пользователя и его приложения.

1.2.3 Терминология PBC

1. Ресурсом называется любая программная или аппаратная сущность, представленная или используемая в распределенной сети, будь то компьютер, файл, канал связи и т.п.

2. Узел – любое аппаратное устройство в PBC.

3. Сервер – устройство предоставления информации в PBC.

4. Клиент – устройство, запрашивающее и получающее информацию в PBC.

5. Пир – это узел, выполняющий роль как клиента, так и сервера.

6. Сервис – это сетевая сущность, предоставляющая определенные функциональные возможности. В рамках одного узла могут предоставляться несколько различных сервисов.

Из схемы видно, что каждый компьютер или устройство представляется узлом внутри распределённой системы. При этом на каждом узле может располагаться несколько клиентов, серверов, сервисов или пиров. Важно заметить, что все участники системы, за исключением клиента, являются ресурсами распределенной вычислительной системы.

На рисунке 9 приведена схема взаимоотношений между указанными терминами.



Рисунок 9 -- Схема взаимоотношений между терминами PBC

Сервис получает запрос на предоставление определенных данных (почти как аргументы, передаваемые при вызове локальной функции) и возвращает ответ. Таким образом, сервис можно определить как некую замену вызова функции на локальном компьютере. Существует множество технологий, обеспечивающих создание и сопровождение сервисов в распределенных вычислительных системах.

1.2.4 Классификация PBC

Наиболее характерными признаками классификации PBC по уровню централизации сети (от централизованного до децентрализованного) выделяют:

- методы обнаружения ресурсов;
- доступность ресурсов;

Существует множество различных технологий, обеспечивающих поиск и обнаружение ресурсов в PBC. Наиболее известным примером централизованной сети является служба DNS. Принцип работы данной службы многими чертами схож с работой с телефонной книгой. При получении адреса DNS служба передаёт клиенту IP-адрес для подключения к сервису. Таким образом, сервер DNS представляет собой большую базу данных ресурсов, расположенных в PBC. Существует ограниченное количество серверов, которые предоставляют службу DNS. И если указанные сервера отключены или недоступны по другой причине, то процесс обнаружения ресурсов останавливается, если вручную не указать альтернативные сервера. При использовании децентрализованного метода обнаружения ресурсов (например, в сети Gnutella) запрос на поиск отправляется всем узлам, известным отправителю. Эти узлы производят поиск ресурса у себя, и транслируют запрос далее. Таким образом, отсутствуют выделенные узлы для обнаружения и централизованное хранилище информации о ресурсах, доступных в сети. Другим важным фактором является доступность ресурсов PBC. Примером централизованной доступности ресурсов в PBC может являться технология веб-сервисов. Существует только один сервер с выделенным IP-адресом, который

предоставляет определенный веб-сервис или сайт. Если данный узел выйдет из строя или будет отключен от сети, данный сервис станет недоступна. Естественно, можно применить методы репликации для расширения доступности определенного сайта или сервиса, но доступность определенного IP-адреса останется прежней. Существуют системы, предоставляющие децентрализованные подходы к доступности ресурсов посредством множественного дублирования сервисов, которые могут обеспечить функциональность, необходимую пользователю. Наиболее яркими примерами децентрализованной доступности ресурсов могут служить одноранговые вычислительные системы (BitTorrent, Gnutella, Napster), где каждый узел играет роль, как клиента, так и сервера, который может предоставлять ресурсы и сервисы, аналогичные остальным устройствам данной сети. Еще одним критерием классификации РВС могут служить методы взаимодействия узлов. Централизованный подход к взаимодействию узлов основан на том, что взаимодействие между узлами всегда происходит через специальный центральный сервер. Таким образом, один узел не может обратиться к другому непосредственно. Децентрализованный подход к взаимодействию реализуется в одноранговых вычислительных системах. Такой подход основывается на прямом взаимодействии между узлами РВС, т.к. каждый узел играет как роль клиента, так и роль сервера.

1.2.5 Связь в РВС

Понятие «распределенная вычислительная система» подразумевает, что компоненты такой системы распределены, т.е. удалены друг от друга. Очевидно, что функционирование подобных систем невозможно без эффективной связи между ее компонентами. Задачи организации обмена между распределенными (территориально, административно и т.д.) компонентами давно и в значительной мере успешно решаются в вычислительных сетях, и, естественно, что РВС используют наработанный опыт. Взаимодействие в вычислительных сетях базируется на протоколах. Протокол – это набор правил

и соглашений, описывающих процедуру взаимодействия между компонентами системы (в том числе и вычислительной).

На рисунке 10 представлены уровни модели OSI.



Рисунок 10 -- Уровни модели OSI

Если система поддерживает определённый протокол, она, с большой долей вероятности, окажется способна взаимодействовать с другой системой, которая так же поддерживает данный протокол. В области вычислительных коммуникаций уже длительное время существует общепринятая система протоколов – сетевая модель OSI. Эта модель представляет собой стек протоколов разного уровня, которые позволяют описать практически все аспекты взаимодействия компонентов РВС.

1.3 Бессерверные вычисления

1.3.1 Основные понятия

Бессерверные вычисления (или просто serverless) становятся парадигмой развертывания облачных приложений, образованной в связи с переносом предоставляемых приложений в контейнеры и микросервисы. Внимание к данной архитектуре растёт с каждым днём, которое уделяется бессерверным

вычислениям на отраслевых выставках. С точки зрения клиента «Инфраструктура как услуга» (IaaS), это изменение парадигмы представляет как возможность, так и риск. С одной стороны, он предоставляет разработчикам упрощенную модель программирования для создания облачных приложений, которая устраняет большинство, если не все, эксплуатационные проблемы; это снижает стоимость развертывания облачного кода, взимая плату за время выполнения, а не за распределение ресурсов; и это платформа для быстрого развертывания небольших кусков облачного кода, который отвечает на события, например, для координации микросервисных композиций, которые в противном случае выполнялись бы на клиенте или на выделенном промежуточном программном обеспечении. С другой стороны, развертывание таких приложений на бессерверной платформе является сложной задачей и требует отказа от решений по проектированию платформы, касающихся, в частности, свойств контроля качества (QoS), масштабирования и отказоустойчивости.[4]

С точки зрения поставщика облачных вычислений бессерверная передача данных предоставляет дополнительную возможность контролировать весь стек разработки, снижая эксплуатационные расходы за счет эффективной оптимизации и управления облачными ресурсами, предлагая платформу, которая поощряющую расширение списка услуг со временем в получившейся системе, при этом сокращая усилия, необходимые для разработки и поддержки данной системы. Бессерверные вычисления – архитектура, в которой разрабатываемые составляют из себя набора микросервисов, выполняющих одну небольшую функцию. Это ни в коем случае не означает, что серверов нет, просто разработчик должен оставить большую часть оперативных ресурсов, таких как предоставление ресурсов, мониторинг, обслуживание, масштабируемость и отказоустойчивость поставщику облачных услуг. [5]

Тем не менее, читатель может отличать от модели «Платформа как услуга» (PaaS), которая также абстрагируется от управления серверами. Модель без сервера предоставляет «урезанную» модель программирования,

основанную на функциях без сохранения состояния. В отличие от PaaS, разработчики могут писать произвольный код и не ограничиваются использованием предварительно упакованного приложения. В состав поставщиков платформ для организации бессерверных систем входят Amazon, IBM, Microsoft и Google недавно выпустили свои платформы для развертывания бессерверных вычислений. Существуют также проекты с открытым исходным кодом, например, OpenLambda. Бессерверные вычисления находятся в зачаточном состоянии, и в настоящее время исследовательское сообщество выпустило лишь несколько цитируемых публикаций.

Бессерверные вычисления предназначены для краткосрочных взаимодействий клиента с бессерверной системой. Зачастую данная архитектура имеет ограниченное время взаимодействия с клиентом, что не позволяет использовать serverless как классические веб-серверов. [7]

При использовании бессерверных вычислений все сводится к программному коду и вызовам API, приложения раскладываются на компоненты. Например, вместо создания серверов приложений аутентификации можно задействовать стороннего провайдера для выполнения функций аутентификации. Они будут выполняться за счет вызовов API и обмена сообщениями между приложением и внешней системой аутентификации. Каждый процесс аутентификации будет осуществляться независимо и завершаться, как правило, через несколько секунд. [8]

Вы можете решить, что это обойдется слишком дорого. Но это не так, если учесть, что вызов API стоит несколько тысячных долей цента.

Важнее, наверное, то, что эластичное горизонтальное масштабирование становится очень простым.

Уменьшение стоимости инфраструктуры также сокращает общие расходы на предоставление доступа к интернет-приложению. Это позволяет в многих случаях существенно понизить затраты на поддержку инфраструктуры, однако полностью избавиться от затрат данная архитектура не позволяет.

Среди минусов можно отметить зависимость от API, предлагаемой архитектурой. [9]

На рисунке 11 изображена модель бессерверного приложения.

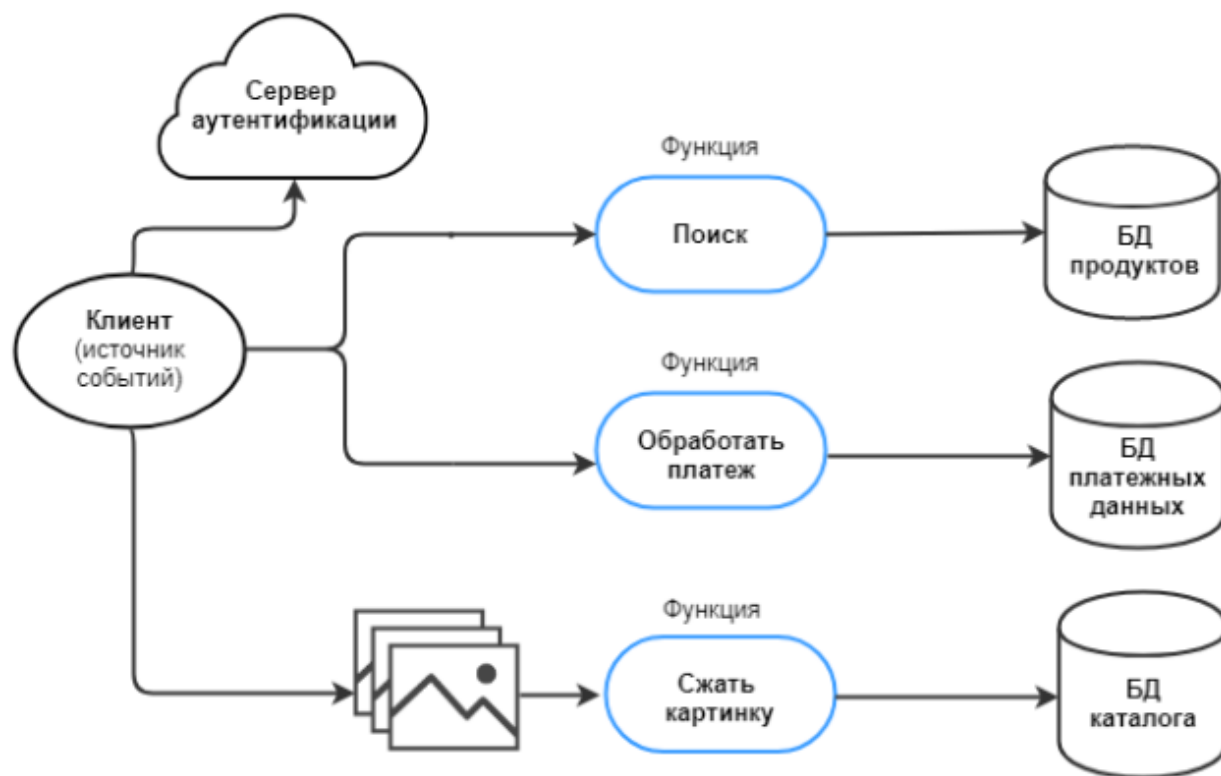


Рисунок 11 -- Модель бессерверного приложения

2 РЕАЛИЗАЦИЯ КЛИЕНТ-СЕРВЕРНОГО ПРИЛОЖЕНИЯ

В данной главе описан процесс реализации задачи, поставленные в итоге исследования. Уделяется внимание анализу и выбору набора инструментов для разработки ПО, разбору данных, обрабатываемых приложениями, структуре и анализу разработанных приложений. Цель раздела состоит в описании решений, использованных в разработке программы.

2.1 Определение инструментов разработки

2.1.1 Выбор языка программирования и среды разработки ПО

При создании модели бессерверного приложения нам необходимо обеспечить совместимость с большинством имеющихся ОС, возможность создания графического интерфейса и поддержка многопоточности для обеспечения одновременного выполнения множества задач на отдельной машине.

В связи с этим было принято решение реализовать программу, используя язык программирования C++ вместе с фреймворком Qt. Qt доступно на множестве домашних (Microsoft Windows, Mac OSX, Linux и др.) и мобильных ОС (Android, IOS).

Помимо этого, в платформе Qt доступна графическая библиотека, позволяющая создать графический интерфейс. Графические элементы можно создавать как прямо в коде, так и через программу QtDesigner, где можно добавить и скомпоновать виджеты и наблюдать за ними в реальном времени.

В пакете Qt5 также входит модуль сетевого взаимодействия QtNetwork, позволяющий обеспечить сетевое взаимодействие между двумя узлами сети вне зависимости от платформы, на которой эти два узла работают. Поддержка многопоточности в Qt позволяет открывать, держать и одновременно обрабатывать множество соединений с различными устройствами.

Уникальной чертой фреймворка Qt является механизм «Сигнал-слот», позволяющий связывать методы между собой и вызывать их в асинхронном режиме. Qt также заботится о том, чтобы сигналы и слоты могли связаться друг с другом и передавать управление на соответствующий поток. Сигнал является меткой, при вызове которого вызываются все слоты, соединённые с данным сигналом. Слоты, помимо вызова через сигнал, ничем не отличаются от обычного метода класса. [6]

На рисунке 12 представлен пример работы модели «сигнал-слот».

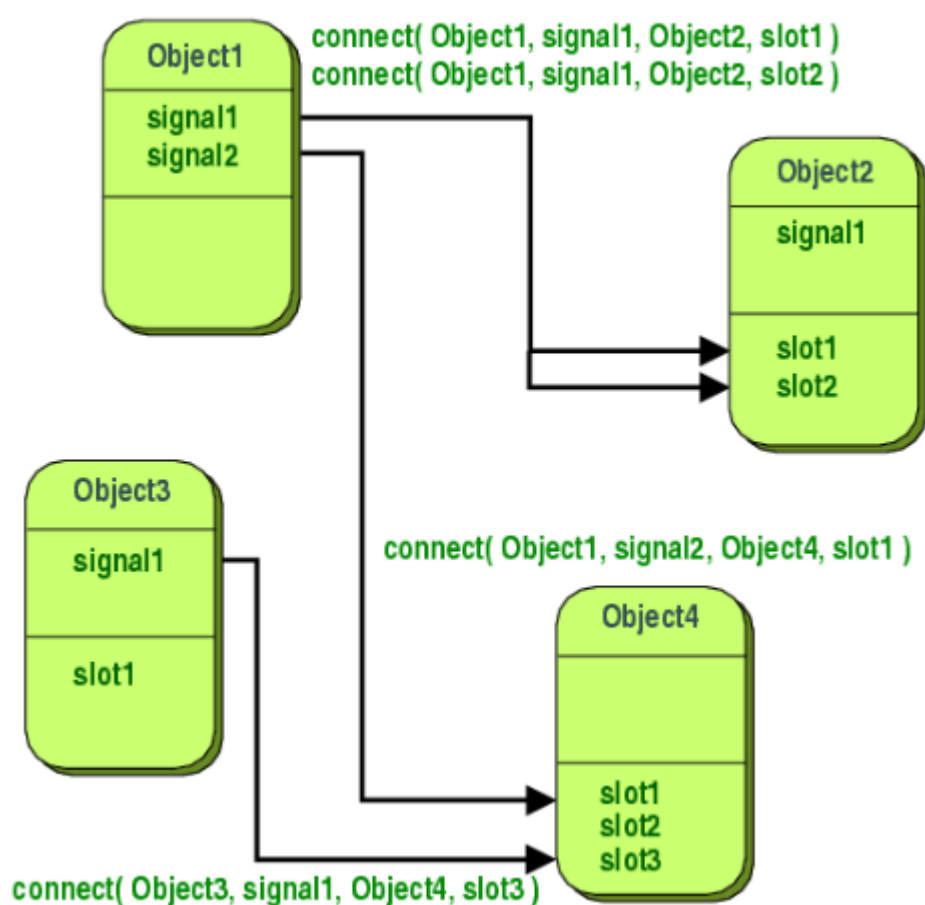


Рисунок 12 -- Пример организации сигналов и слотов в Qt

Средой разработки была выбрана *QtCreator*, т.к. эта среда разработки была специально создана для разработки приложений, использующих фреймворк Qt. Используются библиотеки *Qt* версии *5.15*. В качестве языка программирования будет использован язык *C++*, который создан для разработки множества видов приложений, в том числе клиентских и серверных, что нам необходимо в текущей задаче.

2.1.2 Выбор вида хранимых данных и инструментов для их хранения

Далее нам необходимо определить, где мы собираемся хранить данные, с которыми мы будем вести работу. Для нашей модели использовать базу данных не целесообразно, так как схему алгоритма можно представить в виде простого списка. Поэтому данные о схеме реализуемого алгоритма будет храниться в текстовом файле.

2.2 Структура данных в приложении

2.2.1 Описание функциональных единиц, используемых в алгоритме

Основными функциональными единицами в распределённом алгоритме являются:

- 1) Приложение. Выполняет одну уникальную функцию приложения;
- 2) Результат приложения. Передаётся клиенту либо другому приложению, если в этом имеется необходимость
- 3) Линейный переход. Безусловная передача результата и управления одному приложению;
- 4) Условный переход. Передача результата и управления на выполнение одному из двух узлов в зависимости от результата, полученного при выполнении предыдущего в цепочке узла;
- 5) Параллельный переход. Одновременная передача результата и разрешения на выполнение нескольким узлам приложений распределённого приложения;
- 6) Ожидающий переход. Принимающему узлу передаётся управление только тогда, когда все этому отправили свой результат и разрешение выполнения все предшествующие ему приложения

Таким образом, из указанных выше функциональных единиц можно построить модели функциональных алгоритмов.

2.2.2 Примеры моделей распределённого алгоритма

В примерах моделей распределённого алгоритма используются следующие обозначения:

- A_i – узел-приложение, выполняющее определённую функцию;
- Result – результат выполнения узла-приложения;
- Result:_true – результат, сообщающий об успешном выполнении узла-приложения;
- Result:_false – результат, сообщающий о выполнении узла-приложения с ошибкой;
- Стрелки – объект связи выполненного приложения и результата, а также результата и узла-приложения, следующего следующим в цепочке выполнения.

На Рисунок 13 показан пример распределённого алгоритма, использующего условный переход. Переходы с узлов A_0 и A_2 линейные, а с узла A_1 – условный: если в A_1 получил успешный результат, то он переходит на узел A_2 , в противном случае A_1 повторяет своё выполнение.

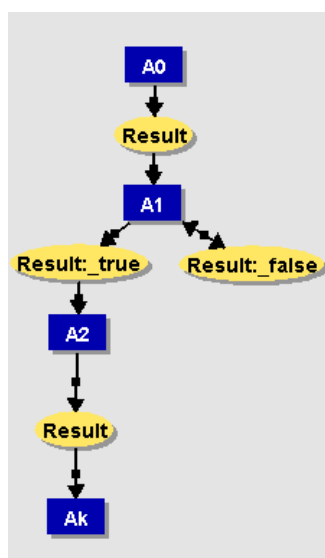


Рисунок 13 – Пример распределённого алгоритма с условным переходом

На Рисунок 14 показан пример распределённого алгоритма, использующего параллельный переход с ожидающим переходом. Переход с узла A1 – параллельный на узлы A2, A3, A4. При завершении выполнения работы всех трёх узлов срабатывает ожидающий переход на Ak.

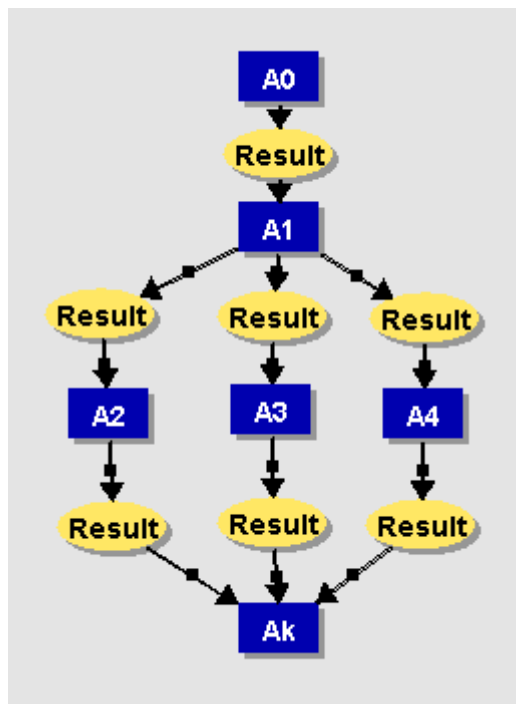


Рисунок 14 – Пример распределённого алгоритма с условным переходом

2.2.3 Формальное представление распределённого алгоритма в бессерверной системе

Для обеспечения стабильной работы приложения нам необходимо знать сведения обо всех узлах-приложениях, входящих в распределённую систему. Минимально необходимый набор сведений об одном узле-приложении состоит из:

- данные для присоединения к узлу-приложению(ip-адрес узла);
- сведения, от каких родительских узлов узел может получить результаты вычислений;
- сведения, на какие дочерние узлы, каким образом и при каком условии надо передавать результат своих вычислений.

Таким образом, был разработан следующий формат хранения данных о конфигурации одного узла распределённого алгоритма:

[Список родительских узлов];[Список дочерних узлов];<Адрес узла> (1)

где:

- [Список родительских узлов] – набор узлов-приложений, разделённых между собой запятой;
- [Список дочерних узлов] – набор узлов-приложений, разделённых между собой запятой, либо два узла-приложения, выполнение которых зависит от результата выполнения текущего узла, перечисленные через знак «/».
- <Адрес узла> – запись IP-адреса и порта узла-приложения через двоеточие.

Примером записи распределённого алгоритма, аналогичного модели из Рисунок 13 можно увидеть ниже:

;A1;192.168.0.2

A0;A2/A1; 192.168.1.3

A1;Ak; 192.168.0.6

Ak;;192.168.0.2

2.3 Структура приложений

Разрабатываемая система состоит из 2 компонент: клиентского и серверного приложений. Каждое приложение имеет отдельную структуру.

2.3.1 Структура клиентского приложения

Клиентское приложение устанавливается на устройство внутри локальной сети. Оно позволяет добавлять узлы-приложения, участвующие в выполнении распределённого алгоритма. Графический интерфейс позволяет добавить узлы алгоритма, наблюдать, какие узлы алгоритма уже включены в сеть и наблюдение за выполнением задачи, предоставленной узлу-приложению.

На рисунке 15 представлена UML-диаграмма вариантов использования клиентской программы.



Рисунок 15 - диаграмма вариантов использования клиентской программы

Программа-клиент управляет списком активных узлов-приложений. В зависимости от действий пользователя изменяют отображаемые узлы-приложения. Активные узлы выполняют свою задачу по следующему циклу работы:

1. Создание узла;
2. Подключение к серверу наблюдателю;
3. Запрос данных о выполняемой задаче;
4. Получение списка родительских и дочерних узлов;
5. Подключение к родительским узлам;
6. Создание подключений для дочерних узлов;
7. Ожидание сигнала к началу выполнения задачи;
8. Выполнение задачи;
9. Передача результата.

2.3.2 Структура серверного приложения

Серверное приложение обеспечивает связь с узлами-приложениями и передаче им данные о выполняемой задаче. Взаимодействие с узлами отображается в консоли приложения. Сервер работает автоматически и не может повлиять на работу сервера.

На рисунке 16 представлена UML-диаграмма вариантов использования серверной программы.

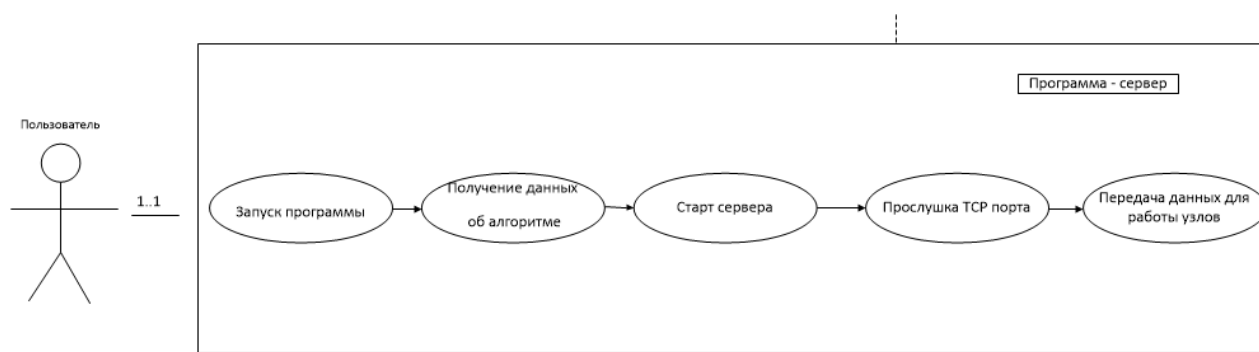


Рисунок 16 - диаграмма вариантов использования серверной программы

Программа-сервер после получения данных об алгоритме из файла, затем запускает сервер, при появлении подключения на котором отправляются данные для одного из участников алгоритма. При получении ответа о готовности от последнего в цепочке алгоритма узла запускает процесс выполнения алгоритма, получая результаты выполнения каждого из узлов-приложений.

2.4 Выбор архитектуры приложения и формирование поведения системы

Работу системы бессерверного вычисления распределённого алгоритма можно разделить на 2 этапа: сбор узлов приложений под схему алгоритма и выполнение распределённого алгоритма.

Сбор узлов приложений начинается с запуска приложения сервера-наблюдателя. При запуске он считывает файл конфигурации алгоритма и запускает сервер и прослушивает порт на наличие новых подключений.

В это время клиентское приложение создаёт новые узлы приложений. При создании, узел подключается к серверу и запрашивает данные для предстоящей задачи. Получив запрос, сервер получает от узла данные для подключения к нему и выдаёт ему данные, необходимые ему для решения задачи, получения результатов от узлов-родителей и отправки результата

выполнения задачи дочерним узлам. После этого узел ожидает получения команды на начало выполнения задачи.

Когда последний узел алгоритма получает свои данные, он отправляет серверу сообщение о готовности к вычислениям. Сервер, получив данные о готовности работы, посылает первому узлу сигнал о начале выполнения алгоритма.

При получении сигнала о начале выполнения задачи узел запускает свою задачу, ожидает её завершения и отправляет результат её выполнения на дочерние узлы вместе с сигналом о начале выполнения задачи. Вместе с этим результат операции передаётся на сервер.

Последний узел в цепочке распределённого алгоритма при завершении выполнения своей задачи отправляет результат своей работы на сервер, а после сообщение о завершении работы алгоритма.

Сервер при получении результатов вычислений выводит их в консоль. При завершении вычислений все имеющиеся узлы снова способны выполнить вычисления по распределённому алгоритму.

2.5 Описание процесса разработки системы

Как уже упоминалось ранее, выбранным языком программирования ПО является C++, а средой разработки — QtCreator. Клиентское приложение была разработана с Графическим интерфейсом при помощи пакета QtWidgets, а сетевое взаимодействие было реализовано при помощи пакета QtNetwork, входящих в состав библиотек Qt5. Серверная часть ПО была разработана как консольное приложение.

2.5.1 Описание разработки клиентского приложения

Клиентская программа разделена на 2 части: представления для отображения списка узлов-приложений и функционала самих узлов.

Представление обеспечивает взаимодействие с пользователем, отображая необходимые данные. Узел-приложение должен получить данные о задаче от сервера и выполнить свою задачу при получении сигнала на выполнение. Узел также хранит сведения от сервера и запускает сервер для подключения к нему дочерних узлов. При подключении к имеющемуся родительскому узлу на экран должно выводиться сообщение о подключении к родительскому узлу. При выполнении задачи узлом должно появляться соответствующее сообщение.

Таким образом, необходимо в клиентском приложении отображаются 3 графических элемента:

1. Окно управления узлами;
2. Диалоговое окно о подключении к родительскому узлу;
3. Окно выполнения задачи.

Окно управления узлами состоит из списка имеющихся в приложении узлов и панели кнопок. Панель кнопок включает в себя следующие кнопки:

- Добавить узел. Добавляет узел в список узлов и создаёт экземпляр узла-приложения;
- Удалить узел. Удаляет выбранный в списке экземпляр узла-приложения из списка узлов.

При нажатии на одну из кнопок, содержимое списка узлов меняется.

Список узлов управляется менеджером узлов. При нажатии кнопки добавления узла менеджер создаёт экземпляр класса узла-приложения. При нажатии кнопки удаления узла менеджер убирает экземпляр класса узла-приложения из списка и удаляет его.

Диалоговое окно о подключении к родительскому узлу предоставляет информацию о родительском узле, к которому подключился текущее приложение. Эта информация состоит из адреса родительского узла.

Окно выполнения задачи появляется при начале выполнении узлом своей задачи. Окно автоматически закрывается при завершении выполнения задачи. Окно не имеет элементов пользовательского интерфейса и лишь показывает,

что узел выполняет свою работу. Внутри окна выполнения задачи указывается, какой узел сейчас выполняет свою работу.

Узел-приложение, получая данные о задаче от сервера, сразу начинает их обрабатывать. Сначала он получает порядковый номер узла в цепи. Затем проверяется наличие адресов родительских узлов и подключение к ним, если таковые имеются. Затем проверяется наличие дочерних узлов. Если узел не является последним в очереди схемы распределённого алгоритма, то создаётся сервер подключения для дочерних узлов. Если имеются адреса дочерних серверов, то происходит подключение к дочерним узлам с запросом на подключение к серверу текущего узла. Если же узел является последним в очереди схемы распределённого алгоритма, то сервер узла-приложения не создаётся и отправляется пакет о готовности к выполнению распределённого алгоритма на сервер-наблюдатель.

Сервер узла при создании начинает прослушивать выделенный под него порт и ожидать подключений от дочерних узлов. При появлении нового подключения создаётся обработчик данного подключения и переносится выполняться в отдельный поток. При получении результата вычислений сервер отправляет результат вычислений обработчикам, которые смотрят, должны ли они выполнять отправку на узел, с которым они связаны.

Обработчик подключений сервера узла получает данные о том, при каких условиях отправлять пакет на дочерний узел от сервера узла и от дочернего узла на другом конце соединения. При получении результата вычисления узла он отправляет его на соответствующий ему дочерний сервер, если условия были выполнены.

Задача узла начинает выполняться при получении команды начать вычисления. При получении результата и типа результата полученные данные отправляются на сервер узла для передачи дочерним узлам(если таковой сервер имеется) и на сервер-наблюдатель.

На рисунке 17 показана UML-диаграмма классов окон, пользовательских элементов управления представления, и модели программы-клиента.

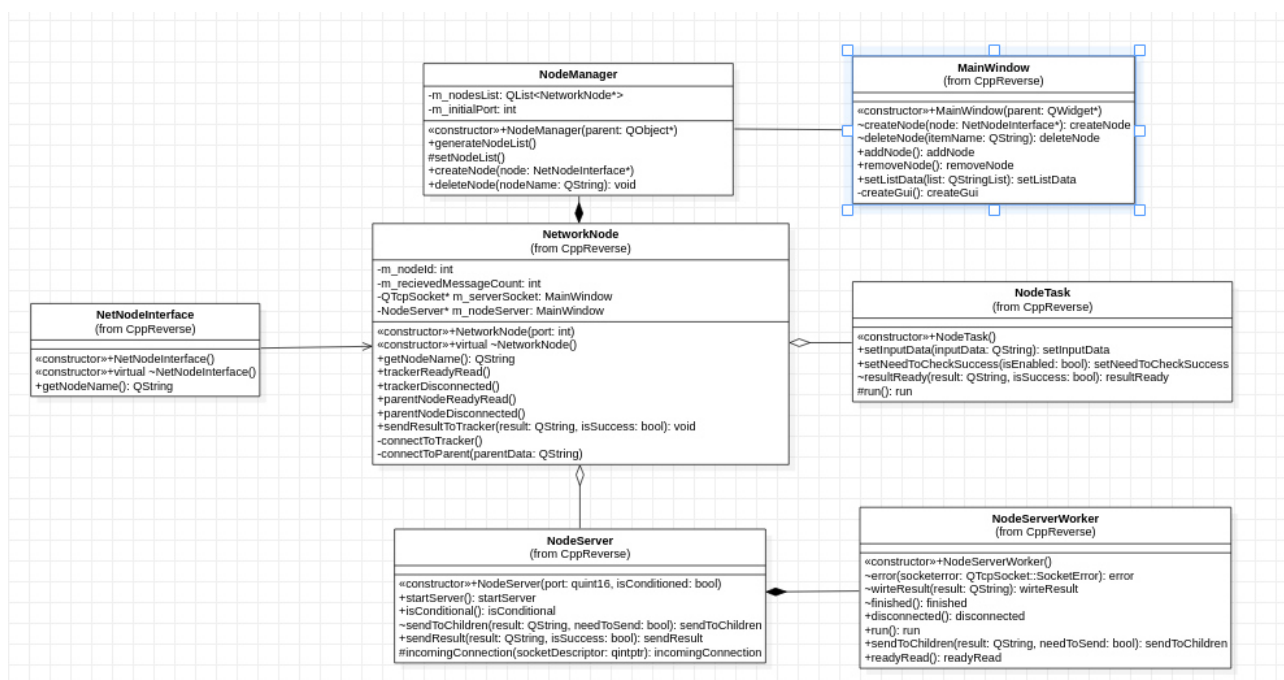


Рисунок 17 -- Диаграмма классов программы-клиента

2.5.2 Описание разработки серверного приложения

В свою очередь, серверная часть ПО спроектирована следующим образом: при запуске считывается файл с конфигурацией распределённого алгоритма. Содержимое файла в виде списка данных об узлах передаётся на сервер наблюдения за узлами-приложениями. Сервер при создании начинает прослушивать порт в ожидании появления соединения от узла.

В обработчике связи с узлами из клиентского приложения происходят по следующему циклу:

1. Сервер находится в состоянии ожидания;
2. Сервер получает запрос на новое соединение;
3. Сервер создаёт обработчик соединения в отдельной нити;
4. Формируется пакет с данными узла, вызвавшего соединение;
5. Обработчик ожидает сообщений от узла;

6. Обработчик получает запрос от узла на запрос данных для него;
7. Обработчик получает данные об узле;
8. Обработчик отправляет данные об узле на сервер;
9. Сервер записывает данные об узле в схему алгоритма;
10. Обработчик отправляет данные для выполнения задачи на узел;
11. Обработчик ожидает сообщения от пользователя;
12. Обработчик получает сообщение с результатом вычислений узла.

Если обработчик связан с последним узлом в схеме распределённого алгоритма, то обработчик должен получить от последнего узла сообщение о готовности к выполнению распределённого алгоритма. Это сообщение передаётся на сервер. После этого обработчик, связанный с первым узлом в схеме отправляет команду первому узлу начать вычисления.

На рисунке 18 представлена UML-диаграмма классов программы-сервера.

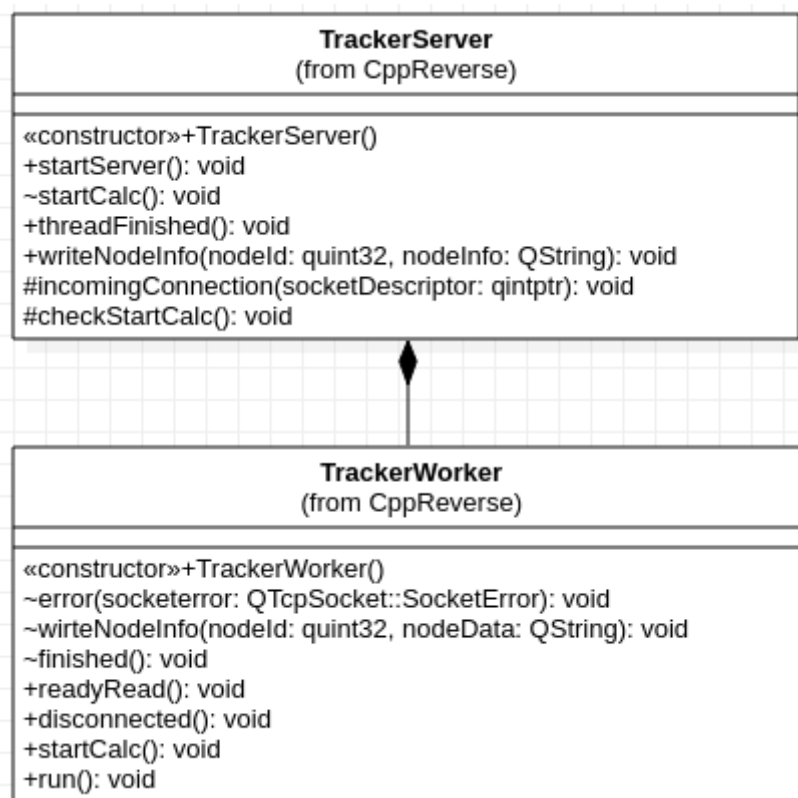


Рисунок 18 — диаграмма классов серверного приложения

3 ОПИСАНИЕ ПРОГРАММЫ

3.1 Описание клиентского приложения

Для запуска клиентского приложения пользователь по соответствующей иконке и названию запускает приложение со своего рабочего устройства ЛВС

Приложение может выполнять основные функции:

- Управлять списком узлов-приложений;
- Образовывать цепочку для вычисления распределённого алгоритма;
- Соединяться с сервером-наблюдателем для получения информации о требуемой задаче;
- Выполнять поставленные сервером-наблюдателем задачи.

Клиентскому приложению необходимо находиться в одной ЛВС с устройством, выполняющим программу сервера-наблюдателя. Далее пользователю необходимо добавить узлы-приложения, которые автоматически подключаются к серверу наблюдателю для получения информации о выполняемой задаче. С помощью полученных сведений узел автоматически подключается к нужным узлам для обеспечения связи в соответствии с распределённым алгоритмом. После чего узел ожидает соединения от дочерних узлов. Если созданный узел является последним в цепочке распределённого алгоритма, то он отправляет на сервер сигнал о начале выполнения алгоритма. Далее пользователь ожидает выполнения вычислений на узле, и если один из подключённых приложением узлов выполняется в данный момент времени то отображается окно с выполнением задачи выбранным узлом. При завершении вычислений узлом, данное окно пропадает.

Работа пользователя в клиентской программе осуществляется через графический интерфейс.

Описание основных событий клиентского приложения указано в таблице

1.

Таблица 1 – описание состояний программы-клиента

Событие	Действие Пользователя	Действие программы
Добавление узла	Нажата кнопка «Добавить»	Добавление узла-приложения в список узлов
Удаление узла	Нажата кнопка «Добавить»	Добавление узла-приложения из списка узлов
Подключение к родительскому узлу		Появление диалогового окна подключения к родительскому узлу
Выполнение задания		Появление окна выполнения задания
Выход из программы	Нажата кнопка «Выход»	Выход из приложения

Исходя из содержимого таблицы, можно отметить большинство событий вызываются автоматически после добавления узла.

На рисунке 19 изображено окно управления узлами приложения.

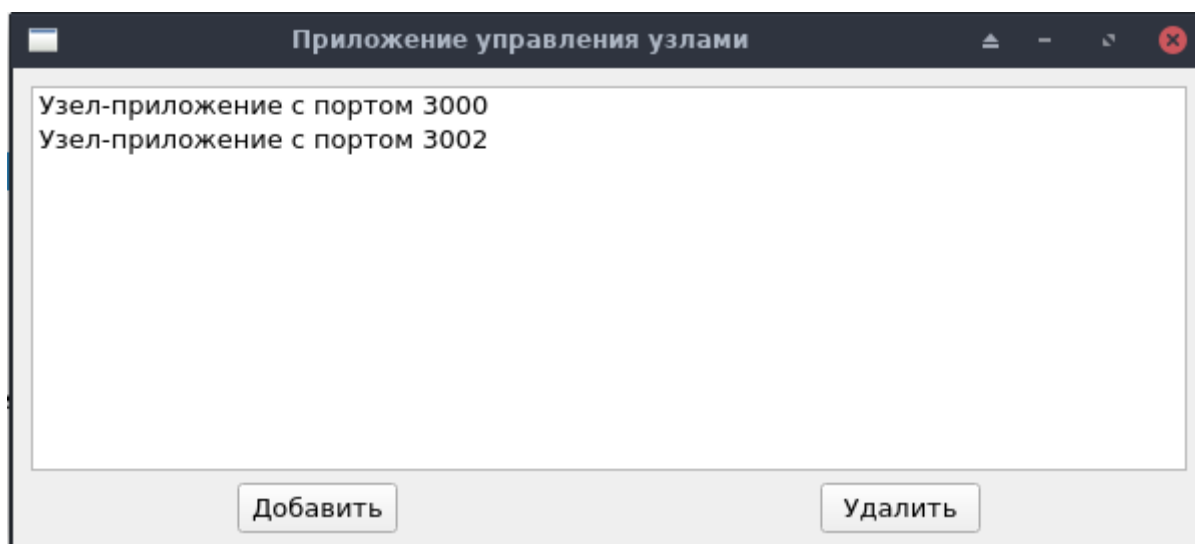


Рисунок 19 -- Окно управления узлами приложения

На рисунке 20 изображено диалоговое окно подключения к узлу.

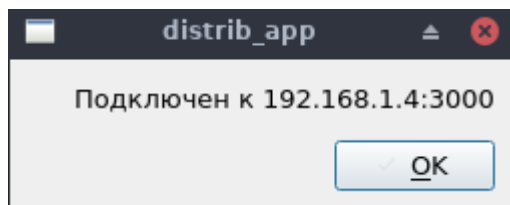


Рисунок 20 -- Окно подключения к узлу

На рисунке 21 изображено окно выполнения задачи распределённого алгоритма.

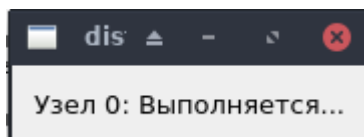


Рисунок 21 -- Окно подключения к узлу

3.2 Описание серверного приложения

После запуска сервера, программа считывает конфигурацию из файла, получая список для формирования распределённого алгоритма, а затем запускает сервер на соединение с узлами-приложениями.

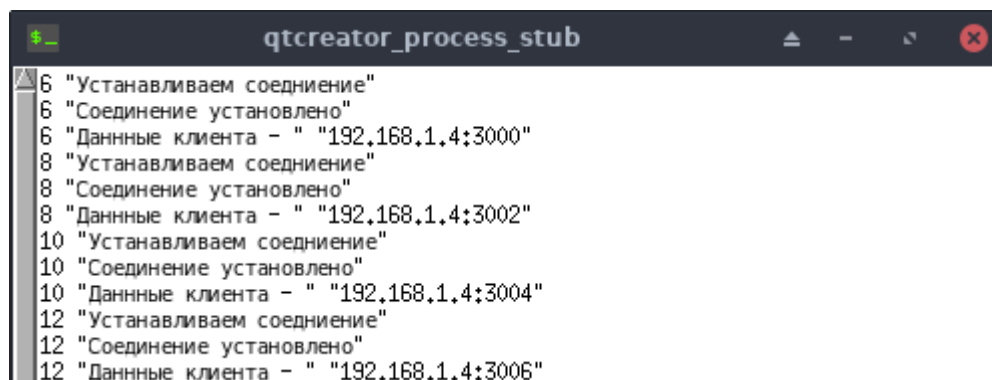
Описание всех событий сервера указано в таблице 2.

Таблица 2 - описание состояний программы-сервера

Событие	Действие Пользователя	Действие программы
Начало прослушивания порта	Запуск программы	Чтение файла конфигурации и запуск сервера с прослушиванием порта 1234
Подключение клиента	-	Отправка обработчика клиентского узла в отдельный поток
Получение адреса узла	-	Отправка сообщения с данными для выполнения задачи узла с соответствующими параметрами
Получение сигнала о готовности системы	-	Отправка сигнала начала вычислений на первый узел схемы распределённого алгоритма
Получение	-	Вывод результатов работы узла в консоль

результатов		
Выход	Нажата кнопка "Выход"	Выход из приложения

На рисунке 22 изображено окно консоли серверного приложения.



```

6 "Устанавливаем соединение"
6 "Соединение установлено"
6 "Данные клиента - " "192.168.1.4:3000"
8 "Устанавливаем соединение"
8 "Соединение установлено"
8 "Данные клиента - " "192.168.1.4:3002"
10 "Устанавливаем соединение"
10 "Соединение установлено"
10 "Данные клиента - " "192.168.1.4:3004"
12 "Устанавливаем соединение"
12 "Соединение установлено"
12 "Данные клиента - " "192.168.1.4:3006"

```

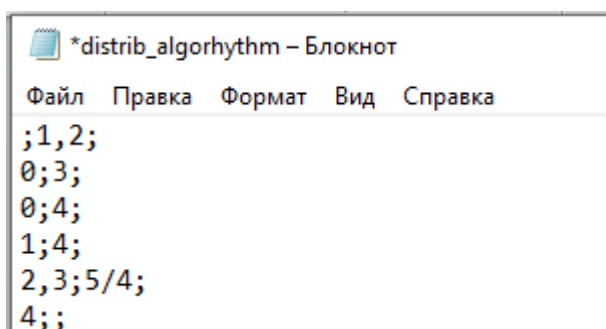
Рисунок 22 -- Окно подключения к узлу

Как видно из таблицы, и диаграммы активности, серверное приложение выполняет свою работу автоматически и взаимодействие с пользователем/администратором сети для её работы не требуется.

3.3 Отладка и тестирование системы

После реализации ПО было решено провести тестирование в среде разработки QtCreator.

Тестирование проводилось на наборе данных, представленном на рисунке 23.



```

;1,2;
0;3;
0;4;
1;4;
2,3;5/4;
4;;

```

Рисунок 23 -- тестовая конфигурация

На рисунке 24 изображено графическое отображение схемы, указанной в текстовом виде.

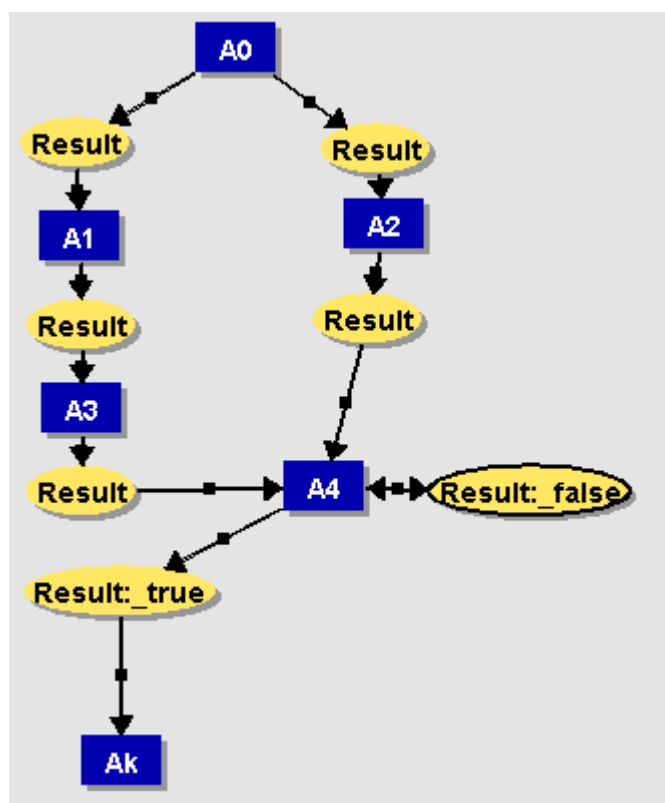


Рисунок 24 -- графическое отображение схемы тестового алгоритма

На рисунке 25 изображены результаты вычислений тестового набора схемы распределённого алгоритма.

```

"Начинаем работу алгоритма"
"[ср июн, 17 23:01:36 2020]Узел 0:Завершён"
"[ср июн, 17 23:01:38 2020]Узел 1:Завершён"
"[ср июн, 17 23:01:38 2020]Узел 2:Завершён"
"[ср июн, 17 23:01:40 2020]Узел 3:Завершён"
"[ср июн, 17 23:01:44 2020]Узел 4:Завершён с ошибкой"
"[ср июн, 17 23:01:46 2020]Узел 4:Завершён"
"[ср июн, 17 23:01:48 2020]Узел 5:Завершён"
"Конец работы алгоритма"

"Начинаем работу алгоритма"
"[ср июн, 17 22:59:20 2020]Узел 0:Завершён"
"[ср июн, 17 22:59:22 2020]Узел 1:Завершён"
"[ср июн, 17 22:59:22 2020]Узел 2:Завершён"
"[ср июн, 17 22:59:24 2020]Узел 3:Завершён"
"[ср июн, 17 22:59:26 2020]Узел 4:Завершён"
"[ср июн, 17 22:59:28 2020]Узел 5:Завершён"
"Конец работы алгоритма"
  
```

Рисунок 25 -- Результаты вычисления тестового распределённого алгоритма

Результаты тестирования совпали с ожидаемым результатом. Тестирование можно считать успешно выполненным.

3.4 Возможные пути последующих исследований

В качестве возможных модификаций программной модели можно отметить:

- Добавление возможности ручного подключения узлов к серверу-наблюдателю;
- Добавление в создание узла типа выполняемого приложения;
- Добавление проверки соединения узлом с сервером наблюдателем перед началом вычислений;
- Улучшение интерфейса между сервером и узлами для предоставления большего объема данных и возможностей работы с узлами и сервером;
- Добавление возможности работы по сети Интернет;
- Возможность поиска сервера по всей ЛВС.

ЗАКЛЮЧЕНИЕ

Результатом работы является проект бессерверного приложения, выполняющего распределённый алгоритм.

Программа написана на языке C++ при помощи фреймворка Qt версии 5 и упакована в архив с файлами, который можно распаковать на любом компьютере имеющейся программой разархивирования. Данный проект достиг целей, указанных при постановке задачи.

К особенностям проекта можно отнести его полуавтоматическую работу, простоту конфигурации и быстрое развёртывание на множестве настольных ОС.

В процессе отладки и тестирования были обнаружены ошибки и недочёты, которые были своевременно исправлены. При повторном выполнении тестирования системы эти ошибки не воспроизводились.

Работа над проектом позволила расширить знания и улучшить навыки с фреймворком Qt.

В текущем виде данное решение не позволяет конкурировать с имеющимися решениями в среде бессерверных вычислений и требует существенных доработок.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Олифер В., Олифер Н. Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов. 5-е изд. — СПб.: Питер, 2016. — 992 с.
2. В.П. Галас. Вычислительные системы, сети и телекоммуникации. Учебник в двух частях. Часть 2. Сети и телекоммуникации – ВлГУ, 2017 – 284 с.
3. Г.И. Радченко Распределенные вычислительные системы. Учебное пособие – Челябинск: Фотохудожник, 2012. – 184 с.
4. I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski, P. Suter. Serverless Computing: Current Trends and Open Problems, 2017 – 20 с.
5. С. Джиджи Осваиваем Kubernetes. Оркестрация контейнерных архитектур. — СПб.: Питер, 2019. — 400 с.
6. Qt Documentation: Signals and slots [Электронный ресурс] – статья с сайта Qt. URL: <https://doc.qt.io/qt-5/signalsandslots.html>
7. Н. Стародубцев. Serverless по стоечкам [Электронный ресурс] – статья с сайта Selectel. URL: <https://selectel.ru/blog/serverless-po-stoechkam/>
8. А. Сидоров. Бессерверные вычисления: что это такое и почему это важно [Электронный ресурс] – статья с сайта ItWeek URL: <https://www.itweek.ru/its/article/detail.php?ID=200238>
9. И. Д. Тугарев. Информационные технологии в науке и образовании: Обеспечение автоматической развёртки распределённого алгоритма для локальной вычислительной сети.— Изд. ПГУ, 2020 – с. 25-27
10. А. Барский. Grid-вычисления. – М.: LAP Lambert Academic Publishing, 2012. – 348 с.
11. Б. Амиршахи. Оптимизация Межресурсного Обмена в Распределённых GRID-Вычислениях. – М.: LAP Lambert Academic Publishing, 2012. – 120 с.
12. П. Фингар. Dot.Cloud: облачные вычисления - бизнес-платформа XXI века. – М.: Акваринариевая Книга, 2011. – 256 с.

13. Ч. Белл, М. Киндал, Л. Талманн. Обеспечение высокой доступности систем на основе MySQL. – М.: Русская Редакция, БХВ-Петербург, 2011. – 624 с.
14. Т. Редкар, Т. Гвидичи. Платформа Windows Azure. – М.: ДМК Пресс, 2012. – 656 с.
15. В.О. Сафонов. Платформа облачных вычислений Microsoft Windows Azure. – М.: Интернет-университет информационных технологий, Бином. Лаборатория знаний, 2013. – 240 с.
16. Э. Таненбаум, Х. Бос. Современные операционные системы. – СПб.: Питер, 2016. – 1120 с.
17. Л. Гаврилов. Интеллектуальные решения в бизнесе и коммерции. – М.: Palmarium Academic Publishing, 2014. – 160 с.
18. А. Герасимов. «Облака» как целевая бизнес-модель трансформации глобальной экономики. – М.: LAP Lambert Academic Publishing, 2014. – 64 с.
19. С. Пономарёв. Теория графов и параллельные вычисления в нефтегазовом комплексе. – М.: LAP Lambert Academic Publishing, 2014. – 84 с.
20. И. Баженова. Разработка приложений баз данных для облачных хранилищ данных. – М.: LAP Lambert Academic Publishing, 2013. – 212 с.
21. В. Шумский. GPU и Cloud Рендеринг. – М.: LAP Lambert Academic Publishing, 2013. – 108 с.
22. С. Макаров. Облачные вычисления. – М.: LAP Lambert Academic Publishing, 2012. – 172 с.
23. В.В. Цехановский, В.Д. Чертовской. Управление данными. Учебник. – СПб.: Лань, 2015. – 432 с.
24. Д.Л. Хеннеси, Д.А. Паттерсон. Компьютерная архитектура. Количественный подход. – М.: , 2016. – 936 с.
25. О.А. Кашелкина, М.А. Круглова, А.А. Макарова, Л.Б. Саратовская. "Computational Thinking" / Компьютерное мышление. – М.: Аргмак-Медиа, 2015. – 120 с.

ПРИЛОЖЕНИЕ А

ЛИСТИНГИ СЕРВЕРА

Приложение А.1

Файл main.cpp

```
#include <QCoreApplication>
#include <QFile>

#include "trackerserver.h"

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    QFile listFile("distrib_algorhythm");
    listFile.open(QIODevice::ReadOnly);
    QString readBuffer = listFile.readAll();
    QStringList list = readBuffer.split('\n');
    list.removeAll("");
    TrackerServer server(list);
    server.startServer();

    return a.exec();
}
```

Приложение А.2

Файл trackerserver.h

```
#ifndef TRACKERSERVER_H
#define TRACKERSERVER_H

#include <QTcpServer>
```

```

#include <QThreadPool>

class TrackerServer : public QTcpServer
{
    Q_OBJECT
public:
    explicit TrackerServer(QStringList& list, QObject *parent
= 0);
    void startServer();

signals:
    void startCalc();

public slots:
    void threadFinished();
    void writeNodeInfo(quint32 nodeId, QString nodeInfo);

protected:
    void incomingConnection(qintptr socketDescriptor);
    void checkStartCalc();

private:
    QStringList m_nodeList;
    quint32 m_connetedNodeCount;
    quint32 m_readyNodeCount;
};

#endif // TRACKERSERVER_H

```

Приложение A.3

Файл trackerserver.cpp

```
#include "trackerserver.h"
#include "trackerworker.h"

const quint16 SERVER_PORT = 1234;

TrackerServer::TrackerServer(QStringList& list, QObject
*parent) :
    QTcpServer(parent), m_nodeList(list),
m_connectedNodeCount(0), m_readyNodeCount(0)
{
}

void TrackerServer::startServer()
{
    if(!this->listen(QHostAddress::Any, SERVER_PORT))
    {
        qDebug() << tr("Невозможно создать сервер");
    }
}

void TrackerServer::incomingConnection(qintptr
socketDescriptor)
{
    qDebug() << socketDescriptor << tr("Устанавливаем
соединение");

    QStringList nodeInfo = this->m_nodeList.at(this-
>m_connectedNodeCount).split(';');
    QString finishedParentNodes;
    QString finishedChildNodes;
    if(nodeInfo.at(0) != "")
    {
```

```

        QStringList parentNodes = nodeInfo.at(0).split(',');

        for(QString parentNode: parentNodes)
        {
            int nodeId = parentNode.toInt();
            QStringList parentNodeInfo = this->m_nodeList.at(nodeId).split(';');
            finishedParentNodes += parentNodeInfo.at(2) +
            ',';
        }

        finishedParentNodes.truncate(finishedParentNodes.lastIndexOf(','))
        ;

        }
        if(nodeInfo.at(1) != "")
        {
            char splitSybmol = nodeInfo.at(1).contains('/') ? '/'
            : ',';

            QStringList childNodes =
            nodeInfo.at(1).split(splitSybmol);

            for(QString childNode: childNodes)
            {
                int nodeId = childNode.toInt();
                QStringList childNodeInfo = this->m_nodeList.at(nodeId).split(';');

                if(nodeInfo == childNodeInfo)
                {
                    finishedChildNodes += tr("*") + splitSybmol;
                }
                finishedChildNodes += childNodeInfo.at(2) +
            splitSybmol;
            }
        }
    }
}

```

```

finishedChildNodes.truncate(finishedChildNodes.lastIndexOf(splitSy
bmol));
    }
    else
    {
        finishedChildNodes = "x";
    }

    QThread* thread = new QThread();
    TrackerWorker *worker = new TrackerWorker(
        socketDescriptor, this->m_connetedNodeCount,
        finishedParentNodes + ";" +
finishedChildNodes);
    worker->moveToThread(thread);
    connect(worker, &TrackerWorker::wirteNodeInfo, this,
&TrackerServer::writeNodeInfo);
    connect(thread, &QThread::started, worker,
&TrackerWorker::run);
    connect(worker, &TrackerWorker::finished, thread,
&QThread::quit);
    connect(worker, &TrackerWorker::finished, worker,
&TrackerWorker::deleteLater);
    connect(thread, &QThread::finished, thread,
&QThread::deleteLater);
    connect(worker, &TrackerWorker::finished, this,
&TrackerServer::threadFinished);
    connect(this, &TrackerServer::startCalc, worker,
&TrackerWorker::startCalc);

    thread->start();
    this->m_connetedNodeCount++;

}

void TrackerServer::threadFinished()

```

```

{
    this->m_connetedNodeCount--;
    this->m_readyNodeCount--;
}

void TrackerServer::writeNodeInfo(quint32 nodeId, QString
nodeInfo)
{
    QString nodeDesc = this->m_nodeList.at(nodeId);
    nodeDesc += nodeInfo;
    this->m_nodeList.replace(nodeId,nodeDesc);
    this->m_readyNodeCount++;
    if(this->m_readyNodeCount == (quint32)this-
>m_nodeList.size())
    {
        emit startCalc();
    }
}

```

Приложение A.4

Файл trackerworker.h

```

#ifndef MYTHREAD_H
#define MYTHREAD_H

#include <QThread>
#include <QTcpSocket>
#include <QDebug>

class TrackerWorker : public QObject
{
    Q_OBJECT
public:
    explicit TrackerWorker(qintptr handle, quint32 nodeId,
QString nodeData, QObject *parent = 0);

```

```

signals:
    void error(QTcpSocket::SocketError socketerror);
    void writeNodeInfo(quint32 nodeId, QString nodeData);
    void finished();

public slots:
    void readyRead();
    void disconnected();
    void startCalc();
    void run();

private:
    QTcpSocket *m_clientSocket;
    qintptr m_handler;
    quint32 m_nodeId;
    QString m_nodeData;
};

#endif // MYTHREAD_H

```

Приложение A.5

Файл trackerworker.cpp

```

#include "trackerworker.h"

#include <QTimer>

TrackerWorker::TrackerWorker(qintptr handle, quint32 nodeId,
    QString nodeData, QObject *parent) :
    QObject(parent), m_nodeId(nodeId), m_nodeData(nodeData)
{
    this->m_handler = handle;
}

```

```

void TrackerWorker::run()
{
    this->m_clientSocket = new QTcpSocket();

    if(this->m_clientSocket->setSocketDescriptor(this-
>m_handler) == false)
    {
        emit error(this->m_clientSocket->error());
        return;
    }

    connect(this->m_clientSocket,      &QTcpSocket::readyRead,
this, &TrackerWorker::readyRead, Qt::DirectConnection);
    connect(this->m_clientSocket,      &QTcpSocket::disconnected,
this, &TrackerWorker::disconnected);

    qDebug() << this->m_handler << tr("Соединение
установлено");\
}

void TrackerWorker::readyRead()
{
    QString data = this->m_clientSocket->readAll();
    if (data.contains("hello") == true)
    {
        data.replace("hello", "");
        emit this->writeNodeInfo(this->m_nodeId, data);
        qDebug() << this->m_handler << tr("Данные клиента -
") << data;
        this->m_nodeData.replace("*", data);
        QString sendData = "Id:" + QString::number(this-
>m_nodeId) + ";" + this->m_nodeData;
        this->m_clientSocket-
>write(sendData.toUtf8().constData());
        return;
    }
}

```

```

    }

    qDebug() << data;
}

void TrackerWorker::disconnected()
{
    qDebug() << this->m_handler << tr("Закрытие соединения");

    this->m_clientSocket->deleteLater();
    emit finished();
}

void TrackerWorker::startCalc()
{
    if(this->m_nodeId != 0) return;
    qDebug() << tr("Начинаем работу алгоритма");
    this->m_clientSocket->write("Start calc");
}

```

ПРИЛОЖЕНИЕ Б

ЛИСТИНГИ ПРОГРАММЫ КЛИЕНТА

Приложение Б.1

Файл main.cpp

```
#include "mainwindow.h"
#include "nodemanager.h"

#include <QApplication>
#include <QObject>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    NodeManager manager;

    QObject::connect(&w, &MainWindow::createNode, &manager,
&NodeManager::createNode);
    QObject::connect(&w, &MainWindow::deleteNode, &manager,
&NodeManager::deleteNode);
    QObject::connect(&manager, &NodeManager::setNodeList, &w,
&MainWindow::setListData);

    w.show();
    return a.exec();
}
```

Приложение Б.2

Файл mainwindow.h

```
#ifndef MAINWINDOW_H
```

```

#define MAINWINDOW_H

#include <QMainWindow>
#include <QList>
#include <QListWidget>

#include "netnodeinterface.h"

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

signals:
    void createNode(NetNodeInterface* node);
    void deleteNode(QString itemName);

public slots:
    void addNode();
    void removeNode();
    void setListData(QStringList list);

private:
    void createGui();

    QList<NetNodeInterface*> m_nodeList;
    QListWidget m_nodeListWidget;

};

#endif // MAINWINDOW_H

```

Приложение Б.3

Файл mainwindow.cpp

```
#include "mainwindow.h"

#include <QFrame>
#include <QBoxLayout>
#include <QPushButton>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    this->m_nodeList.clear();

    createGui();
}

MainWindow::~MainWindow()
{
}

void MainWindow::createGui()
{
    QFrame* centerWidget = new QFrame(this);
    QVBoxLayout* mainLayout = new QVBoxLayout(centerWidget);

    mainLayout->addWidget(&this->m_nodeListWidget, 5);

    QHBoxLayout* buttonsLayout = new QHBoxLayout();

    QPushButton* addButton = new QPushButton(tr("Добавить"));
    connect(addButton, &QPushButton::clicked, this,
    &MainWindow::addNode);
    buttonsLayout->addWidget(addButton, 1, Qt::AlignCenter);
```

```

        QPushButton* deleteButton = new QPushButton(tr("Удалить"));
        connect(deleteButton, &QPushButton::clicked, this,
&MainWindow::removeNode);
        buttonsLayout->addWidget(deleteButton, 1, Qt::AlignCenter);

        mainLayout->addLayout(buttonsLayout);
        centerWidget->setLayout(mainLayout);
        this->setCentralWidget(centerWidget);
    }

void MainWindow::addNode()
{
    NetNodeInterface* node = nullptr;
    emit createNode(node);

    if(node == nullptr)
    {
        return;
    }

    this->m_nodeList.append(node);
    this->m_nodeListWidget.addItem(node->getNodeName());

    update();
}

void MainWindow::removeNode()
{
    int rowAtDeletion = this->m_nodeListWidget.currentRow();
    if( rowAtDeletion < 0)
    {
        return;
    }

    QListWidgetItem* item = this->
m_nodeListWidget.item(rowAtDeletion);

```

```

        this->m_nodeListWidget.removeItemWidget(item);
        emit deleteNode(item->text());

        update();
    }

void MainWindow::setListData(QStringList list)
{
    this->m_nodeListWidget.clear();
    this->m_nodeListWidget.addItem(list);
}

```

Приложение Б.4

Файл nodemanager.h

```

#ifndef NODEMANAGER_H
#define NODEMANAGER_H

#include <QObject>

#include "networknode.h"

class NodeManager : public QObject
{
    Q_OBJECT
public:
    explicit NodeManager(QObject *parent = nullptr);

    void generateNodeList();

signals:
    void setNodeList(QStringList list);

public slots:
    void createNode(NetNodeInterface* node);

```

```

        void deleteNode(QString nodeName);

private:
        QList<NetworkNode*> m_nodesList;

        int m_initialPort = 3000;
};

#endif // NODEMANAGER_H

```

Приложение Б.5

Файл nodemanager.cpp

```

#include "nodemanager.h"

NodeManager::NodeManager(QObject *parent) : QObject(parent)
{

}

void NodeManager::createNode(NetNodeInterface* node)
{
    node = new NetworkNode(this->m_initialPort);
    this->m_initialPort += 2;
    this->m_nodesList.append((NetworkNode*)node);

    generateNodeList();
}

void NodeManager::deleteNode(QString nodeName)
{
    for(auto node : this->m_nodesList)
    {
        if(node->getNodeName() == nodeName)
        {

```

```

        this->m_nodesList.removeOne(node);
        break;
    }
}

generateNodeList();
}

void NodeManager::generateNodeList()
{
    QStringList nodeList;

    for(auto node: this->m_nodesList)
    {
        nodeList.append(node->getNodeName());
    }

    emit setNodeList(nodeList);
}

```

Приложение Б.6

Файл netnodeinterface.h

```

#ifndef NETNODEINTERFACE_H
#define NETNODEINTERFACE_H

#include <QString>
#include <QObject>

class NetNodeInterface: public QObject{
    Q_OBJECT
public:
    NetNodeInterface() {}
    virtual ~NetNodeInterface() {}

```

```

        virtual QString getNodeName() = 0; //Отсюда надо получать все
        данные о узле

};

#endif // NETNODEINTERFACE_H

```

Приложение Б.7

Файл networknode.h

```

#ifndef NETWORKNODE_H
#define NETWORKNODE_H

#include "netnodeinterface.h"
#include "nodeserver.h"

#include <QObject>
#include <QTcpSocket>

class NetworkNode : public NetNodeInterface
{
public:
    NetworkNode(int port);
    virtual ~NetworkNode();

    QString getNodeName();

public slots:
    void trackerReadyRead();
    void trackerDisconnected();
    void parentNodeReadyRead();
    void parentNodeDisconnected();
    void sendResultToTracker(QString result, bool isSuccess =
true);

```

```

private:
    void connectToTracker();
    void connectToParent(QString parentData);

    int m_nodeId = 0;
    int m_nodePort;
    int m_recievedMessageCount = 0;
    QString m_NodeName;
    QTcpSocket* m_serverSocket = nullptr;
    QList<QTcpSocket*> m_parentNodeSockets;
    NodeServer* m_nodeServer = nullptr;
};

#endif // NETWORKNODE_H

```

Приложение Б.8

Файл networknode.cpp

```

#include "networknode.h"
#include "nodetask.h"

#include <QErrorMessage>
#include <QMessageBox>
#include <QNetworkInterface>

NetworkNode::NetworkNode(int port):
    m_nodePort(port)
{
    connectToTracker();
    this->m_parentNodeSockets.clear();
}

NetworkNode::~~NetworkNode()
{
    if(this->m_serverSocket != nullptr)

```

```

    {
        this->m_serverSocket->close();
    }
}

QString NetworkNode::getNodeName()
{
    return "test:" + QString::number(this->m_nodePort);
}

void NetworkNode::connectToTracker()
{
    this->m_serverSocket = new QTcpSocket(this);

    this->m_serverSocket->connectToHost("192.168.1.4", 1234);

    if(this->m_serverSocket->waitForConnected(3000))
    {
        connect(this->m_serverSocket, &QTcpSocket::readyRead,
                this, &NetworkNode::trackerReadyRead);
        connect(this->m_serverSocket, &QTcpSocket::disconnected,
                this, &NetworkNode::trackerDisconnected);

        QString fullAddress;

        for(auto address: QNetworkInterface::allAddresses())
        {
            if (address.protocol() ==
                QAbstractSocket::IPv4Protocol &&
                address !=
                QHostAddress(QHostAddress::LocalHost))
            {
                fullAddress = address.toString();
                break;
            }
        }
    }
}

```

```

        fullAddress += ":" + QString::number(this->m_nodePort);
        QString message = "hello" + fullAddress;

        this->m_serverSocket->write(message.toUtf8().constData());

        this->m_serverSocket->waitForBytesWritten(1000);

    }
    else
    {
        QErrorMessage* errorDialog = new QErrorMessage();
        errorDialog->showMessage(tr("Невозможно подключиться к
трекеру"));

    }
}

void NetworkNode::trackerReadyRead()
{
    QString trackerMessage = m_serverSocket->readAll();
    if(trackerMessage.contains("Start calc"))
    {
        NodeTask* task = new NodeTask(this->m_nodeId);
        task->setInputData("");
        task->setNeedToCheckSuccess();
        connect(task, &NodeTask::finished, task,
&NodeTask::deleteLater);
        connect(task, &NodeTask::resultReady, this->m_nodeServer,
&NodeServer::sendResult);
        connect(task, &NodeTask::resultReady, this,
&NetworkNode::sendResultToTracker);

        task->start();
    }
    else if(trackerMessage.contains("Id:"))

```

```

{
    QStringList trackerInfoList = trackerMessage.split(";");
    this->m_nodeId = trackerInfoList[0].mid(3).toInt();
    connectToParent(trackerInfoList[1]);
    if(trackerInfoList[2] != "x")
    {
        bool isConditional = trackerInfoList[2].contains("/");
        this->m_nodeServer = new NodeServer(this->m_nodePort,
isConditional);
        this->m_recievedMessageCount += isConditional;
        this->m_nodeServer->startServer();
        if(isConditional == true)
        {
            QStringList parentList =
trackerInfoList[2].split('/');
            for (int i = 0; i < parentList.size(); i++)
            {
                QString parentAddress = parentList.at(i);
                if(parentAddress == "") continue;
                QStringList parentAddressList =
parentAddress.split(':');
                QTcpSocket* newParent = new QTcpSocket(this);
                newParent->connectToHost(parentAddressList[0],
parentAddressList[1].toInt());

                if(newParent->waitForConnected(3000))
                {
                    connect(newParent, &QTcpSocket::readyRead,
                        this,
&NetworkNode::parentNodeReadyRead);
                    connect(newParent,
&QTcpSocket::disconnected,
                        this,
&NetworkNode::parentNodeDisconnected);
                    this->
m_parentNodeSockets.append(newParent);

```



```

        newParent->connectToHost(parentAddressList[0],
parentAddressList[1].toInt());

        if(newParent->waitForConnected(3000))
        {
            connect(newParent, &QTcpSocket::readyRead,
                    this, &NetworkNode::parentNodeReadyRead);
            connect(newParent, &QTcpSocket::disconnected,
                    this, &NetworkNode::parentNodeDisconnected);
            this->m_parentNodeSockets.append(newParent);

            newParent->write("true");
//            QMessageBox* msgBox = new QMessageBox();
//            msgBox->setText("Подключен к " + parentAddress);
//            msgBox->show();
        }
        else
        {
            QErrorMessage* errorDialog = new QErrorMessage();
            errorDialog->showMessage(tr("Невозможно подключиться к
серверу"));
            delete newParent;
        }
    }
}

void NetworkNode::parentNodeReadyRead()
{
    QTcpSocket* parentSocket = (QTcpSocket*) this->sender();
    static QString parentMessage;
    parentMessage += "\r\n" + parentSocket->readAll();
    this->m_recievedMessageCount++;
    if(this->m_recievedMessageCount != this-
>m_parentNodeSockets.size())
    {
        return;
    }
}

```

```

    }
    this->m_recievedMessageCount--;

    NodeTask* task = new NodeTask(this->m_nodeId);
    task->setInputData(parentMessage);
    if(this->m_nodeServer != nullptr)
    {
        task->setNeedToCheckSuccess(this->m_nodeServer-
>isConditional());
    }
    else task->setNeedToCheckSuccess();
    connect(task, &NodeTask::finished, task,
&NodeTask::deleteLater);
    if(this->m_nodeServer != nullptr)
    {
        connect(task, &NodeTask::resultReady, this->m_nodeServer,
&NodeServer::sendResult);
    }
    connect(task, &NodeTask::resultReady, this,
&NetworkNode::sendResultToTracker);
    task->start();
}

void NetworkNode::parentNodeDisconnected()
{
    QTcpSocket* parentSocket = (QTcpSocket*) this->sender();
    this->m_parentNodeSockets.removeOne(parentSocket);
}

void NetworkNode::sendResultToTracker(QString result, bool
isSuccess)
{
    Q_UNUSED(isSuccess)

    this->m_serverSocket->write(result.toUtf8());
}

```

```

        if(this->m_nodeServer == nullptr)
        {
            this->m_serverSocket->waitForBytesWritten();
            QString endCalc = "Конец работы алгоритма";
            this->m_serverSocket->write(endCalc.toUtf8());
        }
    }
}

```

Приложение Б.9

Файл nodetask.h

```

#ifndef NODETASK_H
#define NODETASK_H

#include <QObject>
#include <QThread>
#include <QString>

class NodeTask : public QThread
{
    Q_OBJECT
public:
    explicit NodeTask(quint32 id, QObject *parent = nullptr);
    void setInputData(QString inputData);
    void setNeedToCheckSuccess(bool isEnabled = false);

signals:
    int resultReady(QString result, bool isSuccess);

protected:
    void run();

private:
    quint32 m_nodeId;
    bool m_isNeedToCheckSuccess;

```

```

        QString m_inData;
        QString m_resultString;
    };

#endif // NODETASK_H

```

Приложение Б.10

Файл nodetask.cpp

```

#include "nodetask.h"

#include <QRandomGenerator>
#include <QFrame>
#include <QVBoxLayout>
#include <QLabel>
#include <QDateTime>

NodeTask::NodeTask(quint32 id, QObject *parent) :
    QThread(parent), m_nodeId(id)
{

}

void NodeTask::setInputData(QString inputData)
{
    this->m_inData = inputData;
}

void NodeTask::setNeedToCheckSuccess(bool isEnabled)
{
    this->m_isNeedToCheckSuccess = isEnabled;
}

void NodeTask::run()
{

```

```

    QString nodeString = "Узел " + QString::number(this->m_nodeId)
+ ":";

    QFrame* frame = new QFrame();
    QVBoxLayout* layout = new QVBoxLayout(frame);
    QLabel* processLabel = new QLabel(nodeString + "
Выполняется...");
    layout->addWidget(processLabel);
    frame->show();

    int seed = 0x12345;
    for (int i = 0; i < 10; i++)
    {
        seed = (QRandomGenerator::global()->bounded(0x10000) +
seed) / (i + 1);
        this->msleep(200);
    }

    bool result = seed % 2 == true;
    QString resultString;
    if(this->m_isNeedToCheckSuccess == false || result == true)
    {
        resultString = "Завершён";
    }
    else
    {
        resultString = "Завершён с ошибкой";
    }

    QString timestamp = QDateTime::currentDateTime().toString();

    QString taskResult = "[" + timestamp + "]" + nodeString +
resultString;

    emit resultReady(taskResult, result || !this-
>m_isNeedToCheckSuccess);

```

```
    delete processLabel;  
    delete layout;  
    delete frame;  
}
```

Приложение Б.11

Файл nodeserver.h

```
#ifndef NODESERVER_H  
#define NODESERVER_H  
  
#include <QObject>  
#include <QTcpServer>  
  
class NodeServer : public QTcpServer  
{  
    Q_OBJECT  
public:  
    NodeServer(quint16 port, bool isConditioned = false);  
    void startServer();  
    bool isConditional();  
  
signals:  
    void sendToChildren(QString result, bool needToSend);  
  
public slots:  
    void sendResult(QString result, bool isSuccess = true);  
  
protected:  
    void incomingConnection(qintptr socketDescriptor);  
  
private:
```

```

    QStringList m_resultList;
    quint32 m_connetedNodeCount;
    quint32 m_readyNodeCount;
    quint16 m_serverPort;
    bool m_isConditioned;
};

#endif // NODESERVER_H

```

Приложение Б.12

Файл nodeserver.cpp

```

#include "nodeserver.h"
#include "nodeserverworker.h"

#include <QTcpSocket>
#include <QThread>

NodeServer::NodeServer(quint16 port, bool isConditioned):
    m_connetedNodeCount(0),
    m_readyNodeCount(0), m_serverPort(port),
    m_isConditioned(isConditioned)
{

}

void NodeServer::startServer()
{
    if(!this->listen(QHostAddress::Any, this->m_serverPort))
    {
        qDebug() << tr("Невозможно создать сервер");
    }
}

void NodeServer::incomingConnection(qintptr socketDescriptor)

```

```

{
    NodeServerWorker* worker = new
NodeServerWorker(socketDescriptor, 0, this->m_isConditioned);
    QThread* thread = new QThread();

    connect(thread, &QThread::started, worker,
&NodeServerWorker::run);
    connect(worker, &NodeServerWorker::finished, thread,
&QThread::quit);
    connect(thread, &QThread::finished, thread,
&QThread::deleteLater);
    connect(worker, &NodeServerWorker::finished, worker,
&NodeServerWorker::deleteLater);
    connect(this, &NodeServer::sendToChildren, worker,
&NodeServerWorker::sendToChildren);

    worker->moveToThread(thread);
    thread->start();

    m_connetedNodeCount++;
}

void NodeServer::sendResult(QString result, bool isSuccess)
{
    emit sendToChildren(result, isSuccess);
}

bool NodeServer::isConditional()
{
    return this->m_isConditioned;
}

```

Приложение Б.13

Файл nodeserverworker.h

```
#ifndef NODESERVERWORKER_H
#define NODESERVERWORKER_H

#include <QObject>
#include <QTcpSocket>

class NodeServerWorker : public QObject
{
    Q_OBJECT
public:
    explicit NodeServerWorker(qintptr handle, quint32 nodeId, bool
isConditioned, QObject *parent = nullptr);

signals:
    void error(QTcpSocket::SocketError socketerror);
    void writeResult(QString result);
    void finished();

public slots:
    void disconnected();
    void run();
    void sendToChildren(QString result, bool needToSend);
    void readyRead();

private:
    QTcpSocket *m_clientSocket;
    qintptr m_handler;
    quint32 m_nodeId;
    QString m_nodeData;
    bool m_isConditioned;
    bool m_resultBranch;
```

```
};
```

```
#endif // NODESERVERWORKER_H
```

Приложение Б.14

Файл nodeserverworker.cpp

```
#include "nodeserverworker.h"
```

```
#include "nodetask.h"
```

```
NodeServerWorker::NodeServerWorker(qintptr handle, quint32 nodeId,  
bool isConditioned, QObject *parent) :
```

```
    QObject(parent), m_handler(handle), m_nodeId(nodeId),  
m_isConditioned(isConditioned)  
{
```

```
}
```

```
void NodeServerWorker::run()
```

```
{
```

```
    this->m_clientSocket = new QTcpSocket();
```

```
    if(this->m_clientSocket->setSocketDescriptor(this->m_handler)  
== false)
```

```
{
```

```
    emit error(this->m_clientSocket->error());
```

```
    return;
```

```
}
```

```
    connect(this->m_clientSocket, &QTcpSocket::readyRead, this,  
&NodeServerWorker::readyRead);
```

```
    connect(this->m_clientSocket, &QTcpSocket::disconnected, this,  
&NodeServerWorker::disconnected);
```

```
    qDebug() << this->m_handler << tr("Соединение установлено");
```

```
}
```

```

void NodeServerWorker::disconnected()
{
    qDebug() << this->m_handler << tr("Заккрытие соединения");

    this->m_clientSocket->deleteLater();
    emit finished();
}

void NodeServerWorker::sendToChildren(QString result, bool
needToSend)
{
    if(this->m_isConditioned == true && this->m_resultBranch !=
needToSend) return;
    this->m_clientSocket->write(result.toUtf8().constData());
}

void NodeServerWorker::readyRead()
{
    QString data = this->m_clientSocket->readAll();
    if(data == "true")
    {
        this->m_resultBranch = true;
    }
    else if(data == "false")
    {
        this->m_resultBranch = false;
    }
}

```

ПРИЛОЖЕНИЕ В

ПРЕЗЕНТАЦИЯ

Реализация распределённых алгоритмов на вычислительной сети

Выполнил: студент группы 18ВВм1 Тугарев И. Д.
Руководитель: д.т.н. Зинкин С. А.

Цель работы

- Разработка программы получения узлом локальной вычислительной сети сведений о текущей конфигурации сети под настроенный алгоритм, получить соответствующие указания для включения в цепочку выполнения распределённого алгоритма, а также обеспечить проверку выполнения алгоритма внутри сети.

Постановка задачи

- ПО должно хранить и передавать сведения схемы распределённого алгоритма
- Узлы-приложения, создаваемые ПО, должны подключаться к распределённой вычислительной сети согласно порядку в схеме
- ПО производить вычисления в порядке, указанном в схеме распределённого алгоритма

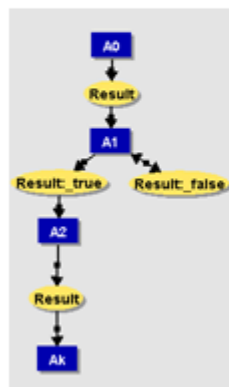
Используемые технологии

- Язык программирования – C++
- Используемый фреймворк – Qt5
- Среда разработки – QtCreator

Структура разрабатываемого приложения



Структура данных приложения



;A1;192.168.0.2

A0;A2/A1; 192.168.1.3

A1;Ak; 192.168.0.6

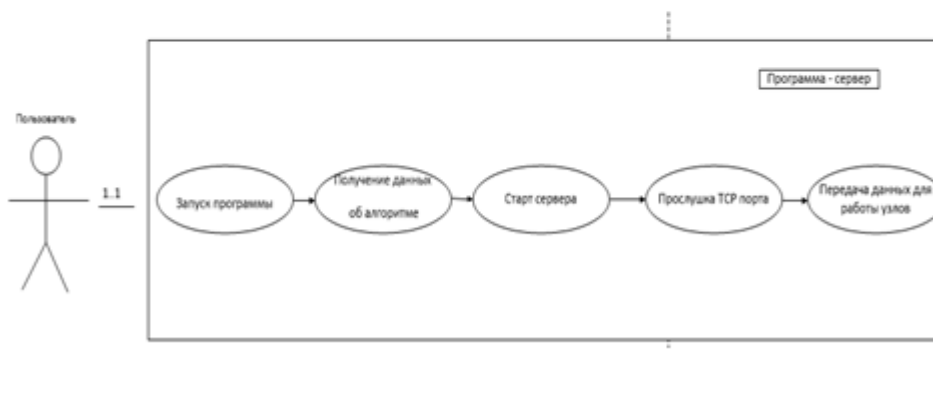
Ak;;192.168.0.2

```

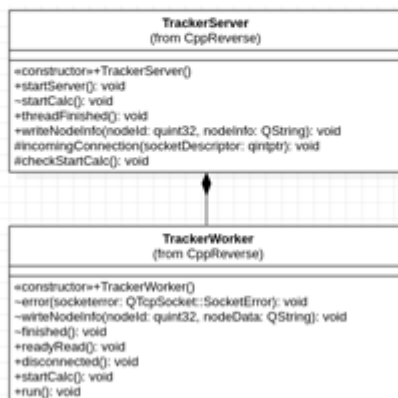
graph LR
    User[Пользователь] -- 1.1 --> Start([Запуск программы])
    Start --> Add([Добавление узла-приложения])
    Start --> Del([Удаление узла-приложения])
    Add --> ConnectServer([Подключение к серверу наблюдения])
    ConnectServer --> ConnectNodes([Подключение к родительским и дочерним узлам])
    ConnectNodes --> Execute([Выполнение соответствующей задачи])
    subgraph SystemBoundary [ ]
        Add
        Del
        ConnectServer
        ConnectNodes
        Execute
    end
    style SystemBoundary fill:none,stroke:#000,stroke-width:1px
    
```

[illegible]

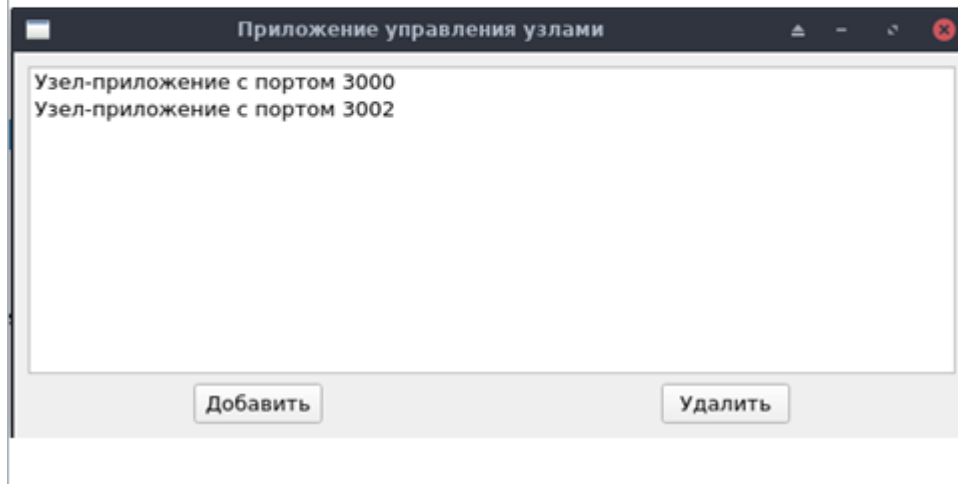
Варианты использования серверного приложения



Серверное приложение



Окно клиентского приложения



Пример получаемого результата

```
16 "Устанавливаем соединение"
16 "Соединение установлено"
16 "Данные клиента - " "192.168.1.4:3010"
"Начинаем работу алгоритма"
"[ср июн, 17 23:01:36 2020]Узел 0:Завершён"
"[ср июн, 17 23:01:38 2020]Узел 1:Завершён"
"[ср июн, 17 23:01:38 2020]Узел 2:Завершён"
"[ср июн, 17 23:01:40 2020]Узел 3:Завершён"
"[ср июн, 17 23:01:44 2020]Узел 4:Завершён с ошибкой"
"[ср июн, 17 23:01:46 2020]Узел 4:Завершён"
"[ср июн, 17 23:01:48 2020]Узел 5:Завершён"
"Конец работы алгоритма"

"Начинаем работу алгоритма"
"[ср июн, 17 22:59:20 2020]Узел 0:Завершён"
"[ср июн, 17 22:59:22 2020]Узел 1:Завершён"
"[ср июн, 17 22:59:22 2020]Узел 2:Завершён"
"[ср июн, 17 22:59:24 2020]Узел 3:Завершён"
"[ср июн, 17 22:59:26 2020]Узел 4:Завершён"
"[ср июн, 17 22:59:28 2020]Узел 5:Завершён"
"Конец работы алгоритма"
```

Результаты работы

- Проект выполнила поставленную задачу
- Работа над проектом укрепила навыки разработки приложений с распределёнными вычислениями
- Программе необходимы доработки для использования в производстве

Спасибо за внимание!