Федеральное государственное автономное образовательное
учреждение высшего образования
"Омский государственный технический университет"

На правах рукописи

Викулов Егор Олегович

МЕТОДЫ И АЛГОРИТМЫ РАСПРЕДЕЛЕНИЯ НАГРУЗКИ МЕЖДУ ВЫЧИСЛИТЕЛЬНЫМИ РЕСУРСАМИ ИНФОРМАЦИОННЫХ СИСТЕМ

Специальность: 2.3.1. Системный анализ,
управление и обработка информации, статистика

Диссертация на соискание ученой степени
кандидата технических наук

Научный руководитель
доктор технических наук, доцент
Денисова Людмила Альбертовна

Омск – 2024

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность проблемы. На сегодняшний день наблюдается значительный рост числа пользователей сети интернет, что приводит к возрастанию нагрузки на клиент-серверные информационные системы. Пользователи ожидают, что информационные системы будут работать быстро и без сбоев. Всё больше клиентов используют мобильные устройства для доступа в интернет, что также приводит к увеличению нагрузки. В свою очередь, клиент-серверные информационные системы предоставляют все больше информации, более качественный и высокоскоростной сервис, чтобы привлечь и удержать клиентов. Повышение требований пользователей к качеству обслуживания приводит к необходимости привлечения дополнительных вычислительных ресурсов для обработки и хранения данных. Для распределения вычислительной нагрузки между узлами (облачными ресурсами или физическими серверными станциями) часто используется механизм балансировки. Термин «балансировка нагрузки» (англ. load balancing) используется для обозначения, как перенаправления данных, получаемых от клиентов, на определённый узел системы, так и распределения вычислительной нагрузки приложений и вычислительных систем. Следует отметить, что в настоящее время в России действуют законодательные требования к обеспечению доступности государственных услуг в электронном виде (в частности требования Федерального закона №152 «О персональных данных»), которые позволяет выполнить балансировка нагрузки между ресурсами информационных систем. Таким образом, задача распределения нагрузки в клиент-серверных информационных системах в настоящее время является актуальной.

Степень разработанности темы исследования. Анализ исследований, посвященных решению задач распределения данных и вычислений в клиент-серверных информационных системах, указывает на существующую потребность в развитии методов и алгоритмов балансировки нагрузки, в том числе на основе интеллектуальных технологий обработки данных о состоянии вычислительных ресурсов.

Вопросам балансировки нагрузки посвящены работы следующих авторов: Лохвицкий В.А., Гончаренко В.А., Никишин К.И., Eslami G., Haghihat A., Farokhi S.,

Leland R. Beaumont, L. Qiu, V. Padmanabhan, G. Voelker, Membrey P., Plugge E., Hows D. В настоящее время учеными: Neves T., Drummond L., Ochi L., Albuquerque C., Uchoa E., Punetha Sarmila, G. Gnanambigai, N. Dinadayalan P., Sahoo J., Salahuddin M. A., Glitho R., Elbiaze H., Ajib W. разработан ряд методов и алгоритмов распределения данных и вычислительной нагрузки клиент-серверных информационных систем.

Анализ результатов, полученных зарубежными и отечественными учеными, показал, что исследования предметной области охватывают вопрос балансировки нагрузки в клиент-серверных информационных системах и содержат теоретические и практические разработки, однако они не учитывают или учитывают частично состояние аппаратных ресурсов и структуру сетей передачи данных.

Указанный недостаток объясняется высокой сложностью исследований влияния каждого отдельного параметра функционирования аппаратного обеспечения на производительность и время обработки запросов клиент-серверных информационных систем, а также возможностью масштабирования серверных аппаратных ресурсов, что обеспечивает увеличение производительности, но *существенно* увеличивает стоимость предоставляемых аппаратных ресурсов. Это в результате приводит к использованию методов и алгоритмов балансировки нагрузки, которые не могут обеспечить необходимую скорость доставки данных пользователю, либо существенно увеличивают затраты пользователей.

Видится перспективным использование данных о состоянии серверов для анализа и выбора наиболее подходящего пользователю вычислительного ресурса.

В диссертации показано, что задача выбора сервера может быть решена при помощи методов кластерного анализа и нечеткой логики. Исследованиями методов кластерного анализа и нечеткой логики посвящены работы таких ученых как Каллан Р., Kohonen T., Lakhmi C. Jain, Mamdani E.H., Zadeh L. A., Штовба С.Д., Хайкин С.

Обоснование выбора критериев функционирования и выбор параметров для дальнейшего проведения кластерного анализа и анализа с применением нечеткой логики целесообразно реализовать на основе хорошо развитых к настоящему времени методов анализа паттернов данных, представленных в работах Алескерова Ф.Т.,

Андрейчикова А. В., Мячина А. Л., Ersel H., Yolalan R., Few S., Mirkin B., Sanders P., Mehlhorn K., и других авторов в России и за рубежом.

На основе информации о недостаточном использовании данных о состоянии серверного комплекса (загруженность вычислительного ресурса), о пользователе (местоположение), состоянии сети передачи данных и стоимости услуг сделан вывод о целесообразности разработки методов и алгоритмов балансировки нагрузки клиент-серверных приложений и информационных систем с применением методов кластерного анализа и нечеткой логики.

Основная идея работы заключается в рациональном выборе вычислительного ресурса для хранения данных и проведения вычислений, основанных на анализе параметров состояния сервера.

Целью диссертационной работы является повышение быстродействия и производительности высоконагруженных клиент-серверных информационных систем путем оптимизации распределения вычислительной нагрузки и статических данных между серверами.

Для достижения указанной цели в работе поставлены и решены следующие **задачи**:

1. Анализ проблемы повышения производительности работы клиент-серверных информационных систем.

2. Обоснование показателей состояния вычислительных ресурсов на основе паттерн-анализа функционирования серверов для выбора сервера при распределении нагрузки.

3. Разработка методов и алгоритмов на основе кластерного анализа и нечеткого логического вывода, обеспечивающих повышение быстродействия и производительности систем распределения нагрузки между серверами в сравнении с известными методами в условиях неполноты данных о состоянии вычислительных ресурсов.

4. Разработка структуры программного комплекса и алгоритма параллельных вычислений для балансировки нагрузки, а также проведение экспериментальных исследований оценки эффективности распределения данных между ресурсами облачного серверного кластера.

Научная новизна. В процессе исследования получены следующие новые научные результаты:

1. разработан комбинированный метод формирования показателей и правил выбора сервера при балансировке нагрузки на основе данных о состоянии серверного комплекса. Отличительными особенностями метода является выделение паттернов параметров состояния серверов, позволяющее выявить значимые критерии выбора, влияющие на скорость обработки данных, а также последующая кластеризация запросов пользователей для формирования правил выбора сервера;

2. предложен аналитико-имитационный метод распределения вычислительной нагрузки с помощью нечеткого логического вывода, положенного в основу работы сервера-балансира. В отличие от существующих метод, включающий аналитическую обработку экспериментальных данных о состоянии вычислительных ресурсов в совокупности с модельными исследованиями, позволяет выбрать и обосновать параметры алгоритма балансировки, обеспечивая повышение быстродействия и отказоустойчивости высоконагруженной информационной системы;

3. разработан алгоритм параллельных вычислений для распределения данных между ресурсами облачного кластера серверов и структура программного комплекса для проведения экспериментальных исследований. Основным преимуществом данного алгоритма в отличие от существующих является возможность выполнения больших объемов вычислений в параллельном потоке, что повышает скорость доставки данных. Отличительной особенностью структуры программного комплекса является декомпозиция системы балансировки на параллельно работающие подсистемы (с выделением модуля принятия решений о выборе сервера), что увеличивает быстродействие системы и позволяет проводить эксперименты с облачными ресурсами.

Практическая значимость работы заключается в разработке:

1) программного комплекса (свидетельство о регистрации программы ЭВМ №2021618977), методики и алгоритма сбора и обработки параметров функционирования серверов, которые позволяют выявить закономерности в данных и обосновать правила выбора сервера для задачи распределения вычислительной

нагрузки;

2) программного комплекса (свидетельство о регистрации программы ЭВМ № 2022662254) параллельных вычислений распределения нагрузки облачных серверов;

3) аналитико-имитационного метода, позволяющего проводить тестирование и имитационное моделирование балансировки вычислительной нагрузки комплекса серверов.

Внедрение результатов исследований. Созданные программные комплексы сбора данных и распределения вычислительной нагрузки, использованные в разработке программных продуктов ООО «РОНАС ИТ», позволили повысить скорость работы, стабильность и отказоустойчивость разрабатываемых высоконагруженных информационных систем. Результаты исследований внедрены в учебный процесс ОмГТУ и используются в учебных дисциплинах кафедры «Автоматизированные системы управления и обработки информации» (АСОИУ).

Основные результаты, полученные автором и выносимые на защиту:

1) Комбинированный метод формирования показателей и правил выбора сервера при балансировке нагрузки на основе данных о состоянии серверного комплекса. Отличительными особенностями метода является выделение паттернов параметров состояния серверов, позволяющее выявить значимые критерии выбора, влияющие на скорость обработки данных, а также последующая кластеризация запросов пользователей для формирования правил выбора сервера.

2) Аналитико-имитационный метод распределения вычислительной нагрузки с помощью нечеткого логического вывода, положенного в основу работы сервера-балансира. В отличие от существующих метод, включающий аналитическую обработку экспериментальных данных о состоянии вычислительных ресурсов в совокупности с модельными исследованиями, позволяет выбрать и обосновать параметры алгоритма балансировки, обеспечивая повышение быстродействия высоконагруженной информационной системы.

3) Алгоритм параллельных вычислений для распределения данных между

ресурсами облачного кластера серверов и структура программного комплекса для проведения экспериментальных исследований. Основным преимуществом данного алгоритма в отличие от существующих является возможность выполнения больших объемов вычислений в параллельном потоке, что повышает скорость доставки данных. Отличительной особенностью структуры программного комплекса является декомпозиция системы балансировки на параллельно работающие подсистемы (с выделением модуля принятия решений о выборе сервера), что существенно увеличивает быстродействие системы и позволяет проводить эксперименты с облачными ресурсами.

Объектом являются способы распределения данных и вычислительной нагрузки в высоконагруженных информационных системах.

Предмет исследования – математические модели, методы и алгоритмы, предназначенные для анализа и оптимизации распределения нагрузки между вычислительными ресурсами информационных систем.

Методы исследования

При решении поставленных задач использовались методы анализа больших объемов данных, нейросетевые технологии, технологии кластерного анализа, технологии систем искусственного интеллекта, методы теории массового обслуживания.

Достоверность полученных результатов

Обоснованность и достоверность теоретических результатов, положений и выводов, полученных в диссертационной работе, базируется на использовании апробированных научных положений и методов исследования, корректном применении математического аппарата, согласованности новых результатов с известными теоретическими положениями.

Апробация работы

Результаты работы отражались в научных докладах, которые представлялись на: V, VII, VIII, X Всероссийских научно-практических конференциях «Информационные технологии и автоматизация управления» (г. Омск, 2013, 2016, 2017, 2019); на IX, XIII Международных IEEE конференциях «Динамика систем,

механизмов и машин» (г. Омск, 2014, 2019); на II, IV Международных научно-технических конференциях «Проблемы машиноведения» (г. Омск, 2018, 2020); на Международной научно-технической конференции «Пром-Инжиниринг» (г. Москва, 2018); на II Международной научно-технической конференции «Научный потенциал молодежи и технический прогресс» (г. Санкт-Петербург, 2019); на Международном семинаре «Передовые технологии в материаловедении, машиностроении и автоматизации» MIP-2019 (г. Красноярск, 2019).

Публикации по теме исследования

По теме диссертации опубликовано 19 научных работ, в том числе 4 научных статьи рецензируемых в научных изданиях, рекомендованных ВАК при Минобрнауки России, 5 научных статей в изданиях, индексируемых в международной реферативной базе данных Scopus, 2 свидетельства о государственной регистрации программ для ЭВМ.

Структура и объем диссертации

Диссертация состоит из введения, четырех глав, заключения, списка использованных источников (119 наименований) и двух приложений. Общий объем работы 135 страниц, в том числе 122 страницы основного текста, включая 50 рисунков и 21 таблицу.

Автор выражает глубокую благодарность научному руководителю д.т.н., доценту, профессору кафедры АСОИУ Денисовой Л.А., за помощь и многолетнюю поддержку при выполнении диссертационных исследований. Автор благодарит заведующего кафедрой АСОИУ, д.т.н., профессора Никонова А.В. за поддержку при выполнении работы.

1 ПРОБЛЕМЫ РАСПРЕДЕЛНИЯ ДАННЫХ И ВЫЧИСЛЕНИЙ МЕЖДУ РЕСУРСАМИ ВЫСОКОНАГРУЖЕННЫХ СИСТЕМ

В **первой** главе рассматривается проблема распределения вычислительной нагрузки между серверами вычислительного комплекса. Приводятся существующие методы и алгоритмы решения задачи балансировки нагрузки. Рассматриваются подходы к интеллектуальной обработке данных о состоянии серверов. Описывается метод имитационного моделирования и оптимизации нагрузки при решении задач балансировки.

1.1 Анализ современного состояния исследований в области балансировки вычислительной нагрузки информационных систем

Ввиду противоречивости требований, предъявляемых к клиент-серверным приложениям и информационным системам в части увеличения скорости доставки и количества передаваемых данных, возникает необходимость в создании дополнительного математического, алгоритмического и программного обеспечения. Разработчики клиент-серверных приложений и информационных систем создают новые системы хранения и кэширования данных пользователей для того чтобы снизить количество передаваемых по телекоммуникационным сетям данных и увеличить скорость доставки данных [8-17, 36, 57, 58]. Разработка программных средств, позволяющих повысить эффективность работы информационных систем при больших объемах данных, является важной и актуальной научно-практической задачей, требующей создания новых методов для ее решения.

Кроме того, актуальным является определение параметров состояния серверов, влияющих на скорость доставки данных конечному пользователю, а также анализ полученных параметров состояния для выполнения дальнейшего распределения запросов пользователей клиент-серверных приложений.

На сегодняшний день облачные сервера, являясь основным компонентом распределенных вычислительных систем, используются при построении высоконагруженных клиент-серверных приложений и сетей доставки данных (*CDN – content delivery network*) [79, 92, 104, 107]. При этом от состояния вычислительных

ресурсов и загруженности серверов зависят временные затраты как на доставку данных, так и на решение вычислительных задач, поэтому рациональное распределение нагрузки влияет на скорость выполнения запросов конечного пользователя. Процедуру распределения вычислительной нагрузки по серверам и перенаправление получаемых от клиентов данных на определенные сервера принято называть балансировкой нагрузки (*load balancing,*), а программы, выполняющие распределение, называются балансирами [66, 109-112].

Стоит отметить, что скорость передачи данных клиент-серверных приложений зависит и от конечных пользователей приложения, их географического местоположения, интересов и предпочтений. Эти данные должны быть учтены при выполнении анализа текущего состояния параметров состояния клиент-серверного приложения или информационной системы для увеличения эффективности работы.

Таким образом, теоретическое и практическое решение проблемы увеличения скорости доставки данных клиент-серверных приложений и информационных систем достигается за счет разработки ПО, выполняющего анализ больших объемов данных параметров состояния серверов и данных о пользователях. А также ПО, обеспечивающего оптимизацию распределения данных между серверами.

В качестве такого ПО используются сети доставки данных CDN и средства балансировки нагрузки.

Важными аспектами успеха CDN являются географические размещения реплик серверов и стратегии распределения копий данных на этих серверах, которые предназначены для оптимизации доставки данных конечному пользователю. Наиболее распространены следующие стратегии распределения данных, рисунок 1.1:

- без дублирования данных. Запросы всех пользователей обрабатываются одним сервером, рисунок 1.1 А;
- помещение копий данных на все сервера, рисунок 1.1 В;
- распределение файлов на сервера по первому требованию и последующее удаление копии после интервала времени, за которое данные не были востребованы с данного сервера, рисунок 1.1 С.

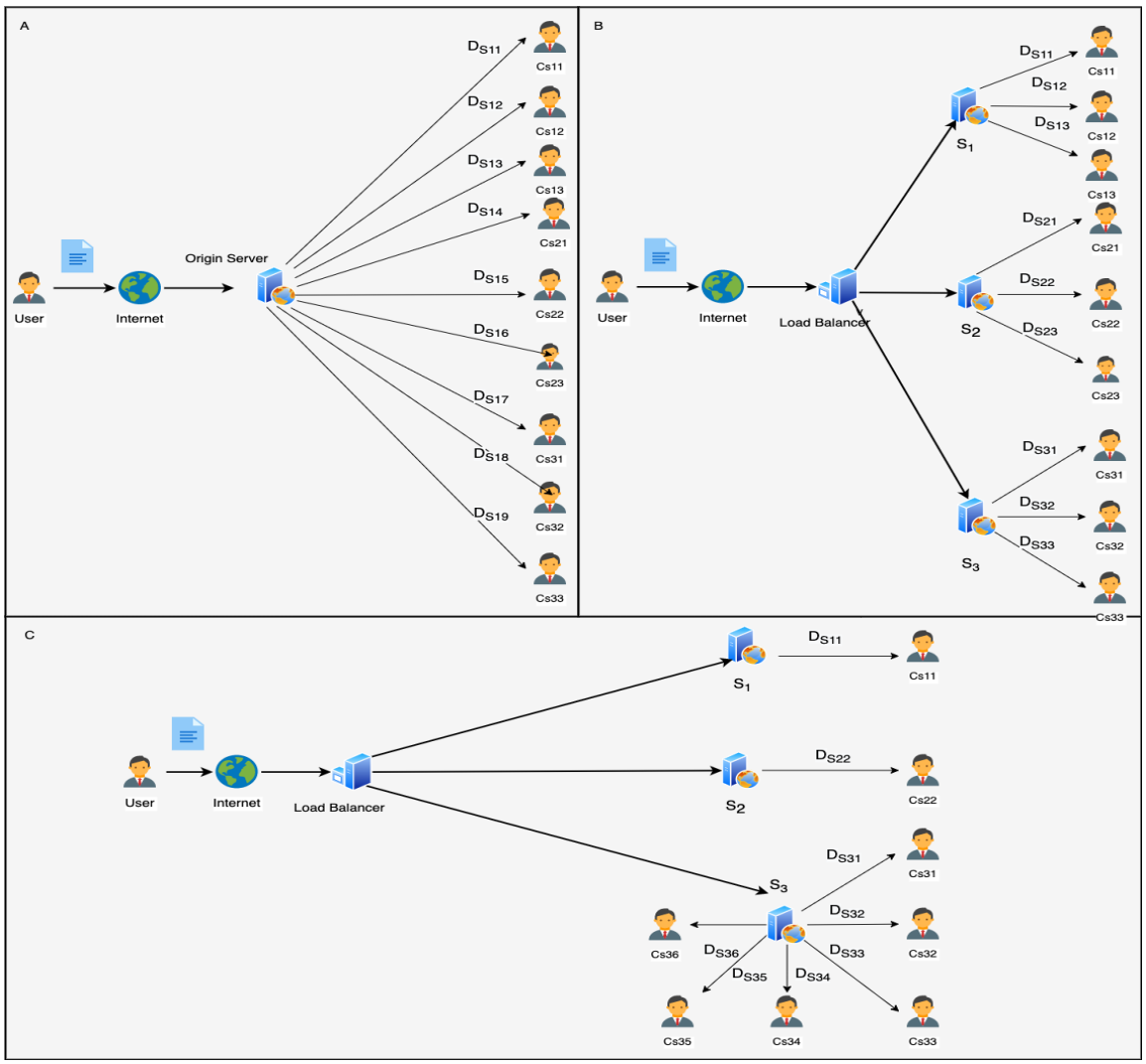


Рисунок 1.1 – Стратегии распределения запросов пользователей.

Размещение копий данных на этих серверах важно, так как рациональное размещение уменьшает как задержку доставки данных для их клиентов, так и доступную пропускную способность. Таким образом, для распределения данных по серверам стоит учитывать сетевые (полоса пропускания канала, географическое расстояние от клиента до сервера или количество граничных узлов (Hop Count)) и ценовые показатели (стоимость хранения и доставки данных).

Для распределения вычислительной нагрузки используется сервер-балансир. Такой сервер принимает запросы пользователей и перераспределяет их на наиболее подходящий сервер для решения вычислительной задачи в данный момент времени. Определение наиболее подходящего сервера происходит на основании заданного алгоритма выбора. В качестве таких алгоритмов выступает алгоритм кругового распределения (Round Robin) [69], взвешенный алгоритм кругового распределения

(Weighted Round Robin), балансировка на основе агента [114, 115]. Схема балансировки вычислительной нагрузки приведена на рисунке 1.2. На рисунке 1.2 А представлена схема без балансировки нагрузки, при которой все запросы от всех пользователей адресуются на единственный сервер, производящий вычисления. На рисунке 1.2 В представлена схема, при которой запросы пользователей попадают из внешней сети на сервер-балансир и перенаправляются на наиболее подходящий сервер для вычислений. При реализации клиент-серверных приложений, схема 1.2 В, повышается отказоустойчивость и надежность клиент-серверного приложения.

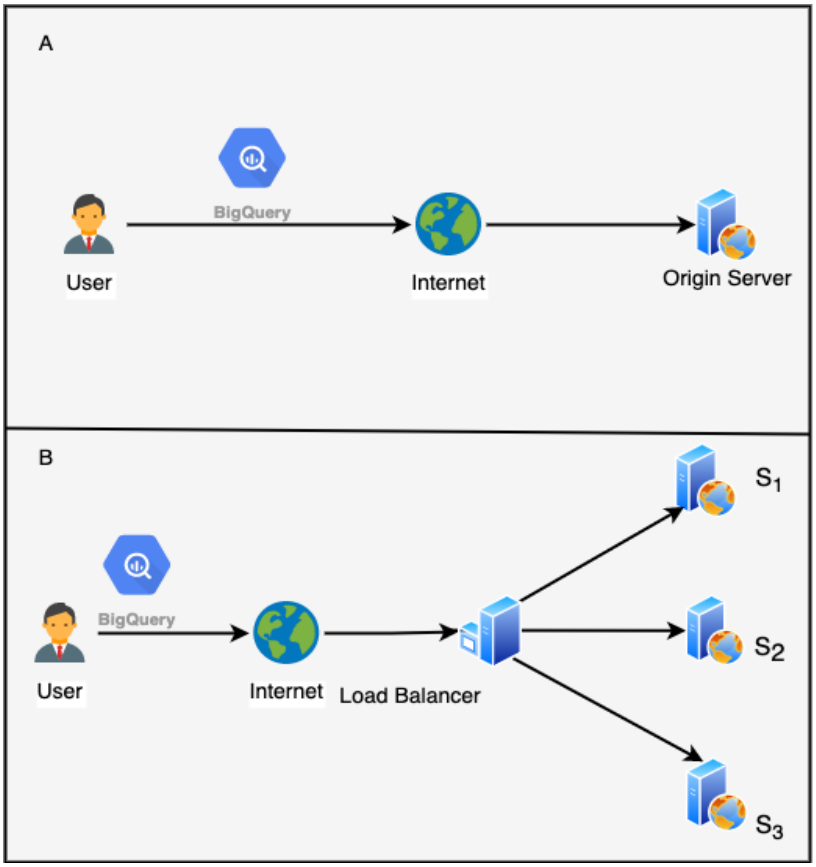


Рисунок 1.2 – Схема балансировки вычислительной нагрузки.

В работе рассматривается задача распределения данных и вычислительной нагрузки по серверам. Описывается обработка параметров состояния серверов. Предлагается метод моделирования распределения данных и вычислительной нагрузки с помощью имитационной модели серверного комплекса. Приводятся результаты натурных экспериментов по распределению вычислительной нагрузки. Основная идея работы заключается в рациональном выборе сервера для хранения данных и проведения вычислений на серверах репликах [84], чтобы минимизировать:

- время ответа клиент-серверного приложения конечным пользователям;
- стоимость хранения и передачи данных по сети;
- сбалансированную нагрузку между серверами, что особенно важно, когда запросы динамически меняются.

1.2 Обзор систем балансировки нагрузки

Балансировка нагрузки осуществляется при помощи как аппаратных, так и программных средств. Рассмотрим основные методы и алгоритмы балансировки вычислительной нагрузки. Процедура балансировки осуществляется при помощи целого комплекса алгоритмов и методов, выполняемых на следующих уровнях модели OSI (Open Systems Interconnection) [100-112]: сетевом, транспортном, прикладном, рисунок 1.3.

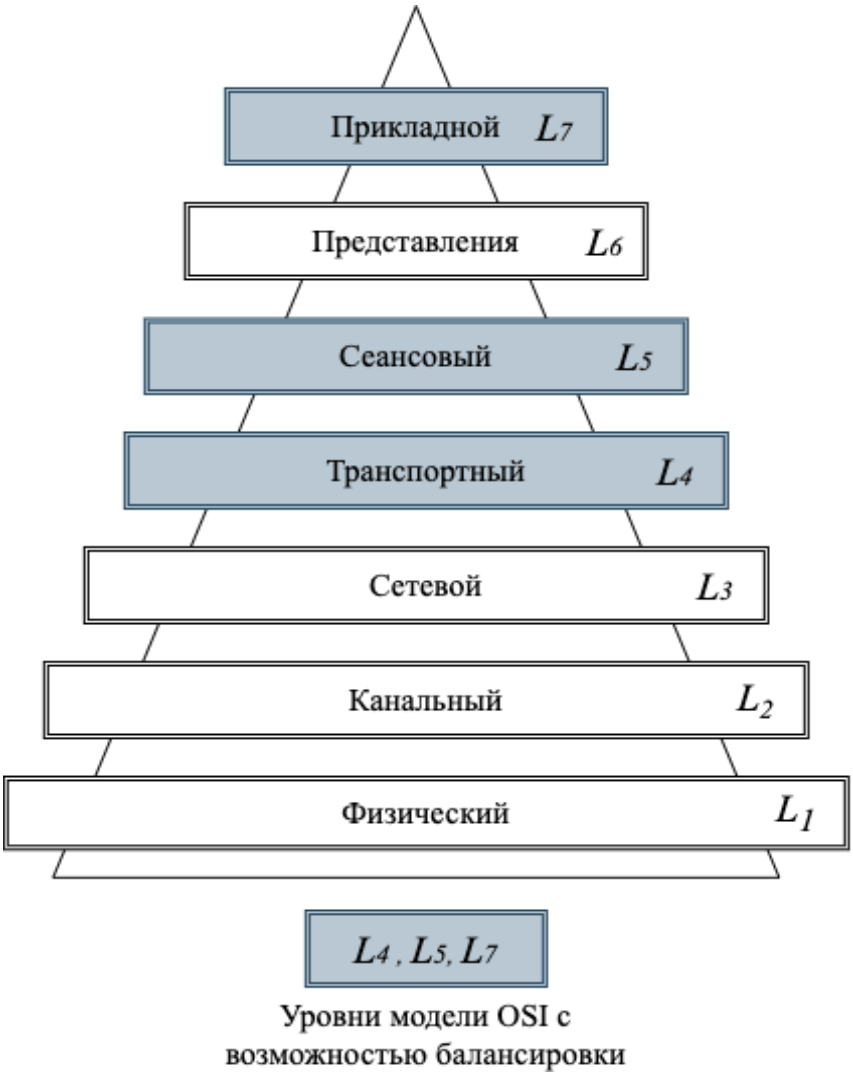


Рисунок 1.3 - Уровни модели OSI.

Рассмотрим эти уровни более подробно. Балансировка на сетевом уровне предполагает решение следующей задачи: необходимо настроить вычислительную систему таким образом, чтобы за один конкретный IP-адрес сервера отвечали разные вычислительные ресурсы. Такая балансировка осуществляется с помощью следующих способов:

DNS-балансировка, при которой одному доменному имени присваивается несколько IP-адресов. Сервер, на который будет направлен клиентский запрос, определяется с помощью алгоритма Round Robin – кругового распределения между доступными вычислительными ресурсами.

Балансировка по географическому положению осуществляется за счет размещения дублирующих копий серверов в различных географически распределенных регионах. При решении вычислительных задач распределяются вычислительные сервера или арендуются вычислительные мощности в различных центрах обработки данных (ЦОД). Для хранения данных арендуются файловые хранилища. Балансировка по территориальному признаку используется во многих сетях доставки данных (CDN).

Балансировка на основе NLB-кластера (Network Load Balancer). При использовании этого способа сервера объединяются в кластер, состоящий из входных и вычислительных узлов. Распределение нагрузки осуществляется при помощи специального алгоритма. Такие решения предоставляются производителями программного обеспечения, а также компаниями, предоставляющими вычислительные ресурсы в аренду, например Amazon, Microsoft [62].

Балансировка на **транспортном уровне** является самым простым методом распределения нагрузки. Пользователь отправляет запрос к балансиру, который в свою очередь перенаправляет запрос одному из серверов, который и будет его обрабатывать. Выбор сервера, на котором будет обрабатываться запрос, может осуществляться на основе различных параметров с применением разных алгоритмов: путём простого кругового перебора, путём выбора наименее загруженного сервера из списка и т.п. Балансировку на транспортном уровне сложно отличить от балансировки на сетевом уровне. Различие между уровнями балансировки

объясняется следующим образом. К сетевому уровню относятся решения, которые не терминируют на себе пользовательские сессии, они перенаправляют трафик и не работают в проксирующем режиме, т.е. не выбирают сервер для дальнейшей обработки запроса, а лишь передают запрос заранее выбранному серверу. На сетевом уровне балансир определяет, на какой сервер передавать данные, а сессию с клиентом осуществляет сервер.

На транспортном уровне взаимодействие пользователя с сервером выполняет балансир, который работает как прокси-сервер [79, 80, 93, 96, 112]. Он взаимодействует с серверами от своего имени, передавая информацию о клиенте в дополнительных данных и заголовках.

При распределении запросов на **прикладном уровне** балансир обрабатывает запросы пользователей и перераспределяет их по доступным серверам на основе заложенного в него алгоритма. Балансир анализирует пользовательские запросы и перенаправляет их на разные сервера в зависимости от характера запрашиваемых данных, положения пользователя, состояния серверов и других аспектов функционирования вычислительной системы. Существует большое количество различных алгоритмов и методов балансировки нагрузки. Для выбора алгоритма нужно основываться на задачах, которые решает вычислительная система, и целях, которых требуется достичь. К целям разработки относятся:

эффективность (все сервера, обрабатывающие запросы, должны быть максимально заняты; необходимо избегать состояния, при котором один или несколько серверов простаивают в ожидании запросов на обработку);

обеспечение всех запросов пользователей ресурсами (необходимо, чтобы на обработку каждого запроса были выделены ресурсы, чтобы не допустить случая, при котором один или несколько запросов ожидают выделения ресурсов или находятся в состоянии «гонки», борьбы за вычислительные ресурсы);

увеличение скорости выполнения запроса (требуется обеспечить как можно более высокую скорость обработки запроса);

сокращение времени отклика (необходимо минимизировать время ответа на запрос пользователя);

масштабируемость (балансировка нагрузки должна корректно выполнятся при любом количестве серверов в системе и любом количестве запросов пользователей);

прозрачность балансировки (требуется иметь возможность проверять как происходит распределение задач по серверам, в каких ситуациях и при каких параметрах состояния вычислительных ресурсов алгоритм будет эффективным для решения поставленных задач).

Рассмотрим существующие алгоритмы балансировки нагрузки [8-17, 104, 109-112].

Алгоритм кругового обслуживания (Round Robin) представляет собой перебор по круговому циклу: первый запрос передаётся одному серверу, затем следующий запрос передаётся другому и так до достижения последнего сервера, а затем всё начинается сначала.

Самой известной имплементацией этого алгоритма является метод балансировки Round Robin DNS [66]. Как известно, любой DNS-сервер хранит пару «имя хоста — IP-адрес» для каждой машины в определённом домене.

В числе плюсов этого алгоритма следует назвать независимость от протокола высокого уровня. Для работы по алгоритму Round Robin используется любой протокол, в котором обращение к серверу идёт по имени. Балансировка на основе алгоритма Round Robin никак не зависит от нагрузки на сервер: кэширующие DNS-серверы позволяют справиться с любым наплывом клиентов.

Использование алгоритма Round Robin не требует связи между серверами, поэтому он может использоваться как для локальной, так и для глобальной балансировки. Решения на базе алгоритма Round Robin отличаются низкой стоимостью, для того чтобы они начали работать, достаточно просто добавить несколько записей в DNS.

Алгоритм Round Robin имеет и целый ряд существенных недостатков. Для того чтобы распределение нагрузки по этому алгоритму отвечало упомянутым выше критериями эффективности, требуется, чтобы каждый сервер имел одинаковую конфигурацию, а для выполнения запросов пользователей должно быть задействовано одинаковое количество ресурсов. На практике одинаковое состояние серверов и равномерное выделение ресурсов для решения задач трудно достижимо.

Также при балансировке по алгоритму Round Robin не учитывается загруженность того или иного сервера в составе кластера. Гипотетически существует такая ситуация, при которой один сервер загружен на 90-100%, в то время как другие сервера мало загружены (10-20%). Алгоритм Round Robin никак не обработает такую ситуацию, и перегруженный сервер продолжит получать запросы. При этом скорость обработки запросов на этом сервере значительно упадет. Тем самым снижается эффективность работы всего приложения в целом. Таким образом, область применения алгоритма Round Robin ограничивается сферой задач, в которых распределение нагрузки равномерно, а вычислительные узлы имеют примерно одинаковые аппаратные ресурсы.

Weighted Round Robin – это усовершенствованная версия алгоритма Round Robin. Усовершенствования заключается в следующем: каждому серверу устанавливается весовой коэффициент в соответствии с его производительностью и мощностью. Такой подход добавляет возможности для гибкого распределения нагрузки. Сервера, обладающие большим весовым коэффициентом, обрабатывают большее количество запросов. Однако всех проблем с отказоустойчивостью этот алгоритм также не решает. Более эффективную балансировку обеспечивают другие методы, в которых при планировании и распределении нагрузки учитывается большее количество параметров. Еще одним недостатком алгоритмов Round Robin и Weighted Round Robin является отсутствие возможности учета количества активных на данный момент подключений. Например, клиент-серверная система состоит из двух серверов. Обозначим их как S_1 и S_2. К серверу S_1 подключено меньше пользователей, чем к серверу S_2, но при этом сервер S_1 более нагружен. Данная ситуация происходит из-за того, что подключения к серверу S_1 поддерживаются в течение более долгого времени по сравнению с подключениями к серверу S_2.

Проблему различного количества подключений к одному серверу позволяет решить алгоритм least connections. Данный алгоритм учитывает количество подключений, поддерживаемых серверами в текущий момент времени. Каждый следующий запрос передаётся серверу с наименьшим количеством активных подключений.

Существует усовершенствованный вариант этого алгоритма, предназначенный

в первую очередь для использования в кластерах, состоящих из серверов с разными техническими характеристиками и разной производительностью. Он называется Weighted Least Connections и учитывает при распределении нагрузки не только количество активных подключений, но и весовой коэффициент сервера.

Кроме того, существует еще ряд усовершенствованных вариантов алгоритма Least Connections, которые следует отметить: Locality-Based Least Connection Scheduling и Locality-Based Least Connection Scheduling with Replication Scheduling.

Locality-Based Least Connection Scheduling метод был создан для использования системах кэширующих прокси-серверов. Основная идея алгоритма заключается в том, что наибольшее количество запросов направляется на сервер с наименьшим количеством активных подключений. Каждому серверу присваивается группа ip-адресов пользователей. Запросы с этих адресов направляются на соответствующий сервер, если он не загружен полностью. Если сервер полностью загружен и не может больше обрабатывать запросы, запрос будет перенаправлен на другой, менее загруженный сервер.

Для алгоритма Locality-Based Least Connection Scheduling with Replication Scheduling ip-адрес или группа ip-адресов присваивается не отдельному серверу, а целой группе серверов. Каждый запрос пользователя адресуется наименее загруженному серверу в группе. Если все сервера группы перегружены, то будет вызвана процедура запуска нового сервера. Этот сервер будет присоединён к группе, обслуживающей ip-адрес, с которого пользователь отправил запрос. Тем временем наиболее загруженный сервер будет удален из группы. Такая реализация позволяет избежать избыточного создания и выделения серверов.

Алгоритм Destination Hash Scheduling разработан для распределения нагрузки в кластере кэширующих прокси-серверов, но он часто используется и в других случаях. В основе этого алгоритма лежит обработка запросов на основе статической таблицы ip-адресов получателей данных.

Алгоритм Source Hash Scheduling работает аналогично предыдущему, однако сервер, который будет обрабатывать запрос, выбирается из таблицы по ip-адресу отправителя.

Алгоритм Sticky Sessions – это алгоритм распределения входящих запросов, при

котором соединения передаются на один и тот же сервер группы. Он используется, например, в веб-сервере Nginx [92]. С помощью метода IP hash - сессии пользователя ставится в соответствие определенный сервер [92]. Дальнейшие запросы распределяются по серверам на основе ip-адреса пользователя. Если сервер, который соответствует тому или иному ip-адресу не доступен, то запрос будет перенаправлен на другой сервер. Данный метод обладает рядом недостатков. При использовании динамических ip-адресов пользователи могут быть переадресованы не на тот сервер, на котором был обработан их предыдущий запрос. Это может привести к некорректным результатам обработки запроса. Кроме того, алгоритм не учитывает состояние сервера, на котором происходят вычисления. Пользователь будет направлен на сервер, с которым он имеет сессию, даже в случае высокой загруженности последнего.

Для того чтобы избежать приведенных выше недостатков, предлагается использовать алгоритм балансировки, который будет учитывать состояние серверов. Такой подход называется балансировкой на базе агента. Программа агент собирает данные о состоянии сервера и передает по запросу серверу-балансиру. Сервер-балансир, в свою очередь, на основе заложенного в него алгоритма, принимает решение о выборе наиболее подходящего сервера для проведения вычислений. Решение о выборе сервера-балансира принимается с использованием интеллектуальных технологий.

1.2 Использование интеллектуальных технологий для распределения нагрузки между вычислительными ресурсами

Требования, предъявляемые к клиент-серверным приложениям противоречивы, необходимо увеличить время обработки запросов пользователей и уменьшить количество ресурсов выделяемых для выполнения этих запросов. В связи с этим видится перспективным рассматривать задачу балансировки нагрузки как задачу кластерного анализа [5, 10, 89, 90]. Требуется разделить запросы пользователей по узлам вычислительного комплекса таким образом, чтобы минимизировать нагрузку на сервера, обеспечив при этом максимальную скорость доставки данных конечному

пользователю. Кластерный анализ - процедура, выполняющая разбиение объектов выборки на группы (классы). Применительно к задаче выбора сервера необходимо разделить (классифицировать) запросы пользователей по серверам так, чтобы разбиение удовлетворяло обозначенным выше требованиям. На сегодняшний день существует несколько алгоритмов кластерного анализа. Результаты, полученные после кластеризации, делятся на четкое (результат кластеризации 0 или 1) отнесение к кластеру и нечеткое, при котором запрос пользователя относится к кластеру с определенной степенью вероятности. В работе рассматриваются наиболее часто используемые алгоритмы кластерного анализа *K-средних* [71] и *C-средних* [65, 119]. Так же задачу распределения вычислительной нагрузки с помощью кластерного анализа позволяют решать нейронные сети [24, 38, 48, 55]. Основная идея алгоритма *K-средних* заключается в том, что запросы пользователей произвольно разбиваются на кластеры, после чего итеративно пере вычисляется центр масс для каждого кластера, полученного на предыдущем шаге; затем запросы разбиваются на кластеры вновь в соответствии с тем, какой из новых центров оказался ближе по выбранной метрике. Целью данного алгоритма является разбиение n запросов пользователей на n серверов (кластеров) таким образом, чтобы каждый запрос принадлежал ровно одному серверу (кластеру). Алгоритм *C-средних* разделяет запросы пользователей по серверам, которые являются нечеткими множествами, и каждая точка принадлежит различным серверам (кластерам) с различной степенью принадлежности. Запрос пользователя направляется на сервер, к которому имеет большую степень принадлежности. Применение данных алгоритмов для решения задачи распределения запросов пользователей по серверам приводится в работах [84, 104, 108].

Стоит отметить, что в результате выполнения кластерного анализа группы запросов пользователей могут быть разделены так, что запросы пользователей будут трудно разделимы по серверам. Такая ситуация возникает, если параметры состояния нескольких серверов схожи. В этом случае имеет смысл применить дополнительный математический аппарат для выбора сервера. В качестве такого математического аппарата применена нечёткая логика (fuzzy logic) [4, 6, 16, 19, 20, 31, 35, 37, 38, 45,

70, 112, 113] - раздел математики, являющийся обобщением классической логики и теории множеств, базирующийся на понятии нечёткого множества [97, 98] как объекта с функцией принадлежности элемента ко множеству, принимающий любые значения в интервале [0,1], а не только строгое значения 0 или 1. На основе этого понятия вводятся различные логические операции над нечёткими множествами и формулируется понятие лингвистической переменной, в качестве значений которой выступают нечёткие множества. Таким образом, предлагается относить пользовательский запрос к тому или иному серверу используя лингвистическую переменную, обозначающую степень его принадлежности.

1.3 Имитационное моделирование и оптимизация при решении задачи балансировки вычислительной нагрузки

Для описания объектов, функционирующих в условиях действия случайных факторов, таких как серверный комплекс облачного ресурса, используется класс математических моделей, называемых системами массового обслуживания (СМО) [7, 26, 30, 32, 34, 43, 44, 41, 46]. В серверный комплекс поступают запросы пользователей в случайные моменты времени и обслуживаются с помощью имеющихся каналов обслуживания – серверов, переключение между которыми выполняется с помощью сервера-балансира.

Для того чтобы осуществить рациональную балансировку запросов, в зависимости от вида запросов, необходимо учитывать разные характеристики вычислительных ресурсов.

В задачах распределения статических данных (РСД) необходимо рассматривать такие параметры как доступность пропускного канала, расстояние от клиента до сервера и стоимость хранения и доставки данных. Тогда как для задачи распределения вычислительной нагрузки (РВН) рассматриваются загруженность аппаратных ресурсов и расстояние от клиента до сервера. Предполагается, что запросы на обслуживание образуют поток, то есть последовательность запросов с различным интервалом времени между их появлением. Понятие потока основано на предположении, что каждое событие будет происходить в заранее неизвестные

случайные моменты времени. Имитационное моделирование систем массового обслуживания складывается из моделирования потока запросов и работы обслуживающих каналов [12, 13, 25, 39].

Для описания серверного комплекса как СМО необходимо задать данные о входящем потоке запросов, которые поступают на обслуживание, порядок постановки запросов в очередь и выбора из нее, а также правило, по которому осуществляется распределение нагрузки (алгоритм работы сервера-балансира). Чтобы описать входящий поток запросов надо описать моменты времени их поступления в серверный комплекс. Закон поступления может быть детерминированный или вероятностный. В нашем случае входящий поток запросов описывается распределением вероятностей интервалов времени между соседними запросами (запросами пользователей). Правила обслуживания запросов серверами характеризуются длительностью обслуживания (распределением времени обслуживания) и дисциплиной обслуживания. Для характеристики свойств каждой линии задается длительность обслуживания запросов сервером или время занятости сервера как случайная величина с заданным законом распределения.

Рассмотрим обслуживающую систему, состоящую из трех каналов (в соответствии с количеством серверов), способных одновременно и независимо друг от друга обслуживать запросы. В любой момент времени сервер находится в одном из двух состояний: свободном или занятом. Запросы принимаются к обслуживанию в порядке очереди, свободный канал приступает к обслуживанию того запроса, который ранее других поступил в систему, то есть используется дисциплина обслуживания запросов, называемая в англоязычной литературе FIFO – First In - First Out (первым поступил – первым обслужился).

Выходными параметрами серверного комплекса являются величины, характеризующие качество его функционирования как СМО. Это такие параметры, как максимальная и средняя длина очередей запросов в информационной системе, среднее время нахождения запросов в очередях и каналах обслуживания, а также количество обслуженных запросов серверами.

Имитационная модель серверного комплекса позволит определять его характеристики и изменения состояния во времени при заданных потоках запросов, поступающих на входы системы, выполнять исследования методов распределения нагрузки по серверам и обосновывать алгоритм выбора сервера для распределения запросов пользователей.

1.4 Выводы по главе

1. Проведен анализ работ, посвященных вопросам балансировки нагрузки, рассмотрены существующие системы и способы решения задачи распределения данных и вычислительных задач между серверами. Сделан вывод о том, что большинство существующих систем использует круговое распределение нагрузки, не учитывая при этом состояния серверного комплекса.

В связи с этим возникают временные задержки при обработке запросов пользователей в случаях, когда серверы неравномерно загружены или имеют различное географическое удаление от пользователя, что снижает пользовательский интерес и ведет к потере прибыли владельцев клиент-серверных приложений и информационных систем.

2. Обоснована необходимость учитывать при распределении нагрузки такие параметры состояния серверов, как загруженность вычислительных ресурсов, взаимное географическое положение клиента и сервера, а также пропускная способность канала передачи данных для повышения скорости выполнения запросов пользователей.

Кроме того, сделан вывод о том, что при распределении статических данных следует учитывать стоимостные характеристики хранения и передачи данных, которые изменяются в широком диапазоне.

3. Выбраны и обоснованы методы интеллектуального анализа данных, позволяющие выявлять значимые параметры состояния серверов, учет которых позволяет повысить быстродействие и производительность системы балансировки нагрузки.

4. Получено, что имитационное моделирование пригодно для определения выходных параметров серверного комплекса для задач распределения статических

данных и вычислительной нагрузки. Кроме того, имитационное моделирование позволяет оценить производительность серверного комплекса и уменьшить затраты на проведение натурных экспериментов.

5. Выявлена необходимость выполнения декомпозиции системы балансировки нагрузки на подсистемы, реализующие задачу распределения статических данных и задачу распределения вычислительной нагрузки в связи с необходимостью учета различных параметров состояния серверного комплекса. Показано, что кроме сетевых показателей в задаче РСД необходимо учитывать стоимостные параметры, а в задаче РВН следует принимать во внимание вычислительные характеристики.

2 МЕТОДЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ О СОСТОЯНИИ СЕРВЕРОВ ДЛЯ РЕШЕНИЯ ЗАДАЧИ БАЛАНСИРОВКИ

Вторая глава **посвящена** сбору и обработке данных о состоянии серверов. Выявлены и обоснованы параметры состояния серверов пригодные для анализа при распределении нагрузки. Предложен метод поиска закономерностей в параметрах состояния серверов на основе анализа паттернов данных. Рассмотрен алгоритм обработки данных о состоянии узлов серверного комплекса. Предложен метод распределения нагрузки между вычислительными ресурсами клиент-серверной информационной системы на основе кластеризации параметров состояния серверов. Рассмотрена процедура формирования правил и показателей выбора сервера для алгоритма распределения вычислительной нагрузки на основе кластерного анализа и деревьев принятия решений.

2.1 Постановка задачи выбора сервера для распределения нагрузки на основе параметров состояния серверного комплекса

В работе рассмотрены вопросы балансировки нагрузки в клиент-серверных системах для двух задач: распределения статических данных и вычислительных запросов [8-11, 108-111]. Для решения первой задачи необходимо распределить по вычислительным ресурсам такие статические данные как текстовые документы, графические файлы, цифровой видеоряд и т.д. для хранения и предоставления по запросу пользователей. Для того чтобы решить вторую задачу, требуется распределить вычислительные запросы по узлам серверного комплекса, а также осуществить поиск меняющейся во времени информации, чтение, добавление или удаление данных и т.д. Предлагается реализовать систему балансировки нагрузки, состоящую из двух подсистем, каждая из которых выполняет распределение нагрузки по вычислительным ресурсам в соответствии с поставленной задачей.

Каждая из задач балансировки нагрузки имеет свои особенности, так как вычислительные задачи обладают отличными от задач хранения и доставки статических данных характеристиками, что приводит к необходимости разбиения системы балансировки на подсистемы. На рисунке 2.1 приведена схема клиент-

серверного приложения с балансиром нагрузки, реализующим подсистемы распределения статических данных и вычислительной нагрузки. Так при распределении вычислительной нагрузки необходимо учитывать сложность задачи и производительность серверов, в то время как для распределения статических данных должен быть учтен их объем, загруженность канала связи и расстояние до конечного пользователя.

При выполнении запроса пользователя о загрузке данных (например, текстовых файлов, графических изображений и т.д.) сервер-балансир с использованием подсистемы распределения статических данных производит выбор севера на основе информации о состоянии серверного комплекса и данных о пользователе. В случае запроса пользователя на решение вычислительной задачи подсистема распределения вычислительной нагрузки (в составе сервера балансира) на основе анализа данных о состоянии массива серверов определяет подходящий сервер, который производит обработку запроса, и результаты вычислений предоставляются пользователю. Схема разрабатываемого клиент-серверного приложения представлена на рисунке 2.1.

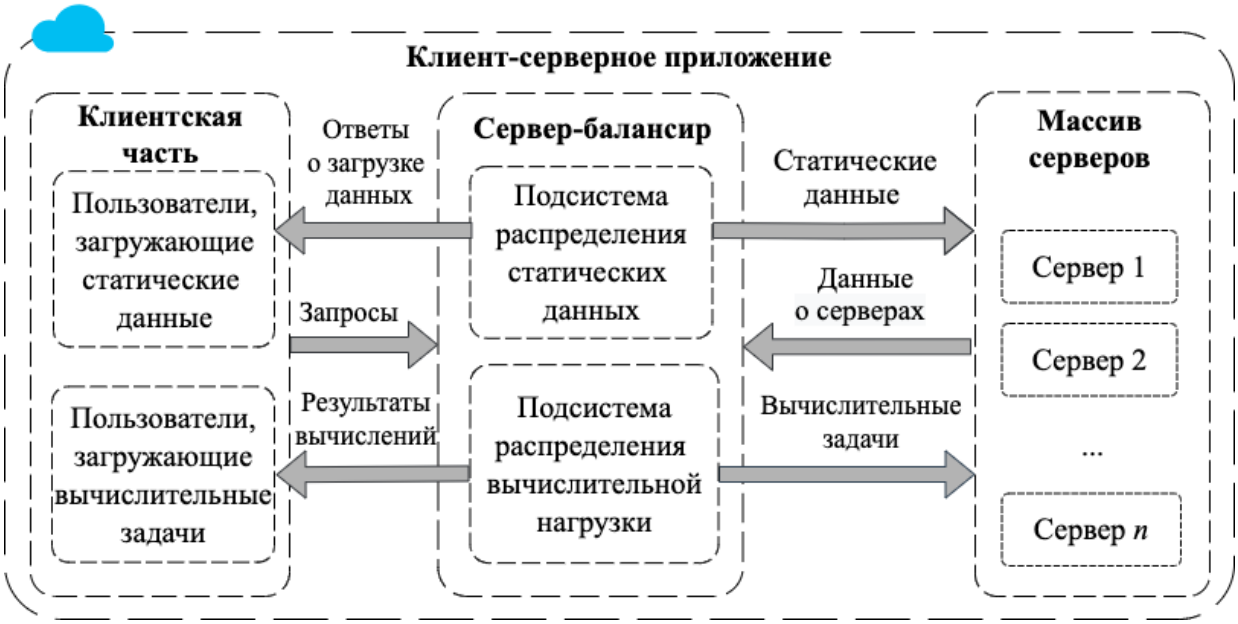


Рисунок 2.1 – Схема клиент-серверного приложения с балансиром нагрузки

Для того чтобы принимать обоснованное решение о выборе сервера для хранения данных и проведения вычислений, необходимо определить набор входных параметров.

Разработанный аналитико-имитационный метод распределения нагрузки по

серверам состоит из трех этапов. На *первом этапе* выполняется сбор данных и их аналитическая обработка с помощью комбинированного метода, формирующего показатели и правила выбора сервера на основе анализа паттернов данных и кластеризации. Алгоритм сбора и обработки данных о состоянии серверов представлен на рисунке 2.2.

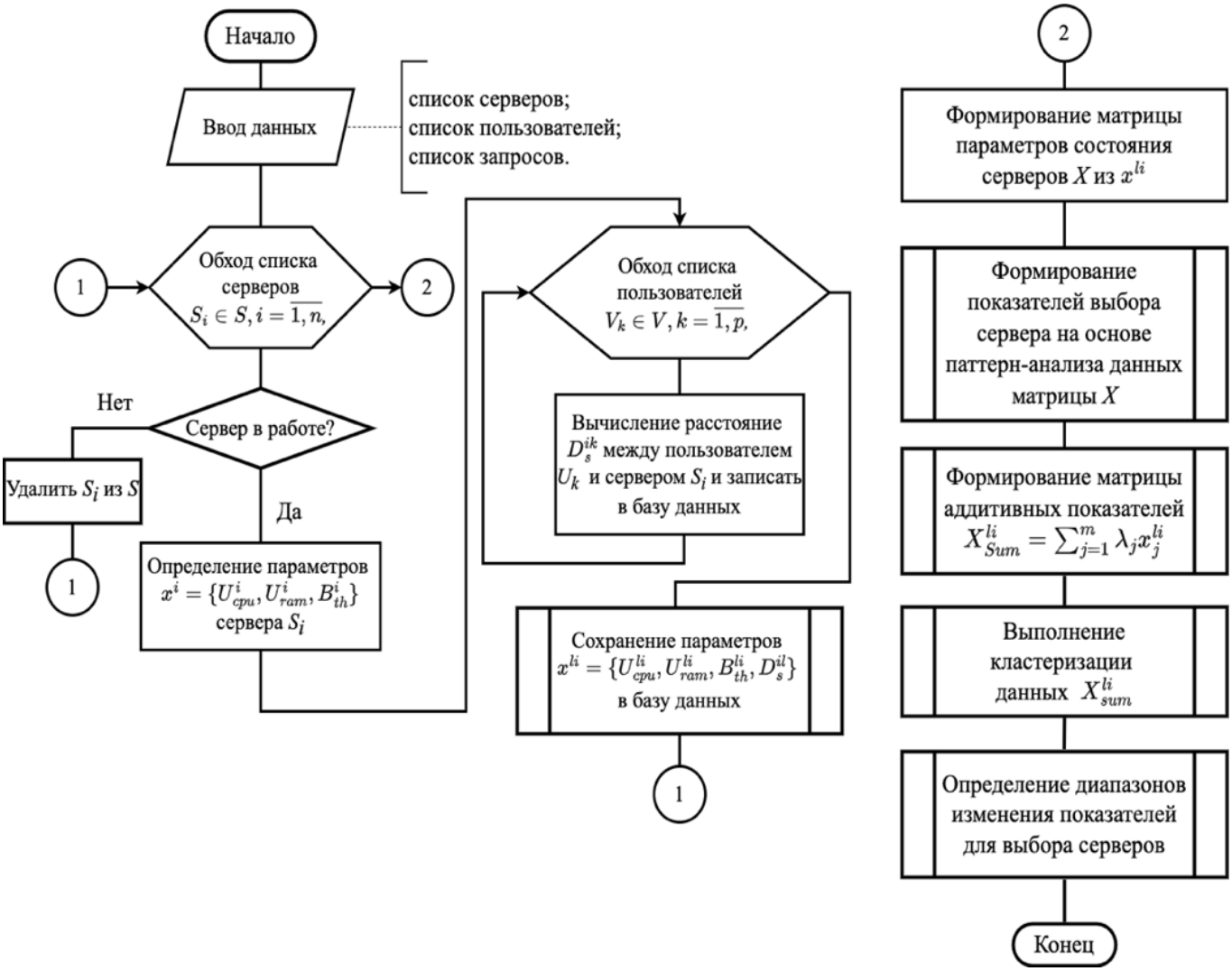


Рисунок 2.2 – Алгоритм обработки данных о состояния серверного комплекса

Основным циклом алгоритма является обход серверов и сбор значений параметров состояния. После окончания цикла данные передаются на следующий шаг, на котором производится анализ паттернов и выявление закономерностей данных. На основе проведенного анализа паттернов выявляются и обосновываются наиболее показательные параметры состояния серверов. Выявленные показатели используются для проведения процедуры кластерного анализа, в результате выполнения которого сервера разделяются на кластеры. На последнем шаге

алгоритма по полученным данным о кластерах определяются границы диапазонов изменения параметров состояния и формируются правила выбора сервера.

На *втором этапе* предусмотрены имитационные исследования для оценки влияния параметров состояния серверов на распределение вычислительной нагрузки с помощью нечеткого логического вывода, положенного в основу работы сервера-балансира. На *третьем этапе* проводятся экспериментальные исследования в облачном кластере серверов для формирования структуры программного модуля балансировки нагрузки, реализующего параллельные вычисления, позволяющие сократить временные затраты на обработку запросов пользователей.

Рассмотрим этапы реализации предлагаемого аналитико-имитационного метода распределения нагрузки. На *первом этапе* проводился натурный эксперимент, заключающийся в сборе данных о состоянии вычислительных ресурсов реально действующего облачного серверного комплекса.

В зависимости от вида задачи необходимо учитывать разные характеристики вычислительных ресурсов. Следует отметить, что обычно в задачах распределения нагрузки рассматриваются *ценовые*, *сетевые* и *вычислительные* параметры, влияющие на распределение данных [69, 88]. К *ценовым* показателям относятся:

- затраты на развертывание сервера путем покупки или аренды вычислительного ресурса, которые складываются из стоимостей вычислительных мощностей и хранилища данных, а также зависят от пропускной способности канала передачи;
- стоимость доставки (скачивания) данных с сервера-реплики (или исходного сервера) конечному пользователю;
- стоимость распространения обновленных данных с исходного сервера на дублирующий сервер или группу серверов. Например, в случае, когда конечный пользователь выполняет операцию обновления ранее загруженных данных.

В качестве *сетевых* показателей для распределения нагрузки обычно принимаются параметры, характеризующие качество связи между сервером и конечным пользователем в топологии сети. К таким параметрам относятся: время

приема-передачи *RTT* (*Round Trip Time*) [109-112], количество граничных участков сети (*Hop Count*) [69] или расстояние от клиента до сервера, доступная пропускная способность (*Bandwith*) [109-112] и тип сетевого подключения. Эти параметры сети используются существующими алгоритмами балансировки нагрузки в сетях доставки данных *CDN* для размещения реплик (т.е. дублирующих вычислительных ресурсов) сервера [72, 84, 104].

К *вычислительным* показателям работы сервера относятся параметры, связанные с производительностью аппаратных ресурсов, которые, в свою очередь, подразделяются на параметры переменные (загруженность оперативной памяти, центрального процессора) и постоянные (тактовая частота процессора, объем оперативной памяти и скорость чтения данных с жесткого диска).

Следует отметить, что существующими алгоритмами балансировки нагрузки вычислительные показатели работы серверов обычно не учитываются в связи со сложностью сбора и обработки данных о состоянии вычислительных ресурсов. Особенностью предлагаемого метода является учет параметров состояния серверного комплекса при балансировке нагрузки с целью уменьшения временных затрат на обработку запросов пользователей. Для постановки и решения задачи балансировки введем следующие обозначения параметров, характеризующих работу серверов:

D_s – расстояние от пользователя до сервера (км) вычисляется по формуле:

$$\Delta\sigma = \arctan\frac{\sqrt{(\cos\varphi_2 * \sin\Delta\lambda)^2 + (\cos\varphi_1 \ \sin\varphi_2 \ - \ \sin\varphi_1 \cos\varphi_2 \cos\Delta\lambda)^2}}{\sin\varphi_1 \ \sin\varphi_2 + \ \cos\varphi_1 \ \cos\varphi_2 \ \cos(\Delta\lambda)}$$

$$D_s = R \ * \Delta\sigma, \qquad (2.1)$$

где R – радиус Земли, $\Delta\sigma$ – угол, вершина которого является центром окружности, а стороны являются радиусами, пересекающими окружность в двух различных точках 1 и 2, φ_1, φ_2 – координаты долготы точек 1 и 2 соответственно, $\Delta\lambda$ – разница широты точек 1 и 2;

U_{cpu} – загруженность центрального процессора. Процент времени, в течение которого не запускается задача простоя операционной системы (ОС). Операционная система хранит общее количество времени, затраченного на выполнение задачи простоя. Таким образом, загруженность центрального процессора рассчитывается в

соответствии со следующей формулой:

$$U_{cpu} = \frac{1}{k}\sum_{p=0}^{k}\left(1 - \frac{t_{k-p}^{idle}}{t_{k-p}^{usage}}\right), \quad (2.2)$$

где $k = 60$ количество наблюдений в интервале времени, p – номер наблюдения, t^{idle} – время простоя центрального процессора, t^{usage} – время выполнения процессов ОС, $U_{cpu} \in [0; 1]$.

U_{ram} – загруженность оперативной памяти (отн. ед.), рассчитывается следующим образом

$$U_{ram} = \frac{U_{ram}^{used}}{U_{ram}^{total}}, \quad (2.3)$$

где U_{ram}^{total} – общий объем оперативной памяти (Кбайт), U_{ram}^{used} – объем использованной оперативной памяти (Кбайт), $U_{ram} \in [0; 1]$.

B_{ch} – доступная пропускная способность канала передачи данных, Кбит/с;

$$B_{ch} = I/t \quad (2.4)$$

где I – объем передаваемых данных (Кбит/с), t – время передачи, с.

C_{sdr} – стоимость затрат на хранение, доставку и репликацию данных, ден. ед.

Параметры, характеризующие расстояние, загруженность ресурсов и пропускную способность канала являются переменными, в то время как стоимость – постоянный параметр, так как сохраняет свое значение для каждого региона расположения вычислительного ресурса и поставщика. В связи с тем, что параметры состояния серверов имеют разные диапазоны изменения, для приведения к диапазону от 0 до 1 параметры D_s, B_{ch}, C_{sdr} нормализованы с помощью деления на максимальное значение каждого параметров массива исходных данных по формуле:

$$\hat{x}_j^{li} = \frac{x_j^{li}}{x_j^{max}}, \quad (2.5)$$

где x_j^{max} – максимальное значение показателя состояния среди всех серверов.

Так как в работе рассматривается задача балансировки для двух случаев, то требуется использовать разные показатели для каждого из них. В первом случае для задачи распределения вычислительной нагрузки следует учитывать следующие параметры: расстояние от пользователя до сервера D_s (сетевой показатель),

загруженность оперативной памяти U_{ram}, загруженность центрального процессора U_{cpu} (вычислительные показатели).

Во втором случае ставится задача распределения статических данных, при решении которой важно учитывать не только сетевые, но и ценовые показатели. Рассмотрены такие параметры, как расстояние D_s, пропускная способность канала B_{ch} (сетевые показатели), стоимость затрат на хранение, доставку и репликацию данных C_{sdr}, (ценовой показатель).

Совокупность серверов обозначим $S = (S_1, S_2, S_3)$. Приведем в соответствие введенные выше обозначения параметров состояния серверного комплекса характеристикам серверов. В случае задачи распределения вычислительной нагрузки вектор параметров состояния запишем в виде:

$$X^{li} = (\hat{x}_1^{li}, x_2^{li}, x_3^{li}) = (D_s^{li}, U_{ram}^{li}, U_{cpu}^{li}) , \quad (2.6)$$

где верхний индекс $l = \overline{1,q}$ – номер запроса на загрузку файла, q – количество запросов на загрузку файла, $i = \overline{1,n}$ – номер сервера, n – количество серверов, равное количеству кластеров, т.е. для каждого l-ого запроса пользователя относительно i-ого сервера первый $\hat{x}_1^{li} = D_s^{li}$ второй параметр $x_2^{li} = U_{ram}^{li}$, третий параметр $x_3^{li} = U_{cpu}^{li}$.

При рассмотрении задачи распределения статических данных вектор параметров состояния примет вид:

$$X^{li} = (\hat{x}_1^{li}, \hat{x}_2^{li}, \hat{x}_3^{li}) = (D_s^{li}, B_{ch}^{li}, C_{sdr}^{li}). \quad (2.7)$$

Это соответствует тому, что для каждого l-ого запроса пользователя к i-ому серверу вектор параметров состояния содержит компоненты: $\hat{x}_1^{li} = D_s^{li}$, $\hat{x}_2^{li} = B_{ch}^{li}$, и $\hat{x}_3^{li} = C_{sdr}^{li}$.

Для того чтобы реализовать процедуру сбора данных, создана программа «Агент» на языке JavaScript [94], которая состоит из трех основных компонентов (модулей): сборщик, модель, маршрутизатор. Фрагменты программных кодов каждого из компонентов приведены на рисунке 2.3.

```
var http = require('http');
// Use response utils
var response = require('../utils/response');

// Models
var ModelStat = require('../models/stat');

router.get('/', function (req, res) {
  var request = require('request');

  var result = ModelStat.find(function(){});

  res.send(result);
});
```

а)

```
var mongoose = require('../utils/mongoose');

var stat = new mongoose.Schema({
  server: { type: String },
  freemem : { type : Number },
  diskFreeSpace: { type: Number, default: 0 },
  diskUsage: { type: Number, default: 0 },
  cpuUsage: { type: Number, default: 0 },
  upTime: { type: Number, default: 0 },
  createdAt : Date
}, {versionKey: false});

module.exports = mongoose.model('stat', stat);
```

б)

```
var j = schedule.scheduleJob('*/1 * * * *', function(){
  var os = require('os-utils');

  var freeMemory = os.freememPercentage();
  var systemUptime = os.sysUptime();

  os.cpuUsage(function(v) {
    var cpuUsage = v;

    var diskspace = require('diskspace');
    diskspace.check('/', function (err, result) {
      var stat = new ModelStat();

      stat.set({
        server : config.get('ip'),
        freemem: freeMemory,
        diskFreeSpace: result.free,
        diskUsage: result.used,
        cpuUsage: cpuUsage,
        upTime: systemUptime,
        createdAt: new Date()
      });

      stat.save()
        .then(function () {
          console.log('Server stats saved')
        })
        .catch(function (error) {
          //send error
          var message = 'Can not save server params';
          console.log(message);
        });
    });
  });
});
```

в)

Рисунок 2.3 – Фрагменты программного кода модулей:

а) – маршрутизатор, б) – модель, в) – сборщик

Первый из модулей, сборщик, представляет собой подпрограмму, которая опрашивает серверы о их состоянии, а полученную информацию в требуемом формате (в зависимости от выполняемого запроса передает в модель). Программный

модуль модель представляет полученные от агента данные в виде векторов параметров состояния серверов и сохраняет их в базе данных MongoDB [91]. Третий модуль, маршрутизатор, выполняет поисковый запрос к базе данных, получает данные и предоставляет их серверу-балансиру.

Программа структурирована таким образом, чтобы каждый модуль выполнял свою функцию, при этом модуль сборщик работает по расписанию (вызывается циклически, раз в минуту), в то время как остальные модули выполняются последовательно. Кроме того, в программном модуле сборщика организованы вложенные вызовы функций для определения параметров состояния серверов.

В качестве тестового набора данных было использовано 50 программно-сгенерированных файлов различного объема в формате .jpeg. Эти файлы были отправлены на описанные выше сервера по средствам http запросов на URI – адрес */upload*.

В результате выполнения запросов на сервера было получено 150 записей тестовых данных с параметрами B_{ch}, D_s, C_{sdr}. Аналогичным образом проводились испытания сбора данных о состоянии серверов для задачи распределения вычислительной нагрузки.

Был подготовлен URI – адрес */search* для всех трех серверов, при обращении на который производился поиск ключевой последовательности символов, переданных в запросе в текстовом документе. В качестве текстовой последовательности параметра запроса было использовано 150 ключевых фраз присутствующих в тексте документа.

После выполнения запросов по тестовой выборке получено 150 записей тестовых данных с параметрами, D_s, U_{ram}, U_{cpu}.

Исходные данные, полученные в результате проведенных экспериментов, содержащие параметры состояния серверов для задач распределения вычислительной нагрузки и данных частично представлены в таблицах 2.1 и 2.2. Для того чтобы поставить задачу распределения нагрузки, представим параметры состояния серверов в соответствии с (2.1) в виде $X^{li} = (x_1^{li}, \dots, x_j^{li}, \dots, x_m^{li}), l = \overline{1,q}, i = \overline{1,n}, j = \overline{1,m}$, где q – количество запросов, n – количество серверов, m – количество параметров.

Таблица 2.1 – Исходные данные для задачи распределения вычислительной нагрузки

№ запроса, l=(1,2,…,50)	Параметры состояния серверов S_i, i=(1,2,3)								
	Расстояние от пользователя до сервера, D_s^{li}			Загруженность оперативной памяти, U_{ram}^{li}			Загруженность центрального процессора, U_{cpu}^{li}		
	S_1	S_2	S_3	S_1	S_2	S_3	S_1	S_2	S_3
1	0,49	0,79	0,84	0,67	0,18	0,75	0,52	0,53	0,58
2	0,58	0,77	0,82	0,29	0,34	0,73	0,53	0,58	0,54
…	…	…	…	…	…	…	…	…	…
50	0,72	0,81	0,64	0,8	0,63	0,7	0,55	0,51	0,58

Таблица 2.2 – Исходные данные для задачи распределения данных

№ запроса, l=(1,2,…,50)	Параметры состояния серверов S_i, i=(1,2,3)								
	Расстояние от пользователя до сервера, D_s^{li}			Доступная пропускная способность канала, B_{ch}^{li}			Стоимость затрат, C_{sdr}^{li}		
	S_1	S_2	S_3	S_1	S_2	S_3	S_1	S_2	S_3
1	0,14	0,21	0,45	0,52	0,53	0,58	0,32	0,45	0,9
2	0,44	0,39	0,66	0,53	0,58	0,54	0,12	0,65	0,82
…	…	…	…	…	…	…	…	…	…
50	0,2	0,56	0,66	0,51	0,59	0,58	0,23	0,44	0,66

Задача распределения вычислительной нагрузки ставится следующим образом: требуется отправлять запросы пользователей на выбранные сервера так, чтобы временные затраты на выполнение задач были минимальны. Тогда критерий выбора сервера (с учетом параметров его состояния) для решения задачи РВН примет вид:

$$\sum_{l=1}^{q}\sum_{i=1}^{n}\alpha^{li}\,F\left(D_s^{li},U_{cpu}^{li},U_{ram}^{li}\right)\;=\;\sum_{l=1}^{q}\sum_{i=1}^{n}\alpha^{li}\,(t^{li})\;\rightarrow min, \qquad (2.8)$$

где α^{li} – признак отправки l-го запроса ($l=\overline{1,q}$, q – количество запросов), на i-ый сервер ($i=\overline{1,n}$, n – количество серверов) комплекса (α^{li} = 1, если запрос отправлен; α^{li} = 0 - если не отправлен); $F\left(D_s^{li},U_{cpu}^{li},U_{ram}^{li}\right)$ – функция, зависящая от параметров состояния серверов и характеризующая временные затраты; D_s (км) – расстояние от клиента до сервера; U_{cpu} – загруженность центрального

процессора, %; U_{ram} – загруженность оперативной памяти, %; t^{li} – время (с), затраченное i-ым сервером на выполнение l-го запроса.

Задача распределения статических данных ставится следующим образом: требуется распределить запросы пользователей на загрузку данных так, чтобы минимизировать стоимостные и временные затраты на обработку данных. Тогда критерий выбора сервера для решения задачи РСД примет вид:

$$\sum_{l=1}^{q}\sum_{i=1}^{n}\alpha^{li}\,F\left(D_b^{li}, C_{sdr}^{li}\right) = \sum_{l=1}^{q}\sum_{i=1}^{n}\alpha^{li}\,(t^{li}+c^{li}) \to min, \qquad (2.9)$$

где α^{li} – признак отправки l-го запроса ($l=\overline{1,q}$), на i-ый сервер ($i=\overline{1,n}$), $F(D_b, C_{sdr})$ - функция, зависящая от параметров состояния серверов и характеризующая временные и стоимостные затраты; D_b – произведение расстояния от клиента до сервера D_s (км) на доступную пропускную способность B_{ch} (Кбит/с) т.е. $D_b = D_s * B_{ch}$; C_{sdr} – стоимость затрат на хранение, доставку и репликацию данных (ден. ед.); t^{li} – время (с), затраченное i-ым сервером на выполнение l-го запроса. Ограничениями целевой функции являются диапазоны изменения каждого из параметров состояния серверов, записанные в виде:

$$\begin{cases} D_s^{min} \le D_s \le D_s^{max}; \\ U_{ram}^{min} \le U_{ram} \le U_{ram}^{max}; \\ U_{cpu}^{min} \le U_{cpu} \le U_{cpu}^{min}; \\ C_{sdr}^{min} \le C_{sdr} \le C_{sdr}^{max}; \\ B_{ch}^{min} \le B_{ch} \le B_{ch}^{max}. \end{cases} \qquad (2.10)$$

Представим в виде матрицы исходные данные задачи распределения нагрузки для каждого l-го запроса с учетом j-го параметра состояния i-го сервера.

$$X = \left[\begin{bmatrix} x_1^{11} & x_2^{11} & \cdots & x_m^{11} \\ x_1^{12} & x_2^{12} & x_1^{12} & \cdots \\ \cdots & \cdots & x_j^{1i} & \cdots \\ x_1^{1n} & x_2^{1n} & \cdots & x_m^{1n} \end{bmatrix} \cdots \begin{bmatrix} x_1^{l1} & x_2^{l1} & \cdots & x_m^{l1} \\ x_1^{l2} & x_2^{l2} & x_1^{l2} & \cdots \\ \cdots & \cdots & x_j^{li} & \cdots \\ x_1^{ln} & x_2^{ln} & \cdots & x_m^{ln} \end{bmatrix} \cdots \begin{bmatrix} x_1^{q1} & x_2^{q1} & \cdots & x_m^{q1} \\ x_1^{q2} & x_2^{q2} & x_1^{q2} & \cdots \\ \cdots & \cdots & x_j^{qi} & \cdots \\ x_1^{qn} & x_2^{qn} & \cdots & x_m^{qn} \end{bmatrix}\right];$$

$$l=\overline{1,q}, i=\overline{1,n}, j=\overline{1,m} \qquad (2.11)$$

Следует принять во внимание, что стандартная задача кластеризации [58, 60, 68, 69, 117] решается, если необходимо разбить по кластерам объекты, характеризующиеся векторами своих параметров. Поэтому для выполнения

процедуры кластеризации (определения нужного сервера для отправки на него задачи) для каждого l-го запроса по отношению к i-му серверу поставим в соответствие вектор, полученный взвешенным суммированием параметров состояния серверов:

$$X_{sum}^{li} = \sum_{j=1}^{m} \lambda_j x_j^{li}, l = \overline{1,q}, i = \overline{1,n}, \quad (2.12)$$

где $\lambda_j \in [0,1]$ весовой коэффициент, значение которого характеризует важность параметра (при этом каждый сервер характеризуется не набором из трех показателей, а лишь одним параметром).

Использование векторного представления X_{sum}^{li} (вместо матричного) при кластеризации позволяет характеризовать запрос набором показателей обслуживания каждым сервером. Тем самым исходные данные задачи распределения нагрузки пригодны для проведения кластерного анализа, направленного на отнесение запроса к определенному серверу.

2.2 Обоснование формирования показателей для выбора сервера на основе анализа паттернов данных о состоянии серверного комплекса

В исследовании выполнялся анализ паттернов [73-76], предполагающий парное сравнение показателей функционирования серверного комплекса, основанный на методах порядково-фиксированной и порядково-инвариантной паттерн-кластеризации [1, 3, 42, 60, 61].

Для проведения анализа паттернов необходимо сформировать обобщенные показатели, в качестве которых будем использовать средние значения параметров состояния серверов.

Использование среднего значения для паттерн-кластеризации основано на результатах, полученных в работе [42], где утверждается, что если рассматриваемые объекты принадлежат по своим параметрам одному порядково-инвариантному паттерн-кластеру, то и объект со средним значением параметров принадлежит данному кластеру.

Для каждого параметра получено среднее значение на основе массива данных тестовой выборки (см. таблицы 2.1, 2.2).

Вектор данных $\bar{X}^i$, для каждой i-ого сервера содержит средние значения параметров состояния:

$$\bar{X}^i = \left(\bar{x}_1^i, \bar{x}_2^i, \dots, \bar{x}_m^i\right), i = \overline{1,n}. \quad (2.13)$$

Компонентами вектора $\bar{x}_j^i$ являются средние значения, вычисленные на основе исходных данных, полученных в результате проведенного эксперимента для количества запросов пользователя q=50.

Среднее значение для каждого j-го параметра вычисляется по формуле:

$$\bar{x}_j^i = \frac{1}{q}\sum_{l=1}^{q} \hat{x}_j^{li}, j = \overline{1,m}, \quad (2.14)$$

где количество параметров состояния m=3.

Формирование паттернов параметров состояния серверов для задачи распределения вычислительной нагрузки произведем с помощью метода порядково-фиксированной паттерн-кластеризации. Выполним графическое представление объектов в системе параллельных координат [43].

Для этого нанесем на плоскость m параллельных, вертикальных и равномерно распределённых координатных осей, каждая из которых предназначена для отображения одного из выбранных показателей. Каждый j-ый параметр функционирования i-ой серверной серверных $\bar{x}_j^i \in \bar{X}^i$ изображается в виде кусочно-линейной функции с вершинами на параллельных осях.

Введем некоторое множество кластеров $Y = (y^1, \dots, y^h, \dots, y^k)$, где $h = \overline{1,k}$ номер сервера, а k - количество кластеров (может быть заранее не известно). При этом каждому объекту $y_h \in Y$ присваивается обозначение, соответствующее номеру кластера (номер сервера, на который будет распределяться нагрузка). Для оценки меры близости параметров использована метрика расстояния Хемминга $d\left(\bar{x}_j^i, \bar{x}_j^h\right), h = \overline{1,k}, i \neq h$ [35]. Вычислим средние значения параметров $D_s^i, U_{ram}^i, U_{cpu}^i$, для каждого i-ого сервера на основании исходных данных, приведенных в таблице

2.1, и представим результаты в таблице 2.3.

Таблица 2.3 – Параметры состояния серверов для задачи распределения вычислительной нагрузки

Обозначение сервера	Средние значения параметров состояния серверов S_i, i=(1,2,3)		
	Расстояние от пользователя до сервера, D_s^i	Загруженность оперативной памяти, U_{ram}^i	Загруженность центрального процессора, U_{cpu}^i
S_1	0,4604	0,5258	0,5502
S_2	0,7948	0,2458	0,5478
S_3	0,6028	0,7986	0,5524

Построим кусочно-линейные функции данных о состоянии серверов для задачи распределения вычислительной нагрузки (рисунок 2.4).

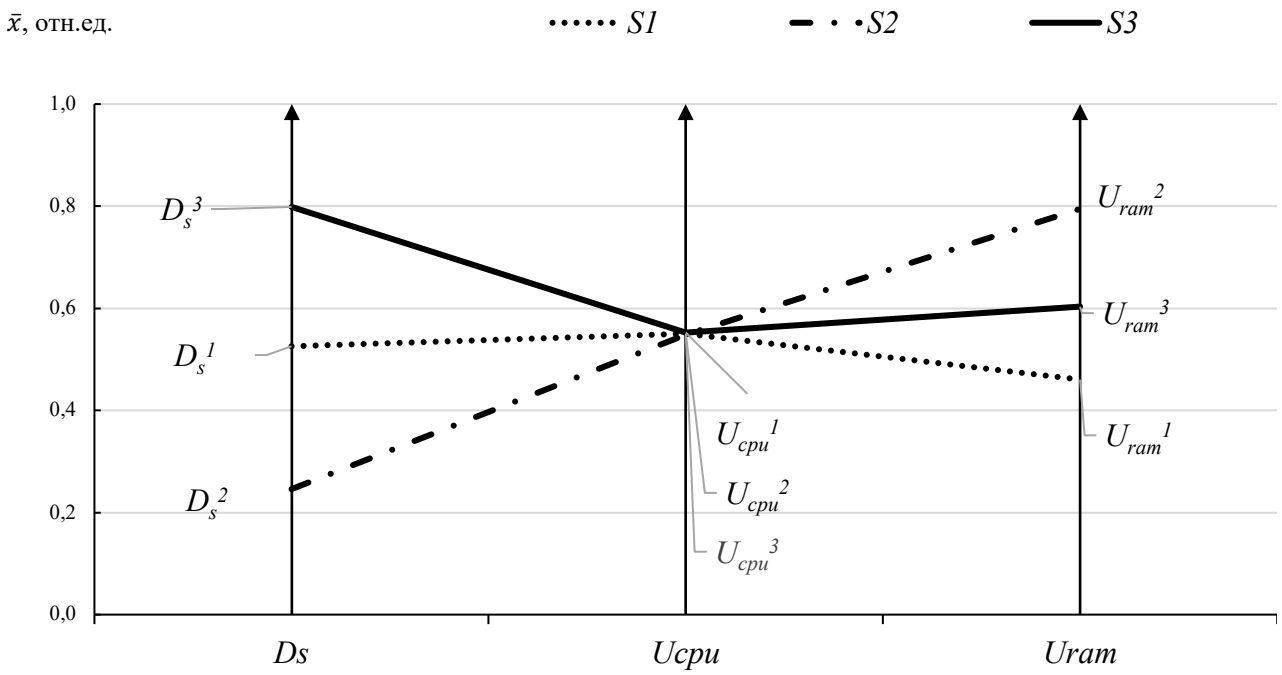


Рисунок 2.4 – Кусочно-линейные функции параметров серверов, соответствующие паттерну P_1=(D_s, U_{ram}, U_{cpu})

Так как сервера обладают разными значениями параметров состояния, то выдвинем гипотезу, что серверный комплекс можно разделить на n кластеров, каждый из которых соответствует одному из серверов. Покажем, что кластеры

различимы при использовании предлагаемого алгоритма, основанного на методе порядково-фиксированной паттерн-кластеризации.

Предлагаемый алгоритм выполняет построение кодовой последовательности, характеризующей паттерн кластера, и включает три шага. *На первом шаге* алгоритма выполняются парные сравнения смежных показателей. Параметру состояния $\bar{x}_j^i \in \bar{X}^i$ ставится в соответствие кодовая последовательность символов $r_j^i = \left(r_1^i, \dots, r_j^i, \dots, r_m^n\right)$, каждый член которой вычисляется по формуле:

$$r_j^i = \begin{cases} 1\,, & \text{если } \bar{x}_j^i < \bar{x}_{j+1}^i \\ 0, & \text{если } \bar{x}_j^i = \bar{x}_{j+1}^i\,, \\ 2, & \text{если } \bar{x}_j^i > \bar{x}_{j+1}^i \end{cases} \tag{2.15}$$

где, $i = \overline{1,n}$ – номера сервера, n – количество серверов, $j = \overline{1,m-1}$– номер параметра состояния, m – количество параметров, $\bar{x}_j^i \in \bar{X}^i$.

На втором шаге алгоритма сформированная последовательность r_j^i представляется в виде десятичного числа. В связи с тем, что параметры состояния серверов нормированы и изменяются в диапазоне $\bar{x}_j^i \in \{0,1\}$, то удобно осуществлять их попарное сравнение в соответствии с (2.15).

В [43] предложено рассматривать кодовую последовательность символов $r_j^i = \left(r_1^i, r_2^i \dots, r_j^i, \dots, r_m^i\right)$, в виде десятичного кода некоторого числа. Это позволяет заменить посимвольное сравнение кодов, соответствующих результатам парных сравнений параметров состояния серверов, на арифметическую операцию сравнения двух десятичных чисел, что позволяет уменьшить вычислительные затраты.

При этом для каждой *i*-ого сервера для всех параметров состояния формируется позиционный десятичный код z^i по формуле:

$$z^i = \sum_{j=1}^{m-1} 10^{j-1}\, r_{m-j}^i\,, i = \overline{1,n}. \tag{2.16}$$

где, n – количество серверов, j – номер параметра состояния, m – количество параметров.

Таким образом, исследуемые объекты (сервера) будем характеризовать вектором средних значений параметров состояния $\bar{X}^i$ и позиционным кодом $r_j^i = \left(r_1^i, r_2^i \dots, r_j^i, \dots, r_m^i\right)$, характеризующим парные отношения смежных параметров.

На третьем шаге алгоритма производится кластеризация параметров путем оценивания близости формируемых кодов с помощью расстояния Хемминга:

$$d\left(r_j^i, r_j^h\right) = \sum_{j=1}^{m-1} \left|r_j^i - r_j^h\right|, i = \overline{1,n}, h = \overline{1,n}, i \neq h. \quad (2.17)$$

где i – номер сервера, n – количество серверов, j – номер параметра, m – количество параметров. В результате объекты относятся к кластерам по следующему правилу: если $d\left(r_j^i, r_j^h\right) = 0$, то объекты, принадлежат одному кластеру $\bar{X}^i, \bar{X}^h \in y^i$, $i = \overline{1,n}$, $h = \overline{1,n}, i \neq h$, в противном случае, $\bar{X}^i, \bar{X}^h$ принадлежат разным кластерам y^i, y^h соответственно.

Рассмотренный алгоритм реализует порядково фиксированную паттерн-кластеризацию, в связи с тем, что последовательность параметров в нем принята фиксированной, а кластеры, полученные в результате его проведения, соответственно, порядково-фиксированными паттерн-кластерами.

Выполним обработку исходных данных, приведенных в таблице 3, для задачи распределения вычислительной нагрузки, чтобы оценить возможность разбиения серверного комплекса на группы серверов, различимых между собой по параметрам. Так как эксперимент проводится для комплекса из трех серверов, то выявим, разделим ли комплекс на три группы. В соответствии с (2.12) вектор данных для каждого сервера содержит средние значения параметров состояния:

$$\begin{aligned} \bar{X}^1 &= (\bar{x}_1^1, \bar{x}_2^1, \bar{x}_3^1) = (0{,}4604;\ 0{,}5258;\ 0{,}5502); \\ \bar{X}^2 &= (\bar{x}_1^2, \bar{x}_2^2, \bar{x}_3^2) = (0{,}7948;\ 0{,}2458;\ 0{,}5478); \\ \bar{X}^3 &= (\bar{x}_1^3, \bar{x}_2^3, \bar{x}_3^3) = (0{,}6028;\ 0{,}7986;\ 0{,}5524). \end{aligned} \quad (2.18)$$

Выполним парные сравнения параметров состояния, приведенных в (2.18), и вычислим позиционные коды паттернов. Результаты приведены в таблице 2.4.

Таблица 2.4 – Результаты порядково-фиксированной паттерн кластеризации серверного комплекса (задача распределения вычислительной нагрузки).

Обозначение сервера	Кодовая последовательность паттернов			
	Парные сравнения параметров		Кодовая последовательность	Позиционный десятичный код
	$\bar{x}_1^i, \bar{x}_2^i$	$\bar{x}_2^i, \bar{x}_3^i$	r_j^i	z
S_1	$\bar{x}_1^1 > \bar{x}_2^1$	$\bar{x}_2^1 > \bar{x}_3^1$	$r_j^1 = (r_1^1, r_2^1) = (2,2)$	22
S_2	$\bar{x}_1^2 > \bar{x}_2^2$	$\bar{x}_2^2 < \bar{x}_3^2$	$r_j^2 = (r_1^2, r_2^2) = (2,1)$	21
S_3	$\bar{x}_1^3 < \bar{x}_2^3$	$\bar{x}_2^3 > \bar{x}_3^3$	$r_j^3 = (r_1^3, r_2^3) = (1,2)$	12

Для разделения серверов по кластерам произведена оценка расстояния Хемминга между полученными кодовыми последовательностями:

$$d(r^1, r^2) = |r^1 - r^2| = |22 - 21| \neq 0;$$

$$d(r^1, r^3) = |r^1 - r^3| = |22 - 12| \neq 0;$$

$$d(r^2, r^3) = |r^2 - r^3| = |21 - 12| \neq 0.$$

Таким образом, в результате порядово-фиксированной патерн-кластеризации получено, что исходное множество параметров функционирования серверов различимо, что подтверждает выдвинутую ранее гипотезу. Поэтому выбранные параметры состояния (D_s, U_{cpu}, U_{ram}) серверов пригодны для решения задачи распределения вычислительной нагрузки. Следующим этапом подтверждения выдвинутой гипотезы о разделимости серверов при решении задачи распределения вычислительной нагрузки на основе параметров состояния является проведение порядково-инвариантной паттерн-кластеризации.

Применим метод порядково-инвариантной паттерн-кластеризации, при которой результат разбиения на кластеры не зависит от последовательности параметров в векторе состояния, для решения задачи распределения вычислительной нагрузки, рисунок 2.5. Для реализации алгоритма порядково-инвариантной паттерн-кластеризации необходимо рассмотреть все возможные последовательности параметров в векторе состояния.

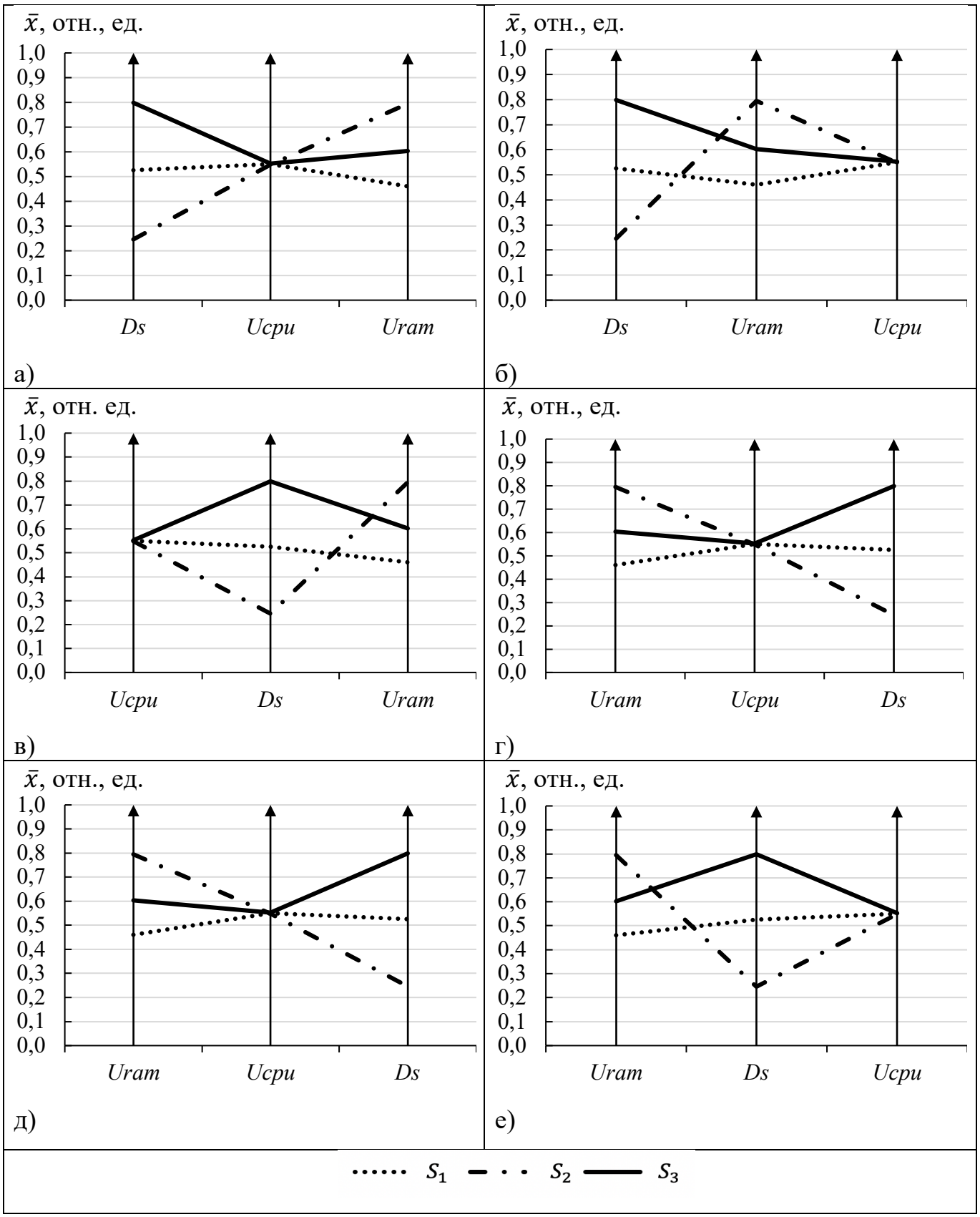


Рисунок 2.5 – Кусочно-линейные функции параметров серверов, соответствующие паттернам: а)$-P_1=(D_s, U_{cpu}, U_{ram})$, б)$-P_2=(D_s, U_{ram}, U_{cpu})$, в)$-P_3=(U_{cpu}, U_{ram}, D_s)$, г)$-P_4=(U_{cpu}, D_s, U_{ram})$, д)$-P_5=(U_{ram}, U_{cpu}, D_s)$, е)$-P_6=(U_{ram}, D_s, U_{cpu})$

Число различных комбинаций параметров в векторе состояния определяется числом перестановок $P_m = m!$. Для рассматриваемой задачи, при количестве параметров m=3 число перестановок $P_m = 6$, а результатом работы алгоритма для всех последовательностей являются сформированные порядково-инвариантные паттерн-кластеры. Построим кусочно-линейные функции, соответствующие паттернам для различных последовательностей параметров состояния серверов: D_s, U_{cpu}, U_{ram} (рисунок 2.5). Как можно видеть, для всех комбинаций параметров серверов полученные паттерны различимы. Это означает, что все сервера можно разделить по не зависимым кластерам. Стоит отметить, что использование прямой и реверсивной последовательностей параметров состояния серверов дают одинаковые результаты кластеризации. Таким образом, для того чтобы оценить результаты порядково-инвариантной паттерн-кластеризации достаточно использовать количество комбинаций равное $m!/2$, то есть, нужно рассмотреть только по три паттерна: $P_1 = (D_s, U_{cpu}, U_{ram})$; $P_2 = (U_{cpu}, D_s, U_{ram})$; $P_3 = (U_{cpu}, U_{ram}, D_s)$, для каждого сервера.

Результаты порядково-инвариантной паттерн-кластеризации для задачи распределения данных, представленные кодовыми последовательностями $r_{jp}^i = \left(r_{1p}^i, r_{2p}^i \ldots, r_{jp}^i, \ldots, r_{mp}^i\right)$, где n – количество серверов, j – номер параметра состояния, m – количество параметров, p – номер паттерна и позиционными десятичными кодами, приведены в таблице 2.5.

Таблица 2.5 – Результаты порядково-инвариантной паттерн-кластеризации для задачи распределения данных.

Обозначение сервера	Кодовые последовательности и позиционные коды паттернов					
	$P_1 = (D_s, U_{cpu}, U_{ram})$		$P_2 = (U_{cpu}, D_s, U_{ram})$		$P_3 = (U_{cpu}, U_{ram}, D_s)$	
	r_{j1}^i	z	r_{j2}^i	z	r_{j3}^i	z
S_1	$r_{j1}^1 = (r_{11}^1, r_{21}^1)$=(1, 2)	12	$r_{j2}^1 = (r_{12}^1, r_{22}^1)$=(1, 2)	12	$r_{j3}^1 = (r_{13}^3, r_{23}^3)$=(2, 1)	21
S_2	$r_{j1}^2 = (r_{11}^2, r_{21}^2)$=(1, 1)	11	$r_{j2}^3 = (r_{12}^2, r_{22}^2)$=(2, 1)	21	$r_{j3}^2 = (r_{13}^3, r_{23}^3)$=(1, 2)	12
S_3	$r_{j1}^3 = (r_{11}^3, r_{21}^3)$=(1, 1)	11	$r_{j2}^3 = (r_{12}^3, r_{22}^3)$=(1, 2)	12	$r_{j3}^3 = (r_{13}^3, r_{23}^3)$=(1, 1)	11
r^j – кодовая последовательность парных сравнений смежных показателей; z – позиционный десятичный код.						

Вычислим оценку расстояний Хемминга между полученными кодовыми последовательностями для трех последовательностей серверов:

$$P_1 - d\left(r_{j1}^1, r_{j1}^2\right) = |12 - 11| \neq 0; d\left(r_{j1}^1, r_{j1}^3\right) = |12 - 21| \neq 0; d\left(r_{j1}^2, r_{j1}^3\right) = |12 - 21| \neq 0;$$

$$P_2 - d\left(r_{j2}^1, r_{j2}^2\right) = |22 - 21| \neq 0; d\left(r_{j2}^1, r_{j2}^3\right) = |22 - 12| \neq 0; d\left(r_{j2}^2, r_{j2}^3\right) = |21 - 21| \neq 0;$$

$$P_3 - d\left(r_{j1}^1, r_{j1}^2\right) = |21 - 12| \neq 0; d\left(r_{j1}^1, r_{j1}^3\right) = |21 - 11| \neq 0; d\left(r_{j1}^2, r_{j1}^3\right) = |12 - 11| \neq 0;$$

Как видно, ни одна кодовая последовательность не повторяется, поэтому метрики расстояния между паттернами не равны нулю. Таким образом, каждый из серверов относится к отдельному кластеру, тем самым, подтверждается гипотеза о том, что исходное множество параметров состояния различимо, и параметры D_s, U_{cpu}, U_{ram} пригодны для выбора сервера в задаче распределения вычислительной нагрузки.

Так как для задачи распределения статических данных необходимо учитывать не только расстояние от пользователя D_s, но также такие показатели, как доступная пропускная способность канала передачи данных B_{ch} и стоимость затрат на хранение, доставку и репликацию данных C_{sdr}, то вектор средних значений параметров состояния:

$$\bar{X}^i = \left(\bar{x}_1^i, \bar{x}_2^i, \bar{x}_3^i\right) = (D_s^i, B_{ch}^i, C_{sdr}^i). \quad (2.19)$$

В таблице 2.6 приведены средние значения параметров вектора состояний для каждого i-ого сервера, вычисленные на основании исходных данных.

Таблица 2.6 – Параметры серверов для задачи распределения данных

Обозначение сервера	Средние значения параметров состояния серверов S_i, i=(1,2,3)		
	Расстояние от пользователя до сервера, D_s^i	Доступная пропускная способность канала передачи данных, B_{ch}^i	Стоимость затрат на хранение, доставку и репликацию данных, C_{sdr}^i
S_1	*0,286*	*0,5848*	*0,374407*
S_2	*0,5466*	*0,7526*	*0,376624*
S_3	*0,787*	*0,2672*	*0,461633*

То есть для задачи распределения данных сервера описываются следующими векторами параметров:

$$\bar{X}^1 = (\bar{x}_1^1, \bar{x}_2^1, \bar{x}_3^1) = (0{,}286;\ 0{,}584;\ 0{,}374);$$
$$\bar{X}^2 = (\bar{x}_1^2, \bar{x}_2^2, \bar{x}_3^2) = (0{,}546;\ 0{,}752;\ 0{,}376); \quad (2.20)$$
$$\bar{X}^3 = (\bar{x}_1^3, \bar{x}_2^3, \bar{x}_3^3) = (0{,}787;\ 0{,}267;\ 0{,}461).$$

Выполним парные сравнения параметров состояния в соответствии с выражением (2.14) и сформируем кодовые последовательности по выражению (2.15) алгоритма порядково-фиксированной паттерн-кластеризации, результаты приведены в таблице 2.7.

Таблица 2.7 – Результаты порядково-фиксированной паттерн кластеризации.

Обозначение сервера	Результаты парных сравнений, кодовые последовательности и позиционные коды			
	Парные сравнения параметров		Кодовая последовательность	Позиционный десятичный код
	$\bar{x}_1^i, \bar{x}_2^i$	$\bar{x}_2^i, \bar{x}_3^i$	r^i	z
S_1	$\bar{x}_1^1 < \bar{x}_2^1$	$\bar{x}_2^1 > \bar{x}_3^1$	$r^1 = (r_1^1, r_2^1) = (1\ 2)$	***12***
S_2	$\bar{x}_1^2 < \bar{x}_2^2$	$\bar{x}_2^2 > \bar{x}_3^2$	$r^2 = (r_1^2, r_2^2) = (1\ 2)$	***12***
S_3	$\bar{x}_1^3 > \bar{x}_2^3$	$\bar{x}_2^3 < \bar{x}_3^3$	$r^3 = (r_1^3, r_2^3) = (2\ 1)$	21

Построим кусочно-линейные функции, соответствующие паттернам для различных последовательностей параметров состояния: D_s, B_{ch}, C_{sdr}.

На рисунке 2.6а показано, что сервера S_1 и S_2 являются структурно близкими объектами, обладают схожими значениями параметров функционирования, что подтверждается результатами порядково-фиксированной паттерн кластеризации.

Таким образом, выдвинем гипотезу о том, что сервера S_1 и S_2 могут быть определены в один кластер. Проверим эту гипотезу, проиллюстрировав остальные кусочно-линейные функции (рисунок 2.6) и выполнив порядково-инвариантную паттерн-кластеризацию для задачи распределения данных.

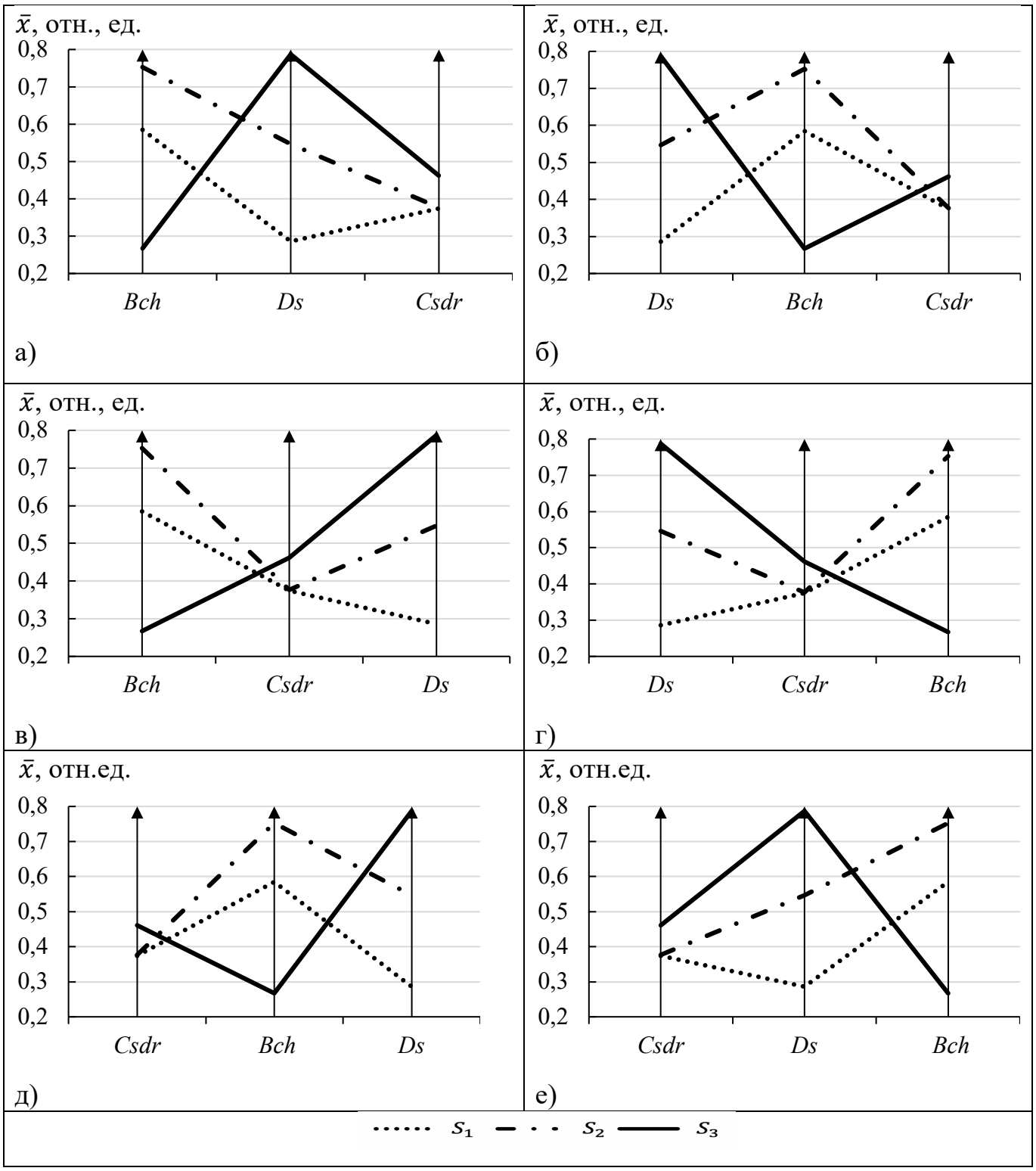


Рисунок 2.6 – Кусочно-линейные функции параметров состояния серверов, соответствующие паттернам: а) – $P_1 = (D_s, B_{ch}, C_{sdr})$, б) – $P_2 = (B_{ch}, C_{sdr}, D_s)$, в) – $P_3=(B_{ch}, D_s, C_{sdr})$, г) – $P_4 = (C_{sdr}, B_{ch}, D_s)$,д) – $P_5 = (B_{ch}, C_{sdr}, D_s)$, е) – $P_6 = (C_{sdr}, B_{ch}, D_s)$ для задачи распределения данных

Результаты паттерн-кластеризации приведены в таблице 2.8.

Таблица 2.8 – Результаты порядково-инвариантной паттерн кластеризации

Сервер	Кодовые последовательности и позиционные коды паттернов					
	$P_1 = (D_s, B_{ch}, C_{sdr})$		$P_2 = (B_{ch}, D_s, C_{sdr})$		$P_3 = (C_{sdr}, B_{ch}, D_s)$	
	r^i_{j1}	z	r^i_{j2}	z	r^i_{j3}	z
S_1	$r^1 = (r^1_{11}, r^1_{12}) = (1\ 2)$	**12**	$r^1 = (r^1_{11}, r^1_{12}) = (2\ 1)$	*21*	$r^1 = (r^1_1, r^1_2) = (2\ 2)$	*22*
S_2	$r^2 = (r^2_{21}, r^2_{22}) = (1\ 2)$	**12**	$r^2 = (r^2_{21}, r^2_{22}) = (2\ 2)$	22	$r^2 = (r^2_1, r^2_{23}) = (2\ 1)$	21
S_3	$r^3 = (r^3_{31}, r^3_{32}) = (2\ 1)$	21	$r^3 = (r^3_{31}, r^3_{32}) = (1\ 2)$	12	$r^3 = (r^3_{31}, r^3_{32}) = (1\ 1)$	11
r^i – кодовая последовательность парных сравнений; z – позиционный десятичный код.						

По результатам, приведенным в таблице 2.8, при порядково-инвариантной кластеризации структуры изменения параметров состояния серверов S_1 и S_2 различаются, то есть эти объекты отделимы. Это отличается от результатов паттерн-фиксированной кластеризации, когда было получено, что сервера S_1 и S_2 относятся к одному кластеру (их позиционные коды совпадают, см. таблицу 2.7). Выдвинутая гипотеза дополнительно проверялась с помощью кластерного анализа данных, и полученные результаты подтвердили, что сервера S_1 и S_2 трудно отличимы.

Для того чтобы улучшить отделимость серверов, было принято решение об использовании произведения параметров, характеризующих расстояние от клиента до сервера и пропускную способность канала. Применение полученного мультипликативного параметра $D_b = D_s * B_{ch}$ позволило улучшить различимость серверов. На рисунке 2.7 показаны кусочно линейные функции параметров серверов для параметров D_b, C_{sdr}.

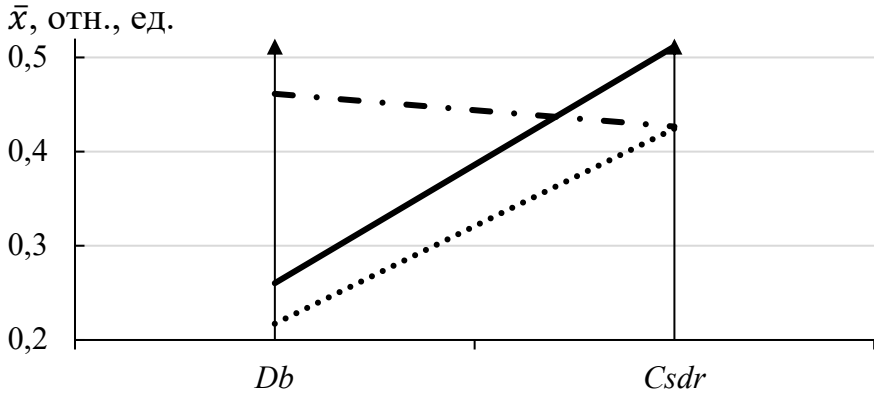


Рисунок 2.7 – Кусочно-линейные функции параметров серверов, соответствующие паттерну P_1=(D_b, C_{sdr})

2.3 Применение методов кластерного анализа данных о состоянии вычислительных ресурсов для решения задачи балансировки

Для решения задачи балансировки нагрузки и экстракции функций принадлежности проведен кластерный анализ методами *C-средних*, *K-средних*. Входными данными для кластерного анализа является вектор параметров состояния серверов X_{sum}^{li}, определяемый в соответствии с выражением (2.12). Результатом работы каждого из методов является вектор выходных значений, соответствующий точной или нечеткой принадлежности *l*-ой задачи *i*-му серверу. Рассмотрим процедуру кластеризации тестовой выборки данных с помощью алгоритма *C-средних.* Для экстракции функций принадлежности кластеризация производится для каждого параметра состояния сервера в отдельности, т.е. $\lambda_1 = 1, \lambda_2 = 0, \lambda_3 = 0$, для кластеризации по параметру D_s^{li}. Для того чтобы получить решение о распределении запроса на тот или иной сервер, используется аддитивный критерий X_{sum}^{li}, для которого $\lambda_1 = \lambda_2 = \lambda_3 = 1$.

2.3.1 Выбор сервера при решении задачи балансировки на основе разбиения серверного комплекса на кластеры

Алгоритм *k* – средних строит *n* кластеров, расположенных на возможно больших расстояниях друг от друга. Заданное фиксированное число *n* кластеров наблюдения сопоставляются кластерам так, что средние в кластере (для всех переменных) максимально возможно отличаются друг от друга. Количество кластеров *n* соответствует количеству сервера.

Метод *k-средних* разделяет *l* запросов пользователей на n серверов S_n(кластеров), чтобы минимизировать суммарное квадратичное отклонение точек кластеров от центроидов (центральных точек) этих кластеров в соответствии с формулой:

$$min\left[\sum_{i=1}^{n}\sum_{x^j \in s_i}\left|\left|x^j - \mu_i\right|\right|\right]. \qquad (2.21)$$

где $\mu_{li} \in R^n$, μ_i - центроид кластера S_i.

В качестве меры близости используется Евклидово расстояние [58]:

$$p(x,y) = ||x - y|| = \sqrt{\sum_{p=1}^{n} (x_p - y_p)^2}, \quad (2.22)$$

где $x, y \in R^n$.

Зная меру расстояния между точкой и центроидом, сводим задачу к начальному выбору центров кластеров и итерационному перестроению кластеров. Так как из исходных данных нельзя вычленить идеальный вариант распределения данных, за начальные центры кластеров примем первый набор наблюдений.

Отнесём наблюдения к тем кластерам, чье среднее (центроид) к ним ближе всего. Каждое наблюдение принадлежит только к одному кластеру, даже если его можно отнести к двум и более кластерам.

Затем центроид каждого *i*-го кластера перевычисляется по следующему правилу:

$$\mu_i = \frac{1}{s_i} \sum_{x^j \in s_i} x^j. \quad (2.23)$$

Таким образом, алгоритм метода *k* - средних заключается в перевычислении на каждом шаге центроида для каждого кластера, полученного на предыдущем шаге.

Алгоритм останавливается, когда значения μ_i не меняются: ${\mu_i}^t = {\mu_i}^{t+1}$, где $t -$ шаг работы алгоритма или количество шагов перестроения превышает заданное.

При неверном начальном выборе метод существенно замедлит приближение к оптимальному разбиению точек на кластеры. Также при определённом условии остановки итерационного процесса существует вероятность не получить полностью оптимальное разбиение кластеров. На рисунке 2.7 приведен алгоритм работы метода *k* – средних.

Применим к решению поставленной задачи метод *C* – средних [117]. Данный метод несколько сложнее метода *k* – средних и имеет принципиальные отличия.

Результатом выполнения работы метод *c* – средних является матрица нечеткого разбиения запросов пользователей по серверам *M*,

$$M = [\mu^{li}] = \begin{bmatrix} \mu^{11} & \mu^{12} & \cdots & \mu^{1n} \\ \mu^{21} & \mu^{22} & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \mu^{q1} & \mu^{q2} & \cdots & \mu^{qn} \end{bmatrix}, l = \overline{1,q}, i = \overline{1,n}, \quad (2.24)$$

где q-количество запросов пользователей выражает степень принадлежности l-го запроса к кластеру (серверу) S_i, формируемая согласно степени принадлежности по каждому из параметров состояния. Степень принадлежности $\mu^{li} \in [0, 1]$.

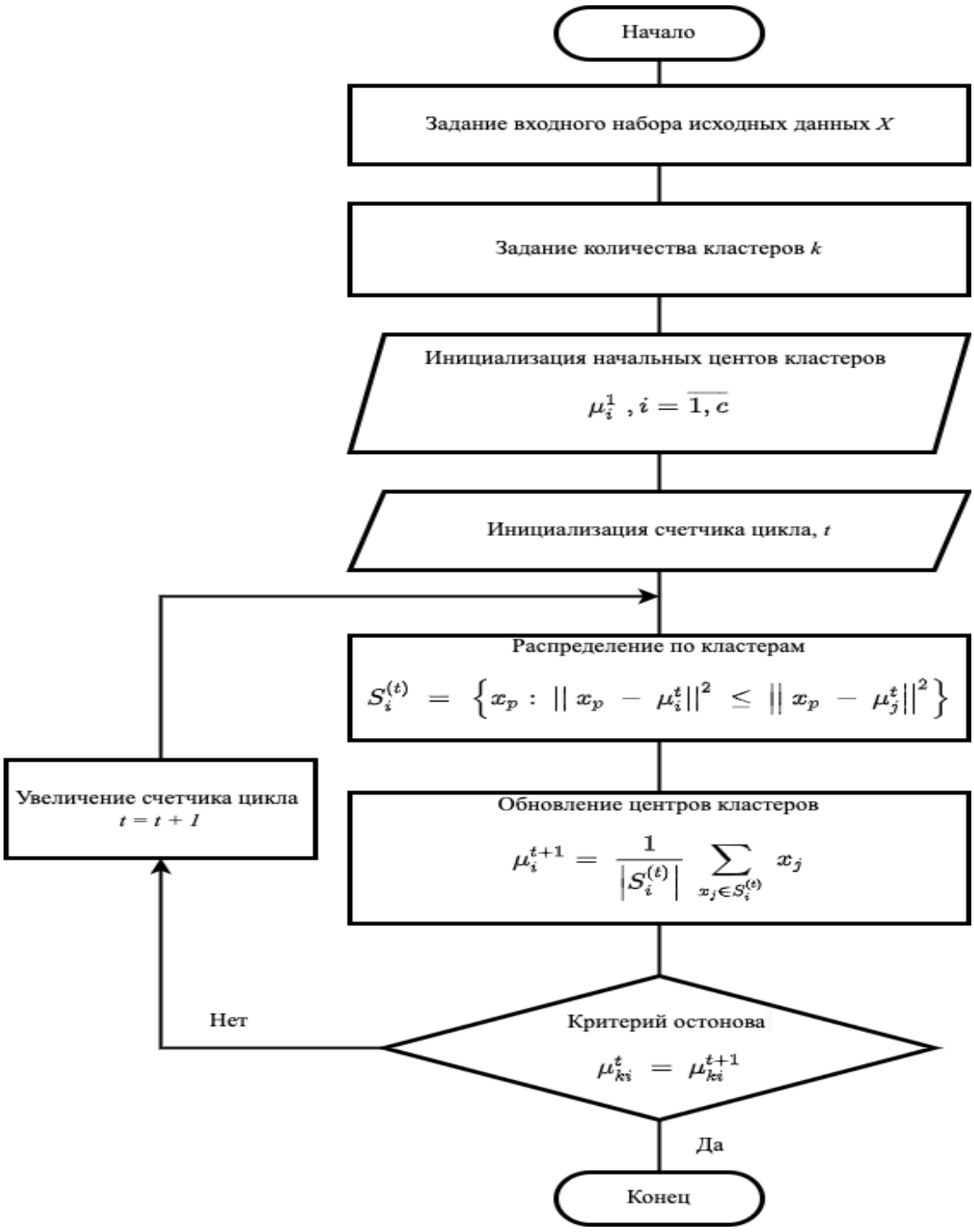


Рисунок 2.7 – Алгоритм работы метода K – средних

На рисунке 2.8 приведен алгоритм работы метода *c* – средних.

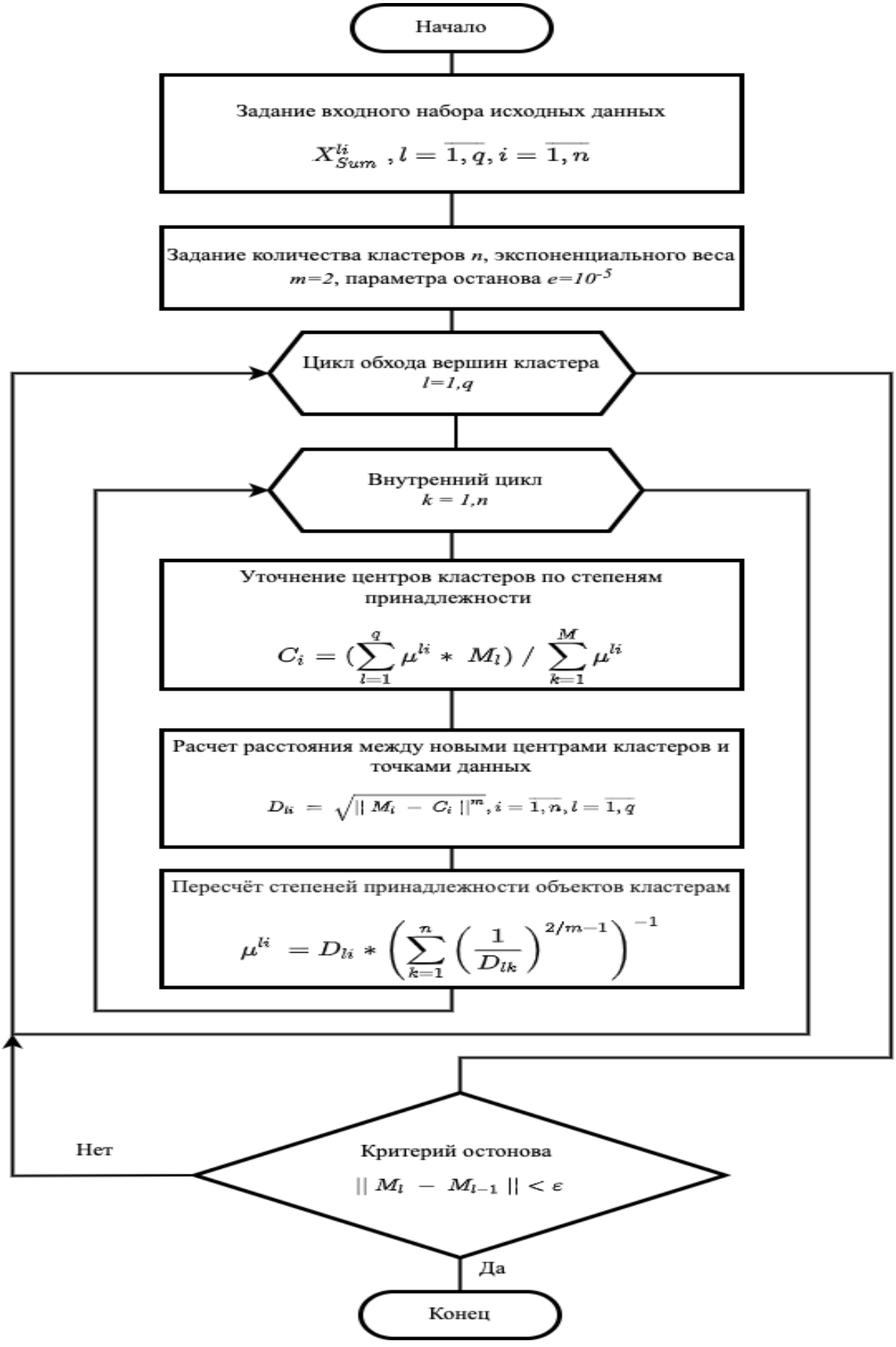


Рисунок 2.8 – Алгоритм работы метода *C* – средних

На первом шаге алгоритма задается количество кластеров, равное количеству серверов, весовой коэффициент элементов (принимаем m=2, далее этот параметр используется для расчета центров кластеров). Также принято условие завершения алгоритма, исходя из желаемой точности решения (ε=1e-5). При кластеризации алгоритмом c – средних множество X_{sum}^{li} разбивается на подмножества S_i, $i = \overline{1,n}$ со следующими свойствами:

$$\bigcup_{i=\overline{1,n}} S_i = X_{sum}^{li}; \tag{2.25}$$

$$S_i \cap S_h = \emptyset;\ i,h = \overline{1,n};\ i \neq h; \tag{2.26}$$

$$\emptyset \subset S_i \subset X^{li};\ i = \overline{1,n}. \tag{2.27}$$

Условие (2.25) указывает, что все объекты должны быть распределены по кластерам. При этом каждый объект должен принадлежать только одному кластеру (2.26) и ни один из кластеров не может быть пустым или содержать все объекты (2.27).

На втором шаге генерируется матрица нечеткого разбиения M, при этом необходимо исходить из следующих соображений. На первой итерации алгоритма матрица нечеткого разбиения заполняется случайно выбранными значением из исходного набора данных. Требуется, чтобы все объекты были распределены по каждому кластеру в соответствии с выражением:

$$\sum_{i=1}^{n} \mu^{li} = 1, l = \overline{1,q}. \tag{2.28}$$

и, кроме того, ни один кластер не является пустым множеством или содержит все элементы:

$$0 < \sum_{l=1}^{q} \mu^{li} < l, i = \overline{1,n}. \tag{2.29}$$

На третьем шаге алгоритма рассчитываются центры кластеров по формуле:

$$C_i = \frac{\sum_{l=\overline{1,q}} \left(\mu^{li}\right)^m * M_l}{\sum_{l=\overline{1,q}} (\mu_{li})^m},\ i = \overline{1,n}. \tag{2.30}$$

На четвертом шаге рассчитывается расстояние между объектами из матрицы M и центрами кластеров. Применительно к исследуемым данным каждая строка матрицы содержит индексы принадлежности конкретной задачи, каждый столбец

матрицы значение X^{li}, соответственно. Для оценки качества разбиения используется критерий разброса, показывающий сумму расстояний от объектов до центра своего кластера.

На пятом шаге алгоритма выполняется расчет элементов матрицы нечеткого разбиения запросов пользователей по формуле:

$$\mu^{li} = \left(D_{li} \times \sum_{k=1}^{n} \left(\frac{1}{D_{lk}} \right)^{2/m-1} \right)^{-1}. \quad (2.31)$$

На шестом шаге сравниваются матрицы нечеткого разбиения запросов текущего и предыдущего шага. Если $\|M_l - M_{l-1}\| < \varepsilon$, алгоритм завершается, в противном случае выполняется переход к третьему шагу. Здесь M_{l-1}- матрица нечеткого разбиения на предыдущей итерации алгоритма. На рисунке 2.8 представлен алгоритм работы метода *С-средних.*

В приведенном алгоритме самым важным параметром является количество c – кластеров. В работе количество кластеров при поступлении нового файла или вычислительной задачи в систему равно количеству серверов.

Представим результаты кластеризации данных тестовой выборки для задачи распределения данных с выбранными параметрами $X^{li} = (D_s^{li}, B_{ch}^{li}, C_{sdr}^{li})$ и с мультипликативным параметром D_b^{li} $X^{li} = (D_b^{li}, C_{sdr}^{li})$.

Покажем, как изменилась различимость между серверами при использовании мультипликативного параметр D_b^{li}. Результаты кластеризации тестовой выборки (таблица 2.2) показаны на рисунке 2.9.

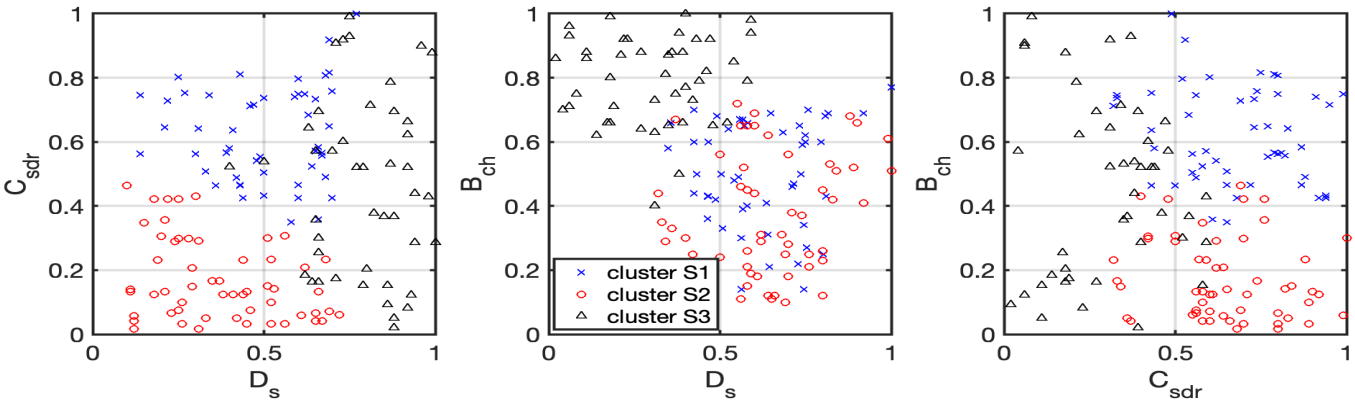


Рисунок 2.9 – Кластеризация запросов при распределении данных, параметры $D_s^{li}, B_{ch}^{li}, C_{sdr}^{li}$

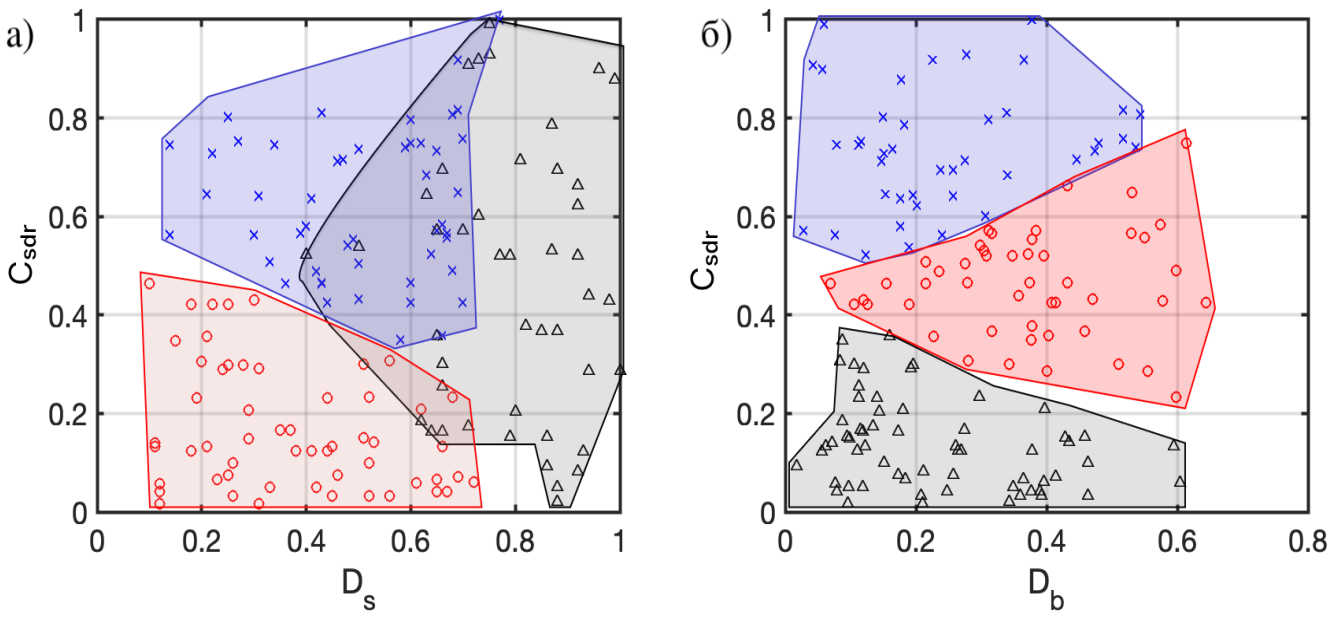


Рисунок 2.10 – Кластеризация запросов при распределении данных:
а) – параметры D_s^{li}, C_{sdr}^{li} б) – параметры D_b^{li}, C_{sdr}^{li}

Сравнив рисунки 2.9 и 2.10, пришли к выводу, что отделимость кластеров улучшилась. Кластеры на рисунке 2.10б больше не накладываются друг на друга, в отличие от рисунка 2.10а, что подтверждает возможность применения мультипликативного параметра $D_b = D_s * B_{ch}$ для проведения дальнейшей процедуры выбора сервера.

Проведем кластеризацию тестовой выборки для задачи распределения вычислительной нагрузки, результаты представлены на рисунке 2.11.

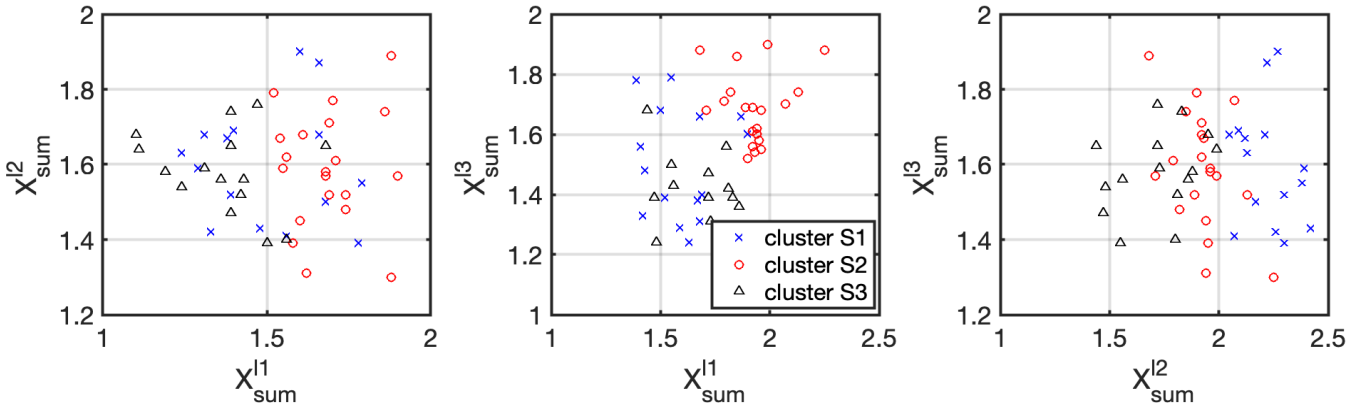


Рисунок 2.11 – Кластеризация тестовой выборки для задачи распределения вычислительной нагрузки с использованием аддитивного параметра . Запросы, распределенные на: 1-ый (+), 2-ой (×) и 3-ий (о) кластеры

Стоит отметить, что для выполнения процедуры выбора сервера с помощью методов кластерного анализа необходимо использовать аддитивный критерий. Для такого критерия каждый сервер будет идентифицирован аддитивным критерием.

Отметим, что для аддитивного параметра X_s^{li} в соответствии с (2.6) отделимость кластеров снижается, что говорит о необходимости применения дополнительных методов для выбора сервера. В качестве таких методов предлагается использовать методы нечеткой логики [43-45, 48-53, 56, 97-99, 102].

2.3.2 Формирование правил и показателей выбора сервера для алгоритма распределения вычислительной нагрузки на основе нечеткого логического вывода

Для формирования правил выбора сервера и определения диапазонов изменения значений каждого из параметров функционирования проведен кластерный анализ по каждому из параметров состояния в отдельности. Экстракция нечеткой базы правил выбора сервера заключается в выполнении следующих шагов:

- кластеризация тестовой выборки данных (таблица 2.1, 2.2) с помощью алгоритма нечетких *С-средних*, функции fcm [2, 21-23] среды Matlab, входными параметрами для которой являются массив векторов X^{li} входных значений параметров состояния каждого сервера и n - количество кластеров (серверов), результатом выполнения функции является матрица нечеткого разбиения запросов пользователей по серверам M;
- построение правил производится с помощью процедуры построения дерева решений, функции fitrtree [63, 57, 77, 101], которая принимает на вход массив векторов X^{li} и возвращает двоичное дерево, в котором каждый узел ветвления разделен на основе значений параметров состояния;
- функции принадлежности каждого параметра состояния аппроксимированы функциями принадлежности термов входных и выходных переменных.

Для построения функций принадлежности используются следующие функции: *S*-образную (smf), гауссову (gaussmf) и z-образную (zmf) [57].

Каждый нечеткий кластер отображается в нечеткое правило [45-47, 85, 86, 102, 116-118], соответственно величина степени принадлежности каждого значения входной переменной (то есть величина принадлежности параметра состояния к кластеру) будет учитываться при формировании каждого нечеткого правила.

На основе кластерного анализа и выявленных центров кластеров получены функции принадлежности термов для каждого из запросов с разной степенью принадлежности к кластерам (серверам) $S = (S_1, S_2, S_3)$, рисунки 2.12-2.16.

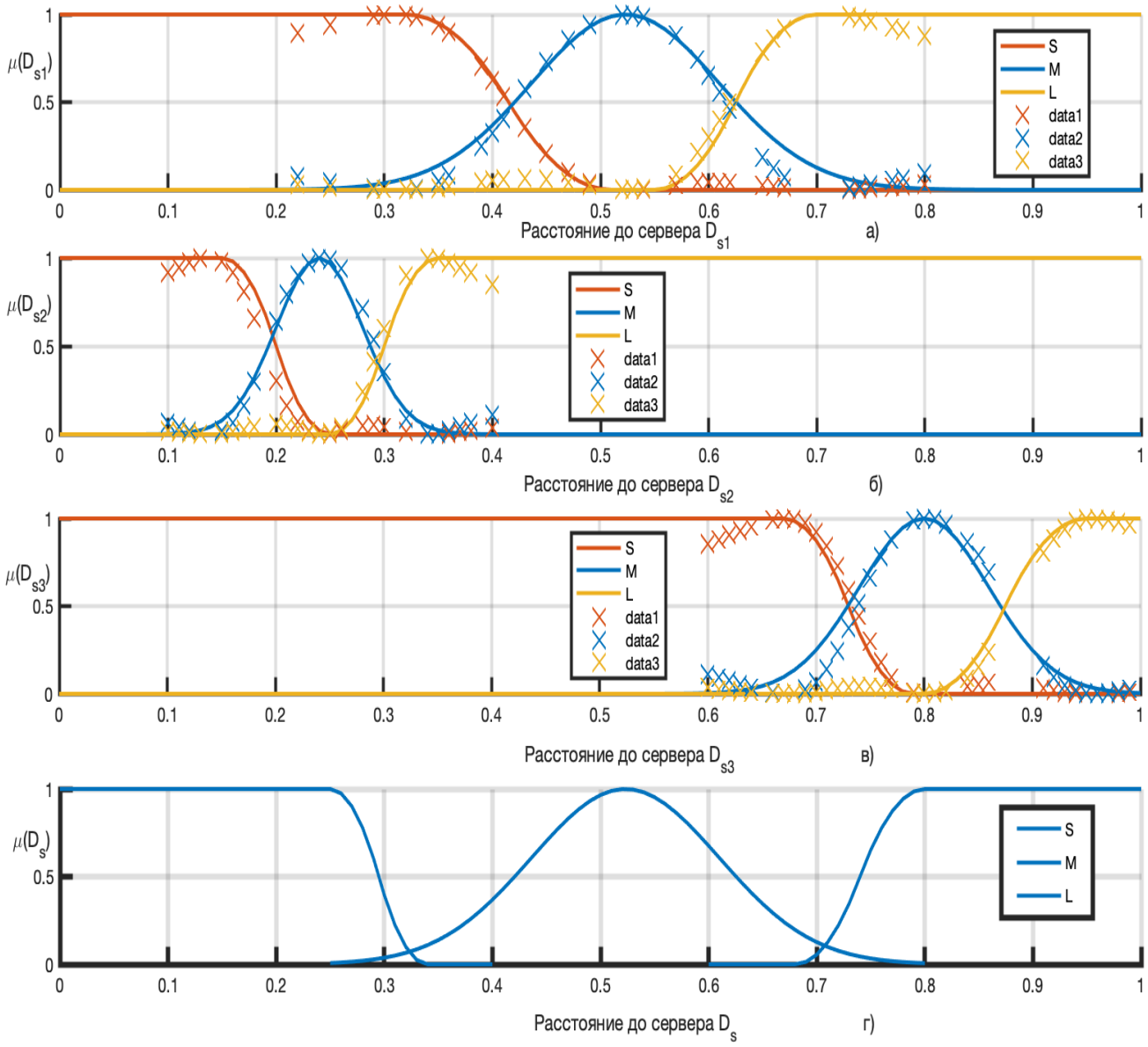


Рисунок 2.12 – Функции принадлежности, полученные в результате кластеризации, для термов входной переменной «расстояние до сервера D_s»: а) ((б), (в)) – экстракция функций принадлежности D_{s1} (D_{s2}, D_{s3}) для сервера S_1(S_2, S_3) , г) – результирующие функции принадлежности D_s к термам S, M, L

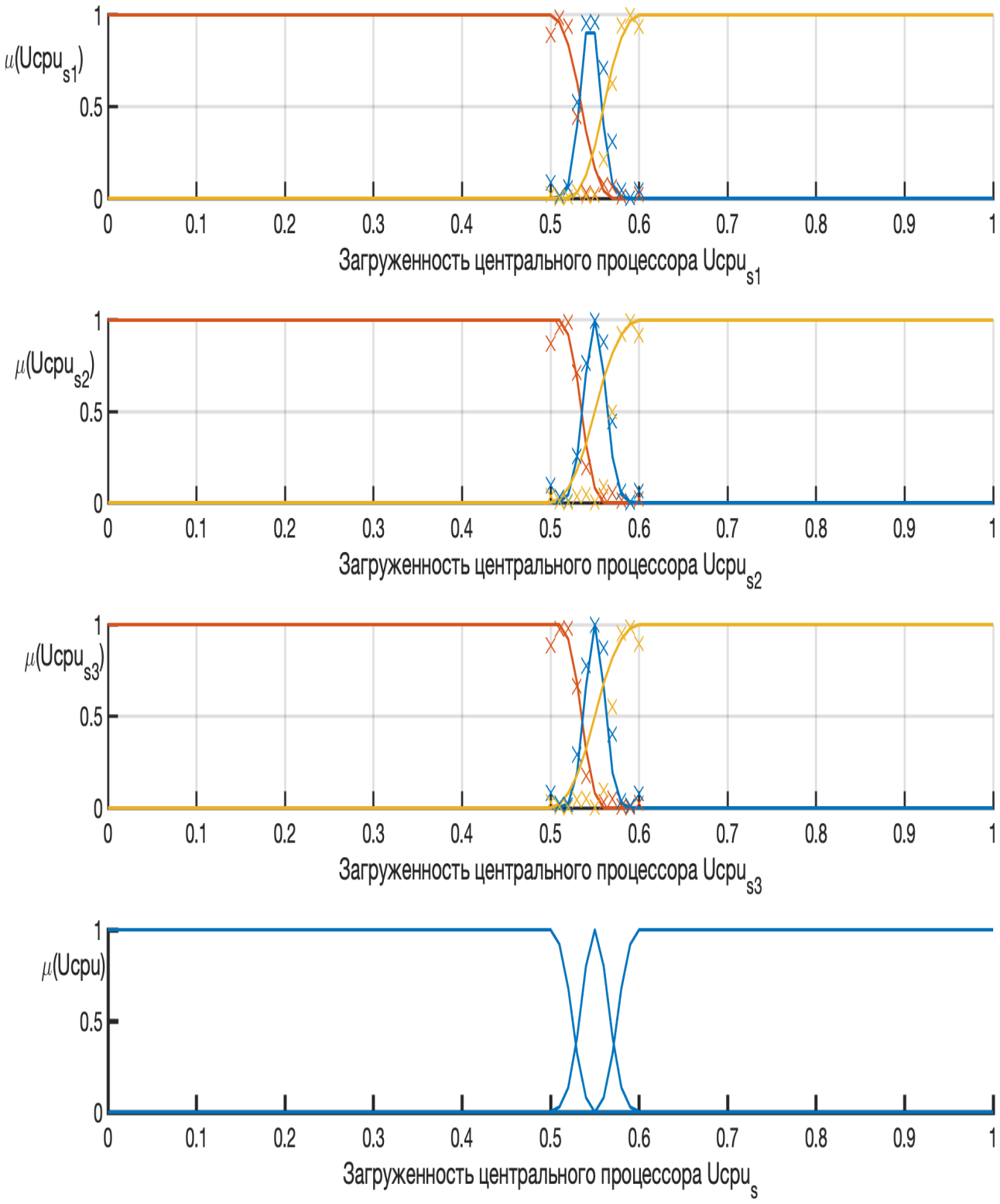


Рисунок 2.13 – Функции принадлежности, полученные в результате кластеризации, для термов входной переменной «загруженность центрального процессора U_{cpu}»: а) ((б), (в)) – экстракция функций принадлежности $U_{cpu\ s1}$ ($U_{cpu\ s2}$, $U_{cpu\ s3}$) для сервера S_1(S_2, S_3) , г) – результирующие функции принадлежности U_{cpu} к термам S, M, L

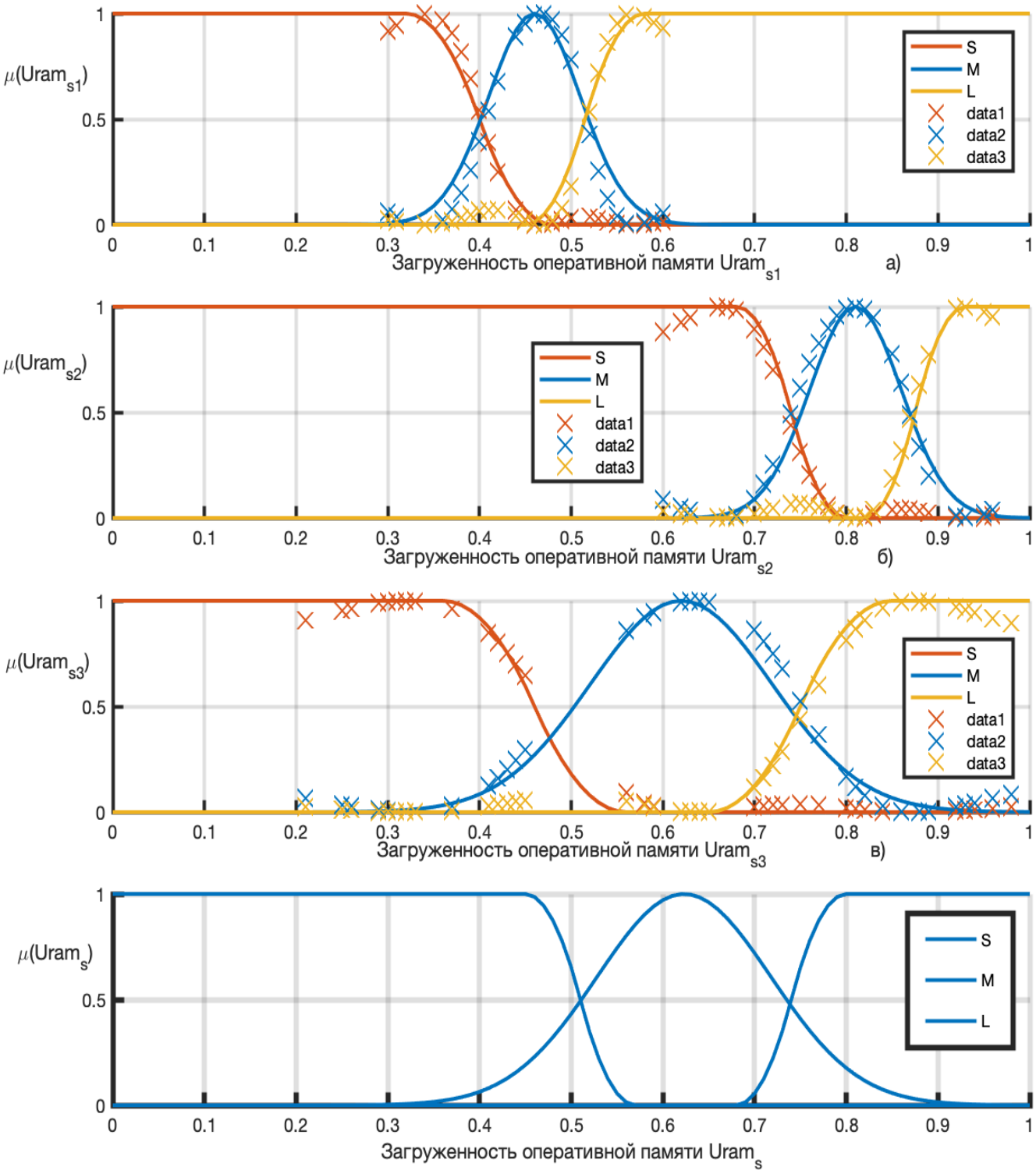


Рисунок 2.14 – Функции принадлежности, полученные в результате кластеризации, для термов входной переменной «загруженность оперативной памяти U_{ram}»: а) ((б), (в)) – экстракция функций принадлежности $U_{ram\ s1}$ ($U_{ram\ s2}$, $U_{ram\ s3}$) для сервера S_1(S_2, S_3) , г) – результирующие функции принадлежности U_{ram} к термам S, M, L

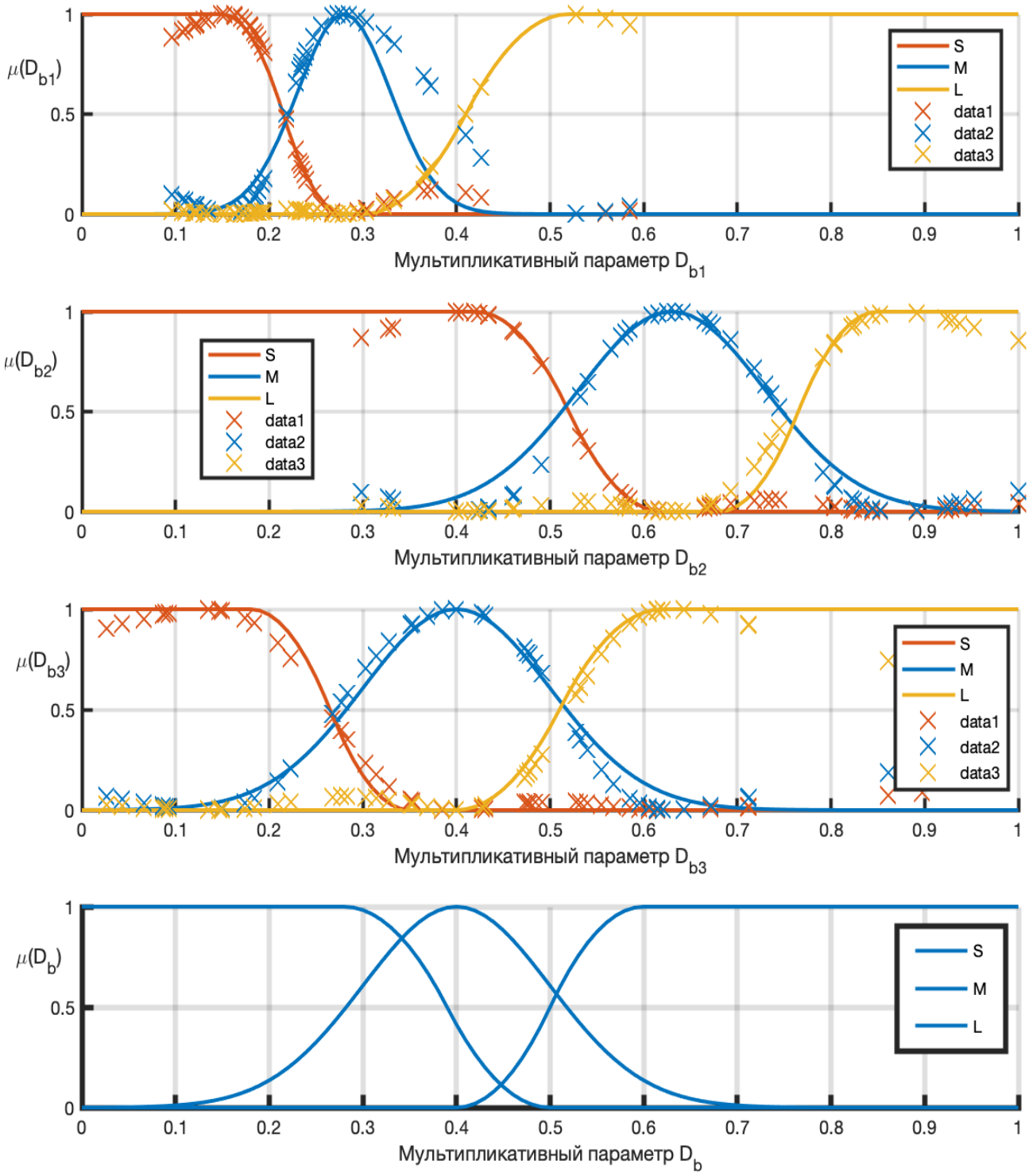


Рисунок 2.15 – Функции принадлежности, полученные в результате кластеризации, для термов входной переменной «мультипликативный параметр D_b»: а) ((б), (в)) – экстракция функций принадлежности $D_{b\,s1}$ ($D_{b\,s2}$, $D_{b\,s3}$) для сервера S_1(S_2, S_3) , г) – результирующие функции принадлежности D_b к термам S, M, L

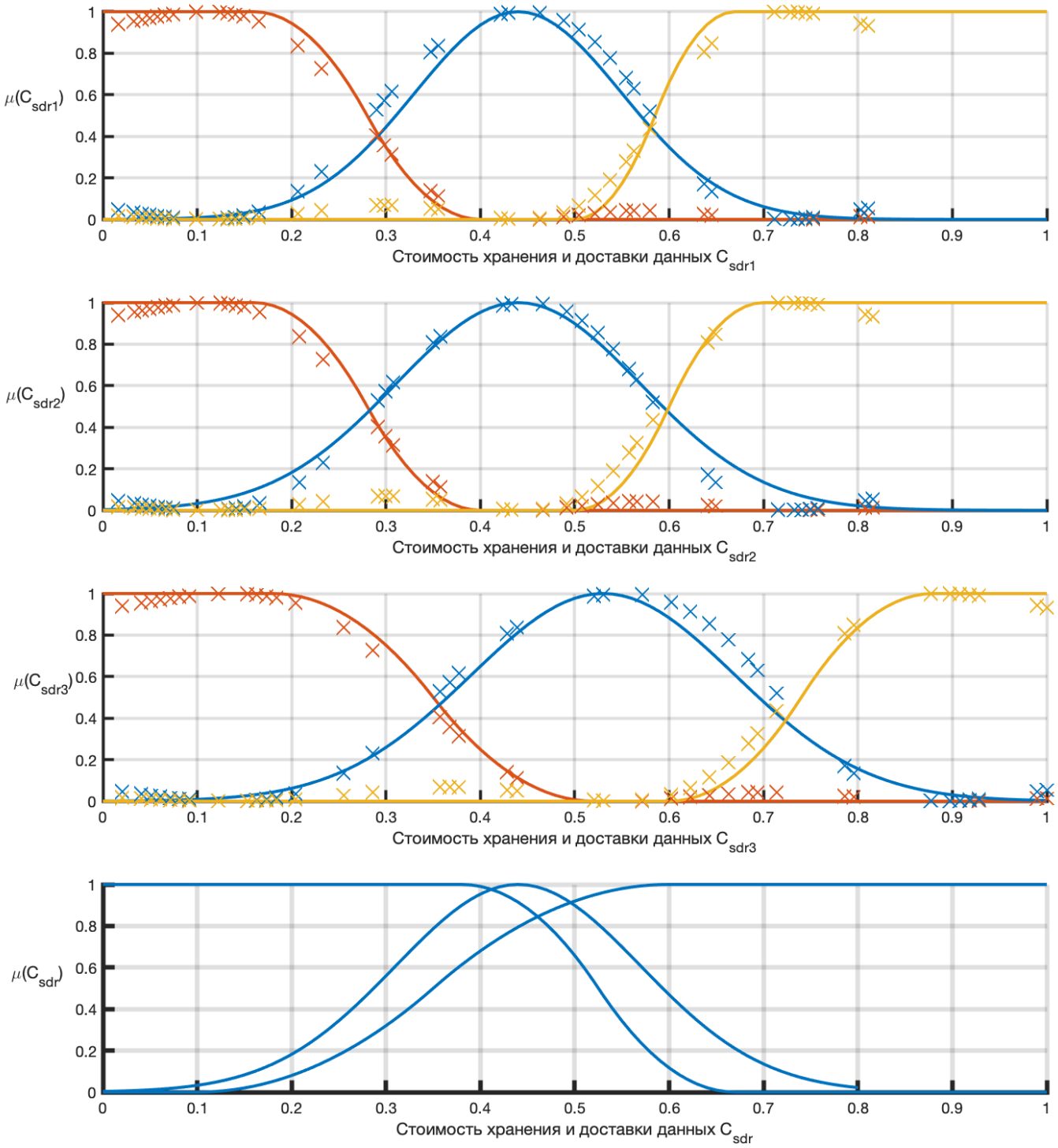


Рисунок 2.16 – Функции принадлежности, полученные в результате кластеризации, для термов входной переменной «стоимость затрат на хранение, доставку и репликацию данных C_{sdr}»: а) ((б), (в)) – экстракция функций принадлежности $C_{sdr\,s1}$ ($C_{sdr\,s2}$, $C_{sdr\,s3}$) для сервера S_1(S_2, S_3), г) – результирующие функции принадлежности C_{sdr} к термам S, M, L

Диапазоны изменения параметров серверов для задач распределения нагрузки и данных приведены в таблицах 2.9, 2.10.

Таблица 2.9 – Диапазоны изменения параметров состояния серверов (задача распределения вычислительной нагрузки)

Обозначение сервера	Параметры состояния сервера					
	Загруженность центрального процессора		Загруженность оперативной памяти		Расстояние от пользователя	
	U_{cpu}^{min}	U_{cpu}^{max}	U_{ram}^{min}	U_{ram}^{max}	D_s^{min}	D_s^{max}
S_1	0,5	0,5	0,22	0,80	0,3	0,6
S_2	0,5	0,6	0,10	0,40	0,60	0,96
S_3	0,5	0,6	0,60	0,99	0,21	1,00

Таблица 2.10 – Диапазоны изменения параметров состояния серверов (задача распределения данных)

Обозначение сервера	Параметры состояния сервера			
	Стоимость затрат на обработку запроса		Мультипликативный параметр	
	C_{sdr}^{min}	C_{sdr}^{max}	D_b^{min}	D_b^{max}
S_1	0,016552	0,01665	0,30	0,60
S_2	0,016665	0,815853	0,10	0,96
S_3	0,020408	1,00	0,21	1,00

Построение нечетких правил вида «ЕСЛИ-ТО» проведено с помощью процедуры построения бинарного дерева fitrtree [57], среды MATLAB, на основании исходных данных (таблица 2.1,2.2) полученные результаты представлены на рисунках 2.17, 2.18 и в таблицах 2.11, 2.12.

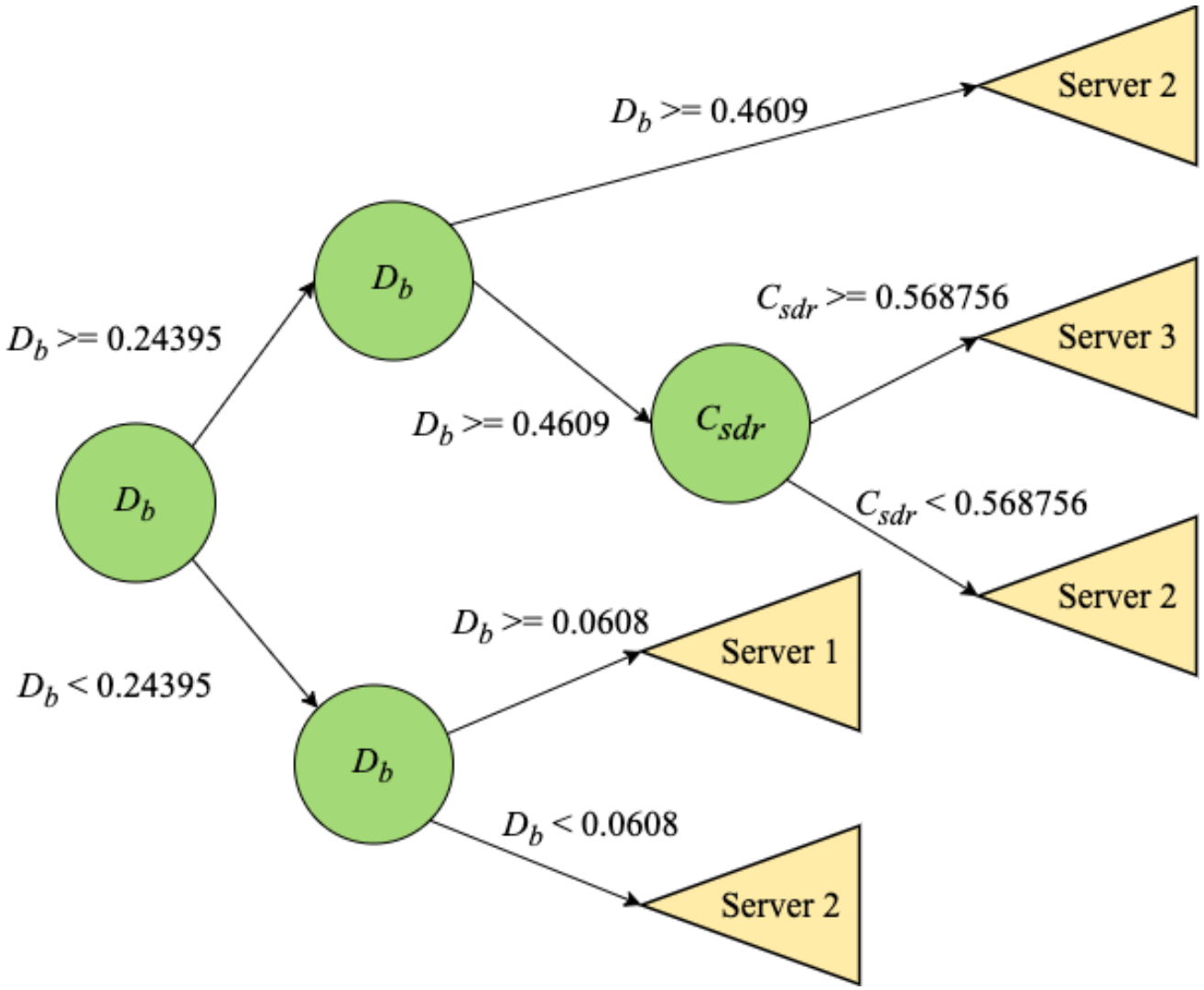


Рисунок 2.17 – Дерево решений для задачи РСД

Лингвистические термы переменных обозначим *S* – малое, *M* - среднее, *L* – большое значение параметра состояния сервера, *R* – решение о выборе сервера, принимающее значение 1 – если сервер выбран, и 0 в противном случае.

Таблица 2.11 – База нечетких правил для задачи распределения статических данных

Обозначение сервера	Нечеткие правила
S_1	***ЕСЛИ*** $D_b = S$, ***ТО*** $R = 1$;
S_2	***ЕСЛИ*** $D_b = L$, ***ТО*** $R = 1$; ***ЕСЛИ*** $D_b = M$ *И* ($C_{sdr} = S$ *ИЛИ* $C_{sdr} = M$), ***ТО*** $R = 1$;
S_3	***ЕСЛИ*** $D_b = S$, ***ТО*** $R = 1$; ***ЕСЛИ*** $D_b = M$ *И* $C_{sdr} = L$, ***ТО*** $R = 1$;

На рисунке 2.18 показано дерево принятия решений для задачи распределения вычислительной нагрузки.

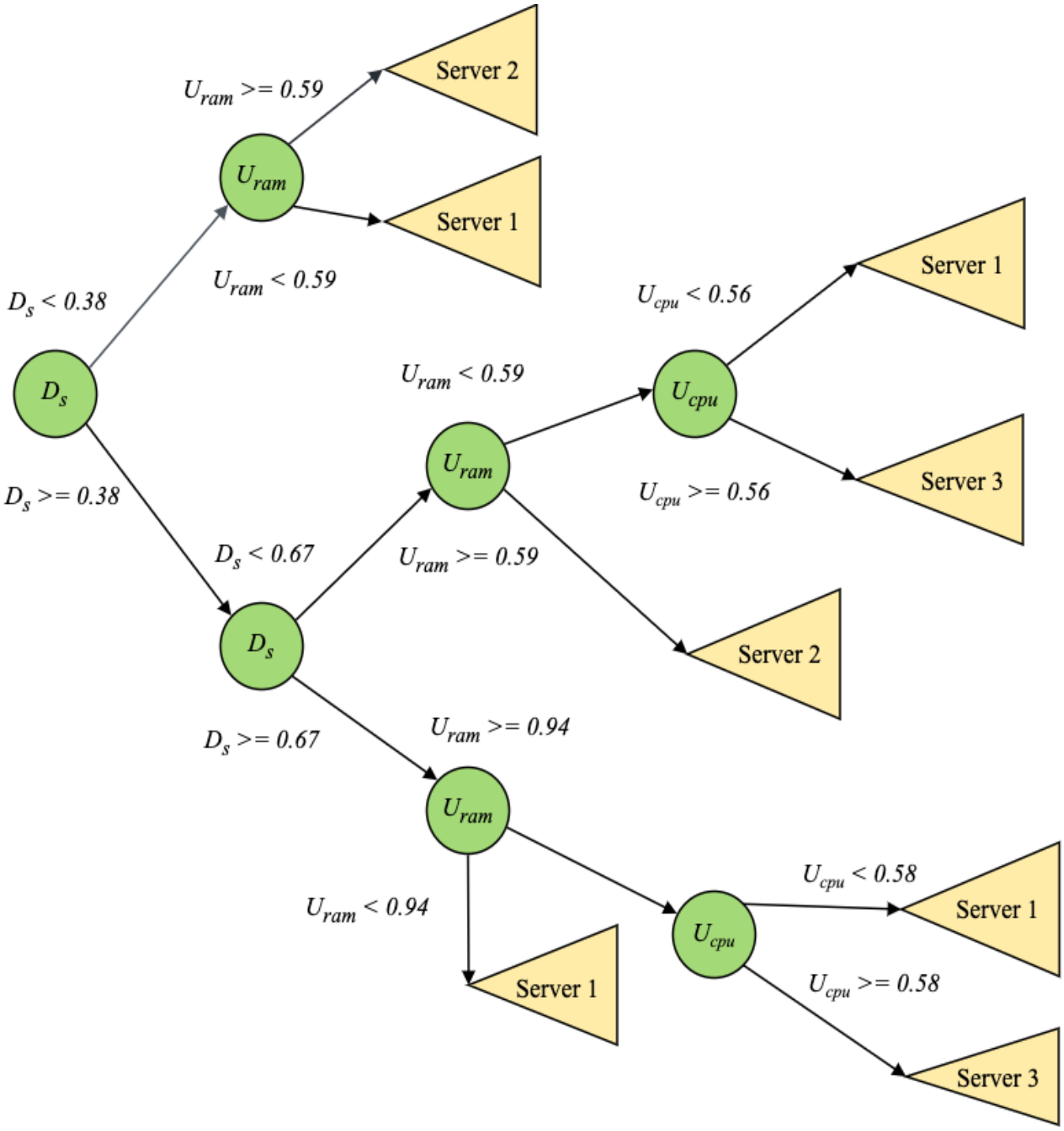


Рисунок 2.18 – Дерево решений для задачи РВН

Диапазоны изменения значений параметров состояния сервера использованы в качестве исходных данных для имитационной модели. А база нечетких правил применена для натурных исследований.

Таблица 2.12 – База нечетких правил для задачи распределения вычислительной нагрузки

Обозначение сервера	Нечеткие правила
S_1	П1: ***ЕСЛИ*** $D_s = S$ *И* ($U_{ram} = S$ *ИЛИ* $U_{ram} = M$) *И* $U_{cpu} = L$, ***ТО*** $R = 1$; П2: ***ЕСЛИ*** $D_s = L$ *И* ($U_{ram} = S$ *ИЛИ* $U_{ram} = M$) *И* $U_{cpu} = S$, ***ТО*** $R = 1$; П3: ***ЕСЛИ*** $D_s = L$ *И* ($U_{ram} = S$ *ИЛИ* $U_{ram} = M$), ***ТО*** $R = 1$;
S_2	П1: ***ЕСЛИ*** ($D_s = S$ *ИЛИ* $D_s = M$) *И* $U_{ram} = L$, ***ТО*** $R = 1$; П2: ***ЕСЛИ*** $D_s = S$ *И* ($U_{ram} = S$ *ИЛИ* $U_{ram} = M$) *И* ($U_{cpu} = S$ *ИЛИ* $U_{cpu} = M$), ***ТО*** $R = 1$; П3: ***ЕСЛИ*** $D_s = S$ *И* $U_{ram} = L$, ***ТО*** $R = 1$;
S_3	П1: ***ЕСЛИ*** $D_s = S$ *И* ($U_{ram} = S$ *ИЛИ* $U_{ram} = M$) *И* $U_{cpu} = L$, ***ТО*** $R = 1$; П2: ***ЕСЛИ*** $D_s = L$ *И* ($U_{ram} = S$ *ИЛИ* $U_{ram} = M$) *И* $U_{cpu} = S$, ***ТО*** $R = 1$;

В таблице 2.13 приведен полный список правил выбора сервера для термов *S*, *M*, *L* и параметров D_s, U_{ram}, U_{cpu}. Серым цветом выделены решения, полученные на основании дерева решений.

Таблица 2.13 – Правила выбора сервера, полученные из дерева решений для РВН

Параметр U_{cpu}	Параметр U_{ram}		
	S	*M*	*L*
При параметре $D_s = S$			
S	*Server* 1	*Server* 2	*Server* 2
M	*Server* 1	*Server* 2	*Server* 2
L	*Server* 1	*Server* 2	*Server* 2
При параметре $D_s = M$			
S	*Server* 1	*Server* 2	*Server* 3
M	*Server* 1	*Server* 2	*Server* 3
L	*Server* 3	*Server* 2	*Server* 3
При параметре $D_s = L$			
S	*Server* 1	*Server* 1	*Server* 1
M	*Server* 1	*Server* 1	*Server* 3
L	*Server* 1	*Server* 2	*Server* 3

Как видно из таблицы, полученное на основе данных обучающей выборки

дерево решений покрывает только 20 из 27 возможных решений, что является недостатком для дальнейшего использования дерева решений в качестве средства выбора сервера.

Рассмотрим пример выполнения *l*-го запроса пользователя.

$$\begin{bmatrix} D_s^{l1} & U_{ram}^{l1} & U_{ram}^{l1} \\ D_s^{l2} & U_{ram}^{l2} & U_{cpu}^{l2} \\ D_s^{l3} & U_{ram}^{l3} & U_{cpu}^{l3} \end{bmatrix} = \begin{bmatrix} 0{,}25 & 0{,}31 & 0{,}7 \\ 0{,}33 & 0{,}2 & 1 \\ 0{,}75 & 0{,}8 & 0{,}2 \end{bmatrix} => Server\ 1$$

В результате использования правил выбора сервера получено, что наиболее подходящим для вычислений является сервер под номером 1.

Для того чтобы оценить пригодность разработанного метода выбора сервера проведен эксперимент по распределению тестовых выборок размерностью от одного до ста пятидесяти запросов (задача РВН). Запросы были распределены с помощью кругового распределения Round Robin [11] и разработанного метода. Результаты проведенных экспериментов представлены на рисунке 2.19.

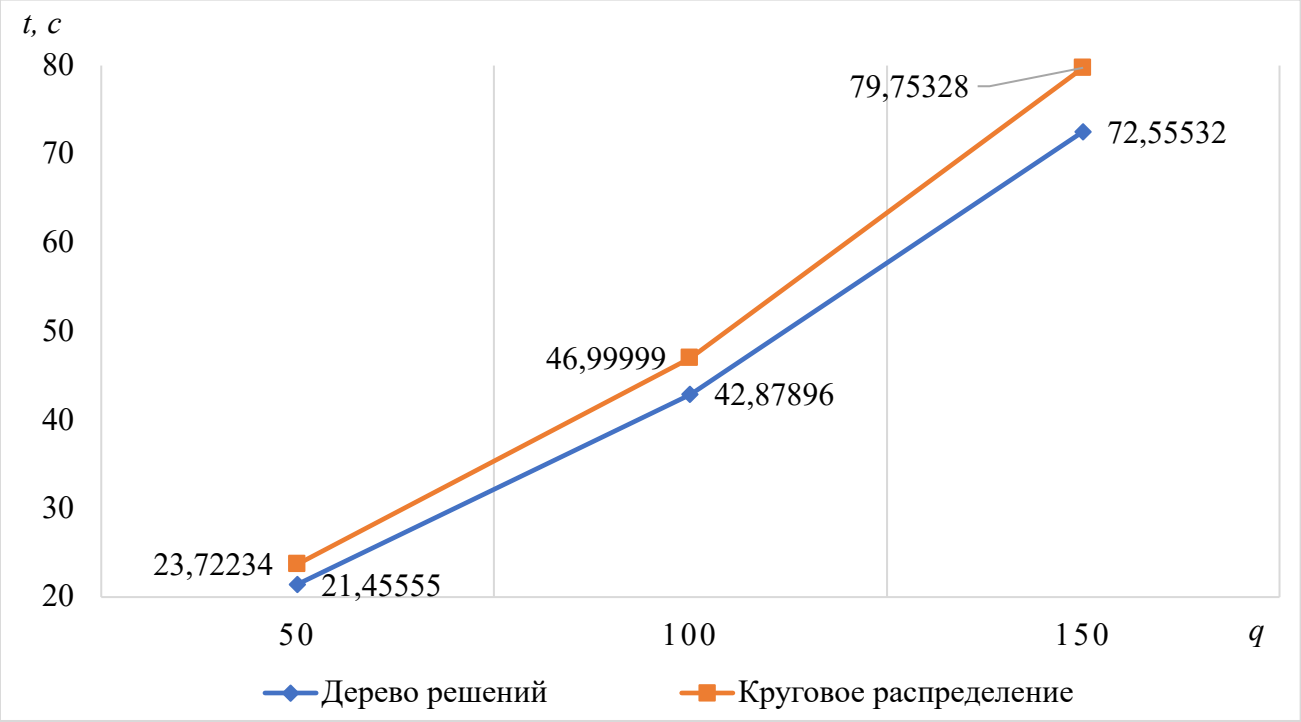


Рисунок 2.19 – Зависимость времени выполнения запросов от размера выборки для кругового распределения и распределения, основанного на дереве решений

Показано, что быстродействие в системе увеличилось. Время выполнения запросов пользователя при распределении нагрузки с помощью дерева решений (характеризуемое значением целевой функции 2.8) уменьшилось по сравнению со

временем обработки запросов круговым распределение Round-Robin. Наибольший выигрыш по времени получен для выборки из 150-ти запросов (**9,7%**). Это объясняется тем, что учет параметров состояния серверов (расстояния до сервера и загруженность его аппаратных ресурсов) позволяет серверу-балансиру посылать запросы для обработки на сервер, находящийся на наиболее близком расстоянии от пользователей, а также наименее загруженный в текущий момент времени.

2.4 Выводы по главе

В результате проведения исследований сформулирована задача выбора сервера для распределения нагрузки, разработаны методы и алгоритмы обработки данных о состоянии вычислительных ресурсов для использования при балансировке. Выполнено следующее:

1. Обоснованы показатели (параметры состояния серверного комплекса) для выбора сервера на основе паттерн-кластеризации. Показано, что для задачи распределения статических данных применение порядково-инвариантной кластеризации позволяет выявить необходимость привлечения мультипликативного показателя (произведение сетевых параметров) для улучшения отделимости запросов по отношению к серверам.

2. Предложено для кластеризации в качестве исходных данных для каждого запроса использовать вектор, полученный взвешенным суммированием параметров состояния для каждого сервера. Использование векторного представления (вместо матричного) при кластеризации позволяет характеризовать запрос набором показателей обслуживания каждым сервером. Тем самым исходные данные задачи распределения нагрузки пригодны для проведения кластерного анализа, направленного на отнесение запроса к определенному серверу.

3. На основе метода кластерного анализа *C-means* выполнена экстракция функций принадлежности параметров состояния к лингвистическим термам, характеризующим параметры состояния серверов, и получены диапазоны изменения параметров.

3 АНАЛИТИКО-ИМИТАЦИОННЫЙ МЕТОД РАСПРЕДЕЛЕНИЯ ВЫЧИСЛИТЕЛЬНОЙ НАГРУЗКИ МЕЖДУ СЕРВЕРАМИ

В третьей главе **описан** аналитико-имитационный метод балансировки нагрузки комплекса серверов. Представлена имитационная модель распределения нагрузки для задач балансировки статических данных и вычислений. Проведены имитационные эксперименты распределения нагрузки. Показаны результаты имитационных исследований балансировки нагрузки по серверам на основе нечеткого логического вывода.

3.1 Построение имитационной модели распределения вычислительной нагрузки в серверном комплексе

Для описания объектов, функционирующих в условиях действия случайных факторов, используется класс математических моделей, называемых системами массового обслуживания (СМО). Функционирование этих объектов носит характер обслуживания поступающих в систему запросов. Для выполнения совокупности действий, включаемых в понятие “обслуживание”, система располагает наборами оборудования, называемых обслуживающими каналами или линиями.

Обычно предполагается, что запросы на обслуживание образуют поток, то есть последовательность запросов, поступающих в систему через разные промежутки времени. Общее понятие потока основано на предположении, что события будут происходить в заранее неизвестные случайные моменты. Математическое моделирование систем массового обслуживания предполагает моделирование потока запросов и работы обслуживающих каналов.

Для описания СМО необходимо задать:

– входящий поток требований или запросов, которые поступают на обслуживание;

– порядок постановки запроса в очередь и выбора из нее;

– правило, по которому осуществляется обслуживание.

Чтобы описать входящий поток требований надо описать моменты времени их

поступления в систему и количество требований, которые поступили одновременно. Закон поступления может быть детерминированный или вероятностный. В общем случае входящий поток требований описывается распределением вероятностей интервалов времени между соседними требованиями.

Правила обслуживания характеризуются длительностью обслуживания (распределением времени обслуживания), количеством требований, которые поступили одновременно и дисциплиной обслуживания. Для характеристики свойств линии задается длительность обслуживания запроса или время занятости линии, как случайная величина с заданным законом распределения.

В общем случае обслуживающая система может состоять из нескольких линий, способных одновременно и независимо друг от друга обслуживать запросы. В любой момент времени линия находится в одном из двух состояний: свободном или занятом. Системы, обслуживаемые с помощью одной линии, называются одноканальными системами. Системы с несколькими линиями – многоканальными.

Относительно порядка принятия запросов в том случае, когда в системе имеется очередь запросов, используются следующие правила.

1. Запросы принимаются к обслуживанию в порядке поступления. Освободившаяся линия приступает к обслуживанию того запроса, который ранее других поступил в систему. Такую дисциплину называют “раньше поступил – раньше обслужился” (в англоязычной литературе FIFO – First In - First Out).

2. Запросы принимаются к обслуживанию по минимальному времени получения отказа. Освободившаяся линия приступает к обслуживанию того запроса, который в кратчайшее время может получить отказ.

3. Запросы принимаются к обслуживанию в случайном [25] порядке в соответствии с заданными вероятностями (RANDOM).

Реальный процесс массового обслуживания состоит из последовательности фаз обслуживания, выполняемой различными устройствами. При этом следующее устройство приступает к обслуживанию запроса и тогда, когда работа предыдущего с данного запроса завершена.

Имитационная модель СМО – это модель, отражающая поведение системы и

изменения ее состояния во времени при заданных потоках запросов, поступающих на входы системы [33]. Выходными параметрами являются величины, характеризующие качество функционирования системы, например, такие как:

– коэффициенты использования каналов обслуживания;

– максимальная и средняя длина очередей в системе;

– среднее время нахождения запросов в очередях и каналах обслуживания.

Рассмотрим возможности среды MATLAB [2, 11, 26, 56] для реализации имитационной модели системы распределения нагрузки как процесса массового обслуживания [21-24, 28, 29, 39]. Для дискретно-событийного моделирования в среде MATLAB / *Simulink* [78, 105] используется компонента *SimEvents* [87]. Программные средства *SimEvents* и *Simulink* создают интегрированную среду для моделирования гибридных динамических систем [47, 59, 64, 67, 68], содержащих непрерывные компоненты и компоненты с дискретными событиями и дискретным временем [78].

При моделировании дискретных событий состояния системы изменяются в результате наступления асинхронных дискретных инцидентов, которые называются событиями. В противоположность этому моделирование можно осуществлять на основе времени (когда состояния системы зависят от времени). Так средства *Simulink* служат для моделирования на основе времени, а средства *SimEvents* предназначены для моделирования дискретных событий [40, 41, 43, 61-65, 75, 81].

При дискретно-событийном моделировании используется понятие сущности (*entity*). Сущности могут перемещаться через сети очередей (*queues*), серверов (*servers)* и переключателей (*switches*), управляемых дискретными событиями, в процессе моделирования. Графические блоки *SimEvents* представляют компоненты, которые обрабатывают сущности, при этом сущности не имеют графического представления.

Под событием (*event*) понимается мгновенное дискретное явление, которое изменяет переменную состояния, выход блока модели или является причиной появления других событий.

Событиями в модели, построенной средствами *SimEvents,* являются, например, перемещения сущности от одного блока к другому, а также завершения

обслуживания сущности в сервере.

При дискретно-событийном моделировании очереди (*queues*) хранят сущности в течение некоторого заранее неизвестного интервала времени. Очередь выпускает сущность так быстро, как быстро следующий блок может принять новую сущность.

Свойствами очереди являются:

- емкость (*capacity*), то есть максимальное количество сущностей, которые очередь может хранить одновременно;

- дисциплина очереди, определяющая, какая из сущностей покинет очередь первой (если хранится несколько сущностей).

Сервер (server) хранит сущности в течение некоторого промежутка времени, называемого временем обслуживания (service time) и затем может выпустить сущность.

Во время периода обслуживания говорят, что сервер обслуживает сущность. Время обслуживания для каждой сущности вычисляется в момент ее прибытия в сервер. В отличие от этого время хранения блока в очереди заранее неизвестно. Но если следующий блок не принимает сущность, которую уже обслужил сервер, то сервер должен продолжить хранить сущность.

Отличительными свойствами сервера являются:

– число сущностей, которые можно обслужить одновременно;

– методы определения времени обслуживания сущностей;

– возможность прибывающим сущностям занимать сервер при наличии других сущностей (сервер с конечной емкостью при заполнении может не принимать к обработке новые сущности).

Линии для передачи сущностей (entity connection line) осуществляют связь между двумя блоками (или между их входными и выходными портами для сущностей) с помощью отображения пути, по которому сущность может покинуть один из блоков или одновременно прибыть в следующий блок.

При моделировании сущность, которая покидает выходной порт (OUT), одновременно прибывает в входной порт (IN) следующего связанного блока. Сущности, обработка которых завершена, поглощаются в блоке Entity Sink.

Сигналы в модели, построенной средствами SimEvents, представляют собой численные величины, определенные в любой момент времени в течение процесса моделирования. Сигналы появляются на соединительных линиях между сигнальными портами двух блоков.

В работе предлагается осуществлять балансировку нагрузки с помощью переключателя серверов, реализованного с использованием математического аппарата нечеткой логики [31].

Применение нечеткого логического вывода позволяет принимать решение (в данном случае при выборе сервера) в условиях, когда данные о параметрах объекта являются недостаточно определенными.

Для того, чтобы разработать и исследовать систему балансировки нагрузки, создана имитационная модель распределения данных между серверами, реализованная с помощью инструментов моделирования пакета MATLAB / Simulink / SimEvents [105, 106].

Рассмотрим схему модели серверного комплекса при распределении нагрузки с помощью сервера-балансира, которая приведена на рисунке 3.1.

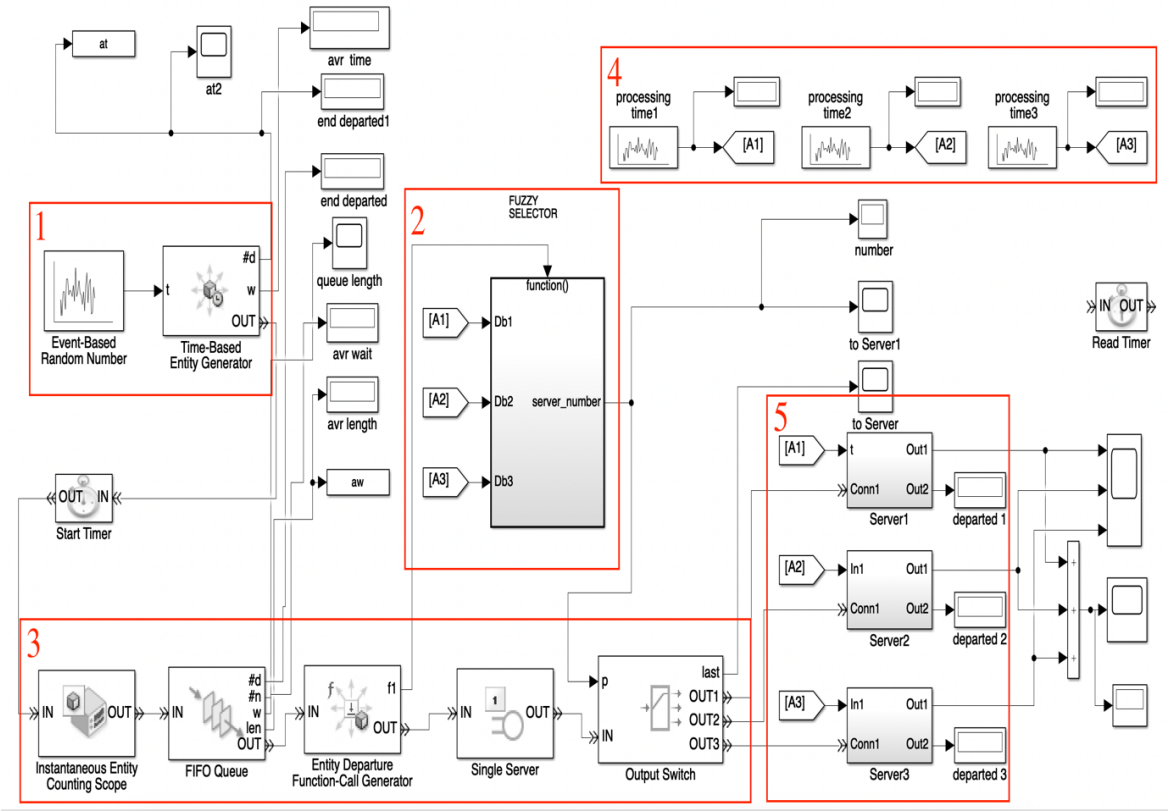


Рисунок 3.1 – Схема модели серверного комплекса

Используемая среда Simulink содержит средства визуального моделирования, обладающие гибкостью, необходимой для построения сложных моделей, и удобством применения, так как позволяет строить модель из библиотечных блоков.

Модель серверного комплекса состоит из пяти подсистем и блока переключателя серверов Server switcher, которые выполняют основные (генерация запросов, создание очереди, выбор сервера, генерация исходных данных, обработка запросов сервером) функции разрабатываемой системы.

Подсистема 1 генерирует входные параметры серверов и состоит из блоков Processing time 1-3, которые являются генераторами случайных чисел с равномерным законом распределения. Эти числа записываются в рабочую область Matlab и используются в качестве входных параметров для нечеткого селектора. Кроме того, они используются как время задержки при обработке запроса сервером в подсистеме 5. На основании этого значения подсистема 6, организует задержку при обработке запроса (моделирование задержки прибывания запросов в очереди).

Подсистема 2 предназначен для генерации запросов. Элемент Event-Based Random Number генерирует случайное число, которое используется в качестве временного интервала между двумя последовательными событиями, сгенерированными блоком Time-Based Entity Generator.

Подсистема 3 выполняет формирование очереди запросов. Вызов функции выбора сервера с помощью нечеткого селектора (подсистема 4) и переключение между серверами, обрабатывающими запросы (подсистема 6). Блок Instantaneous Entity Counting Scope – выводит количество запросов, поступивших за временной интервал. Освободившийся сервер приступает к обслуживанию того запроса, который ранее других поступил в систему. Элемент Entity Departure Function-Call Generator преобразует событие отправки запроса в вызов функции выбора сервера с помощью нечеткого селектора (подсистема 4). Блок Single Server отправляет запрос из очереди на обработку переключателем серверов (Server switcher), который распределяет ее со входа IN, на сервер *i*, в зависимости от результата балансировки, поступившего на порт *p* (подсистема 5).

В подсистеме 3 выполняется процедура выбора сервера на основе параметров

состояния серверов на момент времени поступления запросов с помощью нечетких правил. Для сравнения с круговым распределением (подсистема 4) удаляется, а элемент Server switcher (подсистема 5) настраивается таким образом, чтобы выбирать сервера по кругу.

Подсистема 6 состоит из трех серверов. Она принимает на обработку запрос пользователя распределенный на выбранный в Server switcher сервер. И обрабатывает ее с задержкой, определяемой в подсистеме 4.

Модель собрана из библиотечных блоков SimEvents и Simulink, для настройки свойств и параметров которых вызываются диалоговые окна. Данная модель является системой массового обслуживания с очередями.

Источником запросов (сущностей) в модели служит блок Time-Based Entity Generator. Закон распределения и среднее время между возникновением запросов заданы в параметрах блока генерации случайных чисел [18] Event-Based Random Number. Для учета времени обслуживания запросов в модель введены таймерные блоки. Для инициации вычисления временного интервала пребывания запросов в системе служит блок Start Timer. Сгенерированные запросы поступают в блок очереди FIFO Queue и далее должны распределяться по серверам для обслуживания.

Для реализации алгоритма балансировки нагрузки и выбора сервера (для загрузки файла пользователем) на основе нечеткого логического [37] вывода в модели предназначена подсистема FUZZY SELECTOR. С целью синхронизации этой подсистемы (ее тип Function-Call Subsystem) с событийно-управляемой имитационной моделью использованы блоки Entity Departure Function-Call Generator (генерирует управляющий сигнал в Function-Call Subsystem при каждом убытии запросов из блока) и Single Server (вспомогательный блок, применяется совместно с блоком Entity Departure Function-Call Generator). Затем запросы поступают в блок автоматического переключателя Output Switch, на управляющий вход которого поступает (от подсистемы FUZZY SELECTOR) сигнал о номере выбранного сервера. Следует отметить, что в случае моделирования режима кругового распределения нагрузки Round robin, возможность реализации которого предусмотрена в самом блоке Output Switch, этот входной сигнал не используется. Далее от блока

автоматического переключателя Output Switch запросы перемещаются к одному из трех серверов, которые представлены подсистемами Server1… Server3.

Требуется распределить данные по серверам, таким образом, чтобы по возможности уменьшить время, затраченное на обработку данных. Для создания системы распределения данных, учитывающей состояние серверного комплекса, необходимо установить показатели производительности сети, от которых зависит время обработки данных.

Рассмотрим в качестве входной информации для принятия решения о выборе сервера такие его параметры, как удаленность от пользователя и доступные ресурсы. В качестве входных данных системы использованы параметры . Это входные сигналы для принятия решения о выборе сервера S_i (i=1,2,3). Введем обозначения для выходного параметра: это R – решение о выборе сервера для загрузки файла. Этот параметр может иметь значения: P – позитивное; $Z;$ – нейтральное; N – негативное.

Введем следующие обозначения: D_s – расстояние от пользователя, загружающего файл, до сервера S_1, относит. ед.; U_{cpu} – загруженность центрального процессора, %; U_{ram} – использование оперативной памяти, %; B_{ch} – доступная пропускная способность канала передачи данных, Кбит/с; C_{sdr} – стоимость затрат на хранение, доставку и репликацию данных, ден. ед.

Это входные сигналы для принятия решения о выборе. Обозначим выходной параметр R_1 – решение о выборе сервера S_1 для решения вычислительной задачи или распределения статических данных, который может иметь значения: P_1 – позитивное; $Z_1;$ – нейтральное; N_1 – негативное.

Входные сигналы переводятся в значения нечетких переменных в подсистемах фаззификации (от англ. *fuzzy* – нечеткий) Input MF. На рисунке 3.2 показан скрипт для вывода графиков функций принадлежности для задачи распределения вычислительной нагрузки, на рисунке 3.3 представлены функции принадлежности входных и выходного параметров.

Диапазоны изменения входных переменных представляются термами значений: S – малое, M–среднее, L –большое.

```
    % Графики функций принадлежности
    figure
    x1 = 0:0.01:0.4;
    y1 = zmf(x1,[0.25 0.34]);
    plot(x1,y1,'Color', [0 0.4470 0.7410],'LineWidth',2);
    %
    x2 = 0.25:0.01:0.8;
    sig1=0.087;
    c1=0.523;
    y2=gaussmf(x2,[sig1 c1]);
    plot(x2,y2,'Color',[0 0.4470 0.7410],'LineWidth',2);
    %
    x3 = 0.6:0.01:1;
    ka1_=0.68;
    kc1_=0.8;
    y3=smf(x3,[ka1_ kc1_]);

    subplot(4,1,1);
    hLines=plot(x1,y1,x2,y2,x3,y3);
    set(hLines(1),'LineWidth',2);
    set(hLines(2),'LineWidth',2);
    set(hLines(3),'LineWidth',2);grid on
       axis([0 1 0 1.4])
      set(gca,'LineWidth',2)
     xlabel('Расстояние до сервера D_s'),ylabel('\mu(D_s)')
    text(0.1,1.18,'Малое'), ...
    text(0.5,1.18,'Среднее'), text(0.85,1.18,'Большое'),...

    x1 = 0.0:0.01:1;
    sig1=0.015;
    c1=0.54;
    y1 = zmf(x1,[0.5 0.55]);

    x2 = 0.5:0.01:0.6;
    sig2=0.015;
    c2=0.55;
    y2=gaussmf(x2,[sig2 c2]);

    x3 = 0:0.01:1;
    ka1_=0.55;
    kc1_=0.6;
    y3=smf(x3,[ka1_ kc1_]);

    subplot(4,1,2);
    hLines=plot(x1,y1,x2,y2,x3,y3);
    set(hLines(1),'LineWidth',2);
    set(hLines(2),'LineWidth',2);
    set(hLines(3),'LineWidth',2);grid on
    axis([0 1 0 1.4])
    set(gca,'LineWidth',2)
    xlabel('Загруженность          центрального          процессора
U_c_p_u'),ylabel('\mu(U_c_p_u)')
    text(0.1,1.18,'Малая'), ...
```

```
    text(0.5,1.18,'Средняя'), text(0.85,1.18,'Большая'),...

    x1 = 0:0.01:1;
    y1 = zmf(x1,[0.45 0.57]);
    %
    x2 = 0:0.01:1;
    sig1=0.095;
    c1=0.623;
    y2=gaussmf(x2,[sig1 c1]);
    %
    x3 = 0:0.01:1;
    ka1_=0.68;
    kc1_=0.8;
    y3=smf(x3,[ka1_ kc1_]);

    subplot(4,1,3);
    hLines=plot(x1,y1,x2,y2,x3,y3);
    set(hLines(1),'LineWidth',2);
    set(hLines(2),'LineWidth',2);
    set(hLines(3),'LineWidth',2);grid on
    axis([0 1 0 1.4])
    set(gca,'LineWidth',2)
    xlabel('Загруженность оперативной памяти
U_r_a_m'),ylabel('\mu(U_r_a_m)')
    text(0.1,1.18,'Малая'), ...
    text(0.5,1.18,'Средняя'), text(0.85,1.18,'Большая'),...

    xm=[0 0 0 1 1 1 2 2 2];  y1=[ 0 1 0 0 1 0 0 1 0];%выход R1
    subplot(4,1,4),plot(xm,y1)
    hLines=plot(xm./1,y1);set(hLines(1),'LineWidth',2);
    axis([-0.1 2.1 0 1.4]),grid on
    set(gca,'LineWidth',2)
    text(-0.05,1.14,'Отрицательное'),
text(1.6,1.14,'Положительное'),...
    text(0.85,1.14,'Нейтральное');
    xlabel('Решение о пригодности сервера1 для выбора
R_1'),ylabel('\mu(R_1)')
    set(gca,'LineWidth',2)
```

Рисунок 3.2 – Программный код вывода графиков функций принадлежности для задачи распределения вычислительной нагрузки

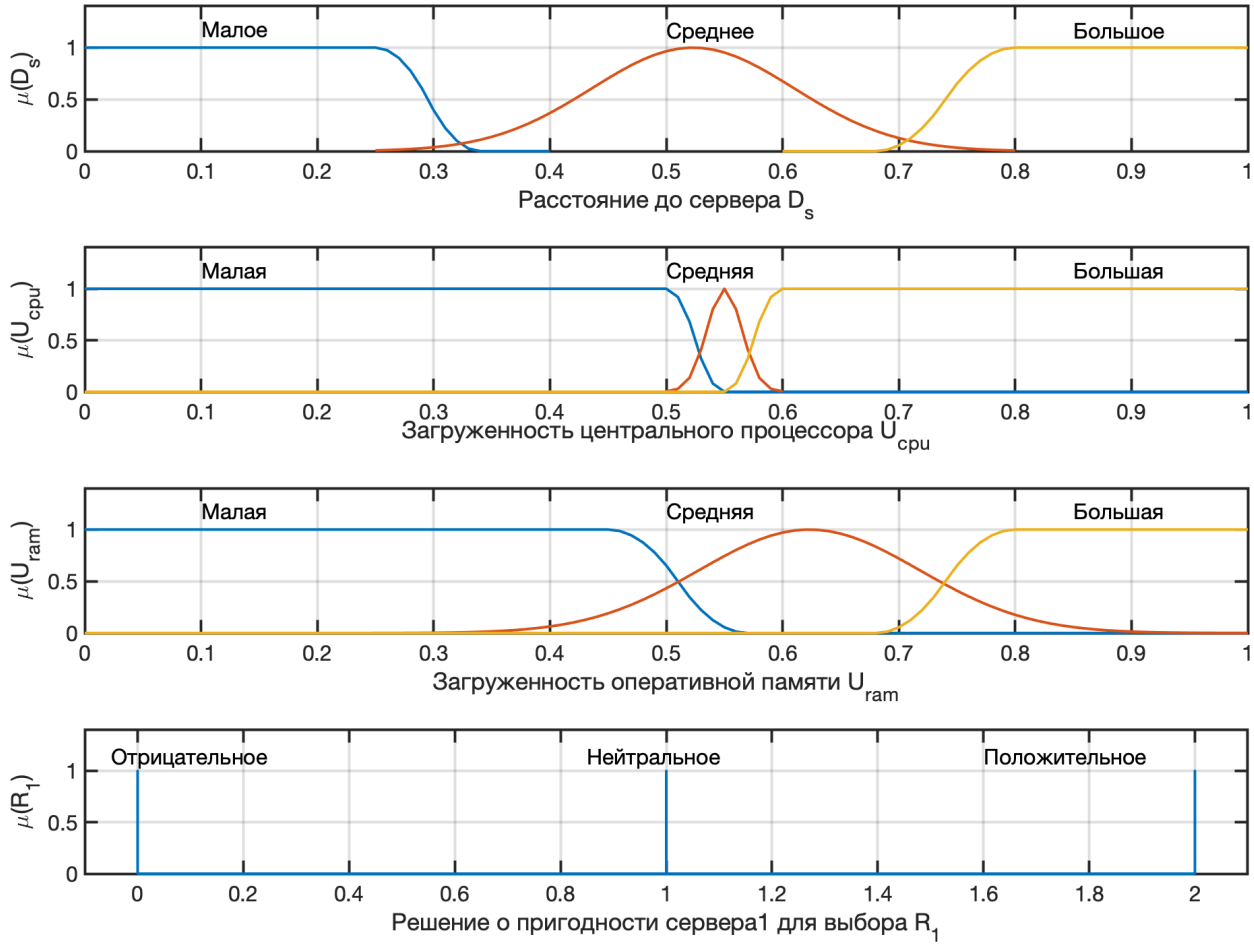


Рисунок 3.3 – Функции принадлежности входных и выходного параметров

Аналогичным образом строятся функции входных и выходных параметров для задачи распределения статических данных.

В качестве функций принадлежности для внутренних лингвистических термов *M* (англ. *mean value* – среднее значение) входных переменных селектора принята гауссова функция принадлежности (*gaussmf*). Функция формируется в соответствии с выражением:

$$\mu(x) = e^{\frac{-(x-c)^2}{2\sigma^2}}, \tag{3.1}$$

Параметр σ задает коэффициент концентрации функции принадлежности, а параметр *c* отвечает за ширину кривой. Вызов функции имеет синтаксис *y=gaussmf*(*x*,[*sig c*]) и возвращает выходной аргумент *y*, содержащий степень принадлежности входного аргумента *x*.

Для задания крайних лингвистических термов *S* (англ. *small value*) используются

Z-образные функции принадлежности (*zmf*), зависящие от двух параметров. Такая функция принадлежности является невозрастающей, ее параметры определяют интервал, внутри которого функция нелинейно убывает от 0 до 1.

Вызов функции имеет синтаксис *y= zmf*(*x*,[*a b*]) и возвращает выходной аргумент *y*, содержащий степень принадлежности входного аргумента *x*. Параметры *a* и *b* определяют крайние точки наклонной части функции принадлежности. Функция формируется в соответствии с выражением

$$\mu(x)=\begin{cases}1, & x\le a;\\ 1-2\left(\dfrac{x-a}{b-a}\right), & a\le x\le \dfrac{a+b}{2};\\ 2\left(\dfrac{x-b}{b-a}\right), & \dfrac{a+b}{2}\le x\le b;\\ 0, & x\ge b\ .\end{cases} \tag{3.2}$$

Таким же образом для задания крайних лингвистических термов *L* (англ. *large value*) используются *S*-образные функции принадлежности (*smf*), зависящие от двух параметров.

Такая функция принадлежности является невозрастающей, ее параметры определяют интервал, внутри которого функция нелинейно убывает от 0 до 1.

Вызов функции имеет синтаксис *y= smf*(*x*,[*a b*]) и возвращает выходной аргумент *y*, содержащий степень принадлежности входного аргумента *x*. Параметры *a* и *b* определяют крайние точки наклонной части функции принадлежности.

Функция формируется в соответствии с выражением

$$\mu(x)=\begin{cases}0, & x\le a;\\ 2\left(\dfrac{x-a}{u-a}\right), & a\le x\le \dfrac{a+b}{2};\\ 1-2\left(\dfrac{x-a}{u-a}\right), & \dfrac{a+b}{2}\le x\le b;\\ 1, & x\ge b\ .\end{cases} \tag{3.3}$$

Такие функции принадлежности для термов входных переменных выбраны в

связи с тем, что имеют аналитическое представление в виде простых математических формул с малым количеством параметров, что упрощает числовые расчеты и сокращает временные затраты при моделировании.

Четкое число R_1, задающее заключение каждого правила, рассматривается как одноэлементное нечеткое множество с точечной, или синглтонной (от англ. single – единственный) функцией принадлежности.

Полученная база знаний эквивалентна базе знаний Сугено нулевого порядка, в которой посылки (входные параметры) заданы нечеткими множествами, а заключения правил (решения о пригодности выбора сервера) – четкими числами.

Операция дефаззификации (обратного преобразования нечетких переменных в четкие) осуществляет четкий вывод нахождением взвешенного среднего для получения решения о пригодности первого сервера для выбора

$$R_1 = \frac{\sum\limits_{i=1,m} \mu(R_{1i}) R_{1i}}{\sum\limits_{i=1,m} \mu(R_{1i})}, \qquad (3.4)$$

где R_1 – четкое значение выходной переменной; R_{1i} – значение выходной переменной для i-го терма с единичным значением степени принадлежности; $\mu(R_1)$ – степень принадлежности к этому терму; m – число термов.

Таким образом, совокупность нечетких правил и переменных используется для нечеткого логического вывода, результатом которого является значение степени пригодности сервера для выбора.

На рисунках 3.4, 3.5 приведены модели выбора сервера для обеих задач. На рисунках 3.4а, 3.5а приведен вид системы нечеткого выбора сервера. Схема одной из подсистем Input1 MF представлена на рисунке 3.4б. На рисунке 3.4в представлена схема одного из правил нечеткого вывода для задачи РВН (П0).

Четкое число R_1, задающее заключение каждого правила, принимает значения N_1=0; Z_1=1; P_1=2. Далее нечеткие переменные используются при нечетком логическом выводе: над ними производятся операции, соответствующие продукционным правилам.

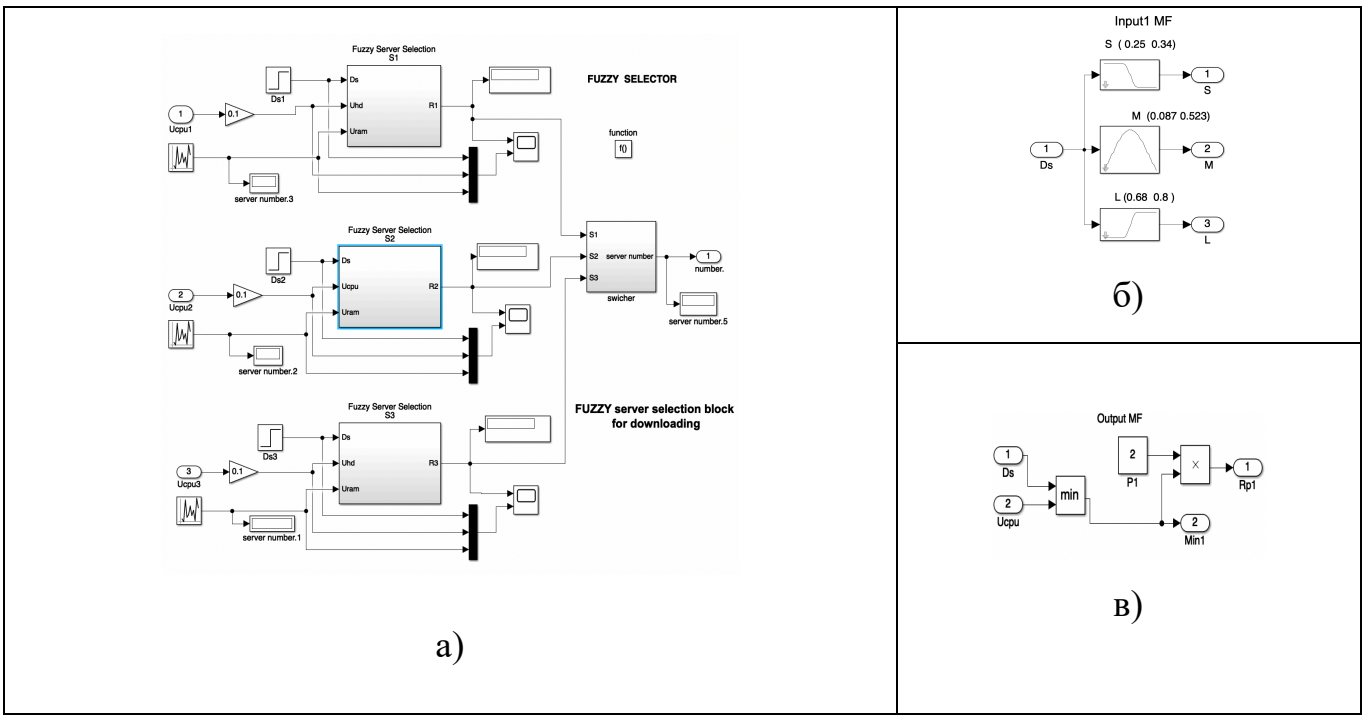


Рисунок 3.4 - Модель выбора сервера для задачи РВН: а) общий вид системы нечеткого выбора сервера; б) подсистема фаззификации Input1 MF; в) правило нечеткого вывода (П0).

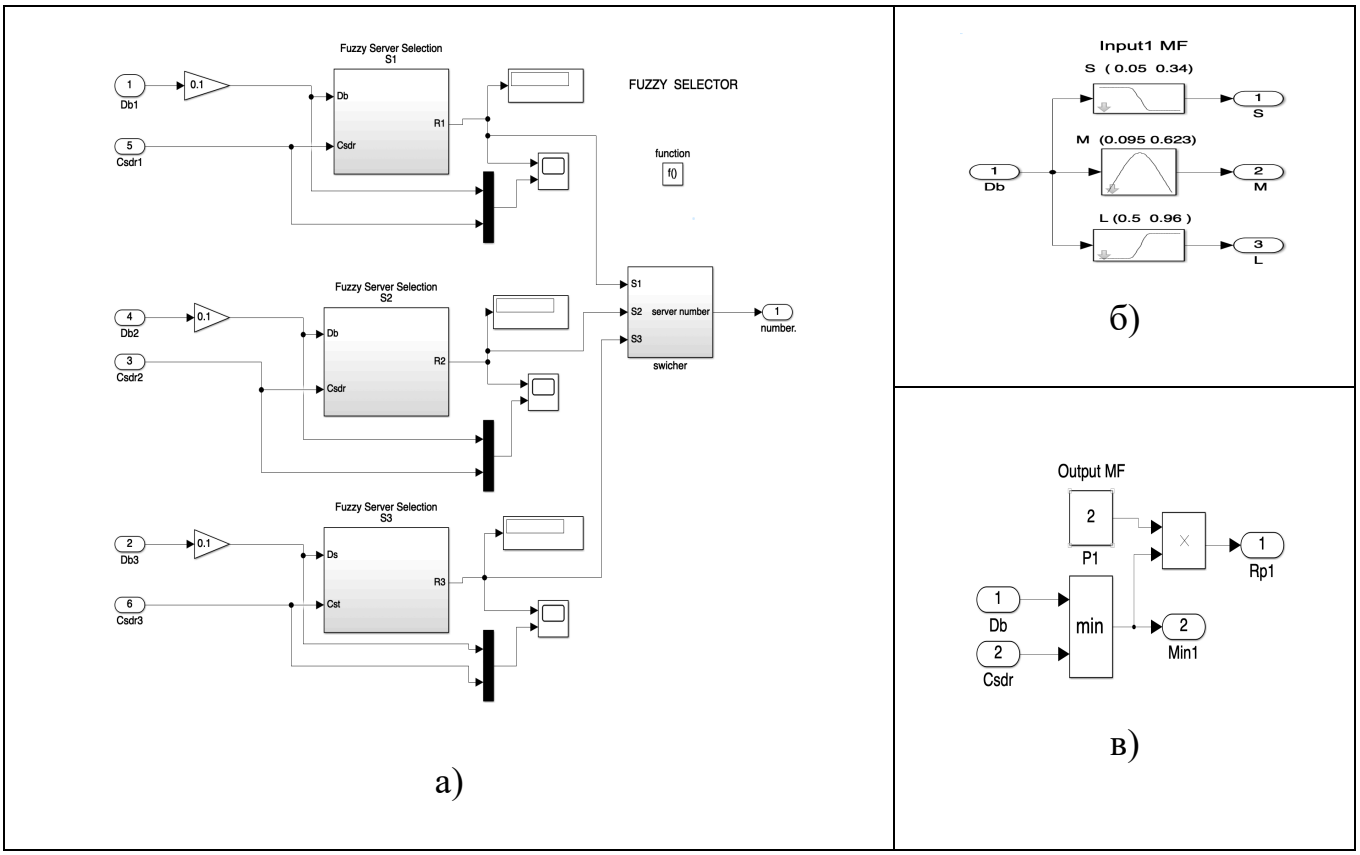


Рисунок 3.5 - Модель выбора сервера для задачи РСД: а) общий вид системы нечеткого выбора сервера; б) подсистема фаззификации Input1 MF; в) правило нечеткого вывода (П0).

Нечеткие правила были получены на основе построения дерева решений [100] по данным таблиц 2.1, 2.2. База нечетких продукционных правил для выбора сервера S_1 (такая же, как и для серверов S_2 и S_3), которая реализуется в подсистеме Fuzzy Server Selection S_1, представлена в таблицах 3.1, 3.2.

Таблица 3.1 – База нечетких правил выбора сервера для задачи РСД

Показатель D_b	Показатель C_{sdr}		
	S	*M*	*L*
S	*P*	*P*	*N*
M	*P*	*Z*	*N*
L	*Z*	*N*	*N*

Таблица 3.2 – База нечетких правил выбора сервера для задачи РВН

Показатель U_{cpu}	Показатель U_{ram}		
	S	*M*	*L*
При показателе $D_s = S$			
S	*P*	*P*	*P*
M	*P*	*P*	*P*
L	*Z*	*N*	*N*
При показателе $D_s = M$			
S	*P*	*P*	*P*
M	*P*	*Z*	*N*
L	*N*	*N*	*N*
При показателе $D_s = L$			
S	*N*	*N*	*N*
M	*N*	*N*	*N*
L	*N*	*N*	*N*

На рисунке 3.6, 3.7 приведены схемы подсистем нечеткого выбора сервера. Соответствия нечетких правил схемам реализации для подсистемы нечёткого выбора

сервера приведены в таблице соответствующим схемам правил. Каждому продукционному правилу соответствует схема, построенная средствами Simulink, выходами которой являются степень принадлежности выхода (принятого решения о выборе сервера R_1) и значение выхода R_1.

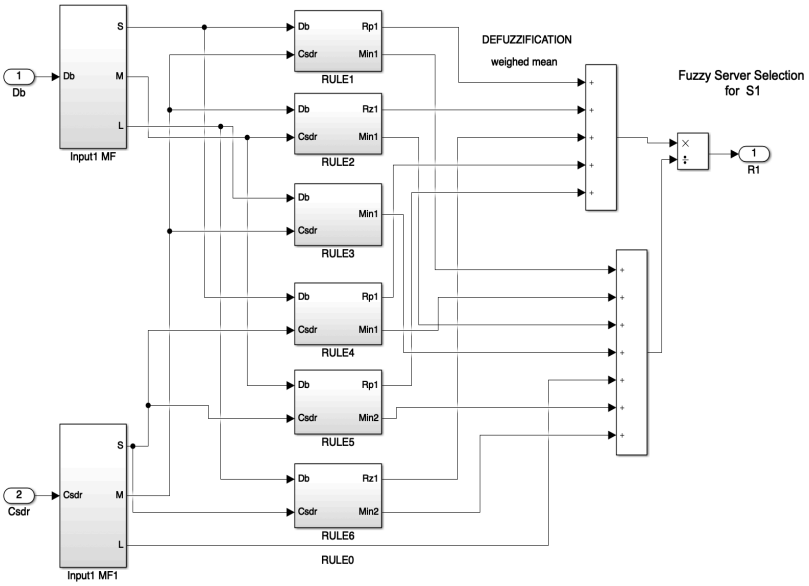


Рисунок 3.6 – Схема подсистемы нечеткого выбора сервера для задачи РСД

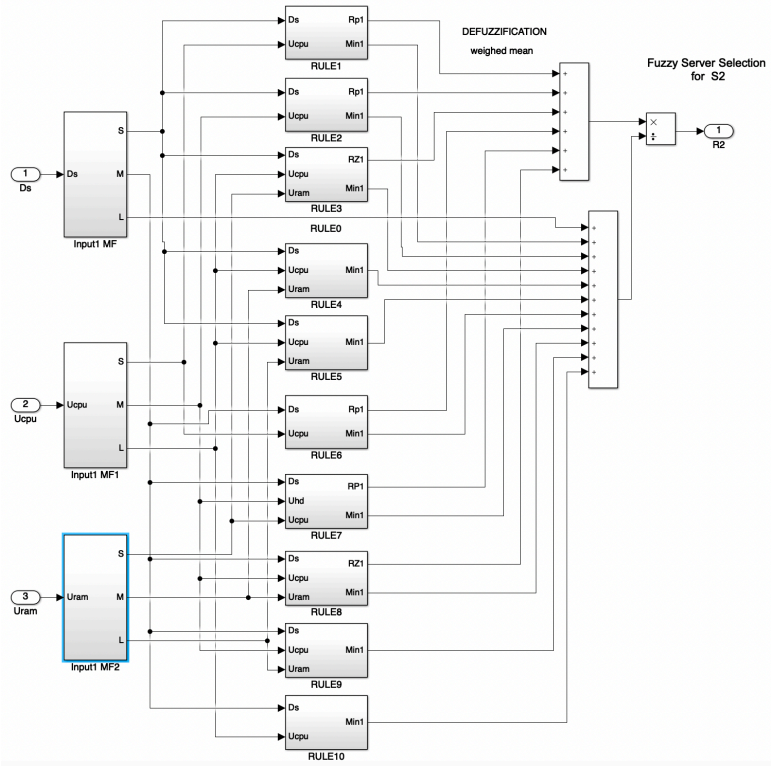
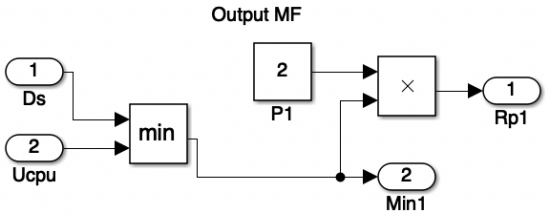
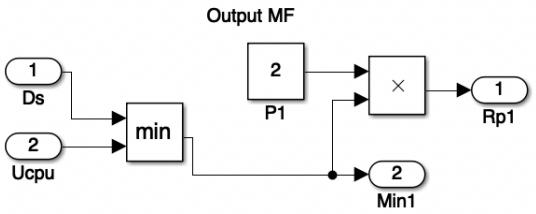
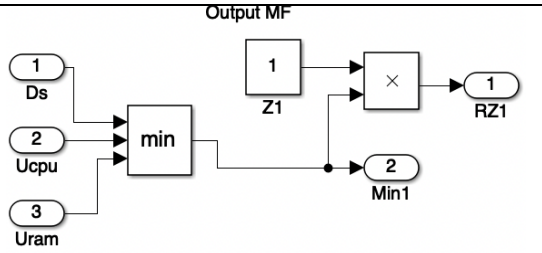
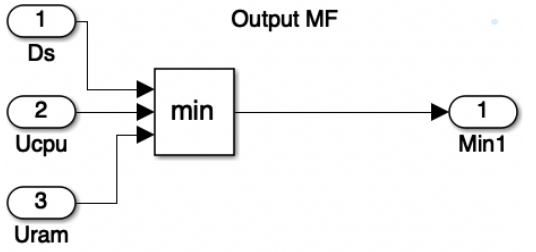
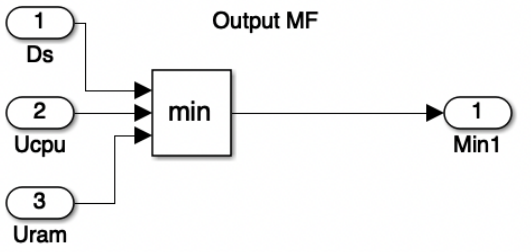
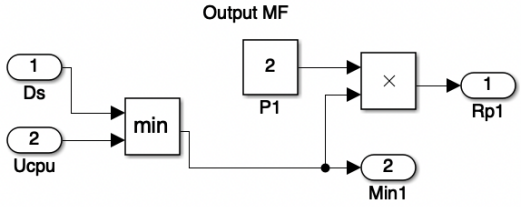
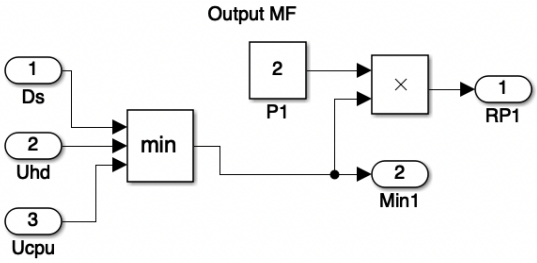
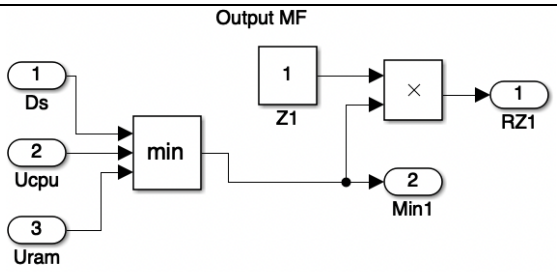
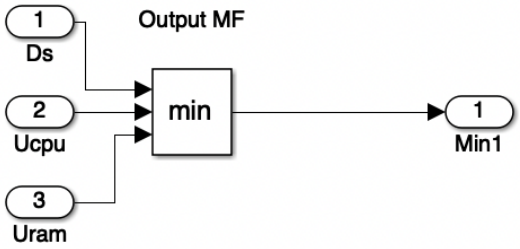
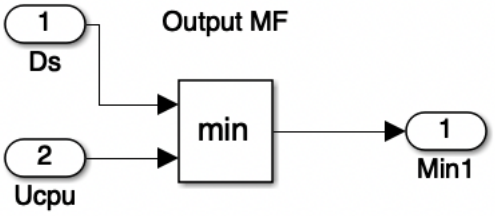


Рисунок 3.7 – Схема подсистемы нечеткого выбора сервера для задачи балансировки вычислительной нагрузки.

Реализация базы правил, выполненная в схемах на рисунках 3.6, 3.7, приведена в таблице 3.3.

Таблица 3.3 – Соответствие нечетких правил и схем реализации

	Нечеткое правило выбора сервера S_1	Схема реализации правила нечеткого вывода
1	П1: ***ЕСЛИ*** $D_s = S$ ***И*** $U_{cpu} = S$, ***ТО*** $R_1 = P_1$;	
2	П2: ***ЕСЛИ*** $D_s = S$ ***И*** $U_{cpu} = M$, ***ТО*** $R_1 = P_1$;	
3	П3: ***ЕСЛИ*** $D_s = S$ ***И*** $U_{cpu} = L$ ***И*** $U_{ram} = S$, ***ТО*** $R_1 = Z_1$;	
4	П4: ***ЕСЛИ*** $D_s = S$ ***И*** $U_{cpu} = L$ ***И*** $U_{ram} = M$, ***ТО*** $R_1 = N_1$;	
5	П5: ***ЕСЛИ*** $D_s = S$ ***И*** $U_{cpu} = L$ ***И*** $U_{ram} = L$, ***ТО*** $R_1 = N_1$;	

	Нечеткое правило выбора сервера S_1	Схема реализации правила нечеткого вывода
6	П6: ***ЕСЛИ*** $D_s = M$ ***И*** $U = S$, ***ТО*** $R_1 = P_1$;	
7	П7: ***ЕСЛИ*** $D_s = M$ ***И*** $U_{cpu} = M$ ***И*** $U_{ram} = S$, ***ТО*** $R_1 = P_1$;	
8	П8: ***ЕСЛИ*** $D_s = M$ ***И*** $U_{cpu} = M$ ***И*** $U_{ram} = M$, ***ТО*** $R_1 = Z_1$;	
9	П9: ***ЕСЛИ*** $D_s = M$ ***И*** $U_{cpu} = M$ ***И*** $U_{ram} = L$, ***ТО*** $R_1 = N_1$;	
10	П10: ***ЕСЛИ*** $D_s = M$ ***И*** $U_{cpu} = L$, ***ТО*** $R_1 = N_1$;	

На основе обработки входных параметров для каждого сервера S_1, S_2 и S_3 получены результаты в виде степени принадлежности к множеству положительных

решений. Далее принятие окончательного решения о выборе номера сервера осуществляется подсистемой Switch. Выбирается тот сервер, значение функции, принадлежности которого является максимальным.

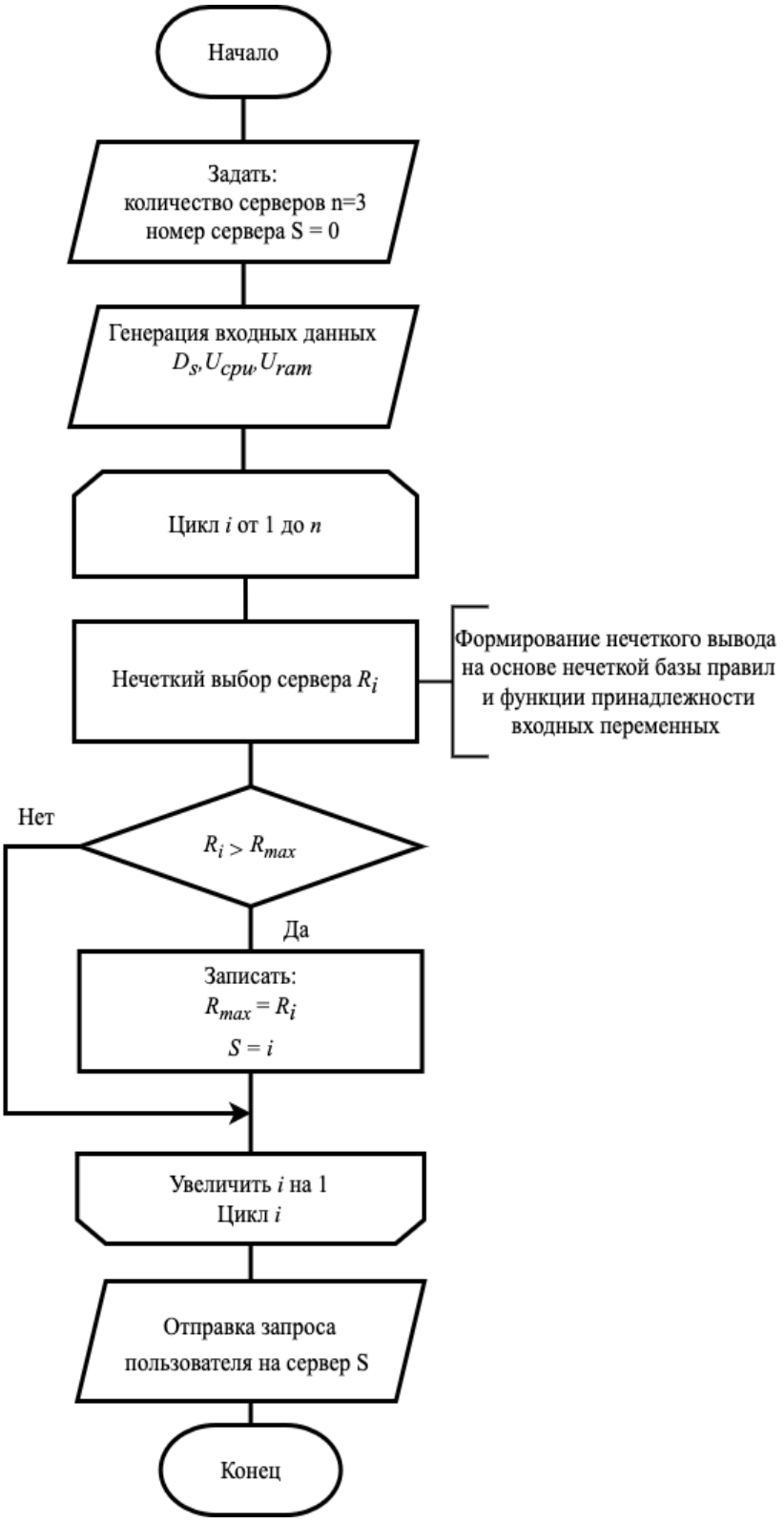


Рисунок 3.8 – Алгоритм распределения нагрузки на основе нечеткой логики

Таким образом, получено обоснованное решение о выборе сервера для загрузки данных, учитывающее параметры и возможности сервера. На рисунке 3.8 приведен алгоритм распределения нагрузки с использование нечеткой логики. Для каждого из серверов используется нечеткий классификатор, приведенный на рисунках 3.6, 3.7. Выбор сервера окончательно осуществляется после сравнения оценок функций принадлежности для всех серверов.

3.2 Результаты имитационных исследований балансировки нагрузки по серверам на основе нечеткого логического вывода

С целью получения исходных данных, необходимых для построения имитационной модели серверного комплекса, был проведен натурный эксперимент по распределению запросов на загрузку файлов на реальные сервера. Исследовались случайные процессы поступления запросов в системе, определялись такие случайные величины, как интервалы между моментами поступления запросов пользователей в сети, и оценивались показатели состояния серверов. Получено, что поток запросов на загрузку файлов в систему имеет экспоненциальный закон распределения, при котором плотность вероятности длительности пауз между генерациями запросов $p(t) = \lambda e^{-\lambda t}$, где λ – интенсивность потока (средняя длительность паузы между генерациями запросов $t_{ig} = \lambda^{-1}$). Определено, что длительность паузы между поступлением запросов t_{ig} изменялась в диапазоне от 1 с до 4 с. Принятые для моделирования исходные данные о диапазонах изменения показателей состояния серверов приведены в таблице 3.4. Для задачи РВН в качестве основного параметра, влияющего на время обработки запросов сервером, принималась загруженность оперативной памяти, которая изменялась в принятых диапазонах по равномерному закону. С использованием данных, полученных от реальных серверов, проведено имитационное моделирование системы распределения нагрузки на основе нечеткого логического вывода и (для сравнения) балансировки с помощью метода кругового распределения нагрузки Round Robin. В результате моделирования получены такие характеристики системы, как средняя длина очереди в серверном комплексе и количество обработанных запросов (рисунок 3.9).

Таблица 3.4 – Исходные данные для проведения моделирования, задача РВН

Сервер	Показатели состояния сервера				
	Загруженность центрального процессора U_{cpu}		Загруженность оперативной памяти U_{ram}		Расстояние от пользователя до сервера D_s
	U_{cpu} min	U_{cpu} max	U_{ram} min	U_{ram} max	D_s
S_1	0,53	0,58	0,43	0,67	0,3
S_2	0,54	0,57	0,32	0,40	0,6
S_3	0,51	0,60	0,72	0,93	0,8

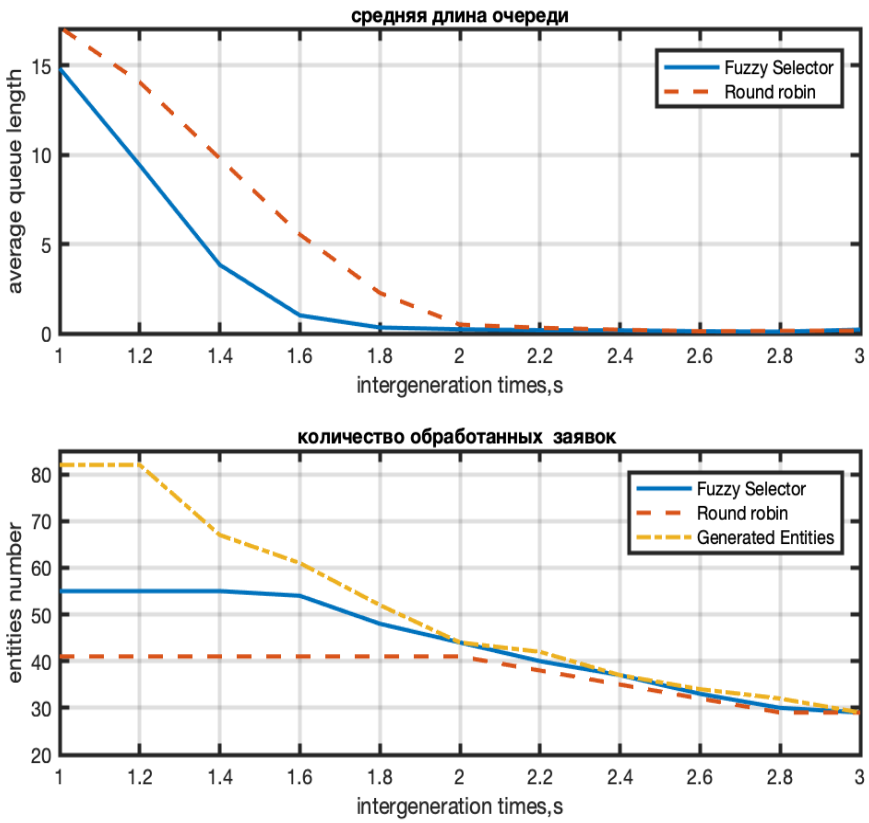


Рисунок 3.9 – Средняя длина очереди и количество обработанных запросов в зависимости от интервала их поступления

Расчеты проводились для различных значений интервалов поступления запросов в систему, и оценивались характеристики работы системы балансировки при круговом распределении и с помощью нечеткого селектора. Видно, что в обоих

случаях средняя длина очереди уменьшается при увеличении длительности пауз между генерациями запросов до тех пор, пока эти длительности не будут больше времени обработки задач серверами. При этих же условиях количество обработанных запросов становится равным количеству сгенерированных запросов. Получено, что среднее значение длины очереди запросов при использовании нечеткого селектора, определяющего номер сервера для загрузки данных на основе информации о текущем состоянии серверов комплекса, существенно меньше, чем в случае, когда запрос отправляется на сервера просто по очереди (по кругу). Также и количество обработанных запросов (при высокой частоте их поступления) больше, если учитывать данные о показателях работы серверов, то есть производительность серверного комплекса увеличивается.

Динамику процесса можно проследить по графикам на рисунке 3.10, соответствующим временным диаграммам работы серверов для случая высокой интенсивности поступления запросов (t_{ig}=1.2 с), превышающей интенсивности потоков обслуживание запросов серверами.

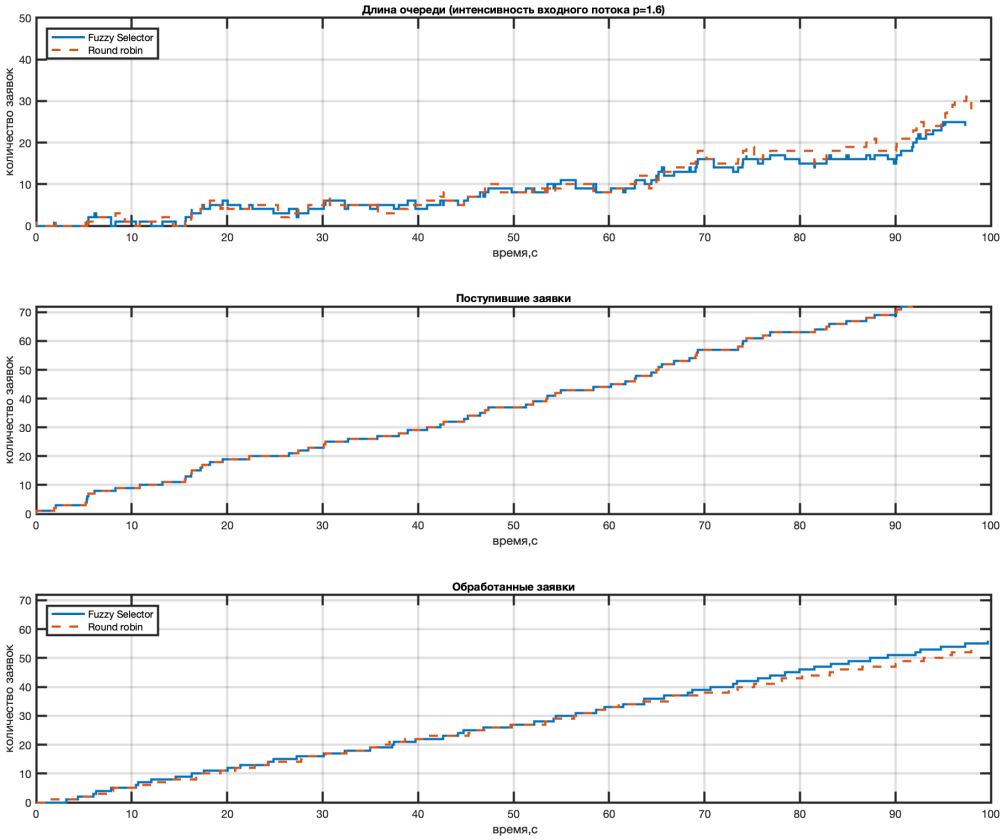


Рисунок 3.10 – Длина очереди и количество поступивших и обработанных запросов (при интенсивности входного потока со средним t_{ig} =1. с)

На протяжении 100 секунд длина очереди возрастает и достигает 30 запросов, когда распределение нагрузки осуществляется с помощью кругового метода Round robin. При использовании нечеткого селектора длина очереди меньше – 24 запросов. Общее количество поступивших запросов – 82, обработана системой с нечетким распределением нагрузки 56 запрос. В случае распределения нагрузки с помощью кругового метода количество обработанных запросов меньше – 53.

На рисунках 3.11, 3.12 представлены результаты распределения запросов по серверам. В случае использования метода кругового распределения нагрузки Round Robin запросы распределены по серверам примерно одинаково. Серверами S_1 и S_2 обработано 18 запросов, а сервером S_3 – 19 запросов. Всего обработано серверным комплексом 55 запросов из 82 поступивших.

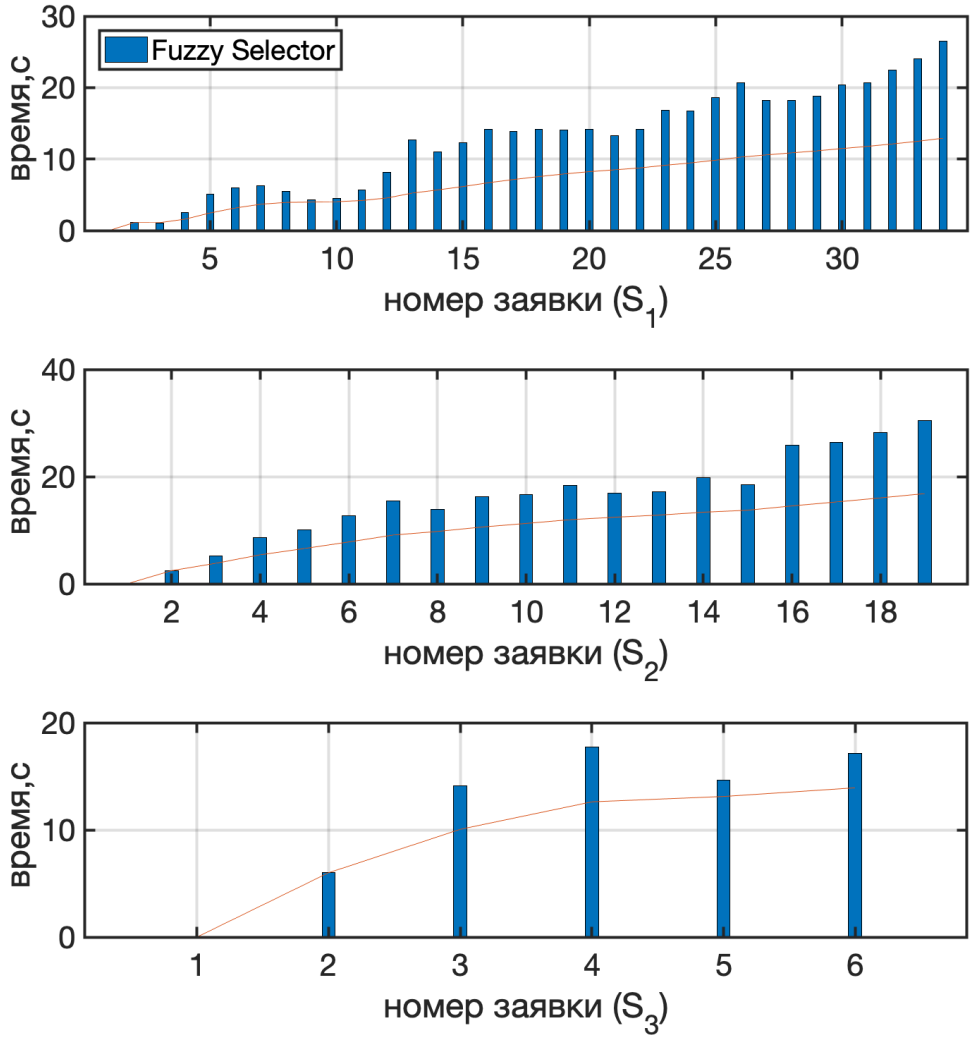


Рисунок 3.11 – Длительности пребывания запросов в системе с нечетким классификатором (Fuzzy Selector)

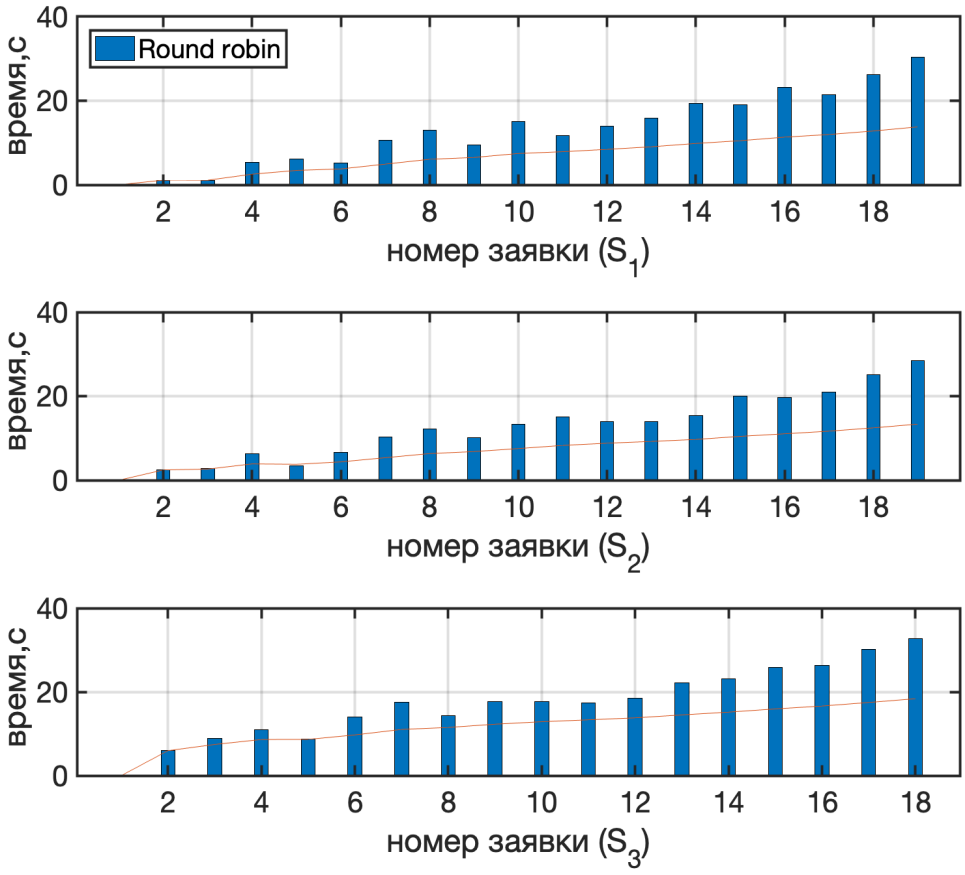


Рисунок 3.12 – Длительности пребывания запросов в системе с круговым распределением (Round Robin)

Система с нечетким распределением нагрузки на основе данных о состоянии серверов работает лучше. Всего обработана серверным комплексом 56 запросов, причем основное количество запросов было распределено на сервер S_1, имеющий наибольшую скорость обработки данных (33 запроса), на менее производительный сервер S_2 направлено 18 запросов, и 6 запросов обработано третьим сервером S_3. Длительности пребывания запросов в системе в случае учета показателей работы серверов существенно уменьшились (не более 18 с, а при круговом распределении превышают 24 с). Среднее время пребывания запросов в системе сократилось на 20 - 60% для среднего времени между поступлением запросов t_{ig} от 1 с до 1.8 с. Таким образом, при распределении нагрузки с помощью нечеткого селектора существенно улучшены временные характеристики обслуживания запросов серверным комплексом.

Для задачи РСД в качестве основного параметра, влияющего на время обработки запросов сервером, принимался мультипликативный параметр D_b, который

изменялся в принятых диапазонах по равномерному закону. С использованием данных, полученных от реальных серверов, проведено имитационное моделирование системы распределения нагрузки на основе нечеткого логического вывода и (для сравнения) балансировки с помощью метода кругового распределения нагрузки Round Robin. Исходные данные для моделирования приведены в таблице 3.5

Таблица 3.5 – Исходные данные для проведения моделирования, задача РСД

Сервер	Показатели состояния сервера			
	Стоимость затрат на обработку данных, C_{sdr}		Мультипликативный критерий, D_b	
	C_{sdr}^{min}	C_{sdr}^{max}	D_b^{min}	D_b^{max}
Server 1	0,10	0,70	0,1	0,3
Server 2	0,42	0,56	0,1	0,4
Server 3	0,30	0,60	0,5	0,9

Получены стоимостные (рисунок 3.13) и временные (рисунок 3.14) характеристики работы системы для каждого из методов. За одну итерацию моделирования методы обрабатывают разное количество запросов, поэтому рассчитаны средние значения временных и стоимостных характеристик, полученных для различной длительности паузы между поступлениями запросов. Система с нечетким методом распределения показывает лучшие результаты по сравнению с круговым распределением нагрузки.

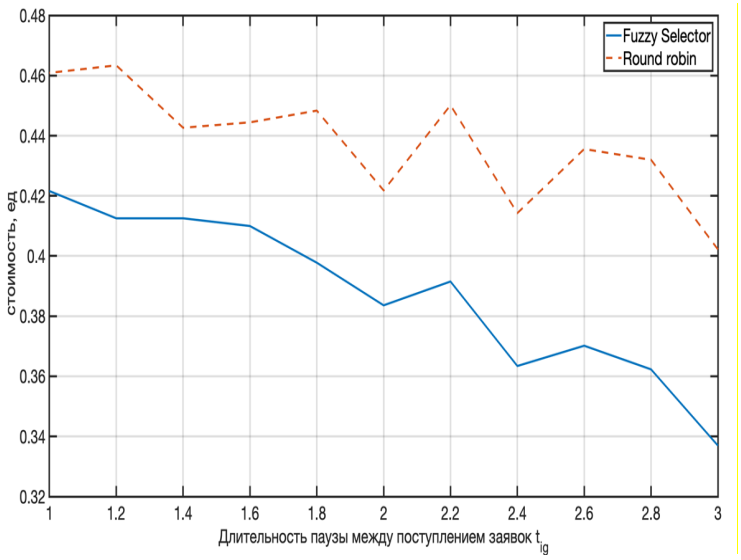


Рисунок 3.13 – Средняя стоимость обработки запросов серверами

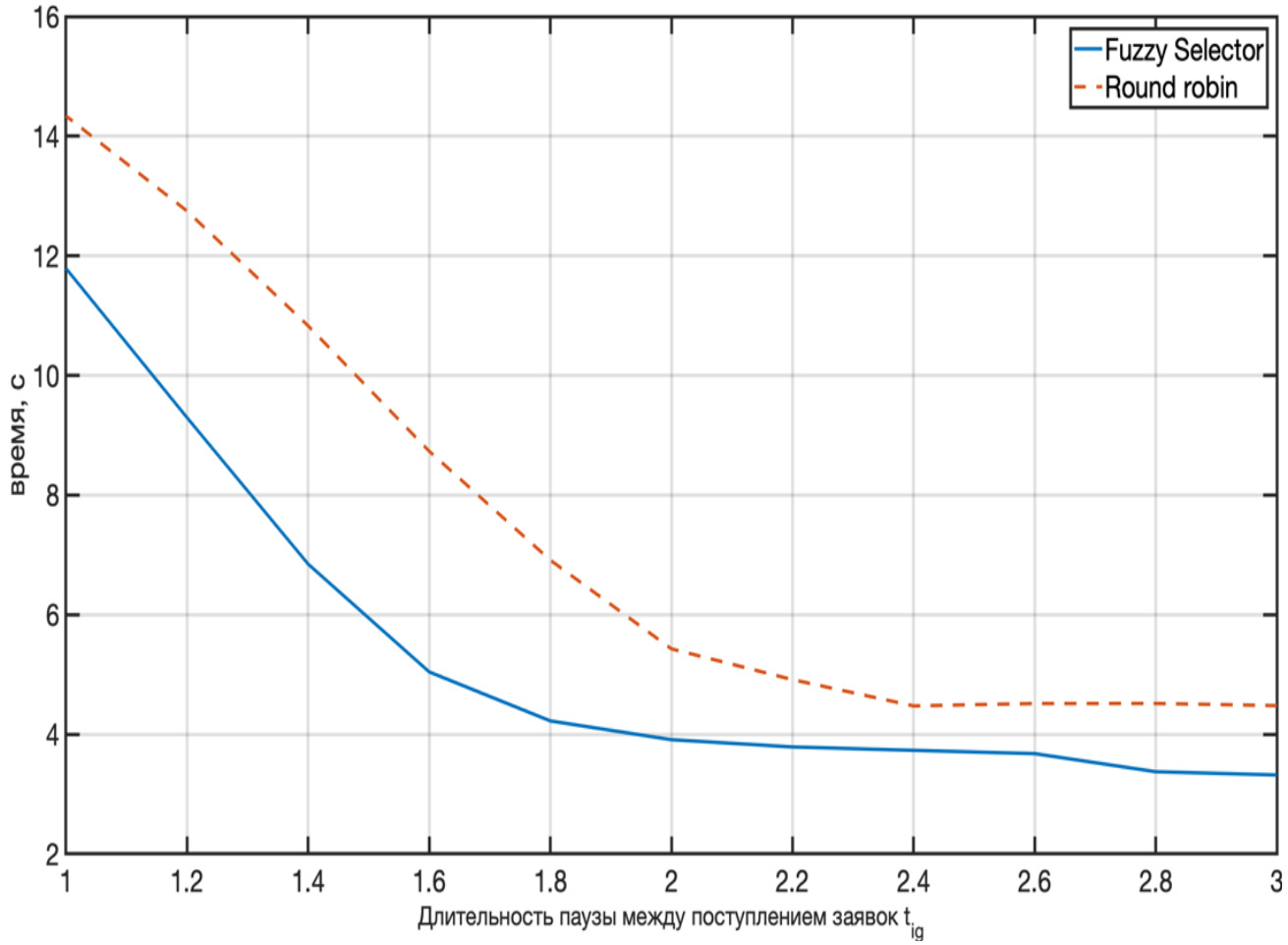


Рисунок 3.14 – Среднее время обработки запросов серверами

Средняя стоимость обработки запросов уменьшается на 10 - 17% по сравнению с круговым методом распределения (рисунок 3.13). Одновременно с уменьшением стоимости обработки запросов снизилось и время, затраченное на обработку запросов. Получено, что среднее время обработки запросов уменьшилось на величину от 5 до 20% (рисунок 3.14). Отметим, что при длительности паузы между поступлением запросов t_{ig} от 1 с до 1,6 с. Изменения стоимости перестает происходить, так как система не успевает обработать большего количества запросов. Рассмотрим распределение запросов по серверам для t_{ig} =1,2, рисунок 3.15. В случае использования метода кругового распределения нагрузки Round Robin запросы распределены по серверам равномерно, но при этом одинаково загружаются как высоко нагруженные серверы с высокой стоимостью, так и серверы с низкой нагрузкой и низкой стоимостью. Тогда как нечеткий метод распределяет больше запросов на менее загруженные и более близкие к пользователю сервера с низкой и

средней стоимостью, обеспечивая более высокую скорость обработки запросов пользователя.

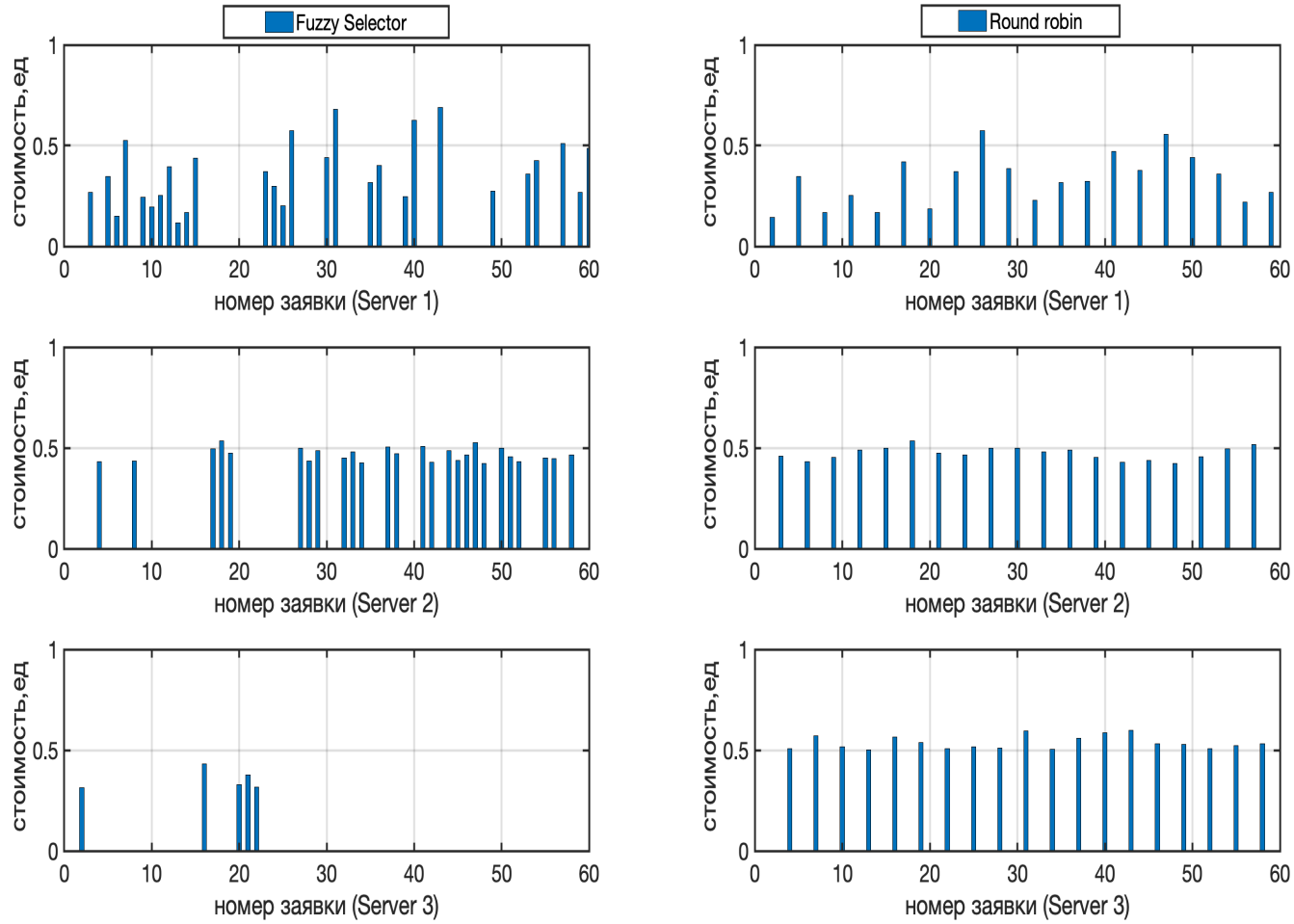


Рисунок 3.15 – Распределение запросов по серверам с помощью:
нечеткого селектора (слева), кругового циклу (справа)

Таким образом, распределение данных с помощью нечеткого селектора улучшает временные и стоимостные характеристики обслуживания запросов серверным комплексом, тем самым минимизируя целевую функцию (2.8).

Выполненные численные эксперименты по распределению данных по серверам показали преимущество подхода к выбору сервера с учетом состояния серверного комплекса на основе нечеткого логического вывода, которое выражается в уменьшении стоимости обработки запросов и сокращению времени обслуживания пользователей.

Отметим, что разработанная имитационная модель позволяет исследовать работу сервера-балансира при распределении нагрузки между серверами, учитывая

случайный характер моментов поступления запросов на обработку, а также переменную длительность задержек при поступлении задач. Предлагаемая модель представляет возможность испытывать программное обеспечение сервера-балансира для сервера в условиях, приближенных к реальным.

3.3 Выводы по главе

1. На основании проведенного исследования сделан вывод, что выбранные программные средства MATLAB/Simulink/SimEvents пригодны для моделирования работы серверного комплекса облачного ресурса как системы с дискретными состояниями на основе теории систем массового обслуживания.

2. Разработанная имитационная модель позволяет исследовать работу сервера-балансира при распределении нагрузки между серверами, учитывая случайный характер моментов поступления запросов на обработку, а также переменную длительность задержек при поступлении задач. То есть такая модель представляет возможность испытывать программное обеспечение сервера-балансира для серверов в условиях, приближенных к реальным.

3. В результате выполнения имитационных экспериментов установлено, что разработанный метод распределения нагрузки на основе нечеткого классификатора (Fuzzy Selector) увеличивает количество обработанных запросов на 20-60% по сравнению с круговым распределением (Round Robin). Средняя стоимость обработки запросов уменьшается на 10-17% по сравнению с круговым методом распределения. Получено, что среднее время обработки запросов уменьшилось на величину от 5 до 20%.

4 РАЗРАБОТКА ПРОГРАММНОГО КОМПЛЕКСА И РЕЗУЛЬТАТЫ ЭКСПЕРИМЕНТАЛЬНЫХ ИССЛЕДОВАНИЙ РАСПРЕДЕЛЕНИЯ НАГРУЗКИ В ОБЛАЧНОМ СЕРВЕРНОМ КОМПЛЕКСЕ

В четвертой главе **представлен** алгоритм параллельных вычислений для распределения вычислительной нагрузки между ресурсами облачного кластера серверов и структура программного комплекса для проведения экспериментальных исследований. Приводятся результаты экспериментальных исследований распределения вычислительной нагрузки по ресурсам облачного кластера серверов.

4.1 Разработка структуры программного комплекса и алгоритма параллельных вычислений балансировки нагрузки

На основании представленной на рисунке 2.1 структуры системы, а также, основываясь на результатах экспериментальных исследований было разработано клиент-серверное приложение, одним из элементов которого является сервер балансир. Серверное приложение реализует обработку двух видов запросов: на загрузку статических данных (изображения, видео ряд, текстовые файлы) и на решение вычислительных задач (поиск в базе данных, проведение расчетов, обработка информации). В состав приложения входят: программа «Агент», программа «Балансир», программа «Веб-сервер», база данных (БД) пользователей и БД параметров состояния серверов. Программа «Агент» собирает с вычислительных серверов данные о их состоянии и сохраняет в БД серверов. Программа «Балансир», работая по расписанию (раз в минуту) проверяет изменения в данных о состоянии серверов, а программа «Веб-сервер» перенаправляет запросы пользователей на нужный сервер. Если нагрузка на сервера изменилась, пользователи перераспределяются в соответствии с текущей загруженностью серверов. Перераспределение пользователей осуществляется с использованием одного из алгоритмов: кластеризации «Clustering», нечеткого выбора «Fuzzy Selector» и на основе кругового цикла «Round Robin». На рисунке 4.1 представлена схема серверного приложения с балансиром нагрузки.

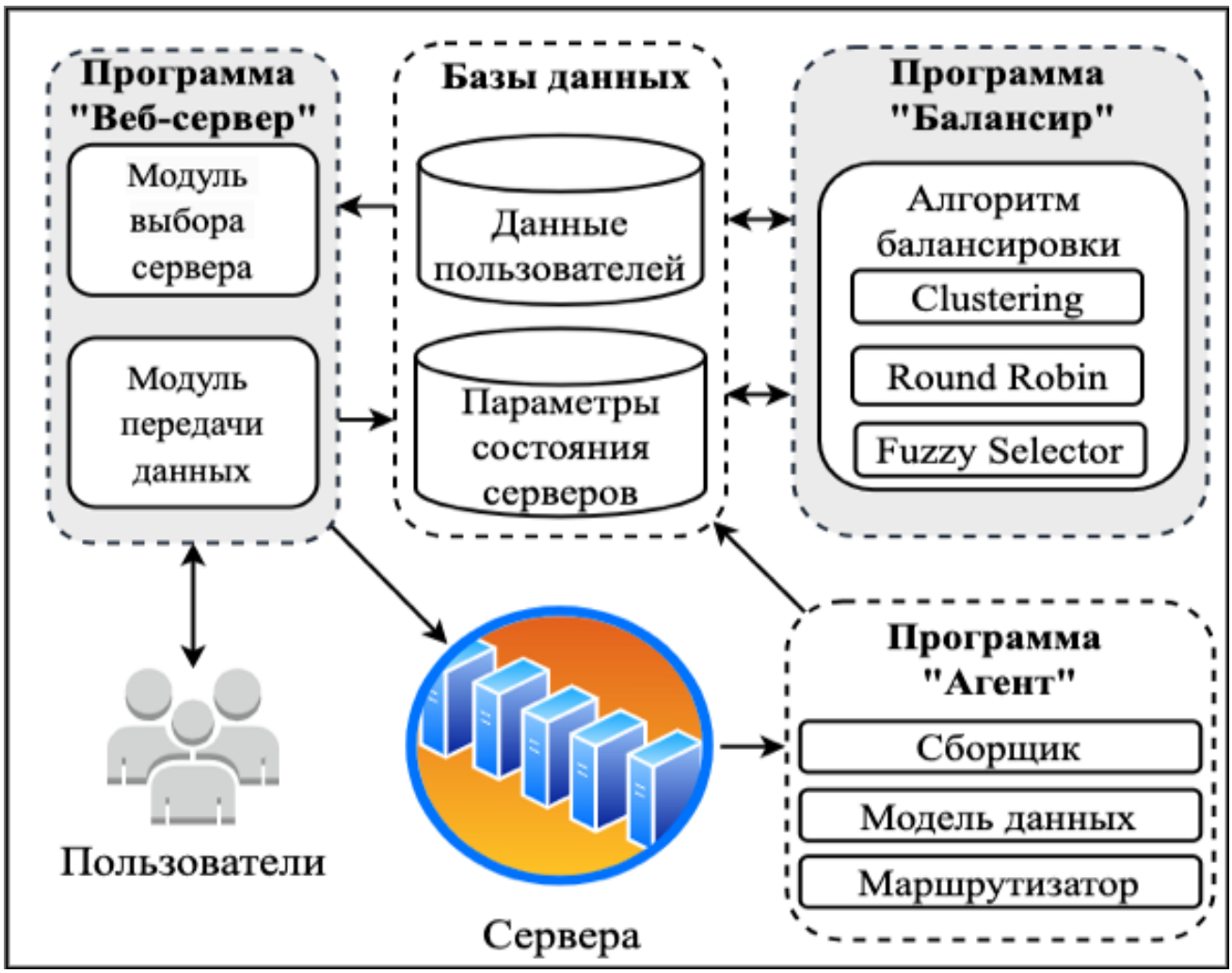


Рисунок 4.1 – Схема серверного приложения с балансиром нагрузки на сервере параллельных вычислений

При выполнении экспериментальных исследований получено, что выполнение процедур по выбору сервера внутри основного потока выполнения запроса приводит к значительному увеличению времени его выполнения. Так при выполнении вычислительного запроса длительность запроса увеличилась на 0,3-0,5 секунды, что в 1,5-2 раза увеличивает время выполнения запроса. Для того чтобы процедура выбора подходящего сервера не влияла на время выполнения запроса пользователя было принято решение о переносе вычислений по выбору сервера в параллельный процесс. На рисунке 4.2 представлен алгоритм параллельных вычислений [103] при выполнении запроса пользователя.

Таким образом, серверное приложение разделено на три логических части: веб-сервер, состоящий из модуля обработки запроса и модуля передачи данных, вычислительных серверов, решающих поставленную задачу, и сервера параллельных

вычислений, который осуществляет непосредственный выбор сервера.

При отправке l-го запроса пользователя n происходит отправка данных о пользователе (ip-адрес, файл, в случае задачи загрузки статических данных или текстовая подстрока в случае вычислительной задачи) серверу балансиру.

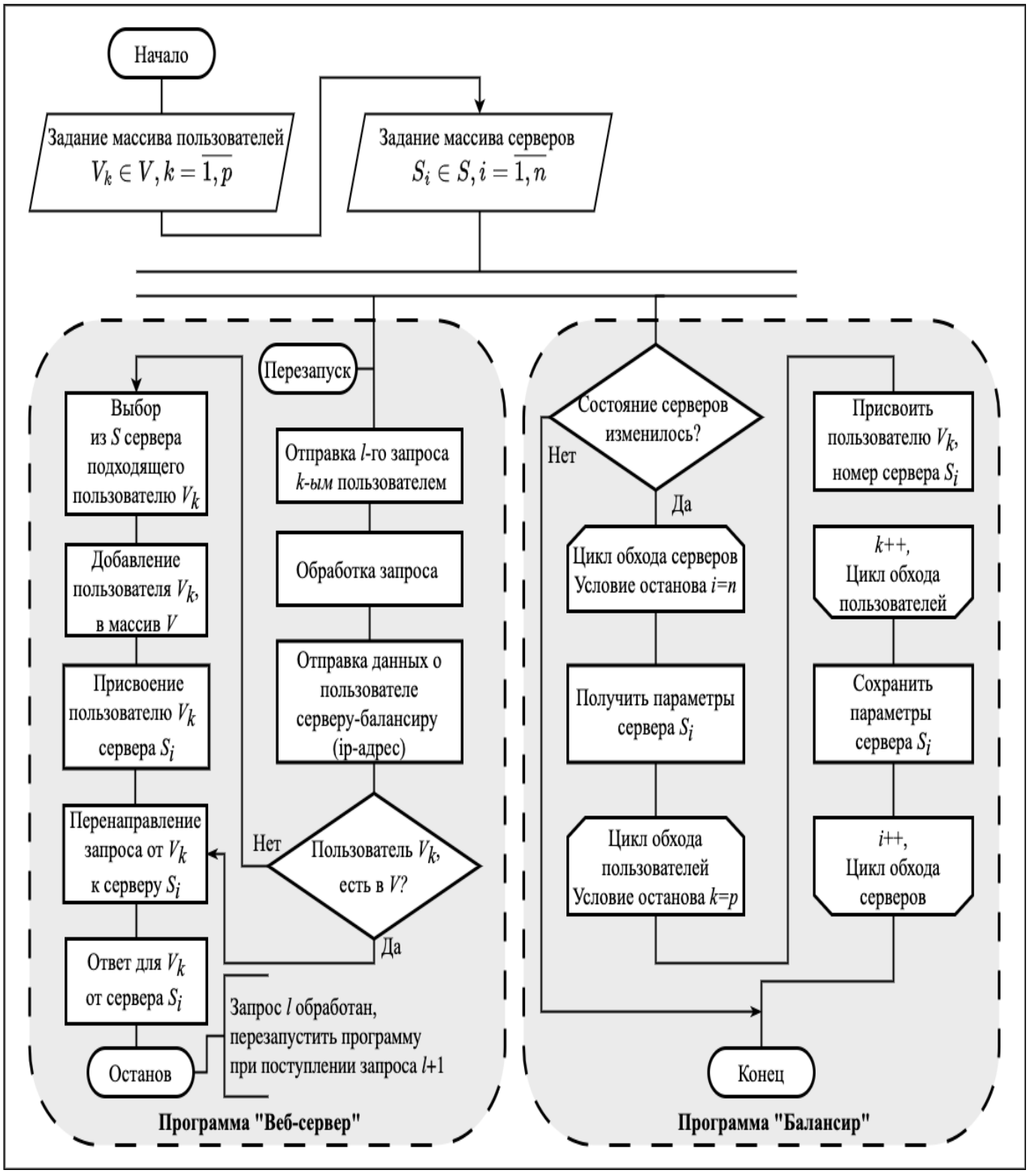


Рисунок 4.2 – Алгоритм параллельных вычислений балансировки нагрузки

Если пользователь выполняет первый запрос к серверу балансиру, то этот запрос будет переадресован первому доступному серверу, а ip-адрес пользователя будет перехвачен и попадет в список известных серверному приложению. Программный код добавления пользователя в базу данных приведен на рисунках 4.3 и 4.4.

```
class ServerCookies
{
    /**
     * Set cookie id to user
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle($request, Closure $next)
    {
        $clientId = $request->cookie('client_id', null);

        if (empty($clientId)) {
            $uuid = (string) Str::uuid();

            $server = Arr::wrap($request->server->all());

            dispatch(new CreateUserJob($server, $uuid));

            return $next($request)
            ->withCookie(cookie()->forever('client_id', $uuid));
        }
        return $next($request);
    }
}
```

Рисунок 4.3 – Программный код обработки запроса пользователя

На рисунке 4.3 показано, что все запросы пользователей перехватываются и обрабатываются. Если у пользователя нет идентификатора в файлах, то создается новый фоновый процесс, который получает данные о текущем местоположении пользователя и записывает их в базу данных, рисунок 4.4.

Пользователю устанавливается идентификатор, по которому в дальнейшем сервер балансир будет определять пользователя даже в случае смены пользователем ip-адреса. Если пользователь уже обращался к серверному приложению с одного ip-адреса или имеет в заголовках http запроса параметр cookie с идентификатором

запроса, ранее установленный серверным приложением, то веб-сервер получает из базы данных, выбранный сервером балансиром адрес вычислительного ресурса и передает ему запрос. После обработки запроса пользователь получает ответ от сервера с результатами вычислений.

```
class CreateUserJob implements ShouldQueue
{
    use Dispatchable, InteractsWithQueue, Queueable,
SerializesModels;

    protected $request;
    protected $uuid;

    public $tries = 3;

    public function __construct($request, $uuid)
    {
        $this->request = $request;
        $this->uuid = $uuid;

        $this->onQueue('users');
    }

    public function handle()
    {
        $clientIp = $this->request['REMOTE_ADDR'];
        $currentUserInfo = Location::get($clientIp);

        if (!empty($currentUserInfo)) {
            User::create([
                'ip_address' => $clientIp,
                'latitude' => $currentUserInfo['latitude'],
                'longitude' => $currentUserInfo['longitude'],
                'uuid' => $this->uuid,
                'server_ip' => null
            ]);
        } else {
            Log::debug("User with ip address: {$clientIp} is not
recognized");
        }
    }
}
```

Рисунок 4.4 – Программный код добавления нового пользователя

Процедуры балансировки и выбора сервера в этой ситуации мало влияют на время выполнения запроса пользователя. Программный код выбора сервера приведен на рисунке 4.5.

```
class SetUsersServer extends Command
{
  CONST CLUSTERING = 1;
  CONST FUZZY = 2;

  protected $signature = 'users:set-server {method=1}';

  protected $description = 'Set most suitable server for user and
store in in Redis';

  public function handle()
  {
      $serverService = app(ServerService::class);

      $users = User::all()->toArray();
      $servers = Server::all()->toArray();

      foreach ($users as $user) {
        foreach ($servers as $server) {
        $serverIp = ($this->argument('method') === self::CLUSTERING)
          ? $serverService->detectServerClustering($user, $servers)
          : $serverService->detectServerFuzzy($user, $servers);

User::update(['id' => $user['id']], ['server_ip' => $server['ip']]);
Redis::set("user_{$user['uuid']}_sever_ip", $serverIp);
            }
        }

        $this->line('User servers are updated');
    }
}
```

Рисунок 4.5 – Программный код параллельного процесса выбора сервера

После выполнения процедуры балансировки каждому пользователю будет установлен сервер, который и будет в дальнейшем обрабатывать его запросы. В связи с динамически меняющимся характером параметров состояния серверов процедуру балансировки предлагается выполнять по расписанию с периодичностью от одного раза в минуту до одного раза в сутки, в зависимости от количества пользователей приложения и количества вычислительных серверов. Функции *detectServerClustering* и *detectServerFuzzy* реализуют процедуры выбора сервера на основе деревьев принятия решений и нечеткой логики соответственно.

Распределение новых запросов пользователей, уже присутствующих в системе выполняет веб-сервер, данные о сервере для вычислений веб-сервер получает из базы

данных, с помощью модуля OpenResty [95] и модуля Nginx Http Upstream [40]. Программный код конфигурационного файла веб-сервера приведен на рисунке 4.6.

```
worker_processes 1;
error_log logs/error.log info;
events
{
    worker_connections 512;
}

http {
    upstream app {
        server 0.0.0.1;
        balancer_by_lua_block {
            local balancer = require "ngx.balancer"

            local port = 80
            local host = ngx.var.server_host

            local ok, err = balancer.set_current_peer(host, port)

            if not ok then
            ngx.log(ngx.ERR, "failed to set the current peer: ", err)
                return ngx.exit(500)
            end;
        }
    }

    server {
        listen 80;

        location /api {
            access_by_lua '
                local redis = require "resty.redis"
                local red = redis:new()
                local ok, err = red:connect("172.17.0.1", 6379)

                local uuid = ngx.var.cookie_Uuid
                local user_ip = red:get("user_{uuid}_sever_ip")
                local ip = if user_ip == nil or user_ip == '' < 0 then
'127.0.0.1' else user_ip end
 ngx.var.server_host = ip
            ';
            proxy_pass http://app;
        }
    }
}
```

Рисунок 4.6 – Программный код конфигурационного файла сервера балансира

Для того чтобы проверить результаты аналитико-имитационных исследований и

оценить эффективность применения сервера балансира, были проведены натурные эксперименты по распределению запросов по трем облачным серверам.

4.2 Проведение экспериментальных исследований распределения вычислительных ресурсов в облачном серверном комплексе

Для проведения экспериментальных исследований сбора и обработки данных о состоянии серверов была разработана схема экспериментальных исследований (рисунок 4.7).

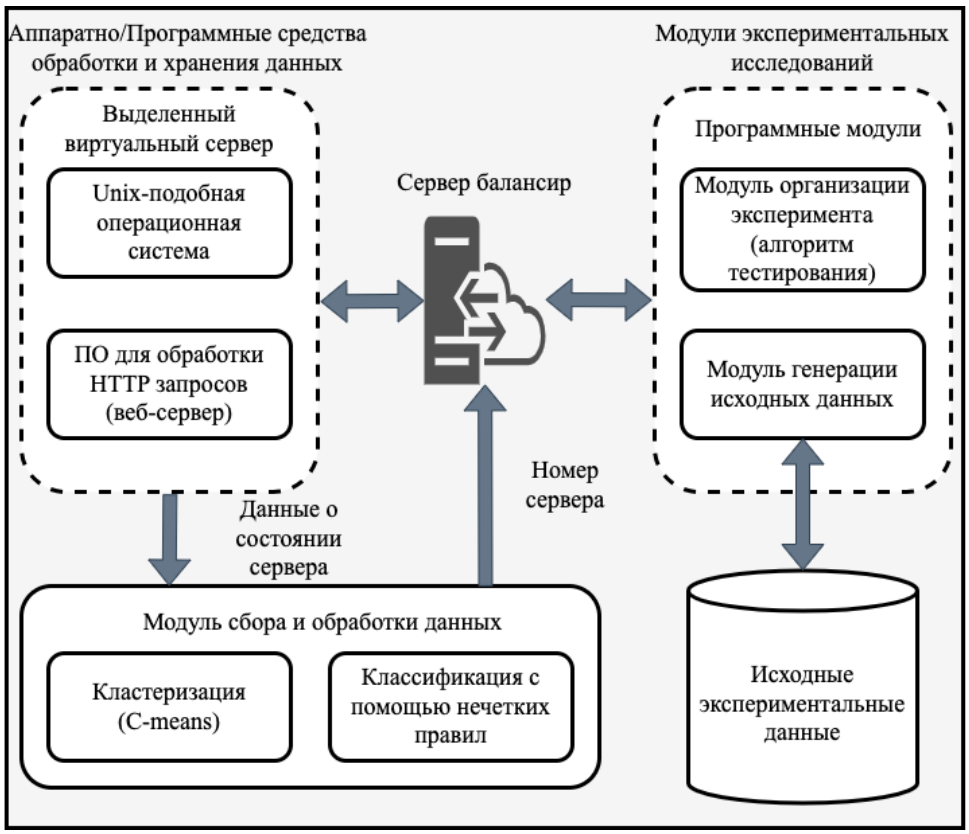


Рисунок 4.7 – Схема проведения экспериментальных исследований

Система включает в себя следующие компоненты: облачные сервера, компании Google с установленным дистрибутивом операционной системы Ubuntu Linux и веб-сервером Nginx. Программа проведения испытания разработанного серверного комплекса приведена в приложении В.

На данном этапе разрабатываемая система состоит из трех серверов, позволяющих хранить, обрабатывать и предоставлять пользователям данные. Используемые сервера распределены по трем центрам обработки данных.

Рассматриваемые центры обработки данных позволяют промоделировать работу

реальных клиент-серверных приложений географически распределенных информационных систем.

На арендованных вычислительных ресурсах провайдера Google Cloud Platform были развернуты приложения, выполняющие обработку вычислительных запросов пользователей, позволяющие выполнить поиск подстроки в тексте и сервер-балансир, выполняющий выбор оптимального ресурса.

На серверное приложение были отправлены тестовые наборы запросов с ключевыми фразами из текста, количество запросов q изменялось от 100 до 5000.

Выполнение запросов происходило с помощью запуска команды ab -kc 100 -n 1000 http://35.242.185.30/search?query=’substring’, где ab – название программы для выполнения тестирование, ключ -kc – позволяет выполнить параллельную отправку определенного количества запросов, указанную следующим параметром, идущим за ключом, ключ -n означает общее количество запросов, 35.242.185.30 – ip-адрес сервера балансира. Результат выполнения теста показывает среднее время обработки одного запроса (мс) и общее время выполнения теста (с).

4.3 Результаты экспериментальных исследований распределения нагрузки

Для того чтобы оценить время обработки запросов, серверным приложением было проведено три натурных эксперимента. Тестовая выборка запросов, созданная при помощи утилиты -ab, распределялась по трем серверам S_1, S_2, S_3, с использованием разработанных методов балансировки (кластеризации *С-средних*, нечеткой логики). Кроме того, использовались алгоритмы балансировки, встроенной в ПО сервера Nginx (круговое распределение Round Robin), и методы балансировки, предоставляемые провайдером вычислительного ресурса (Google Cloud Platform Balancer).

При выполнении первого эксперимента все сервера находились в состоянии покоя, т.е. в одинаковом состоянии, а ресурсы полностью доступны.

При помощи команды ab -kc 100 -n l http://35.242.185.30/search?query=’substring’, где $l = \overline{1,q}$, q=5000, выполним запросы к серверам. Результаты выполнения эксперимента приведены в таблице 4.1.

Таблица 4.1 – Результаты эксперимента балансировки нагрузки по трем одинаково нагруженным серверам

Кластеризация (*C-means*)			Нечеткая логика (*Fuzzy Selector*)			Круговое распределение (*Round Robin*)			Балансир провайдера (*Google Balancer*)		
q	T_{AV}, *c*	T_S, *c*	*q*	T_{AV},*c*	T_S, *c*	*q*	T_{AV}, *c*	T_S, *c*	*q*	T_{AV}, *c*	T_S, *c*
100	0,49	49	100	0,492	49,2	100	0,5	50	100	0,5	50
200	0,524	104,8	200	0,517	103,4	200	0,523	104,6	200	0,53	106
300	0,529	158,7	300	0,533	159,9	300	0,538	161,4	300	0,53	161,1
400	0,534	213,6	400	0,54	216	400	0,544	217,6	400	0,55	221,2
500	0,543	271,5	500	0,543	271,5	500	0,555	277,5	500	0,56	282
…	…	…	…	…	…	…	…	…	…	…	…
5000	0,68	3402,7	5000	0,664	3320	5000	0,686	3434,3	5000	0,69	3466,7

Графики общего и среднего времени выполнения тестовой выборки запросов приведены на рисунках 4.8, 4.9. Показано, что для одинакового состояния серверов нагрузка распределяется равномерно, и скорость обработки запросов близка для всех методов балансировки.

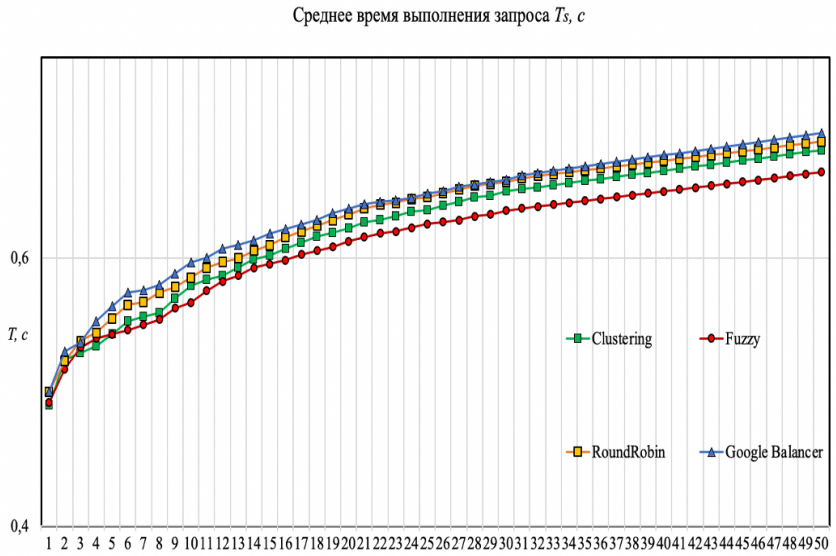


Рисунок 4.8 – Среднее время выполнения запроса, эксперимент 1

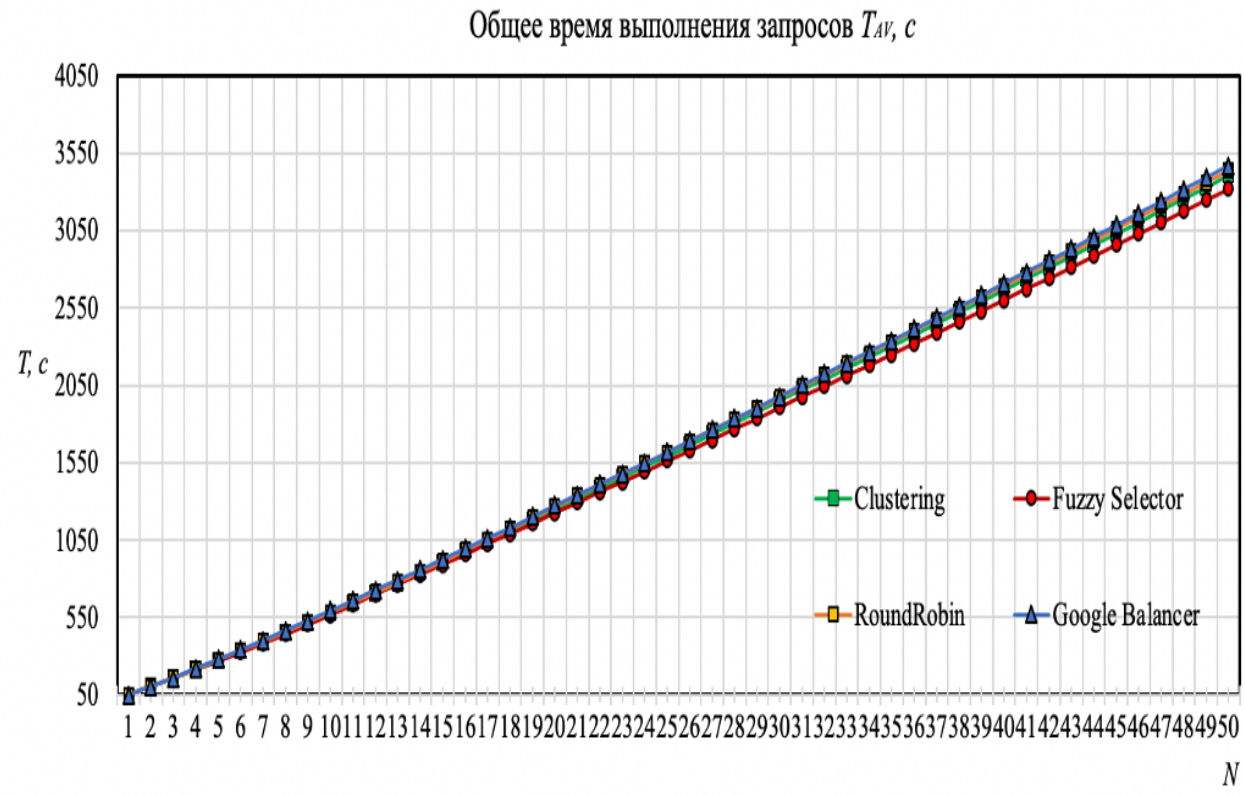


Рисунок 4.9 – Общее время выполнения запросов, эксперимент 1

Для проведения дальнейших экспериментов на сервера S_2, S_3 была оказана дополнительная нагрузка.

Таблица 4.2 – Результаты эксперимента балансировки нагрузки по трем одинаково нагруженным серверам

Кластеризация (*C-means*)			Нечеткая логика (*Fuzzy Selector*)			Круговое распределение (*Round Robin*)			Балансир провайдера (*Google Balancer*)		
q	T_{AV}, *c*	T_S, *c*	*q*	T_{AV},*c*	T_S, *c*	*q*	T_{AV}, *c*	T_S, *c*	*q*	T_{AV}, *c*	T_S, *c*
100	0,49	49	100	0,492	49,2	100	0,5	50	100	0,5	50
200	0,519	103,8	200	0,522	104,4	200	0,54	109	200	0,523	104,6
300	0,533	159,9	300	0,538	161,4	300	0,55	167,7	300	0,534	160,2
400	0,54	216	400	0,553	221,2	400	0,57	230	400	0,543	217,2
500	0,546	273	500	0,561	280,5	500	0,58	291,5	500	0,546	273
…	…	…	…	…	…	…	…	…	…	…	…
5000	0,684	3424	5000	0,685	3426	5000	0,71	3565	5000	0,705	3525

При помощи вызова команды stress, ОС Ubuntu центральный процессор и оперативная память серверов были нагружены на значение около 50% (по параметрам U_{ram}, U_{cpu}) от имеющейся мощности. Затем выполнина команда утилиты -ab аналогичную эксперименту 1. Результаты тестирования представлены в таблице 4.3 и на рисунках 4.10, 4.11. При выполнении второго эксперимента между разрабатываемыми методами балансировки (кластеризации *С-средних*, нечеткой логики), круговым распределением и балансиром нагрузки провайдера появилась разница, составляющая около **1-7%** в зависимости от метода балансировки.

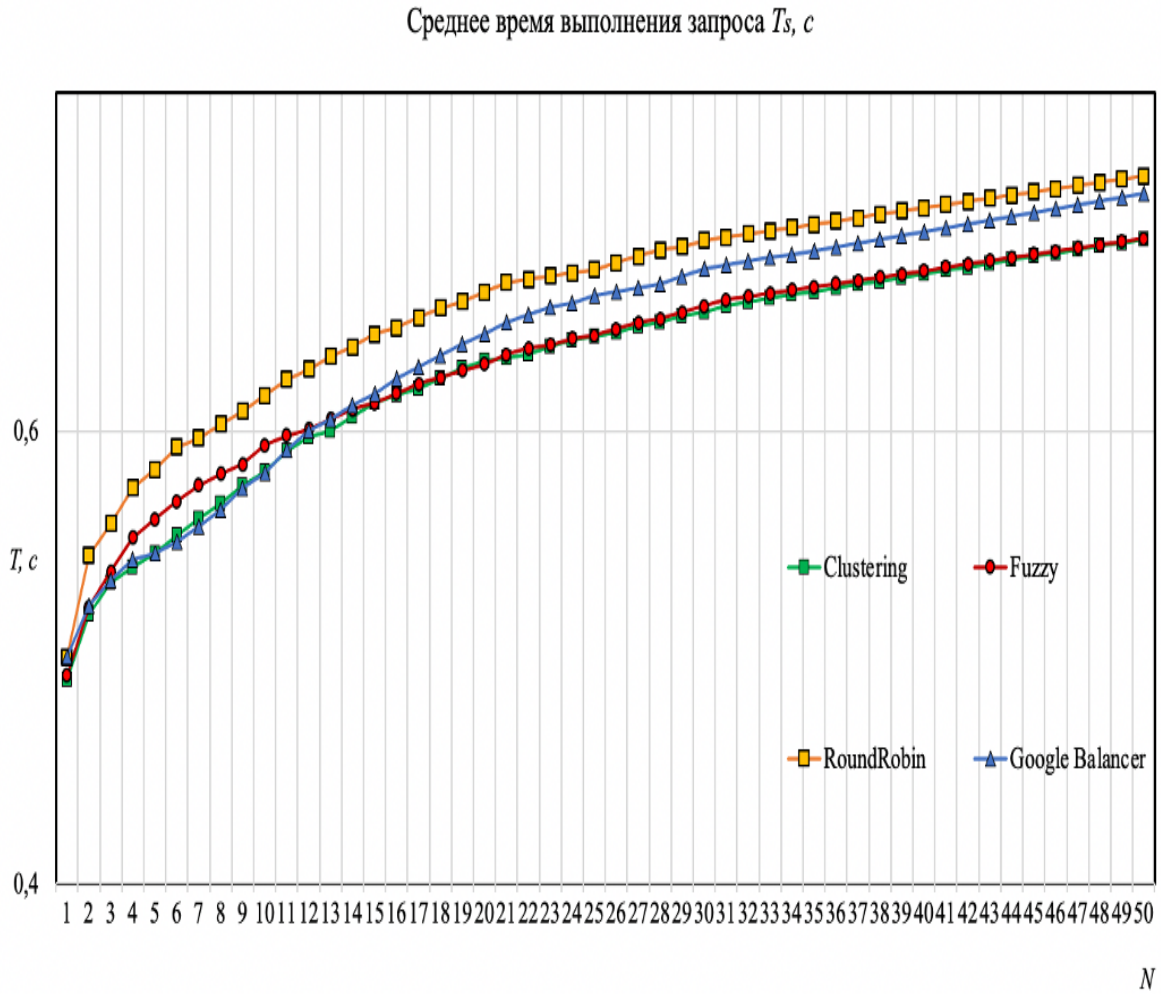


Рисунок 4.10 – Среднее время выполнения запроса, эксперимент 2

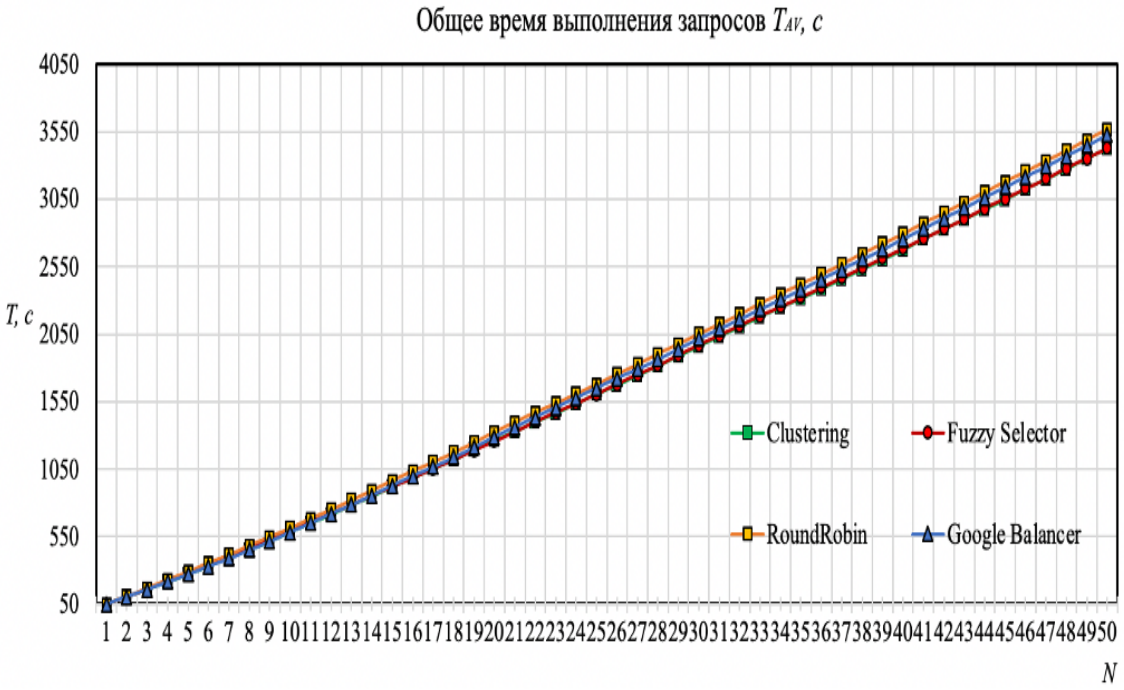


Рисунок 4.11 – Общее время выполнения запросов, эксперимент 2

При проведении третьего эксперимента нагрузка на сервера S_2, S_3 была увеличена до 90% от мощности аппаратного ресурса (параметры U_{ram}, U_{cpu}). Результаты эксперимента представлены в таблице 4.3 и на рисунках 4.12, 4.13.

Таблица 4.3 – Результаты эксперимента балансировки нагрузки по серверам, при нагрузке около 90% от имеющейся мощности для серверов S_2, S_3

Кластеризация (*C-means*)			Нечеткая логика (*Fuzzy Selector*)			Круговое распределение (*Round Robin*)			Балансир провайдера (*Google Balancer*)		
q	T_{AV}, *c*	T_S, *c*	*q*	T_{AV}, *c*	T_S, *c*	*q*	T_{AV}, *c*	T_S, *c*	*q*	T_{AV}, *c*	T_S, *c*
100	0,49	49	100	0,492	49,2	100	0,5	50	100	0,5	50
200	0,507	101,4	200	0,518	103,6	200	0,541	108,2	200	0,54	108
300	0,521	156,3	300	0,529	158,7	300	0,566	169,8	300	0,562	168,6
400	0,53	212	400	0,542	216,8	400	0,577	230,8	400	0,578	231,2
500	0,534	267	500	0,546	273	500	0,582	291	500	0,583	291,5
…	…	…	…	…	…	…	…	…	…	…	…
5000	0,688	3440	5000	0,669	3345	5000	0,750	3750	5000	0,714	3571

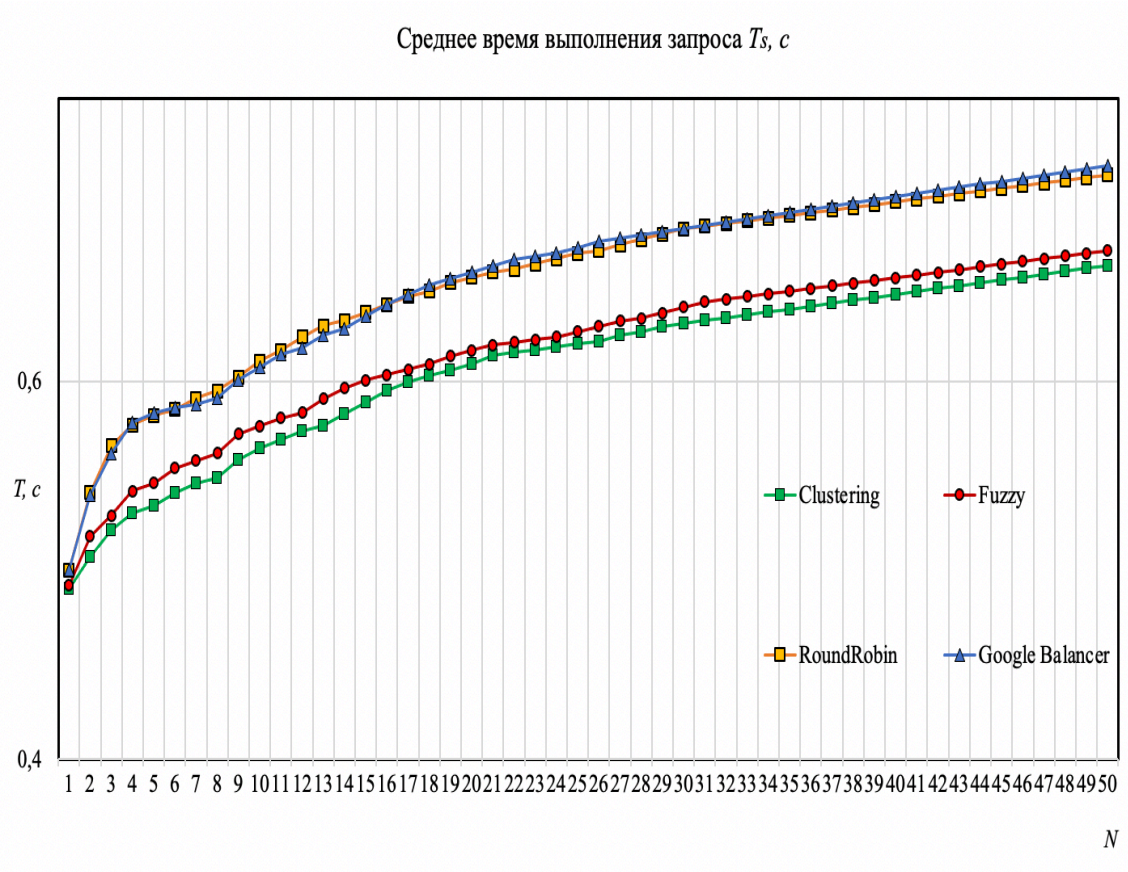


Рисунок 4.12 – Среднее время выполнения запроса, эксперимент 3

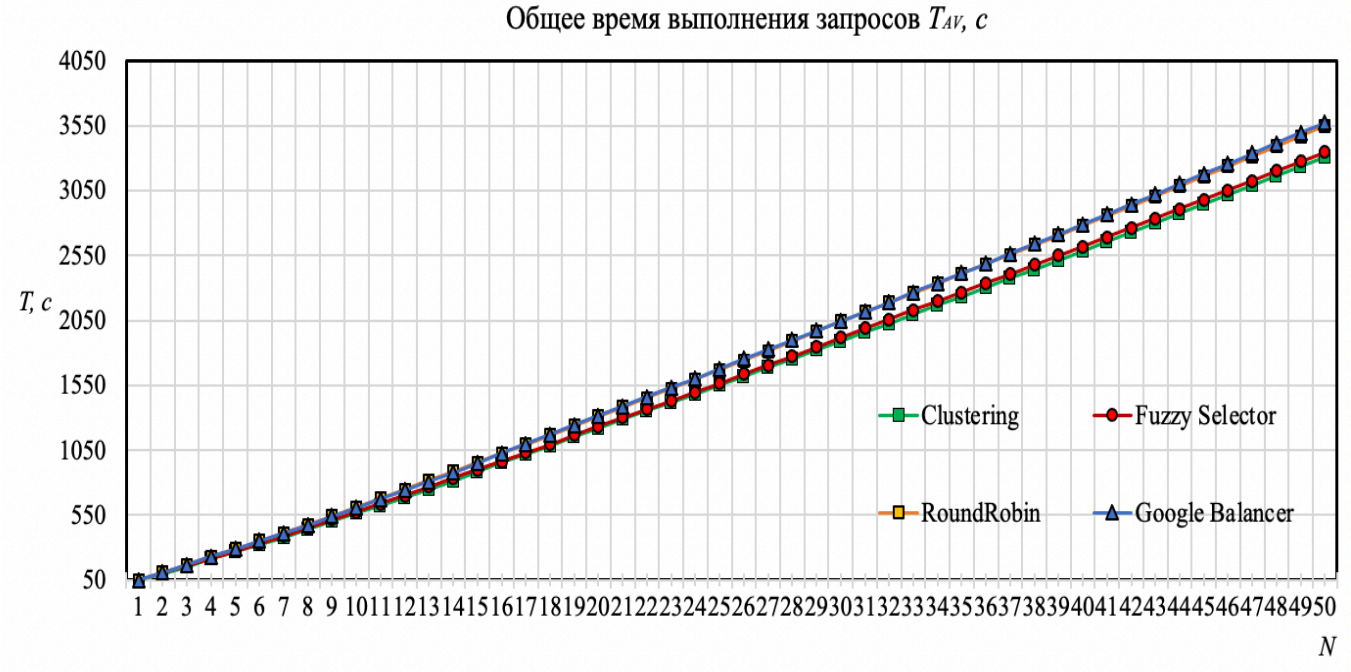


Рисунок 4.13 – Общее время выполнения запросов, эксперимент 3

После выполнения третьего эксперимента выявлено, что между разрабатываемыми методами балансировки (кластеризации *C-средних*, нечеткой

логики), круговым распределением и балансиром нагрузки провайдера увеличилось разница во времени выполнения запросов.

Установлено, что время выполнения запросов при распределении с помощью разработанных методов уменьшилось на 2-10 % по сравнению с круговым распределением. Отметим, чем выше нагрузка оказывалась на сервера S_2, S_3, тем более производительными оказывались разработанные методы балансировки.

4.4 Выводы по главе

1. Разработан программный комплекс и алгоритм параллельных вычислений балансировки нагрузки. В ходе проведения натурных экспериментов выявлено, что выполнение балансировки в основном потоке запроса пользователя существенно увеличивает время выполнения запроса.

Это приводит к необходимости реализации выбора подходящего для пользователя сервера в вычислительном процессе, отдельном от выполнения запроса. Для реализации такого метода балансировки разработан алгоритм параллельных вычислений для системы распределения вычислительной нагрузки.

2. На вычислительных ресурсах облачного серверного комплекса проведены экспериментальные исследования по распределению вычислительной нагрузки. Выполнено три эксперимента. В первом эксперименте все сервера находились в состоянии покоя (на сервера не оказывалась нагрузка).

При проведении второго эксперимента с помощью утилиты stress ОС Ubuntu сервера были нагружены на 50% (параметры U_{ram}, U_{cpu}) от имеющейся мощности. В третьем эксперименте нагрузка составила 90%. Для каждого из экспериментов получены временные характеристики среднего времени выполнения запроса и общего времени выполнения запросов тестовой выборки.

3. В результате экспериментальных исследований распределения вычислительной нагрузки выявлено, что при равной нагрузке или в случае отсутствия нагрузки, все методы балансировки нагрузки работают с приблизительно одинаковой скоростью.

При росте нагрузки на аппаратный ресурс серверов появляется разница между

скоростью работы методов балансировки. Предложенные методы балансировки нагрузки показали увеличение скорости обработки запросов на **2-10** % по сравнению с методом кругового распределения запросов Round Robin, и встроенным балансиром нагрузки провайдера аппаратного ресурса Google Cloud Platform.

ЗАКЛЮЧЕНИЕ

В результате анализа проблемы повышения быстродействия и производительности работы клиент-серверных приложений и информационных систем выявлено, что известные методы распределения нагрузки не учитывают или учитывают не в полной мере состояние вычислительных ресурсов (данные о доступности ресурса, а не о его состоянии). Установлено, что для увеличения количества обработанных запросов и уменьшение временных затрат на обработку необходимо распределять нагрузку в зависимости от состояния вычислительных ресурсов.

2. Выполнено обоснование показателей (параметров состояния серверного комплекса) для выбора сервера на основе паттерн-кластеризации. Показано, что для выбора показателей качества балансировки вычислительной нагрузки (вычислительных и сетевых параметров) пригодны методы как порядково-фиксированной (порядково-зависимой), так и порядково-инвариантной (порядково-независимой) паттерн-кластеризации.

Для задачи распределения статических данных применение порядково-инвариантной кластеризации позволило выявить необходимость привлечения мультипликативного показателя (сетевых параметров) для лучшей отделимости данных.

3. Показано, что для решения задачи балансировки нагрузки пригодны методы кластерного анализа данных о состоянии серверов, применение которых позволило разделить запросы пользователей по серверам. Установлено, что применение кластерного анализа позволяет получить диапазоны изменения параметров состояния и синтезировать продукционные правила для процедуры выбора сервера на основе нечетного логического вывода.

4. Разработан аналитико-имитационный метод распределения нагрузки на основе нечеткого логического вывода, включающий создание структуры данных с использованием кластерного анализа, имитационной модели серверного комплекса и алгоритма распределения нагрузки на основе нечеткого логического вывода. Проведены имитационные исследования, показывающие преимущества разработанного метода распределения нагрузки по сравнению с существующими.

Установлено, что средняя стоимость обработки запросов уменьшилась на 10-17%, среднее время пребывания запросов сократилось на 20-60%, а количество обработанных увеличилось на 20-60% по сравнению с круговым методом распределения.

5. Разработанный программный комплекс, структура и алгоритм параллельных

вычислений для балансировки нагрузки. Проведены экспериментальные исследования эффективности распределения данных между серверами. *Получено*, что применение разработанного алгоритмов распределения нагрузки, увеличивает скорость обработки запросов пользователей на 2-10% по сравнению с известными алгоритмами балансировки.

СПИСОК СОКРАЩЕНИЙ

РСД – распределение статических данных;

РВН – распределение вычислительной нагрузки;

СМО – система массового обслуживания;

ЦОД – центр обработки данных;

ВС – веб-сервер;

ВП – веб-приложение;

БД – база данных;

ОС – операционная система;

ПО – программное обеспечение;

CDN – сеть доставки данных (*content delivery network*);

RTT – время приема-передачи (*round-trip time*);

DNS – система доменных имен (*domain name system*);

LB – балансировка нагрузки (*load balancing*),

NLB – балансировки сетевой нагрузки (*network Load Balancer*),

OSI – сетевая модель стека сетевых протоколов (Open Systems Interconnection model).

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Анализ паттернов в статике и динамике. Часть 1: Обзор литературы и уточнение понятия / Ф. Т. Алескеров, В. Ю. Белоусова, Л. Г. Егорова, Б. Г. Миркин // Бизнес-информатика. – 2013. – № 3 (25). – С. 3–18.

2. Ануфриев, И. Е. MATLAB 7 / И. Е. Ануфриев, А. Б. Смирнов, Е. Н. Смирнова. – Санкт-Петербург : БХВ-Петербург, 2005. – 1104 с. – ISBN 5-94157-494-0.

3. Андрейчиков, А. В. Анализ, синтез, планирование решений в экономике : учеб. / А. В. Андрейчиков, О. Н. Андрейчикова. – Москва : Финансы и статистика, 2000. – 364 с. – ISBN 5-279-02188-1.

4. Борисов, А. Н. Принятие решений на основе нечетких моделей: Примеры использования / А. Н. Борисов, О. А. Крумберг, И. П. Федоров. – Рига : Знание, 1990. – 184 с. – ISBN 5-7966-0459-7.

5. Бендат, Дж. Прикладной анализ случайных данных / Дж. Бендат, А. Пирсол ; пер. с англ. В. Е. Привольского, А. И. Кочубинского. – Москва : Мир, 1989. – 540 с. – ISBN 5-03-001071-8.

6. Васильев, В. И. Интеллектуальные системы управления с использованием нечеткой логики : учеб. пособие / В. И. Васильев, Б. Г. Ильясов ; Уфим. гос. авиац. техн. ун-т. – Уфа : Изд-во УГАТУ, 1995. – 99 с. – ISBN 5-86911-082-3.

7. Вентцель, Е. С. Теория случайных процессов и ее инженерные приложения : учеб. пособие / Е. С. Вентцель, В. А. Овчаров. – 2-е изд., стер. – Москва : Высшая школа, 2000. – 383 с. – ISBN 5-06-003831-9.

8. Викулов, Е. О. Распределение данных и вычислений высоконагруженных веб-приложений / Е. О. Викулов, Е. А. Леонов // Информационные технологии и автоматизация управления : материалы V Всерос. науч.-практ. конф. студентов, аспирантов, работников образования и пром-сти (Омск, 23–26 апр. 2013 г.) / Ом. гос. техн. ун-т. – Омск : Изд-во ОмГТУ, 2013. – С. 141–145.

9. Викулов, Е. О. Автоматизированное распределение больших объемов данных высоконагруженных систем / Е. О. Викулов, Е. А. Леонов, Л. А. Денисова // Динамика систем, механизмов и машин. – 2014. – № 3. – С. 146–149.

10. Викулов, Е. О. Применение методов кластеризации при распределении больших объёмов данных высоконагруженных систем / Е. О. Викулов, Е. А. Леонов // Информационные технологии и автоматизация управления : сб. науч. тр. / Ом. гос. техн. ун-т. – Омск : Изд-во ОмГТУ, 2014 – С. 7–12.

11. Викулов, Е. О. Распределение данных высоконагруженных веб-приложений / Е. О. Викулов // Информационные технологии и автоматизация управления : материалы VIII Всерос. науч.-практ. конф. студентов, аспирантов, работников образования и пром-сти (Омск, 26–28 апр. 2016 г.) / Ом. гос. техн. ун-т. – Омск : Изд-во ОмГТУ, 2016. – С. 73–78.

12. Викулов, Е. О. Моделирование распределения нагрузки между серверными станциями / Е. О. Викулов, О. В. Денисов, М. А. Рудгальский // Информационные технологии и автоматизация управления : материалы VIII Всерос. науч.-практ. конф. студентов, аспирантов, работников образования и пром-сти (Омск, 26–28 апр. 2016 г.) / Ом. гос. техн. ун-т. – Омск : Изд-во ОмГТУ, 2016. – С. 38–43.

13. Викулов, Е. О. Исследование влияния системы доменных имен на скорость доставки статических данных / Е. О. Викулов // Информационные технологии и автоматизация управления : материалы VIII Всерос. науч.-практ. конф. студентов, аспирантов, работников образования и пром-сти (Омск, 18–19 мая 2017 г.) / Ом. гос. техн. ун-т. – Омск : Изд-во ОмГТУ, 2017. – С. 91–96.

14. Викулов, Е. О. Анализ данных о состоянии серверных станций высоконагруженных веб-приложений / Е. О. Викулов // Информационные технологии и автоматизация управления : материалы X Всерос. науч.-практ. конф. студентов, аспирантов, работников образования и пром-сти (Омск, 15–16 мая 2019 г.) / Ом. гос. техн. ун-т. – Омск : Изд-во ОмГТУ, 2019. – С. 62–67.

15. Викулов, Е. О. Исследование распределения данных высоконагруженных веб-приложений с применением нейросетевых технологий / Е. О. Викулов // Омский научный вестник. – 2018. – № 6 (162). – С. 244–246.

16. Викулов, Е. О. Имитационное моделирование распределения вычислительной нагрузки между серверными станциями с использованием нечеткого логического вывода / Е. О. Викулов, О. В. Денисов, В. А. Мещеряков, Л. А. Денисова // Автоматизация в промышленности. – 2021. – № 9. – С. 7–14.

17. Денисов, О. В. Распределение данных в информационной системе с помощью сервера-балансира / О. В. Денисов, Е. О. Викулов // Прикладная математика и фундаментальная информатика. – 2019. – Т. 6, № 4. – С. 46–57.

18. Денисова, Л. А. Событийное моделирование системы управления с переменными параметрами / Л. А. Денисова // Проектирование инженерных и научных приложений в среде MATLAB : тр. V Междунар. науч. конф. / Нац. техн. ун-т «Харьк. политехн. ин-т». – Харьков : Изд-во НТУ ХПИ, 2011. – С. 608–614.

19. Денисова, Л. А. Событийное моделирование цифровой системы регулирования / Л. А. Денисова // Омский научный вестник. – 2011. – № 3 (103). – С. 261–265.

20. Денисова, Л. А. Модели и методы проектирования систем управления объектами с переменными параметрами : моногр. / Л. А. Денисова / Ом. гос. техн. ун-т. – Омск : Изд-во ОмГТУ, 2014. – 167 с. – ISBN 978-5-8149-1835-2.

21. Дьяконов, В. П. Справочник по применению системы PC MATLAB / В. П. Дьяконов. – Москва : Физматлит, 1993. – 111 с. – ISBN 5–02–015101–7.

22. Дьяконов, В. П. MATLAB 7.*/R2006/R2007 : самоучитель / В. П. Дьяконов. – Москва : ДМК Пресс, 2008. – 768 с. – ISBN 978-5-94074-424-55-5.

23. Дьяконов, В. П. MATLAB 6.5 SP1/7 + Simulink 5/6 в математике и моделировании / В. П. Дьяконов. – Москва : Солон-Пресс, 2005. – 581 с. – ISBN 5-98003-209-6.

24. Зиновьев, А. Ю. Визуализация многомерных данных : моногр. / А. Ю. Зиновьев. – Красноярск : Изд-во Красноярского гос. техн. ун-та, 2000. – 168 с. – ISBN: 5-7636-0333-8.

25. Задорожный, В. Н. О качестве программных генераторов случайных чисел / В. Н. Задорожный // Омский научный вестник. – 2009. – № 2 (80). – С. 199–205.

26. Иглин, С. П. Математические расчеты на базе MATLAB / С. П. Иглин. – Санкт-Петербург : БВХ-Петербург, 2005. – 634 с. – ISBN 5-941572-90-5.

27. Каллан, Р. Основные концепции нейронных сетей / Р. Каллан. – Москва : Вильямс, 2001. – 287 с. – ISBN 5-8459-0210-X.

28. Колесов, Ю. Б. Визуальное моделирование сложных динамических систем / Ю. Б. Колесов, Ю. Б. Сениченков. – Санкт-Петербург : Мир и Семья, 2000. – 242 с.

29. Колесов, Ю. Б. Объектно-ориентированное моделирование сложных динамических систем / Ю. Б. Колесов ; С.-Петерб. гос. политехн. ун-т. – Санкт-Петербург : Изд-во СПбГПУ, 2004. – 238 с. – ISBN 5-7422-0575-9.

30. Колесов, Ю. Б. Математическое моделирование гибридных динамических систем : учеб. пособие / Ю. Б. Колесов, Ю. Б. Сениченков ; С.-Петерб. гос. политехн. ун-т. – Санкт-Петербург : Изд-во СПбГПУ, 2014. – 235 с. – ISBN 978-5-7422-4183-6.

31. Круглов, В. В. Интеллектуальные информационные системы: компьютерная поддержка систем нечеткой логики и нечеткого вывода : учеб. пособие / В. В. Круглов, М. И. Дли. – Москва : Физматлит, 2002. – 254 с. – ISBN 5-94052-062-6.

32. Купер, Дж. Вероятностные методы анализа сигналов и систем / Дж. Купер, К. Макгиллем ; пер. с англ. Е. М. Липовецкого, А. И. Папкова. – Москва : Мир, 1989. – 376 с. – ISBN 5-03-000366-5.

33. Лазарев, Ю. Моделирование процессов и систем в MatLab : учеб. курс / Ю. Лазарев. – Санкт-Петербург : Питер, 2005. – 511 с. – ISBN 5-469-00600-X.

34. Леоненков, А. В. Нечеткое моделирование в среде MATLAB и fuzzyTECH / А. В. Леоненков. – Санкт-Петербург : БХВ-Петербург, 2003. – 736 с. – ISBN 5941570872.

35. Леонов, Е. А. Балансировка нагрузки: основные алгоритмы и методы / Е. А. Леонов, Л. А. Денисова // Информационные технологии и автоматизация управления : материалы VI Всерос. науч.-практ. конф. студентов, аспирантов, работников образования и пром-сти (Омск, 27–30 апр. 2015 г.) / Ом. гос. техн. ун-т. – Омск : Изд-во ОмГТУ, 2015. – С. 192–197.

36. Лоу, А. М. Имитационное моделирование / А. М. Лоу, В. Д. Кельтон. – 3-е изд. – Санкт-Петербург : Питер, 2004. – 846 с. – ISBN 5-94723-981-7.

37. Автоматизация синтеза и обучение интеллектуальных систем управления : сб. / отв. ред.: И. М. Макаров, В. М. Лохин. – Москва : Наука, 2009. – 228 с. – ISBN 978-5-02-036022-8.

38. Манжула, В. Г. Нейронные сети Кохонена и нечеткие нейронные сети в интеллектуальном анализе данных / В. Г. Манжула, Д. С. Федяшов // Фундаментальные исследования. – 2011. – № 4. – С. 108 –114.

39. Морозов, В. К. Моделирование информационных и динамических систем : учеб. пособие / В. К. Морозов, Г. Н. Рогачев. – Москва : Академия, 2011. – 377 с. – ISBN 978-5-7695-4221-3.

40. Модуль ngx_http_upstream_module. – URL: http://nginx.org/ru/docs/http/ngx_http_upstream_module.html (дата обращения: 30.03.2015).

41. Мещеряков, В. А. Обучение студентов имитационному моделированию систем массового обслуживания в MATLAB / В. А. Мещеряков, В. П. Денисов, Л. А. Денисова // Сборник материалов II торгового форума Сибири. – Омск : Изд-во РЭУ им. Г.В. Плеханова (Омский филиал), 2013. – С. 179–181.

42. Мячин, А. Л. Анализ паттернов: порядково-инвариантная паттерн-кластеризация / А. Л. Мячин // Управление большими системами : сб. тр. – 2016. – Вып. 61. – С. 41–59.

43. Нечеткие множества в моделях управления и искусственного интеллекта / под. ред. Д. А. Поспелова. – Москва : Наука, 1986. – 312 с.

44. Пегат, А. Нечеткое моделирование и управление / А. Пегат ; пер. с англ. А. Г. Подвесовского, Ю. В. Тюменцева. – 2-е изд. – Москва : БИНОМ. Лаб. знаний, 2013. – 798 с. – ISBN 978-5-9963-1495-9.

45. Прикладные нечеткие системы / под ред. Т. Тэрано, К. Асаи, М. Сугэно ; пер. с яп. Ю. Н. Чернышова. – Москва : Мир, 1993. – 368 с. – ISBN 5-03-002326-7.

46. Плизга, В. В. Построение моделей систем массового обслуживания средствами пакета MATLAB+Simulink / В. В. Плизга, А. Г. Киселев // WEBKURSOVIK.RU – ПОМОЩЬ СТУДЕНТУ. – URL: https://www.webkursovik.ru/kartgotrab.asp?id=-179781 (дата обращения: 06.07.2019).

47. Рогачев, Г. Н. Использование гибридно-автоматного метода для описания систем автоматизации и управления / Г. Н. Рогачев // Мехатроника, автоматизация, управление. – 2009. – № 12. – С. 14–19.

48. Рутковская, Д. Нейронные сети, генетические алгоритмы и нечеткие системы / Д. Рутковская, М. Пилиньский, Л. Рутковский. – Москва : Горячая линия - Телеком, 2008. – 383 с. – ISBN 5-93517-103-1.

49. Сениченков, Ю. Б. Численное моделирование гибридных систем / Ю. Б. Сениченков. – Санкт-Петербург : Изд-во Политехн. ун-та, 2004. – 206 с. – ISBN 5-7422-0730-1.

50. Теория систем с переменной структурой / под ред. С. В. Емельянова. – Москва : Наука, 1970. – 592 с.

51. Ульянов, С. В. Нечеткие модели интеллектуальных систем управления: теоретические и прикладные аспекты (обзор) / С. В. Ульянов // Известия Академии наук СССР. Техническая кибернетика. – 1991. – № 3. – С. 3–28.

52. Усков, А. А. Интеллектуальные системы управления на основе методов нечеткой логики / А. А. Усков, В. В. Круглов. – Смоленск : Смоленская гор. тип., 2003. – 176 с. – ISBN 5-94223-038-2.

53. Уоссермен, Ф. Нейрокомпьютерная техника: Теория и практика / Ф. Уоссермен ; пер. с англ. Ю. А. Зуева, В. А. Точенова. – Москва : Мир, 1992. – 237 с. – ISBN 5-03-002115-9.

54. Черных, И. В. Simulink: среда создания инженерных приложений / И. В. Черных. – Москва : Диалог-МИФИ, 2004. – 491 с. – ISBN 5-86404-186-6.

55. Хайкин, С. Нейронные сети : полный курс : пер. с англ. / С. Хайкин. – Москва : Вильямс, 2016. – 1104 с. – ISBN 978-5-8459-2069-0.

56. Штовба, С. Д. Проектирование нечетких систем средствами MATLAB / С. Д. Штовба. – Москва : Горячая линия - Телеком, 2007. – 288 с. – ISBN 5-93517-359-X.

57. Alakeel, A. M. A Guide to Dynamic Load Balancing in Distributed Computer Systems / A. M. Alakeel // International Journal of Computer Science and Network Security (IJCSNS). – 2010. – Vol. 10, no. 6. – P. 153–160.

58. AlKhatib, A. A. Load Balancing Techniques in Software-Defined Cloud Computing: an overview / A. A. AlKhatib, T. Sawalha, S. AlZu'bi. – DOI: 10.1109/SDS49854.2020.9143874 // The Seventh International Conference on Software Defined Systems (Paris, France, 20–23 April 2020). – IEEE, 2020. – P. 240–244.

59. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems / R. Alur, C. Courcoubetis, T. Henzinger, P. Ho // Hybrid Systems. – New York : Springer-Verlag, 1993. – Vol. 736. – P. 209–229.

60. Aleskerov, F. Personnel allocation among bank branches using a two-stage multicriterial approach / F. Aleskerov, H. Ersel, R. Yolalan // European Journal of Operational Research. – 2003. – Vol. 148, no. 1. – P. 116–125.

61. Aleskerov, F. A clustering approach to some monetary facts: a long-run analysis of cross-country data / F. Aleskerov, C. E. Alper // The Japanese Economic Review. – 2000. – Vol. 51, no. 4. – P. 555–567.

62. Amazon Web Services (AWS) - сервисы облачных вычислений. – URL: http://aws.amazon.com/ru/ (date accessed: 03.05.2020).

63. Classification and regression trees / L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone // Biometrics. – 1984. – Vol. 40, no. 3. – P. 874. New York, 1984. – 368 p. – ISBN 9781315139470.

64. How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime / A. Cervin, D. Henriksson, B. Lincoln [et al.] // IEEE Control Systems Magazine. – 2003. – Vol. 23, no. 3. – P. 16–30.

65. Arthur, D. How Slow is the k-means Method? / D. Arthur, S. Vassilvitskii // Proceedings of 22the annual symposium on Computational geometry (SoCG - 2006). – New York, 2006. – P. 144–153.

66. Denisov O. V. Load balancing in data distribution systems / O. V. Denisov, E. O. Vikulov // Journal of Physics: Conference Series. – 2020. – Vol. 1546, no. 1. – P. 012003.

67. Denisova, L. A. Event-Driven Simulation of Control Systems with Variable Parameters / L. A. Denisova // IFAC Proceedings Volumes. – 2013. – Vol. 46, no. 9. – P. 2179–2184.

68. Denisova, L. A. A Mathematical Model of a Digital Control System with Variable Parameters / L. A. Denisova // Automation and Remote Control. – 2012. – Vol. 73, no. 11. – P. 1895–1901.

69. DNS Round Robin: The Technology chronicle. – URL: http://thetechnologychronicle.blogspot.in/2013/11/dns-round-robin.html (date accessed: 21.02.2020).

70. Dunn, J. C. A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters / J. C. Dunn // Journal of Cybernetics. – 1973. – Vol. 3. – P. 32–57.

71. Mirkes, E. M. K-means and K-medoids applet / E. M. Mirkes ; University of Leicester.– 2011. – URL: http://www.math.le.ac.uk/people/ag153/homepage/KmeansKmedoids/Kmeans_Kmedoids.html (date accessed: 21.02.2020).

72. Eslami, G. New Replica Server Placement Strategies using Clustering Algorithms and SOM Neural Network in CDNs / G. Eslami, A. Haghihat, S. Farokhi //

The International Arab Journal of Information Technology. – 2017. – Vol. 14, no. 2. – P. 260–266.

73. Few, S. Multivariate Analysis Using Parallel Coordinates / S. Few. – URL: http://www.perceptualedge.com/articles/b-eye/parallel_coordinates.pdf (date accessed: 11.01.2016).

74. Flury, B. Principal points / B. Flury // Biometrika. – 1990. – Vol. 77, no. 1. – P. 33–41.

75. Goncharenko, V. A. Cluster Load Balancing Algorithms Based on Shortest Queue Models / V.A. Goncharenko, V.A. Lokhvitsky // Intellectual Technologies on Transport. – Mozhaisky Military Space Academy, Saint Petersburg, Russia, 2023. – P. 37–45.

76. Principal Manifolds for Data Visualisation and Dimension Reduction / eds.: A. N. Gorban, B. Kegl, D. Wunsch, A. Y. Zinovyev. – Berlin ; Heidelberg ; New York : Springer, 2007. – XXIV. – 340 p. – (Series: Lecture Notes in Computational Science and Engineering. – Vol. 58). – ISBN 978-3-540-73749-0.

77. The elements of statistical learning: Data mining, inference, and prediction / T. Hastie, R. Tibshirani, J. H. Friedman, J. Franklin // The Mathematical Intelligencer. – 2004. – Vol. 27, no. 2. – P. 83–85.

78. Henriksson, D. TrueTime: Simulation of control loops under shared computer resources / D. Henriksson, A. Cervin, K. E. Årzén // IFAC Proceedings Volumes. – 2002. – Vol. 35, no. 1. – P. 417–422.

79. Hofmann, M. Content Networking: Architecture, Protocols, and Practice / M. Hofmann, L. Beaumont. – Morgan Kaufmann Publisher, 2005. – 352 p. – ISBN 1-55860-834-6.

80. IBM:website.–URL:http://www-03.ibm.com/systems/ru/x/hardware/rack/x3250m3/index.html (date accessed: 08.02.2019).

81. Kohonen, T. The Self Organizing Map / T. Kohonen. – URL: https://sci2s.ugr.es/keel/pdf/algorithm/articulo/1990-Kohonen-PIEEE.pdf (date accessed: 09.03.2022).

82. Fusion of Neural Networks, Fuzzy Systems and Genetic Algorithms: Industrial Applications / ed.: Lakhmi C. Jain, N. M. Martin. – Boca Raton : CRC Press, CRC Press LLC, 1999. – 362 p. – ISBN 0-8493-9804-5.

83. Dynamical Properties of Hybrid Automata / J. Lygeros, K. H. Johansson, S. N. Simic [et al.] // IEEE Transactions on Automatic Control. – 2003. – Vol. 48, no. 1. – P. 2–17.

84. Qiu, L. On the placement of Web server replicas / L. Qiu, V. Padmanabhan, G. Voelker // Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Anchorage, AK, 22–26 April 2001). – IEEE, 2001. P. 1587–1596.

85. MacQueen, J. Some methods for classification and analysis of multivariate observations / J. MacQueen // Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability Proc. 5th Berkeley Symp. on Math. Statistics and Probability. – Berkeley, 1967. – P. 281–297. – URL: http://www.cs.cmu.edu/~bhiksha/courses/mlsp.fall2010/class14/macqueen.pdf (date accessed: 09.03.2022).

86. Mamdani, E. H. Application of fuzzy set theory to control systems: a survey / E. H. Mamdani // Fuzzy Automata and Decision Processes. – North-Holland, 1977. – C. 77–88.

87. MATLAB SimEvents User's Guide. – URL: http://www.mathworks.com (date accessed:12.09.2021).

88. Membrey, P. Practical Load Balancing: Ride the Performance Tiger / P. Membrey, E. Plugge, D. Hows. – New York, 2012. – 272 p. – ISBN 978-1-4302-3680-1.

89. Mirkin, B. Clustering for data mining: A data recovery approach / B. Mirkin. – New York : Chapman and Hall/CRC, 2005. – 296 p. – ISBN 9780429137549.

90. Mirkin, B. Mathematical classification and clustering / B. Mirkin. – Boston : Springer, 1996. – 448 p. – ISBN 978-0-7923-4159-8.

91. MongoDb : website. – URL: https://www.mongodb.com/ (date accessed: 17.03.2017).

92. Solving Replica Placement and Request Distribution in Content Distribution Networks / T. Neves, L. Drummond, L. Ochi [et al.]. – DOI:10.1016/j.endm.2010.05.012 // Electronic Notes in Discrete Mathematics. – 2010. – Vol. 36. – P. 89–96.

93. Nginx: документация. – URL: http://nginx.org/ru/docs (date accessed: 30.03.2015).

94. NodeJs : website. – URL: https://nodejs.org/ (date accessed: 17.03.2017).

95. OpenResty : website. – URL: http://openresty.org/en/ (date accessed:12.11.2019).

96. Балансировка нагрузки сети: описание технологии // Oszone.net : компьют. информац. портал. – URL: http://www.oszone.net/4187/Network_Load_Balancing (дата обращения: 30.03.2017).

97. Pedrycz, W. An Introduction to Fuzzy Sets: Analysis and Design / W. Pedrycz, F. Gomide. – MIT Press, 1998. – 465 p. – ISBN 9780262161718.

98. Chen, G. Introduction to Fuzzy Sets, Fuzzy Logic and Fuzzy Control Systems / G. Chen, T. Pham. – Boca Raton : Lewis Publishers, 2000. – 328 p. – ISBN 0-8493-1658-8.

99. Krishnan, P. The cache location problem / P. Krishnan, D. Raz, Y. Shavitt. – DOI: 10.1109/90.879344 // IEEE/ACM Transactions on Networking. – 2000. – Vol. 8, no. 5. – P. 568–582.

100. Sarmila, G. P. Survey on fault tolerant - Load balancing algorithmsin cloud computing / G. P. Sarmila, N. Gnanambigai, P. Dinadayalan. – DOI: 10.1109/ECS.2015.7124879 // 2nd International Conference on Electronics and Communication Systems (ICECS). – Coimbatore, India : IEEE, 2015. – P. 1715–1720.

101. Quinlan, J. R. Induction of Decision Trees. – DOI.org/10.1007/BF00116251 / J. R. Quinlan // Machine Learning. – 1986. – Vol. 1. – P. 81–106.

102. Roubos, J. A. Learning Fuzzy Classification Rules from Labeled Data / J. A. Roubos, M. Setnes, J. Abonyi. – DOI.org/10.1016/S0020-0255(02)00369-9 // Information Sciences. – 2003. – Vol. 150. – P. 77–93.

103. Sequential and Parallel Algorithms and Data Structures / P. Sanders K. Mehlhorn, M. Dietzfelbinger, R. Dementiev. – Switzerland AG : Springer Nature, 2019. – 509 p. – ISBN 978-3-030-25208-3.

104. A Survey on Replica Server Placement Algorithms for Content Delivery Networks / J. Sahoo, M. A. Salahuddin, R. Glitho [et al.]. – DOI: 10.1109/COMST.2016.2626384 // IEEE Communications Surveys & Tutorials. – 2017. – Vol. 19, no. 2. – P. 1002–1026.

105. Simulink: software for numerical simulation of continuous processes. – URL: http://www.mathworks.com/products/simulink/ (date accessed: 30.03.2015).

106. Stateflow: framework for simulation of eventdriven systems. – URL: http://www.mathworks.com/products/stateflow/ (date accessed: 30.03.2015).

107. Suherman, Z. M. Applying a rateless code in content delivery networks / Z. M. Suherman, S. Stirous, M. Al-Akaidi. – DOI 10.1088/1757-899X/237/1/012016 // IOP Conference Series: Materials Science and Engineering. – 2017. – Vol. 237, no. 1. – P. 012016.

108. Szymaniak, M. Latency-Driven Replica Placement / M. Szymaniak, G. Pierre, M. Steen // IPSJ Digital Courier. – 2006. – Vol. 2. – P. 561–572.

109. Vikulov, E. O. Data distribution system preparation of server stations data / E. O. Vikulov, O. V. Denisov, L. A. Denisova. – DOI:10.1088/1742-6596/1050/1/012097 // Journal of Physics: Conference Series. – 2018. – Vol. 1050. – P. 012097.

110. Vikulov, E. Event-Driven Simulation of Server Stations Load Balancing / E. Vikulov, O. Denisov, V. Meshcheryakov // International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM). – Moscow : IEEE, 2018. – P. 8728849.

111. Vikulov, E. O. Data distribution system: clustering based on neural network technologies / E. O. Vikulov, L. A. Denisova. — DOI:10.1088/1757-899X/537/5/052030 // IOP Conference Series Materials Science and Engineering. – 2019. – Vol. 537, no. 5. – P. 052030.

112. Simulation of data distribution between server stations using fuzzy technologies / E. O. Vikulov, O. V. Denisov, V. A. Meshcheryakov L. A. Denisova. – DOI:10.1088/1742-6596/1441/1/012175 // Journal of Physics: Conference Series. – 2020. – Vol. 1441. – P. 012175.

113. Wang, L. X. A Course in Fuzzy Systems and Control / L. X. Wang. – CliEs, NJ : Prentice Hall PTR, 1997. – 424 p.

114. Webmascon : website. – URL: http://webmascon.com/topics/technologies/4d.asp (date accessed: 01.07.2017).

115. Wilson, S. Server-based Dynamic Load Balancing / S. Wilson, P. Prakash, P. Deepalakshmi // International Conference on Networks & Advances in Computational Technologies (NetACT). – Thiruvananthapuram, India : IEEE, 2017. – P. 25–28.

116. Yin, H. Learning Nonlinear Principal Manifolds by Self-Organising Maps / H. Yin. – https://doi.org/10.1007/978-3-540-73750-6_3 // Principal Manifolds for Data Visualization and Dimension Reduction : Lecture Notes in Computational Science and Enginee. – Berlin, Heidelber : Springer, 2007. – P. 68–95.

117. Zadeh, L. A. Fuzzy sets / L. A Zadeh. – DOI.org/10.1016/S0019-9958(65)90241-X // Information and Control. – 1965. –Vol. 8, no.3. – P. 338–353.

118. Zimmermann, H.-J. Fuzzy Set Theory and its Applications / H.-J. Zimmermann. – 3 rd ed. – Dordreecht : Kluwer Academic Publishers, 1996. – 435 p. – ISBN 8170235251.

119. A new validity measure for a correlation-based fuzzy c-means clustering algorithm / M. Zhang, W. Zhang, H. Sicotte, P. Yang // Annual International Conference of the IEEE Engineering in Medicine and Biology Society. – Minneapolis, USA : IEEE, 2009. – P. 3865–3868.

ПРИЛОЖЕНИЕ А. АКТЫ ВНЕДРЕНИЯ

«УТВЕРЖДАЮ»
Директор ООО «РОНАС ИТ»
_____ Суриков Р.О.
«01» февраля 2021 г.

АКТ ВНЕДРЕНИЯ

от «01» февраля 2021 г., Омская обл., г. Омск, ООО «РОНАС ИТ»

Об использовании научных исследований и разработок в производственном процессе

В связи с выполненными ООО «РОНАС ИТ» работами для проекта «FreedomAdvisor» freedomadvisor.io (2020-2021 гг.) созданы и введены в эксплуатацию программные комплексы сбора и обработки данных серверных станций.

В рамках выполненных работ с целью обеспечения эффективного функционирования серверного комплекса и повышения скорости работы прикладных приложений (программных продуктов) реализованы следующие программные комплексы (исполнитель Викулов Е.О.).

1. «Программный комплекс сбора данных серверных станций». Используется для сбора данных о параметрах состояния вычислительных ресурсов.

2. «Программный комплекс балансировки нагрузки вычислительных ресурсов». Предназначен для распределения нагрузки в серверном комплексе между облачными вычислительными ресурсами.

Руководитель отдела разработки Е.А. Леонов

Федеральное государственное автономное образовательное учреждение
высшего образования
«Омский государственный технический университет»
Кафедра «Автоматизированные системы обработки информации и управления (АСОИУ)»

от «10» 05 2022г.
г. Омск
Об использовании научных
исследований и разработок
в учебном процессе

УТВЕРЖДАЮ
Проректор по образовательной
деятельности
_____ А.С. Полынский
«10» 05 2022 г.

АКТ ВНЕДРЕНИЯ

Основание: научные исследования, выполненные старшим преподавателем кафедры АСОИУ Викуловым Е.О.

Составлен комиссией в составе:
Никонов А.В. – зав. кафедрой АСОИУ, председатель комиссии;
Цыганенко В.Н., доцент кафедры АСОИУ;
Малков О.Б. – доцент кафедры АСОИУ.

1. Теоретические разработки Викулова Е.О., опубликованные в статьях:

Викулов Е.О., Леонов Е.А., Денисова Л.А. Автоматизированное распределение больших объёмов данных высоконагруженных систем. / Динамика систем, механизмов и машин, 2014. № 3. С. 146-149.

Викулов Е.О., Распределение больших объемов данных, Информационные технологии и автоматизация управления, материалы VIII Всероссийской научно-практической конференции студентов, аспирантов, работников образования и промышленности. 2016. С. 73-78.

Викулов Е.О. Исследование распределения данных высоконагруженных веб-приложений с применением нейросетевых технологий Омский научный вестник. 2018. № 6 (162). С. 244-246.

Vikulov E. O., Denisov O. V. and Denisova L. A. Data distribution system preparation of server stations data Journal of Physics: Conference Series Сер. "Mechanical Science and Technology Update, MSTU 2018" 2018. С. 012097.

Vikulov E, Denisov O, Meshcheryakov V 2018 Event-Driven Simulation of Server Stations Load Balancing International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM) Moscow, Russia.

Vikulov E O, Denisova L A 2019 Data distribution system: clustering based on neural network technologies J. Phys.: Conf. Ser. 537 052030

Викулов Е.О. Программный комплекс сбора данных о состоянии серверных станций / Е.О. Викулов, // Свидетельство о регистрации программы для ЭВМ 2021617727, 25.05.2021. Заявка № 2021616204 от 02.06.2021;

Викулов Е.О., Денисов О.В., Мещеряков В.А., Денисова Л.А. Имитационное моделирование распределения вычислительной нагрузки между серверными станциями с использованием нечеткого логического вывода, Автоматизация в промышленности. 2021. № 9. С. 7-14.

используются в следующих учебных дисциплинах кафедры АСОИУ:

– «Web-программирование»
– «Web-технологии»
– «Сетевые технологии»;
– «Основы теории управления»;

а также при руководстве курсовым и дипломным проектированием и научно-исследовательской деятельностью студентов.

2. В основе учебно-методических разработок, используемых в перечисленных учебных дисциплинах кафедры АСОИУ, лежат следующие научные результаты исследований Викулова Е.О.:

– комбинированный метод формирования показателей и правил выбора сервера при балансировке нагрузки на основе данных о состоянии серверного комплекса;
– аналитико-имитационный метод распределения вычислительной нагрузки с помощью нечеткого логического вывода, положенного в основу работы сервера-балансира;
– алгоритм параллельных вычислений для распределения данных между ресурсами облачного кластера серверных станций и структура программного комплекса для проведения экспериментальных исследований.

Председатель комиссии _____/А.В. Никонов/

Члены комиссии _____/В.Н. Цыганенко/

_____/О.Б. Малков/

ПРИЛОЖЕНИЕ Б. СВИДЕТЕЛЬСТВА О РЕГИСТРАЦИИ ПРОГРАММ ЭВМ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2021618977

Программный комплекс сбора данных о состоянии серверных станций

Правообладатель: ***Викулов Егор Олегович (RU)***

Автор(ы): ***Викулов Егор Олегович (RU)***

Заявка № **2021617727**
Дата поступления **25 мая 2021 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***02 июня 2021 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

ДОКУМЕНТ ПОДПИСАН ЭЛЕКТРОННОЙ ПОДПИСЬЮ
Сертификат 0x02A5CFBC00B1ACF59A40A2F08092E9A118
Владелец **Ивлиев Григорий Петрович**
Действителен с 15.01.2021 по 15.01.2035

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2022662254

Программный комплекс параллельных вычислений распределения нагрузки облачных серверных станций

Правообладатель: ***Викулов Егор Олегович (RU)***

Автор(ы): ***Викулов Егор Олегович (RU)***

Заявка № **2022660916**
Дата поступления **14 июня 2022 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***30 июня 2022 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

ДОКУМЕНТ ПОДПИСАН ЭЛЕКТРОННОЙ ПОДПИСЬЮ
Сертификат 68b80577e114e1090a94edbd24145d5c7
Владелец Зубов Юрий Сергеевич
Действителен с 20.12.2022 по 26.06.2023

Ю.С. Зубов

ПРИЛОЖЕНИЕ В. ПРОЦЕДУРА ПРОВЕРКИ И ПРОГРАММА ИСПЫТАНИЙ СИСТЕМЫ БАЛАНСИРОВКИ НАГРУЗКИ

Проведена верификация корректности работы функций системы распределения запросов пользователей клиент-серверных приложений по вычислительным ресурсам серверного комплекса. Для этого составлены процедуры проверки работы системы и представлены результаты сверки полученных результатов работы системы с результатами распределения нагрузки другими известными методами.

С помощью процедуры проверки и методики испытания программы проведена верификация разработанной системы балансировки нагрузки. Составлен список процедур проверки работы системы и представлены результаты сравнения с требованиями к системе.

Для верификации разрабатываемой системы балансировки нагрузки необходимо проводить проверку всех ее подпрограмм («Агент», «Балансир», «Веб-сервер»). В таблице В.1, представлен состав программного обеспечения системы балансировки нагрузки.

Таблица В.1 – Состав ПО системы балансировки нагрузки.

№	Наименование программы/ идентификатор	Тип файла	Выполняемые функции
1	Программа «Агент» Agent.js	js-файл сценарий	Сбор данных о состоянии серверов
2	Программа «Балансир» SetUserServer.php	php-файл интерпретируемый в машинный код	Выбор сервера для обработки запроса пользователя
3	Программа «Веб-сервер» nginx.conf	conf-файл конфигурации	Распределение запросов пользователей по серверам
4	Программа виртуализации	Docker-файл Конфигурации	Запуск системы и управление программными продуктами
5	Программа запуска контейнерных приложений	yml-файл конфигурации	Определение и запуска много контейнерных приложений
6	База данных Servers	Json-файл	БД параметров состояния серверов
7	База данных Users	Хранение данных в оперативной памяти	БД пользователей

В таблице 2 представлены процедуры верификации разработанных программ с требованиями к ожидаемым результатам и критериям оценки корректности отработки системы и ее компонентов.

Таблица 2 – Процедуры проверки и методика испытаний.

Процедура проверки	Методика испытаний	Критерии оценки / Ожидаемый результат
Проверка возможности сбора данных параметров серверов	а)Запустить систему выполнив команду *docker compose up* б)Организовать очередь запросов пользователей выполнив команду *locust -u=100 -r=1 -t=600 --host=http://localhost/ --autostart --autoquit=10*	Параметры состояния серверов сохраняются в БД и соответствуют состоянию сервера на момент записи
Проверка возможности выбора сервера для пользователя	а)Запустить систему выполнив команду *docker compose up* б)Организовать очередь запросов пользователей Выполнив команду *locust -u=100 -r=1 -t=600 --host=http://localhost/ --autostart --autoquit=10* в)с помощью окна командной строки контролировать выбор номера сервера, поставленного в соответствие пользователю	На протяжении 60 минут система отвечает на запросы 100 параллельно работающих пользователей, каждому из пользователей ставится в соответствие сервер $S_{1\text{-}3}$
Проверка работоспособности системы при оказании нагрузки	а)Запустить систему выполнив команду *docker compose up* б)Организовать очередь запросов пользователей Выполнив команду *locust -u=100 -r=1 -t=600 --host=http://localhost/ --autostart --autoquit=10* в)Контролировать среднее время выполнения запросов	Среднее время выполнения запросов алгоритмами «Clustering» и «Fuzzy Selector» меньше, чем у алгоритмов «Round Robin», «Weighted Round Robin», «Least Connections», «Destination Hash Scheduling», «Source Hash Scheduling», «Sticky Sessions»