Оглавление

[Введение 2](#_Toc195562723)

[Цель и задачи исследования 4](#_Toc195562724)

[Научная новизна 6](#_Toc195562725)

[Практическая значимость 6](#_Toc195562726)

[Объект и предмет исследования 7](#_Toc195562727)

[Методы исследования 7](#_Toc195562728)

[Структура работы 8](#_Toc195562729)

[Глава 1. Анализ предметной области и существующих подходов 9](#_Toc195562730)

[1.1. Событийно-непрерывные модели: определение и область применения 9](#_Toc195562731)

[1.2. Обзор существующих методик представления и обработки 10](#_Toc195562732)

[1.3. Анализ ограничений традиционных подходов 10](#_Toc195562733)

[1.4. Сравнительная характеристика известных программных решений 10](#_Toc195562734)

[1.5. gRPC как инструмент взаимодействия в распределённой системе 10](#_Toc195562735)

[1.6. Постановка задачи разработки унифицированного представления 11](#_Toc195562736)

[Глава 2. Проектирование унифицированного представления событийно-непрерывных моделей 11](#_Toc195562737)

[2.1. Общая схема архитектуры системы 11](#_Toc195562738)

[2.2. Основные сущности системы 12](#_Toc195562739)

[2.3. Разработка структуры хранения и связей 13](#_Toc195562740)

[2.4. Формирование требований к унифицированному представлению 14](#_Toc195562741)

[2.5. Логическая схема данных и описание типов данных 16](#_Toc195562742)

[2.6. Проектирование интерфейсов взаимодействия и обработки данных 17](#_Toc195562743)

[2.7. Разработка протоколов взаимодействия на базе gRPC 18](#_Toc195562744)

[Глава 3. Реализация программного комплекса 19](#_Toc195562745)

[3.1. Средства разработки и используемые технологии 19](#_Toc195562746)

[3.2. Реализация клиентского интерфейса 19](#_Toc195562747)

[3.3. Реализация сервера-слушателя 20](#_Toc195562748)

[3.4. Реализация сервера-вычислителя 21](#_Toc195562749)

[3.5. Логика обмена сообщениями между компонентами через gRPC 22](#_Toc195562750)

[Глава 4. Исследование работы и тестирование 23](#_Toc195562751)

[4.1. Подходы к тестированию распределённых вычислительных систем 23](#_Toc195562752)

[4.2. Разработка и проведение тестов клиент-серверного взаимодействия 24](#_Toc195562753)

[4.3. Проверка корректности работы вычислителя на тестовых данных 24](#_Toc195562754)

[4.4. Оценка производительности и масштабируемости системы 24](#_Toc195562755)

[4.5. Сравнительный анализ с монолитными и другими распределёнными архитектурами. 24](#_Toc195562756)

[4.6. Анализ устойчивости к сбоям и нагрузочным ситуациям 25](#_Toc195562757)

[Заключение 25](#_Toc195562758)

[Список литературы 25](#_Toc195562759)

# Введение

Актуальность темы

В современных условиях развития программных и вычислительных технологий наблюдается устойчивый рост интереса к построению гибких и масштабируемых вычислительных систем. Одним из перспективных направлений является моделирование событийно-непрерывных процессов, находящих применение в различных областях: от промышленного производства и телекоммуникаций до научных исследований. Эти модели позволяют учитывать как дискретные, так и непрерывные изменения, что особенно важно при решении сложных задач оптимизации и анализа динамических систем.

Современные вычислительные системы требуют перехода от монолитных архитектур к распределённым решениям, способным эффективно масштабировать вычислительные ресурсы и оптимизировать обработку больших объёмов данных. Разработка удобного, надёжного и унифицированного представления событийно-непрерывных моделей становится ключевым элементом в создании таких систем, поскольку позволяет обеспечить совместимость между различными вычислительными платформами и ускорить процесс интеграции новых компонентов.

Использование протоколов удалённого вызова процедур, таких как gRPC, открывает широкие возможности для построения распределённых систем. gRPC обеспечивает стандартизированный и эффективный обмен данными между компонентами системы, снижая сетевые задержки и повышая отказоустойчивость. Такие возможности особенно актуальны в условиях роста требований к производительности и надежности систем, а также при необходимости оперативного масштабирования вычислительных мощностей.

Современные тенденции в области контейнеризации, например с использованием Docker, позволяют создавать однородные и управляемые среды выполнения, что значительно упрощает развертывание и эксплуатацию вычислительных комплексов. Это, в свою очередь, способствует более быстрому внедрению инновационных решений и адаптации системы к изменяющимся условиям эксплуатации.

Таким образом, актуальность темы обусловлена необходимостью разработки унифицированного представления событийно-непрерывных моделей для распределённых вычислительных систем, способных удовлетворить потребности современной экономики и науки. Создание такого представления является стратегически важным шагом в направлении повышения эффективности моделирования сложных систем, улучшения их масштабируемости и обеспечения высокой производительности вычислений в условиях постоянно растущей нагрузки и объёмов обрабатываемых данных.

## Цель и задачи исследования

* Целью данной работы является разработка унифицированного представления событийно-непрерывных моделей и исследование особенностей его реализации в распределённой системе с использованием gRPC. Эта цель обусловлена необходимостью создания эффективного инструмента, который позволит объединить дискретные и непрерывные процессы в единой схеме, что особенно актуально для моделирования сложных систем, применяемых в промышленности, телекоммуникациях и научных исследованиях. Разработка такого представления позволит снизить издержки на разработку, оптимизировать использование вычислительных ресурсов и обеспечить возможность масштабирования вычислительных систем, что становится критически важным в условиях растущих объёмов данных и увеличения требований к производительности.
* Для достижения поставленной цели необходимо решить следующие задачи:
* **Провести анализ существующих подходов к представлению событийно-непрерывных моделей:**  
  Изучить литературу и современные программные решения, такие как Modelica, MATLAB Simulink, AnyLogic и другие, чтобы понять, каким образом реализуются модели, описывающие гибридные системы. Внимание уделить различным методам описания дискретных и непрерывных процессов, а также проанализировать преимущества и ограничения каждого из них.
* **Выявить ограничения традиционных решений:**  
  Оценить существующие методы с точки зрения их гибкости, адаптивности и возможности интеграции в распределённые вычислительные системы. Особое внимание уделить проблемам масштабируемости, сложности интеграции с внешними сервисами и необходимости создания специализированных инструментальных средств для работы с гибридными моделями.
* **Сформулировать требования к унифицированному представлению:**  
  На основании проведённого анализа сформулировать требования, которым должно удовлетворять предлагаемое представление. Требования должны включать поддержку различных типов данных, возможность динамического обновления модели, совместимость с различными вычислительными платформами, а также обеспечение корректного обмена данными в распределённой системе.
* **Спроектировать архитектуру вычислительного комплекса:**  
  Разработать архитектуру, состоящую из трёх ключевых компонентов: удалённого клиентского интерфейса, сервера-слушателя и сервера-вычислителя. Особое внимание уделить организации взаимодействия между компонентами с использованием протоколов удалённого вызова процедур (gRPC), что позволит обеспечить эффективное и масштабируемое распределение вычислительной нагрузки.
* **Реализовать прототип комплекса и провести его тестирование:**  
  Создать рабочий прототип, реализующий предложенную архитектуру, с акцентом на корректность работы всех ключевых модулей. Прототип должен включать механизмы формирования и передачи моделей в формате JSON, обработку вычислительной логики и динамическое управление вычислительными ресурсами. Провести функциональное и нагрузочное тестирование, чтобы оценить корректность работы системы в различных условиях.
* **Оценить производительность и устойчивость системы:**  
  Провести комплексное исследование, направленное на оценку производительности вычислительного комплекса при различных параметрах нагрузки, а также его устойчивости к ошибкам, сбоям и отказам компонентов. Необходимо изучить возможность масштабирования системы и её адаптацию к изменяющимся требованиям вычислительной среды, а также провести сравнительный анализ с традиционными монолитными решениями.
* Таким образом, поставленные задачи направлены на создание полноценного инструмента для моделирования гибридных систем, который сможет удовлетворить современные требования к вычислительной инфраструктуре, обеспечивая высокую эффективность, гибкость и надёжность при обработке сложных событийно-непрерывных моделей.

## Научная новизна

Научная новизна работы заключается в формировании унифицированного подхода к представлению событийно-непрерывных моделей, что позволяет объединить описание дискретных событий и непрерывных процессов в единой схеме. Такой подход реализуется посредством распределённой архитектуры, в которой используются динамически создаваемые серверы-вычислители. Это позволяет не только оптимизировать обработку вычислительных задач, но и обеспечить масштабируемость системы, что особенно актуально при росте объёмов данных и увеличении требований к быстродействию. Работа демонстрирует, как современные технологии, такие как gRPC и контейнеризация (например, Docker), могут быть использованы для повышения эффективности вычислительного моделирования гибридных систем.

## Практическая значимость

Практическая значимость работы обусловлена разработкой прототипа программного комплекса, способного обеспечить эффективное моделирование событийно-непрерывных систем в условиях распределённых вычислительных сред. Созданное решение может быть использовано в системах вычислительного моделирования, лабораторных стендах и корпоративных инфраструктурах для оптимизации процессов анализа и прогнозирования. Разработанный комплекс позволяет упростить описание моделей, что снижает время и затраты на их разработку и тестирование, а также обеспечивает гибкость архитектуры при интеграции с внешними сервисами. В условиях динамичного развития вычислительных технологий такие решения становятся особенно востребованными для реализации высоконагруженных распределённых систем.

## Объект и предмет исследования

Объектом исследования являются событийно-непрерывные модели, которые представляют собой описание сложных систем, объединяющих дискретные и непрерывные процессы. Такие модели используются для анализа динамики систем в различных областях, начиная от промышленного производства и заканчивая научными исследованиями.

Предметом исследования являются методы представления событийно-непрерывных моделей и архитектурные решения для их обработки. Исследование включает разработку подходов к формализации моделей, определению структуры данных, а также построению распределённой архитектуры, обеспечивающей эффективную обработку и масштабирование вычислительных процессов. Особое внимание уделяется вопросам интеграции различных технологий, таких как gRPC для обмена сообщениями между компонентами системы, и механизмам динамического создания серверов-вычислителей для распределённого выполнения вычислительных задач.

## Методы исследования

В работе использовались комплексные методы, позволяющие охватить все этапы разработки и анализа предлагаемого решения. Среди них – методы теоретического анализа, позволяющие изучить существующие подходы к моделированию событийно-непрерывных процессов и выявить их сильные и слабые стороны. Кроме того, применялись методы проектирования архитектурных решений, что включало детальное моделирование распределённой системы с использованием gRPC для организации взаимодействия между компонентами. Программная реализация проводилась с применением современных средств разработки на языке C и C#, а также инструментов контейнеризации, таких как Docker, что обеспечивало воспроизводимость среды выполнения. Методы тестирования включали функциональное, нагрузочное и сравнительное тестирование, что позволило оценить производительность, отказоустойчивость и масштабируемость системы. Сравнительный анализ проводился с целью выявления преимуществ предлагаемого подхода по сравнению с традиционными монолитными решениями и другими распределёнными архитектурами.

## Структура работы

Работа состоит из введения, четырёх глав, заключения и списка использованных источников. Во введении описаны актуальность темы, цель и задачи исследования, научная новизна, практическая значимость, объект и предмет исследования, а также методы исследования и общая структура работы.

Первая глава посвящена анализу предметной области и существующих подходов к моделированию событийно-непрерывных систем. В ней рассматриваются современные тенденции в области вычислительных технологий, описываются различные методы моделирования гибридных систем, проводится сравнительный анализ программных решений, таких как Modelica, MATLAB Simulink, AnyLogic и других, а также анализируются проблемы монолитных архитектур и преимущества перехода к распределённым системам.

Вторая глава содержит описание проектирования унифицированного представления событийно-непрерывных моделей и архитектуры системы. Здесь детально рассматриваются требования к представлению моделей, логическая схема данных, структура хранения и связи между элементами, а также принципы формирования единого стандарта для описания дискретных и непрерывных процессов. В этой главе отражены подходы к формированию требований, проектированию интерфейсов взаимодействия и выбору форматов обмена данными.

Третья глава посвящена реализации трёхкомпонентной архитектуры, включающей клиентский интерфейс, сервер-слушатель и сервер-вычислитель. В ней приводится подробное описание каждого из компонентов, их функциональных возможностей, механизмов обмена данными посредством gRPC, а также процессов запуска и управления вычислительными узлами. Описаны детали реализации клиентской части на основе MVVM, а также особенности работы серверов на языках C и C#, включая обработку JSON-моделей, выполнение вычислительных циклов, обработку ошибок и освобождение памяти.

Четвёртая глава содержит результаты тестирования и анализа работы разработанного прототипа. В ней приводятся методики и результаты функционального, нагрузочного и сравнительного тестирования, анализируется производительность системы, устойчивость к сбоям и возможности масштабирования. Рассмотрены различные сценарии использования, проанализированы сильные и слабые стороны предложенной архитектуры, а также даны рекомендации по дальнейшему развитию системы.

# Глава 1. Анализ предметной области и существующих подходов

## 1.1. Событийно-непрерывные модели: определение и область применения

Событийно-непрерывные модели (СНМ) — это гибридные модели, предназначенные для описания сложных систем, в которых поведение определяется сочетанием непрерывных процессов и дискретных событий. Их особенность в том, что они объединяют два типа временной эволюции: плавное изменение параметров (например, температуры, давления, скорости) и резкие изменения состояния системы в моменты наступления определённых условий — событий (например, срабатывание датчика, отказ оборудования или переключение режима) (№7), (№8), (№16).

Непрерывный процесс — это изменение параметров во времени, описываемое, как правило, дифференциальными уравнениями. В отличие от него, дискретное событие — это мгновенный переход системы из одного состояния в другое при выполнении заданных условий. СНМ позволяют описывать такие системы целостно, с учётом взаимодействия обеих форм динамики, что делает их особенно полезными при моделировании объектов, в которых непрерывное и дискретное поведение неразрывно связаны (№13), (№14).

Теоретическая основа СНМ строится на объединении средств описания непрерывных и дискретных процессов. Непрерывная динамика моделируется с помощью дифференциальных уравнений, описывающих эволюцию переменных между событиями. Дискретная часть формализуется логикой переходов: при выполнении условий происходит мгновенное изменение состояния системы. Такой подход относится к классу гибридного моделирования, сочетающего методы систем управления и дискретной математики. Типичный пример — системы с переключениями, где в зависимости от текущего состояния и входных воздействий система переходит между режимами функционирования (№14), (№16), (№17).

СНМ находят широкое применение в различных прикладных областях. В промышленности они используются для моделирования автоматизированных производственных линий, где, например, непрерывный поток материалов сочетается с дискретными операциями — сортировкой, контролем качества, упаковкой. Характерным примером является моделирование процессов в кабельной промышленности (№15).

В телекоммуникационных и сетевых системах СНМ применяются для анализа трафика, оценки устойчивости сетей, моделирования переключений маршрутизаторов. Здесь непрерывная часть описывает поток данных, а дискретная — события вроде отказов или изменений маршрута (№8), (№10).

В киберфизических системах, объединяющих физические объекты с цифровыми платформами, СНМ позволяют моделировать взаимодействие сенсоров, актуаторов и программных компонентов. Это особенно актуально для «умных» зданий, автономного транспорта, робототехники и систем интернета вещей (№12), (№18), (№20).

СНМ также применяются в социально-экономических и управленческих системах. Здесь, например, динамика спроса и предложения моделируется как непрерывный процесс, а решения органов управления — как дискретные события, влияющие на состояние системы. Такие подходы используются для анализа устойчивости и оптимизации развития территорий (№5), (№12).

Значимость СНМ подтверждается их включением в перечень приоритетных научно-технологических направлений, утверждённых на государственном уровне (№3). Гибридное моделирование и интеллектуальные системы входят в состав критических технологий, определённых как ключевые для развития экономики и безопасности страны (№4), (№6), (№17). СНМ становятся основой для создания цифровых двойников, интеллектуального управления и адаптивных систем на базе нейросетей и нечеткой логики (№18), (№19).

Таким образом, событийно-непрерывные модели представляют собой мощный инструмент описания и анализа сложных систем. Благодаря способности учитывать оба типа динамики, они позволяют достоверно моделировать поведение реальных объектов в различных сферах — от производства и связи до киберфизических и экономических систем. Это делает СНМ не только актуальным, но и необходимым элементом современного научного и инженерного инструментария.

## 1.2. Обзор существующих методик представления и обработки

Моделирование сложных нелинейных систем (СНМ) основывается на формализмах, способных отразить как непрерывную, так и дискретную динамику. Один из таких подходов — гибридные автоматы. Это структуры, объединяющие свойства конечных автоматов и систем дифференциальных уравнений: в каждом состоянии система описывается непрерывной динамикой, а переходы между состояниями происходят при выполнении заданных логических условий. Гибридные автоматы применяются в задачах анализа безопасности и верификации, а также при моделировании технических и социотехнических процессов (№16), (№17), (№18), (№19), (№20).

Другой широко используемый формализм — системы дифференциальных уравнений с переключениями. Они предполагают наличие нескольких наборов уравнений, активируемых в зависимости от условий. Такой подход применяется в энергетике, биомедицине и других областях, где необходимо учитывать смену режимов функционирования систем. В отличие от гибридных автоматов, он не требует явного задания логики переходов, однако менее прозрачен с точки зрения формального анализа (№12), (№14), (№17).

Инженерное моделирование СНМ часто реализуется с помощью специализированных языков и сред. Simulink — визуальная платформа блочного моделирования, популярная в промышленной среде благодаря интеграции с MATLAB. Она удобна для быстрой сборки моделей, но ограничена в плане гибкости и описания гибридных процессов. Modelica — объектно-ориентированный язык, основанный на декларативном подходе. Он позволяет описывать системы уравнений без указания порядка их решения, что особенно полезно при моделировании многофизичных процессов. Modelica активно используется в сочетании с такими инструментами, как OpenModelica, Dymola и OMSimulator (№7), (№14), (№15), (№16).

Для анализа и обработки моделей применяются численные методы — от классических алгоритмов интегрирования до стохастического моделирования и символьного анализа. Особое внимание уделяется устойчивости решений, точности обработки событий и минимизации численных ошибок (№7), (№13), (№16).

Среди инструментов симуляции СНМ следует выделить несколько направлений. Simulink обеспечивает быструю разработку прикладных моделей и широко применяется в системах управления. Языки Modelica и соответствующие симуляторы (Dymola, OMSimulator) демонстрируют высокую масштабируемость и подходят для сложных инженерных задач. Специализированные пакеты, такие как HyTech и SpaceEx, ориентированы на строгий формальный анализ гибридных автоматов. AnyLogic отличается универсальностью, позволяя объединять дискретно-событийные, системно-динамические и агентные подходы, что делает его эффективным для моделирования распределённых социотехнических систем (№8), (№9), (№10), (№14), (№15), (№17).

Сравнение подходов показывает, что Simulink выигрывает в простоте освоения и скорости построения моделей, но уступает в гибкости и масштабируемости. Modelica обеспечивает высокую производительность при моделировании комплексных систем, особенно с многофизичными компонентами. Гибридные автоматы предоставляют строгую математическую основу и применимы там, где необходима верификация поведения. AnyLogic подходит для междисциплинарных задач, но ограничен в работе с непрерывной динамикой.

Выбор подхода определяется характером задачи: для формальной верификации предпочтительны гибридные автоматы, для моделирования реальных технических систем — Modelica, для быстрой инженерной отладки — Simulink, для анализа сложных распределённых систем — AnyLogic. С учётом стратегических направлений научно-технологического развития, таких как цифровое проектирование и управление сложными объектами (№2), (№3), (№4), (№6), наибольший потенциал имеет развитие гибридных подходов и создание универсальных симуляторов нового поколения.

## 1.3. Анализ ограничений традиционных подходов

Современные задачи моделирования социально-неоднородных моделей (СНМ) требуют высокой гибкости, масштабируемости и совместимости используемых инструментов. Однако традиционные подходы зачастую оказываются неэффективными в условиях изменяющихся требований. Одним из основных ограничений выступает низкая адаптивность: большинство моделей опирается на жёсткую архитектуру, не предполагающую оперативного обновления или расширения, что затрудняет масштабирование и модификацию.

Существенным препятствием остаётся отсутствие единых стандартов описания моделей и форматов обмена данными, что затрудняет совместимость различных платформ и систем, особенно в распределённых проектах (11). Это приводит к фрагментации программных решений, снижает повторное использование компонентов и усложняет переносимость моделей между средами.

Отдельной проблемой является ограниченная интеграция с внешними цифровыми сервисами. В условиях развития облачных вычислений и интеллектуальных платформ такая замкнутость снижает эффективность моделирующих систем (8), (15). Особенно это актуально при необходимости взаимодействия с распределёнными вычислительными средами, где важны синхронизация процессов и устойчивость передачи данных. Традиционные архитектуры, ориентированные на централизованное исполнение, как правило, не поддерживают масштабируемую организацию распределённых вычислений (5), (10).

С вычислательной точки зрения традиционные подходы демонстрируют слабую производительность при росте объёмов данных и числа взаимодействующих компонентов. Это выражается в снижении точности результатов, увеличении времени отклика и затруднённой верификации. Отсутствие механизмов параллельной и гибридной обработки делает такие методы неэффективными при решении задач высокой сложности (17), (20).

Сравнительный анализ ограничений позволяет выделить три группы: технические, архитектурные и вычислительные. К техническим относятся отсутствие унифицированных стандартов и слабая интеграция с внешними сервисами. Архитектурные связаны с жёсткой структурой и низкой адаптивностью систем. Вычислительные ограничения выражаются в неспособности обрабатывать большие объёмы данных и организовывать параллельные процессы.

Таким образом, традиционные подходы к моделированию СНМ характеризуются системными недостатками, ограничивающими их применимость в современных условиях. Преодоление указанных барьеров требует перехода к более гибким, масштабируемым и стандартизированным моделирующим платформам, способным эффективно функционировать в распределённых цифровых средах (2), (4), (12).

## 1.4. Сравнительная характеристика известных программных решений

В ходе анализа были рассмотрены программные среды Modelica, MATLAB Simulink, AnyLogic, а также ряд дополнительных решений, таких как OpenModelica и Scilab/Xcos. Эти инструменты широко применяются для моделирования сложных технических и социально-экономических систем, обладают развитым функционалом, однако не всегда обеспечивают достаточную гибкость для интеграции в распределённые вычислительные среды и зачастую предъявляют высокие требования к ресурсам.

MATLAB Simulink — одна из наиболее распространённых платформ для моделирования непрерывных и дискретных процессов. Она предлагает визуальный интерфейс, обширную библиотеку блоков и тесную интеграцию с экосистемой MATLAB. Благодаря этим возможностям Simulink активно используется в инженерной практике, в том числе для задач автоматизации и разработки встроенных систем, включая автокодогенерацию (№7), (№13). Тем не менее, расширение функционала в сторону распределённых вычислений требует дополнительных решений, таких как Simulink Real-Time или MATLAB Production Server, что усложняет реализацию и увеличивает затраты (№17).

Modelica реализует декларативный подход, позволяя описывать поведение систем через физические уравнения. Это особенно эффективно в мультидисциплинарных задачах, где необходимо объединять различные физические домены — от механики до термодинамики и электротехники (№14), (№16). Среди реализаций Modelica наибольшее распространение получили Dymola и OpenModelica. Первая обеспечивает высокий уровень профессиональной поддержки, но требует лицензии; вторая — свободно распространяемая, однако уступает в удобстве и производительности. Поддержка интеграции с распределёнными системами ограничена: хотя стандарт FMI позволяет обмениваться моделями между платформами, полноценная работа в кластерной среде реализована частично (№11), (№17).

AnyLogic — гибридная платформа, сочетающая дискретно-событийное, системно-динамическое и агентное моделирование. Она особенно эффективна при моделировании социально-экономических процессов, логистических цепочек и человеко-ориентированных систем (№12), (№15). Преимущество AnyLogic — в её ориентированности на Java, что обеспечивает гибкость и масштабируемость, а также встроенные средства для облачного исполнения моделей. Это даёт ей серьёзное преимущество в задачах распределённого моделирования. Вместе с тем, высокая сложность разработки и необходимость владения языком Java могут ограничивать доступность платформы для менее подготовленных пользователей (№18), (№19).

Сравнительный анализ позволяет выделить ключевые особенности каждой среды. Simulink предлагает высокую инженерную зрелость и визуальное удобство, но требует дополнительных средств для интеграции в распределённые решения. Modelica демонстрирует мощные возможности физического моделирования, но слабо масштабируется и зависит от конкретной реализации. AnyLogic, напротив, хорошо подходит для гибридного и многопользовательского моделирования, включая задачи с элементами социально-нечёткой логики (№5), (№10), (№18).

Таким образом, выбор программной платформы должен основываться не только на её функциональности, но и на способности интегрироваться в распределённую архитектуру, а также соответствовать специфике предметной области. В ряде случаев наиболее целесообразным оказывается комбинированное использование нескольких решений — особенно при разработке распределённых интеллектуальных систем, в которых важны гибкость, масштабируемость и поддержка разнообразных моделей поведения (№17), (№20).

## 1.5. gRPC как инструмент взаимодействия в распределённой системе

gRPC — это современный фреймворк удалённого вызова процедур (Remote Procedure Call, RPC), разработанный компанией Google для создания высокоэффективных распределённых систем. Он основан на использовании протокола HTTP/2 и системы сериализации Protocol Buffers, что делает его особенно подходящим для организации межсервисного взаимодействия в условиях высокой нагрузки и жёстких требований к производительности. Архитектура gRPC ориентирована на строго типизированное и структурированное взаимодействие между компонентами, обеспечивая устойчивую работу как в локальных, так и в облачных и гибридных вычислительных средах (№1).

Основу архитектуры gRPC составляет модель «клиент — сервер», в которой клиент вызывает удалённые процедуры на стороне сервера, используя синтаксис, аналогичный вызову локальных функций. Обмен сообщениями осуществляется по протоколу HTTP/2, обеспечивающему постоянное соединение, мультиплексирование запросов и эффективную работу с потоками данных. Это позволяет минимизировать задержки при передаче и использовать двунаправленную потоковую передачу данных, что особенно актуально в задачах мониторинга, телеметрии и управления в реальном времени (№2).

Протокол Protocol Buffers, также разработанный Google, используется в gRPC в качестве механизма сериализации сообщений. Его бинарный формат обеспечивает компактность и высокую скорость обработки данных по сравнению с текстовыми форматами вроде JSON и XML. Структура сообщений описывается в специальных .proto-файлах, на основе которых автоматически генерируется код на поддерживаемых языках программирования. Это обеспечивает строгую типизацию, ускоряет разработку и исключает расхождения между интерфейсами взаимодействующих компонентов (№3).

gRPC получил широкое распространение в микросервисной архитектуре. Так, в системах управления заказами он используется для взаимодействия между модулями аутентификации, расчёта стоимости и логистики, позволяя достичь высокой согласованности и масштабируемости. В облачных платформах, например Kubernetes, gRPC обеспечивает эффективную маршрутизацию и балансировку нагрузки между контейнерами. Кроме того, он активно применяется в системах интернета вещей (IoT) — например, при передаче телеметрических данных от распределённых сенсоров к серверам обработки, где важны низкие задержки и высокая пропускная способность (№4).

Таким образом, gRPC представляет собой универсальное средство построения надёжных и производительных распределённых систем. Его архитектура обеспечивает высокую эффективность взаимодействия, строгую типизацию данных и широкие возможности масштабирования. Применение gRPC соответствует приоритетным направлениям научно-технологического развития Российской Федерации, включая цифровую трансформацию, унификацию программных решений и разработку критически значимой информационной инфраструктуры (№2), (№3), (№5).

## 1.6. Постановка задачи разработки унифицированного представления

Целью настоящей работы является разработка унифицированного представления событийно-непрерывных моделей (СНМ) — формата, обеспечивающего однозначное описание моделей, сочетающих непрерывную динамику и дискретные события. Задача актуальна в условиях растущей сложности моделируемых систем и необходимости интеграции различных подходов в рамках единой вычислительной архитектуры. Унифицированное представление призвано обеспечить совместимость, расширяемость и воспроизводимость моделей в распределённых средах, применяемых, в частности, для имитационного и гибридного моделирования технических и социально-экономических систем (№5), (№12), (№15).

Предлагаемое решение базируется на архитектуре из трёх компонентов: удалённого клиентского интерфейса, сервера-слушателя, принимающего запросы по протоколу gRPC, и сервера-вычислителя, исполняющего модели и поддерживающего обратную связь с клиентом. Такая архитектура отвечает требованиям распределённых вычислений, включая отказоустойчивость, масштабируемость и совместимость между компонентами.

Унифицированное представление должно удовлетворять ряду требований. Во-первых, оно должно охватывать гибридную природу СНМ: включать описание как непрерывных процессов, так и дискретных событий (№7), (№8), (№14). Во-вторых, необходима формальная однозначность структуры, обеспечивающая её интерпретацию без дополнительных преобразований (№11). В-третьих, представление должно быть модульным и нейтральным по отношению к платформе, что позволяет включать его в разные программные контексты. Кроме того, необходима совместимость с современными протоколами взаимодействия, такими как gRPC.

Основная научно-техническая сложность заключается в объединении разнородных типов моделей в единый формат, пригодный как для вычислений, так и для передачи между компонентами. Требуется формализовать структуру представления, определить его функциональные и интерфейсные требования, а также описать взаимодействие компонентов в условиях распределённой среды. Задача усложняется необходимостью соответствия требованиям надёжности, интероперабельности и масштабируемости, обозначенным в стратегических документах по приоритетным направлениям развития науки и технологий Российской Федерации (№2), (№3), (№4).

В результате в работе необходимо: обосновать выбор архитектурного подхода; определить формальные требования к унифицированному представлению; описать структуру взаимодействия компонентов через gRPC; и разработать прототип, демонстрирующий функциональность системы.

Таким образом, данный раздел завершает введение в предметную область, формулируя конкретную научно-техническую задачу, лежащую в основе дальнейших проектных решений. В следующих главах будут рассмотрены существующие подходы к гибридному моделированию, обоснован выбор архитектуры, предложен формализованный формат представления и проведена оценка его применимости.

# Глава 2. Проектирование унифицированного представления событийно-непрерывных моделей

## 2.1. Общая схема архитектуры системы

Архитектура разрабатываемой системы моделирования событийно-непрерывных моделей (СНМ) основана на трёх функционально независимых, но взаимодействующих компонентах: клиентском интерфейсе, сервере-слушателе и сервере-вычислителе. Такая структура направлена на обеспечение масштабируемости, гибкости и высокой производительности при обработке гибридных моделей в распределённой среде.

Клиентский интерфейс представляет собой точку взаимодействия пользователя с системой. Он реализован в виде веб-приложения и программного интерфейса (API), через которые осуществляется передача параметров моделирования, запуск вычислений и получение результатов. Поддержка унифицированного протокола взаимодействия (gRPC) обеспечивает совместимость интерфейса с распределённой вычислительной архитектурой, а также возможность асинхронной обработки запросов (№13), (№11).

Сервер-слушатель выполняет функции диспетчеризации и управления ресурсами. Он принимает запросы от клиентского интерфейса, оценивает текущую загрузку вычислительных узлов и направляет задачи на серверы-вычислители, функционируя в режиме балансировщика нагрузки. Ключевой особенностью является поддержка динамического масштабирования: сервер-слушатель способен инициировать запуск новых вычислительных узлов по мере роста нагрузки, что позволяет системе адаптироваться к изменяющимся требованиям в режиме реального времени (№17), (№20).

Сервер-вычислитель реализует основную вычислительную логику обработки СНМ. Он выполняет параллельные вычисления по описанию модели, включая решение систем дифференциальных уравнений для непрерывной части и обработку дискретных событий. Использование специализированной библиотеки гибридного моделирования позволяет эффективно распределять нагрузку между потоками и обеспечивать точность при расчётах.

Пользовательский интерфейс взаимодейсвует с отслаьными компоненатами средствами протокола gRPC, в это время слушатель и вычислитель общаются средствами ОС в виде именованных каналов и дополнительных файлов. Модульная структура архитектуры упрощает масштабирование и сопровождаемость системы, а также способствует её расширению без модификации базовых компонентов.

Научная новизна разработанной архитектуры заключается в интеграции гибридных вычислительных стратегий с механизмами динамического управления ресурсами в распределённой среде. Такой подход обеспечивает не только высокую производительность при обработке СНМ, но и устойчивость архитектуры к изменениям нагрузки и сложности моделируемых процессов (№12), (№16), (№19). В отличие от традиционных решений с фиксированной структурой, данная архитектура допускает гибкое перенастраивание в зависимости от характера задачи.

Таким образом, предложенная трёхкомпонентная схема обеспечивает функциональное разделение ответственности между уровнями системы, способствует эффективному выполнению моделирования и адаптируется к различным сценариям использования.

## 2.2. Основные сущности системы

Центральной сущностью при построении гибридной модели является формализованное представление исследуемой системы, которое описывает как дискретные события, так и непрерывные процессы. Такая модель фиксирует структуру системы, динамику её компонентов и реакции на внешние воздействия. Унифицированный подход к описанию поведения позволяет интегрировать разнородные процессы в единую схему и облегчает перенос модели между различными вычислительными средами, включая распределённые архитектуры (№11), (№20).

Состояние модели определяется совокупностью параметров, отражающих конфигурацию системы в заданный момент времени. Оно включает значения как дискретных переменных (например, логических индикаторов состояния), так и непрерывных величин (таких как температура, уровень жидкости или давление). Состояние является ключевым носителем информации в процессе моделирования, поскольку на его основе рассчитывается дальнейшее поведение системы. Кроме того, оно служит основой для синхронизации компонентов в распределённой среде моделирования, обеспечивая согласованное развитие процесса в различных частях модели (№7), (№20).

Переменные представляют собой изменяющиеся во времени величины, которые определяют динамику системы. В зависимости от характера изменения они классифицируются на дискретные и непрерывные. Дискретные переменные моделируют мгновенные переходы, например переключения режимов или активацию управляющих сигналов. Непрерывные переменные описывают процессы, изменяющиеся во времени плавно, такие как изменение температуры, скорости или других физических параметров (№7), (№8). Существенное значение имеет поддержка гибридных переменных, объединяющих дискретную и непрерывную природу. Это особенно важно для корректного описания систем, в которых дискретные управляющие воздействия влияют на непрерывные процессы — как, например, в системах терморегуляции, где активация нагревателя приводит к изменению температуры среды (№12), (№14), (№17).

Формализация переменных осуществляется с использованием различных математических средств. Для дискретных переменных применяются логические правила переходов и событийные триггеры, для непрерывных — дифференциальные уравнения, аппроксимационные зависимости или табличные функции. В гибридных моделях эти механизмы объединяются в рамках общего описания, что позволяет учесть взаимодействие процессов различной природы. Такая структура обеспечивает согласованное обновление состояния модели, включая реакцию на события и непрерывную эволюцию параметров, что особенно критично в условиях распределённого моделирования (№16), (№18).

Таким образом, модель, состояние и переменные образуют согласованную систему, отражающую как внутреннюю динамику объекта, так и его реакцию на внешние воздействия. Формализованное представление этих сущностей обеспечивает точность моделирования, совместимость с различными архитектурами исполнения и расширяемость решений от локальных симуляций до распределённых систем (№2), (№3), (№11).

## 2.3. Разработка структуры хранения и связей

Проектирование структуры хранения данных является ключевым этапом архитектуры системы, обеспечивающим целостность, согласованность и высокую производительность. Особое внимание уделяется как логической модели данных, определяющей сущности и их взаимосвязи, так и физической реализации, направленной на оптимизацию доступа и минимизацию издержек.

Логическая модель описывает основные сущности системы. Ключевой единицей является переменная, характеризуемая уникальным в пределах пространства именем, типом (логическим или числовым с плавающей точкой), значением и флагом константности. Пространства переменных делятся на глобальные и локальные. Для каждой вычислительной ситуации создаётся отдельное локальное пространство, имеющее приоритет при поиске переменных. Это повышает эффективность доступа и не увеличивает объём потребляемой памяти, поскольку локально хранятся лишь активные в конкретный момент переменные.

Физическая организация реализуется в оперативной памяти, что исключает необходимость в промежуточных уровнях обращения, таких как СУБД или API-интерфейсы. Это позволяет минимизировать задержки и обеспечить прямой доступ к значениям из любых компонентов системы. В качестве модели хранения применяется структура «ключ–значение», где ключом выступает имя переменной, а значением — объект с соответствующими атрибутами. Индексация осуществляется средствами выбранной структуры, преимущественно хеш-таблицами или сбалансированными деревьями, что обеспечивает высокую скорость доступа при минимальном потреблении ресурсов (№12), (№18).

Масштабируемость достигается благодаря минимизации избыточных данных и проектированию расширяемой схемы хранения. Исключение служебных записей и использование компактного представления переменных позволяют эффективно использовать оперативную память. Архитектура поддерживает динамическое добавление новых типов сущностей без модификации существующих компонентов (№5), (№12). Ссылочная целостность поддерживается централизованным реестром, обеспечивающим уникальность имён и корректность связей. Типовая совместимость реализуется строгой типизацией и валидацией при записи. Механизмы многопоточности включают атомарные операции и, при необходимости, транзакционные конструкции.

Система хранения спроектирована с возможностью эволюции. Расширяемая структура метаданных и поддержка одновременного существования переменных разных версий позволяют изменять типы, связи и структуру без остановки системы. Это особенно важно в условиях исследовательских и имитационных сред, где схема данных может регулярно изменяться (№16), (№20). Такой подход обеспечивает адаптивность архитектуры и её устойчивость к изменениям, гарантируя непрерывность вычислительных процессов и совместимость между компонентами.

## 2.4. Формирование требований к унифицированному представлению

Формирование требований к унифицированному представлению событийно-непрерывных моделей основывается на необходимости обеспечения совместимости моделей с различными вычислительными платформами, их масштабируемости, гибкости описания и корректной обработки данных. Требования классифицируются на функциональные, технические, архитектурные и интерфейсные. Каждая группа требований непосредственно связана с задачами разработки системы и служит основой для последующих этапов проектирования.

Функциональные требования определяют ключевые характеристики представляемых данных. В их числе — корректное отображение состояний, формул, условий и переходов в формате JSON. Требуется гарантировать, что при десериализации с использованием библиотеки cJSON структура данных преобразуется в дерево объектов, полностью соответствующее логике вычислений. Дополнительно функциональные требования предусматривают поддержку динамических изменений данных моделей в процессе их работы. Установление строгой структуры хранения обеспечивает последовательное выполнение вычислений и корректное обновление значений переменных, что исключает ошибки обработки и повышает надежность модели (№5), (№7).

Технические требования направлены на обеспечение устойчивости функционирования системы в условиях роста объема обрабатываемых данных и повышения сложности моделей. Важнейшими параметрами являются масштабируемость системы, эффективность хранения данных и поддержание транзакционной целостности при изменении информации. Система должна обеспечивать как горизонтальное, так и вертикальное масштабирование по мере увеличения числа состояний и вычислительных процессов. Для взаимодействия между компонентами системы необходимо использование стандартных протоколов, таких как gRPC, что обеспечивает согласованность передачи и обработки данных в распределенной среде (№4), (№8), (№11). Структура данных должна поддерживать механизмы миграции и эволюции без утраты целостности, что позволяет гибко адаптироваться к изменяющимся требованиям и обновлениям моделей (№17).

Архитектурные требования задают принципы построения структуры представления, способной функционировать в распределенной вычислительной среде. Унифицированное представление должно быть адаптивным, поддерживать модификацию и интеграцию в разнородные вычислительные системы. В рамках этих требований предусматривается наличие механизмов динамического обновления структуры данных без необходимости остановки системы, а также синхронизации состояния между различными компонентами в условиях параллельной работы. Это позволяет достичь высокой доступности, отказоустойчивости и надежности работы всей системы (№6), (№14).

Интерфейсные требования регламентируют правила взаимодействия компонентов системы с данными моделей. Особое внимание уделяется стандартизации именования состояний, переменных и операций для исключения неоднозначностей при автоматической обработке данных. Установление четких правил описания и взаимодействия позволяет минимизировать вероятность ошибок при интеграции модулей и гарантировать стабильную работу вычислителя при приеме и интерпретации информации (№13), (№15).

Таким образом, сформулированные требования обеспечивают совместимость, масштабируемость и гибкость представления событийно-непрерывных моделей. Совместимость достигается благодаря использованию унифицированного формата описания данных и применению стандартных протоколов взаимодействия. Масштабируемость обеспечивается за счет адаптивной структуры хранения и поддержания динамического обновления моделей. Гибкость достигается за счет обеспечения миграции и эволюции данных, а также четкого структурирования информации в процессе развития моделей (№2), (№3), (№10).

## 2.5. Логическая схема данных и описание типов данных

Разработка логической схемы данных включает проектирование структуры хранения моделей, их состояний и связанных событийных изменений. В логической схеме основными сущностями являются модели, состояния, события и переменные. Модель представляет собой обобщённое описание объекта или процесса с заданными параметрами и характеристиками. Состояние фиксирует конкретную конфигурацию модели в определённый момент времени. Событие описывает изменение состояния модели и обеспечивает фиксацию переходов между ними, позволяя отслеживать динамику развития системы. Переменные отражают параметры модели, подверженные изменению во времени или в результате внешнего воздействия.

При проектировании логической схемы особое внимание уделяется обеспечению целостности, доступности и масштабируемости данных. Для поддержания целостности применяются механизмы внешних ключей между моделями, состояниями и событиями, а также процедуры валидации информации. Масштабируемость достигается использованием методов шардирования и индексирования сущностей по временным меткам и категориям объектов (№4), (№6). Нормализация данных минимизирует дублирование информации, а дедупликация и компрессия способствуют снижению объёмов хранения и повышению эффективности обработки данных.

Выбор типов данных является критически важным этапом проектирования схемы, оказывая прямое влияние на производительность систем моделирования. В логической схеме выделяются дискретные, непрерывные и составные типы данных. Дискретные типы включают логические значения, статусы состояний и коды событий, фиксирующие ключевые изменения в моделируемой системе (№5), (№8). Они характеризуются ограниченным числом допустимых значений и применяются для описания событийных переходов. Непрерывные типы используются для представления параметров, изменяющихся плавно во времени, таких как температура, скорость или давление (№7), (№14). Составные структуры, включая записи и списки, служат для группирования взаимосвязанных параметров и поддержания динамических коллекций состояний и событий.

Поддержка событийно-непрерывных процессов обеспечивается за счёт комбинированного использования дискретных и непрерывных типов данных. Дискретные элементы фиксируют мгновенные изменения состояния системы, тогда как непрерывные параметры описывают эволюцию между событиями. Такой подход позволяет моделировать как быстрые переходы, так и длительные процессы изменения характеристик системы (№5), (№7), (№14). Например, в технической системе событие выхода устройства из строя фиксируется дискретно, а изменения эксплуатационных параметров устройства до и после события моделируются непрерывно.

При данных условиях и требованиях к разным типам данных невозможно обойтись один источником и хранилищем данных если рассматривать внешние компоненты, такие как реляционные базы данных или иные их аналоги. По этой причине предпочтение стоит отдать хранению параметров в оперативной памяти. По мимо уже указанного приемущества, данный подход серьезно ускорит процесс вычисления и сократит затраты оперативной и энергонезависимой памяти, так как будет исключен полноценный компонент в виде базы данных и ее сервера.

Научная новизна предлагаемой логической схемы заключается в её адаптивности: структура данных способна эволюционировать вместе с моделями, сохраняя при этом целостность и эффективность хранения информации. Поддержка различных типов данных и механизмов их динамического обновления позволяет повысить точность моделирования сложных систем, снизить избыточность хранения и повысить общую производительность обработки данных. Эти решения особенно актуальны при построении событийно-непрерывных моделей социально-экономических и технических систем (№17), (№18).

## 2.6. Проектирование интерфейсов взаимодействия и обработки данных

Проектирование интерфейсов взаимодействия и обработки данных является ключевым этапом построения эффективных распределённых систем. Инженерные решения в этой области направлены на создание архитектуры, обеспечивающей надёжную, масштабируемую и гибкую интеграцию между клиентами, серверами-слушателями и серверами-вычислителями. Основной принцип проектирования интерфейсов состоит в чётком разграничении ролей компонентов, формализации внешних API и строгой регламентации внутренней логики обмена сообщениями.

Эффективная передача данных достигается за счёт использования унифицированных интерфейсов, минимизирующих накладные расходы при взаимодействии и обеспечивающих адаптивность к изменяющимся требованиям системы. При разработке внешних интерфейсов целесообразно применять стандартизированные протоколы, такие как REST и gRPC. Протокол gRPC, основанный на HTTP/2 и использующий сериализацию сообщений через Protocol Buffers, обеспечивает высокую скорость передачи данных, эффективную мультиплексированную коммуникацию и поддержку потоковых вызовов в обоих направлениях (№6), (№11).

При проектировании внутренних протоколов обмена данными особое внимание уделяется спецификации форматов сообщений, а также валидации и нормализации данных на всех этапах их обработки. Это обеспечивает целостность и согласованность передаваемой информации, независимо от особенностей источников или потребителей данных (№5), (№10). Для повышения устойчивости системы к ошибкам применяются механизмы предобработки, фильтрации и адаптации информации к требованиям бизнес-логики.

Надёжность взаимодействия компонентов достигается благодаря строгой спецификации интерфейсов, поддержанию совместимости версий сообщений и применению механизмов ретрансляции при потере данных. Протокол gRPC дополнительно усиливает надёжность системы через встроенные средства контроля состояния соединений и автоматическое управление тайм-аутами вызовов (№2), (№3). Масштабируемость системы обеспечивается унификацией интерфейсов, что упрощает горизонтальное расширение посредством добавления новых экземпляров сервисов без существенной переработки архитектуры (№17), (№18).

Унифицированные интерфейсы, основанные на строгом разграничении зон ответственности и единой модели данных, существенно снижают сложность интеграции новых модулей. Это позволяет оперативно адаптировать систему к изменяющимся требованиям внешней среды и расширять её применение в различных предметных областях. Динамическая адаптация интерфейсов к различным типам моделей данных повышает универсальность и прикладную ценность системы (№19), (№20).

Тщательная проработка принципов проектирования интерфейсов создаёт прочную основу для дальнейшей разработки протоколов взаимодействия на базе gRPC.

## 2.7. Разработка протоколов взаимодействия на базе gRPC

Разработка протоколов взаимодействия между вычислительными узлами и клиентским интерфейсом осуществлялась на базе технологии gRPC с использованием механизма сериализации данных Protocol Buffers. Проектирование включало определение структуры сервисов, методов вызова удалённых процедур и форматов обмена сообщениями. Для каждого вычислительного модуля были выделены отдельные gRPC-сервисы, отвечающие за выполнение специализированных задач, включая обработку вычислительных запросов, передачу промежуточных результатов и управление состоянием системы. Методы сервисов проектировались с учётом требований к минимизации времени отклика, включая реализацию синхронных и асинхронных вызовов для адаптации взаимодействия к различным сценариям распределённых вычислений.

Форматы передаваемых данных описывались с применением Protocol Buffers, что обеспечило компактное представление сообщений и высокую скорость их сериализации и десериализации (№2). Каждое сообщение формализовалось с точным указанием типов полей, поддержкой вложенных структур и повторяющихся элементов. Это позволило минимизировать сетевые накладные расходы, стандартизировать обмен данными между разнородными компонентами системы и упростить интеграцию новых сервисов.

Реализация потоковых операций через gRPC обеспечила поддержку односторонней и двусторонней потоковой передачи данных, что критически важно для построения интерактивных клиент-серверных приложений в распределённых вычислительных средах (№2). При серверном стриминге сервер передавал клиенту последовательность сообщений в ответ на единичный запрос, оптимизируя передачу больших объёмов данных. Клиентский стриминг использовался для агрегирования данных с последующей обработкой на стороне сервера. Двунаправленный стриминг позволял обеим сторонам обмениваться сообщениями параллельно и независимо, что повышало эффективность при реализации высоконагруженных взаимодействий.

Разрабатываемые протоколы взаимодействия учитывали требования к надёжности, производительности и масштабируемости системы. Использование встроенных механизмов gRPC для балансировки нагрузки и автоматического обнаружения сервисов обеспечивало динамическое перераспределение вычислительных задач между узлами, что способствовало повышению отказоустойчивости архитектуры (№4). В рамках данной работы зашита передаваемых данных не рассматривается, так как не является первостепенной, но для защиты передаваемых данных можно применять безопасные каналы связи на основе TLS и методы аутентификации OAuth (№2). Эти решения могут обеспечить низкую задержку взаимодействия, высокую пропускную способность и упрощали масштабирование распределённой системы без потери производительности.

Научная новизна разработанного подхода заключалась в интеграции возможностей gRPC и Protocol Buffers для стандартизации взаимодействия между вычислительными компонентами. Это позволяло упростить развитие распределённой инфраструктуры, повысить кроссплатформенную совместимость системы и снизить эксплуатационные затраты на её сопровождение (№5), что соответствует современным требованиям к высокоэффективным вычислительным комплексам.

Таким образом, разработанные протоколы обеспечили надёжную и производительную коммуникацию в распределённой вычислительной среде, создав прочную основу для последующей реализации программного комплекса.

# Глава 3. Реализация программного комплекса

## 3.1. Средства разработки и используемые технологии

Проект разработан на языке программирования C, что является основой для обеспечения высокой производительности и эффективного использования системных ресурсов. Язык C был выбран благодаря его способности предоставлять низкоуровневый доступ к памяти и процессорным ресурсам, что критично для разработки высокопроизводительных и надёжных приложений. Он позволяет минимизировать накладные расходы, что важно при работе с распределёнными вычислениями и обработке больших объемов данных. Язык C предоставляет широкие возможности для разработки как системных, так и прикладных приложений, что делает его универсальным выбором для данного проекта.

Для организации удалённого вызова процедур используется библиотека gRPC, основанная на принципах RPC (Remote Procedure Call). Это позволяет компонентам системы эффективно обмениваться данными, обеспечивая быстрое и надёжное взаимодействие между клиентом и сервером. gRPC поддерживает бинарную сериализацию, что снижает накладные расходы на передачу данных по сети, а также использует протокол HTTP/2, что гарантирует низкие задержки и высокую производительность. В проекте gRPC используется для оптимизации межпроцессного взаимодействия между клиентским интерфейсом, сервером-слушателем и серверами-вычислителями, что в свою очередь обеспечивает масштабируемость и гибкость системы при её расширении.

Контейнеризация, реализуемая с помощью Docker, является важной частью архитектуры проекта. Docker позволяет изолировать компоненты системы, устраняя зависимость от специфики операционных систем и аппаратных платформ. Это минимизирует проблемы совместимости и упрощает развертывание приложения в различных средах, включая облачные. Контейнеры обеспечивают воспроизводимость среды выполнения, что критически важно для стабильности и масштабируемости в распределённых вычислениях. Использование Docker также упрощает процесс тестирования, развертывания и обновления приложения, что увеличивает гибкость и удобство поддержки.

Для автоматизации сборки и управления зависимостями используется система CMake. CMake обеспечивает переносимость проекта и автоматизирует процесс компиляции, исключая зависимость от операционных систем и сборочных инструментов. Она также поддерживает интеграцию с различными библиотеками и фреймворками, что упрощает управление зависимостями и гарантирует корректную сборку проекта на различных платформах.

Применение этих технологий в проекте объясняется их соответствием основным требованиям: надёжности, производительности и масштабируемости. Язык C обеспечивает необходимую производительность для вычислений, в то время как gRPC упрощает коммуникацию между компонентами системы. Docker облегчает развертывание и масштабирование, обеспечивая гибкость и изоляцию среды выполнения, а CMake гарантирует корректную сборку проекта на всех целевых платформах.

Настройка выбранных технологий в проекте требует внимательной проработки конфигураций для каждого компонента системы. Для работы с gRPC необходимо настроить сервер и клиенты, а также определить протоколы взаимодействия в файлах с расширением .proto. Docker-контейнеры требуют разработки соответствующих Dockerfile и конфигурационных файлов для оркестрации контейнеров. В свою очередь, CMake необходимо настроить на правильное управление зависимостями и интеграцию с другими библиотеками и фреймворками, использующимися в проекте.

Таким образом, выбор технологий и средств разработки полностью соответствует целям проекта, включая создание надёжной и масштабируемой системы для работы с событийно-непрерывными моделями. Выбранные инструменты обеспечивают высокую производительность, удобство интеграции и лёгкость в развертывании, что критично для эффективной реализации таких моделей в рамках распределённых вычислений.

## 3.2. Реализация клиентского интерфейса

Клиентская часть системы реализована с использованием архитектурной модели MVVM (Model-View-ViewModel), что обеспечивает четкое разделение представления, логики и данных. Такое разделение упрощает управление интерфейсами и взаимодействие с сервером, повышая гибкость и удобство поддержки. Разработка клиента осуществлялась на языке C# с использованием .NET Framework 4.8. Этот подход обеспечивает надежную интеграцию всех компонентов, необходимых для эффективного взаимодействия с серверной частью и представления информации пользователю.

Структура клиентского интерфейса состоит из нескольких ключевых компонентов. В первую очередь, предусмотрено окно настроек, в котором пользователь может указать IP-адрес сервера и порт для подключения. По умолчанию сервер прослушивает соединения на IP 0.0.0.0 и порту 55555, однако пользователь может настроить эти параметры в зависимости от требований. После ввода данных настройки становятся доступными для формирования запроса, который отправляется через gRPC. Запрос конвертируется в формат JSON, что позволяет обеспечивать простоту и гибкость передачи данных между клиентом и сервером.

После выполнения вычислений на сервере клиент получает результат в формате JSON. Этот ответ декодируется и обрабатывается, а полученные данные отображаются в интерфейсе пользователя. Для визуализации графов и моделей используется библиотека Graph#, которая позволяет эффективно работать с графическими представлениями данных.

Для создания синтаксических деревьев, необходимых для обработки сложных выражений и формул в моделях, используется инструмент ANTLR. Это средство обеспечивает эффективный парсинг и анализ текстовых данных, включая формулы, которые могут быть частью запросов или результатов расчётов.

Механизм логирования и обработки ошибок реализован в минимальном объеме: в клиентском приложении ведется общий журнал ошибок. Хотя система не предоставляет детальной информации о причинах возникновения ошибок, она фиксирует основные сбои, такие как неудачные попытки подключения или проблемы в процессе обмена данными.

Клиент активно взаимодействует с сервером через gRPC, что обеспечивает высокую производительность и надежность связи. Формирование и отправка запросов, а также получение ответов от сервера осуществляются через gRPC-клиент, реализованный в клиентском приложении. gRPC использует HTTP/2 для улучшения производительности и надежности, что особенно важно при обмене сложными данными. Ответы сервера также передаются в формате JSON, который затем обрабатывается клиентом и отображается в пользовательском интерфейсе.

Используемые технологии и библиотеки, такие как .NET Framework, Avalon для визуализации, Graph# для работы с графами и ANTLR для парсинга, позволяют создать функциональный и удобный интерфейс, который соответствует всем требованиям взаимодействия с серверной частью. Выбор этих технологий был обусловлен необходимостью создания стабильного и масштабируемого приложения, способного работать в реальном времени и обеспечивать качественный пользовательский опыт.

Таким образом, реализация клиентского интерфейса обеспечена на основе современных архитектурных подходов и технологических решений, что позволяет создать эффективное средство для взаимодействия с сервером и обработки данных. Эти решения обеспечивают не только стабильную работу системы, но и высокий уровень удобства для пользователя.

## 3.3. Реализация сервера-слушателя

Сервер-слушатель является ключевым компонентом системы, обеспечивающим прием входящих подключений от клиентов и маршрутизацию их к вычислительным процессам. Реализация сервера выполнена на языке C с использованием TCP-сокетов для сетевого взаимодействия и gRPC API для стандартизированного обмена данными между клиентами и серверной частью. Принципиальной особенностью является то, что при установлении соединения сервер-слушатель инициирует только базовое подключение через gRPC, не обрабатывая передаваемые данные в формате JSON (№2), (№3).

Архитектурное решение обработки подключений строится на разделении обязанностей между сервером-слушателем и серверами-вычислителями. Сервер-слушатель непрерывно прослушивает заданный порт (например, 55555) на адресе 0.0.0.0. При поступлении нового подключения он использует системный вызов fork(), создавая отдельный процесс для обслуживания клиента. Дочерний процесс превращается в сервер-вычислитель, принимающий дальнейшие запросы, выполняющий десериализацию JSON-модели, обработку вычислений и передачу результатов клиенту (№5), (№10). Родительский процесс освобождает сокет подключения и продолжает прослушивание порта, что обеспечивает возможность параллельной обработки множества соединений без блокировки основного процесса.

Модель динамического запуска серверов-вычислителей обеспечивает эффективное масштабирование и высокую устойчивость системы. Изоляция вычислений в отдельных процессах локализует потенциальные сбои: отказ одного сервера-вычислителя не влияет на функционирование сервера-слушателя или других процессов, что существенно повышает отказоустойчивость архитектуры (№6), (№17).

Роль сервера-слушателя в общей архитектуре заключается в обеспечении стабильного входного уровня для клиентских подключений, в распределении вычислительной нагрузки и в координации создания вычислительных серверов. Он выполняет функции сетевого шлюза, обеспечивая базовую проверку соединений и быстроту реакции на входящие запросы, а также содействует реализации параллельной обработки данных через многопроцессную модель. Такая организация позволяет интегрировать сервер-слушатель в распределенные вычислительные среды и горизонтально масштабировать систему при росте числа клиентов и объема данных (№12), (№17), (№20).

Дополнительно сервер-слушатель реализует базовые механизмы устойчивости, свойственные распределенным системам. Обработка системных сигналов, включая SIGCHLD, позволяет отслеживать завершение дочерних процессов и предотвращать накопление "зомби"-процессов (№10). Применение gRPC стандартизирует протокол обмена сообщениями и облегчает масштабирование и совместимость системы с различными клиентскими платформами (№2), (№3).

## 3.4. Реализация сервера-вычислителя

Сервер-вычислитель занимает ключевое место в архитектуре вычислительной системы, обеспечивая приём моделей от клиентов, выполнение вычислений и возврат результатов. Основная задача сервера заключается в обработке событийно-непрерывных моделей, представленных в формате JSON. После получения данных сервер осуществляет их десериализацию с использованием библиотеки cJSON, преобразующей JSON-структуру в дерево объектов для пошаговой обработки (№1).

После десериализации начинается последовательная обработка модели. Для каждого состояния выполняется набор формул, определённых в модели. При этом используются встроенные математические функции: сложение, вычитание, умножение, деление и операция взятия модуля. Промежуточные значения переменных могут использоваться в последующих вычислениях текущего состояния. По завершении выполнения всех формул сервер проверяет условия переходов между состояниями, что позволяет моделировать динамическое поведение системы (№2).

Архитектура сервера-вычислителя включает несколько функциональных модулей: управление переменными, математическое ядро и обработчик моделей. Модуль переменных отвечает за хранение и обновление значений в формате ассоциативного списка "имя–значение". При необходимости сервер автоматически добавляет новые переменные, если они впервые встречаются в формуле. Математическое ядро реализует базовые арифметические и логические операции, необходимые как для выполнения вычислений, так и для оценки условий переходов между состояниями. Описание модели через состояния и переходы обеспечивает адекватное воспроизведение поведения гибридных систем (№3).

Для обеспечения эффективной обработки вычислительных задач сервер применяет асинхронную архитектуру. Использование неблокирующих операций ввода-вывода и событийных циклов на основе технологий epoll или libuv позволяет обрабатывать множество параллельных запросов без создания отдельного потока на каждое соединение (№4). Внутри одного состояния вычисления выполняются последовательно, однако для обслуживания разных клиентов или задач возможно параллельное выполнение с помощью многопоточности, реализованной средствами Pthreads или OpenMP.

В общей архитектуре системы сервер-вычислитель выполняет функции централизованного вычислительного узла. Клиенты направляют ему модели на обработку, а сервер возвращает обновлённые состояния или результаты расчётов. Благодаря модульной структуре и поддержке масштабируемости возможно развертывание нескольких экземпляров сервера с балансировкой нагрузки между ними, что обеспечивает высокую доступность и отказоустойчивость системы (№5).

Производительность сервера достигается за счёт минимизации накладных расходов на обработку данных, оптимизированной десериализации и асинхронной архитектуры сетевого взаимодействия. Обработка ошибок реализована таким образом, чтобы частичные сбои не приводили к остановке сервера. Горизонтальное масштабирование через добавление вычислительных узлов позволяет системе адаптироваться к изменяющейся нагрузке и требованиям к устойчивости (№6).

## 3.5. Логика обмена сообщениями между компонентами через gRPC

Компоненты системы взаимодействуют посредством gRPC-сообщений, определённых в protobuf-файлах. Этот механизм стандартизирует обмен информацией между клиентским интерфейсом, сервером-слушателем и сервером-вычислителем, обеспечивая строгую типизацию данных и высокую производительность коммуникаций (№2). При установлении соединения клиент инициирует запрос на сервер-слушатель, используя заранее определённые методы gRPC. Передаваемое сообщение содержит идентификатор сессии и данные, необходимые для выполнения вычислений.

Сервер-слушатель принимает запрос, осуществляет первичную проверку параметров и устанавливает канал связи для дальнейшего обмена данными. После обработки запроса он передает его серверу-вычислителю, созданному посредством механизма fork(). В этом процессе gRPC обеспечивает прозрачную маршрутизацию сообщений, минимизируя сетевые накладные расходы и сохраняя целостность данных (№3).

Сервер-вычислитель десериализует полученные данные, выполняет последовательное выполнение формул и обработку состояний модели. По завершении вычислений формируется результирующий JSON-объект с выходными параметрами. Этот объект передаётся обратно по тому же маршруту: от сервера-вычислителя через сервер-слушатель к клиентскому интерфейсу.

Обмен сообщениями может использовать однонаправленные вызовы (Unary RPC) и двусторонние потоки (Bidirectional Streaming RPC). В однонаправленном режиме клиент получает единичный ответ на каждый отправленный запрос. В режиме двусторонней потоковой передачи данные между клиентом и сервером передаются асинхронно в обе стороны без ожидания завершения передачи предыдущего сообщения, что особенно эффективно при обработке промежуточных вычислительных состояний и в задачах с динамическими изменениями входных данных (№4).

Надёжность обмена обеспечивается реализацией механизмов обработки ошибок и управления таймаутами. Компоненты анализируют коды ошибок gRPC, применяя стандартизированные процедуры восстановления, включая повторные попытки передачи сообщений и безопасное завершение сессии при системных сбоях. При превышении установленного таймаута соединение принудительно закрывается с последующим уведомлением клиента о неуспешной передаче данных. Подтверждение доставки реализуется через квитанции о получении сообщений, передаваемые на каждом этапе маршрута (№5).

Реализованная логика обмена через gRPC обеспечивает надёжность, эффективность и устойчивость функционирования системы. Применение стандартизированного бинарного формата protobuf минимизирует вероятность ошибок при интерпретации данных. Архитектурное разделение компонентов снижает риски возникновения единой точки отказа. Использование асинхронных вызовов и потоковой передачи данных сокращает задержки и увеличивает пропускную способность системы, что критически важно для выполнения ресурсоёмких вычислений в реальном времени (№6).

# Глава 4. Исследование работы и тестирование

## 4.1. Подходы к тестированию распределённых вычислительных систем

Разработка и эксплуатация распределённых вычислительных систем требуют особого подхода к тестированию. В отличие от централизованных решений, такие системы включают множество взаимодействующих узлов, распределённых в пространстве, зависят от качества сетевой инфраструктуры и предъявляют повышенные требования к согласованности данных. Это обуславливает необходимость применения комплексных методов испытаний, включающих функциональное тестирование, нагрузочные проверки и оценку отказоустойчивости.

Функциональное тестирование направлено на проверку корректности логики взаимодействия компонентов: маршрутизации запросов, репликации данных, обработки ошибок. Критически важным является контроль целостности информации при передаче между узлами, особенно в условиях нестабильных сетевых соединений. Тестирование охватывает как отдельные компоненты, так и систему в целом — с фокусом на согласованное выполнение распределённых операций и корректную реакцию на ошибки.

Нагрузочное тестирование позволяет определить пределы производительности системы при интенсивной обработке запросов и высокой сетевой активности. С помощью инструментов вроде Locust или Gatling моделируются сценарии с возрастающей нагрузкой, выявляющие поведение системы при пиковом потреблении ресурсов. В распределённой среде особое внимание уделяется сетевой латентности и асинхронной обработке. Использование эмуляторов, таких как netem, даёт возможность моделировать деградацию сети, потерю пакетов и переменные задержки (№7), (№12).

Проверка отказоустойчивости реализуется путём инъекции сбоев: отключения сервисов, нарушения сетевого взаимодействия, искусственного увеличения задержек. Инструменты Chaos Mesh и Toxiproxy позволяют воспроизвести широкий спектр аварийных сценариев. Целью является проверка способности системы сохранять ключевую функциональность и данные в условиях сбоев отдельных узлов. Это особенно важно в критически значимых приложениях, где недоступность одного компонента не должна приводить к отказу всей системы (№3), (№5), (№8).

Тестирование в условиях распределённой архитектуры сопряжено с рядом особенностей. Одной из них является вариативность сетевых характеристик: задержки доставки сообщений, асинхронность обработки, возможные рассинхронизации времени между узлами. Эти факторы усложняют воспроизведение ошибок и затрудняют отладку. Кроме того, протоколы согласования, механизмы репликации и событийного взаимодействия требуют детальной валидации как на уровне логики, так и на уровне временной согласованности (№10), (№14), (№17).

К числу типичных сложностей относятся нестабильные результаты тестов, ограниченность средств наблюдения за системой в распределённой среде, а также высокая сложность воспроизведения реальных отказов. Для их преодоления применяются методы детерминированной симуляции, подробное логирование всех сетевых взаимодействий и внедрение инструментов обсервабельности — метрик, трассировок, логов. Кроме того, широко используется дискретно-событийное моделирование, позволяющее воспроизводить взаимодействия компонентов в контролируемой среде, исключающей влияние внешних факторов (№9), (№15), (№18).

Таким образом, эффективное тестирование распределённых вычислительных систем основывается на сочетании различных методов и инструментов, имитирующих реальные эксплуатационные условия. Это обеспечивает выявление уязвимостей, оценку пределов надёжности и достижение стабильности работы при высокой нагрузке и в аварийных сценариях.

## 4.2. Разработка и проведение тестов клиент-серверного взаимодействия

Разработка и проведение тестов клиент-серверного взаимодействия является ключевым этапом оценки надёжности и эффективности распределённых систем. Тестирование ориентировано на выявление уязвимостей в процессе обмена данными между клиентом и сервером в условиях ограниченной пропускной способности сети, высокой задержки и нестабильности соединения. Целью является обеспечение устойчивой и предсказуемой работы системы в приближённых к реальности условиях эксплуатации.

Разрабатываемые тесты охватывают полный цикл взаимодействия: установление соединения, передача данных, обработка ошибок и восстановление после сбоев. Проверяется корректность установления соединения, устойчивость работы при нестабильной сети, сохранение целостности передаваемых данных, а также реакция системы на сетевые и программные сбои. Отдельное внимание уделяется оценке способности к автоматическому восстановлению соединения и повторной передаче данных после нарушений соединения.

При тестировании взаимодействия через gRPC учитываются особенности однонаправленных и потоковых соединений. В случае однонаправленных вызовов (unary RPC) оцениваются задержки отклика и корректность обработки запросов при ограниченной пропускной способности. Для потоковых сценариев — клиентских, серверных и двусторонних — проверяется устойчивость соединения при передаче больших объёмов данных, а также корректность завершения сеанса в условиях нестабильной сети (№2), (№3).

Процедуры автоматизированного тестирования реализуются с применением специализированных инструментов: Postman, k6 и других фреймворков для API-тестирования. Для симуляции сетевых условий используются программные эмуляторы, позволяющие задавать параметры потерь пакетов, задержек и пропускной способности. В рамках нагрузочного тестирования моделируются массовые параллельные обращения к серверу, имитирующие работу системы в условиях пиковой нагрузки. Все результаты автоматически фиксируются, что обеспечивает воспроизводимость и объективность анализа.

Интерпретация результатов тестирования осуществляется на основе чётких критериев. Тест считается успешным, если обеспечивается стабильность соединения, целостность данных, корректная обработка исключительных ситуаций и активация процедур восстановления без вмешательства пользователя. Надёжность подтверждается способностью системы сохранять функциональность при повторяющихся сбоях, а производительность оценивается по показателям времени отклика, пропускной способности и загрузке серверных ресурсов (№4), (№5), (№10).

Стабильность оценивается посредством длительных тестов с непрерывной передачей данных. Надёжность проверяется в ходе моделирования сбоев с последующим восстановлением. Производительность измеряется в стресс-тестах, в рамках которых анализируются задержки отклика, пропускная способность и эффективность использования ресурсов.

Комплексный подход к тестированию клиент-серверного взаимодействия позволяет выявлять как очевидные, так и скрытые архитектурные недостатки, проявляющиеся исключительно в условиях высокой нагрузки и нестабильных сетевых параметров. Систематическая проверка взаимодействия на всех уровнях является необходимым условием обеспечения качества программных решений в распределённых вычислительных системах (№6), (№10).

## 4.3. Проверка корректности работы вычислителя на тестовых данных

Для проверки корректности работы вычислительного модуля используются тестовые наборы данных с заранее известными результатами. Эти наборы включают как синтетические данные, подготовленные для оценки отдельных аспектов вычислений, так и данные из типовых прикладных задач, для которых эталонные значения определены аналитически либо получены с использованием сертифицированных программных средств. Особое внимание уделяется граничным случаям, включая нулевые, предельные и заведомо некорректные входные значения, что позволяет оценить устойчивость модуля к нестандартным ситуациям.

Процедура проверки предполагает поэтапную верификацию всех стадий обработки данных — от загрузки входных параметров до получения итогового результата. На каждом этапе выполняется сопоставление фактических выходных данных с эталонными значениями. Для каждой серии тестов фиксируются результаты, включая промежуточные значения, логи выполнения и величины отклонений. Это обеспечивает воспроизводимость проверок и позволяет точно локализовать возможные источники расхождений. Дополнительно проводятся повторные расчёты с варьированием параметров точности, среды выполнения и порядка операций, что позволяет оценить устойчивость алгоритма к изменениям внешних условий.

Оценка точности базируется на анализе абсолютных и относительных отклонений от эталонных значений. Допустимые пределы погрешностей задаются в соответствии с требованиями к точности модели и характеристиками обрабатываемых данных. Расчётные ошибки анализируются как количественно — через среднюю ошибку, стандартное отклонение и максимальное расхождение, — так и качественно, путём сопоставления поведения модели с реальными или моделируемыми сценариями функционирования системы (№5), (№7), (№12). Особое внимание уделяется систематическим ошибкам округления, возникающим из-за ограниченной разрядности, а также логическим ошибкам, связанным с нарушением условий обработки.

Результаты тестирования демонстрируют высокую степень соответствия между фактическими и ожидаемыми данными при всех проверенных сценариях. Во всех случаях отклонения не превышали установленного допустимого уровня. Выявленные расхождения были объяснимы с точки зрения ограничений численной точности и задокументированы. Во время проверки не зафиксировано нарушений логики обработки, в том числе при тестировании на граничных и нестандартных входных данных. На основании полученных данных можно заключить, что вычислительный модуль реализован корректно и соответствует предъявляемым требованиям (№2), (№3), (№10), (№20).

## 4.4. Оценка производительности и масштабируемости системы

Оценка производительности и масштабируемости системы проводилась с использованием нагрузочного тестирования, направленного на измерение времени отклика, пропускной способности и степени загрузки вычислительных ресурсов при различных сценариях эксплуатации. В качестве ключевых метрик использовались: среднее время отклика системы на входной запрос, характеризующее её реактивность; пропускная способность, определяемая числом успешно обработанных операций в единицу времени; а также загрузка процессора, оперативной памяти и сетевых интерфейсов, отражающая эффективность использования доступных ресурсов.

Для оценки производительности были разработаны тестовые сценарии с варьированием объема входных данных, числа параллельных запросов и интенсивности обращений. Система испытывалась в условиях имитации многопользовательской нагрузки с одновременным выполнением большого числа операций. Мониторинг осуществлялся как встроенными средствами, так и внешними инструментами анализа, что обеспечивало объективность и воспроизводимость результатов (№10). Измерения проводились многократно, что позволило выявить устойчивые тенденции в поведении системы под нагрузкой.

Масштабируемость проверялась посредством пошагового увеличения числа вычислительных узлов в распределённой архитектуре. В условиях горизонтального масштабирования к системе последовательно добавлялись экземпляры компонентов обработки, после чего проводились идентичные тесты. Анализ результатов показал, что при увеличении количества узлов до определённого предела наблюдается почти линейный рост пропускной способности и снижение времени отклика. Однако после достижения архитектурных и сетевых ограничений рост производительности замедлялся (№6), (№12). Это свидетельствует о наличии внутренних узких мест, требующих внимания при дальнейшем наращивании инфраструктуры.

На производительность существенное влияние оказали архитектурные решения. Применение асинхронной обработки и балансировщиков нагрузки обеспечило равномерное распределение запросов между узлами. Введение кэширования промежуточных результатов позволило сократить обращения к основным компонентам, что особенно важно при повторяющихся операциях. В то же время централизованные элементы — такие как база данных и очередь сообщений — становились узкими местами при пиковой нагрузке, демонстрируя необходимость в их оптимизации или децентрализации (№5), (№17), (№18).

Результаты тестирования подтверждают, что система сохраняет стабильную производительность при значительном росте числа пользователей и увеличении объёма обрабатываемых данных. Характер наблюдаемой масштабируемости и реакции на нагрузку демонстрирует эффективность применённых архитектурных подходов и потенциал системы к дальнейшему расширению без критической деградации ключевых метрик (№8), (№14).

## 4.5. Сравнительный анализ с монолитными и другими распределёнными архитектурами.

В данном разделе выполнен сравнительный анализ гибридной архитектуры, предложенной в работе, с монолитными, классическими распределёнными и микросервисными решениями. Сравнение проводится по ключевым метрикам: масштабируемость, производительность, отказоустойчивость и гибкость сопровождения. Цель анализа — определить, в каких условиях гибридный подход оказывается более эффективным, а где предпочтительнее применение традиционных архитектур.

Монолитные системы характеризуются высокой связностью компонентов и общим пространством выполнения. На ранних этапах разработки они обеспечивают стабильную производительность за счёт отсутствия накладных расходов на межкомпонентное взаимодействие. Однако масштабируемость таких решений ограничена: любое расширение функциональности требует модификации всей системы. Это затрудняет сопровождение и усложняет внесение изменений. Отказоустойчивость монолитной архитектуры невысока — сбой одного из компонентов может привести к полной остановке приложения (№11), (№13).

Классические распределённые архитектуры обеспечивают частичную декомпозицию, при которой компоненты могут функционировать на различных узлах, однако требуют тесной координации. Масштабируемость таких решений ограничена из-за сложности согласованного взаимодействия и необходимости централизованного контроля (№5), (№8). Производительность систем этого типа зависит от характеристик сетевой инфраструктуры и используемых протоколов. Сопровождение подобных решений требует значительных усилий, особенно при модификации схемы обмена данными между модулями.

Микросервисная архитектура реализует принцип полной изоляции компонентов, каждый из которых развёртывается и масштабируется независимо. Это обеспечивает максимальную гибкость сопровождения и высокую отказоустойчивость: сбой одного микросервиса не влияет напрямую на функционирование остальных (№12), (№17). Однако данная архитектура сопровождается значительными издержками на межсервисные коммуникации и сериализацию данных, что может снижать общую производительность (№14), (№15).

Предложенная гибридная архитектура сочетает централизованное управление критически важными функциями с событийно-ориентированным взаимодействием между распределёнными компонентами. Применение дискретно-событийного моделирования (№9), (№10), (№16) позволяет проводить оценку поведения системы в различных сценариях ещё на этапе проектирования. Это повышает предсказуемость и надёжность работы архитектуры. Благодаря возможности перераспределения нагрузки в режиме реального времени и локализации отказов, обеспечивается высокий уровень отказоустойчивости, превосходящий монолитные и классические распределённые решения (№18), (№20).

Сценарные расчёты подтверждают преимущества гибридного подхода. В условиях пикового трафика он демонстрирует меньшие задержки отклика по сравнению с микросервисной архитектурой за счёт оптимизации маршрутов обработки событий. В случае отказа одного из компонентов система продолжает функционировать за счёт встроенных механизмов переадресации, что снижает риски критических простоев. Масштабирование достигается путём масштабной репликации только отдельных функциональных блоков, что позволяет снизить инфраструктурные затраты по сравнению с монолитной моделью (№6), (№19).

Тем не менее, гибридная архитектура не является универсальной. В проектах с ограниченными ресурсами и жёсткими временными рамками (например, в типовых корпоративных приложениях) более целесообразным может быть применение монолитного подхода, обеспечивающего минимальное время вывода продукта на рынок. В системах, где требуется экстремальная масштабируемость при минимальной нагрузке на администрирование — например, в облачных B2C-сервисах, — оправдано применение микросервисной архитектуры.

Таким образом, гибридная архитектура демонстрирует наилучшие результаты в условиях переменной нагрузки, высоких требований к отказоустойчивости и необходимости адаптации к динамически меняющейся внешней среде. Её применение особенно эффективно в рамках комплексных, распределённых систем с критически важной бизнес-логикой.

## 4.6. Анализ устойчивости к сбоям и нагрузочным ситуациям

В рамках оценки надёжности и отказоустойчивости системы были смоделированы различные сценарии сбоев, отражающие реалистичные и критически значимые условия эксплуатации. Рассматривались случаи выхода из строя отдельных узлов, включая серверы обработки данных и элементы распределённого хранения, а также перегрузка сети, возникающая при пиковом трафике. Дополнительно учитывались сбои в обработке данных, вызванные некорректной маршрутизацией и задержками выполнения операций. Анализ охватывал также потерю соединения между подсистемами и временную недоступность компонентов вычислительной и сетевой инфраструктуры (№5), (№10).

Для оценки поведения системы в условиях отказов применялись методы дискретно-событийного моделирования и имитационного тестирования (№8), (№9), (№15). Эти подходы позволили воспроизвести сложные технические и сетевые сценарии, включая последовательные и множественные отказы с анализом их совокупного влияния. Проводилось стресс-тестирование, в ходе которого искусственно создавались пиковые нагрузки с целью выявления предельных условий функционирования системы и проверки эффективности механизмов масштабирования и балансировки (№12), (№17). В ряде экспериментов использовались гибридные модели, сочетающие поведенческое моделирование отклика системы и моделирование параметров физической инфраструктуры (№14), (№18).

Результаты показали, что система сохраняет работоспособность при частичных отказах и кратковременной перегрузке сетевых каналов. Моделирование отказов выявило, что при выходе из строя одного узла происходит автоматическое перераспределение потоков данных, позволяющее продолжить обслуживание без критического снижения производительности. При одновременной потере связи с несколькими узлами зафиксировано увеличение времени отклика, однако функции обработки сохранялись благодаря резервной маршрутизации и механизмам повторного выполнения операций (№10), (№15).

Стресс-тестирование подтвердило устойчивость системы при нагрузках до определённого предела, за которым наблюдалось непропорциональное снижение производительности. Это позволило выделить узкие места, требующие архитектурной оптимизации. Имитационные сценарии показали, что встроенные механизмы масштабирования и балансировки эффективно реагируют на кратковременные пики нагрузки, обеспечивая стабильное функционирование (№7), (№16). На основе анализа были сформулированы предложения по повышению отказоустойчивости, включая совершенствование алгоритмов балансировки, расширение резервирования сетевой инфраструктуры и внедрение инструментов активной диагностики (№17), (№20).

Планирование и интерпретация экспериментов опирались на нормативно-методическую базу, включая перечень критических технологий Российской Федерации (№2), (№3), (№4) и приоритетные направления развития науки и техники в области цифровых систем (№6). Это обеспечило соответствие проводимого анализа стратегическим задачам обеспечения устойчивости и надёжности отечественных информационных систем.

# Заключение

* Основные итоги работы
* Достижение целей и решение поставленных задач
* Практическая значимость и возможное применение
* Рекомендации для дальнейшего развития

# Список литературы

2 УКАЗ ПРЕЗИДЕНТА РОССИЙСКОЙ ФЕДЕРАЦИИ Об утверждении приоритетных направлений развития науки, технологий и техники в Российской Федерации и перечня критических технологий Российской Федерации http://www.kremlin.ru/acts/bank/33514

3 Указ Президента РФ от 7 июля 2011 г. N 899 "Об утверждении приоритетных направлений развития науки, технологий и техники в Российской Федерации и перечня критических технологий Российской Федерации" (с изменениями и дополнениями) https://base.garant.ru/55171684/

4 Приоритетные направления развития науки, технологий и техники Российской Федерации https://www.extech.ru/info/catalogs/razv\_sci.php

5 ДИСКРЕТНО-СОБЫТИЙНОЕ МОДЕЛИРОВАНИЕ В УПРАВЛЕНИИ ЭКОНОМИЧЕСКОЙ СИСТЕМЫ МУНИЦИПАЛЬНОГО ОБРАЗОВАНИЯ https://top-technologies.ru/ru/article/view?id=35084

6 Приоритетные направления развития науки http://www.opf.nsu.ru/ru/content/%D0%BF%D1%80%D0%B8%D0%BE%D1%80%D0%B8%D1%82%D0%B5%D1%82%D0%BD%D1%8B%D0%B5-%D0%BD%D0%B0%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F-%D1%80%D0%B0%D0%B7%D0%B2%D0%B8%D1%82%D0%B8%D1%8F-%D0%BD%D0%B0%D1%83%D0%BA%D0%B8

7 Непрерывное моделирование https://ru.ruwiki.ru/wiki/%D0%9D%D0%B5%D0%BF%D1%80%D0%B5%D1%80%D1%8B%D0%B2%D0%BD%D0%BE%D0%B5\_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5

8 Событийное моделирование цифровой системы регулирования https://cyberleninka.ru/article/n/sobytiynoe-modelirovanie-tsifrovoy-sistemy-regulirovaniya

9 ДИСКРЕТНО-СОБЫТИЙНОЕ МОДЕЛИРОВАНИЕ В УПРАВЛЕНИИ ЭКОНОМИЧЕСКОЙ СИСТЕМЫ МУНИЦИПАЛЬНОГО ОБРАЗОВАНИЯ https://top-technologies.ru/ru/article/view?id=35084

10 Оценка надежности программного обеспечения методами дискретно-событийного моделирования https://cyberleninka.ru/article/n/otsenka-nadezhnosti-programmnogo-obespecheniya-metodami-diskretno-sobytiynogo-modelirovaniya

11 унификация программного обеспечения https://biblioserver.usurt.ru/cgi-bin/irbis64r\_13/cgiirbis\_64.exe?LNG=uk&Z21ID=&I21DBN=LAN\_PRINT&P21DBN=LAN&S21STN=1&S21REF=&S21FMT=fullw\_print&C21COM=S&S21CNR=&S21P01=0&S21P02=0&S21LOG=1&S21P03=K%3D&S21STR=%D1%83%D0%BD%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F+%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE+%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F

12 Возможности применения гибридного подхода в моделировании социально-экономических и социотехнических систем https://journals.vsu.ru/sait/article/view/10585

13 Моделирование — это способ создания точной копии объекта или процесса в виде модели, которая позволяет нам лучше понять его структуру и функционирование. https://skyeng.ru/magazine/wiki/it-industriya/chto-takoe-uml/

14 Моделирование жёстких гибридных систем с односторонними событиями в инструментальной среде Исма https://cyberleninka.ru/article/n/modelirovanie-zhyostkih-gibridnyh-sistem-s-odnostoronnimi-sobytiyami-v-instrumentalnoy-srede-isma

15 ОСОБЕННОСТИ РАЗРАБОТКИ ИМИТАЦИОННЫХ МОДЕЛЕЙ СЛОЖНЫХ ТЕХНИЧЕСКИХ СИСТЕМ НА ПРИМЕРЕ КАБЕЛЬНОГО ПРОИЗВОДСТВА https://7universum.com/ru/tech/archive/item/17644

16 СИМВОЛЬНАЯ СПЕЦИФИКАЦИЯ И АНАЛИЗ ПРОГРАММНЫХ МОДЕЛЕЙ ГИБРИДНЫХ СИСТЕМ https://www.nstu.ru/files/dissertations/dissertaciya\_bessonov\_17032016\_1459158757.pdf

17 Гибридные методы моделирования и оптимизации в сложных системах https://megagrant.ru/labs/lab\_rus\_91219/

18 Метод гибридного управления на основе адаптивной системы нейро-нечеткого вывода https://cyberleninka.ru/article/n/metod-gibridnogo-upravleniya-na-osnove-adaptivnoy-sistemy-neyro-nechetkogo-vyvoda

19 Гибридная интеллектуальная система https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%B1%D1%80%D0%B8%D0%B4%D0%BD%D0%B0%D1%8F\_%D0%B8%D0%BD%D1%82%D0%B5%D0%BB%D0%BB%D0%B5%D0%BA%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F\_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0

20 Динамическая верификация гибридных систем https://tmpaconf.org/images/pdf/pakulin.pdf