МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное
образовательное учреждение высшего образования
«Череповецкий государственный университет»

Институт (факультет) Информационных технологий
Направление подготовки (специальность) 27.04.04 Управление в технических системах
Выпускающая кафедра Автоматизации и управления

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Название работы РАЗРАБОТКА СИСТЕМЫ АВТОМАТИЧЕСКОГО РАЗВЕРТЫВАНИЯ МИКРОСЕРВИСНЫХ ПРИЛОЖЕНИЙ В ОБЛАЧНОЙ ИНФРАСТРУКТУРЕ НА ОСНОВЕ АЛГОРИТМА КОМБИНАТОРНОЙ ОПТИМИЗАЦИИ

Студента Смирнова Алексея Борисовича
Ф.И.О.

ДОПУСТИТЬ К ЗАЩИТЕ

Директор института (декан факультета)	Е.В. Ершов
Заведующий выпускающей кафедрой	А.Л. Смыслова
Руководитель выпускной квалификационной работы	Е.А. Маслов
Консультант по технико-экономическому обоснованию	Н.Л. Макарова
Нормоконтролер	А.Л. Смыслова
Выпускник	А.Б. Смирнов

Череповец
2021 г.

МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное
образовательное учреждение высшего образования
«Череповецкий государственный университет»

АННОТАЦИЯ

выпускной квалификационной работы студента

по теме:

Разработка системы автоматического развертывания микросервисных приложений в облачной инфраструктуре на основе алгоритма комбинаторной оптимизации

Студент

Смирнов Алексей Борисович

фамилия, имя, отчество — подпись

Руководитель работы

АО «Северсталь-инфоком»

место работы

Руководитель направления

должность

Маслов Евгений Александрович

фамилия, имя, отчество

(Перечислить основные вопросы, которые рассматривались; результаты работы)

Выпускная квалификационная работа посвящена исследованию способов минимизации количества задействованных виртуальных и физических серверов при распределении программных компонентов на серверной инфраструктуре.

Структура работы представлена введением, разделом, содержащим анализ факторов, определяющих потребность в разработке системы, математическим описанием задачи распределения ресурсов на серверах, результатами экспериментальной проверки предложенных технических решений, проектированием системы развертывания микросервисных приложений, технико-экономическим обоснованием, заключением, списком литературы и приложениями.

В ходе выполнения выпускной квалификационной работы был выполнен анализ проблемы управления ресурсами на различных уровнях автоматизации производства, произведен патентный поиск, результаты которого показали необходимость разработки новой системы для решения, данного проблемы. Разработка данной системы включала в себя математическое описание задачи распределения ресурсов на серверах, реализацию эвристических алгоритмов комбинаторной оптимизации, разработку функций оптимизации и выбор наиболее подходящего алгоритма

На основе выбранного алгоритма разработана система, позволяющая в автоматическом режиме разворачивать программные компоненты на сервера. Выполнено технико-экономическое обоснование данного проекта.

В выпускной квалификационной работе использовано 37 таблиц, 43 рисунка, 2 приложения, 62 источника литературы. Общее количество страниц работы - 141.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ, СОКРАЩЕНИЙ И ТЕРМИНОВ

АСУ ТП – автоматизированная система управления технологическим процессом

АСУП – автоматизированная система управления предприятием

ВО – вычислительное облако

ИТ – информационные технологии

ОЗУ – оперативное запоминающее устройство

ОС – операционная система

ПЗУ - постоянное запоминающее устройство

ПО – программное обеспечение

СУБД – система управления базами данных

СХД – система хранения данных

ЦОД – центр обработки данных

ЦП – центральный процессор

AMQP – advanced message queuing protocol (протокол для передачи сообщений между компонентами системы)

API – application programming interface (программный интерфейс приложения)

BF – best fit (наилучший подходящий)

BFD – best fit decreasing (наилучший подходящий с упорядочиванием)

CPU – central processing unit (центральный процессор)

CQRS – command query responsibility segregation (шаблон разделения ответственности на команды и запросы)

CRUD – create, read, update, delete (набор простых операций: создание, чтение, обновление, удаление)

DDD – domain-driven design (проблемно-ориентированный подход)

DNS – domain name system (система доменных имен)

DTO – data transfer object (объект передачи данных)

FF – first fit (первый подходящий)

FFD – first fit decreasing (первый подходящий с упорядочиванием)

GUID – globally unique identifier (глобальный уникальный идентификатор)

HTTP – hypertext transfer protocol (протокол передачи данных прикладного уровня)

JSON – javascript object notation (текстовый формат обмена данными)

MVVM – model-view-viewmodel (паттерн «модель – представление - модель представления»)

NF – next fit (следующий подходящий)

ORM – object-relational mapping (объектно-реляционное отображение)

REST – representational state transfer (архитектурный стиль взаимодействия компонентов распределённого приложения в сети)

SLA – service level agreement (соглашение об уровне предоставления услуги)

URI – uniform resource identifier (универсальный идентификатор ресурса)

WMS – warehouse management system (система управления складом)

WPF – windows presentation foundation (система для построения клиентских приложений Windows)

XML – extensible markup language (расширяемый язык разметки)

ВВЕДЕНИЕ

В современных экономических условиях предприятия заинтересованы в сокращении издержек. С целью увеличения прибыли компании доходы инвестируются в модернизацию технологий и внедрение информационных систем. Для совершенствования управления отраслями и отдельными предприятиями на основе применения математических методов, современных средств вычислительной техники и средств связи внедряются системы управления производственными процессами, системы планирования ресурсов предприятия, системы управления складами. Иными словами, информационные технологии используются для наилучшего использования производственных фондов, увеличения выпуска продукции, снижения ее себестоимости, повышения производительности труда, рентабельности производства и роста прибылей.

С целью более эффективного обеспечения безопасности данных и сохранности коммерческой тайны компании под данные информационные системы разрабатывают собственные центры обработки данных. Такие вычислительные ресурсы нуждаются в управлении. Если на уровне автоматизированных систем управления предприятием для решения данного вопроса используется такие подходы, как контейнеризация и оркестрация, т.е. автоматическое распределение, масштабирование программных компонентов и балансировка нагрузки на них, то на уровне автоматизированных систем управления технологическими процессами и систем диспетчерского управления и сбора данных применяются простые и надежные технические решения, и программное обеспечение размещается на серверах без применения контейнеризации. Данное ограничение вызвано тем, что сервера цеховых АСУ ТП отдалены друг от друга, могут находиться в демилитаризованных зонах, а также, ограничены по ресурсам. Однако, такие сервера также нуждаются в управлении ресурсами, и вручную такие задачи, как, например, оптимизация размещения

приложений в серверной инфраструктуре, становится решать крайне трудно ввиду временных ограничений в условиях непрерывно функционирующего предприятия с большим уровнем автоматизации.

В этой связи заслуживают изучения способы оптимизации автоматического развертывания микросервисных приложений на северах ЦОД для минимизации количества задействованных физических и виртуальных машин, что и определяет актуальность данной работы.

Объектом исследования является инфраструктура развертывания программного обеспечения на физических и виртуальных серверах центра обработки данных АО «Северсталь-инфоком».

Предметом исследования являются методы оптимизации автоматического развертывания микросервисных приложений на серверах.

Цель работы: минимизация количества задействованных виртуальных и физических серверов при распределении программных компонентов на серверной инфраструктуре.

Для достижения цели выпускной квалификационной работы необходимо выполнить следующие задачи:

1. Проанализировать существующие решения по развертыванию ПО на серверных фермах. Провести патентный обзор.
2. Рассмотреть существующие алгоритмы комбинаторной оптимизации, разработать математическую модель системы, адаптировать алгоритмы и реализовать симулятор системы для выбора наиболее подходящего алгоритма оптимизации.
3. Реализовать программное обеспечение оптимизации развертывания ПО, провести экспериментальные исследования работы системы.
4. Выполнить технико-экономическое обоснование проекта.

ГЛАВА 1. АНАЛИЗ ФАКТОРОВ, ОПРЕДЕЛЯЮЩИХ ПОТРЕБНОСТЬ В РАЗРАБОТКЕ СИСТЕМЫ

1.1 Описание объекта исследования

Главной особенностью современного этапа развития техники, в частности средств производства, является широкое использование вычислительной техники для автоматизации процессов умственного и физического труда. На сегодняшний день коренным образом изменяется характер средств производства, по существу, создается новая материально-техническая база общества. Современная автоматизация производства объединяет множество самых разных задач - от технических до управленческих. Попытка систематизировать эти задачи привела к появлению так называемой "пирамиды автоматизации" [21] (рис. 1.1). Это модель, объединяющая все сферы деятельности современного предприятия в единую информационную среду. В основе пирамиды технологические объекты – станки, конвейеры и т.п.

В структуре пирамиды компьютерной автоматизации различают 5 уровней, связанных между собой как по горизонтали, так и по вертикали информационными каналами:

– уровень 1 представляет из себя набор датчиков, исполнительных устройств, которые предназначены для сбора первичной информации и реализации управляющих воздействий;

– уровень 2 содержит программируемые контроллеры, осуществляющие локальное управление технологических объектом;

– уровень 3 включает системы диспетчеризации, сбора данных и оперативного управления технологическим процессом;

– уровень 4 занимает система управления производством, позволяющая управлять производственными и людскими ресурсами в ходе технологического

процесса, управлять качеством продукции, следить за обслуживанием оборудования;

– уровень 5 представляет из себя системы, оснащённые компьютерным оборудованием с программным обеспечением, позволяющим иметь полную информацию о всем производстве и осуществлять планирование ресурсов.

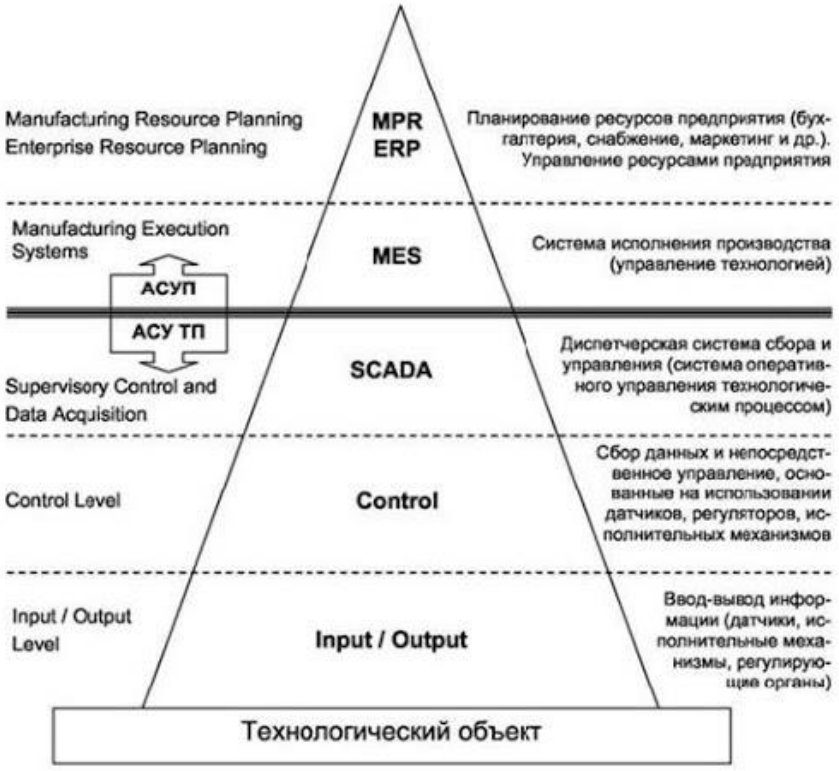


Рисунок 1.1 – Пирамида комплексной автоматизации предприятия

Первые три уровня образуют автоматизированную систему управления технологическим процессом, а четвертый и пятый - автоматизированную систему управления предприятием.

Работа данных информационных систем невозможна без использования центров обработки данных, которые поддерживают жизненно важные процессы. Центр обработки данных – это сложный комплекс, включающий в себя вычислительные мощности, элементы ИТ инфраструктуры, строительных и инженерных систем, основными функциями которого являются – хранение,

обработка и передача информации. В ЦОД на относительно небольшой площади сосредоточены мощные вычислительные ресурсы: сервера и системы хранения данных (СХД), осуществляющие хранение и обработку информации; сетевое оборудование, отвечающее за обмен данными внутри ЦОД, а также за связь с внешними потребителями; инженерные системы, системы безопасности, системы диспетчеризации и мониторинга, обеспечивающие эффективную работу и защиту сосредоточенного в ЦОД вычислительного центра.

При увеличивающихся темпах цифровизации возникает задача рационального использования ресурсов серверов, т.е. уменьшения количества задействованных виртуальных машин.

Приведем пример. На складе холоднокатанных рулонов производства плоского проката ПАО «Северсталь» развернута система слежения за продукцией на складе. При развертывании данной WMS системы существует необходимость в размещении нескольких программных компонентов на различных серверах с учетом таких параметров, как операционная система, уровень автоматизации, т.е. уровень АСУ ТП или уровень АСУП, необходимое количество оперативной памяти, свободное место на жестком диске. Количество таких программных компонентов в одной системе может насчитываться несколько десятков, в то время, как такие системы развернуты в нескольких цехах. На данный момент задача размещения программных компонентов решается следующим образом – на каждый сервер они загружаются вручную, обновление также производится вручную. Решение такой задачи вручную влечет за собой следующие последствия:

1. Увеличивается время развертывания приложений.

2. С числом программ растет вероятность ошибки размещения компонента.

3. Перебор вариантов размещения программных компонентов человеком за приемлемое время может привести к задействованию большего количества серверов, чем автоматическое решение задачи.

Данные последствия в совокупности влекут за собой увеличение потребления электроэнергии, расходов на аренду серверов.

Необходимо автоматизировать процесс управления ресурсами серверов, что, в свою очередь, позволит:

1. Снизить число задействованных серверов.
2. Уменьшить потребление электроэнергии.
3. Увеличить скорость развертывания приложений.

В процессе управления ресурсами серверов необходимо учитывать ограничения, такие как необходимое количество оперативной памяти, место на жестком диске, загрузка процессора.

Рассмотрим инструменты и готовые решения, позволяющие управлять распределением программных компонентов на серверах.

1.2 Обзор существующих решений и методов управления распределением программных компонентов на серверах

Перед тем, как провести обзор существующих решений в данной области, рассмотрим два варианта архитектуры приложений – монолитную и микросервисную, которые активно применяются в серверном программном обеспечении. Монолитная архитектура программного обеспечения – это архитектура, в которой приложение представляется в виде одного исполняемого файла с точкой входа. Приложения, построенные с использованием такой архитектуры, отличаются такими недостатками, как сложность горизонтального масштабирования, зависимость от применения одной технологии или языка, сложность изменения кода. В свою очередь, микросервисная архитектура — вариант программного обеспечения, направленный на взаимодействие насколько это возможно небольших, слабо связанных и легко изменяемых модулей — микросервисов, лишена данных недостатков, легко масштабируется, не зависит от применения одной технологии или языка и может поддерживаться

сравнительно небольшой командой разработчиков. На сегодняшний день микросервисная архитектура становится все более популярной в серверных приложениях [19]. Современные серверные приложения могут содержать сотни микросервисов, взаимодействующих между собой с помощью сетевых протоколов.

Наряду с широким внедрением микросервисных приложений распространение получили технологии контейнеризации [22]. Контейнеризация – это легковесная виртуализация и изоляция ресурсов на уровне операционной системы, которая позволяет запускать приложение и необходимый ему минимум системных библиотек в полностью стандартизованном контейнере, соединяющемся с хостом или чем-либо внешним по отношению к нему при помощи определенных интерфейсов. Контейнер не зависит от ресурсов или архитектуры хоста, на котором он работает, именно поэтому данная технология стала популярна.

Все компоненты, необходимые для запуска приложения, упаковываются как один образ и могут быть использованы повторно. Приложение в контейнере работает в изолированной среде и не использует память, процессор или диск операционной системы, в которой запущен контейнер. Это гарантирует изолированность процессов внутри контейнера. В результате контейнеризованное приложение может быть запущено на различных типах инфраструктуры. Благодаря такой высокой эффективности, контейнеризация обычно используется для упаковки множества отдельных микросервисов, из которых состоят современные приложения. На сегодняшний день самая распространенная система, позволяющая упаковать приложение со всем его окружением и зависимостями в контейнер – Docker [57].

Когда количество программных компонентов увеличивается, а процесс развертывания программного обеспечения рассматривается, как повторяющийся цикл разработки, тестирования и размещения программных продуктов на

серверах, вручную решать задачу оптимального размещения программных компонентов на серверах становится не только очень сложно, но и крайне неэффективно. Появляется необходимость управлять ресурсами северов.

Отдельные контейнеры требуют координации их взаимодействия. Такая координация называется оркестрацией. Технически существует возможность обойтись без оркестрации, т.е. создать контейнер, в котором будут запущены все необходимые процессы. Однако, такой подход лишен гибкости, масштабируемости, а также возникают вопросы безопасности, поскольку запущенные в одном контейнере процессы не будут изолированы и смогут влиять друг на друга. Оркестрация позволяет создавать информационные системы из множества контейнеров, каждый из которых отвечает только за одну определенную задачу, а общение осуществляется через сетевые порты и общие каталоги. При необходимости каждый такой контейнер можно заменить другим, что позволяет, например, быстро перейти на другую версию базы данных при необходимости. Существуют различные платформы для оркестрации контейнеров. Они позволяют реализовать удобные и эффективные средства развертывания контейнерных систем, построения единой централизованной консоли для применения политик управления. Рассмотрим наиболее известные: Docker Swarm и Kubernetes. Рассмотрим их более подробно.

Docker Swarm — это система кластеризации для Docker, которая превращает набор хостов Docker в один последовательный кластер, называемый Swarm. Docker Swarm отвечает за балансировку нагрузки и назначение уникальных DNS-имен, чтобы приложение, развернутое в кластере, можно было использовать так же, как и, если приложение было бы развернуто на одном Docker-узле, другими словами, управляющий узел распределяет запросы между рабочими узлами в кластере. Количество контейнеров может динамически масштабироваться как в сторону увеличения, так и в сторону уменьшения, управляющий узел отвечает за добавление или удаление контейнеров на узлах.

Kubernetes — система с открытым исходным кодом для управления контейнерными кластерами. Появилась в результате наработок Google при использовании механизма для изоляции процессов в виртуальной среде. Kubernetes распределяет контейнеры по узлам кластера в зависимости от текущей нагрузки и имеющихся потребностей в работе сервисов и обеспечивает распределение сетевой нагрузки. Распределение ресурсов [62] в процессе работы Kubernetes состоит из того, что ресурсы узлов динамически распределяются между выполняемыми на них контейнерами, и не нужно заботиться о том, как распределить контейнеры в кластере. Kubernetes следит за тем, чтобы не размещать на сервере больше контейнеров, чем есть ресурсов CPU для суммы потребностей всех контейнеров. Иначе, автоматически подключаются дополнительные виртуальные машины в кластер.

Балансировка сетевой нагрузки обеспечивается следующим образом. Каждое приложение разворачивается как в своем экземпляре контейнера и ему назначается IP-адрес, контейнеры скрыты за сервисом балансировки, также имеющим сетевой адрес [55]. IP-адрес сервиса используется только как точка входа и не обслуживается каким-либо процессом, слушающим этот ip-адрес и порт. В Kubernetes реализована псевдобалансировка. На рис. 1.2 показано направление сетевого запроса.

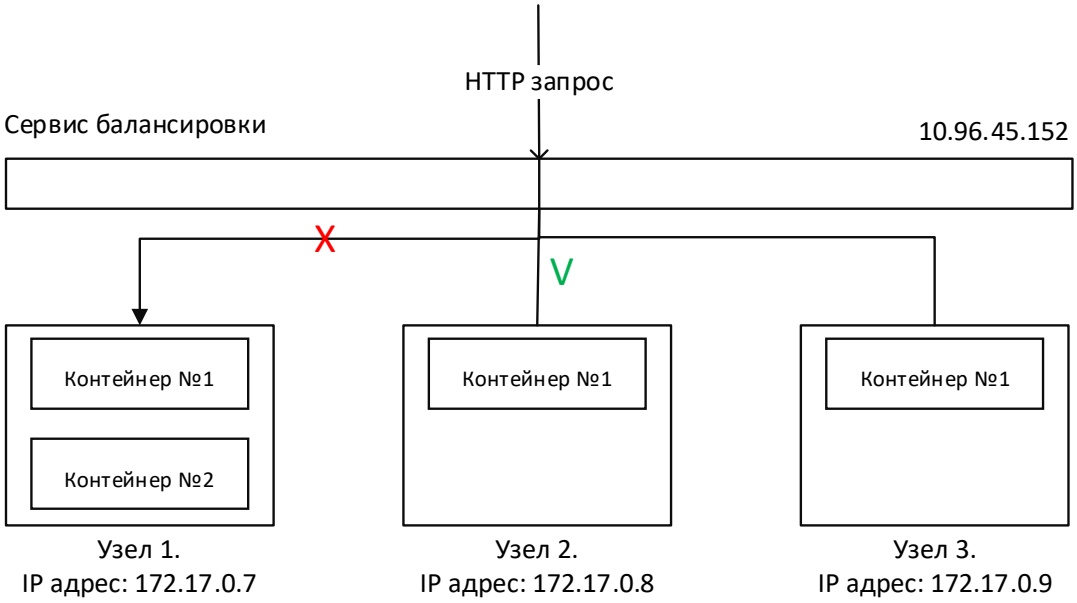


Рисунок 1.2 – Балансировка нагрузки в Kubernetes

То есть, если запущено три виртуальных узла, нагрузка будет распределена по следующим правилам:

1. Выбрать первый узел с вероятностью 33%, иначе перейти к следующему правилу.

2. Выбрать второй узел с вероятностью 50%, иначе перейти к следующему правилу.

3. Выбрать третий узел.

Такая система приводит к тому, что каждый узел выбирается с вероятностью 33%.

Использование метрик, как, например, потребность CPU в ресурсах, описанных в механизме распределения ресурсов в Kubernetes выше, может быть применено при разработке алгоритма оптимизации развертывания микросервисных приложений. Рассмотрим способы управления динамическими рабочими нагрузками в облачных вычислениях. Облачные вычисления — модель обеспечения сетевого доступа по требованию к определенному разделяемому фонду вычислительных ресурсов, например, серверам, устройствам хранения

данных, приложениям и сервиса, которые могут быть оперативно предоставлены и освобождены с минимальными эксплуатационными затратами или обращениями к провайдеру этих услуг.

Потребители облачных вычислений могут значительно уменьшить расходы на инфраструктуру информационных технологий (в краткосрочном и среднесрочном планах) и гибко реагировать на изменения вычислительных потребностей, используя свойства вычислительной эластичности облачных услуг. Одной из основных возможностей для динамического перераспределения ресурсов является живая миграция виртуальных машин [54]. Она позволяет облачным провайдерам перемещать виртуальные машины с перегруженных хостов, поддерживая их производительность при заданном SLA (соглашение об уровне услуг: средняя доступность, выраженная как среднее число сбоев на период предоставления сервиса, минимальная доступность для каждого пользователя, среднее время отклика сервиса и т.д.) и динамически консолидировать виртуальные машины на наименьшем числе хостов, чтобы экономить электроэнергию при низкой загрузке. Используя «живую» миграцию и применяя онлайн-алгоритмы, которые позволяют принимать решения о миграции в реальном времени, можно эффективно управлять облачными ресурсами, адаптируя распределение ресурсов к нагрузкам виртуальной машины, поддерживая уровни производительности виртуальной машины в соответствии с SLA и снижая энергопотребление инфраструктуры. Важной проблемой в контексте живой миграции является обнаружение состояния перегрузки или недогрузки хоста. Перспективным является подход принятия решений о живой миграции на основе прогнозов использования ресурсов на несколько шагов вперед [49]. Это не только повышает стабильность, так как миграционные действия начинаются только когда нагрузка сохраняется в течение нескольких временных интервалов, но также позволяет облачным провайдерам прогнозировать состояние перегрузки до того, как это произойдёт. Для

обнаружения перегрузки используется долгосрочное прогнозирование временных рядов [45]. Хост объявляется перегруженным, если фактическое и прогнозируемое общее использование CPU для установленного числа временных интервалов в будущем превышают порог перегрузки. Таким же образом определяется и недостаточная загрузка сервера. Глобальный агент принимает решения о распределении ресурсов провайдера с помощью живых миграций виртуальных машин с перегруженных или недогруженных хостов на другие узлы для снижения нарушений SLA и потребления энергии. Он получает уведомления от хост-агента, если узел будет перегружен или недогружен в будущем, и выполнит перенос виртуальной машины, если оно того стоит.

Рассмотренные выше технологии применяются на уровне автоматизированных систем управления предприятием. На уровне автоматизированных систем управления технологическим процессом не используется оркестрация и облачные вычисления, что обусловлено ограниченным количеством серверов, но на данном уровне все больше получают распространение микросервисные приложения. Соответственно, необходим инструмент для решения задачи оптимального распределения полезной нагрузки на серверах. Проанализируем сведения об изобретениях в области оптимизации ресурсов и балансировки нагрузки на серверах.

1.3 Патентный обзор

В ходе патентного поиска были найдены следующие зарегистрированные изобретения и полезные модели:

1. Управление ресурсами сервера, анализ и предотвращение вторжения к ресурсам сервера [RU 2 316 045 C2]

Сущность системы заключается в выполнении мониторинга выбранных ресурсов на одном или более компьютерных серверов и снижении коэффициента загрузки ресурсов в случае перегрузки указанных ресурсов сервера. Система

предотвращения вторжения к ресурсам сервера выполняет мониторинг выбранных ресурсов на одном или более компьютерных серверах. Программный комплекс может включать один или более программный хост-компонент и программный консольный компонент. Хост-компонент располагается на компьютере сервера и отслеживает использование ресурсов сервера. В случае, если уровень загрузки конкретного ресурса превышает текущий пороговый уровень, хост-компонент может принять корректирующие меры. Указанные меры могут включать в себя, например, снижение уровня использования ресурсов или уведомление пользователя через консольный компонент [2]. Недостатком такого подхода к управлению ресурсами состоит в жестком ограничении доступа к серверу в случае превышения порога загрузки ресурсов сервера.

2. Программно-определяемая автоматизированная система и архитектура [RU 2 729 885 C2]

Данная система предусматривает конфигурирование сети и развертывание функций/приложений автоматизации на лету на системном уровне с помощью технологий виртуализации. В исследуемом патенте описан алгоритм распределения программных компонентов. Алгоритм представляет из себя простую условную конструкцию. Компонент выбора вычислительных узлов может использовать одно или несколько правил, регулирующих требования к ресурсам данного хоста, ассоциированного с виртуальной машиной, чтобы выбирать вычислительный узел для развертывания. Примеры правил, которые может применять компонент выбора вычислительных узлов:

1. Если технология виртуализации хостов представляет собой виртуальную машину, то выбирается вычислительный узел с высокопроизводительным процессором (например, многоядерным Xeon-процессором).

2. Если технология виртуализации хостов представляет собой контейнер, то выбирается вычислительный узел со среднепроизводительным процессором (например, многоядерным Atom-процессором).

3. Если виртуальная машина имеет небольшой размер (например, менее 32 МВ, между 16 МВ и 64 МВ), то выбирается вычислительный узел без программного обеспечения [6].

Недостатком данного алгоритма является отсутствие отслеживания загрузки каждого вычислительного узла и наиболее полного использования ресурсов каждого вычислительного узла.

3. Способ и система интеллектуального управления распределением ресурсов в вычислительных средах [RU 2 609 076 C2]

Способ управления распределением информационных ресурсов заключается в том, что посредством компьютера формируют модель использования и перераспределения ресурсов в облачных вычислительных средах. Отличительной особенностью способа является то, что модель использования и перераспределения ресурсов в облачных вычислительных средах в вычислительном облаке формируют с использованием концепции интеллектуальных алгоритмов, последовательно выполняя совокупность операций, включающую три основных этапа: на первом этапе в вычислительном облаке выделяют ресурсы запускаемому экземпляру, на втором этапе проводят прогноз динамических параметров функционирования хостов (серверов) вычислительного облака, на третьем этапе осуществляют динамическое перераспределение ресурсов между экземплярами вычислительных облаков. При этом модель формируют для максимализации показателя $E_{\text{ВО}}$ - эффективности функционирования ВО в соответствии с выражением (1.1) и учетом ограничения (1.2):

$$E_{\text{ВО}} = F\left(E_{\text{ЦП}}, E_{\text{ОЗУ}}, E_{\text{Д}}, E_{\text{С}}, T\right) \quad (1.1)$$

$$R_{\text{ИСП}} \leq R_{\text{ИМ}} \quad (1.2)$$

где $E_{\text{ВО}}$ – комплексный показатель эффективности вычислительного облака, отражающий совокупную синергетическую эффективность выполнения

экземпляров в рамках каждого типа ресурсов вычислительного облака; $E_{\text{ЦП}}$, $E_{\text{ОЗУ}}$, $E_{\text{Д}}$, $E_{\text{С}}$ – совокупность частных показателей, равных числу экземпляров с удовлетворенной потребностью в ресурсах, соответственно центрального процессора (ЦП), оперативной памяти (ОЗУ), в дисковых (Д) ресурсах, в сетевых (С) ресурсах; T – время функционирования вычислительного облака (процессорное время); $R_{\text{ИСП}}$ – общий объем использующихся экземплярами ресурсов вычислительного облака; $R_{\text{ИМ}}$ – общий объем имеющихся ресурсов вычислительного облака [4].

Недостатки: используется все дисковое пространство, нет стремления к минимизации использования ресурсов серверов. Такой подход может применяться при управлении всеми ресурсами вычислительного центра, но не при хостированию отдельных микросервисных приложений. Также, распределение ресурсов выполняется только на основе сформированного вычислительного облака, что приводит к увеличению времени развертывания новых экземпляров приложений.

4. Система и способ оптимизации использования ресурсов компьютера [RU 2 475 819 C1]

Система удаления неиспользуемых объектов, включающая средство контроля, предназначенное для определения степени загруженности, по крайней мере, одного ресурса компьютера и нахождения неиспользуемых объектов.

Согласно изобретению, алгоритм работы системы состоит из следующих итераций:

1. На первом этапе совокупности операций по формированию модели производится оценка поступления ресурсов от источников, каждый источник характеризуется индексом поступления ресурсов, который задается как отношение поступивших ресурсов к запрошенным у источника.

2. На втором этапе дополнительно учитываются необходимые ресурсы для функционирующих экземпляров приложений, а также составляется перечень ресурсов, которые необходимы для экземпляров приложений, в которых имеется потребность.

3. На третьем этапе запрашиваются необходимые ресурсы у источников с более высоким индексом поступления ресурсов с целью получения недостающих ресурсов и последующего динамического распределения между экземплярами приложений.

Перечисленная новая совокупность существенных признаков позволяет за счет дополнительного учета характеристик источников при запросе дополнительных ресурсов не только распределять ресурсы, имеющиеся в вычислительном облаке, но и запрашивать недостающие ресурсы для вычислительного облака и выделять их экземплярам, которым не хватило ресурсов, тем самым повысить устойчивость функционирования информационно-вычислительной системы путем запуска всех необходимых экземпляров приложений в вычислительном облаке [3].

Недостатки: в данном изобретении не учитывается возможность нерационального использования ресурсов, то есть возможность с помощью данного алгоритма получения схемы распределения ресурсов, заполненность которых не стремится к минимуму.

5. Система управления и диспетчеризации контейнеров [RU 2 666 475 C1]

Система управления размещением программных контейнеров предполагает, что программные контейнеры могут быть запущены с возможностью выполнения в качестве задач в соответствии с определением задач, а определение задач может сохраняться в форме файла определения задач. Файл определения задач может описывать один или большее количество программных контейнеров, назначенных для запуска в качестве группы. Образы программного обеспечения программных контейнеров, которые могут представлять собой полную копию

конкретного состояния программного контейнера на момент создания образа программного обеспечения, выполненные с возможностью выполнения в экземплярах программных контейнеров, могут предоставляться поставщику служб вычислительных ресурсов или в его местоположениях, указанных в определении задач. Определение задач также может определять потребности в ресурсах, отношения между контейнерами, используемые сетевые порты и совместно используемые ресурсы. При получении запроса на запуск задач из определения задач планировщик может определить, в соответствии со схемой размещения, какие экземпляры программных контейнеров в кластере будут выполнять задачи. В некоторых случаях поставщик служб вычислительных ресурсов может предоставлять многопользовательский планировщик для определения того, где выполнять программные контейнеры, а в некоторых случаях поставщик служб вычислительных ресурсов может позволить клиентам обеспечивать и настраивать собственные планировщики для настройки работы планировщика. В некоторых случаях планировщик может быть выполнен с возможностью использования случайной схемы выбора для случайного (согласно некоторой схеме стохастического распределения) или циклического выбора экземпляра контейнера для размещения указанного программного контейнера, учитывая потребность в ресурсах, указанную в определении задач [5].

Недостатки: данная система использует жестко заданный алгоритм распределения ресурсов, требующий настройки под определенный набор серверов.

Обзор существующих технических средств показал, что на данный момент создано достаточно решений управления развертыванием программного обеспечения на уровне автоматизации управления производством, однако, готовых систем, позволяющих управлять ресурсами серверной инфраструктуры на уровне автоматизированных систем управления технологическим процессом нет, что подтверждает необходимость разработки собственного решения.

1.4 Цель и задачи работы

Цель работы: минимизация количества задействованных виртуальных и физических серверов при распределении программных компонентов на серверной инфраструктуре.

Для достижения цели выпускной квалификационной работы необходимо выполнить следующие задачи:

1. Проанализировать существующие решения по развертыванию ПО на серверных фермах. Провести патентный обзор.

2. Рассмотреть существующие алгоритмы оптимизации, разработать математическую модель системы, адаптировать алгоритмы и реализовать симулятор системы для выбора наиболее подходящего алгоритма оптимизации.

3. Реализовать программное обеспечение оптимизации развертывания ПО, провести экспериментальные исследования работы системы.

4. Выполнить технико-экономическое обоснование проекта.

1.5 Выводы по главе 1

В данной главе были рассмотрены факторы, обуславливающие необходимость автоматического управления развертыванием приложений. Были проанализированы существующие на данный момент программные продукты, позволяющие балансировать использование ресурсов и сетевую нагрузку на сервера ЦОД, однако такие средства используются на уровне автоматизированных систем управления предприятием. В ходе анализа выявлено, что готовых систем управления ресурсами серверной инфраструктуры предприятия нет, что подтверждает необходимость разработки собственного решения.

ГЛАВА 2. РАЗРАБОТКА МАТЕМАТИЧЕСКОЙ МОДЕЛИ АВТОМАТИЧЕСКОГО РАЗВЕРТЫВАНИЯ МИКРОСЕРВИСНЫХ ПРИЛОЖЕНИЙ В ОБЛАЧНОЙ ИНФРАСТРУКТУРЕ

2.1 Разработка математической модели системы

Проблема минимизации количества серверов при распределении на них программных компонентов, описанная в главе 1, более строго звучит следующим образом: «имеется коллекция программных компонентов (далее будем называть их сервисами) и коллекция виртуальных машин (далее будем называть их серверами). Необходимо распределить все сервисы на минимальное количество серверов в предположении, что один сервис может располагаться только на одном сервере». Данная задача может быть описана с использованием терминологии комбинаторной оптимизации, как упаковка множества C на (U, S), где упаковка – это подсемейство $C \subseteq S$ множеств, такое, что все множества из C попарно не пересекаются, т.е. набор серверов, в которых сервис не входит в два разных сервера, U – множество серверов, а S – семейство подмножеств серверов. Эта задача, в свою очередь, является классической NP-полной задачей в теории вычислительной сложности и комбинаторике и более известна, как задача об упаковке в контейнеры [27, 17].

Для NP-полных вычислительных задач не существует алгоритма решения, способного вернуть результат за полиномиальное время. Это значит, что количество итераций или время поиска решения не полиномиально зависит от числа наблюдений исходных данных. Трудоёмкость таких задач экспоненциально растёт с увеличением объема данных.

Неотъемлемой частью алгоритма оптимизации является критерий оптимальности – количественная мера оптимизируемого качества объекта. Значение критерия оптимальности выражается целевой функцией. Таким

образом, задача оптимизации сводится к нахождению минимума или максимума целевой функции.

Приведем постановку задачи размещения сервисов на ограниченном количестве серверов. Возьмем N - множество сервисов и M – множество серверов. Рассмотрим сервисы и сервера с точки зрения задачи упаковки, как предметы и контейнеры, тогда w_j – размер j-го сервиса, а c_i – вместимость $i-$го сервера. Тогда, требуется найти такое разбиение множества серверов N на непересекающиеся подмножества $N_1, \dots, N_k$, чтобы сумма размеров сервисов в каждом подмножестве N_j не превосходила заданную вместимость конкретного сервера, и чтобы k было наименьшим возможным.

Математическая формулировка данной задачи следующая (2.1):

$$\begin{cases} \text{минимизировать } k = \sum_{i=1}^{n} y_i \\ \text{при условии } \sum_{j=1}^{n} w_j x_{ij} \le c_i y_i, i \in N = \{1, \dots, n\} \\ \sum_{i=1}^{n} x_{ij} = 1, j \in N \end{cases} \quad (2.1)$$

где

$$y_i = \begin{Bmatrix} 1, \text{если } i \text{ сервер используется} \\ 0, \text{если } i \text{ сервер не используется} \end{Bmatrix}, i \in N$$

$$x_{ij} = \begin{Bmatrix} 1, \text{если сервис } j \text{ добавлен на сервер } i \\ 0, \text{если сервис } j \text{ не добавлен на сервер } i \end{Bmatrix}, i \in N$$

Можно считать, что сервисы, принадлежащие каждому множеству N_i, размещаются на сервера разного размера, а цель состоит в размещении сервисов из множества N на как можно меньшее число серверов k.

Критерием оптимальности в данном случае будем считать число серверов k. Решение считается тем более оптимальным, чем ниже значение k.

Размер каждого сервиса w_j описывается кортежем $< r_j, h_j, f_j, t_j >$, где r_j – количество занимаемой оперативной памяти, h_j - количество занимаемой памяти на постоянном запоминающем устройстве, f_j - количество занимаемого процессорного времени, t_j – тип операционной системы, на которой запускается j-й сервис [18]. Введем следующие ограничения (2.2):

$$\begin{gathered} r_j \in (0, \mathrm{R}], \mathrm{j} \in N, \\ h_j \in (0, \mathrm{H}), \mathrm{j} \in N, \\ f_j \in (0, \mathrm{F}), \mathrm{j} \in N, \\ t_j \in [T1, \dots, Tn], j \in N \end{gathered} \tag{2.2}$$

где R и H – верхние пределы размера файла сервиса, выражаемые в гигабайтах, F – верхний предел занимаемого процессорного времени, выраженный, как процент вычислений в секунду каждым процессом, T – элементы множества типов операционных систем.

Вместимость каждого сервера c_i описывается кортежем $< \dot{r}_i, \ddot{r}_i, \dot{h}_i, \ddot{h}_i, \dot{f}_i, t_i >$, где $\dot{r}_i$ – количество свободной оперативной памяти, $\ddot{r}_i$ – общий объем оперативной памяти, $\dot{h}_i$ - количество свободной памяти на постоянном запоминающем устройстве, $\ddot{h}_i$ – общий объем памяти на постоянном запоминающем устройстве, $\dot{f}_i$ – свободное количество процессорного времени, t_i – тип операционной системы сервера. Введем следующие ограничения (2.3):

$$\begin{gathered} \dot{r}_i \in (0, \dot{R}], \mathrm{i} \in M, \\ \ddot{r}_i \in (0, \ddot{R}], \mathrm{i} \in M, \\ \dot{h}_i \in \left(0, \dot{H}\right), \mathrm{i} \in M, \\ \ddot{h}_i \in \left(0, \ddot{H}\right), \mathrm{i} \in M, \\ \dot{f}_i \in \left(0, \dot{F}\right), \mathrm{i} \in M, \\ t_i \in [T1, \dots, Tn], i \in M \end{gathered} \tag{2.3}$$

где $\dot{R}$, $\ddot{R}$ – количество свободной оперативной памяти и ее общий объем соответственно, выражаемые в гигабайтах, $\dot{H}$, $\ddot{H}$ – количество свободной памяти на постоянном запоминающем устройстве и ее общий объем соответственно, выражаемые в гигабайтах; $\dot{F}$ – верхняя граница занимаемого процессорного времени ЦП; T – элементы множества типов операционных систем.

В ходе решения задачи положение элементов, обозначенное заданным типом, в пространстве задается множеством 2.4:

$$S = \{M_1\{N_1, \dots, N_j\}, \dots, M_i\{N_1, \dots, N_j\}\}. \quad (2.4)$$

2.2 Описание существующих алгоритмов комбинаторной оптимизации

Для NP-полных вычислительных задач не существует алгоритма решения, способного вернуть результат за полиномиальное время [56]. Это значит, что количество итераций или время поиска решения не полиномиально зависит от числа наблюдений исходных данных. Трудоёмкость таких задач экспоненциально растёт с увеличением объема данных. Поэтому рассмотрим ряд эвристических алгоритмов, позволяющих найти решение, находящееся близко к оптимальному. В данной работе за критерий оптимальности примем количество максимально заполненных серверов.

Простейшими алгоритмами упаковки являются [37]:

1. Алгоритм «Следующий подходящий» (Next fit, NF). Упаковка предметов происходит в произвольном порядке по следующему правилу: первый предмет помещается в первый контейнер. На k-м шаге помещается k-й предмет в текущий контейнер. Если предмет помещается, то переходим к следующему шагу, иначе помещаем предмет в новый контейнер.

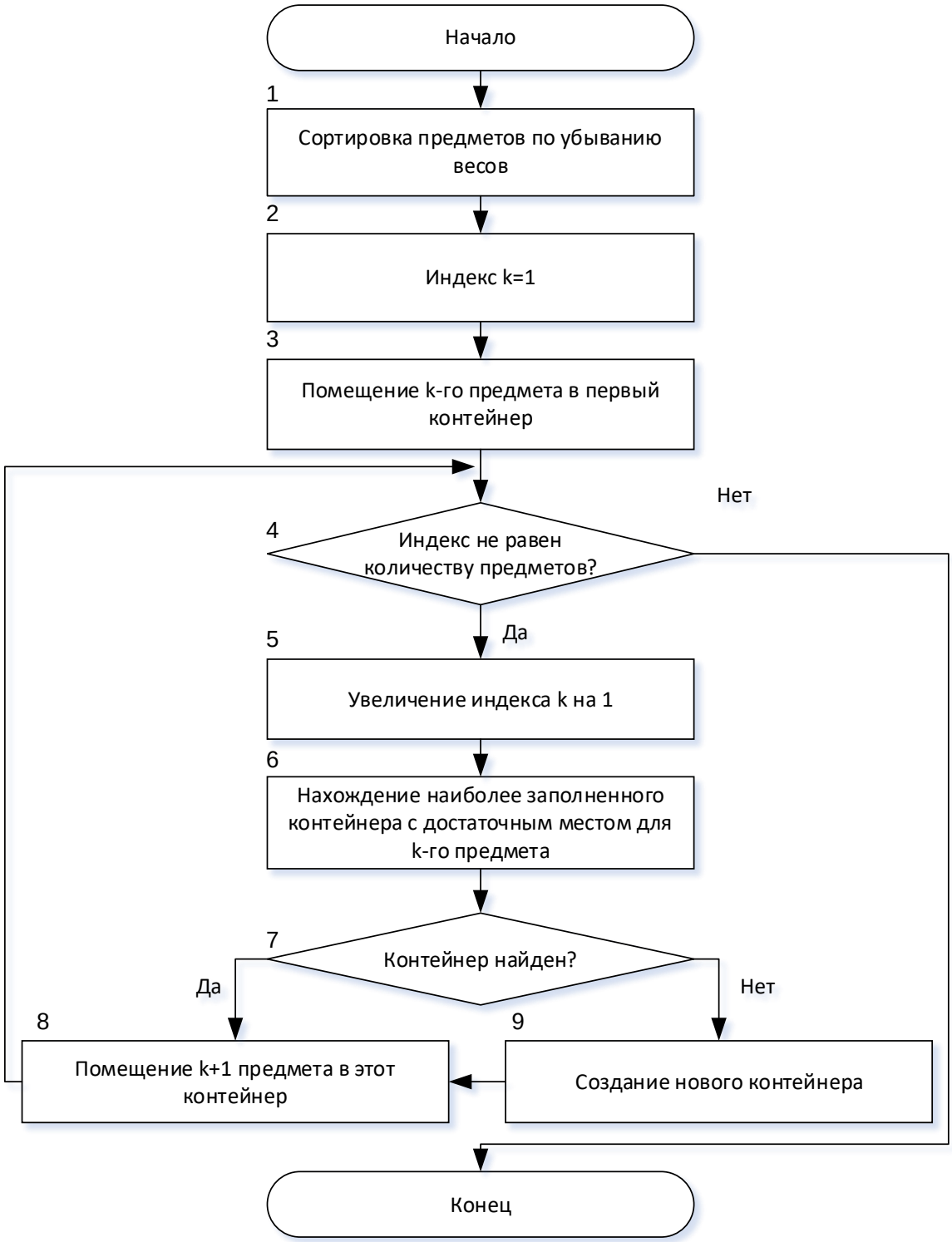
2. Алгоритм «Первый подходящий» (First fit, FF). Первый предмет помещается в первый контейнер. На k-м шаге ищется контейнер с наименьшим

номером, куда помещается k-й предмет, и помещаем его туда. Если такого контейнера нет, то берем новый пустой контейнер и помещаем предмет в него.

3. Алгоритм «Наилучший подходящий» (Best fit, BF). В произвольном порядке упаковываем предметы по следующему правилу: первый предмет помещаем в первый контейнер. На k-м шаге размещаем k-й предмет. Находим частично заполненные контейнеры, где достаточно для него свободного места и выбираем среди них наиболее заполненный. Если таких нет, то берем новый пустой контейнер и помещаем k-й предмет в него.

4. Алгоритм «Первый подходящий с упорядочиванием» (First fit decreasing, FFD). Предметы сортируются по убыванию весов $w1 \geq w2 \geq \ldots \geq wn$ и применяется алгоритм «первый подходящий» (FF).

5. Алгоритм «Наилучший подходящий с упорядочиванием» (Best fit decreasing, BFD). Предметы сортируются по убыванию весов $w1 \geq w2 \geq \ldots \geq wn$ и применяется алгоритм BF. На странице 29 приведена блок-схема алгоритма «наилучший подходящий с упорядочиванием»



					ЧГУ.Д.ВКР.270404.00.00.21 ПЗ	
Изм.	Лист	№ докум.	Подпись	Дата	Разработка системы автоматического развертывания микросервисных приложений в облачной инфраструктуре на основе алгоритма комбинаторной оптимизации	Лит. / Масса / Масштаб
Разраб.		Смирнов А.Б.				
Провер.		Маслов Е.А.				Лист 29 / Листов
Н.контр		Смыслова А.Л.			Алгоритм BFD	ЧГУ 1УТСм-01-21оп
Утверд.		Смыслова А.Л.				

Алгоритмы NF, FF, BF являются on-line алгоритмами. При работе таких алгоритмов упакованный предмет нельзя перемещать в другой контейнер. Такие алгоритмы применимы, например, при упаковке на конвейере. Алгоритмы FFD и BFD используются тогда, когда существует возможность доступа ко всем контейнерам [26].

Идеи локального поиска получили свое дальнейшее развитие в так называемых метаэвристиках, то есть в общих схемах построения алгоритмов, которые могут быть применены практически к любой задаче дискретной оптимизации. Идея этих методов основана на предположении, что целевая функция имеет много локальных экстремумов, а просмотр всех допустимых решений невозможен, несмотря на конечность их числа. В такой ситуации нужно сосредоточить поиск в наиболее перспективных частях допустимой области. Таким образом, задача сводится к выявлению таких областей и быстрому их просмотру. Каждая из метаэвристик решает эту проблему по-своему [9, 35]. Рассмотрим два метаэвристических алгоритма – алгоритм имитации отжига и генетический алгоритм.

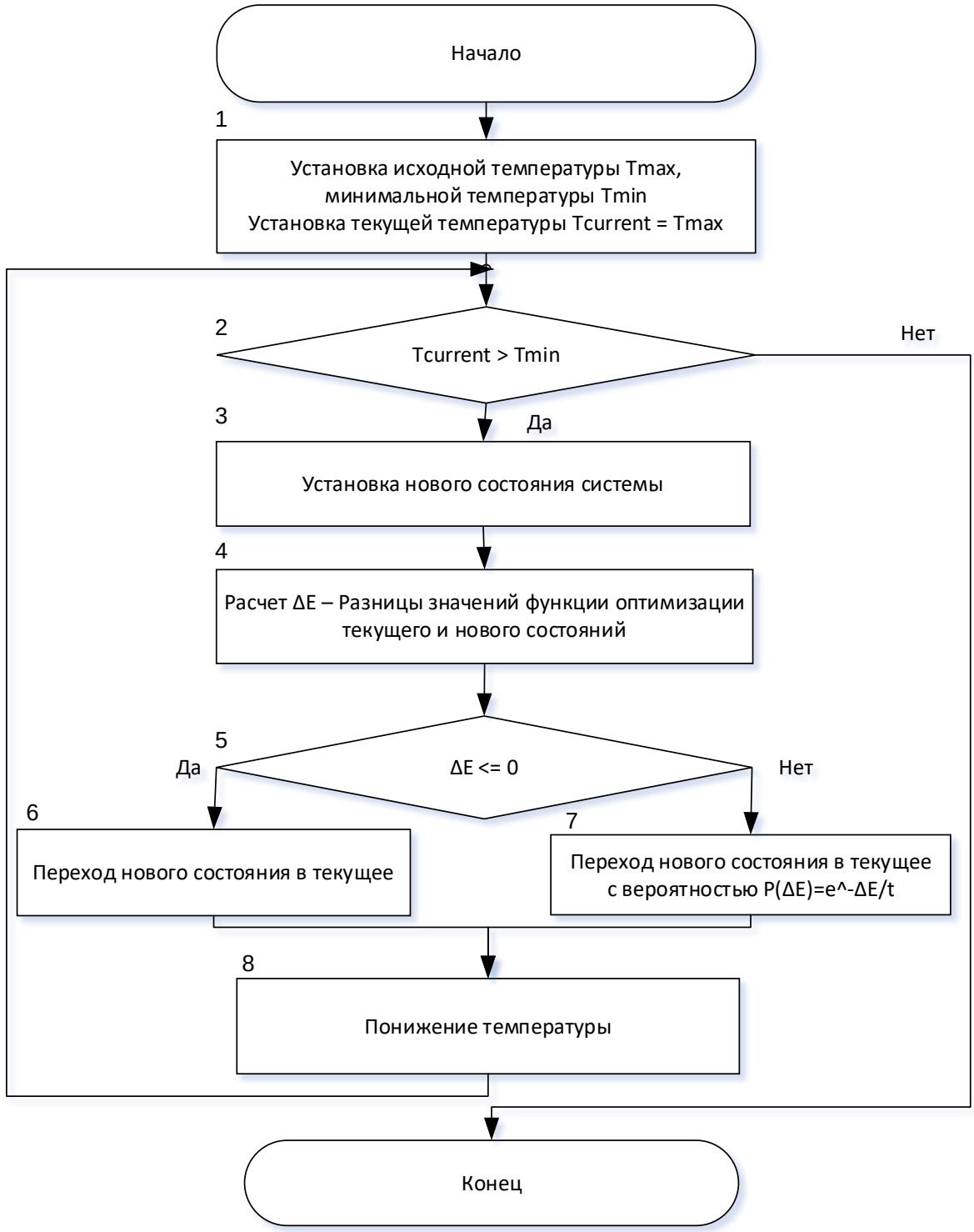
Для решения многих оптимизационных задач, в т.ч. комбинаторных [42, 51], применяется метод отжига, также известный, как алгоритм имитации отжига – алгоритм оптимизации, использующий упорядоченный случайный поиск на основе аналогии с процессом образования в веществе кристаллической структуры с минимальной энергией при охлаждении.

Метод отжига служит для поиска глобального минимума некоторой функции $f(x)$, заданной для x из некоторого пространства S, дискретного или непрерывного. Элементы множества S представляют собой состояния воображаемой физической системы (энергетические уровни), а значение функции f в этих точках используется, как энергия системы $E = f(x)$. В каждый момент предполагается заданной температура системы T, уменьшающаяся с течением времени. После попадания в состояние x при температуре T, следующее

состояние системы выбирается в соответствии с порождающим семейством вероятностных распределений $g(x,T)$, которое при фиксированных x и T задает случайный элемент $G(x,T)$ со значениями в пространстве S. После генерации нового состояния $x' = G(x,T)$ система с вероятностью $h(\Delta E,T)$ переходит к следующему шагу в состояние x', в противном случае, процесс генерации x' повторяется. Здесь ΔE означает приращение функции энергии $f(x') - f(x)$. Величина $h(\Delta E,T)$ называется вероятностью принятия нового состояния. Как правило, в качестве функции $h(\Delta E,T)$ выбирается либо точное значение соответствующей физической величины $h(\Delta E,T) = \frac{1}{1+\exp(\Delta E/T))}$, либо приближенное значение $h(\Delta E,T) = \exp(-\Delta E/T))$. Вторая формула используется наиболее часто. При ее использовании $h(\Delta E,T)$ оказывается больше единицы в случае $\Delta E < 0$, и тогда соответствующая вероятность считается равной 1. Таким образом, если новое состояние дает лучшее значение оптимизируемой функции, то переход в это состояние произойдет в любом случае. Существуют различные законы уменьшения температуры T с течением времени. Для Больцмановского отжига [44] используется закон $T(k) = \frac{T_0}{\ln(1+k)}, k > 0$. Доказано [41, 44], что для достаточно больших T_0 и общем количестве шагов k, выбор такого закона гарантирует нахождение глобального минимума, однако, скорость работы алгоритма относительно невысока, например, чтобы понизить исходную температуру в 40 раз, требуется $e^{40} \approx 2{,}35 * 10^{17}$ итераций. При использовании сверхбыстрого отжига по закону $T(k) = \frac{T_0}{k^{1/D}}$, где D – размерность пространства состояний, для решения задач в пространстве размерности больше 1, скорость алгоритма меньше, чем при применении Больцмановского закона. Также, существует метод под названием «Сверхбыстрый отжиг», в этом случае температура изменяется по закону $T(k) = T_0 \exp\left(-c_i k^{1/D}\right)$, $c_i > 0$, где c_i – декремент затухания температуры.

Вероятность пребывания системы в состоянии с энергией E при температуре T равна $h(\Delta E, T) = C \exp(-\Delta E / T)$, где C — нормировочный множитель. Это свойство общее для любых макроскопических систем и известно под названием распределения Гиббса [8, 50].

На странице 33 приведена блок-схема алгоритма имитации отжига.



					ЧГУ.Д.ВКР.270404.00.00.21 ПЗ	
					Разработка системы автоматического развертывания микросервисных приложений в облачной инфраструктуре на основе алгоритма комбинаторной оптимизации	Лит. / Масса / Масштаб
Изм.	Лист	№ докум.	Подпись	Дата		
Разраб.		Смирнов А.Б.				Лист 33 / Листов
Провер.		Маслов Е.А.				
Н.контр.		Смыслова А.Л.			Алгоритм имитации отжига	ЧГУ 1УТСм-01-21оп
Утверд.		Смыслова А.Л.				

Наряду с алгоритмом имитации отжига используется и генетический алгоритм – это эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путём случайного подбора, комбинирования и вариации искомых параметров с использованием механизмов, аналогичных естественному отбору в природе [46]. Цель генетического алгоритма при решении задачи оптимизации состоит в том, чтобы найти приближённое решение, близкое, но не гарантированно оптимальное решение. Впервые эти нестандартные идеи были применены к решению оптимизационных задач в середине 70-х годов [28, 39]. Примерно через десять лет появились первые теоретические обоснования этого подхода [34,47,48,11]. На сегодняшний день генетические алгоритмы доказали свою конкурентоспособность при решении многих NP-трудных задач [25,33,7,43]. Применение генетических методов для решения NP-трудных комбинаторных задач оптимизации полезно тогда, когда необходимый объем вычислительных затрат может оказаться большим, но скорость, с которой этот объем увеличивается при экспоненциальном росте размерности задачи дискретной оптимизации, часто может расти лишь линейно [9].

В теории генетических алгоритмов применяется следующая терминология [12]:

– ген (свойство) – атомарный элемент хромосомы. Ген может быть битом, числом или неким другим объектом;
– аллель – значение конкретного гена;
– локус – положение конкретного гена в хромосоме;
– хромосома (цепочка) – упорядоченная последовательность генов;
– генотип (код) – упорядоченная последовательность хромосом;
– особь (индивидуум) – конкретный экземпляр генотипа;

– фенотип – аргумент (набор аргументов) целевой функции, соответствующий генотипу (т.е. интерпретация генотипа с точки зрения решаемой задачи).

Структура популяции представлена на рис. 2.1. Хромосома является частью популяции, ген является частью хромосомы. Аллель – это значение конкретного гена.

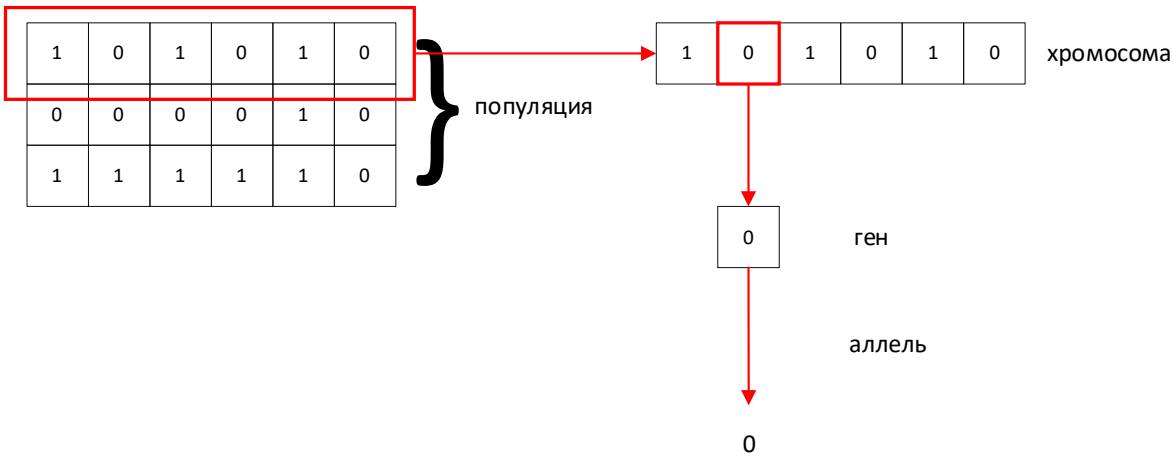


Рисунок 2.1 – Структура популяции

На каждой итерации генетический алгоритм обновляет популяцию путем создания новых особей и уничтожения худших. Некоторым, обычно случайным, образом создаётся множество генотипов начальной популяции. Они оцениваются с использованием функции приспособленности, в результате чего с каждым генотипом ассоциируется определённое значение - приспособленность, которое определяет насколько хорошо фенотип, им описываемый, решает поставленную задачу. Функция приспособленности – вещественная или целочисленная функция одной или нескольких переменных, подлежащая оптимизации в результате работы генетического алгоритма, направляет эволюцию в сторону оптимального решения. К функции приспособленности применяются следующие требования:

1. Функция должна быть адекватно заданной. Это означает, что распределение значений должно совпадать с распределением реального качества решений.

2. Функция должна иметь разнообразный рельеф, без больших «плоских» участков, так как иначе, несмотря на то что решения будут различаться, они будут иметь одинаковую оценку, а значит при работе алгоритма не будет возможности выбрать лучшее решение, направление дальнейшего развития. Эта проблема еще упоминается как «проблема поля для гольфа», где все пространство абсолютно одинаково, за исключением лишь одной точки, и является оптимальным решением - в этом случае алгоритм просто остановится или поиск решения будет происходить совершенно случайным образом.

3. Функция приспособленности должна требовать минимум ресурсов. Поскольку это наиболее часто используемая деталь алгоритма, она оказывает существенное влияние на его скорость работы [29].

Генерация новых особей происходит на основе моделирования процесса размножения с помощью оператора скрещивания. Порождающие особи называются родителями, а порожденные - потомками. Выбор родителей (пары кодовых строк) для скрещивания выполняется различными способами, наиболее известные из которых метод колеса рулетки, когда хромосома выбирается случайным образом, а вероятность ее выбора зависит от ее приспособленности [12], селекция отсечением, когда гены выбираются на основе значения функции приспособленности, превышающей заданный порог, турнирная селекция, когда из популяции, содержащей N хромосом, выбирается случайным образом t хромосом (тур), и лучшая хромосома из тура попадает в родительскую популяцию. Родительская пара, как правило, порождает пару потомков.

Изменение особей осуществляется за счет работы оператора мутации, применяемого к случайно выбранным потомкам за счет изменения случайного выбранного гена (генов). Это позволяет алгоритму выходить из локальных

экстремумов. При недостаточном значении вероятности мутации и попадания алгоритма в локальный экстремум становится возможной преждевременная сходимость генетического алгоритма (также говорят о вырождении популяции). Преждевременная сходимость популяции может быть исправлена путем:

– изменения стратегии выбора родительских пар для скрещивания;

– отслеживания появления в популяции идентичных особей и их удаления;

– использования сильно разрушающего оператора кроссовера;

– увеличения вероятности мутации.

Критерием остановки работы генетического алгоритма может быть одно из следующих событий:

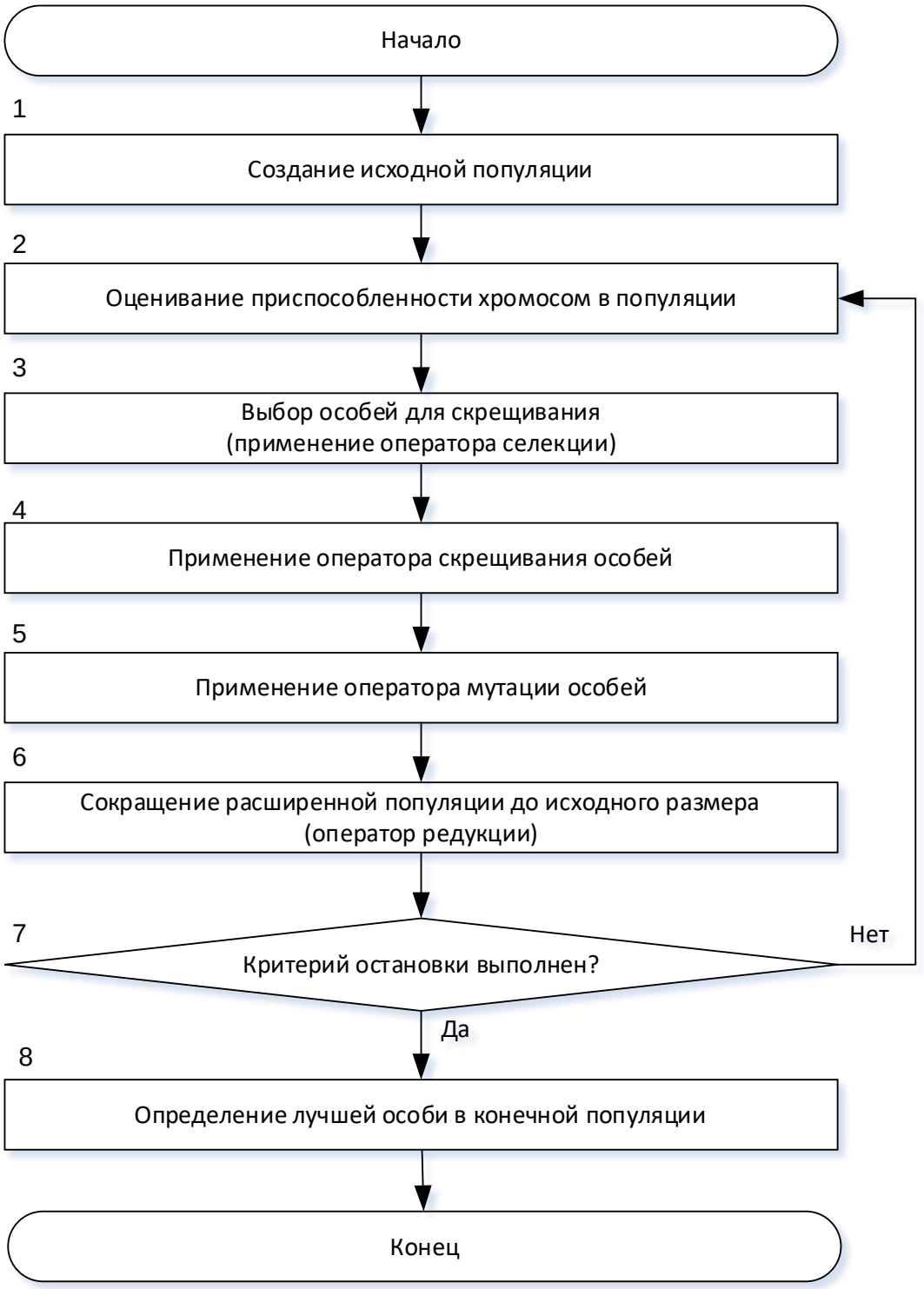
– сформировано заданное число поколений;

– исчерпано время, отведенное на эволюцию;

– популяция достигла заданного качества (значение критерия одной (нескольких, всех) особей превысило заданный порог);

– достигнут некоторый уровень сходимости (особи в популяции стали настолько подобными, что дальнейшее их улучшение происходит чрезвычайно медленно);

Блок-схема генетического алгоритма представлена на странице 38.



					ЧГУ.Д.ВКР.270404.00.00.21 ПЗ		
					Разработка системы автоматического развертывания микросервисных приложений в облачной инфраструктуре на основе алгоритма комбинаторной оптимизации	Лит.	Масса / Масштаб
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Смирнов А.Б.				Лист 38	Листов
Провер.		Маслов Е.А.					
Н.контр.		Смыслова А.Л.			Генетический алгоритм	ЧГУ 1УТСм-01-21оп	
Утверд.		Смыслова А.Л.					

На данный момент нет возможности выбрать наиболее подходящий алгоритм для решения задачи распределения сервисов по серверам. Вышеописанные алгоритмы необходимо реализовать и сравнить результаты, полученные в результате их работы.

2.3 Разработка симулятора сетевой инфраструктуры

При проектировании системы оптимизации автоматического развертывания микросервисных приложений в облачной инфраструктуре, необходимо создать симулятор данной системы, состоящий из программного обеспечения для хранения и получения данных об облачной инфраструктуре и программного обеспечения, целью которого является решение задачи распределения сервисов на сервера. Симулятор - система, которая ведет себя подобно настоящей системе, но реализована совершенно по-другому. Симулятор обеспечивает базовое поведение системы, но может не обязательно соответствовать всем правилам моделируемой системы. Симулятор облачной инфраструктуры необходим для моделирования работы системы распределения микросервисов в данной инфраструктуре. Программная реализация алгоритмов распределения сервисов на сервера будет описана в следующих пунктах.

Критериями выбора технологии реализации симулятора облачной инфраструктуры являются:

1. Возможность работы приложения в локальной вычислительной сети.
2. Использование стандартизованных форматов хранения и обмена информацией.
3. Кроссплатформенность.

Исходя из вышеперечисленных требований была выбрана модульная платформа для разработки программного обеспечения с открытым исходным кодом .NET. Структура симулятора представлена на рис. 2.2.

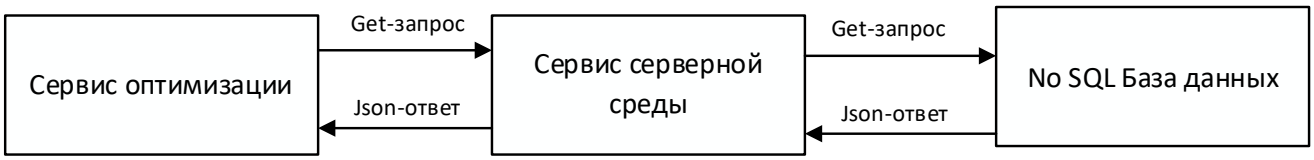
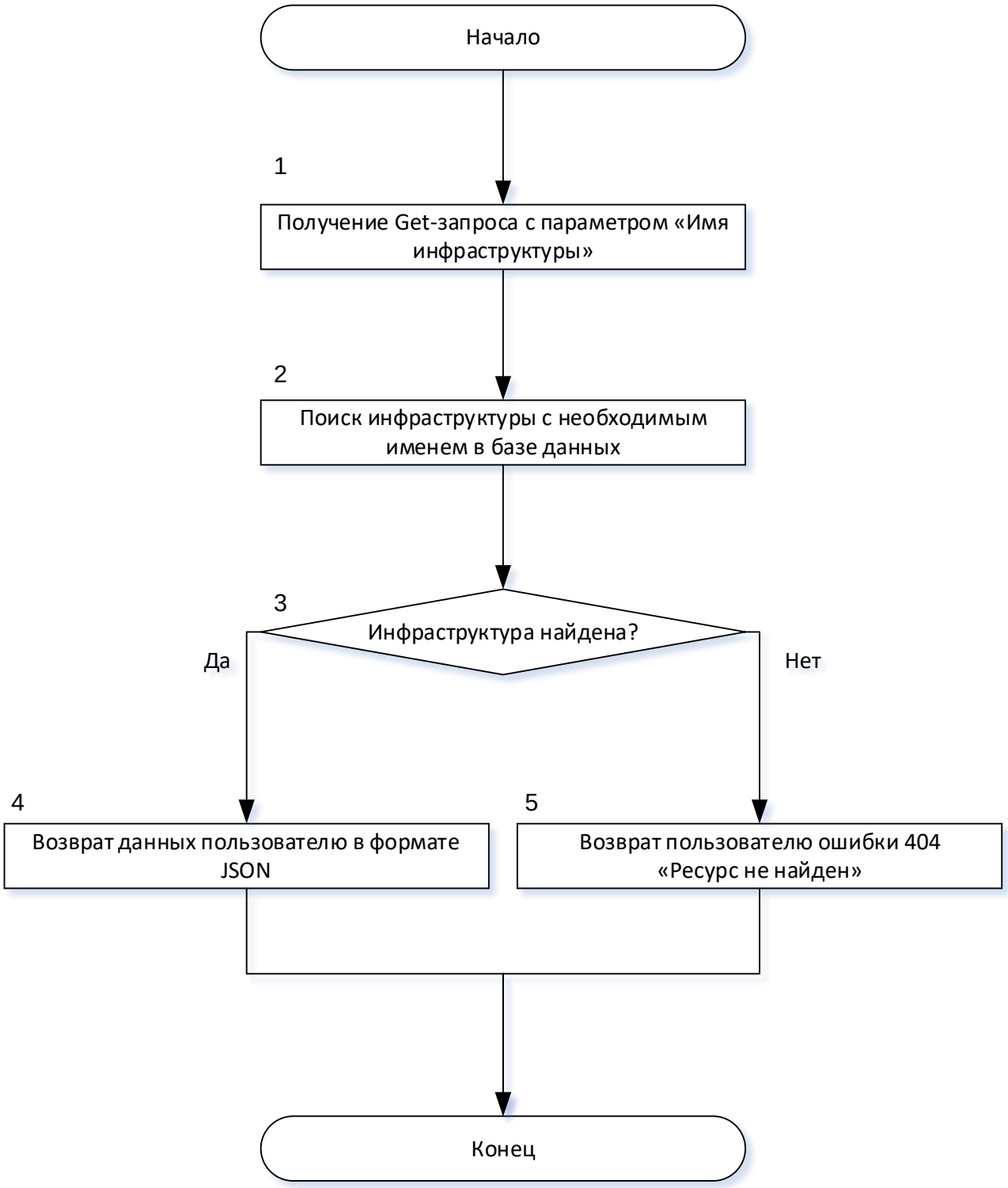


Рисунок 2.2 – Структура симулятора облачной инфраструктуры

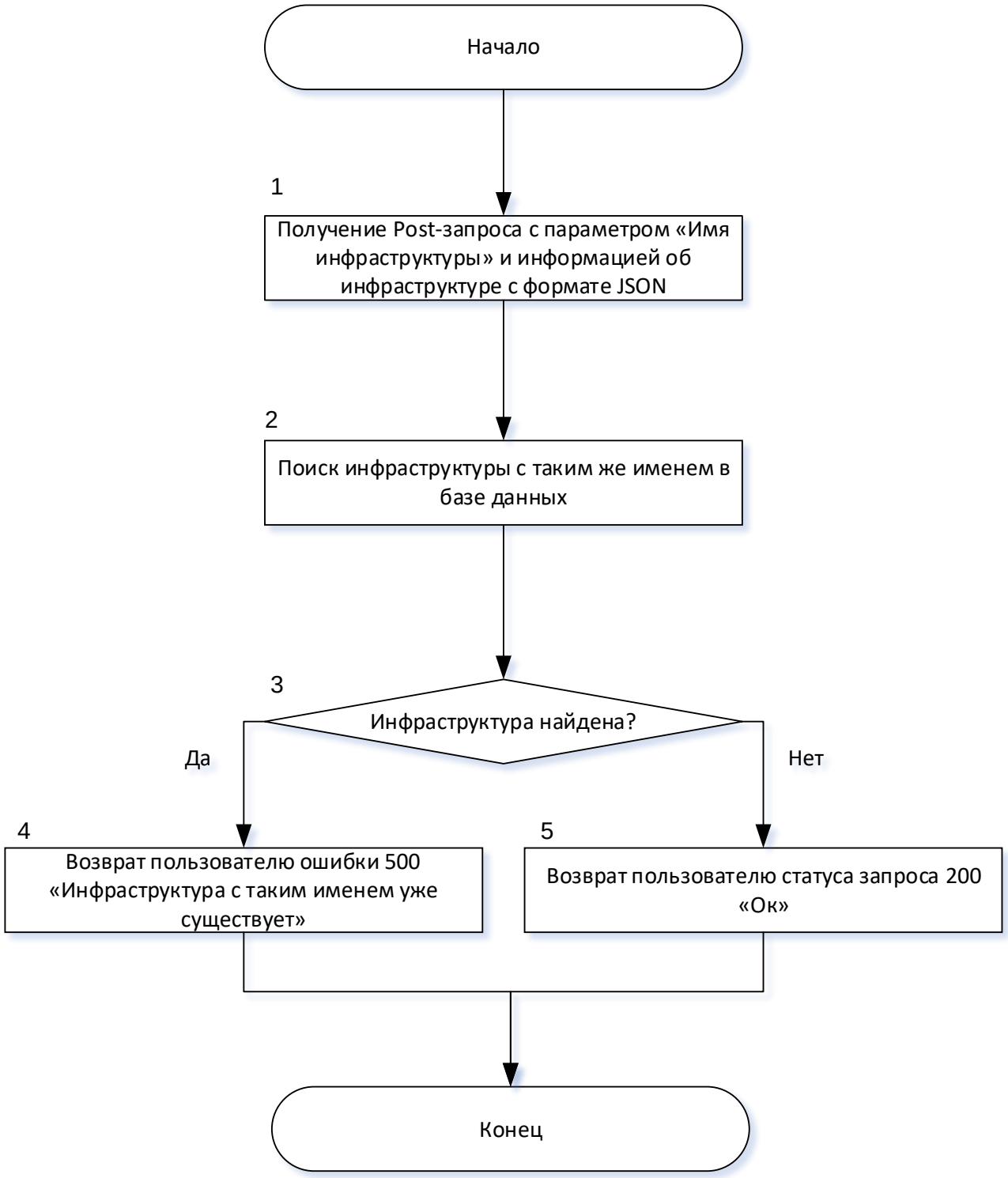
Симулятор облачной инфраструктуры – это веб-приложение, созданное на платформе .NET с использованием языка программирования C#. Для хранения данных используется система управления базами данных класса NoSQL с открытым исходным кодом Redis, работающая со структурами данных типа «ключ — значение». Входные и выходные данных представлен в формате JSON в приложении А.

Для записи и получения данных используется веб-api. При запросе данных об облачной инфраструктуре приложение возвращает информацию о каждом сервере и развернутом на нем программного обеспечении. Эти данные являются входными при работе алгоритма хостирования сервисов на сервера. Также, в процессе анализа работы вышеописанных алгоритмов потребуется вносить изменения в симулируемую инфраструктуру, для этого симулятор имеет API для записи данных. Формат входных данных - JSON. На странице 41 представлена блок-схема алгоритма запроса данных у симулятора инфраструктуры.



					ЧГУ.Д.ВКР.270404.00.00.21 ПЗ		
					Разработка системы автоматического развертывания микросервисных приложений в облачной инфраструктуре на основе алгоритма комбинаторной оптимизации	Лит.	Масса / Масштаб
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Смирнов А.Б.					
Провер.		Маслов Е.А.				Лист 41	Листов
Н.контр.		Смыслова А.Л.			Алгоритм запроса данных у симулятора	ЧГУ 1УТСм-01-21оп	
Утверд.		Смыслова А.Л.					

На странице 43 представлена блок-схема алгоритма запроса данных у симулятора инфраструктуры. Пользователь или сервис может послать Get-запрос с параметром «Имя инфраструктуры» симулятору инфраструктуры и в ответ получить либо данные в формате JSON, либо ошибку 404 «Ресурс не найден». На странице 46 представлена блок-схема алгоритма записи данных в базу данных симулятора инфраструктуры. Пользователь или сервис может послать Post-запрос с параметром «Имя инфраструктуры» и информацией об инфраструктуре в формате JSON симулятору инфраструктуры и в ответ получить либо статус с кодом 200 «Ок», либо ошибку 500 «Инфраструктура с таким именем уже существует».



					ЧГУ.Д.ВКР.270404.00.00.21 ПЗ		
					Разработка системы автоматического развертывания микросервисных приложений в облачной инфраструктуре на основе алгоритма комбинаторной оптимизации	Лит.	Масса / Масштаб
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Смирнов А.Б.					
Провер.		Маслов Е.А.				Лист 43	Листов
					Алгоритм записи данных в симулятор	ЧГУ 1УТСм-01-21оп	
Н.контр.		Смыслова А.Л.					
Утверд.		Смыслова А.Л.					

Таким образом, симулятор сетевой инфраструктуры подготовлен и необходимо реализовать вышеописанные алгоритмы.

2.4 Реализация алгоритма «наилучший подходящий с упорядочиванием»

Для решения задачи распределения сервисов по серверам с помощью эвристических алгоритмов будем использовать offline-алгоритм «BFD» или «Наилучший подходящий с упорядочиванием». Выбор данного алгоритма обусловлен тем, что в процессе упаковки производится поиск наиболее заполненных серверов, куда может быть размещен каждый сервис, т.е. при использовании данного алгоритма мы будем стремиться использовать наименьшее количество серверов для размещения всех сервисов.

При решении задачи с использованием данного алгоритма сервисы сортируются по убыванию весов r, h, f и применяется алгоритм «наилучший подходящий» (BF). Список серверов остается неотсортированным. В произвольном порядке размещаем сервисы по следующему правилу: первый сервис размещаем на первом сервере. На k-м шаге размещаем k-й сервис. Находим частично заполненные сервера, удовлетворяющие следующим ограничениям по формуле 2.5:

$$\begin{gathered} \dot{h} \to min, \\ \dot{r} \to min, \\ \dot{f} \to min, \\ \dot{h} \geq \dot{H}_k, \\ \dot{r} \geq \dot{R}_k, \\ \dot{f} \geq \dot{C}_k, \end{gathered} \tag{2.5}$$

Т.е. сервера, где достаточно свободного места для k-го сервиса и выбираем среди них наиболее заполненный. Если таких нет, то берем новый пустой контейнер и помещаем k-й предмет в него.

Проведем эксперимент, в ходе которого запустим алгоритм BFD на тестовой выборке (Приложение А), которая содержит 5 серверов и 50 сервисов. Усредненные результаты представим в таблице после 50 запусков алгоритма.

Результаты работы алгоритма получены с помощью эмулятора инфраструктурной среды на первой тестовой выборке (5 серверов и 50 сервисов) и представлены в таблице 2.1 и графике (рис. 2.3):

Таблица 2.1 – Зависимость количества свободных серверов от количества размещаемых сервисов

Алгоритм BFD		
Кол-во сервисов	**Кол-во свободных серверов**	**% свободных серверов**
0	5	100
5	3	60
10	2	40
15	1	20
20	0	0
25	0	0
30	0	0
35	0	0
40	0	0
45	0	0
50	0	0

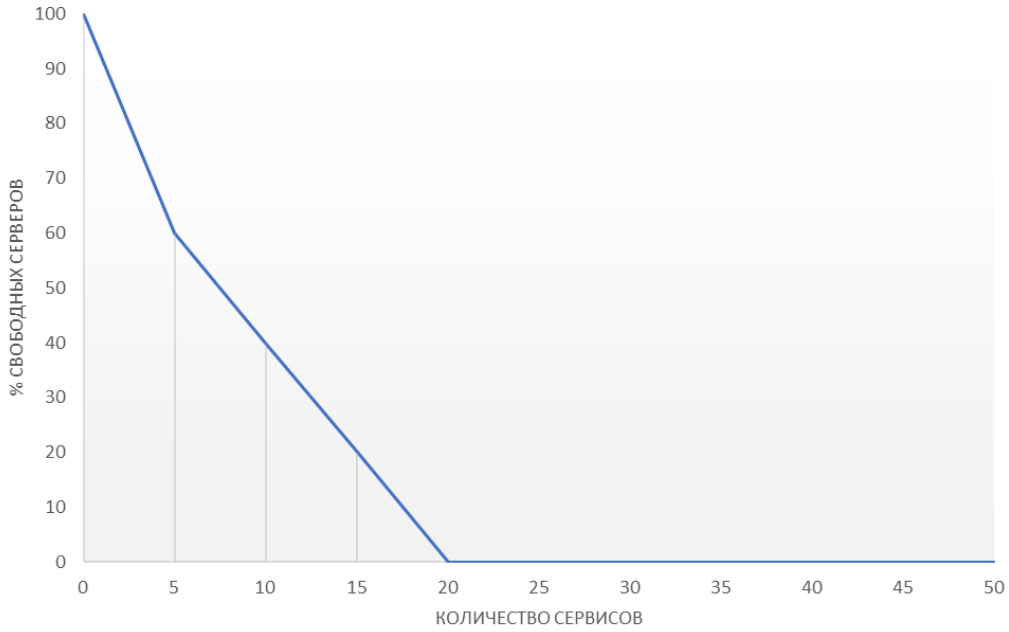


Рисунок 2.3 – График зависимости количества свободных серверов от количества размещаемых сервисов

В таблице 2.2 и графике (рис. 2.4) представлены результаты работы алгоритма, полученные с помощью эмулятора инфраструктурной среды на второй тестовой выборке (10 серверов и 50 сервисов).

Таблица 2.2 – Зависимость количества свободных серверов от количества размещаемых сервисов

Алгоритм BFD		
Кол-во сервисов	**Кол-во свободных серверов**	**% свободных серверов**
0	10	100
5	7	70
10	6	60
15	5	50
20	5	50
25	4	40
30	4	40
35	2	20
40	2	20
45	0	0
50	0	0

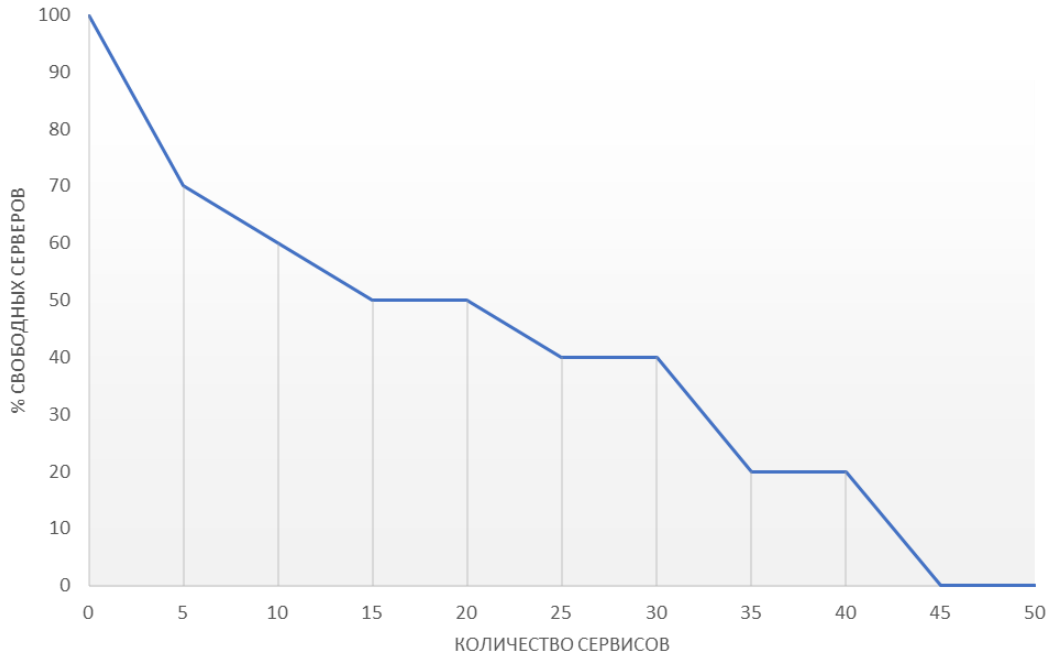


Рисунок 2.4 – График зависимости количества свободных серверов от количества размещаемых сервисов

Результаты работы алгоритма BFD, представленные в таблице будут использованы при сравнении с результатами работы других алгоритмов, описание реализации которых будет ниже.

2.5 Реализация генетического алгоритма

Решение задачи распределения сервисов на сервера с помощью генетического алгоритма представляет собой итерационную процедуру, позволяющую получать улучшенное решение на каждом шаге, до тех пор, пока не будет выполнен критерий остановки.

На каждой итерации рассматриваются сразу несколько альтернативных вариантов (особей) решения задачи. Перед запуском генетического алгоритма происходит первоначальное создание хромосом, которое заключается в случайном распределении сервисов на серверах. При распределении сервисов учитывается только ограничение на операционную систему, т.е. для каждого

сервиса $n_j \in \{N_1, \dots N\ \}$ находится случайный элемент из множества серверов $m_i \in \{M_1, \dots M\ \}$ при ограничении на тип операционной системы $\forall\, n \in \{N_1, \dots N'\}$, $m_{i_t} = n_{j_t}$. На данном этапе сервисы могут быть распределены на серверах так, что свободное место на сервере имеет отрицательное значение. Позднее, в процессе работы алгоритма это будет исправлено.

При создании исходной популяции выбирается размер популяции хромосом [40]. Размер популяции хромосом выбран подбором. Рекомендуемый размер популяции варьируется в зависимости от задачи и в среднем составляет 30-50 хромосом [20]. Однако, для задачи, решаемой в данной работе, используется меньший размер популяции. В таблице 2.2 показано, что максимальная заполненность ПЗУ r, ОЗУ h и ЦП f у серверов колеблется от 89 до 98, от 92 до 99 и от 92 до 99 соответственно, при увеличении числа особей в популяции с 3 до 13, однако, начиная с числа особей 9 результаты работы алгоритма практически не меняются, а лишь растет время работы алгоритма.

Таблица 2.3 – Зависимость показателей работы генетического алгоритма от количества особей в популяции.

Кол-во особей в популяции	**Время работы алгоритма**	**Макси-мальная заполнен-ность HDD, %**	**Макси-мальная заполнен-ность RAM, %**	**Макси-мальное использова-ние CPU, %**
3	0,859	89	92	92
5	1,012	92	94	93
7	1,514	96	94	95
9	1,921	98	99	99
11	2,122	98	99	99
13	2.588	98	99	99

Таким образом, подходящее число особей в популяции 9.

Для того, чтобы понять, двигается ли алгоритм в сторону лучшего решения, необходимо ввести критерии, по которым будет производиться сравнение

решений. Критерии будут выяснены путем коллективной оценки коллективного мнения экспертов. Исходные данные, по которым будет проведен опрос экспертов, приведены в таблице 1 приложения Б. Анкета для проведения опроса экспертов содержит перечень критериев, влияющих на итоговый результат работы генетического алгоритма и соответствующие критериям вербальные и числовые оценки их важности. Задача экспертов состоит в оценке данных критериев по предложенным шкалам.

При подборе экспертов применен метод шара [16]. Один эксперт, являющийся наиболее уважаемым специалистом, рекомендует ряд других. Для проведения выбрано 5 экспертов. Результаты экспертной оценки представлены в таблице 2 приложения Б. Проанализируем данные результаты. Согласованность мнения экспертов можно оценивать по величине коэффициента конкордации, рассчитываемому по формуле 2.6:

$$W = \frac{12S}{n^2(m^3 - m)} \quad (2.6)$$

где S - сумма квадратов отклонений всех оценок рангов каждого объекта экспертизы от среднего значения, n -число экспертов, m - число объектов экспертизы.

Коэффициент конкордации изменяется в диапазоне 0<W<1, причем 0 - полная несогласованность, 1 - полное единодушие.

Коэффициент конкордации по данным таблицы 2 приложения Б составляет 0,006, что говорит о согласованности мнения экспертов.

При решении задачи распределения сервисов на сервера необходимо занимать как можно большее свободное пространство сервера, соблюдая ограничения по свободному месту на постоянном и оперативном запоминающих устройствах. Представим критерии и их усредненные оценки, определенные в ходе опроса экспертов в таблице 2.4.

Таблица 2.4 – Критерии сравнения решений

Критерий	**Метрика**	**Важность**	**Коэффициент важности**
Количество не занятых серверов	Больше - лучше	Высокая	2,56
Количество северов с отрицательным значением свободной памяти	Меньше - лучше	Высокая	2,82
Количество северов с положительным значением свободной памяти	Больше - лучше	Средняя	1,4
Количество северов с отрицательным значением свободной оперативной памяти	Меньше - лучше	Высокая	2,8
Количество северов с положительным значением свободной оперативной памяти	Больше - лучше	Средняя	1,24

Кроме критериев, определенных в таблице 2.4, на качество решения также влияет показатель разброса количества наиболее занятых и наименее занятых серверов. Чем выше дисперсия, тем ближе решение к оптимальному. Данный показатель можно получить, рассчитав по каждому серверу отношения полезного объема, занимаемого сервисами, к общему объему сервера, а далее, получить дисперсию данных показаний.

Расчет отношения R занятого пространства к общему на примере трехмерного контейнера с занятым объемом $V_{\text{зан}}$ и общим объемом $V_{\text{общ}}$ рассчитывается по формуле 2.7:

$$R = \frac{V_{\text{зан}}}{V_{\text{общ}}} \tag{2.7}$$

В случае с сервером, отношение занятого пространства сервера к общему объему может быть рассчитано по формуле 2.8:

$$R = \frac{\ddot{r} - \dot{r}}{\ddot{r}} \cdot k_r + \frac{\ddot{h} - \dot{h}}{\ddot{h}} \cdot k_h + \frac{100 - \dot{f}}{\ddot{f}} \cdot k_f, \tag{2.8}$$

где k_r – коэффициент важности индекса заполненности постоянной памяти сервера, k_h – коэффициент важности индекса заполненности оперативной памяти сервера, k_f – коэффициент важности индекса использования процессорного времени сервера.

Представим коэффициенты заполненности сервера и их усредненные оценки, определенные в ходе опроса экспертов, в таблице 2.5.

Таблица 2.5 – Коэффициенты заполненности сервера

Коэффициент	**Метрика**	**Важность**	**Коэффициент важности**
Коэффициент занятости HDD	Больше - лучше	Средняя	0,27
Коэффициент занятости RAM	Больше - лучше	Высокая	0,41
Коэффициент занятости CPU	Больше - лучше	Высокая	0,45

Экспертные оценки получены с использованием того же метода, как и в случае с оценкой критериев из таблицы 2.4. Анкета опроса экспертов и его результаты представлены в приложении Б (таблица 3 и 4). Коэффициент

конкордации по данным таблицы 4 приложения Б составляет 0,002, что говорит о согласованности мнения экспертов.

Формула расчета дисперсии (2.9) в теории вероятностей имеет вид:

$$RD(X) = \sigma^2 = M[X - M(X)]^2 \quad (2.9)$$

то есть дисперсия — это математическое ожидание отклонений от математического ожидания. На практике при анализе выборок математическое ожидание, как правило, не известно. Поэтому вместо него используют оценку – среднее арифметическое. Расчет дисперсии произведем по формуле 2.10:

$$s = \frac{\sum_{i=1}^{n}(R - \bar{R})^2}{n} \quad (2.10)$$

где s –дисперсия, R – отдельные значения отношения занятого пространства к общему для каждого сервера, $\bar{R}$– среднее арифметическое по выборке.

В конечном итоге, формула расчета значения функции приспособленности 2.11 для каждой хромосомы из популяции задана таким образом, чтобы характеризовать параметр, по которому проводится поиск оптимального решения.

$$F = S_f \cdot k_1 + H_n \cdot k_2 + H_p \cdot k_3 + R_n \cdot k_4 + R_p \cdot k_5 + s \quad (2.11)$$

где S_f - количество не занятых серверов, H_n – количество северов с отрицательным значением свободной памяти, H_p - количество северов с положительным значением свободной памяти, R_n - количество северов с отрицательным значением свободной оперативной памяти, R_p - количество северов с положительным значением свободной оперативной памяти, s – дисперсия, k_1 = 2.56, k_2 = 2,82, k_3 = 1,4, k_4 = 2,8, k_5 = 1,24 - коэффициенты поощрения или штрафа (таблица 2.4).

В качестве поощрения за повышение количества не занятых серверов используется коэффициент k_1, за превышение свободного места на диске и оперативной памяти назначается штраф k_2 и k_4, а за освобождение пространства на диске и оперативной памяти назначается поощрение k_3 и k_5.

Функция приспособленности отвечает основным требованиям, а именно:

– распределение значений функции совпадает с распределением реального качества решений. Чем лучше решение, тем значение функции приспособленности выше;

– не имеет плоских участков;

– не требует больших вычислительных ресурсов.

Для создания новой популяции используется селекция усечением, т.е. отбираются особи с наивысшим значением функции приспособленности.

Метод скрещивания – многоточечное скрещивание. На этом этапе извлекается список сервисов, расположенных на первой родительской хромосоме $N_1(Cr_1)$. Элементы этого списка совпадают с элементами списка на второй родительской хромосоме $N_1(Cr_1) = N_2(Cr_2)$.

Далее, итеративно выполняется перебор элементов списка сервисов, сервис с четным индексом располагается в хромосоме-потомке на том же сервере, на котором он расположен на первой родительской хромосоме $\sum_{i=2}^{i} N_i \epsilon\, Cr_1$, где $i\ mod\ 2 = 0$ и тип ОС $N_i \in Cr_1$. Сервис с нечетным индексом располагается на хромосоме-потомке на том же сервере, на котором он расположен на второй родительской хромосоме $\sum_{i=1}^{i-1} N_i \epsilon\, Cr_2$ где $i\ mod\ 2 \neq 0$ и тип ОС $N_i \in Cr_2$. (рис. 2.5).

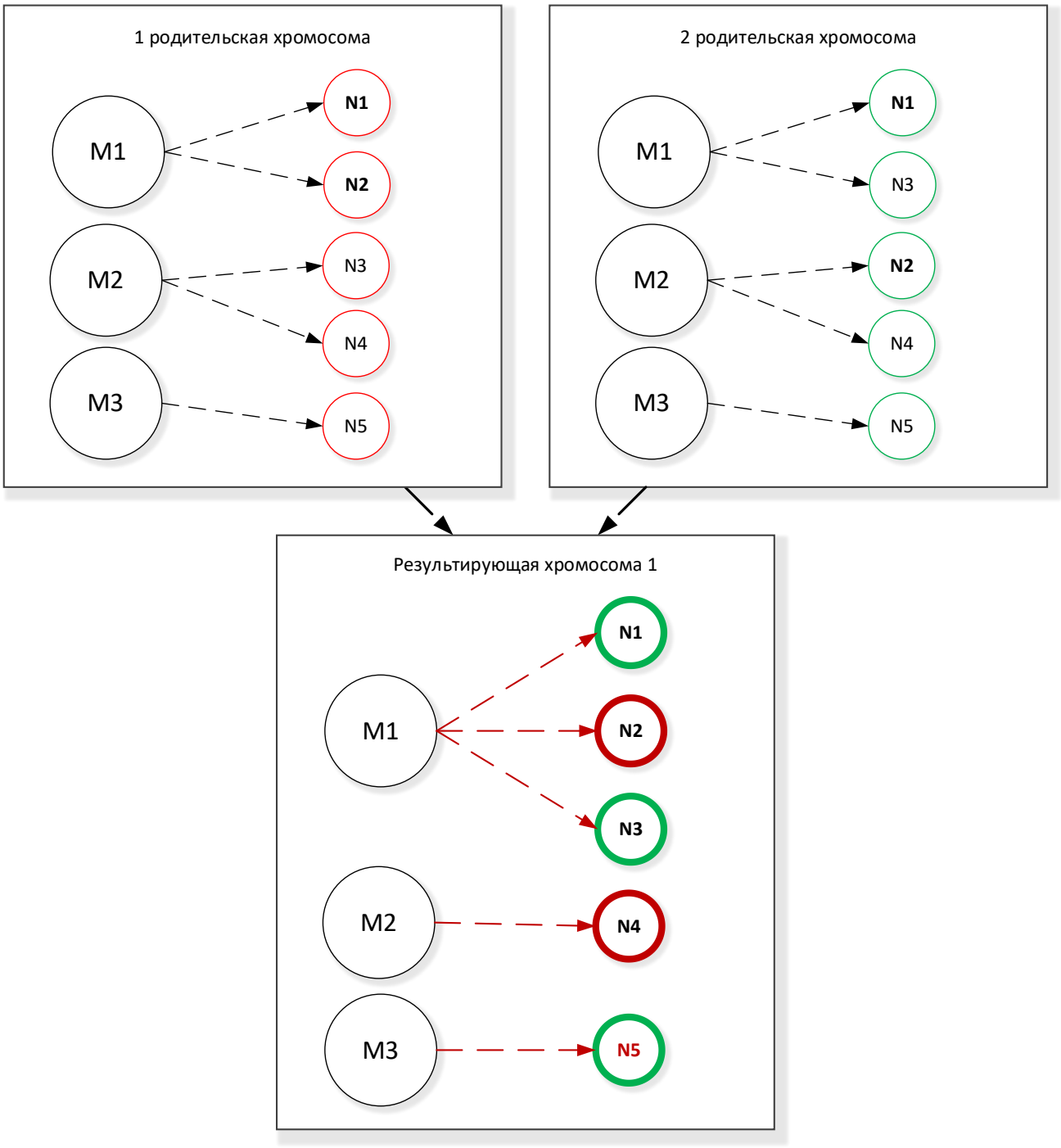


Рисунок 2.5 – Механизм скрещивания хромосом

Во второй хромосоме-потомке сервис с четным индексом располагается на том сервере, на котором он расположен на второй родительской хромосоме $\sum_{i=2}^{i} N_i \epsilon\, Cr_2,\ i\ mod\ 2 = 0$ и тип ОС $N_i \in Cr_2$., а сервис с нечетным индексом

располагается на том сервере, на котором он расположен на первой родительской хромосоме $\sum_{i=1}^{i-1} N_i \epsilon\, Cr_1$, где $i\ mod\ 2 \neq 0$ и тип ОС $N_i \in Cr_1$. (рис. 2.6).

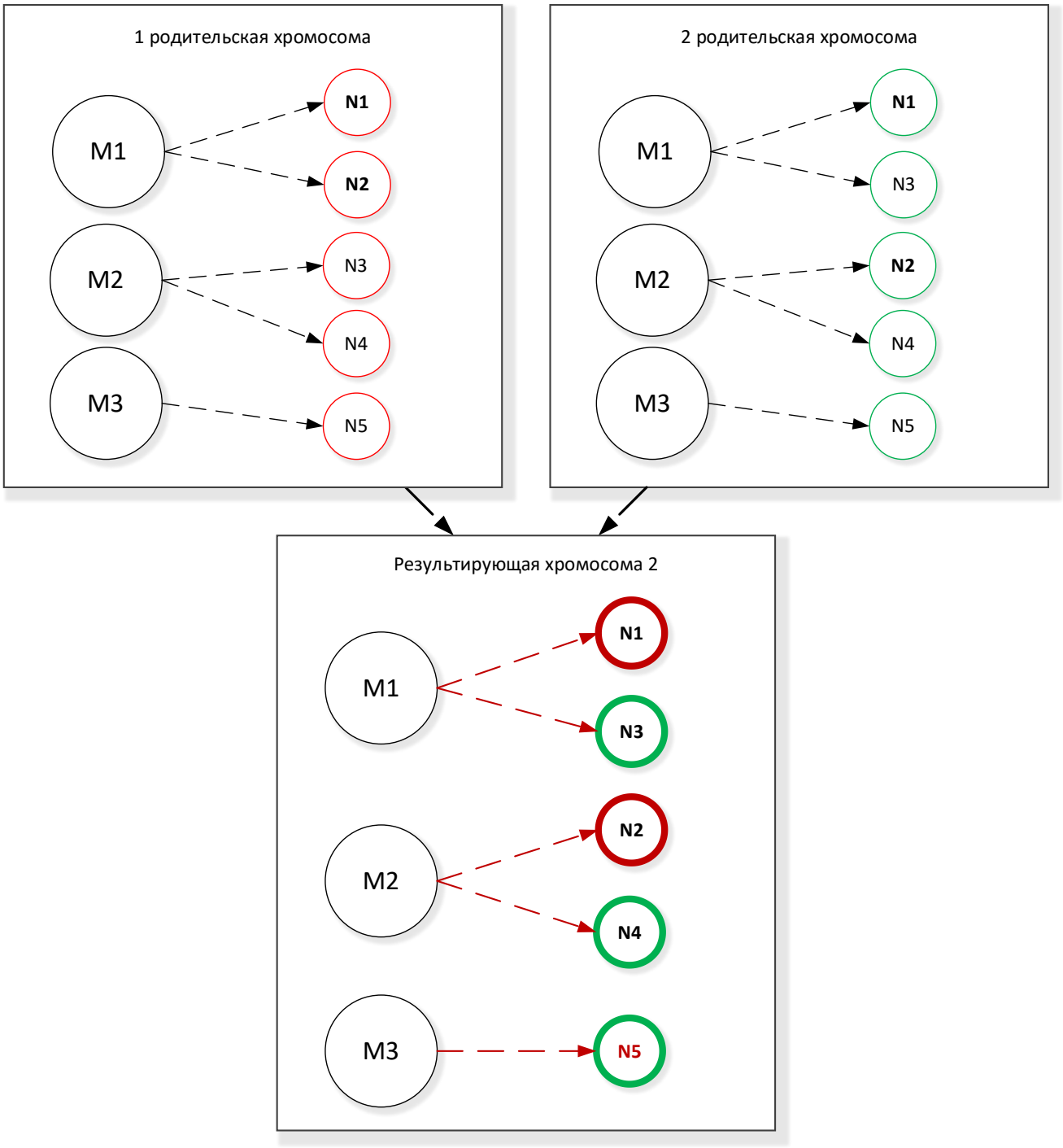


Рисунок 2.6 – Механизм скрещивания хромосом

При таком скрещивании возможен выход за пределы памяти и др. метрик, что продемонстрировано на рис. 2.7, но этот недостаток компенсируется при приближении решения к оптимальному.

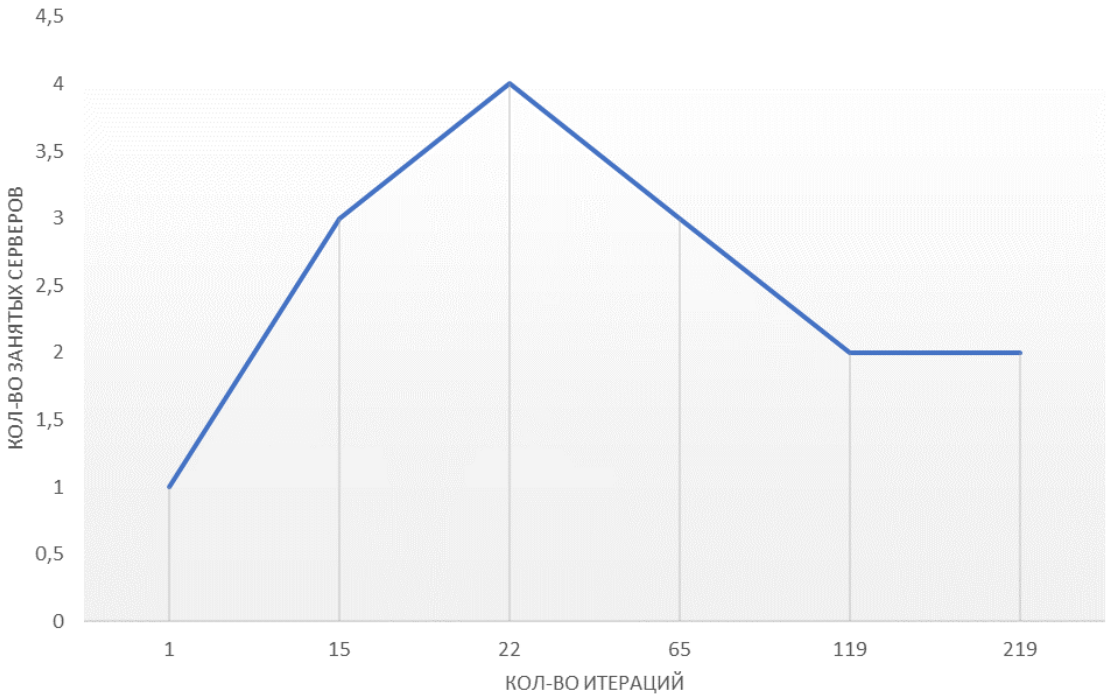


Рисунок 2.7 – Выход за пределы метрик

Рис. 2.7 показывает, как изменялось количество занятых серверов по мере схождения генетического алгоритма. Алгоритм запускался с использованием тестовой выборки из приложения А. На 1 итерации очевидно, что имеет место выход за пределы доступной памяти ПЗУ и ОЗУ и процессорного времени, т.к. все сервисы размещены на одном сервере.

После выполнения операции скрещивания выполняется операция мутации, т.е. случайного изменения полученных в результате скрещивания хромосом. Применительно к задаче размещения сервисов на серверах выбирается хромосома, в которой случайный сервис из случайного сервера и перемещается на другой случайный сервер. Ограничения по ОС соблюдаются.

Вероятность применения оператора мутации влияет на вырождаемость популяции и нахождение алгоритмом решения с наименьшей погрешностью. С целью определить погрешность работы алгоритма, произведено вычисление значений функции приспособленности решений, полученных в результате работы алгоритма с различными значениями вероятности мутации с использованием данных из тестовой выборки (приложение А). Данные приведены в таблице 2.6.

Таблица 2.6 – Результаты работы генетического алгоритма с различной вероятностью применения оператора мутации

Вероятность мутации	**Значение функции приспособленности**
0,05	-12,58
0,1	59,14
0,2	68,88
0,3	79,47
0,4	79,47
0,5	79,47
0,6	79,47
0,7	79,47
0,8	79,47
0,9	79,47
1	79,47

Согласно таблице 2.6 значение функции приспособленности алгоритма не меняется при значении вероятности мутации в диапазоне от 0,3 до 1. Примем значение вероятности мутации равным 0,3.

На рис. 2.8 приведен график зависимости значения функции приспособленности алгоритма от вероятности мутации.

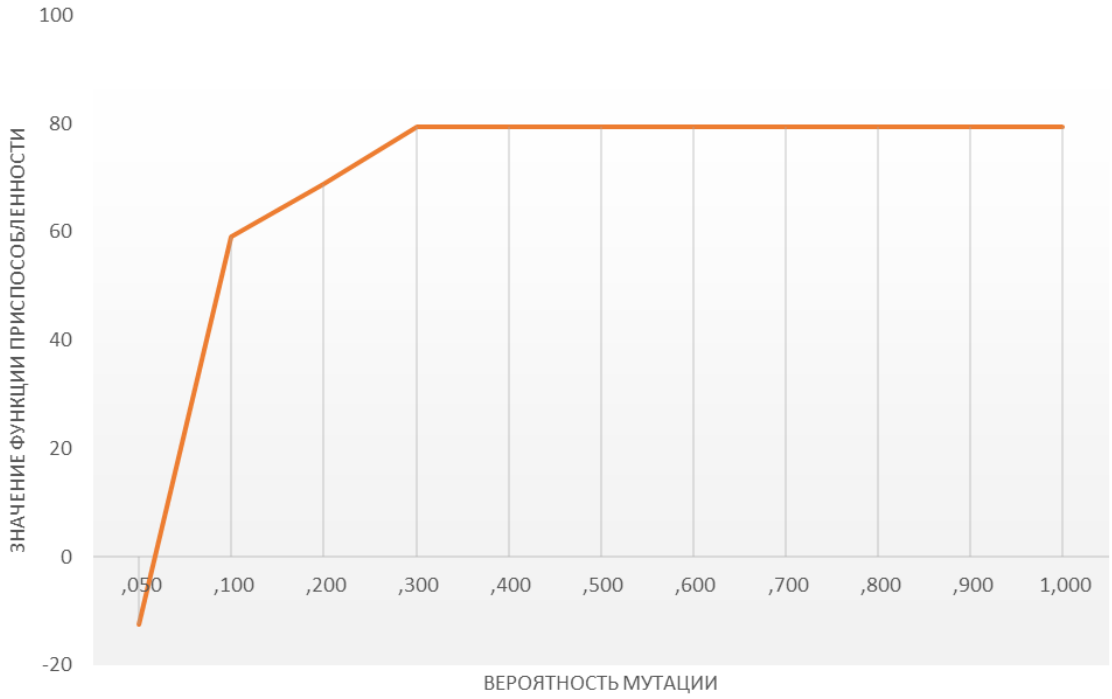


Рисунок 2.8 – Зависимость значений функции приспособленности решений алгоритма от вероятности мутации

Алгоритм останавливается, если функция приспособленности не изменяется на протяжении 100 итераций.

В процессе работы алгоритма не выявлено признаков преждевременной сходимости и вырождения популяции, все особи популяции различаются между собой.

В таблице 2.7 представлены результаты работы алгоритма на первой тестовой выборке (5 серверов и 50 сервисов).

Таблица 2.7 – Зависимость количества свободных серверов от количества размещаемых сервисов

Генетический алгоритм		
Кол-во сервисов	**Кол-во свободных серверов**	**% свободных серверов**
0	5	100
5	3	60
10	3	60
15	3	60
20	3	60
25	3	60
30	2	40
35	2	40
40	0	0
45	0	0
50	0	0

На рис. 2.9 представлен график зависимости количества свободных серверов от количества размещаемых сервисов.

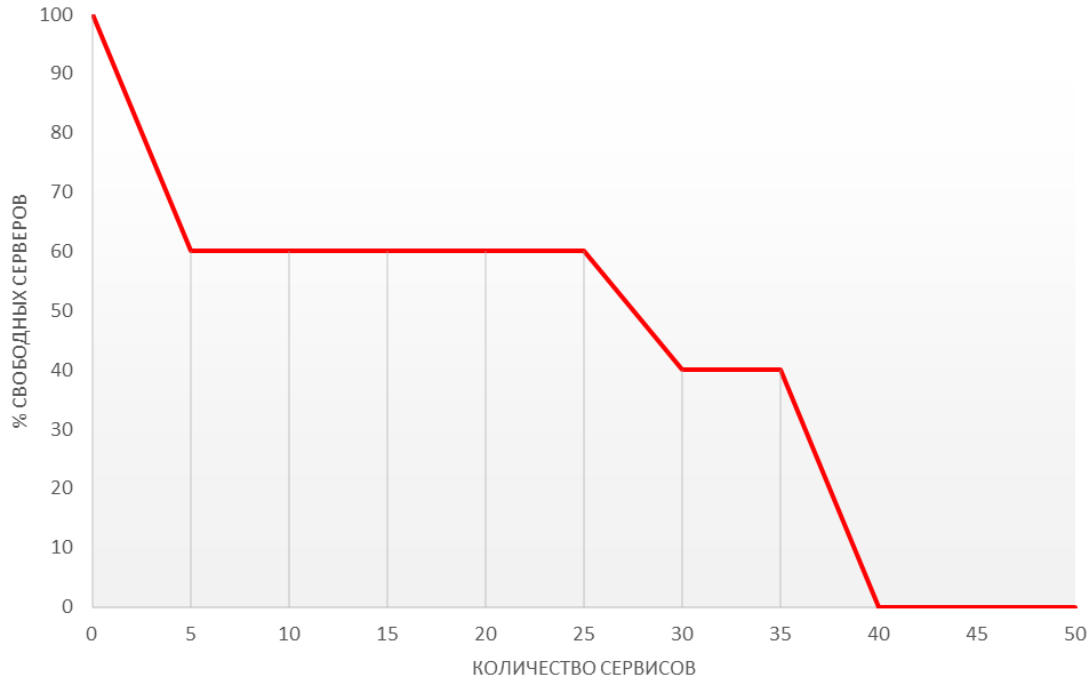


Рисунок 2.9 – График зависимости количества свободных серверов от количества размещаемых сервисов

В таблице 2.8 представлены результаты работы алгоритма на второй тестовой выборке (10 серверов и 50 сервисов).

Таблица 2.8 – Зависимость количества свободных серверов от количества размещаемых сервисов

Генетический алгоритм		
Кол-во сервисов	**Кол-во свободных серверов**	**% свободных серверов**
0	10	100
5	8	80
10	8	80
15	7	70
20	7	70
25	7	70
30	6	60
35	6	60
40	6	60
45	5	50
50	5	50

На рис. 2.10 представлен график зависимости количества свободных серверов от количества размещаемых сервисов.

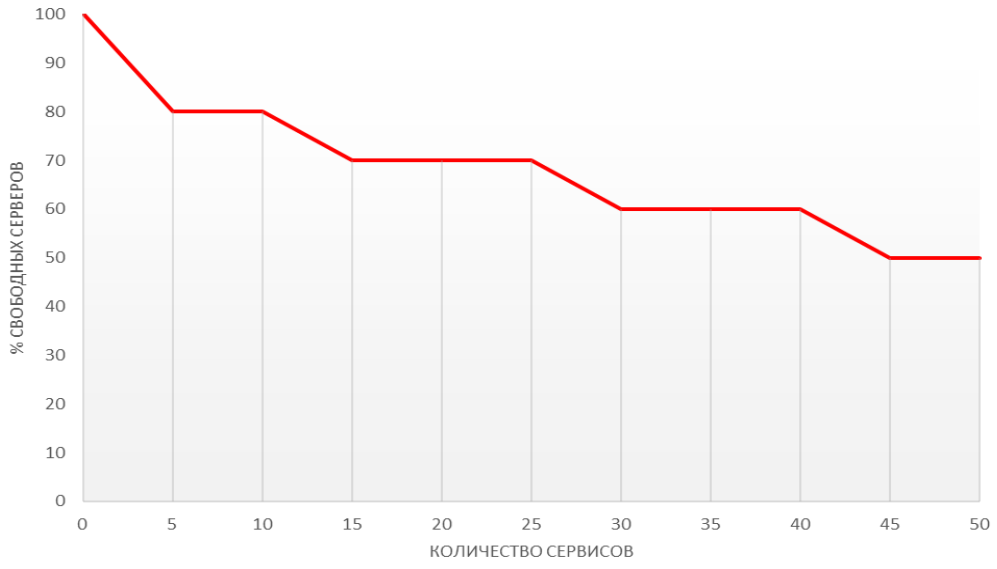


Рисунок 2.10 – График зависимости количества свободных серверов от количества размещаемых сервисов

2.6 Реализация алгоритма имитации отжига

Как сказано в пункте 2.2, метод отжига служит для поиска глобального минимума некоторой функции $f(x)$, заданной для x из некоторого пространства S, дискретного или непрерывного. Элементы множества S представляют собой состояния воображаемой физической системы (энергетические уровни) – в нашем случае состоянием будет вариант распределения сервисов на сервера. В данном случае энергия системы E это значение функции 2.12:

$$E = -(S_f \cdot k_1 + H_n \cdot k_2 + H_p \cdot k_3 + R_n \cdot k_4 + R_p \cdot k_5 + s), \qquad (2.12)$$

где S_f - количество не занятых серверов, H_n – количество северов с отрицательным значением свободной памяти, H_p – количество северов с положительным значением свободной памяти, R_n – количество северов с отрицательным значением свободной оперативной памяти, R_p – количество северов с положительным значением свободной оперативной памяти, s – показатель разброса количества наиболее занятых и наименее занятых серверов, а коэффициенты k_1 = 2.56, k_2 = 2,82, k_3 = 1,4, k_4 = 2,8, k_5 = 1,24 – коэффициенты штрафа и поощрения, определенные экспертным путем в пункте 2.5.

В данном случае E рассчитывается так же, 2.5 при расчете фитнесс функции генетического алгоритма, за тем исключением, что значение E берется с отрицательным знаком.

В каждый момент предполагается заданной температура системы T, уменьшающаяся с течением времени. При уменьшении температуры T, следующее состояние системы выбирается в соответствии с порождающим семейством вероятностных распределений $g(x, T)$, которое при фиксированных x и T задает новое местоположение случайного элемента из множества сервисов $n_j \in N$ на случайном сервере $m_j \in M$ в пространстве S. После генерации нового состояния $x' = G(x, T)$ система с вероятностью $h(\Delta E, T)$ переходит к следующему шагу в состояние x', в противном случае, процесс генерации x' повторяется. Здесь

ΔE означает приращение функции энергии, т.е. разность значений функции энергии текущего и предыдущего состояний $\Delta E = E - E'$. Если $\Delta E < 0$, то вероятность перехода в новое состояние считается равной 1. Таким образом, если новое состояние дает лучшее значение оптимизируемой функции, то переход произойдет в любом случае. Если же $\Delta E > 0$, то переход осуществляется с вероятностью $h(\Delta E, T) = \exp(-\Delta E / T)$). Понижение температуры осуществляется по закону Больцмановского отжига $T(k) = \frac{T_0}{\ln(1+k)}, k > 0$ [36].

В таблице 2.9 представлены результаты работы алгоритма имитации отжига на первой тестовой выборке (5 серверов и 50 сервисов). На рис. 2.11 представлен график зависимости количества свободных серверов от количества размещаемых сервисов.

Таблица 2.9 – Зависимость количества свободных серверов от количества размещаемых сервисов

Алгоритм имитации отжига		
Кол-во сервисов	**Кол-во свободных серверов**	**% свободных серверов**
0	5	100
5	3	60
10	3	60
15	3	60
20	3	60
25	2	40
30	2	40
35	2	40
40	0	0
45	0	0
50	0	0

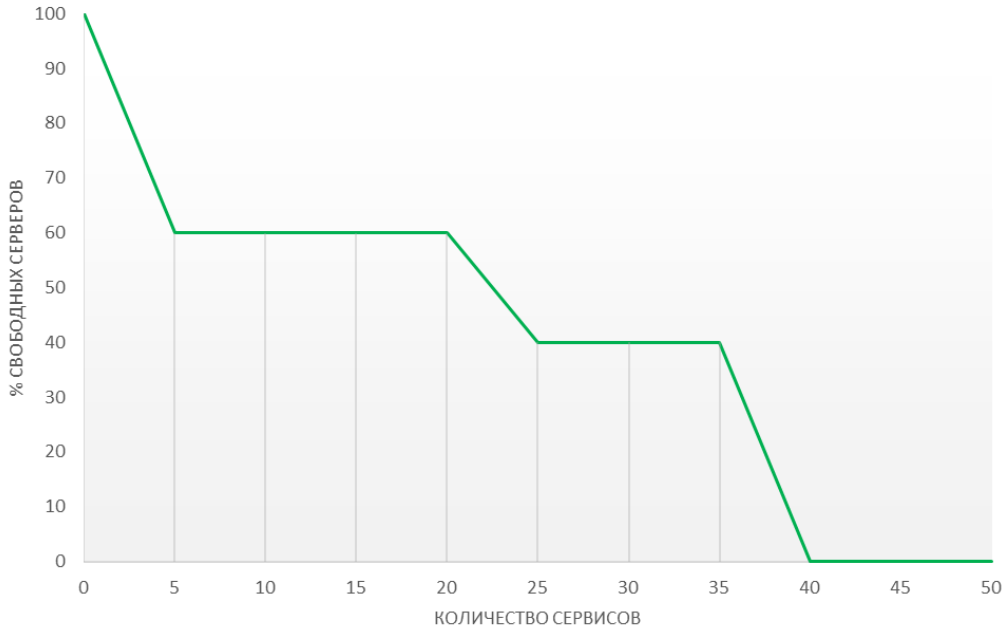


Рисунок 2.11 – Зависимость количества свободных серверов от количества размещаемых сервисов

В таблице 2.10 представлены результаты работы алгоритма имитации отжига на второй тестовой выборке (10 серверов и 50 сервисов).

Таблица 2.10 – Зависимость количества свободных серверов от количества размещаемых сервисов

Алгоритм имитации отжига		
Кол-во сервисов	**Кол-во свободных серверов**	**% свободных серверов**
0	10	100
5	8	80
10	8	80
15	7	70
20	7	70
25	6	60
30	6	60
35	6	60
40	6	60
45	5	50
50	5	50

На рис. 2.12 представлен график зависимости количества свободных серверов от количества размещаемых сервисов.

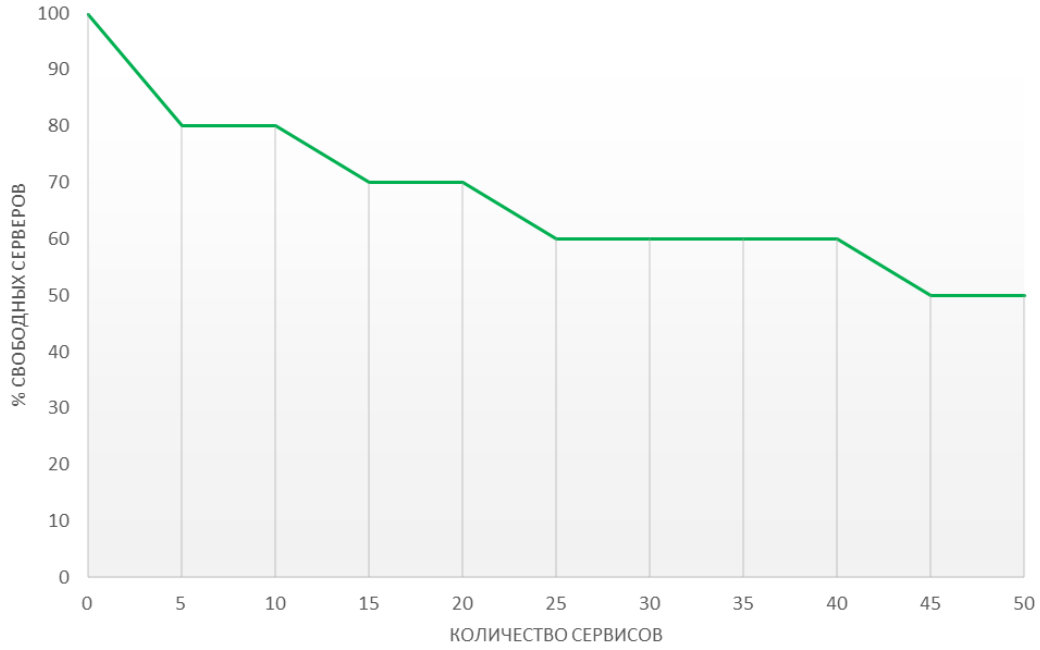


Рисунок 2.12 – Зависимость количества свободных серверов от количества размещаемых сервисов

2.7 Анализ работы алгоритмов и выбор наиболее подходящего алгоритма

С целью сравнения результатов алгоритмов и выбора, наиболее подходящего из них, данные алгоритмы были проверены на тестовом наборе данных (приложение А).

Для того, чтобы сравнить вышеописанные алгоритмы, необходимо определить критерии их сравнения. Путем опроса группы экспертов, определенной в пункте 2.5, были определены критерии сравнения для выбора алгоритма. Первый критерий – время работы алгоритма. Данный критерий имеет среднюю важность и ограничение работы до 10 минут. Следующий критерий – заполненность серверов. Заполненность серверов может быть рассчитана как убывающий ряд, каждое значение которого является процентным соотношением кол-ва заполненных серверов к общему кол-ву серверов при увеличении кол-ва

сервисов. Критерий имеет высокую важность, чем выше значение заполненности сервера, тем более полно используется его свободное место. Если сервер заполнен полностью, значение заполненности равно единице. Последний критерий - вероятность ошибочного решения. Вероятность ошибочного решения будет рассчитываться по 50 независимым запускам. Решение считается ошибочным, если в результате работы алгоритма полученное решение не отвечает ограничениям, установленным в пункте 2.1.

Представим критерии сравнения алгоритмов, определенные в ходе опроса экспертов, в таблице 2.11. Экспертные оценки получены с использованием того же метода, как и в случае с оценкой критериев из таблицы 2.4. Критерии сравнения алгоритмов, анкета опроса экспертов и его результаты представлены в приложении Б (таблицы 5, 6 и 7). Коэффициент конкордации по данным таблицы 7 приложения Б составляет 0,035, что говорит о согласованности мнений экспертов.

Критерии выбора и их экспертная оценка представлены в таблице 2.11.

Таблица 2.11 – Критерии сравнения алгоритмов комбинаторной оптимизации

Название критерия	**Важность**	**Единицы измерения**	**Метод расчета**
Время работы алгоритма	Средняя, время работы до 10 минут	Секунды	Измерение времени работы с момента запуска до получения решения

Продолжение таблицы 2.11

Заполняемость	Высокая	-	Отношение количества свободных серверов к общему кол-ву серверов при увеличении количества сервисов
Вероятность ошибочного решения	Высокая	-	Отношение кол-ва ошибочных решений к общему числу запусков

В качестве исходных данных было использовано 2 группы: 5 виртуальных машин (серверов) и 50 сервисов и 10 виртуальных машин (серверов) и 50 сервисов, которые необходимо распределить по данным серверам.

В таблице 2.12 представлены результаты сравнения времени работы алгоритмов.

Таблица 2.12 – Время работы для различных алгоритмов.

Название алгоритма	**Время работы алгоритма, с**
Алгоритм BFD	0,0022
Генетический алгоритм	1,2631
Алгоритм имитации отжига	0,2365

Время работы алгоритмов не выходит за рамки, обозначенные экспертным решением, максимальное время, затраченное на работу алгоритма, составляет порядка 1,2 секунд.

Тенденция к сохранению количества свободных серверов при увеличении количества размещаемых сервисов (таблица 2.13, рис. 2.13) показывает, что

генетический алгоритм более полно задействует сервера, чем алгоритм имитации отжига при последовательном увеличении количества сервисов от 0 до 50.

Таблица 2.13 – Зависимость количества свободных серверов от количества размещаемых сервисов

	% свободных серверов										
Кол-во сервисов	0	5	10	15	20	25	30	35	40	45	50
Алгоритм BFD	100	60	40	20	0	0	0	0	0	0	0
Генетический алгоритм	100	60	60	60	60	60	40	40	0	0	0
Алгоритм имитации отжига	100	60	60	60	60	40	40	40	0	0	0

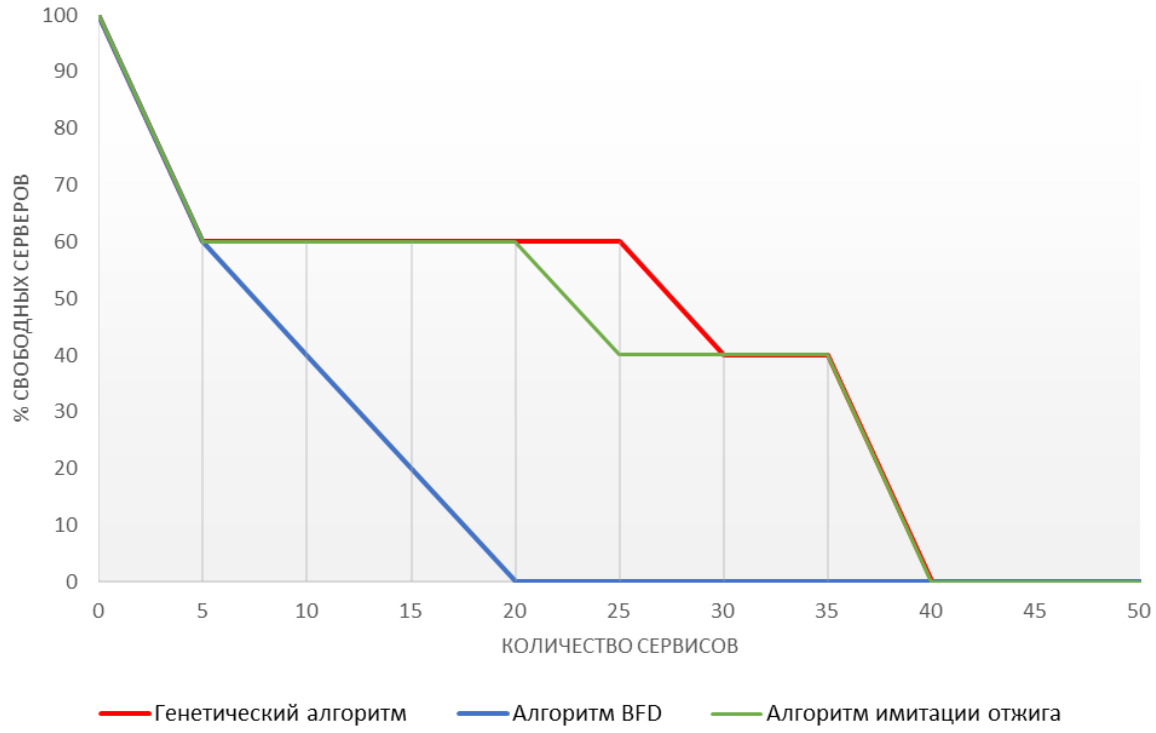


Рисунок 2.13 – Зависимость количества свободных серверов от количества размещаемых сервисов

Зависимость количества свободных серверов от количества размещаемых сервисов (таблица 2.14, рис. 2.14) на примере второй тестовой выборки (10

серверов и 50 сервисов) показывает схожую тенденцию к сокращению количества занимаемых серверов.

Таблица 2.14 – Зависимость количества свободных серверов от количества размещаемых сервисов

	% свободных серверов										
Кол-во сервисов	0	5	10	15	20	25	30	35	40	45	50
Алгоритм BFD	100	70	60	50	50	40	40	20	20	0	0
Генетический алгоритм	100	80	80	70	70	70	60	60	60	50	50
Алгоритм имитации отжига	100	80	80	70	70	60	60	60	60	50	50

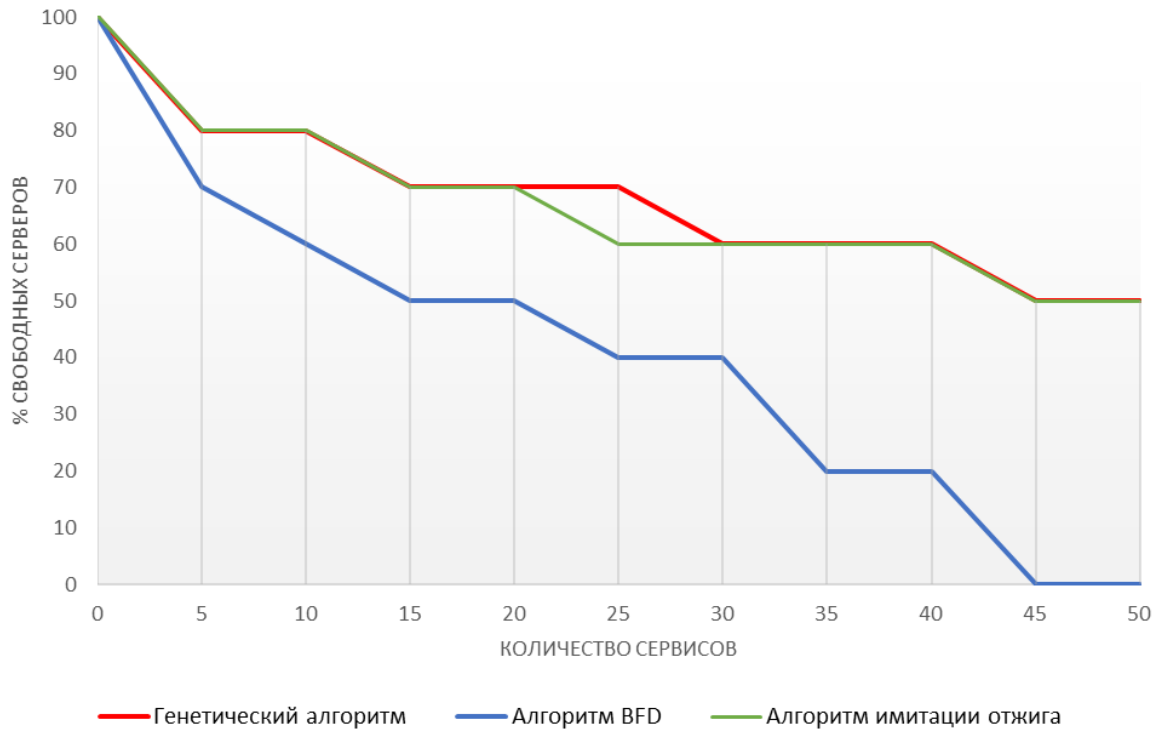


Рисунок 2.14 – Зависимость количества свободных серверов от количества размещаемых сервисов

Ниже в таблицах 2.15-2.17 представлены данные отдельно по заполненности ПЗУ, ОЗУ, ЦП. На графиках (рис. 2.15 - 2.20) показано, что использование свободного пространства серверов тем выше, чем более круто падает кривая.

Таблица 2.15 – Заполненность Hdd для серверов

Имя сервера	Заполненность ПЗУ, %		
	Алгоритм BFD	Генетический алгоритм	Алгоритм имитации отжига
WIN_1	94	100	99
WIN_2	87	100	95
WIN_3	84	60	66
LIN_4	61	95	78
LIN_5	61	28	45

На графике видно, что заполняемость ПЗУ серверов при применении генетического алгоритма выше, чем при применении других.

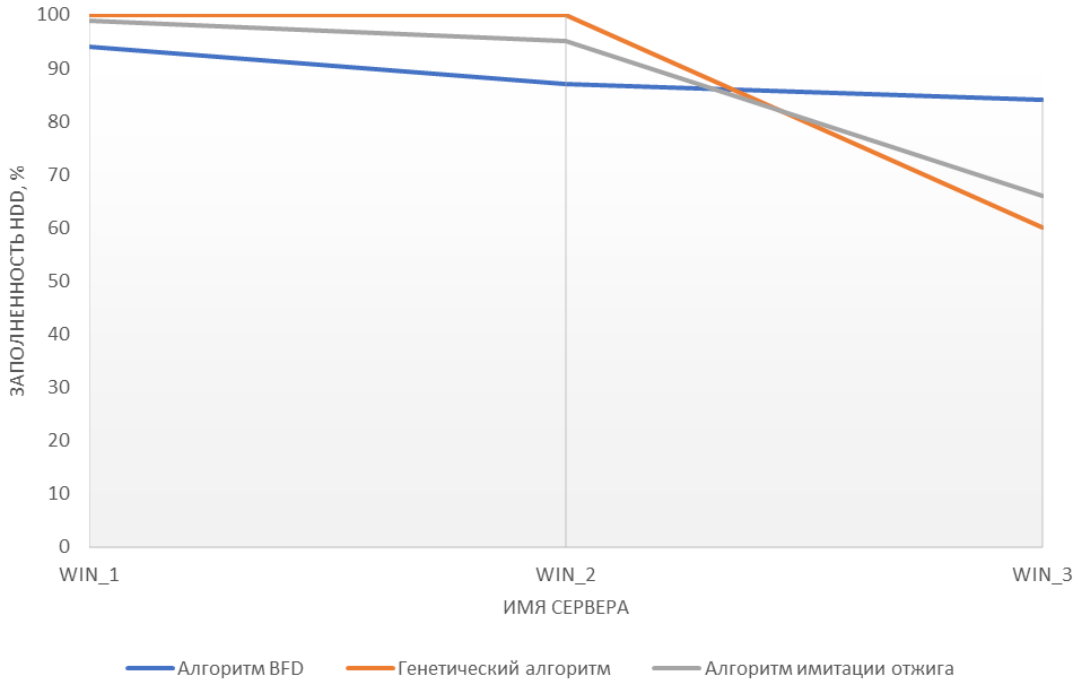


Рисунок 2.15 – Заполненность ПЗУ Windows серверов

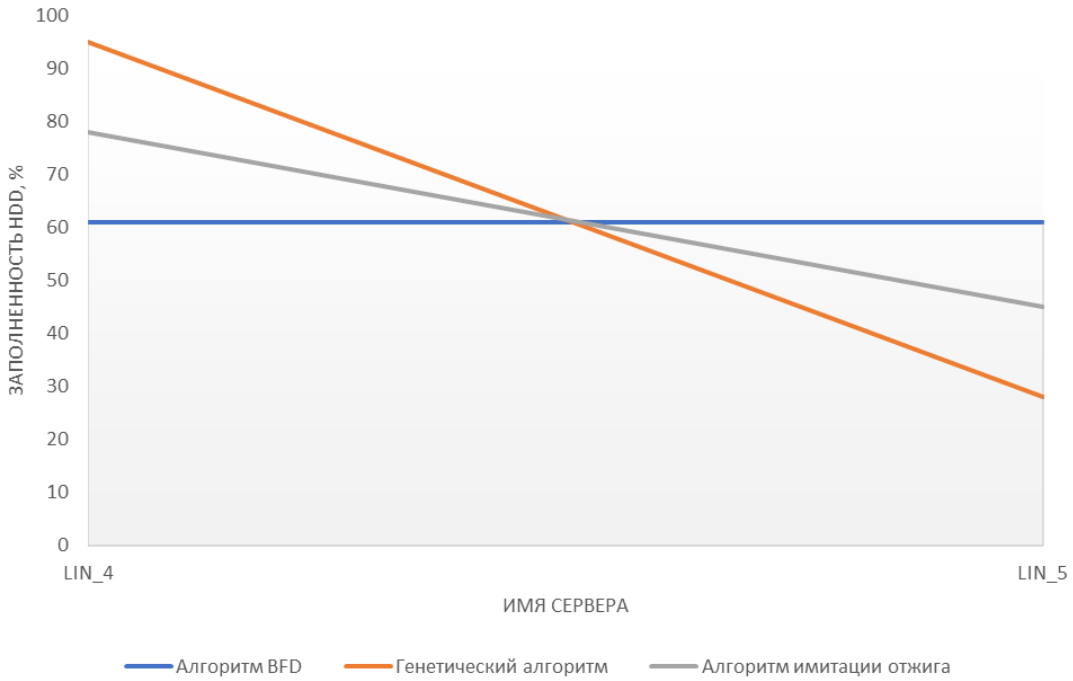


Рисунок 2.16 – Заполненность ПЗУ Linux серверов

В таблице 2.16 представлены данные по заполненности ОЗУ для серверов.

Таблица 2.16 – Заполненность ОЗУ для серверов

Имя сервера	**Заполненность ОЗУ, %**		
	Алгоритм BFD	**Генетический алгоритм**	**Алгоритм имитации отжига**
WIN_1	85	99	93
WIN_2	51	58	58
WIN_3	79	48	66
LIN_4	46	81	35
LIN_5	46	10	56

График 2.17 также показывает, что заполняемость ОЗУ серверов при применении генетического алгоритма выше, чем при применении других.

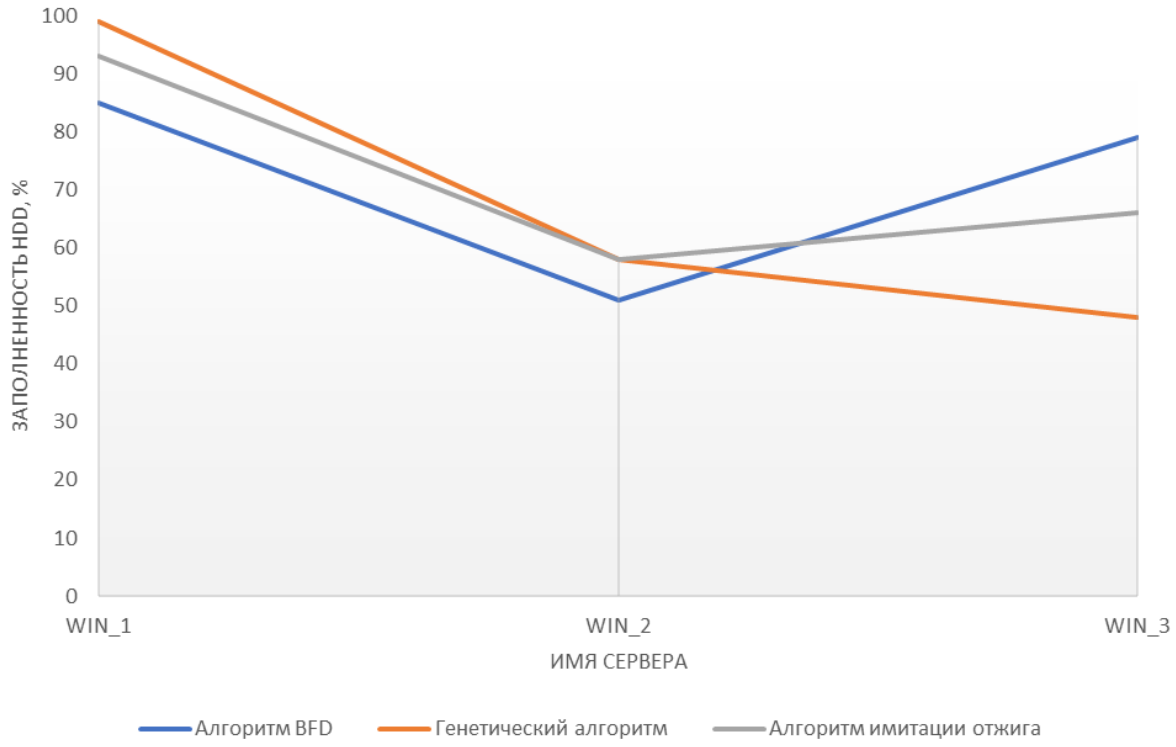


Рисунок 2.17 – Заполненность ОЗУ Windows серверов

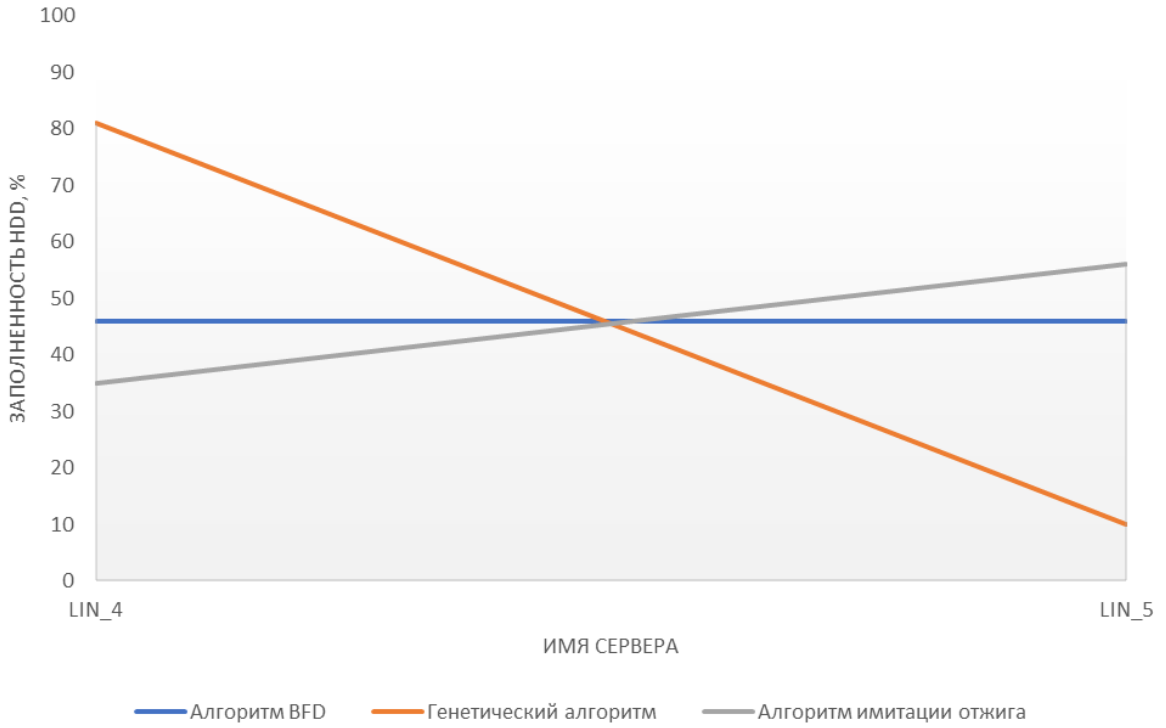


Рисунок 2.18 – Заполненность ОЗУ Linux серверов

В таблице 2.13 представлены результаты использования ЦП серверов, полученные при запуске алгоритмов.

Таблица 2.17 – Использование ЦП для серверов

Имя сервера	Использование ЦП, %		
	Алгоритм BFD	Генетический алгоритм	Алгоритм имитации отжига
WIN_1	100	100	98
WIN_2	50	38	49
WIN_3	22	33	24
LIN_4	14	18	21
LIN_5	15	11	13

На графике 2.19 видно, что использование ЦП на серверах при применении генетического алгоритма более полное.

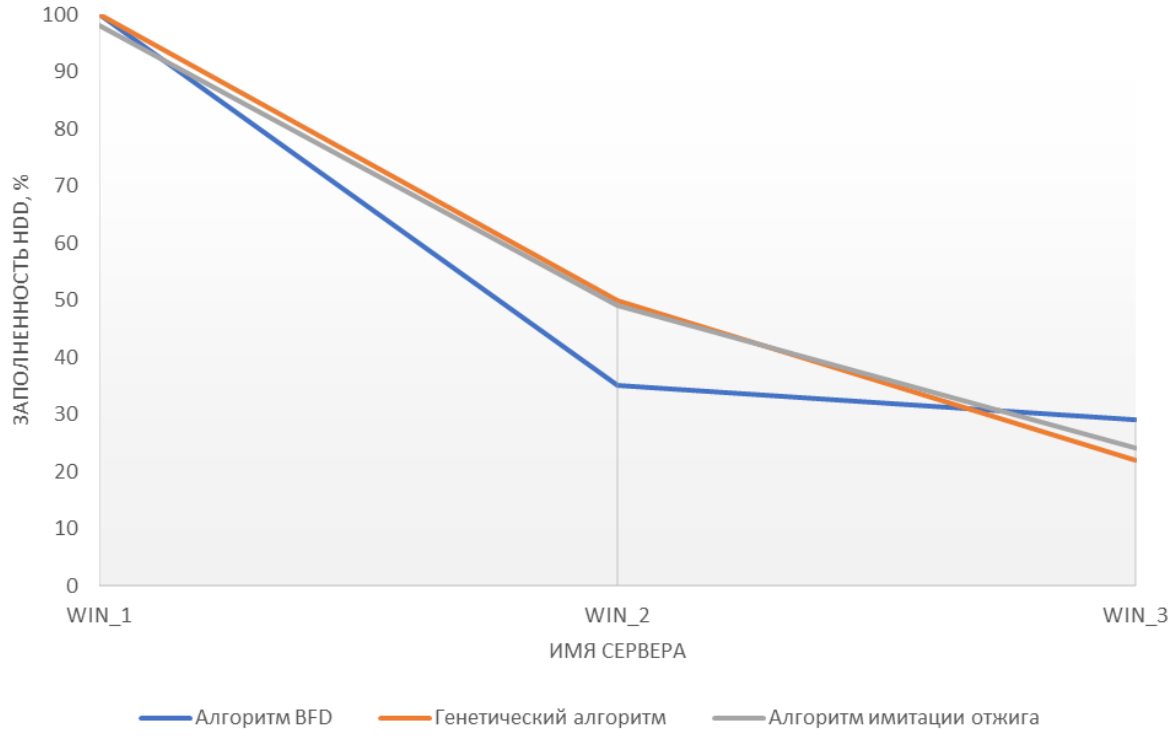


Рисунок 2.19 – Использование ЦП Windows серверов

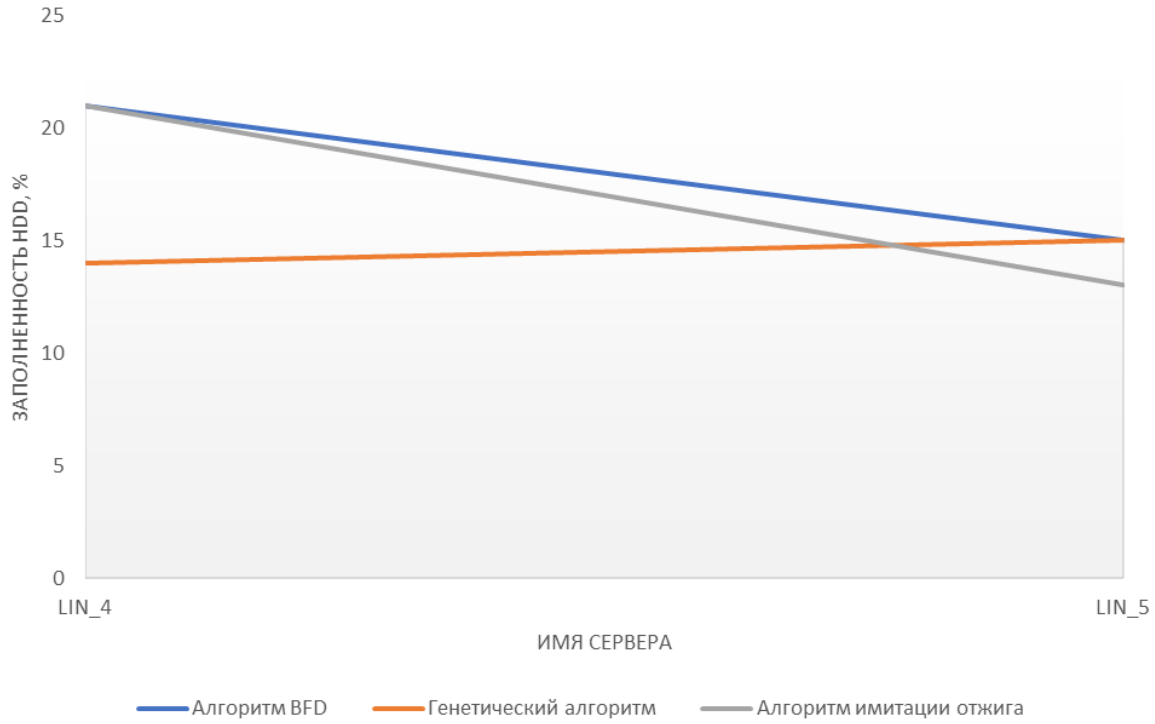


Рисунок 2.20 – Использование ЦП Linux серверов

В таблице 2.18 представлены результаты расчета вероятности ошибочного решения. Значение вероятности ошибочного решения рассчитано, как отношение количества ошибочных решений к общему числу запусков.

Таблица 2.18 – Вероятность ошибочного решения для различных алгоритмов.

Название алгоритма	**Вероятность ошибочного решения**
Алгоритм BFD	0
Генетический алгоритм	0,01
Алгоритм имитации отжига	0,07

Генетический алгоритм обладает наименьшей вероятностью ошибочного решения среди проанализированных.

Принимая во внимание показатели работы вышеописанных алгоритмов, представленные в таблице 2.19, необходимо сделать вывод о том, какой алгоритм

является наиболее подходящим для решения задачи распределения сервисов по виртуальным машинам (серверам).

Таблица 2.19 – Сравнение показателей работы алгоритмов.

Название алгоритма	**Время работы алгоритма, с**	**Вероятность ошибочного решения**	**Дисперсия значений заполненности ПЗУ**	**Дисперсия значений заполненности ОЗУ**	**Дисперсия значений заполненности ЦП**
Алгоритм BFD	0,0022	0	189,84	289,84	1404,36
Генетический алгоритм	1,2631	0,01	814,24	921,36	1425,96
Алгоритм имитации отжига	0,2365	0,07	390,64	351,44	1381,56

По показателям затраченного времени работа всех алгоритмов укладывается в установленные рамки. Вероятность ошибочного решения у генетического алгоритма имеет минимальное значение, а дисперсия значений заполненности, наоборот, имеет максимальные значения, что говорит о том, что часть серверов заполнена максимально, а оставшаяся минимально возможно. Таким образом, сделан вывод, что наиболее подходящим для решения задачи распределения сервисов по виртуальным машинам (серверам) является генетический алгоритм.

2.8 Выводы по главе 2

В данной главе была разработана математическая модель, описывающая задачу распределения программных компонентов на виртуальные сервера. Данная задача является классической NP-полной задачей в теории комбинаторной оптимизации, соответственно, принято решение сравнить ряд существующих эвристических алгоритмов. Создан прототип инфраструктурной среды,

включающий в себя симулятор сетевой инфраструктуры и непосредственно приложение, реализующее вышеописанные алгоритмы. В результате сравнения показателей работы алгоритмов на тестовой выборке доказано, что наиболее подходящим для решения задачи распределения программных компонентов на виртуальные сервера является генетический алгоритм.

3 ГЛАВА. РАЗРАБОТКА СИСТЕМЫ АВТОМАТИЧЕСКОГО РАЗВЕРТЫВАНИЯ ПРИЛОЖЕНИЙ В ОБЛАЧНОЙ ИНФРАСТРУКТУРЕ

3.1 Определение требований к системе автоматического развертывания приложений

Перед проектированием системы развертывания приложений в облачной инфраструктуре необходимо определить основной функционал. Представим данные требования в виде таблицы 3.1.

Таблица 3.1 – Необходимый функционал системы автоматического развертывания приложений в облачной инфраструктуре.

Категория	**Подробное описание**
Основной функционал	1. Развертывание приложений. 2. Удаление приложений. 3. Обновление приложений. 4. Оптимизация серверной инфраструктуры. 5. Сбор информации о кол-ве свободной ПЗУ и ОЗУ, запущенных процессах.
Источники файлов приложений	Система непрерывной интеграции программного обеспечения Jenkins
Хранение данных	Обеспечение возможности управления версиями, отката и хранения истории операций.
Сетевой стек	Передача данных по HTTP
Поддерживаемые ОС	Windows, Linux
Пользовательский интерфейс	Возможность работы как веб-приложение или как десктоп приложение

Рассмотрим требования, перечисленные в таблице 3.1 подробнее. Приложение предназначено для автоматического развертывания приложений в облачной инфраструктуре, соответственно, в качестве основного функционала рассматривается возможность:

– развертывания новых приложений,

– обновления уже установленных приложений,

– удаления приложений.

Оптимизация серверной инфраструктуры подразумевает под собой минимизацию количества задействованных серверов при развертывании, обновлении и удалении приложений.

Для использования данного функционала, пользователю необходимо получать следующую информацию о текущем состоянии инфраструктуры:

– запущено или остановлено развернутое ранее приложение;

– общее количество постоянной и оперативной памяти на сервере;

– занятое количество постоянной и оперативной памяти на сервере;

– использование CPU сервера.

Также, пользователю необходимо хранить информацию о развернутых приложениях. Сборка релизов осуществляется в системе непрерывной интеграции Jenkins, релиз хранится в виде архива. Каждое приложение можно рассматривать, как программный компонент, имеющий имя и версию. Хранить имя и версию релиза возможно в текстовом формате в базе данных. При таком формате хранения данных возможно отслеживать используемые в настоящий момент версии приложений, а также, историю обновления, установки или удаления приложений [14].

Работа приложения в локальной сети предприятия подразумевает под собой передачу данных по HTTP.

На виртуальных серверах центра обработки данных применяются операционные системы Windows и Linux, поэтому, также, необходимо обеспечить работы системы развертывания приложений в таких операционных системах.

Преобразуем вышеописанные требования в более формализованный вид (таблица 3.2):

Таблица 3.2 – Описание функционала системы и перечень возможных действий пользователя

Описание функционала	**Перечень действий пользователя**
Пользователю нужно обеспечить возможность выбора серверов (хостов), на которые будет установлено необходимое программное обеспечение, данный перечень пользователю необходимо создавать.	Создание, редактирование, удаление хостов.
Пользователь должен знать, какие программные компоненты он имеет возможность установить на сервера, поэтому в системе должны храниться имена этих компонентов.	Создание, редактирование, удаление компонентов.
Релиз создается на основе компонента и имеет название и версию. Группировка релизов предназначена для установки коллекции приложений на сервера «в один клик».	Создание, редактирование, удаление релизов и их групп.

Продолжение таблицы 3.2

Пользователь должен иметь возможность хранить текущую конфигурацию серверов и устанавливаемых на них релизов, то есть создать среду развертывания, содержащую цель развертывания и компонент развертывания. Цель развертывания – целевой сервер, на котором планируется произвести установку приложения, а компонент развертывания – необходимый релиз. К примеру, целями развертывания будут сервера системы слежения, компонентами развертывания – устанавливаемые приложения, а вместе это будет называться средой развертывания	Создание, редактирование, удаление сред развертывания.
Пользователь по желанию должен иметь возможность воспользоваться возможностями программы, позволяющими определить размещение программных компонентов на серверах с целью минимизации числа задействованных при развертывании серверов.	Оптимизация среды развертывания.

Продолжение таблицы 3.2

Развертывание очередного релиза должно происходить по следующему сценарию: пользователь выбирает необходимую среду развертывания, путем сравнения версий, установленных компонентов и компонентов, находящихся в последнем релизе в системе автоматически определяется, какие компоненты необходимо установить на сервер, а какие обновить или удалить.	Разворачивание релизов на среде развертывания.
При развертывании приложения необходима настройка его конфигурации. Данные по настройке конфигурации пользователь должен иметь возможность создавать и хранить в виде пар ключ-значение.	Создание, редактирование, удаление параметров конфигурации.

Исходя из вышеописанных требований данное приложение подразумевает внедрение нескольких программных модулей, а именно:

– компонента, являющего главным и отвечающего за логику выполнения команд пользователя;

– компонентов, запущенных на каждом виртуальном сервере, и выполняющих команды и запросы пользователя;

– клиентского приложения, отвечающего за взаимодействие с пользователем;

– компонента, выполняющего роль шлюза между клиентским и серверным компонентами.

В первой главе данной работы было показано, что при проектировании современных веб-приложений отдается предпочтение микросервисной архитектуре, поэтому выбор технологий для разработки системы развертывания приложений будем учитывать и данный критерий.

3.2 Выбор технологии проектирования

В пункте 3.1 были перечислены необходимые программные компоненты для разработки системы развертывания приложений, поэтому выбор технологий для их проектирования произведем в том же порядке.

Технологией разработки веб-приложений, отвечающей требованиям, перечисленным в пункте 3.1, является использующаяся на предприятии платформа .Net Core [23]. Приложения .NET написаны на языке программирования C# и запускают управляемый код в среде выполнения, известной как среда CLR.

На каждом виртуальном сервере предполагается использовать программных агентов, выполняющих команды и запросы пользователя. Эти агенты также будут являться веб-приложениями, реализованными на платформе .Net core.

Так как разрабатываемое приложение будет иметь микросервисную архитектуру, его компоненты будут коммуницировать между собой с использованием протокола HTTP через вызовы API микросервисов и по протоколу AMQP [32]. Также, вызовы API микросервисов будут осуществляться пользователем с помощью клиентского приложения через шлюз.

Выбранная реализация серверного приложения с использованием Rest Api [61] расширяет возможности для реализации клиентского приложения. В этом

случае является возможным создать как сетевое desktop-приложение, реализованное с использованием паттерна MVVM, так и веб-клиента, для работы с приложением из веб-браузера.

В качестве системы маршрутизации входящих запросов между клиентским приложением и микросервисами будет выступать шлюз API. В данном случае, ограничимся одним шлюзом API, который будет представлять единую точку входа в приложение. В качестве такого подходит простой легковесный шлюз Ocelot [59], предназначенный для работы с веб-приложениями на платформе .net core.

В качестве системы управления базами данных были рассмотрены 3 наиболее используемых варианта (таблица 3.3):

Таблица 3.3 – Наиболее распространенные СУБД.

Название СУБД	**Лицензия**	**Запуск образа в Docker**	**Совместимость с Entity Framework Core**
Oracle Database	Коммерческое ПО	Да	Да
PostgreSQL	Открытое ПО	Да	Да
Microsoft SQL Server	Коммерческое ПО	Да	Да

Для работы с базами даннных в приложениях .net core широко используется объектно-реляционный модуль сопоставления Entity Framework Core [57], поддерживающий множество современных систем управления базами данных.

На данный момент все перечисленные СУБД имеют возможность запуска в контейнере Docker, однако PostgreSQL является свободно распространяемым ПО, поэтому остановим выбор именно на СУБД PostgreSQL.

При проектировании программного обеспечения будем следовать концепции проблемно-ориентированного подхода (DDD) [31]. Проблемно-

ориентированный подход - концепция, согласно которой структура и язык программного кода (имена классов, методы классов, переменные класса) должны соответствовать бизнес-области.

Проблемно-ориентированный подход преследует следующие цели:

– сосредоточение внимания проекта на основном домене и логике предметной области;

– построение комплексных проектов на модели предметной области;

– инициирование творческого сотрудничества между техническими специалистами и экспертами в предметной области для итеративного уточнения концептуальной модели, направленной на решение конкретных проблем предметной области.

Концепция модели включает в себя:

– контекст, т.е. обстановку, в которой появляется слово или высказывание, определяющая его значение;

– домен - сферу знаний (онтология), влияния или деятельности. Предметную область, к которой пользователь применяет программу, является областью программного обеспечения;

– модель - систему абстракций, которая описывает выбранные аспекты домена и может использоваться для решения проблем, связанных с этим доменом;

– единый язык - язык, структурированный вокруг модели предметной области и используемый всеми членами группы для связи всех действий группы с программным обеспечением.

Для поддержания целостности модели необходимо придерживаться следующих принципов:

1. Принцип ограниченного контекста

В любом большом проекте задействовано несколько моделей. Тем не менее, когда код, основанный на разных моделях, объединяется, программное

обеспечение становится неполноценным, ненадежным и трудным для понимания. Необходимо определить контекст, в котором применяется модель и соблюдать строгую согласованность модели в этих пределах.

2. Принцип непрерывной интеграции

Когда несколько людей работают в одном и том же ограниченном контексте, модель имеет сильную тенденцию к фрагментации. Чем больше команда, тем серьезнее проблема, но всего три-четыре человека могут столкнуться с серьезными проблемами. Однако при разбиении системы на все более мелкие контексты в конечном итоге теряется ценный уровень интеграции и согласованности. Необходимо организовать процесс слияния всего кода как можно чаще с автоматическими тестами для быстрого выявления фрагментации и использовать единый язык, чтобы выработать общий взгляд на модель по мере того, как концепции развиваются в головах разных людей.

3. Контекстная карта

Индивидуальный ограниченный контекст оставляет некоторые проблемы при отсутствии глобального обзора. Контекст других моделей все еще может быть расплывчатым и изменчивым. Люди в других командах не очень хорошо осведомлены о границах контекста и неосознанно вносят изменения, которые стирают границы или усложняют взаимосвязи. Когда необходимо установить связи между разными контекстами, они имеют тенденцию перетекать друг в друга. В этом случае нужно определить каждую модель, задействованную в проекте, и определить ее ограниченный контекст, назвать каждый ограниченный контекст и сделать имена частью повсеместного языка. Описать точки соприкосновения между моделями, выделив явный перевод для любого общения и выделив любое совместное использование.

Разрабатывая микросервисы, необходимо учитывать, каким образом будет построено их взаимодейтсвие. Здесь применим метод контрактного программирования. Это метод проектирования программного обеспечения, предполагающий, что проектировщик должен определить формальные, точные и верифицируемые спецификации интерфейсов для компонентов системы. Данные спецификации называются «контрактами» в соответствии с концептуальной метафорой условий и ответственности в гражданско-правовых договорах. Подключая такие контракты как динамические библиотеки можно быть уверенным, что при работе с данными используются одни и те же типа вне зависимости от микросервиса.

При разработке системы развертывания приложений будет применен шаблон проектирования микросервисов CQRS - шаблон разделения ответственности на команды и запросы [52]. Это архитектурный шаблон для разделения операции чтения и обновления для хранилища данных. CQRS выгоден для контрактного программирования, так как любой возвращающий значение метод (любой запрос) можно вызывать в утверждениях, не беспокоясь о возможном изменении состояния программы.

3.3 Проектирование структуры программного обеспечения

Представим в данном пункте следование принципам проблемно-ориентированного проектирования на практике:

1. Необходимо сосредоточить внимание проекта на основном домене и логике предметной области.

2. Разработать домен каждой предметной области.

3. Установить связи между разными контекстами и предотвратить перетекания контекстов друг в друга.

4. В процессе разработки использовать непрерывное тестирование интеграцию кода.

Система автоматического развертывания приложений в облачной инфраструктуре работает следующим образом. На каждом виртуальном сервере уровня L2 и L3 расположены агенты, а на выделенном веб-сервере работает серверный компонент приложения. Агенты – программы-демоны, запущены на каждом сервере. Агенты выполняют команды и запросы, приходящие по локальной вычислительной сети от сервера. Такими запросами могут быть запросы типа операционной системы, кол-ва свободной оперативной и постоянной памяти, запущенных процессов, а командами – команды развертывания, удаления или обновления приложения на сервере.

На выделенном сервере расположена серверная часть системы, отвечающая за обработку управляющих воздействий пользователя, а также шлюз, выполняющие маршрутизацию запросов клиента. Для хранения данных, необходимых в процессе работы приложения используется сервер баз данных.

При разработке микросервиса размер не должен быть важным фактором. Главным должно быть создание слабо связанных служб, что позволяет добавиться автономности при разработке, развертывании и масштабировании каждой сервиса. Конечно же, при определении и проектировании микросервисов следует стремиться к тому, чтобы они были как можно меньше, если только они не имеют слишком много прямых зависимостей от других микросервисов. Внутренняя связанность микросервиса и его независимость от других сервисов важнее его размера.

Согласно принципу предметно-ориентированного проектирования, структура и язык программного кода должны соответствовать бизнес-области. Как было сказано ранее, создавать микрослужбы необходимо на основе ограниченного контекста (в рамках части предметной области). В некоторых случаях ограниченный контекст может состоять из нескольких физических служб, но не наоборот [38].

В пункте 3.1 в таблице 3.2 приведен перечень необходимых пользователю функций для работы с системой развертывания приложений. По данному перечню мы можем выделить ограниченные контексты, представленные в таблице 3.4. В таблице 3.4 представлен вариант разбиения системы на микросервисы согласно их функционалу.

Таблица 3.4 – Перечень микросервисов, соответствующих определенному функционалу.

Функция	**Микросервис**
Создание перечня хостов, на которые будет произведена установка приложений	Микросервис хостов
Создание перечня релизов и их группировка	Микросервис релизных групп
Создание перечня программных компонентов (приложений), устанавливаемых на сервера	Микросервис компонентов
Создание сред развертывания	Микросервис сред развертывания
Создание и хранение параметров развертывания	Микросервис параметров развертывания
Формирование заданий на обновление, удаление или установку программных компонентов	Микросервис развертывания
Размещение программных компонентов на серверах с целью максимально возможного использования ресурсов серверов.	Микросервис оптимизации

На рис. 3.1 представлена диаграмма компонентов разрабатываемого приложения. В качестве протокола связи между клиентским приложением и микросервисами служит HTTP. Кроме того, поддерживается асинхронная связь для передачи обновленных данных нескольким службам на основе

открытого протокола для передачи сообщений между компонентами системы AMQP. В качестве брокера сообщений выступает RabbitMQ.

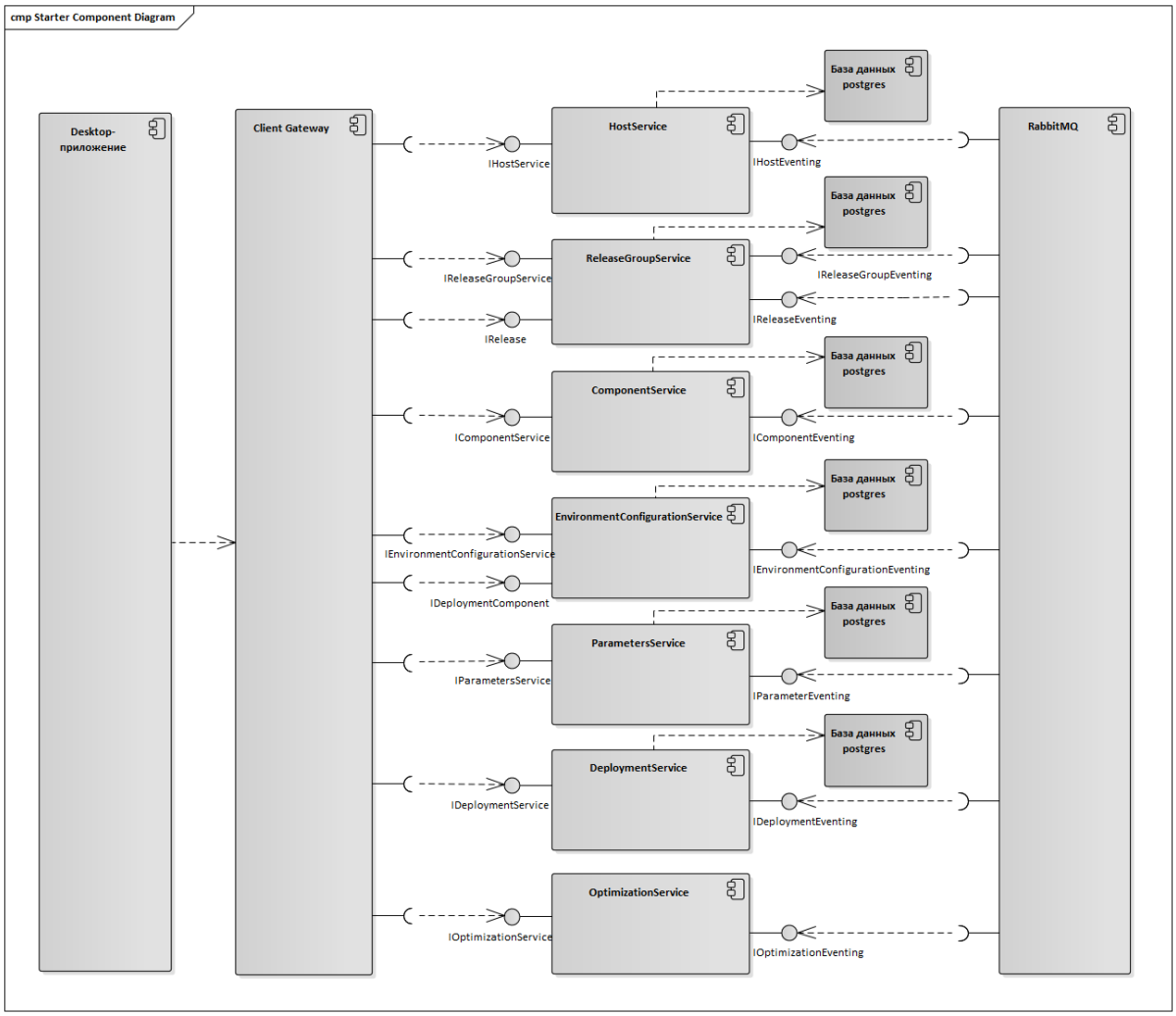


Рисунок 3.1 – Диаграмма компонентов системы развертывания приложений

Единой точкой входа для клиентских приложений является шлюз API. Шлюз API — это точка доступа, которую система предоставляет извне. Шлюз API инкапсулирует внутреннюю архитектуру системы и предоставляет индивидуальный API для каждого клиента.

Ограничительные контексты определены и приложение разделено на микросервисы. Теперь нужно разработать домен каждой предметной области.

При разработке API микросервисов применим архитектурный стиль взаимодействия компонентов REST [61]. Требование архитектуры REST – идентификация сущностей. Все ресурсы идентифицируются в запросах, например, с использованием URI в интернет-системах. Ресурсы концептуально отделены от представлений, которые возвращаются клиентам. Например, сервер может отсылать данные из базы данных в виде HTML, XML или JSON, ни один из которых не является типом хранения внутри сервера. Каждый представленный далее доменный объект является сущностью, имеющей идентификатор, т.е. каждый доменный объект - это наследник базового класса, имеющего уникальный идентификатор. Этот функционал вынесен в общую библиотеку классов, подключаемую в каждом проекте при разработке микросервисов, поэтому не представлен на диаграммах классов доменов.

Представим диаграммы классов домена с помощью унифицированного языка моделирования UML.

На рис. 3.2 представлена диаграмма классов CRUD-сервисов, а именно:

1. Сервиса хостов, домен которого содержит единственный объект Host, отражающий информацию об уникальном виртуальном сервере.

2. Сервиса релизных групп, в котором для организации связи многие-ко-многим создан класс ReleaseGroupMember.

3. Сервиса компонентов, содержащего единственный класс Component.

4. Сервиса параметров развертывания, отвечающего за работу с пользовательскими параметрами, необходимыми при развертывании приложений на серверах.

5. Сервиса сред развертывания. Как сказано в пункте 3.1, в том случае, когда пользователь знает, на какой группе серверов необходимо совершить установку необходимых релизов, а на каких удаление или обновление, он может воспользоваться заранее созданной средой развертывания, которая должна

содержать список целей развертывания, а каждая цель развертывания должна содержать список компонентов развертывания. Один компонент развертывания может содержаться как в одной среде развертывания, так и в нескольких, поэтому, предусмотрен объект TargetComponent для организации связи многие-ко-многим.

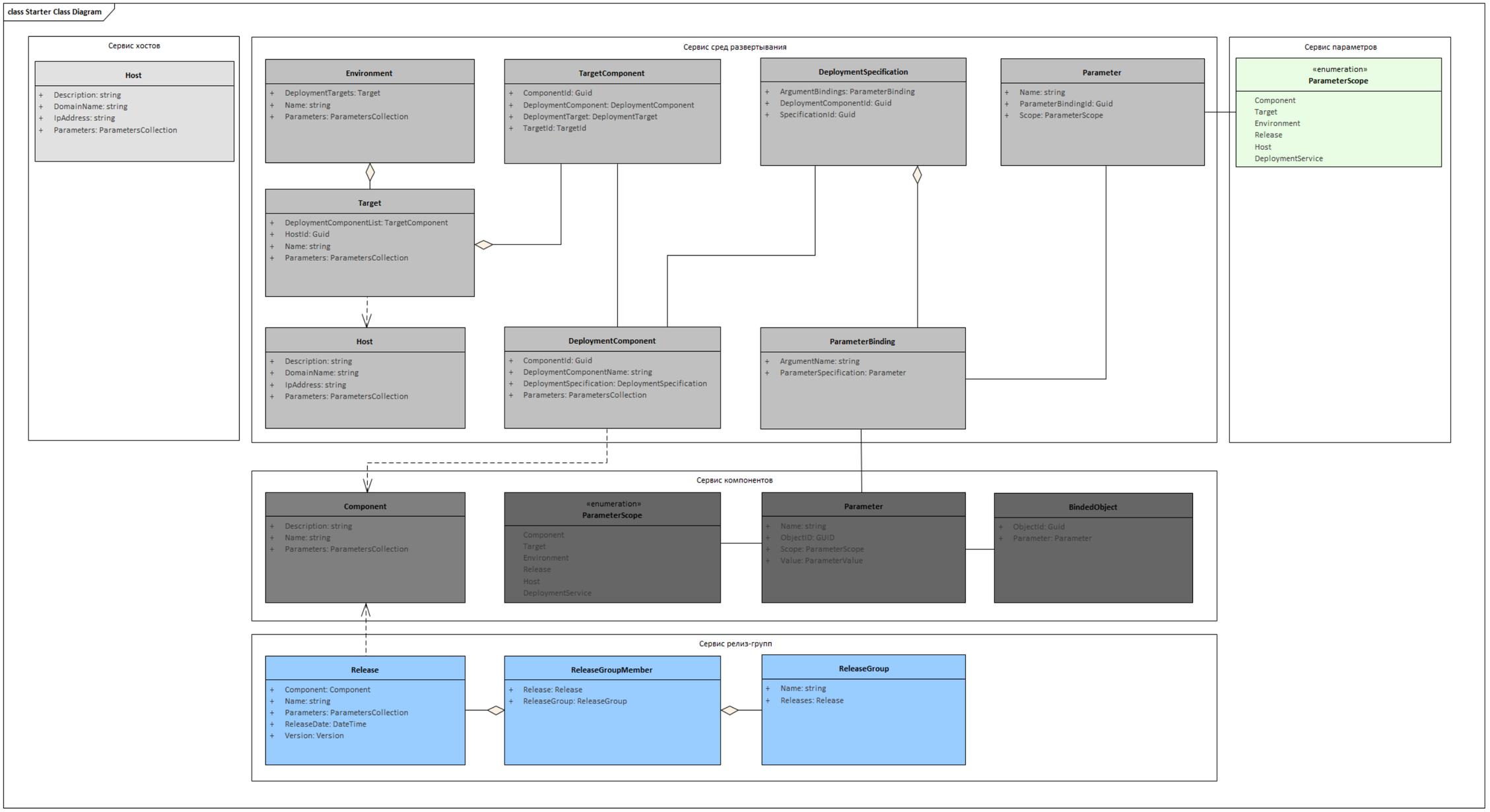


Рисунок 3.2 – Диаграмма классов CRUD сервисов

Рассмотрим связь между ограниченными контекстами сервисов сред, оптимизации и развертывания.

В сервисе сред пользователь может создать цели, на которые он рассчитывает установить компоненты, но компоненты при создании среды могут быть размещены на одну цель развертывания, либо распределены случайным образом.

Сервис оптимизации получает запрос, содержащий сформированную пользователем среду развертывания. Для работы с доменными объектами сервису оптимизации необходимо произвести маппинг dto-объектов в доменные объекты и получить недостающую информацию от сервиса агентов. По IP-адресу цели развертывания делается запрос сервису агентов для получения таких параметров, как загрузка центрального процессора в процентах, количество свободного места на жестком диске в мегабайтах, количество свободной оперативной памяти, тип операционной системы. По идентификаторам компонентов развертывания также будут получены данные о необходимом проценте процессорного времени, количестве занимаемой памяти на жестком диске в мегабайтах, количестве необходимой оперативной памяти, типе операционной системы. Полученные данные передаются на вход алгоритма оптимизации. На этом маппинг считается завершенным.

Когда все сервисы распределены на нужные сервера при помощи алгоритма оптимизации, обратное преобразование в контракты сервиса сред происходит следующим образом: создаются dto-объект среды развертывания с вложенными коллекциями целей и компонентов, расположенных в нужном месте. Диаграмма классов сервиса оптимизации представлена на рис. 3.3.

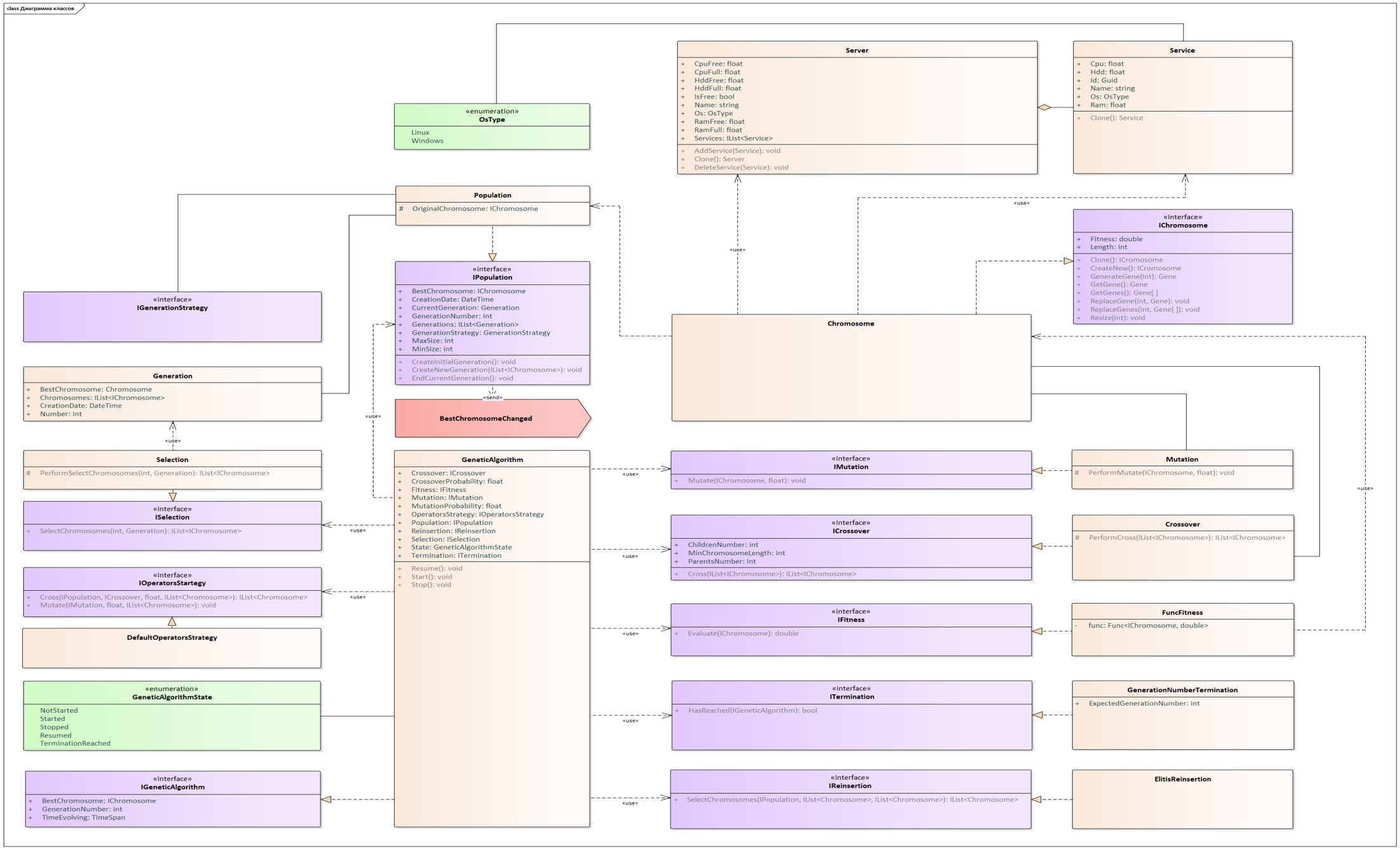


Рисунок 3.3 – Диаграмма классов сервиса оптимизации

Среда развертывания принимается на входе сервиса развертывания. Сервис развертывания несет ответственность за сравнение переданной среды с предыдущей развернутой средой. В зависимости от версии релизов программных компонентов и наличия их в среде развертывания сервис принимает решение об обновлении, удалении или развертывании приложения на сервере. Результатом работы сервиса является объект типа Task, созданный с применением паттерна «Цепочка обязанностей» [24]. При вызове метода Start у данного объекта будут созданы события в брокер сообщений, указывающие агентам, расположенным на серверах, какие операции совершать с программными компонентами. Только при завершении очередного задания будет выслано следующее. Метод Start вызывается в отдельном потоке и не блокирует основной. Ожидание выполнения задания агентами на сервисе развертывания реализовано с помощью объекта AutoResetEvent, который применяется для синхронизации потоков. Диаграмма классов сервиса развертывания представлена на рис. 3.4.

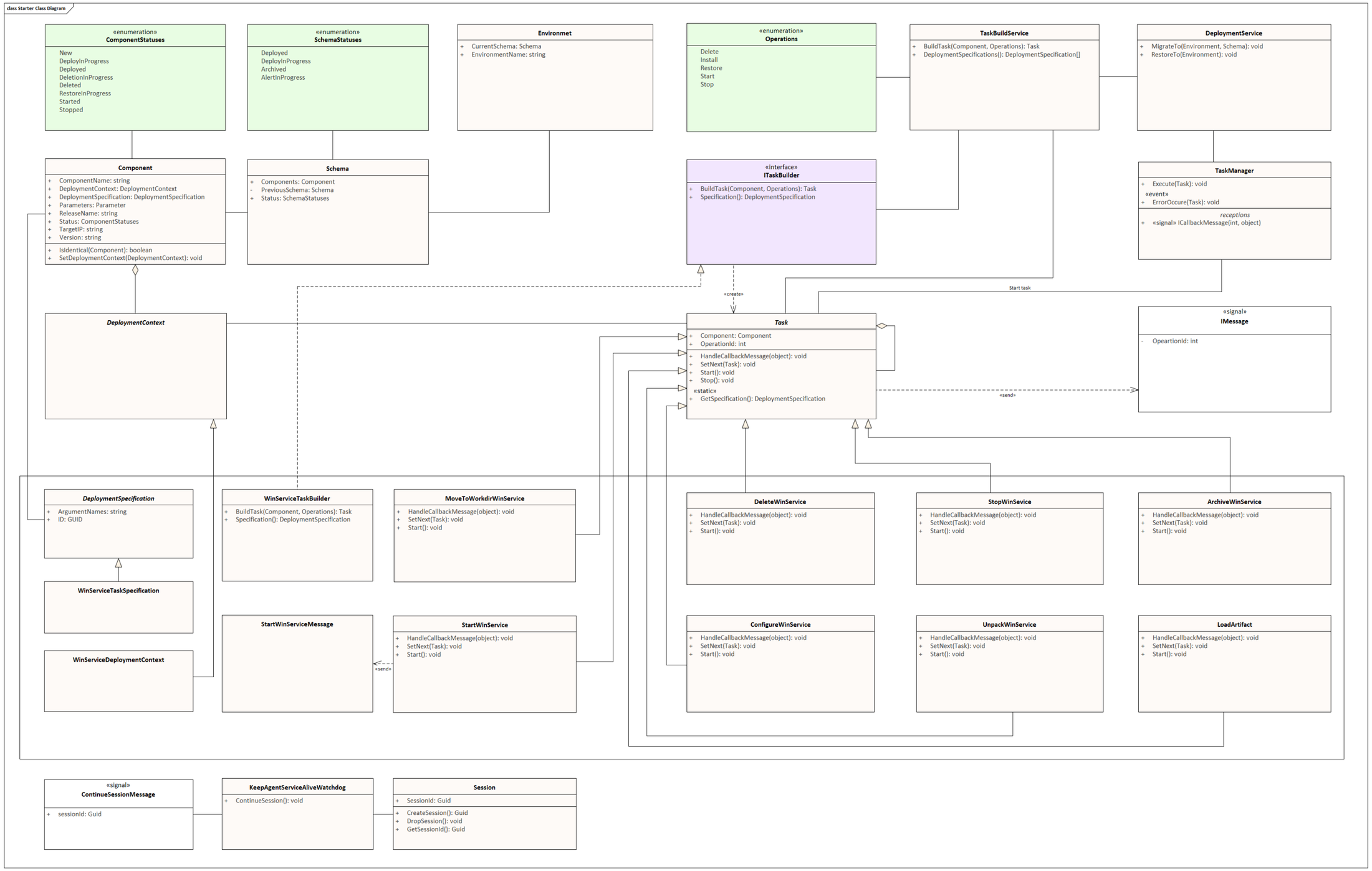


Рисунок 3.4. Диаграмма классов сервиса развертывания

Согласно рис. 3.2 - 3.4 можно судить, что границы и размеры всех ограниченных контекстов и моделей предметной области выбраны правильно, так как между этими моделями существуют несколько прочных связей, и не обязательно объединять информацию из нескольких моделей предметной области при выполнении типичных операций в приложении.

Программный интерфейс приложения (API) — описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой. API определяет функциональность, которую предоставляет программа (модуль, библиотека), при этом API позволяет абстрагироваться от того, как именно эта функциональность реализована. Для передачи информации, необходимой для обмена сервисами по HTTP, а также возврата ее конечному пользователю, используется шаблон проектирования Data Transfer Object (DTO) [30]. Объект DTO не содержит какого-либо поведения и является лишь объектом, содержащим данные.

Документация по API сгенерирована с использованием фреймворка Swagger UI [60], позволяющего создать веб-страницу с интерактивной документацией. Скриншоты веб-страниц с документацией представлены ниже. В приложении В приведена документация по API с использованием спецификации OpenAPI, содержащую, также, описание используемых DTO. Сервисы, предназначенные для выполнения вспомогательных функций, такие как сервис хостов, релизных групп, компонентов и параметров развертывания являются CRUD-приложениями, т.е. приложениями, реализующими четыре базовые функции, используемые при работе с базами данных: создание (англ. create), чтение (read), модификация (update), удаление (delete). Опишем программные интерфейсы каждого микросервиса.

API сервиса хостов (рис. 3.5) позволяет записать информацию о хосте, отредактировать, удалить хост с необходимым идентификатором, а также, получить коллекцию хостов или единственный хост по идентификатору.

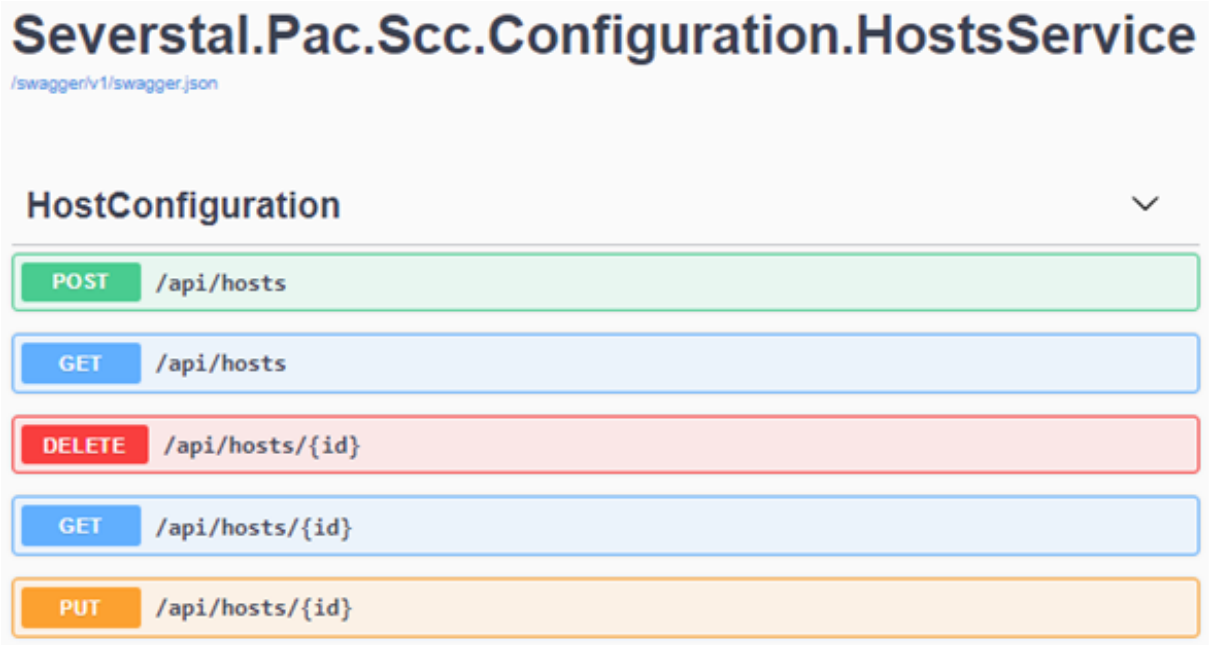


Рисунок 3.5 – API сервиса хостов

API сервиса релиз-групп (рис. 3.6) позволяет записать информацию о релизе, отредактировать, удалить релиз с необходимым идентификатором, а также, получить коллекцию релизов или единственный релиз по идентификатору.

Из коллекции релизов можно создать группу релизов, добавить/удалить необходимый релиз по идентификатору.

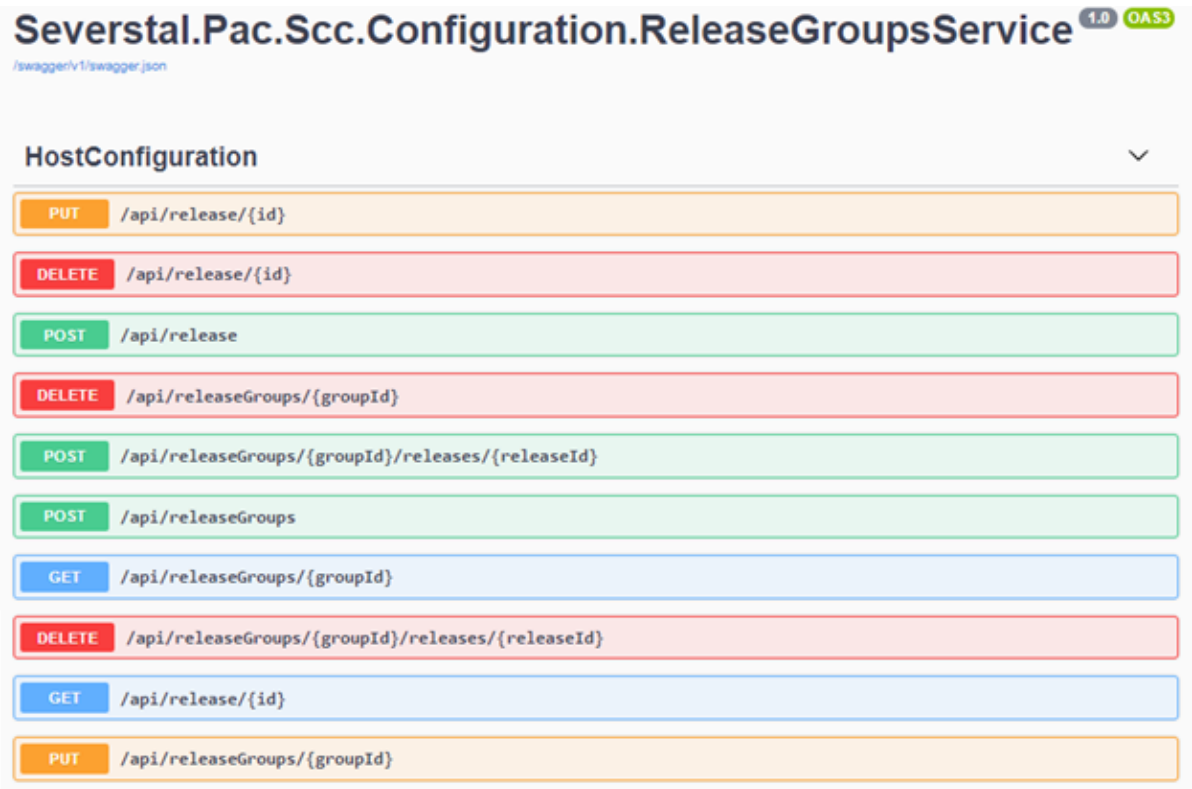


Рисунок 3.6 – API сервиса релиз-групп

API сервиса компонентов (рис. 3.7) позволяет записать информацию о компоненте, отредактировать, удалить компонент с необходимым идентификатором, а также, получить коллекцию компонентов или единственный компонент по идентификатору.

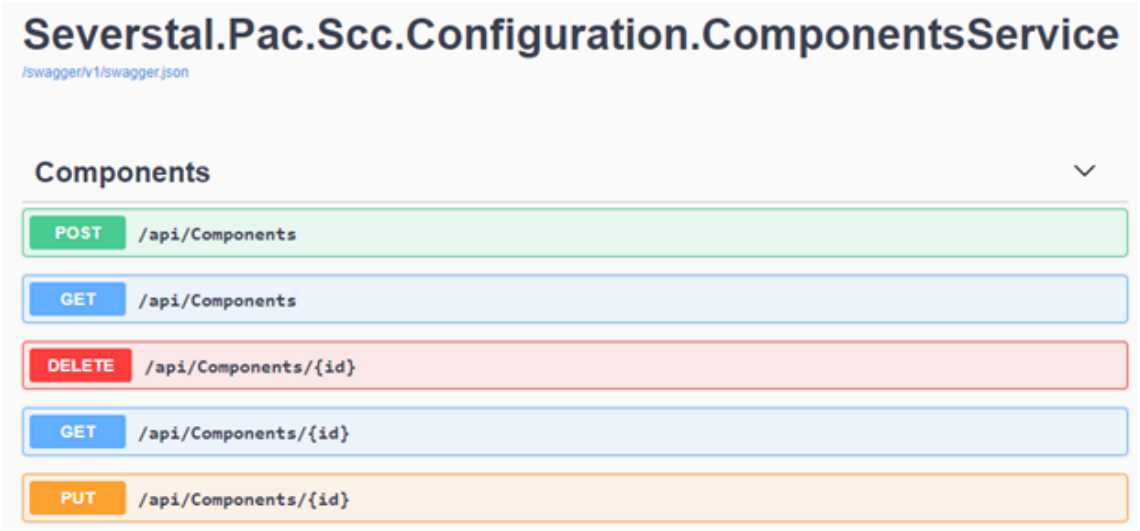


Рисунок 3.7 – API сервиса компонентов

Среда развертывания, как было описано в пункте 3.1, является объектом, содержащим коллекцию целей развертывания, т.е. виртуальных серверов, на которых предполагается установка, обновление или удаление программных компонентов, и вложенные коллекции компонентов развертывания, содержащихся на каждой цели развертывания. С точки зрения API, работа с данным функционалом реализована следующим образом.

Сервис сред развертывания имеет более расширенный API и разделен на две категории:

– компонентов развертывания;

– сред развертывания.

API компонентов развертывания (рис. 3.8) позволяет осуществлять такие же базовые операции, как и вышеописанные сервисы, т.е. запись информации о компоненте развертывания, редактирование, удаление компонента развертывания с необходимым идентификатором, а также, получение коллекции компонентов или единственного компонента развертывания по идентификатору, а также, имеет функционал для записи, получения и редактирования спецификации развертывания каждого компонента.

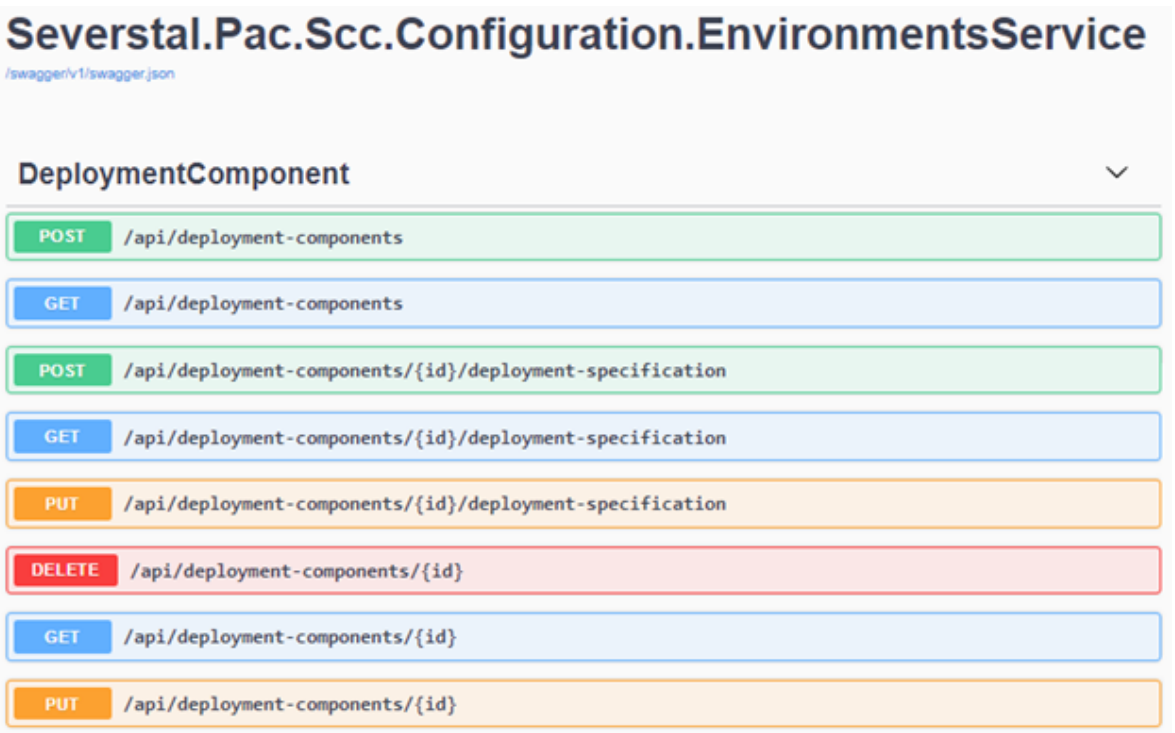


Рисунок 3.8 – API сервиса компонентов

API сред развертывания (рис. 3.9), кроме базовых CRUD операций со средой развертывания, позволяет добавлять, получать, редактировать и удалять цели развертывания и компоненты развертывания, назначенные на эти цели. Также, реализована возможность прямого запроса коллекции созданных целей развертывания.

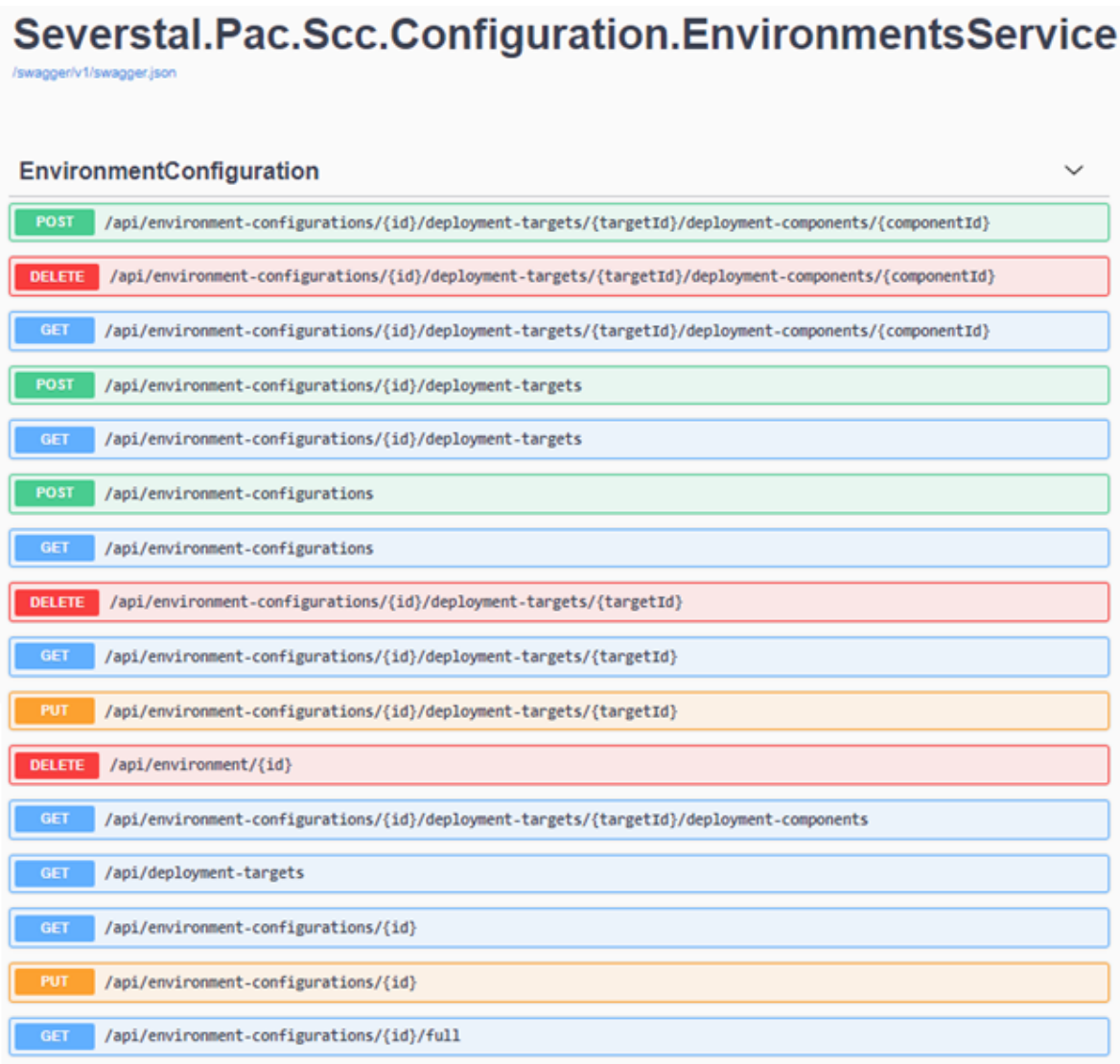


Рисунок 3.9 – API сервиса сред развертывания

API сервиса параметров (рис. 3.10) предоставляет также типовой функционал, позволяющие производить CRUD операции с параметрами развертывания.

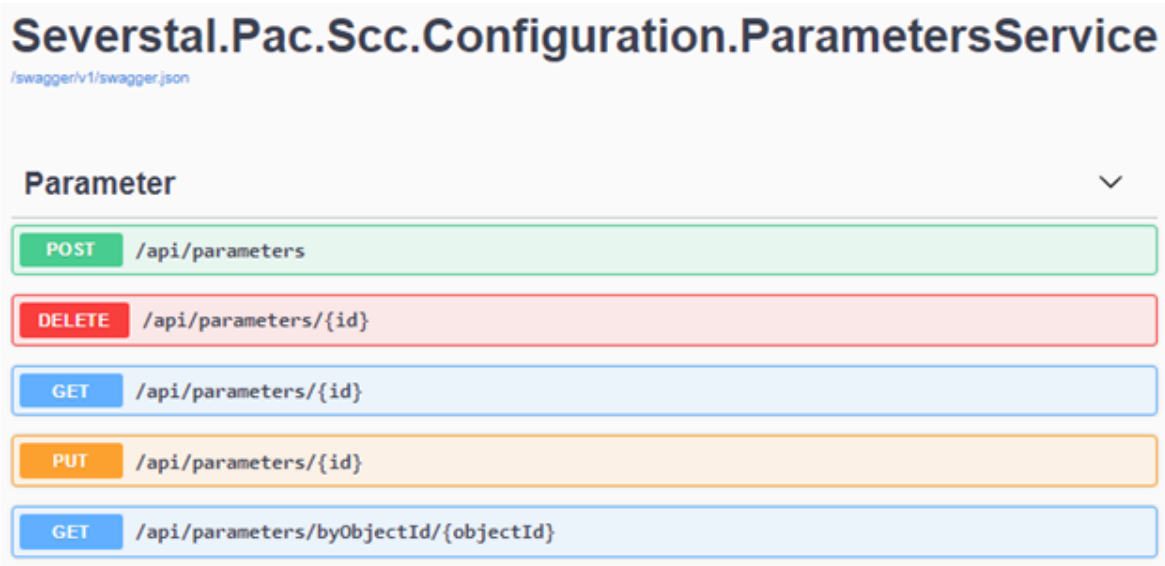


Рисунок 3.10 – API сервиса параметров

Среда развертывания в части распределения программных компонентов может быть оптимизирована при помощи вызова метода optimize API сервиса оптимизации (рис. 3.11). Метод Post возвращает оптимизированную среду развертывания.

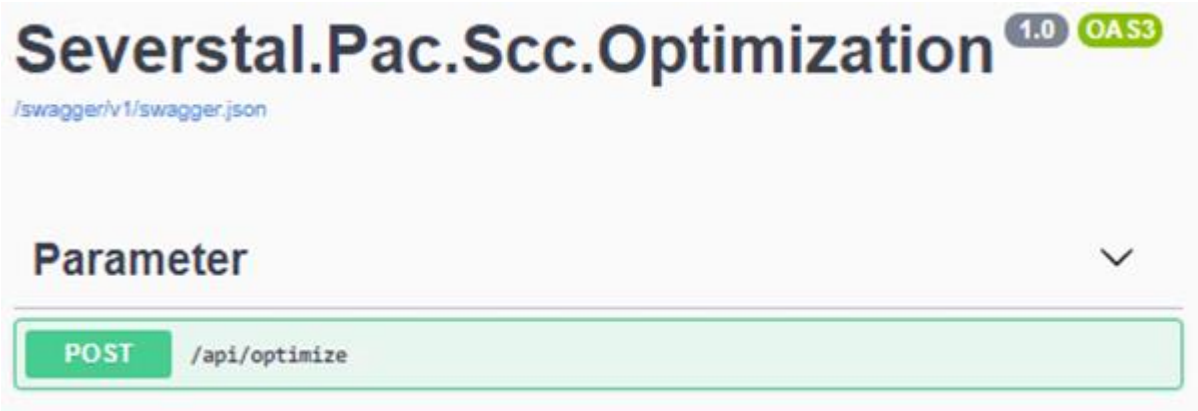


Рисунок 3.11 – API сервиса компонентов

Информация, подготовленная ранее, в частности, среда развертывания и группа релизов, используется при развертывании приложений. API сервиса развертывания представляет собой один метод, принимающий идентификатор среды развертывания и идентификатор группы релизов (рис. 3.12).

Рисунок 3.12 – API сервиса компонентов

3.4 Разработка БД

В пункте 3.3 данной главы путем сравнения наиболее популярных СУБД выбрана PostgreSQL. При разработке данной системы применен ORM Entity Framework, управляющий созданием и изменением базы данных [58]. База данных создается на основе доменной модели приложения [13].

Структура базы данных, созданной в СУБД postgres, представляет из себя, собственно базу данных с именем sccdb, и схемы, созданные в данной базе для всех микросервисов. Схема представляет собой пространство имён: она содержит именованные объекты (таблицы, типы данных, функции и операторы), имена которых могут совпадать с именами других объектов, существующих в других схемах. Для обращения к объекту нужно либо дополнить его имя именем схемы в виде префикса, либо установить путь поиска, включающий требуемую схему. Для уникальной идентификации сущностей используется тип идентификатора GUID — статистически уникальный 128-битный идентификатор. Имена схем представлены в таблице 3.5.

Таблица 3.5 – Схемы базы данных системы развертывания приложений

Микросервис	Схема базы данных
Микросервис хостов	scc_host
Микросервис релизных групп	scc_release

Продолжение таблицы 3.5

Микросервис компонентов	scc_component
Микросервис сред развертывания	scc_environment
Микросервис развертывания	scc_deployment
Микросервис параметров развертывания	scc_parameter

Модели данных представлены на рис. ниже. Сервисы релиз-групп, хостов, компонентов, параметров, представляющие из себя простые CRUD-сервисы используют упрощенные схемы для хранения данных (рис. 3.13 – 3.14). В микросервисе сред развертывания реализована возможность создания среды развертывания, добавления целей развертывания в среду развертывания, а также, добавления одинаковых компонентов развертывания на различные цели развертывания. Компонент развертывания содержит, также, информацию о сопоставлении переменных, используемых при развертывании с пользовательскими переменными.

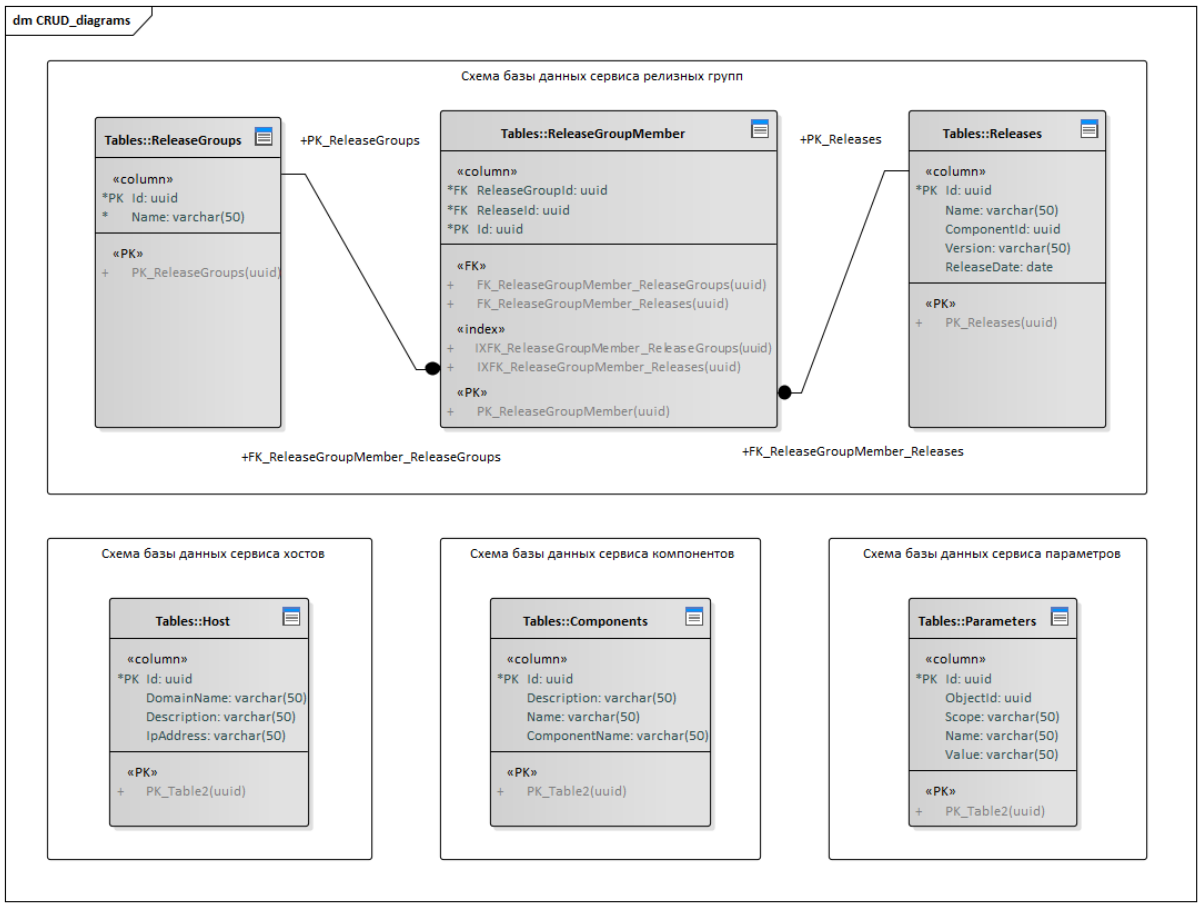


Рисунок 3.13 – Структура баз данных CRUD-сервисов

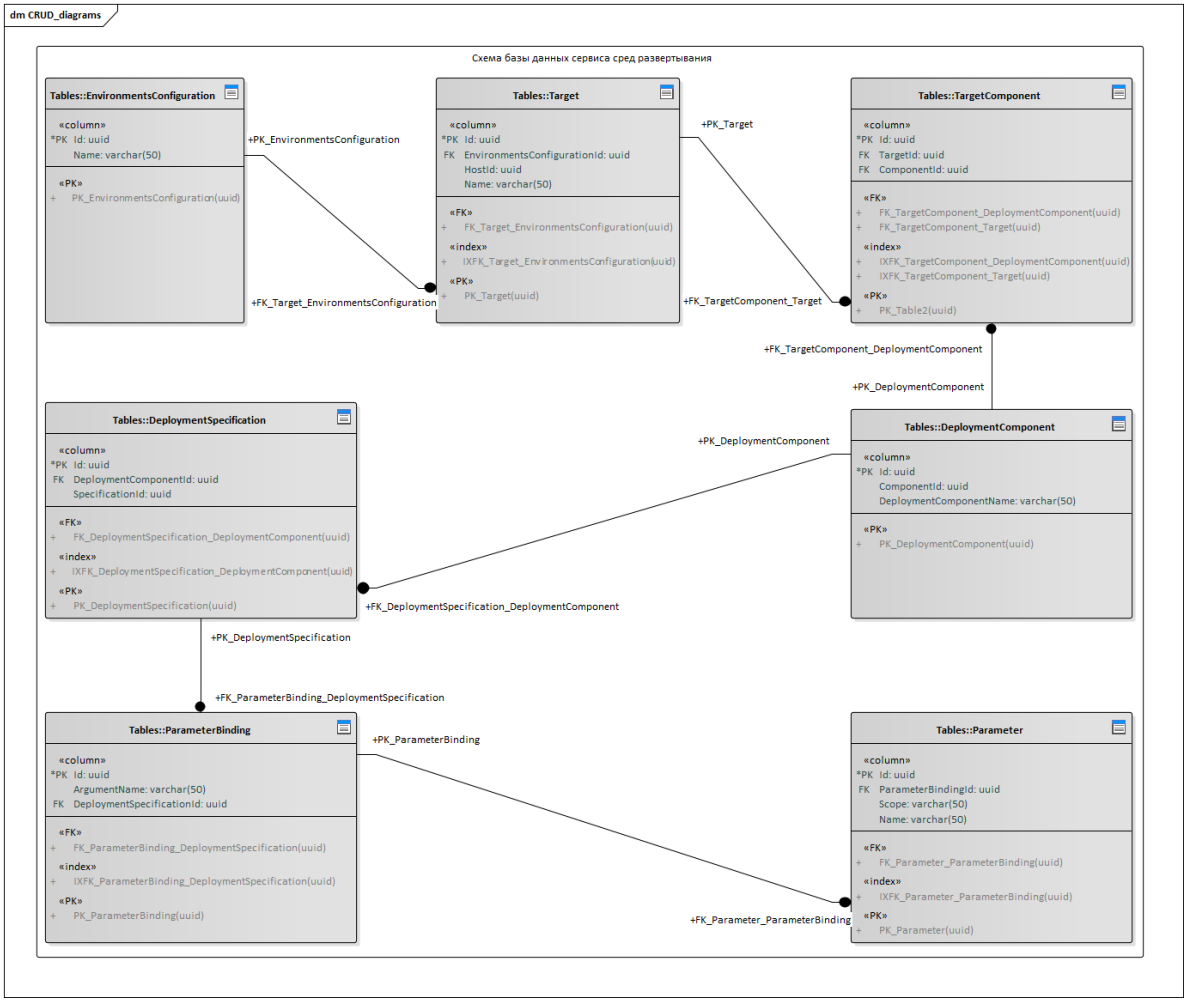


Рисунок 3.14 – Структура базы данных сервиса сред развертывания

В микросервисе развертывания хранение данных реализовано с целью обеспечения отслеживания версий установленного программного обеспечения и возможности возврата к предыдущей версии. Схема базы данных сервиса развертывания представлена на рис. 3.15.

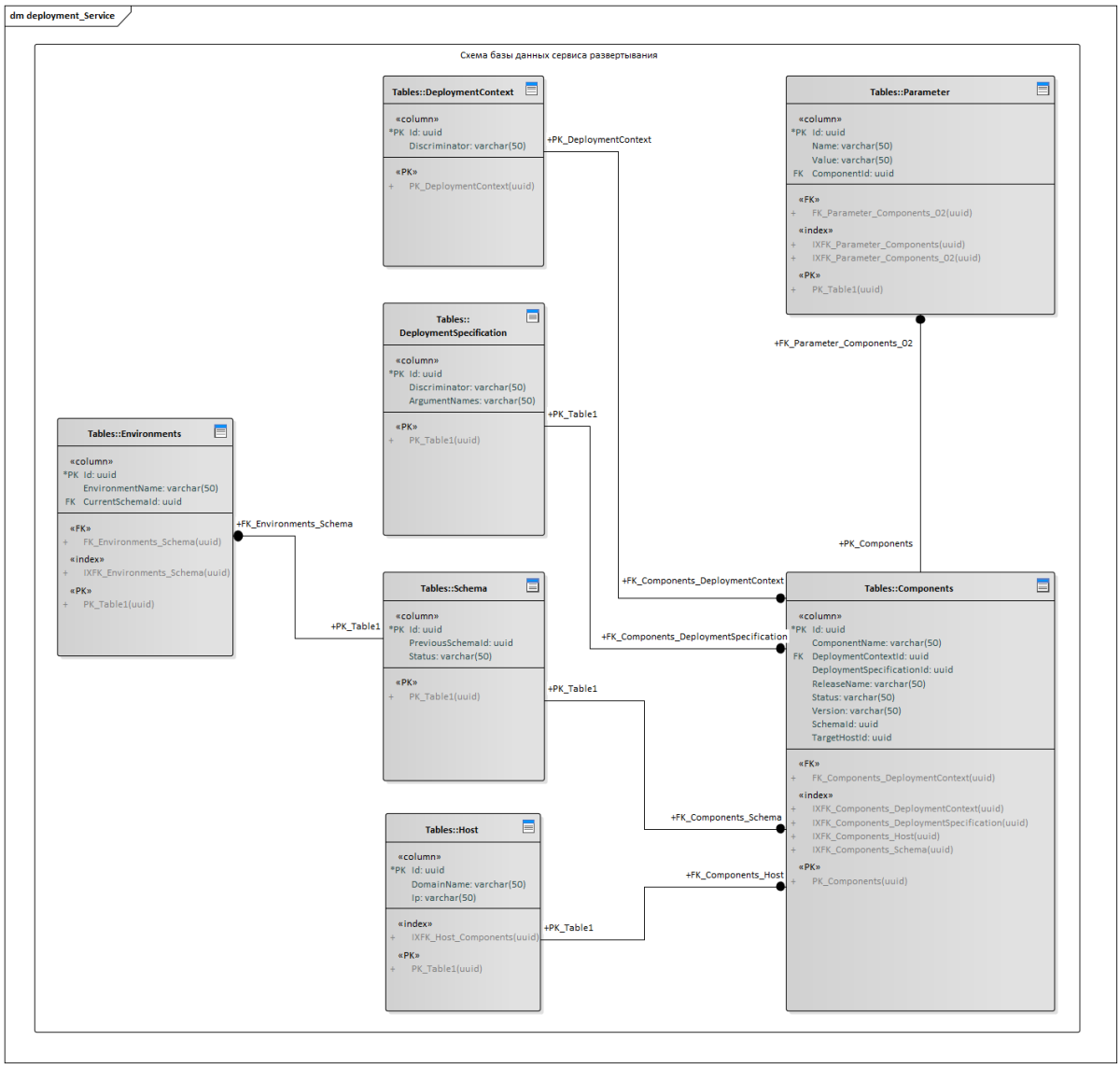


Рисунок 3.15 – Структура базы данных сервиса развертывания

3.5 Разработка интерфейса

При разработке интерфейса используется система построения клиентских приложений WPF, а также, шаблон проектирования MVVM – модель, представление, модель представления.

Данный подход позволяет разделить приложение на три функциональные части:

– модель — основная логика программы (работа с данными, вычисления, запросы и так далее);

– представление — вид или представление (пользовательский интерфейс);

– модель представления — модель представления, которая служит прослойкой между View и Model.

Такое разделение позволяет ускорить разработку и поддерживаемость программы — можно менять один компонент, не затрагивая код другого.

Как правило, требования к приложению могут меняться с течением времени. Могут появиться новые бизнес-возможности и проблемы, могут стать доступными новые технологии, или даже отзывы клиентов во время цикла разработки могут существенно повлиять на требования к приложению. Поэтому важно писать приложения так, чтобы они были гибкими и могли быть легко изменены или расширены в будущем. Это может потребовать архитектуру, которая позволит отдельным частям приложения разрабатываться и тестироваться независимо, и которые могут быть изменены или обновлены позже, в изоляции, без ущерба для остального приложения. Для упрощения расширяемости и поддержки проекта в будущем будет использован фреймворк Prizm [53], который является эффективным средством для решения этих проблем. Приложение разделяется на некоторое количество дискретных, слабо связанных, полунезависимых модулей, которые затем могут быть легко интегрированы в приложение «оболочку» для формирования цельного решения. Интерфейс построен с применением библиотеки стилей AdonisUI. Ниже представлены скриншоты пользовательского интерфейса различных модулей. Пользователь может создать релиз-группу, компоненты, цели и среду развертывания.

Пользователь создает коллекцию релизов (рис. 3.16).

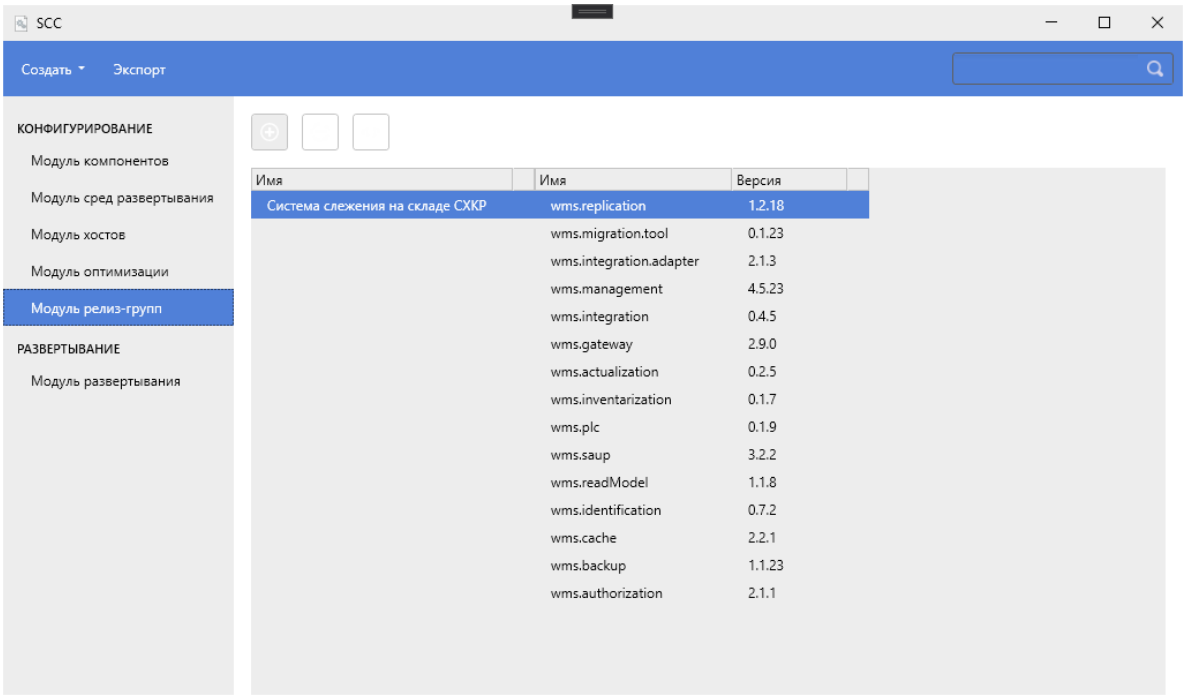


Рисунок 3.16 – Модуль релизных групп

После создания коллекции релизов необходимо добавить нужные программные компоненты в модуле компонентов (рис. 3.17).

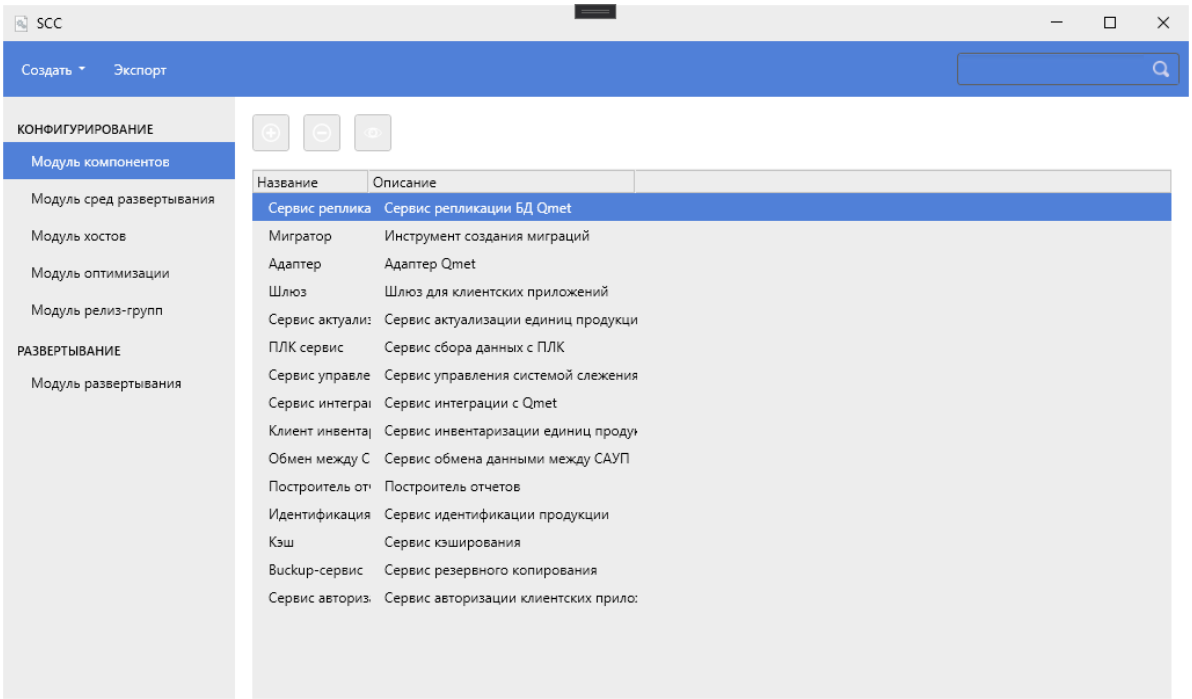


Рисунок 3.17 – Модуль компонентов

Когда созданы программные компоненты, нужно добавить целевые сервера, на которые необходимо установить программные компоненты (рис. 3.18).

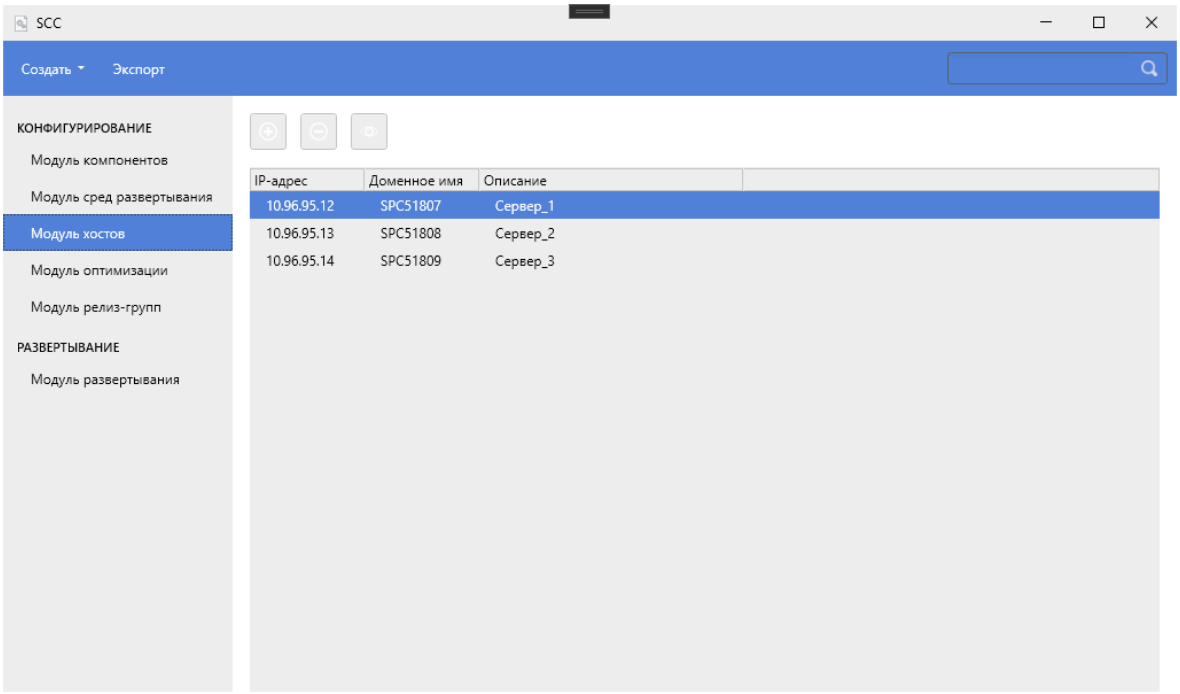


Рисунок 3.18 – Модуль хостов

Среда развертывания создается путем выбора целевых серверов и компонентов, предназначенных для установки на них (рис. 3.19). Пользователь может назначить установку программных компонентов как вручную, так и автоматически. В последнем случае достаточно добавить все компоненты на один выбранные сервер и перейти в модуль оптимизации.

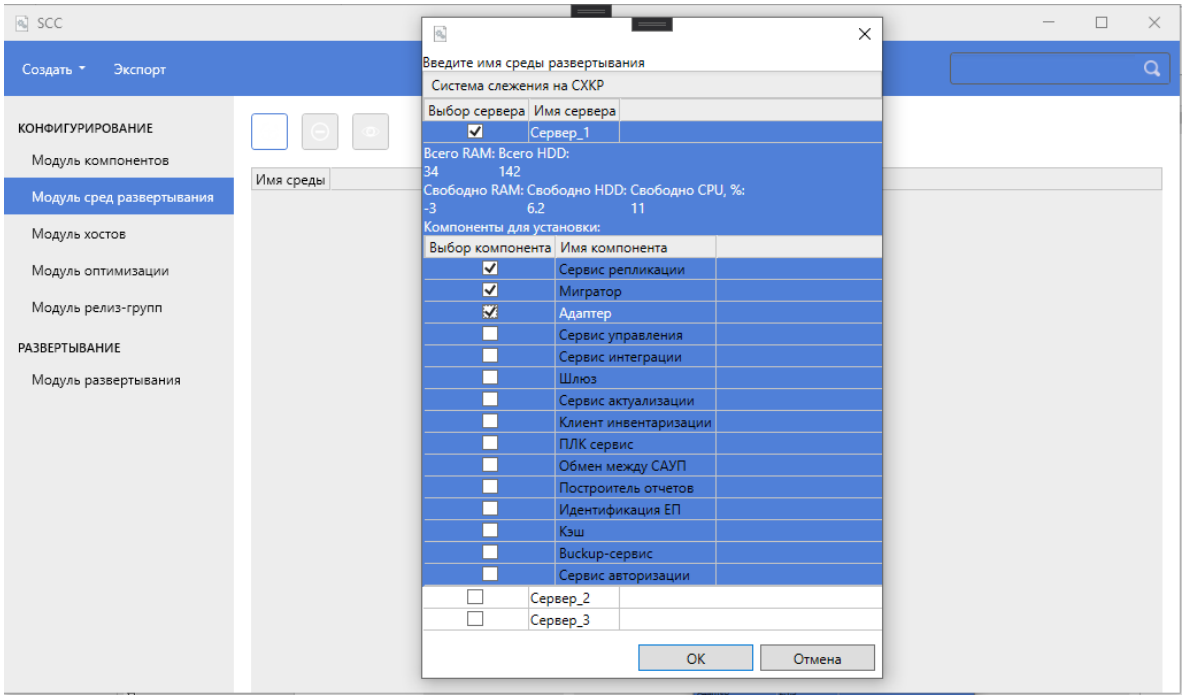


Рисунок 3.19 – Модуль сред развертывания

Когда пользователь создал среду развертывания, он может оптимизировать расположение программных компонентов на ней, запустив механизм оптимизации путем нажатия на кнопку «Оптимизировать». На рис. 3.20 показан результат работы алгоритма оптимизации.

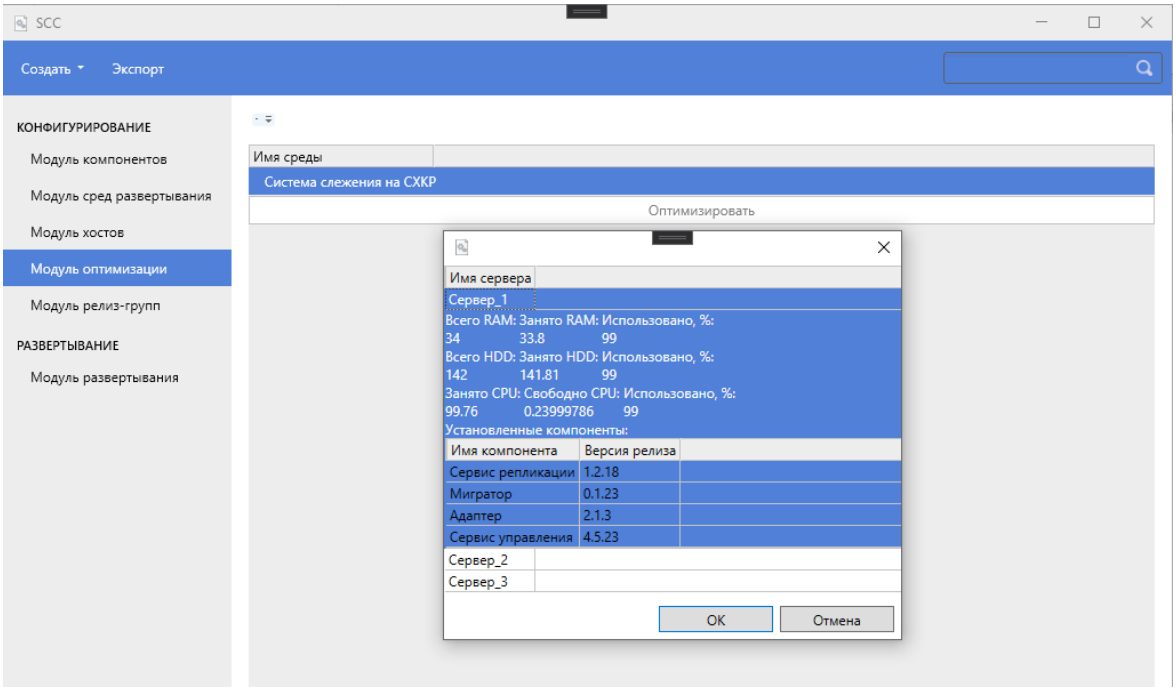


Рисунок 3.20 – Модуль оптимизации

Интерфейс модуля развертывания представлен на рис. 3.21.

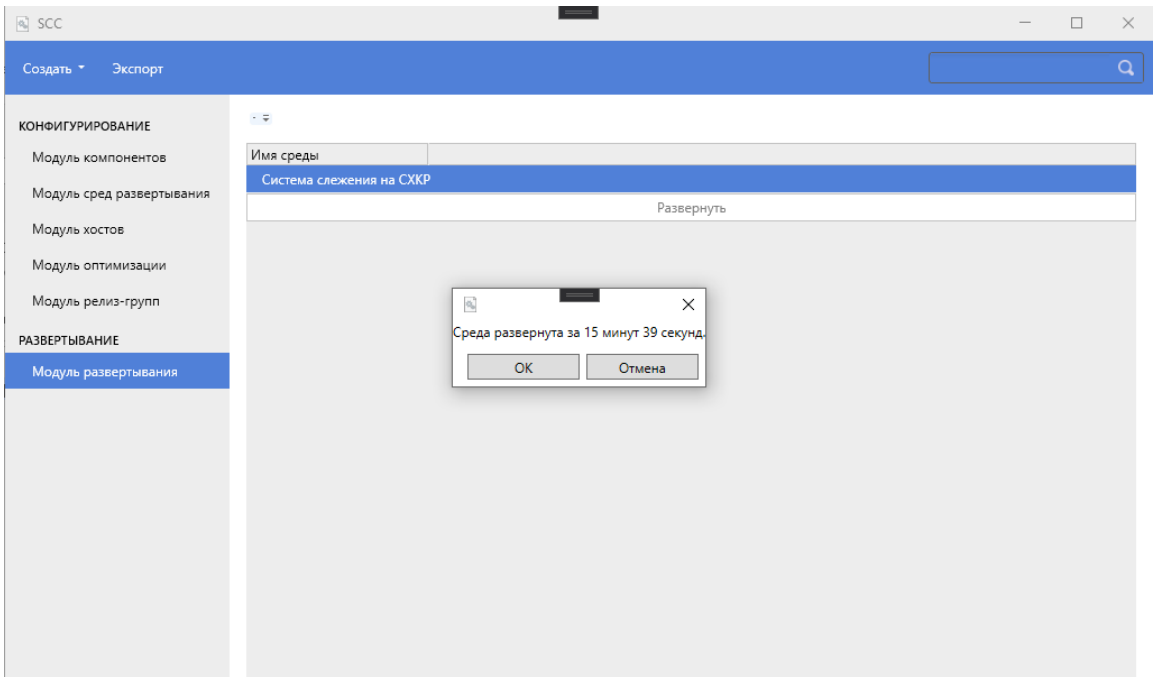


Рисунок 3.21 – Модуль развертывания

Разворачивание программных компонентов на серверах происходит в автоматическом режиме с последующим оповещением пользователя об

окончании работы, либо о том, что произошла ошибка и изменения отменены (рис. 3.21).

3.6 Выводы по главе 3

Перед началом разработки системы автоматического развертывания приложений в облачной инфраструктуре в данной главе был определен список основных функциональных требований. Согласно данному списку, был определен стек технологий, которые будут применены при разработке данной системы. Принято решение, что система будет реализовывать клиент-серверную архитектуру. В качестве клиента будет выступать приложение, созданное с использованием системы построения клиентских приложений WPF. На серверной стороне программное обеспечение представляет из себя микросервисное приложение на платформе .NET Core, предоставляющее API для клиентских программ и хранящее информацию в базе данных postgresql. Единой точкой доступа для клиентских программ служит шлюз Ocelot. Основным функционалом приложения является возможность автоматически оптимизировать расположение программных компонентов на виртуальных серверах и развернуть данные компоненты в облачной инфраструктуре.

4 ГЛАВА. ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ

4.1 Трудозатраты на разработку и отладку программы

Одной из составляющей статей калькуляции себестоимости проекта является оплата труда. В табл. 4.1 представлена трудоёмкость работ по разработке системы развертывания приложений в облачной инфраструктуре.

Таблица 4.1 – Трудоёмкость работ по разработке системы развертывания приложений в облачной инфраструктуре

Наименование работ	**Трудоёмкость, чел./дни**	
	Руководитель	**Разработчик**
Обзор существующих средств и систем. Проведение патентного обзора.	-	1
Обзор информационной сети, в которой предполагается внедрение системы.	-	1
Разработка требований для внедрения и успешного функционирования системы.	0,5	2
Разработка функциональной схемы и алгоритма работы системы	-	3
Математические расчёты	-	5
Разработка программного симулятора среды	-	25
Разработка программы	-	30
Тестирование и отладка	1	7
Пояснительная записка	-	23
Итого:	1,5	97

Расчёт себестоимости осуществляется по таким направлениям [10]:

– заработная плата.

– отчисление страхового взноса.

– издержки на амортизацию сервера.

– расходы потребителя, связанные с эксплуатацией программы.

Заработная плата разработчиков системы рассчитывается на основании таких данных:

1. Трудоёмкость выполнения работ, где $Т_{Руководителя}$ = 1,5 чел. /дней, $Т_{Разработчика}$ = 97 чел./дней.

2. Ставка в день руководителя проекта $Д_{Руководителя}$ = 4 000 руб.

3. Ставка в день разработчика $Д_{Разработчика}$ = 2 000 руб.

Заработную плату исполнителей проекта можно рассчитать по формуле 4.1:

$$С_{оз} = Т_{Руководителя} \cdot Д_{Руководителя} + Т_{Разработчика} \cdot Д_{Разработчика} \quad (4.1)$$

где $С_{оз}$ – заработная плата исполнителей проекта, руб., $Т_{Руководителя}$ – время трудозатрат руководителя, чел/дней, $Д_{Руководителя}$ – ставка в день руководителя проекта, руб., $Т_{Разработчика}$ – время трудозатрат разработчика, чел/дней, $Д_{Разработчика}$ – ставка в день разработчика проекта, руб.

$С_{оз}$ = 1,5 · 4 000 + 97 · 2 000 = 200 000 руб. Отчисления от заработной платы страхового взноса составляют 30 % и рассчитываются по формуле 4.2:

$$С_{СН} = С_{ОЗ} \cdot СТ, \quad (4.2)$$

где $С_{СН}$ - заработная плата исполнителей проекта, руб., СТ – процентная ставка страхового взноса, %.

$С_{СН}$ = 200 000 · 0,3 = 60 000 руб.

Издержки на амортизацию сервера определяются линейным методом по государственным нормам [1]. За год эксплуатации сумму амортизации можно посчитать по формуле 4.3:

$$А_{Г} = \frac{К_{об} \cdot Н_{ам}}{100} \quad (4.3)$$

где $К_{об}$ – балансовая стоимость оборудования, руб., $Н_{ам}$ – норма амортизации = 20 %.

Балансовая стоимость сервера, на котором развернута разработанная система составляет 142 000 рублей. Амортизация за год составляет 284 руб.

Расходы потребителя, связанные с эксплуатацией программы, определяются по формуле 4.4:

$$Р_{э.п} = Вр_{п.п} \cdot С_{м.вр}, \quad (4.4)$$

где $Р_{э.п}$ – эксплуатационные расходы потребителя, руб, $Вр_{п.п}$ – объем машинного времени в течение года, необходимый для решения данной задачи с использованием программы, ч., $С_{м.вр}$– стоимость одного часа машинного времени, руб./ч.

Стоимость одного часа работы сервера по данным предприятия составляет 0,78 рублей.

$Р_{э.п}$ = 365 · 24 · 0,78 = 6 832 рубля в год компания расходует на использование данной системы. Подведем итоги затрат в таблице 4.2.

Таблица 4.2 – Себестоимость проекта.

Статья затрат	Сумма, руб/год
Заработная плата исполнителей	200 000
Отчисления на страховые взносы	60 000
Расходы потребителя, связанные с эксплуатацией программы	6 832
Амортизация сервера	284
Итог:	267 116

4.2 Расчет экономической эффективности

Для расчета эффективности внедрения данной системы необходимо определить количество денежных средств, которое расходует предприятие на приобретение нового сервера при нехватке ресурсов на существующей серверной инфраструктуре для развертывания нового приложения.

Как показали эксперименты по реструктуризации существующей инфраструктуры с применением разработанного программного обеспечения, возможно высвободить ресурсы на существующей инфраструктуры до 13%. При текущих тенденциях развития информационных систем на предприятии внедрение нового проекта происходит каждый квартал, а значит, существует необходимость закупки нового оборудования под новый проект при нехватке ресурсов на существующих серверах. При текущей загруженности части серверов, на которых развернуты системы адресного учета продукции, существует возможность снизить количество занимаемых виртуальных машин и высвободить существующие вычислительные мощности для использования их под новые внедряемые проекты. На данный момент в эксплуатации находится 11 систем адресного учета продукции, которые, в общей сложности, занимают 154 виртуальных сервера (по 2 физических сервера на каждую систему).

Соответственно, в использовании находится 22 физических сервера. Перераспределение программных компонентов позволяет снизить количество занимаемых виртуальных серверов до 20, высвобождая, таким образом, 2 физических сервера.

Цена блейд-системы серверов, закупаемой предприятием под внедряемый проект, составляет 282 000 рублей.

Проект внедрения систем адресного учета продукции рассчитан на 6 лет. Учитывая тот факт, что развертывание новой системы учета продукции происходит ежеквартально, то предприятие закупает 8 физических серверов в год. При применении разработанной системы впоследствии каждый год будет происходить высвобождение еще 13% серверной инфраструктуры, что составляет 1 физический сервер в год.

Тогда годовая экономия эксплуатационных расходов рассчитывается по формуле 4.5 [15]:

$$Э = Р_{э.руч} \cdot Р_{э.п}, \quad (4.5)$$

где Э – годовая экономия эксплуатационных расходов у одного потребителя, руб., $Р_{э.руч}$ – эксплуатационные расходы потребителя при решении задачи без применения разработанной системы, руб., $Р_{э.п}$ – эксплуатационные расходы потребителя, руб.

Экономия эксплуатационных расходов в первый год составит:

Э = 282 000 · 2 – 6832 = 557 168 руб., а впоследствии

Э = 282 000 – 6 832 = 275 168 руб.

Срок окупаемости программного продукта рассчитывается по формуле 4.6.

$$Т_{ок} = \frac{Р_{кап}}{Э}, \quad (4.6)$$

$Т_{ок}$ = 267 116 ÷ 557 168 = 0,48 года или 5,75 месяца.

Годовой экономический эффект – показатель абсолютный. Он определяется как разность между годовой экономией и долей капитальных затрат, относимых на этот год. Годовой экономический эффект рассчитывается по формуле 4.7.

$$ЭЭ = Э - Е_н \cdot Р_{кап}, \quad (4.7)$$

где $Е_н$ – нормативный коэффициент эффективности дополнительных капитальных вложений, равный 0,15.

Экономия расходов в первый год составляет 557 168 рублей, соответственно, ЭЭ = 557 168 – 0,15 · 267 116 = 517 100,6 рублей.

Экономия расходов в последующие годы составляет 275 168 рублей, соответственно, ЭЭ = 275 168 – 0,15 · 6 832 = 274 143,2 рубля.

В таблице 4.3 представлены итоговые результаты технико-экономического обоснования проекта.

Таблица 4.3 – Итоговые результаты технико-экономического обоснования проекта.

Наименование параметра	**Значение параметра**
Себестоимость разработки	267 116 рублей
Годовая экономия эксплуатационных расходов	275 168 рублей
Срок окупаемости	5,75 месяцев
Годовой экономический эффект	274 143,2 рубля

4.3 Выводы по главе 4

В данной главе было проведено обоснование экономической эффективности разрабатываемой системы. В результате анализа затраченных средств и срока окупаемости при внедрении данной системы выяснено, что проект оправдывает затраченные на него средства менее, чем за полгода и позволяет сэкономить более 270 тысяч рублей в год.

ЗАКЛЮЧЕНИЕ

В работе рассмотрены факторы, обуславливающие необходимость разработки системы автоматического развертывания микросервисных приложений. В качестве алгоритма решения задачи оптимизации количества задействованных серверов применен генетический алгоритм. Разработана система автоматического развертывания микросервисных приложений в облачной инфраструктуре. Проведен анализ экономической эффективности данной системы. В главе 1 проанализированы существующие на данный момент программные продукты, позволяющие балансировать использование ресурсов и сетевую нагрузку на сервера ЦОД. Анализ показал, что такие средства используются на уровне автоматизированных систем управления предприятием, а готовых систем, позволяющих автоматически хостировать приложения на уровне автоматизированных систем управления технологическим процессом нет. В ходе патентного поиска по базам данных ФИПС также не было найдено готовых решений, однако, использованы полезные наработки. По результатам поиска существующих решений выяснено, что необходима разработка собственного решения.

Перед проектированием системы развертывания приложений поставленная в первой главе задача минимизации количества занимаемых серверов в облачной инфраструктуре предприятия была сформулирована в терминах комбинаторной оптимизации, в результате чего была разработана математическая модель, описывающая задачу распределения программных компонентов на виртуальные сервера. Выяснено, что данная задача является классической NP-полной задачей в теории комбинаторной оптимизации, соответственно, принято решение использовать для ее решения эвристические алгоритмы. С целью выбора наиболее подходящего алгоритма по критериям, определенным в главе 2, было выполнено сравнение ряда существующих эвристических алгоритмов путем их программной

реализации в прототипе инфраструктурной среды и запуска данного прототипа с применением тестовой выборки данных. Прототип инфраструктурной среды включает в себя симулятор сетевой инфраструктуры и непосредственно приложение, реализующее вышеописанные алгоритмы. В результате сравнения показателей работы алгоритмов на тестовой выборке, выяснено, что наиболее подходящим для решения задачи распределения программных компонентов на виртуальные сервера является генетический алгоритм, для него разработана и проверена на практике функция приспособленности, количество особей в популяции, вероятность мутации. Скрещивание особей выполняется многоточечным методом.

Перед началом разработки системы автоматического развертывания приложений в облачной инфраструктуре в третьей главе был определен список основных функциональных требований. Согласно данному списку, был определен стек технологий, которые будут применены при разработке данной системы. Принято решение, что система будет реализовывать клиент-серверную архитектуру. В качестве клиентского будет выступать приложение, созданное с использованием системы построения клиентских приложений WPF, фреймворка модульных приложений Prizm и библиотеки стилей AdonisUI. На серверной стороне программное обеспечение представляет из себя микросервисное приложение на платформе .NET Core, предоставляющее API для клиентских программ и хранящее информацию в базе данных postgresql. В качестве ORM использован EntityFramework Core. Единой точкой доступа для клиентских программ служит шлюз Ocelot. Основным функционалом приложения является возможность автоматически оптимизировать расположение программных компонентов на виртуальных серверах и развернуть данные компоненты в облачной инфраструктуре.

После разработки данной системы проведено обоснование ее экономической эффективности. В результате эксперимента на серверах системы адресного учета продукции предприятия «Северсталь-инфоком» выяснено, что применение системы автоматического развертывания микросервисных приложений в облачной инфраструктуре помогает сократить в среднем 13% серверных ресурсов. Анализ затраченных средств и срока окупаемости при внедрении данной системы показывает, что проект оправдывает затраченные на него средства менее, чем за полгода и позволяет сэкономить более 270 тысяч рублей в год.

СПИСОК ЛИТЕРАТУРЫ

1. Об утверждении Положения по бухгалтерскому учету "Учет основных средств" ПБУ 6/01 – Доступ из справ.-правовой системы КонсультантПлюс. – Текст: электронный.

2. Патент РФ № 2005130256/09, 06.11.2003. Управление ресурсами сервера, анализ и предотвращение вторжения к ресурсам сервера // Патент России № 2316045. 27.02.2006. Бюл. № 6. / Ч. Сэмпл.

3. Патент РФ № 2011132619/08, 04.08.2011. Система и способ оптимизации использования ресурсов компьютера // Патент России № 2475819. 04.08.2011. Бюл. № 5. / Зайцев О.В.

4. Патент РФ № 2015109182, 16.03.2015. Способ и система интеллектуального управления распределением ресурсов в вычислительных средах // Патент России № 2609076. 16.03.2015. Бюл. № 20. / Хантимиров Р.И.

5. Патент РФ № 2017116433, 10.11.2015. Система управления и диспетчеризации контейнеров // Патент России № 2666475. 07.09.2018. Бюл. № 25. / Синх Д., Суарес Э., Серстон У. [и др.].

6. Патент РФ № 2018117280, 12.10.2016. Программно-определяемая автоматизированная система и архитектура // Патент России № 2729885. 13.08.2020. Бюл. № 32. / Шове А., Вилхем Ф., Харриман М. [и др.].

7. Алексеева, Е.В. Генетический алгоритм для конкурентной задачи о р-медиане / Е.В. Алексеева, А.В. Орлов // Труды 14 Байкальской международной школы-семинара "Методы оптимизации и их приложения". – Том 1. – Северобайкальск, 2008. – С. 570-585.

8. Базаров, И. П. Термодинамика и статистическая физика. Теория равновесных систем / И.П. Базаров, Э.В. Геворкян, П.Н. Николаев. — Москва: МГУ, 1986. — С. 312.

9. Батищев, Д.И. Применение генетических алгоритмов к решению задач дискретной оптимизации / Д.И. Батищев, Е.А. Неймарк, Н.В. Старостин. – Нижний Новгород. – 2007. – С. 85.

10. Белов, А.М.. Экономика организации (предприятия): практикум / А.М. Белов, Г.Н. Добрин, А.Е. Карлик // Учебное пособие для вузов. – Москва: ИНФРА-М, 2013. – С.305.

11. Береснев, В. Л. Экстремальные задачи стандартизации / В.Л. Береснев. – Новосибирск: Наука, – 1978. – С. 336.

12. Бураков, М.В. Генетический алгоритм: теория и практика / М. В. Бураков. – СПб.: ГУАП, 2008. – С. 164.

13. Дейт, К. Введение в системы баз данных / К. Дейт. – К.; М.; СПб.: Изд.дом "Вильямс", 2000. – С. 298.

14. Ким, Д. Руководство по DevOps. Как добиться гибкости, надежности и безопасности мирового уровня в технологических компаниях / Д. Ким, П. Дебуа, Д. Уиллис. – Москва: Манн, Иванов и Фербер, 2018. – С. 512.

15. Киселева, А.В. Экономика предприятия / А.В.Киселева, Ю.В.Кудряшова // Учебно-методическое пособие. – Череповец: ЧГУ, 2009. – С. 80.

16. Клещева, И.В. Оценка эффективности научно-исследовательской деятельности студентов / И.В. Клещева. – СПб: НИУ ИТМО, 2014. – С. 91.

17. Кормен, Т Алгоритмы: построение и анализ / Т. Кормен. – Москва: «Вильямс», 2006. – С. 1296.

18. Нужнов, Е.В. Трехмерная упаковка несвязных элементов на основе эвристических процедур / Е.В. Нужнов, А.В. Барлит. – Таганрог: ТРТУ, 2002. – С. 23.

19. Ньюмен, С. От монолита к микросервисам / С. Ньюмен. – Санкт-Пеетербург: БХВ-Петербург, 2021. – С. 272.

20. Панченко, Т. В. Генетические алгоритмы / Т.В. Панченко. — Астрахань: Астраханский университет, 2007. — 87 с.

21. Попова, Е.П. Автоматизированные системы управления технологическими процессами / Е.П. Попова. – Краснодар: ГБПОУ КК КТК, 2015. – С. 44.

22. Ренцо, Э. Контейнеризация с применением Ansible 2 / Э.Ренцо. – Бирмингем: Пакт паблишинг, – 2018. – 355 с.

23. Скит Д. C# для профессионалов. Тонкости программирования // Д. Скит. – Вильямс, 2014. – С. 608.

24. Швец, А. Погружение в паттерны проектирования / А. Швец. – Интернет-издание, 2018. – 306 с.

25. Еремеев, А.В. Разработка и анализ генетических и гибридных алгоритмов для решения задач дискретной оптимизации: диссертация кандидата физико-математических наук: защищена 05.13.16: / Еремеев А.В. – Омск, 2000. – С. 119.

26. Лекции по дискретной математике. Федеральное государственное бюджетное учреждение науки Институт математики им. С. Л. Соболева Сибирского: сайт. – URL: http://www.math.nsc.ru/LBRT/k5/TPR/lec6.pdf (дата обращения: 11.03.2021). – Текст: электронный.

27. Смирнов, А.В. О задаче упаковки в контейнеры / А.В. Смирнов // Общероссийский математический портал. – том 46. – выпуск 4(280). – С. 173-174.

28. Растригин, Л. А. Случайный поиск — специфика, этапы истории и предрассудки / Л.А. Растригин // Вопросы кибернетики. – №. 33. – 1978, С. 3–16.

29. Уральский, Н.Б. Оптимизация вычислительного процесса фитнесс-функции генетического алгоритма в распределённых системах обработки данных

/ Н.Б. Уральский, В.А. Сизов, Н.К. Капустин // Интернет-журнал «Науковедение». – 2015. – Т. 7, вып. 6. – С. 38-52.

30. Anderson, E. Software Engineering for Internet Applications / E. Anderson . – MIT Press, 2006 . – P. 399.

31. Evans, E. Domain-Driven Design: Tackling Complexity in the Heart of Software / E. Evans. – Addison-Wesley, 2004. – P. 560.

32. Gavin, R RabbitMQ in Depth / R. Gavin. – NY: Shelter Island, 2017. – P. 264.

33. Goldberg, D. E. Genetic algorithms in search, optimization, and machine learning / D.E. Goldberg. – Reading, MA: Addison-Wesley. – 1989. – P. 372.

34. Holland, J. H. Adaptation in natural and artificial systems / J.H. Holland. – A Bradford Book, 1975. – P. 232.

35. Khan, Z. Machining condition optimization by genetic algorithms and simulated annealing / Z Khan // Computers & Operations Research, 1967. – P. 647-657.

36. Kirkpatrick S. Optimization by simulated annealing / S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi // Science, 1983. – Vol. 220. – P. 671–680.

37. Martello, S. Knapsack problems: algorithms and computer implementations/ S. Martello // Library of Congress Cataloging-in-Publication Data, 1990. – P. 221-224.

38. Newman, S. Building Microservices: Designing Fine-Grained Systems / Sam Newman. – O'reilly, 2015. – P. 280.

39. Bremermann, H. J. Global properties of evolution processes / H. J. Bremermann // Natural automata and useful simulations. – London: Macmillan, 1966. – P. 3-42.

40. Jamal, A.M. An optimal batch size for a production system operating under a just-in-time delivery system / A.M. Jamal // International Journal of Production Economics, №32. – 1993. – P. 255-260.

41. Jansen, K. Approximation algorithms for time constrained scheduling / K. Jansen // Proceeding of Workshop on Parallel Algorithms and Irregularly Structured Problems. – IEEE, 1995. – P. 143–157

42. Jeong, C. Fast Parallel Simulated Annealing for Traveling Salesman Problem on SIMD Machines with Linear Interconnections / C. Jeong, M. Kim // Parallel Computing, 1991. – P. 221-228.

43. Kim, J.-U. Simulated annealing and genetic algorithms for scheduling products with multi-level product structure / J.-U. Kim // Computers & Operations Research, 1996. – P. 857-868.

44. Metropolis, N. Equation of State Calculations by Fast Computer Machines / N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller // Chemical Physics, 1953. – P. 1087-1092.

45. Minarolli, D. Tackling uncertainty in long-term predictions for host overload and underload detection in cloud computing / Dorian Minarolli, Artan Mazrekaj, Bernd Freisleben // Journal of Cloud Computing. – 2017. – №4. – P. 1-18.

46. Monnier, Y. A genetic algorithm for scheduling tasks in a real-time distributed system / Y. Monnier, J.P. Beauvis, J.M. Deplanche. – IEEE, 1998. – P.708–714

47. Rechenberg I. Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der Biologischen Information / I. Rechenberg. – Freiburg: Fromman, 1973. – P. 299.

48. Schwefel, H. P. Numerical optimization of computer models / H.P. Schwefel. – Chichester: Wiley, 1981. – P. 97.

49. Tindell, K. Allocating hard real-time tasks (an np-hard problem made easy) / K. Tindell, A. Burns, A. Wellings // Real-Time Systems, 1992. – №4. – P.17-25.

50. Yao, X. A new simulated annealing algorithm / X. Yao // International Journal of Computer Mathematics, 1995. – P. 161-168.

51. Yao, X. Call routing by simulated annealing / X. Yao. – International Journal of Electronics, 79 (4) 1995. – P. 379-387.

52. Command Query Separation: сайт. – URL:https://martinfowler.com/bliki/CommandQuerySeparation.html (дата обращения: 20.04.2021). – Текст: электронный.

53. Introduction to Prism: сайт. – URL: https://prismlibrary.com/docs/ (дата обращения: 17.04.2021). – Текст: электронный.

54. Балансировка нагрузки в облаках: сайт. – URL: https://habr.com/ru/company/cloud4y/blog/329416/ (дата обращения: 11.02.2021). – Текст: электронный.

55. Балансировка нагрузки и масштабирование соединений в Kubernetes: сайт. – URL: https://habr.com/ru/company/mailru/blog/493820/ (дата обращения: 11.02.2021). – Текст: электронный.

56. Классификация алгоритмов по временной сложности: сайт. – URL: https://studref.com/333680/informatika/klassifikatsiya_algoritmov_vremennoy_slozhnosti (дата обращения: 19.04.2021). – Текст: электронный.

57. Начало работы с Docker сайт. – URL: https://docs.microsoft.com/ru-ru/visualstudio/docker/tutorials/docker-tutorial (дата обращения: 15.02.2021). – Текст: электронный.

58. Начало работы с EF Core в веб-приложении MVC ASP.NET: сайт. – URL: https://docs.microsoft.com/ru-ru/aspnet/core/data/ef-mvc/intro?view=aspnetcore-5.0 (дата обращения: 11.04.2021). – Текст: электронный.

59. Реализация шлюзов API с помощью Ocelot: сайт. – URL: https://docs.microsoft.com/ru-ru/dotnet/architecture/microservices/multi-container-microservice-net-applications/implement-api-gateways-with-ocelot (дата обращения: 14.04.2021). – Текст: электронный.

60. Руководство swagger ui: сайт. – URL: https://starkovden.github.io/swagger-ui-tutorial.html (дата обращения: 18.04.2021). – Текст: электронный.

61. Руководство по использованию REST API: сайт. – URL: https://api.mail.ru/docs/guides/restapi/ (дата обращения: 17.04.2021). – Текст: электронный.

62. Управление ресурсами в Kubernetes: сайт. – URL: https://habr.com/ru/company/flant/blog/459326/ (дата обращения: 11.02.2021). – Текст: электронный.

ПРИЛОЖЕНИЕ А

В таблицах 1 и 2 представлены характеристики серверов первой и второй тестовой выборки. Первая и вторая тестовая выборка формируется, как комбинация серверов и сервисов, которые необходимо распределить на сервера. Используемые при этом сервисы представлены в таблице 3.

Таблица 1. Характеристики серверов первой тестовой выборки (5 серверов)

№ п/п	**Характеристики**			
	Имя сервера	**Тип операционной системы**	**Свободное место на жестком диске, Гб**	**Количество оперативной памяти, Гб**
1	Server_1	Windows	142	34
2	Server _2	Windows	54	22
3	Server _3	Windows	66	20
4	Server _4	Windows	36	20
5	Server _Linux	Linux	36	20

Таблица 2. Характеристики серверов первой тестовой выборки (10 серверов)

№ п/п	**Характеристики**			
	Имя сервера	**Тип операционной системы**	**Свободное место на жестком диске, Гб**	**Количество оперативной памяти, Гб**
1	Server_1	Windows	142	34
2	Server _2	Windows	54	22

Продолжение таблицы 2

3	Server _3	Windows	66	20
4	Server _4	Windows	36	20
5	Server _Linux	Linux	36	20
6	Server_5	Windows	142	34
7	Server _6	Windows	54	22
8	Server _7	Windows	66	20
9	Server _8	Windows	36	20
10	Server _Linux_2	Linux	36	20

Таблица 3. Характеристики сервисов

№ п/п	**Характеристики**			
	Имя сервиса	**Тип операционной системы**	**Количество занимаемого места на жестком диске, Гб**	**Количество необходимой оперативной памяти, Гб**
1	Сервис_1	Windows	5	1
2	Сервис_2	Windows	12	2
3	Сервис_3	Windows	3	1
4	Сервис_4	Windows	7	2
5	Сервис_Linux_1	Linux	8	2
6	Сервис_5	Windows	2	1
7	Сервис_6	Windows	10	2
8	Сервис_7	Windows	13	3

Продолжение таблицы 3

9	Сервис_8	Windows	4	2
10	Сервис_Linux_2	Linux	2	3
11	Сервис_9	Windows	5	1
12	Сервис_10	Windows	10	2
13	Сервис_11	Windows	3	1,5
14	Сервис_12	Windows	8,5	2
15	Сервис_Linux_3	Linux	11	2
16	Сервис_13	Windows	4	1
17	Сервис_14	Windows	12	2
18	Сервис_15	Windows	8	1,5
19	Сервис_16	Windows	6,3	2
20	Сервис_Linux_4	Linux	1	2
21	Сервис_17	Windows	1	0,5
22	Сервис_18	Windows	0,5	0,2
23	Сервис_19	Windows	1	0,2
24	Сервис_20	Windows	0,3	0,12
25	Сервис_Linux_5	Linux	0,1	0,1
26	Сервис_21	Windows	5	1
27	Сервис_22	Windows	12	2
28	Сервис_23	Windows	3	1
29	Сервис_24	Windows	7	2
30	Сервис_Linux_6	Linux	8	2
31	Сервис_25	Windows	2	1
32	Сервис_26	Windows	10	2
33	Сервис_27	Windows	13	3

Продолжение таблицы 3

34	Сервис_28	Windows	4	2
35	Сервис_Linux_7	Linux	2	3
36	Сервис_29	Windows	5	1
37	Сервис_30	Windows	12	2
38	Сервис_31	Windows	3	1,5
39	Сервис_32	Windows	8,5	2
40	Сервис_Linux_8	Linux	11	2
41	Сервис_33	Windows	4	1
42	Сервис_34	Windows	12	2
43	Сервис_35	Windows	8	1,5
44	Сервис_36	Windows	6,3	2
45	Сервис_Linux_9	Linux	1	2
46	Сервис_37	Windows	1	0,5
47	Сервис_38	Windows	0,5	0,2
48	Сервис_39	Windows	1	0,2
49	Сервис_40	Windows	0,3	0,12
50	Сервис_Linux_10	Linux	0,1	0,1

Данные в формате json

[{ "Name" : "Server_1",
"Os" : "Windows",
"HddFull" : 142,
"HddFree" : 142,
"RamFull" : 34,
"RamFree" : 34,
"CpuFull" : 100,
"CpuFree" : 100,
"Services" : [] },
{ "Name" : "Server_2",
"Os" : "Windows",
"HddFull" : 54,

```
    "HddFree" : 54,
    "RamFull" : 22,
    "RamFree" : 22,
    "CpuFull" : 100,
    "CpuFree" : 100,
      "Services" : []      },
  { "Name" : "Server_3",
    "Os" : "Windows",
    "HddFull" : 66,
    "HddFree" : 66,
    "RamFull" : 20,
    "RamFree" : 20,
    "CpuFull" : 100,
    "CpuFree" : 100,
      "Services" : []      },
  { "Name" : "Server_4",
    "Os" : "Windows",
    "HddFull" : 36,
    "HddFree" : 36,
    "RamFull" : 20,
    "RamFree" : 20,
    "CpuFull" : 100,
    "CpuFree" : 100,
      "Services" : []      },
  { "Name" : "Server_Linux",
    "Os" : "Windows",
    "HddFull" : 36,
    "HddFree" : 36,
    "RamFull" : 20,
    "RamFree" : 20,
    "CpuFull" : 100,
    "CpuFree" : 100,
    "Services" : []      }]
```

ПРИЛОЖЕНИЕ Б

Таблица 1. Данные для проведения анкетирования экспертов.

№ п/п	Критерий	Метрика	Важность	Коэффициент важности
1	Количество не занятых серверов	Больше - лучше	Высокая Средняя Низкая	1-3
2	количество северов с отрицательным значением свободной памяти	Меньше - лучше	Высокая Средняя Низкая	1-3
3	количество северов с положительным значением свободной памяти	Больше - лучше	Высокая Средняя Низкая	1-3
4	количество северов с отрицательным значением свободной оперативной памяти	Меньше - лучше	Высокая Средняя Низкая	1-3
5	количество северов с положительным значением свободной оперативной памяти	Больше - лучше	Высокая Средняя Низкая	1-3

Таблица 2. Результаты проведения экспертной оценки

№ критерия	№ эксперта	Оценки экспертов		Сумма рангов	Квадрат отклонения
		Важность	Коэффициент важности		
1	1	Средняя	2,3	12,8	0,37
	2	Высокая	3		
	3	Средняя	2		
	4	Высокая	2,6		
	5	Высокая	2,9		
2	1	Высокая	3	14,1	0,27
	2	Высокая	3		
	3	Средняя	2,3		
	4	Высокая	2,8		
	5	Высокая	3		
3	1	Средняя	2,2	7	0,44
	2	Средняя	1,5		
	3	Низкая	1,3		
	4	Низкая	1		
	5	Низкая	1		
4	1	Высокая	3	14	0,31
	2	Высокая	2,8		
	3	Высокая	3		
	4	Средняя	2,2		
	5	Высокая	3		

Продолжение таблицы 2

5	1	Низкая	1	6,2	0,26
	2	Средняя	1,6		
	3	Низкая	1		
	4	Средняя	1,5		
	5	Низкая	1,1		

Таблица 3. Данные для проведения анкетирования экспертов

№ критерия п/п	**Критерий**	**Метрика**	**Важность**	**Коэффициент важности**
1	Коэффициент занятости HDD	Больше - лучше	Высокая Средняя Низкая	0-1
2	Коэффициент занятости RAM	Больше - лучше	Высокая Средняя Низкая	0-1
3	Коэффициент занятости CPU	Больше - лучше	Высокая Средняя Низкая	0-1

Таблица 4. Результаты проведения экспертной оценки

№ критерия	№ эксперта	Оценки экспертов		Сумма рангов	Квадрат отклонения
		Важность	Коэффициент важности		
1	1	Низкая	0,2	1,35	0,15
	2	Низкая	0,15		
	3	Низкая	0,1		
	4	Средняя	0,5		
	5	Средняя	0,4		
2	1	Высокая	0,5	2,05	0,17
	2	Высокая	0,6		
	3	Средняя	0,4		
	4	Низкая	0,1		
	5	Высокая	0,45		
3	1	Средняя	0,45	2,25	0,13
	2	Высокая	0,5		
	3	Высокая	0,6		
	4	Высокая	0,5		
	5	Низкая	0,2		

Таблица 5. Критерии сравнения алгоритмов, определенные экспертной группой

Название критерия	**Единицы измерения**	**Метод расчета**
Время работы алгоритма	Минуты	Измерения времени работы с момента запуска до получения решения
Заполняемость	-	процентным соотношением кол-ва заполненных серверов к общему кол-ву серверов при увеличении кол-ва сервисов
Вероятность ошибочного решения	-	Отношение кол-ва ошибочных решений к общему числу запусков

Таблица 6. Данные для проведения анкетирования экспертов

№ критерия п/п	**Критерий**	**Единицы измерения**	**Важность**	**Коэффициент важности**
1	Время работы алгоритма	Количество минут	Высокая (до 1 минуты) Средняя (до 10 минут) Низкая (до 30 минут)	1-3

Продолжение таблицы 6

2	Заполняемость	нет	Высокая Средняя Низкая	1-3
3	Вероятность ошибочного решения	нет	Высокая Средняя Низкая	1-3

Таблица 7. Результаты проведения экспертной оценки

№ критерия	**№ эксперта**	**Оценки экспертов**		**Сумма рангов**	**Квадрат отклонения**
		Важность	**Предел измерений**		
1	1	Низкая	1	8	0.49
	2	Средняя	2		
	3	Средняя	2		
	4	Средняя	2		
	5	Низкая	1		
2	1	Высокая	3	13	0.49
	2	Высокая	3		
	3	Средняя	2		
	4	Средняя	2		
	5	Высокая	3		
3	1	Средняя	2	13	0.8
	2	Высокая	3		
	3	Высокая	3		
	4	Высокая	3		
	5	Низкая	1		