Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.03 Прикладная информатика**

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

По дисциплине «Микропроцессорные системы»

НА ТЕМУ:

МК-система управления приборами жилого помещения

Студент	ИУ6-75Б		В.Д. Шульман
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Руководитель			В.Я. Хартов
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2019

Оглавление

Реферат

Расчётно-пояснительная записка с. 60, рис. ?, табл. ?, источников ?, приложений ?.

ГЕНЕТИЧЕСКИЙ АЛГОРИТМ, СЕЛЕКЦИОННЫЙ МЕТОД, ОТБОР, АГЕНТ, ХРАНИЛИЩЕ ГЕНОВ, КОНЕЧНЫЙ АВТОМАТ, КЛЕТОЧНЫЙ АВТОМАТ, ОКРЕСТНОСТЬ КЛЕТОЧНОГО АВТОМАТА, ЭВОЛЮЦИОНИРУЮЩИЙ КЛЕТОЧНЫЙ АВТОМАТ

Рассматриваемыми объектами в данной работе являются генетические алгоритмы и методы отбора при решении прикладных задач, где в основе лежит использование цифровых автоматов.

Целью работы является поиск и нахождения примеров эффективного применения цифровых автоматов в генетических алгоритмах, а также формулирование основных принципов при построении моделей и использованием цифровых автоматов. В работе рассматриваются способы представления цифрового автомата и также подходы к его модификации в процессе работы генетического алгоритма. Также в данном исследовании рассматриваются некоторые подходы к выработке метода отбора и выявления оптимальных параметров для него.

Обозначения и сокращения

Генетический алгоритм – алгоритм, осуществляющий поиск решения задачи оптимизации или моделирования путем случайного комбинирования параметров и отбора лучших решений.

Селекция – процесс, в рамках которого производится отбор по определенным критерия наиболее оптимального решения.

Селекционный метод – алгоритм, согласно которому производится оценка решений и формирование нового поколения.

Агент – элементарная эволюционная единица, осуществляющая решение поставленной перед ней задачи.

Популяция – множество агентов.

Поколение – текущая популяция агентов, сформированная путем на основании предыдущих поколений, прошедших через селекционный отбор.

Жизненный цикл – период времени от появления агента в системе до его исключения из системы в результате работы селекционного алгоритма.

Генетический код – набор параметров и инструкций, которые выполнят агент в процессе своего жизненного цикла.

Мутация – случайное изменение генетического кода агента, приводящее к приближению или отдалению агента от оптимального решения.

Оптимальное решение – решение, которое по набору признаков является наиболее предпочтительным.

Эвристический алгоритм – алгоритм решения задачи, который не является гарантированно точным или оптимальным.

Детерминированный алгоритм – алгоритмический процесс, который выдает уникальный и предопределенный результат для заданных входных данных.

Цифровой автомат – математическая модель дискретного устройства, которое принимает и выдает сигналы, принимая различные состояния.

Абстрактный автомат – математическая абстракция, модель дискретного устройства, имеющая один вход, один выход, которое в каждый момент времени находится в одном из множества возможных состояний.

Бесконечный автомат - автомат, способный принимать бесконечное множество состояний.

Конечный автомат – автомат, у которого количество внутренних состояний, которые он может принимать, ограничено.

Элементарный автомат – автомат, который описывается как автомат Мура, имеет двоичный алфавит, обладает двумя внутренними состояниями, обладает полной системой переходов и системой выходов.

Клеточный автомат – дискретная модель, изучаемая в математики и теории вычислимости, представляющая из себя решетку ячеек, каждая из которых может принимать одно из доступных состояний согласно окрестности и правилу клеточного автомата.

ГА – генетический алгоритм.

ПГА – параллельный генетический алгоритм.

АА – абстрактный автомат.

КА – конечный автомат.

БА – бесконечный автомат.

ЭА – элементарный автомат.

Введение

На данный момент достаточно широкое распространение получили методы решения задач, основанные на использовании генетических алгоритмов. Эти методы в основном используются для решения задач, решение для которых либо слишком сложно найти аналитически, составив алгоритм, либо они являются настолько абстрактными, что решение для них в виде детерминированного алгоритма вовсе не существует. Генетические алгоритмы позволяют решать такие задачи.

Генетические алгоритмы (ГА) являются частью более общей области – эволюционного моделирования. В отличии от других технологий оптимизации, ГА содержат популяцию пробных решений, которые конкурентно управляются с помощью определенных операторов. Генетическим алгоритмам присуще итеративное обучение популяции агентов.

Генетические алгоритмы относится к классу эвристических алгоритмов поиска, т.е. они решают задачи, используя практические методы, которые не являются абсолютно точными и оптимальными, однако достаточно точны для решений поставленной перед ними задачи при заданных условиях [1].

На волне популярности искусственного интеллекта генетические алгоритмы являются поводом для постоянных дискуссий и исследований. Несмотря на то, что ИИ представляет из себя целый комплекс различных систем, такие как системы распознавания, анализа, обучения, адаптации и т.п., генетические алгоритмы вполне могу решить часть этих задач.

Генетические алгоритмы актуальны на данный момент тем, что уже сейчас способы решать широкий спектр задач. Генетические алгоритмы могут использоваться для управления процессорами, балансируя нагрузку в многопроцессорной системе, они могут использоваться для решения задач по поиску наилучшего соотношения веса/прочности/размера/плотности, для решения задач по размещению объектов различных форм на определенной площади (трассировка плат или нарезка ткани с наименьшими потерями, например) и т. д. и т. п.

Один из недавних коммерческих примеров: израильская компания Schema разработала программный продукт Channeling для оптимизации работы сотовой связи путем выбора оптимальной частоты, на которой будет вестись разговор. В основе этого программного продукта и используются генетические алгоритмы.

Генетические алгоритмы приобрели большую популярность на фоне новой волны подъема интереса к нейронным сетям. Сейчас довольно быстро развиваются технологии по распознаванию изображения и звука. Для обучения нейронных сетей также используется генетический алгоритм, где после каждого раунда нейронные сети сравниваются по своей эффективности, после чего часть их них отсеивается, а на основании оставшихся генерируется новое поколение нейронных сетей и измененными весами переходов.

Одно из больших достоинств генетических алгоритмов – это возможность распараллеливания вычислений при поиске решений. Т.к. агенты, которые пытаются решить задачу, чаще всего не должны взаимодействовать с друг-другом, то обработку каждого из них можно вести параллельно. Раньше это не давало больших преимуществ из-за одноядерных процессоров, но сейчас с развитием многопроцессорных архитектур и асинхронных языков программирования поиск решения с использованием генетических алгоритмов может проходить в разы быстрее.

Основной целью НИР является нахождение и оценка оптимальных подходов к реализации генетических алгоритмов с использованием цифровых автоматов и определению круга задач, для которых использование такого вида генетических алгоритмов было бы оправдано и эффективно.

1 Исследовательская часть

1.1 Генетические алгоритмы

1.1.1 Идея генетических алгоритмов

Идея генетических алгоритмов не нова. Еще в 60-ых годах прошлого века начались первые симуляции эволюционных процессов на тогдашних маломощных огромных ЭВМ. В 70-ых годах была разработана концепция эволюционного программирования с использованием конечных автоматов. Достаточно большую популярность генетические алгоритмы обрели после экспериментов Джона Холланда с так называемыми клеточными автоматами

Однако из-за скромных вычислительных мощностей длительно время генетические алгоритмы оставались в основном в теоретической области и почти не использовались для решения прикладных задач. Лишь к концу 80-годов с ростом вычислительны мощностей стало внедряться оборудование, включавшее в себя генетические алгоритмы для решения поставленных перед ними задач. В 89-ом году компания Axcelis выпустила свой первый продукт для персональных компьютеров, в котором применялись генетические алгоритмы. Носил он символичное название “Evolver”.

Сейчас существует много вариантов реализации генетических алгоритмов, каждый из которых больше подходит для решения определенных задач. Например, нейронные сети хорошо решают задачу распознавания изображений, но решить задачу самообучающегося ИИ нейронные сети так и не смогли, несмотря на то, что прошло несколько витков их популярности.

1.1.2 Сравнение генетических алгоритмов с детерминированными алгоритмами

Генетические алгоритмы применяются чаще для нахождения приближенного решения. Это обосновывается тем, что для сложной задачи чаще всего требуется найти не идеальное решение, а удовлетворяющее бы имеющимся требованиям [4]. При этом достижения идеального «оптимального» решения отходит на второй план. Однако при этом другие

методы, ориентированные на нахождение оптимального решения, из-за их чрезвычайно высокой сложности становятся и вовсе нереализуемыми [6].

Основные отличия генетических алгоритмов от традиционных методов:

1) генетические алгоритмы работают с кодами, в которых представлен набор параметров, напрямую зависящих от аргументов целевой функции;

2) для поиска генетический алгоритм использует несколько точек поискового пространства одновременно, а не переходит от точки к точке, как это делается в традиционных методах;

3) генетические алгоритмы в процессе работы не используют никакую дополнительную информацию;

4) генетический алгоритм использует как вероятностный правила для порождения новых точек, так и детерминированные правила для перехода от одних точек к другим.

1.1.3 Классификация генетических алгоритмов

Генетические алгоритмы являются частным случаем еще более обобщенной сферы – генетического моделирования (рис. 1), но генетические алгоритмы достаточно разнообразны и обладают собственной классификацией [3].

Рисунок 1 – Классификация алгоритмов эволюционного моделирования

ГА не обязательно относится только лишь к одно из возможных классификаций, т. к. не все из них являются взаимоисключающими. Генетический алгоритм может быть отнесен сразу к нескольким категориям, которые представлены на рисунке 1.

1.1.3.1 Стационарные генетические алгоритмы

Стационарные генетические алгоритмы подразумевают использовать подход, в котором популяция обновляется частями, а не вся сразу, Идея заключается в итеративном создании одного или двух потомков и добавления их непосредственно в популяцию, с предварительным исключением некоторых существующих агентов, чтобы освободить необходимое пространство.

Данный генетический алгоритм, по сравнению с поколенческим, использует примерно в два раза меньше памяти, т.к. всегда рассматривается только одна популяция. Также в этом алгоритме агенты-родители остаются в популяции на более длительное время, т. к. длительность их жизненного цикла обусловлена не «жесткими параметрами моделирования», а параметрами

самого агента, которые формируются под воздействием генетического алгоритма случайным образов.

Такой тип алгоритмов более характерен для реального мира, поколения не имеют между собой четких временных границ, а сменяемость поколений происходит не резко, «между раундами», а плавно, в процессе конкуренции.

1.1.3.2 Динамические генетические алгоритмы.

Противоположен по своим идеям стационарному генетическому алгоритму. Здесь подразумевается использования итеративного подхода, где после каждой итерации популяция проходит полное обновление, в рамках которого удаляются и добавляются агенты.

1.1.3.3 Поколенческие генетические алгоритмы

Поколенческие генетические алгоритмы подразумевают итеративное следованию определенной последовательности действий:

- создание первоначальной популяции;
- вычисление функций приспособленности для агентов популяции (оценивание);
- выборка агентов из текущей популяции (селекция);
- мутации агентов;
- вычисление функций приспособленности для всех агентов;
- формирование нового поколения;
- оценить результаты и, если они не удовлетворительны, то повторить, начиная с 3 пункта.

1.1.3.4 Адаптивные генетические алгоритмы

Основная идея адаптивного генетического алгоритма заключается в том, что такой алгоритм должен уметь изменять свои параметры в процессе работы. В качестве параметров здесь выступает размер популяции, вероятность мутации, количество удаляемых и добавляемых агентов, а также множество других параметров, которые зависят от специфики самой задачи. Целью

создания адаптивного алгоритма является ускорение поиска решения или уменьшения затрат ресурсов при его поиске (например, память).

1.1.3.5 Многоуровневые генетические алгоритмы.

Такие алгоритмы представляют из себя многоуровневую систему, где каждый уровень генетического алгоритма оптимизирует работу вышестоящего генетического алгоритма и так по цепочке. Говоря более формально, многоуровневый генетический алгоритм – это система, в которой генетические алгоритмы нижнего уровня (оптимизируемые генетические алгоритмы) оптимизируют генетические алгоритмы верхнего уровня.

Этот принцип часто используется в нейронных сетях, где вершины нейронной сети разбиты по слоям, представляющие собой уровни генетического алгоритма. При этом возникает очень сложная задача оптимизации, где необходимо вычислить оптимальное количество слоев для наиболее быстрого поиска решения задачи, т.к. никакой универсальной формулы, дающая ответ на этот вопрос, не существует.

1.1.3.6 Параллельные генетические алгоритмы.

Такие алгоритмы основаны на разбиении популяции на несколько отдельных подпопуляций, каждая из которых будет, независимо от других подпопуляций, обрабатываться генетическим алгоритмом. Одновременно с этим, разнообразные миграции агентов порождают обмен генетическим кодом среди популяций, которые, как правило, улучшают точность и эффективность алгоритма.

Выделяют три типа параллельных генетическим алгоритмов:

- глобальные однопопуляционные ПГА (master-slave);
- однопопуляционные ПГА;
- многопопуляционные ПГА.

Модель «хозяин-раб» характеризуется тем, что в алгоритмах такого типа селекция принимает во внимание целую популяцию, в отличии от двух других моделей.

В алгоритмах второго класса существует главная популяция, но оценка целевой функции распределена среди нескольких процессоров. Хозяин хранит популяцию, выполняет операции ГА и распределяет агентов между подчиненными. Они же лишь оценивают агентов. Однопопуляционные ГА пригодны для массовых параллельных компьютеров и состоят из одной популяции. Данный класс ПГА может быть эффективно реализован на параллельных компьютерах.

Третий класс - многопопуляционные ГА более сложная модель, так как она состоит из нескольких подпопуляций, которые периодически, по установленным правилам, обмениваются агентами. Такой обмен агентами называется миграцией и управляется несколькими параметрами. Такие ГА очень популярны, но достаточны сложны как для понимания, так и для реализации, потому что последствия от эффекта миграции, на данный момент, не полностью исследованы. В то же время многопопуляционные ГА имеют сходство с «островной моделью» в популяционной генетике, которая рассматривает относительно изолированные общины; поэтому параллельные ГА в некоторых случаях называют «островными» параллельными ГА.

1.1.4 Анализ предметной области

Предметная область информационной системы, реализующая генетические алгоритмы, включает в себя ряд объектов, данных и функций, которые её описывают. Они представлены на рис. 2.

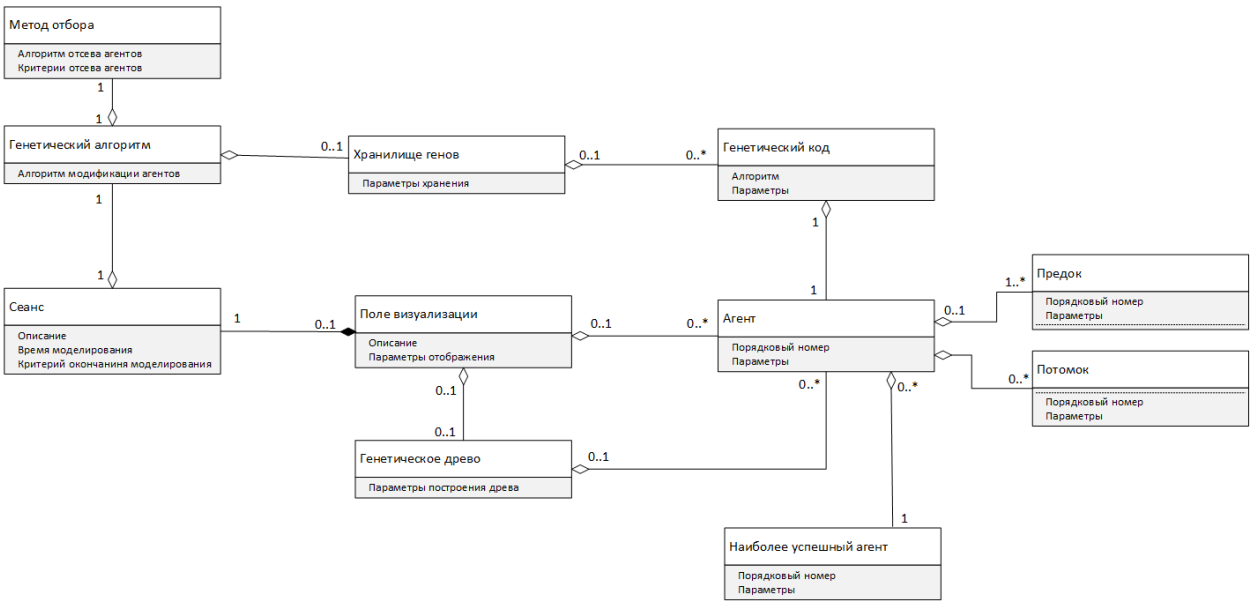

Рисунок 2 – Концептуальная схема предметной области

Объекты представляют собой действующие лица, между которыми идет обмен информацией и которые осуществляют некоторые функции над друг-другом.

1.1.4.1 Агент

Агент представляет из себя элементарную рабочую единицу ИС. Перед агентом ставятся задачи, для которой в системе идет поиск решения, как можно более приближенного к оптимальному. Существует множество агентов, параллельно решающих поставленную перед ними задачу. В начале процесса нахождения оптимального решения задачи все агенты находятся в схожих условиях и имеют идентичную конфигурацию. В течение некоторого отрезка времени в модели порождаются новые агенты с несколько отличной конфигурацией согласно генетическому алгоритму. Периодически с помощью метода отбора часть агентов удаляется из процесса поиска оптимального решения.

1.1.4.2 Хранилище генов.

Хранилище генов представляет из себя банк с данными. В этих данные содержится информацией о конфигурациях агентов. Хранилище генов содержит в себе информацию как о тех агентах, которые участвуют в текущем сеансе. Данные о конфигурации агентов из хранилища генов используются для

формирования конфигурации новых агентов в процессе работы генетического алгоритма.

1.1.5 Основные информационные структуры предметной области

Информационные структуры представляют из себя совокупность данных, которыми обмениваются объекты предметной области. Также данные используются функциями (алгоритмами) для получения результата, который может представлять из себя набор еще каких-то данных.

1.1.5.1 Генетический код.

Включает в себя набор параметров и инструкций, который использует агент в процессе своего жизненного цикла. Генетический код агента также можно определить, как его конфигурацию. Генетический код изменяется случайным образом в процессе работы генетического алгоритма. Генетический код агента, который наиболее эффективно решает поставленную перед ним задачу, является тем самым оптимальным решением, поиск которого производится в системе.

1.1.5.2 Генетическое древо.

Представляет из себя совокупность информации о родителях и потомках каждого агента, когда-либо появившегося в процессе сеанса поиска решения. Генетическое древо показывает какие ветви развития агентов имели какую численность в зависимости, и какая последовательность изменений привела к текущим генетическим кодам агентов

1.1.5.3 Поле визуализации.

Поле визуализации показывает текущее состояние сеанса работы генетического алгоритма.

Оно может в себя включать различные графические элементы:

- графики;
- диаграммы;
- графы состояний и переходов;
- поля пространств, где размещаются и перемещаются агенты;

– визуализации наполнения банков данных;

– генетическое древо.

Информация в поле призвана давать возможность объективно оценить текущие состояние сеанса и определить, был ли достигнут требуемый результат работы генетического алгоритма.

1.1.6 Основные процедуры предметной области

Процедуры (алгоритмы), реализуемые в системе, использующие генетические алгоритмы, необходимы для обработки имеющихся данных и формирования новых.

1.1.6.1 Генетический алгоритм

Наиболее важная часть ИС, определяющая всю логику ее работы и множество задач, которые эта система может решать.

Генетический алгоритм определяет каким образом будет происходить наследование генетического кода от агентов одного поколения к агентам последующего, а также какие случайные изменения будут происходить в генетическом коде новых агентов и с какой вероятностью.

1.1.6.2 Метод отбора

Это алгоритм, который осуществляет отсев агентов в процессе моделирования, решения поставленной задачи которых оказалось наименее эффективно согласно критериям оптимальности решения этой задачи.

1.1.6.3 Сеанс

Сеанс представляет из себя весь временной отрезок, в рамках которого агенты решают представленную перед ними задачу. Сеанс в зависимости от условий и специфики задач, стоящих перед ними, может проходить в различных формах:

– агенты решают поставленную задачу независимо, никак не контактируя и не взаимодействуя друг с другом;

– агенты решают поставленную задачу в одной среде, весь сеанс поделен на периоды времени, представляющие из себя циклы, после каждого

из которых происходит удаление и добавление агентов в зависимости от эффективности решения каждого;

– агенты решают задачу одновременно в одной среде соревнуясь друг с другом. При таком виде сеанса отбор осуществляется естественным образов в ходе конкуренции между агентами. При этом наиболее продвинутые агенты порождают других агентов, согласно генетическому алгоритму, поддерживая тем самым их общее количество. Этот тип сеанса больше всего напоминает естественные процессы, проходящие в живой природе.

1.1.7 Необходимые информационные технологии

Для успешного построения системы, которая будет обрабатывать данные с использованием генетического алгоритма, необходимо использование различных информационных технологий

1.1.7.1 Централизованная обработка информации

Централизованная обработка подразумевает, что система работа системы будет осуществляться на удаленном высокопроизводительном сервере.

Такой подход к обработке эффективен, если решаемые задачи требуют затрат больших вычислительным ресурсов или специализированного программного обеспечения. В случае с генетическими алгоритмами использование такого подхода будет преследовать цель нахождения оптимального решения в максимально сжатые сроки.

Ожидаемый эффект использования технологии – быстрое моделирование работы генетического алгоритма.

1.1.7.2 Децентрализованная обработка информации

Децентрализованная обработка информации подразумевает использование распределенной системы, где в роли вычислительных мощностей выступает компьютеры абонентов.

Этот подход подходит, если скорость работы генетического алгоритма не столь важна, сколько возможность проводить параллельно множество

таким процессов моделирования. При этом это дает каждому абоненту более широкие возможности по настройки параметров моделирования.

Ожидаемый эффект использования технологии – возможность вести множество моделирований работы генетического алгоритма, не ограничивая себя вычислительным мощностями центрального сервера системы.

1.1.7.3 Информационная технология поддержки принятия решений

Эта технология организует взаимодействие пользователя и вычислительной системы, где в результате обработки входных данные от пользователя, выдается рекомендация.

Технология поддержки принятия решения ориентирована на решение плохо структурированных задач. Использование генетического алгоритма позволяет решать такие задачи, где не найдено аналитическое решение, но при этом можно составить математическую модель, которая описывала бы предметную область и критерии оптимальности решения.

Ожидаемый эффект использования технологии – возможность принимать от пользователя набор данных, описывающий задачу, и находить максимально приближенное к оптимальному решение этой задачи с использованием генетического алгоритма.

Особенность применения – необходимо обеспечивать дополнительный анализ информации от пользователя для составления математической модели решаемой задачи.

1.2 Цифровые автоматы

Понятие абстрактного автомата [АА], позволяет рассматривать дискретные объекты с точки зрения алгоритмов их функционирования, то есть реализуемых последовательностей действий по преобразованию дискретной информации [2].

Абстрактным автоматом называют модель, которая описывается кортежем, состоящим из 5 элементов:

$$A = (X, Y, S, f_y, f_s)$$

Здесь первые три компонента – это непустые множества:

X – множество входных сигналов,

Y – множество выходных сигналов,

S – множество состояний.

Остальные два компонента кортежа являются характеристическими функциями:

f_y – функция выходов,

f_s – функция переходов АА из одного состояния в другое

Если множества X, Y, S – конечные, то такой АА называется конечным автоматов [КА]. В том случае, если бы хотя бы одно из множеств автомата было бесконечным, то и сам АА являлся бы бесконечным.

В данной исследовательской работе основным объектом для анализа являются именно КА, который, в отличии от БА, программно-реализуемым и может быть полностью представлен с помощью таблицы или графа переходов, что дает возможность их практического применения.

1.2.1 Классификация абстрактных автоматов

Для классификации автомата рассматривается множество признаков, например, определенность функции переходов и функции выходов, однозначность заданных функций, устойчивость состояний. Все три признака перечислены и определены в таблице 1.

Классификация абстрактных автоматов может быть представлена в виде схемы на рисунке 3.

Таблица 1

Признак	**Определение**
Определенность характеристических функций	В автоматах полностью определенных областью определения функций f_s и f_y является множество всех пар $(s_i, x_k) \in S \times X$, где $s_i \in S$, $x_k \in X$. В автоматах частично определенных либо обе характеристические функции, либо одна из них имеют областью определения строгое подмножество декартова произведения $S \times X$. Таким образом, характеристические функции подобных автоматов определены не для всех пар (s_i, x_k).
Однозначность функции переходов	В детерминированных автоматах выполняется условие однозначности переходов: если АА находится в некотором состоянии $s_i \in S$, то под воздействием произвольного входного сигнала $x_k \in X$ автомат может перейти в одно и только одно состояние $s_j \in S$, причем ситуация $s_i, = s_j$ вовсе не исключается. В автоматах вероятностных при воздействии одного и того же входного сигнала возможны переходы из состояния s_i в различные состояния из множества S с заданной вероятностью.
Устойчивость состояний	В устойчивых автоматах выполняется условие устойчивости: если автомат под воздействием входного сигнала $x_k \in X$ оказался в состоянии $s_i \in S$, то выход из него и переход в иное состояние возможен только при поступлении на вход автомата другого сигнала $x_z \in X$, $x_z \neq x_k$. Если условие устойчивости не выполняется хотя бы для одного состояния $s_j \in S$, то такой автомат называют неустойчивым.

Для практического применения чаще всего используются те автоматы, которые по перечисленным выше признакам являются полностью определенными, детерминированными и устойчивыми конечными автоматами.

Рисунок 3 – Классификация абстрактных автоматов

Также можно классифицировать автоматы по виду ее характеристической функции. Если аргументы характеристической функции являются только текущее значение входного сигнала и ткущее состояние, то такой автомат является автоматов 1 рода или автоматом Мили. Если выходные сигналы автомата зависят исключительно от текущего состояния автомата, то такой автомат принято считать автоматов 2 рода или автоматом Мура.

Пример графов переходов для автомата Мура и автомата Мили представлены на рисунках 4 и 5 соответственно/

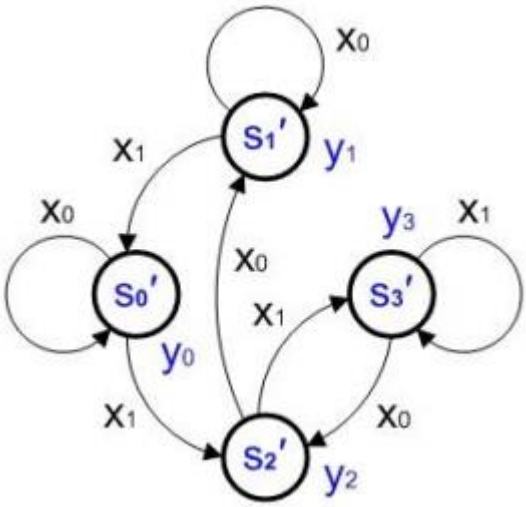

Рисунок 4 – граф переходов автомата Мура

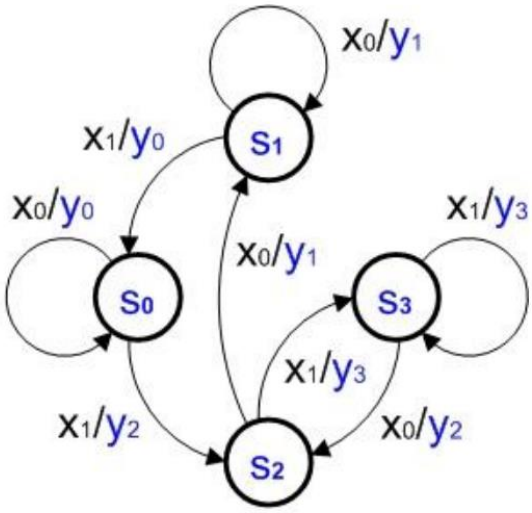

Рисунок 5 – граф переходов автомата Мили

Следует при этом отметить, что автомат Мили запаздывает на одни дискретный момент времени по входному сигналу по отношению к автомату Мура

Если в автомате присутствуют как свойства автомата Мура, так и автомата Мили, то такой автомат можно считать смешанным.

1.2.2 Базовая модель конечного автомата

Конечный автомат, в описание которого входят таким образом определенные множества, называют (n, p, q)-автоматом, а самим множествам усваивают наименование векторов, например, вектор входных сигналов, вектор состояний.

Все автоматы, и в том числе конечные, функционируют в дискретном исчислении времени. Моменты времени образуют ряд целых неотрицательных чисел: t = 0, 1, 2, 3, … В каждый дискретный момент времени КА находится в одном и только одном состоянии Si, воспринимает одно значение вектора X и выдает на выходе одно значение вектора Y.

Принято считать, что в момент времени t = 0 автомат находится в начальном состоянии S0, которое можно включить в кортеж отдельным, шестым компонентом: $A = (X, Y, S, f_y, f_s, S_0)$.

Автомат с выделенным начальным состоянием называют инициальным.

Общую схему автомата можно представить в виде «черного ящика», осуществляющего преобразование вектора входных сигналов в вектор выходных.

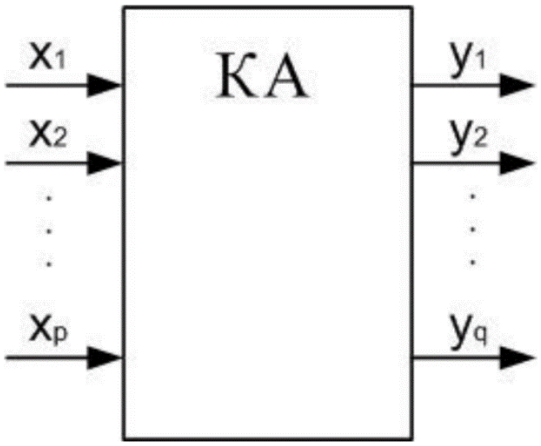

Рисунок 6 – Общая схема конечного автомата

Исходя из этого, можно записать следующее уравнение: Y = f_y(X, t).

Фактор времени в приведенном уравнении учитывается введением вектора состояний S, как своего рода «памяти о прошлом». Действительно, на один и тот же набор входных сигналов (значений компонентов вектора X) автомат будет выдавать разные выходные сигналы (значения компонентов вектора Y) в зависимости от состояния, в котором он находится в данный момент времени. Текущее состояние, в свою очередь, определяется алгоритмом функционирования автомата.

1.3 Клеточные автоматы

1.3.1 Классификация клеточных автоматов

Клеточный автомат – это дискретная модель, которая представляет из себя сетку произвольной размерности. Каждая клетка в каждый момент времени может принимать одно из конечного множества состояний, при этом существует правило, по которому осуществляется переход клеток из одного состояния в другое.

Клеточные автоматы могут быть классифицированы критериями, представленными в таблице 2.

Таблица 2

Критерий	**Возможные варианты**
Размерность решетки	Одномерная, двумерная, трехмерная и т.д.
Количество возможных состояний	Бинарные, троичные и т.д.
Определение окрестности клетки	Окрестность Фон-Неймана, окрестность Мура и др.
Синхронизация	Синхронные и асинхронные клеточные автоматы
Тип поведения	4 класса

Клеточные автоматы можно разделить на 4 класса по типу поведения:

1) все клетки быстро принимают одинаковое состояние, после чего автомат стабилизируется;

2) состояние всех клеток быстра стабилизируется, либо возникают периодические колебания состояний клеток;

3) автомат порождает хаотические, непериодические структуры. Небольшие изменения исходного состояния влекут за собой значительные изменения в будущем;

4) автомат порождает сложные, взаимодействующие между собой структуры, способные выживать длительное время. Однако при этом автомату не удается достичь стабильного состояния.

Самой простой, так называемый простейший клеточный автомат, - это одномерный бинарный клеточный автомат, где состояния клетки в каждый момент времени зависит только от ее собственного состояния и состояний смежных с ней клеток в предыдущей момент времени [7].

Простейших клеточных автоматов существует всего 256, и поведение некоторых из них дублирует другие. Но, несмотря на это, широко известный в узких кругах Стивен Вольфрам посвятил годы жизни их изучению [5].

Вариантов простейших автоматов всего 256. Каждый из вариантов таким автоматов принято называть по порядковому номеру, или же «Правило N». Возьмем для примера наиболее интересное из них, правило 110.

Двоичный код десятичного числа 110 представляет собой последовательность бит 01101110. Данная последовательность бит формирует функцию переходов клетки.

Таблица 3

111	110	101	100	011	010	001	000
0	1	1	0	1	1	1	0

В зависимости от состояний соседа слева, самой клетки, соседа справа (первая строка таблицы) на следующем шаге клетка примет одно из состояний, указанных во второй строке таблицы 3.

Графическая иллюстрация представлена на рисунке 7.

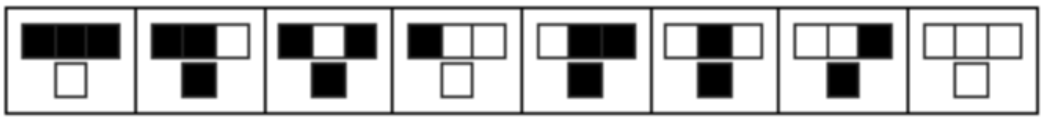

Рисунок 7 – графическое представление правила 110

Клеточные автоматы, в отличии от классических конечным автоматов, являются Тьюринг-полными. Это делает возможным, посредством клеточного автомата реализовывать любую вычислимую функцию, т.е. реализовывать любой возможный алгоритм.

Стоит заметить, что каждая клетка клеточного автомата представляет простой конечный автомат. В простом случае, когда имеется двумерное моле и каждая клетка способна принимать всего два состояния, эта самая клетка реализует автомат, который способен принимать два возможных состояния под воздействием входных сигналов. В данном случае входным сигналами будут являться окрестность клетки, т.е. текущие состояния клеток, входящих в эту окрестность. При этом текущее состояние клетки-автомата будет являться входным сигналом для других клеток.

1.3.2 Эволюционирующий клеточный автомат

1.3.2.1 Базовые концепции эволюционирующего клеточного автомата

Клеточный автомат весьма мощное средство моделирования. Он позволяет строить модели различных динамических систем. Даже простейшие одномерные клеточные автоматы являются Тьюринг-полными и способны генерировать сложные непериодические структуры.

Однако все клетки такого автомата равны между собой и работает каждая из них по заранее определенному правилу. Соединение идеи клеточного автомата и генетического алгоритма позволит избежать статичности правила и позволит его видоизменять при каждой итерации алгоритма [8].

В этом случае правило автомата фактически превращается в генетический код, а сам клеточный автомат становится агентом.

1.3.2.2 Генетический код эволюционирующего клеточного автомата в случае использования одномерного клеточного автомата

В случае простейшего генетического автомата, где правило формируется по состояниям двух смежных клеток и состоянию самой клетки, существует 8 возможных комбинаций, из которых можно породить 256 правил (рис. 8). Как раз переход в одно из двух возможных состояний для каждой из 8 комбинаций будет определять один из соответствующих им 8 генов.

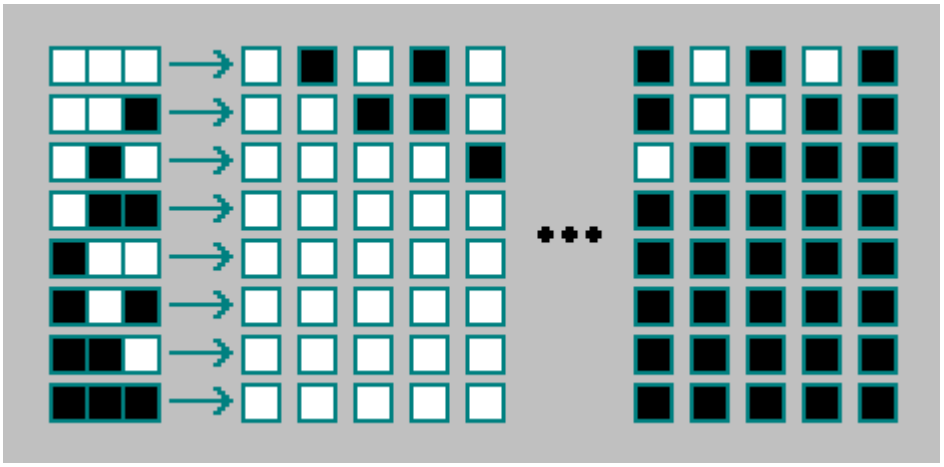

Рисунок 8 – Возможные комбинации состояний одномерного клеточного автомата

Для решения задач моделирования чаще приходится использовать моделирование в двумерном или трехмерном пространстве. Практических задач для одномерного пространства значительное меньше чем, например, задачи на плоскости или связанных с объемным пространством. Поэтому стоит рассмотреть работу генетического алгоритма с использованием двумерного клеточного автомата для большей наглядности и практичности.

У двумерного клеточного автомата для каждой клетки, если использовать окрестность Мура, существует 8 соседних клеток. Существует порядка $2^9 = 512$ возможных комбинаций состояний клетки и её соседей, из чего следует, что поведение данного клеточного автомата определяет комбинация из 512 генов. Комбинация этих генов дает 2^{512} возможных генетических конфигураций клеточного автомата (рис 9), что является огромным множеством всевозможных правил, согласно которым может работать данный клеточный автомат.

Рассмотренные клеточные автоматы являлись автоматами первого порядка, т. е. их последующее состояние зависело только от текущего состояния. Автоматы также могут быть и высших порядков. В таком случае следующее состояние клетки такого автомата будет зависеть не только от текущего состояния клетки и клеток в ее окрестности, от и от её состояния на предыдущих шагах

Такая большая комбинация генов дает возможность генерировать большое множество клеточных автоматов с различными правилами, но, исходя из идеи генетических алгоритмов, необходимо среди большого этого множества клеточных автоматов отбирать те, которые лучше решают поставленную перед ними задачу.

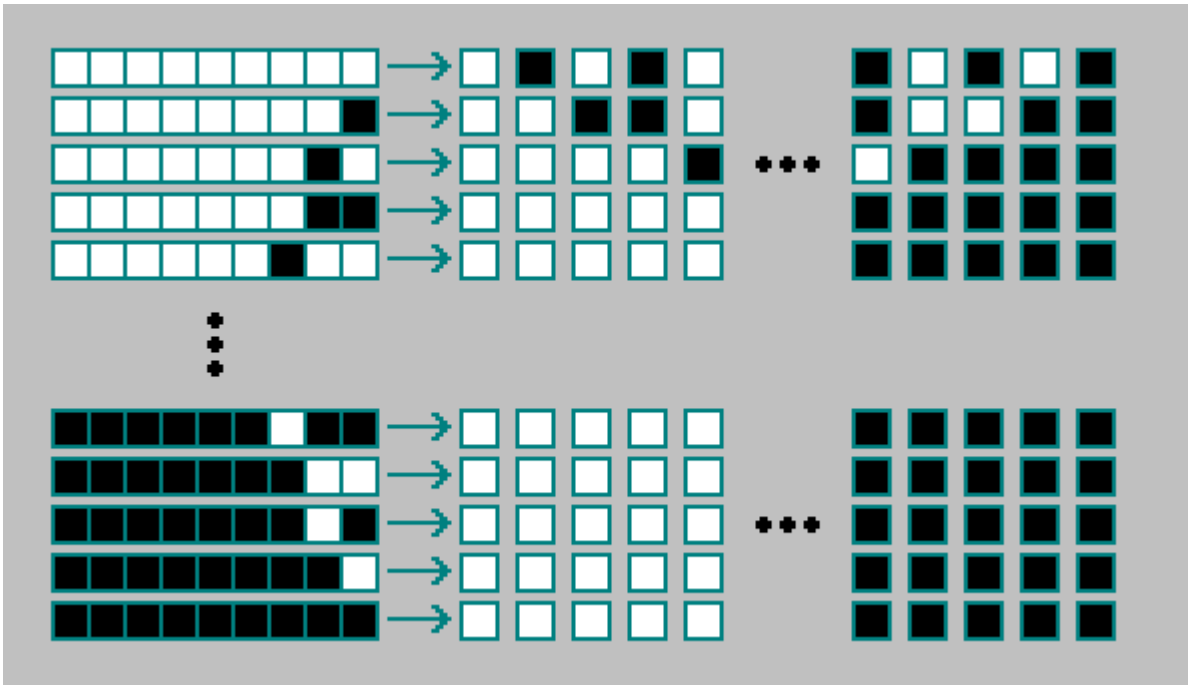

Рисунок 9 – Возможные комбинации состояний двумерного клеточного автомата

1.3.2.3 Отбор и селекция клеточных автоматов

Для оценки, насколько близко эволюционирующий клеточный автомат приблизился к цели в ходе работы генетического алгоритма, необходимо сформировать определенные критерии, оп которым будет происходить оценивание.

Критерии оценивания можно задать любые, в зависимости от поставленных задач. Для двумерного клеточного автомата первого порядка будет достаточно наглядно в качестве примера взять задачу по образованию заданного рисунка или узора.

В этом случае критерием оценивания для клеточных автоматов (агентов) будет выступать количество рисунков и их точность (процент ошибок). Поиск рисунков в этом случае будет производиться относительно каждой клетки в ее окрестности. Стоит отметить, что такая окрестность не должна совпадать с окрестностью, по которой сформировано правило в клеточном автомате. Более того, она может быть произвольного размера и формы (рис. 10).

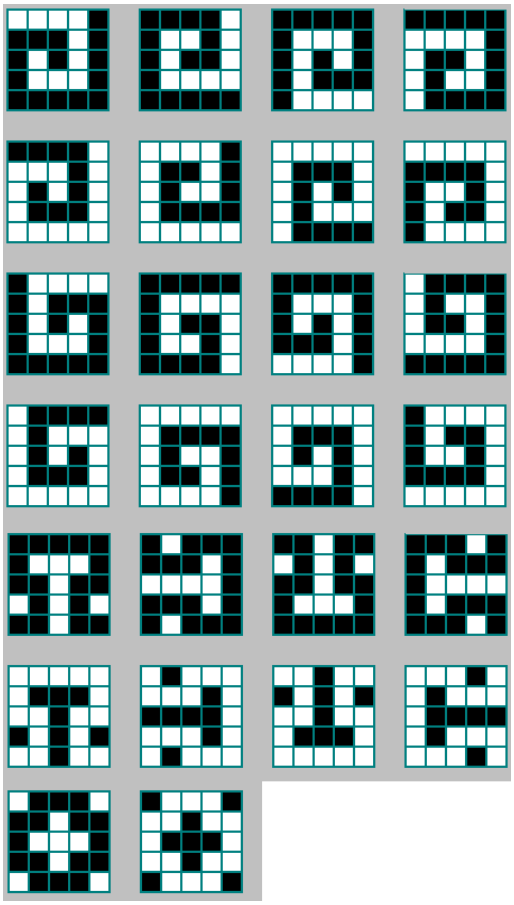

Рисунок 10 – Возможные критерии отбора для клеточного автомата с прямоугольной окрестностью 5x5

Согласно сформулированным критериям должен производится отбор и селекция клеточных автоматов, участвующих в описке решения.

1.3.2.4 Формирования генетического кода клеточного автомата в ходе селекции

Для формирования генетического кода нового клеточного автомата можно использовать генетический код случайного автомата из предыдущего поколения со случайной мутацией одного из гена. Вероятность мутации определяется для каждой задачи индивидуально, при этом вычислить ее аналитические, как правило, не представляется возможным.

Можно использовать для формирования генетического кода нового автомата код сразу нескольких автоматов из предыдущего поколения (рис. 11). При этом автоматы из скрещивания выбираются случайным образом. В

случайном гене полученного генетического кода с некоторой вероятностью также происходит мутация.

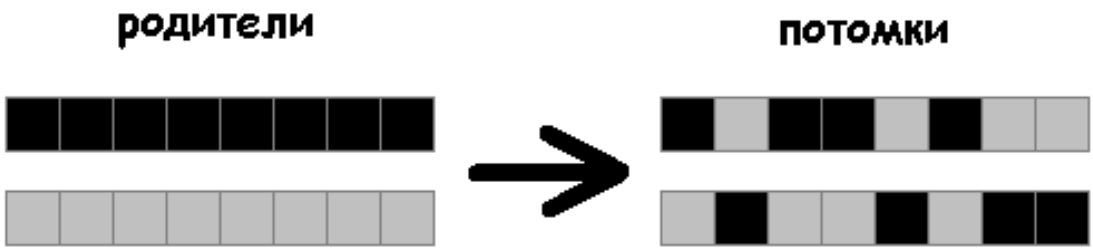

Рисунок 11 – Формирование генетического кода нового клеточного автомата из генетических кодов двух клеточных автоматов

В ходе отбора, селекции, мутаций и оценка полученных результатов происходит эволюционный процесс, в ходе которого происходит постепенное приближение к оптимальному решению поставленной перед генетическим алгоритмом задачи.

2 Конструкторская часть

2.1 Высоконагруженные системы

2.1.1 Понятие высоконагруженной системы

Понятие высоконагруженной системы является расплывчатым, нет конкретных цифр, благодаря которым ту или иную программную системы можно отнести к этому классу.

Высоконагруженная система – это совокупность программных компонентов, обеспечивающая бесперебойный доступ клиентам, предоставляя им набор сервисом, которая постоянно масштабируется для поддержания работоспособности.

Из определения следует, что основной признак высоконагруженной системы – это постоянная необходимость в ее улучшения (масштабировании и добавлении ресурсов) ввиду постоянного роста нагрузки.

2.1.2 Качества высоконагруженной системы

Нельзя точно сказать, начиная с каких количественных показателей система может считаться высоконагруженной, но можно выявить некоторые качества, которые могут помочь отнести систему к классу высоконагруженных

- обслуживает большое количество клиентов;
- системы является распределенной;
- постоянно растущая по количеству клиентов система;
- развивающаяся система;
- система, обладающая большим количеством ресурсов и растущей потребностью в них;

2.1.3 Особенности высоконагруженных систем

Разработка высоконагруженной системы не так сильно отличается от просто грамотной разработки веб-приложения (большинство представителей современных высоконагруженных систем – это веб-приложения, ввиду того, что Интернет обладает громадной аудиторией, где каждый пользователь может быть клиентом высоконагруженного сервиса).

Ключевым элементом в такой разработке является правильное проектирование архитектуры приложения, т. к. повышения нагрузки на разрабатываемую систему не должно вызывать необходимость в изменение логики работы компонентов, добавлению новых, удалению старых.

Обеспечение горизонтальной масштабируемости компонентов системы на всех уровнях и предварительное выявление и ликвидация узких мест – наиболее эффективный шаг для обеспечения эффективного роста и легкой модернизации разрабатываемой системы в отдаленном будущем, когда рост нагрузки приведет к дефициту ресурсов при текущей конфигурации оборудования.

На данный момент для реализации больших систем с большим количеством различных компонентов (сейчас почти любая высоконагруженная система обладает большими размерами и большим

разнообразием используемых компонентов) используются технологии виртуализации.

Независимо от того, сколько отдельных компонентов разрабатывается и развертывается, одна из самых больших проблем, с которой всегда приходится сталкиваться разработчикам и системным администраторам, – это различия в окружениях, в которых они выполняют свои приложения. Мало того, что существует огромная разница между окружением разработки и рабочим окружением, различия даже существуют между отдельными машинами в рабочем окружении. Еще одним неизбежным фактом является то, что окружение одной рабочей машины будет меняться с течением времени.

Виртуализация позволяет абстрагироваться от конфигурации реальных физических машин и запускать компоненты системы в отдельных виртуальных машинах, тем самым делая их выполнение независимым от ОС, инсталированной на физическом сервере.

2.1.4 Использование микросервисов в высоконагруженных системах

Монолитные приложения состоят из компонентов, которые тесно связаны друг с другом и должны разрабатываться, развертываться и управляться как одна сущность, поскольку все они выполняются как один процесс ОС. Изменения в одной части приложения требуют новой выкладки всего приложения, и со временем отсутствие жестких границ между частями приводит к увеличению сложности и последующему ухудшению качества всей системы из-за неограниченного роста взаимосвязей между этими частями.

Для запуска монолитного приложения обычно требуется небольшое количество мощных серверов, которые могут предоставить достаточно ресурсов для запуска приложения.

Для того чтобы справиться с растущей нагрузкой на систему, нужно либо масштабировать серверы вертикально (так называемое масштабирование вверх), добавляя больше процессоров, оперативной памяти и других

серверных компонентов, либо масштабировать всю систему по горизонтали, настраивая дополнительные серверы и запуская несколько копий (или реплик) приложения (масштабирование вширь). Хотя масштабирование вверх обычно не требует каких-либо изменений в приложении, оно относительно быстро становится дорогостоящим и на практике всегда имеет верхний предел.

Масштабирование вширь, с другой стороны, является относительно дешевым аппаратно, но может потребовать больших изменений в программном коде приложения и не всегда возможно – некоторые части приложения с большим трудом поддаются горизонтальному масштабированию или почти невозможны для него (например, реляционные базы данных). Если какая-либо часть монолитного приложения не масштабируется, то все приложение становится немасштабируемым, если только каким-то образом этот монолит не разделить.

Эти и другие проблемы заставили начать разбиение сложных монолитных приложений на небольшие независимые развертывания компонентов, называемых микросервисами. Каждый микросервис выполняется как независимый процесс (рис. 12) и взаимодействует с другими микросервисами через простые, четко определенные интерфейсы (API).

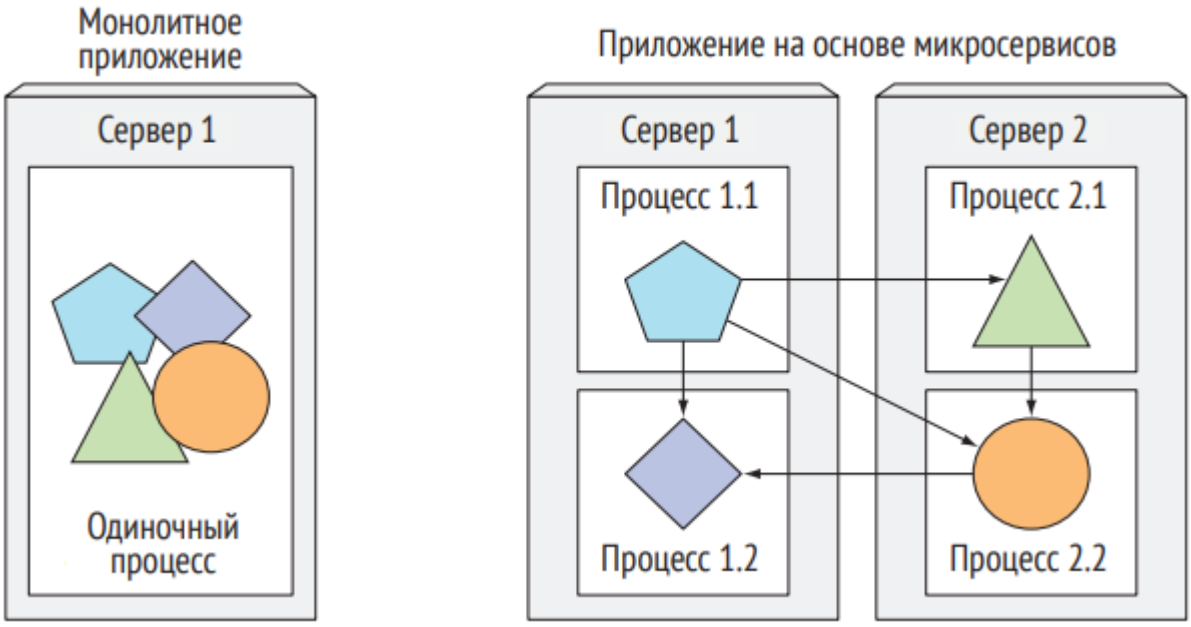

Рисунок 12 – Компоненты программной системы внутри монолитного приложения и независимых микросервисов

Микросервисы взаимодействуют через синхронные протоколы, такие как HTTP, используя которые, они обычно предоставляют RESTful, или через асинхронные протоколы, такие как AMQP (Advanced Message Queueing Protocol, расширенный протокол организации очереди сообщений). Эти протоколы просты, хорошо понятны большинству разработчиков и не привязаны к какому-либо конкретному языку программирования. Каждый микросервис может быть написан на языке, который наиболее целесообразен для реализации конкретных микросервисов.

Поскольку каждый микросервис представляет собой автономный процесс с относительно статическим внешним API, существует возможность разрабатывать и развертывать каждый микросервис отдельно. Изменение одной из них не требует изменений или повторного развертывания какого-либо другого сервиса, при условии, что API не изменяется или изменяется только обратно совместимым образом.

2.2 Технологии контейнеризации

2.2.1 Контейнер как метод виртуализации

Когда приложение состоит лишь из небольшого количества крупных компонентов, вполне допустимо предоставить каждому компоненту выделенную виртуальную машину и изолировать их среды, предоставив каждому из них собственный экземпляр операционной системы.

Однако, когда эти компоненты начинают уменьшаться в объеме и их количество начинает расти, становится невозможным предоставлять каждому из них свою собственную виртуальную машину.

Поскольку каждая виртуальная машина обычно должна настраиваться и управляться индивидуально, увеличение количества виртуальных машин также приводит к трате человеческих ресурсов, поскольку они значительно увеличивают рабочую нагрузку на системных администраторов.

Вместо того чтобы использовать виртуальные машины для изоляции сред каждого микросервиса (или программных процессов в целом), можно использовать контейнерные технологии Linux. Данные технологии позволяют запускать несколько сервисов на одной хост-машине, не только обеспечивая доступ к разным средам, но и изолируя их друг от друга, подобно виртуальным машинам, но с гораздо меньшими затратами.

Контейнеризация - метод виртуализации, при котором ядро операционной системы поддерживает несколько изолированных экземпляров пространства пользователя вместо одного.

Эти экземпляры (обычно называемые контейнерами) с точки зрения пользователя полностью идентичны отдельному экземпляру операционной системы.

Ядро обеспечивает полную изолированность контейнеров, поэтому программы из разных контейнеров не могут воздействовать друг на друга.

При контейнеризации отсутствуют дополнительные ресурсные накладные расходы на эмуляцию виртуального оборудования и запуск

полноценного экземпляра операционной системы, характерные при аппаратной виртуализации.

2.2.2 Сравнение виртуальных машин и контейнеров

По сравнению с виртуальными машинами, контейнеры гораздо облегченнее, что позволяет запускать большее количество программных компонентов на одном и том же оборудовании, главным образом потому, что каждая виртуальная машина должна запускать свой собственный набор системных процессов, который требует еще вычислительных ресурсов в дополнение к тем, которые потребляются собственным процессом компонента.

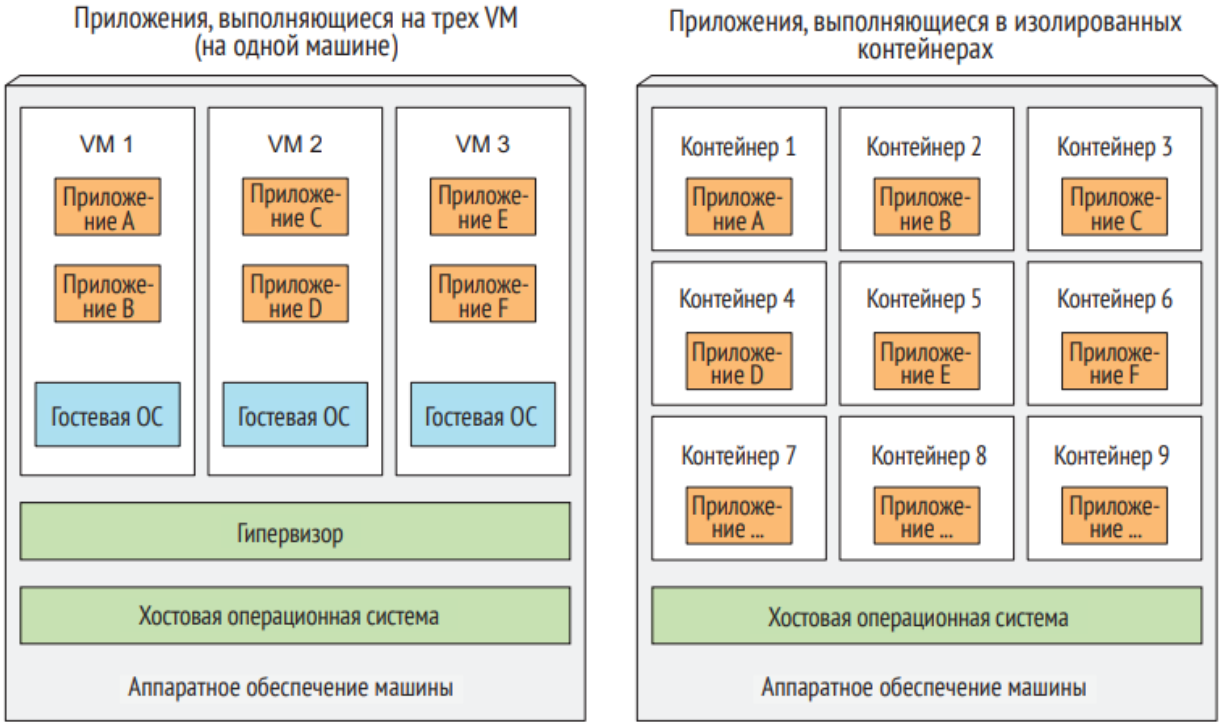

Рисунок 13 – Сравнение виртуальных машин для изоляции групп приложения с изоляцией отдельных приложения с помощью контейнеров

Контейнеры выполняют системные вызовы на одном и том же ядре, работающем в хостовой ОС. Это единственное ядро, выполняющее инструкции на процессоре хоста. ЦП не нужно делать какой-либо виртуализации, как он делает с виртуальными машинами.

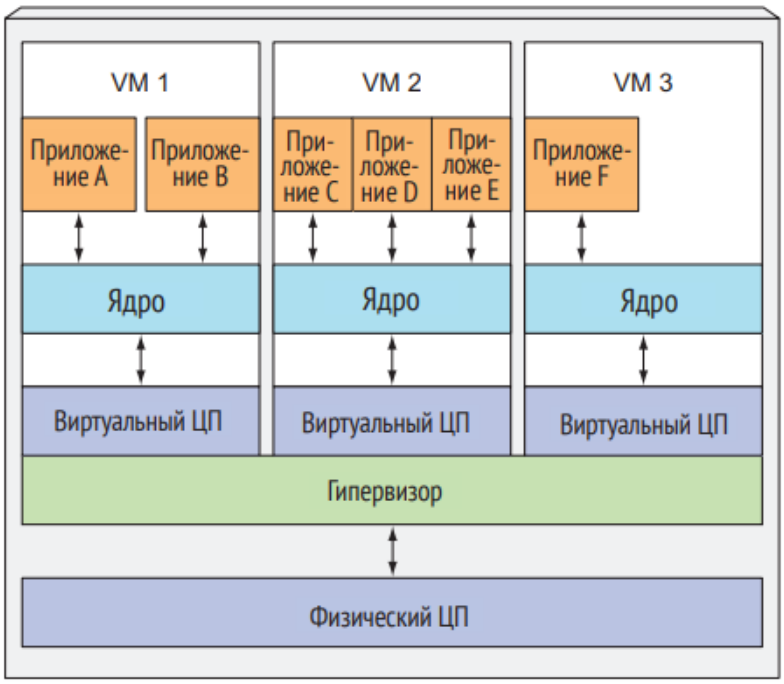

Рисунок 14 – Приложения, выполняющиеся на множестве виртуальных машин

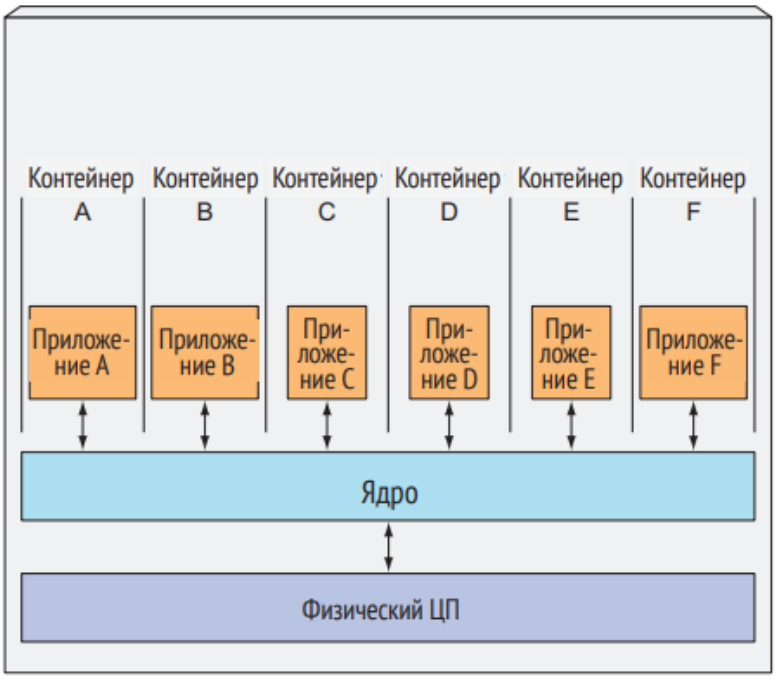

Рисунок 15 – Приложения, выполняющиеся в изолированных контейнерах

2.2.3 Сравнение различных технологий контейнеризации

Существуют реализации, ориентированные на создание практически полноценных экземпляров операционных систем (Solaris Containers, контейнеры Virtuozzo, OpenVZ), так и варианты, фокусирующиеся на

изоляции отдельных сервисов с минимальным операционным окружением (jail, Docker).

Таблица 4 – Сводная таблица технологий контейнеризации

Механизм	Операционная система	Лицензия	Дата выпуска	Особенности					
				Изоляция файловой системы	Квоты на пространство хранения	Лимиты на ввод-вывод	Лимиты на память	Квоты ЦПУ	Изоляция сети
chroot	встроено в большинство Unix-подобных операционных систем	в зависимости от лицензии на операционную систему	1982	Частично	Нет	Нет	Нет	Нет	Нет
Docker	Linux, FreeBSD, Windows, macOS	Apache 2.0	2013	Да	Да	Да	Да	Да	Да
Solaris Containers	Solaris, OpenSolaris	CDDL	01/2005	Да	Да	Нет	Да	Да	Да[1]
FreeVPS	Linux	GNU GPL	-	Да	Да	Нет	Да	Да	Да
iCore Virtual Accounts	Windows XP	Проприетарное	06/2008	Да	Да	Нет	Нет	Нет	Да
Linux-VServer[en]	Linux	GNU GPL v.2	-	Да	Да	Да	Да	Да	Да[3]
LXC	Linux	GNU GPL v.2	2008	Да	Нет	Да	Да	Да	Да
OpenVZ	Linux	GNU GPL v.2	-	Да	Да	Да[4]	Да	Да	Да[5]
Virtuozzo Containers	Linux, Microsoft Windows	Проприетарное	-	Да	Да	Да[6]	Да	Да	Да[5]
FreeBSD Jail	FreeBSD	BSD	03/2000	Да	Да	Нет	Да	Частично	Да
sysjail[en]	OpenBSD, NetBSD	BSD	-	Да	Нет	Нет	Нет	Нет	Да
WPAR[en]	AIX	Проприетарное	10/2007	Да	Да	Да	Да	Да	Да[7]

По таблице 4 можно увидеть, что существует большое количество реализаций технологии контейнеризации с различными особенностями, однако самая популярная на данный момент – это Docker.

2.2.4 Docker

Хотя контейнерные технологии существуют уже давно, они стали более широко известны с появлением контейнерной платформы Docker.

Docker была первой контейнерной системой, которая сделала контейнеры легко переносимыми на разные машины. Это упростило процесс упаковки не только приложения, но и всех его библиотек и других

зависимостей, даже всей файловой системы ОС, в простой, переносимый пакет, который может использоваться для подготовки приложения к работе на любой другой машине, на которой работает Docker.

При выполнении приложения, упакованного с помощью Docker, оно видит точное содержимое файловой системы, поставляемое вместе с ним. Оно видит одни и те же файлы, независимо от того, работает ли оно на машине, предназначенной для разработки, или же на машине из рабочего окружения, даже если на рабочем сервере запущена совершенно другая ОС Linux.

Процесс создания образа и запуска контейнера проиллюстрирован на рисунке ниже.

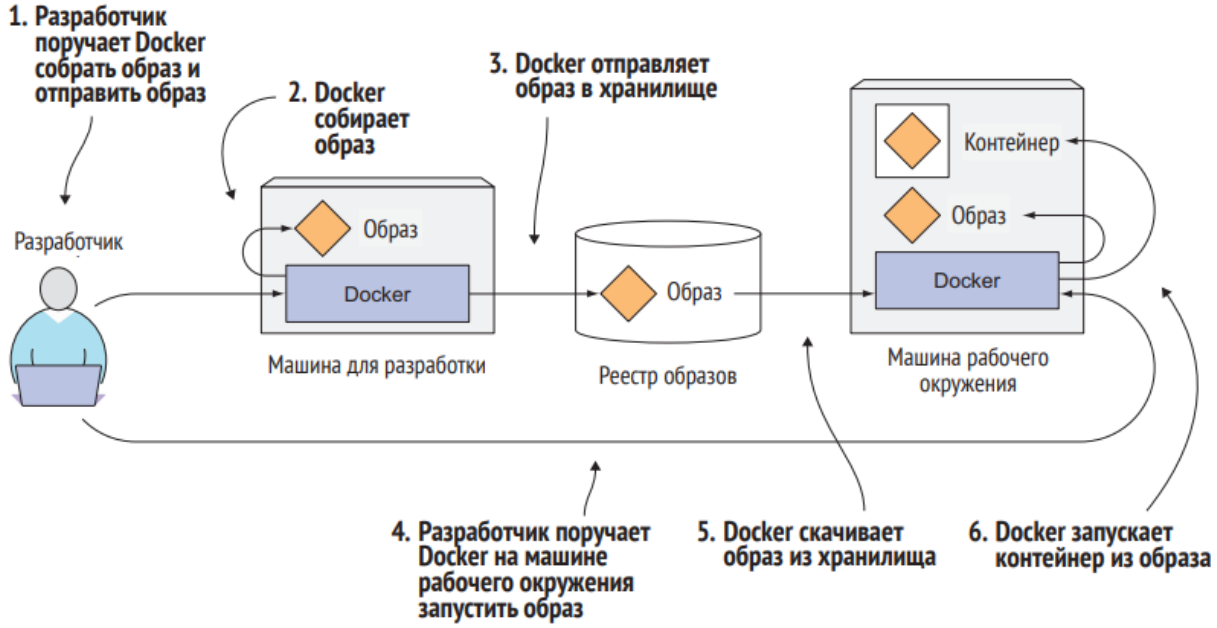

Рисунок 16 – Процесс создания докер-образа и запуска контейнера на рабочем узле программной системы

2.3 Система оркестровки контейнеров Kubernetes

По мере роста программной системы ею становится все сложнее управлять из-за количества разнородных разворачиваемых компонентов. Для решения данной проблемы была разработана такая система, как Kubernetes.

Kubernetes – это программная система, которая позволяет легко развертывать контейнеризированные приложения и управлять ими. Она использует возможности контейнеров Linux для запуска разнородных

приложений без необходимости знать какие-либо внутренние детали этих приложений и без необходимости вручную развертывать эти приложения на каждом хосте.

Поскольку данные приложения работают в контейнерах, они не влияют на другие приложения, работающие на том же сервере, что имеет решающее значение при запуске приложений для совершенно разных организаций на одном и том же оборудовании.

Это имеет первостепенное значение для облачных провайдеров, поскольку они стремятся к максимально возможной задействованности своего оборудования, сохраняя при этом полную изоляцию размещенных приложений.

Kubernetes позволяет выполнять программные приложения на тысячах компьютерных узлов, как если бы все эти узлы были одним огромным компьютером. Она абстрагируется от базовой инфраструктуры и тем самым упрощает разработку, развертывание и управление как для разработчиков, так и для системных администраторов.

Процедура развертывания приложений через Kubernetes всегда одинаковая, независимо от того, содержит ли кластер всего несколько узлов или тысячи. Размер кластера не имеет никакого значения. Дополнительные узлы кластера просто представляют собой дополнительный объем ресурсов, доступных для развернутых приложений.

Простейший вид системы Kubernetes представлен на рисунке ниже. Система состоит из ведущего узла (мастера) и любого количества рабочих узлов. Когда разработчик отправляет список приложений ведущему узлу, Kubernetes развертывает их в кластере рабочих узлов. То, на какой узел приземляется компонент, не имеет (и не должно иметь) значения ни для разработчика, ни для системного администратора.

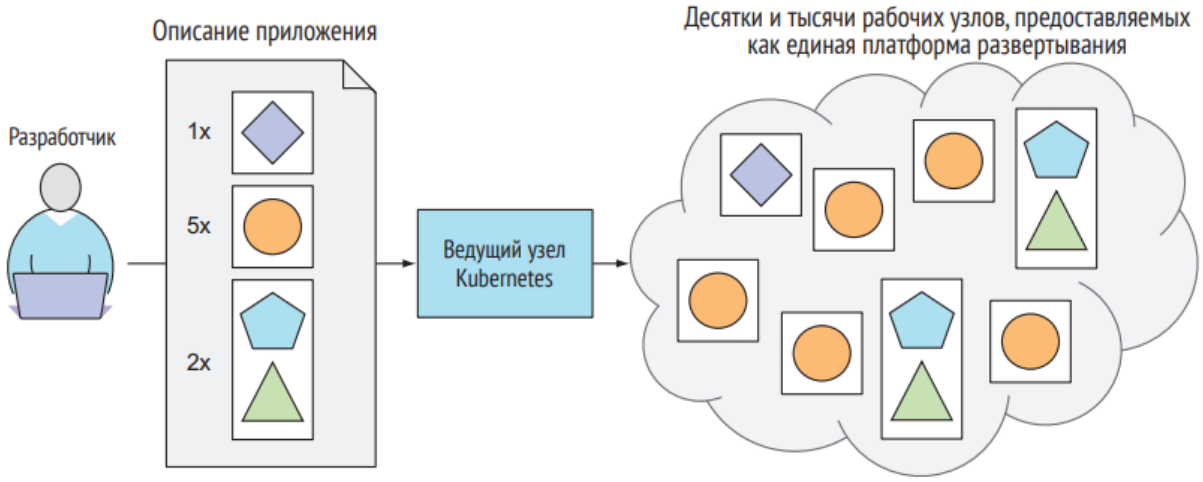

Рисунок 17 – Общий вид системы Kubernetes

Kubernetes можно рассматривать как операционную систему для кластера. Она избавляет разработчиков приложений от необходимости внедрять в свои приложения определенные службы, связанные с инфраструктурой; вместо этого в вопросе предоставления этих служб они опираются на Kubernetes. Это включает в себя такие аспекты, как обнаружение службы, масштабирование, балансировка нагрузки, самовосстановление и даже выбор лидера. Поэтому разработчики приложений могут сосредоточиться на реализации реального функционала приложений и не тратить время на то, чтобы разбираться в том, как интегрировать их с инфраструктурой.

2.3.1 Архитектура кластера Kubernetes

На аппаратном уровне кластер Kubernetes состоит из множества узлов, которые можно разделить на два типа:

– ведущий узел (мастер), на котором размещена плоскость управления (Control Plane) Kubernetes, контролирующая и управляющая всей системой Kubernetes;

– рабочие узлы, на которых выполняются развертываемые приложения.

На рисунке ниже показаны компоненты, работающие на этих двух типах узлов Kubernetes.

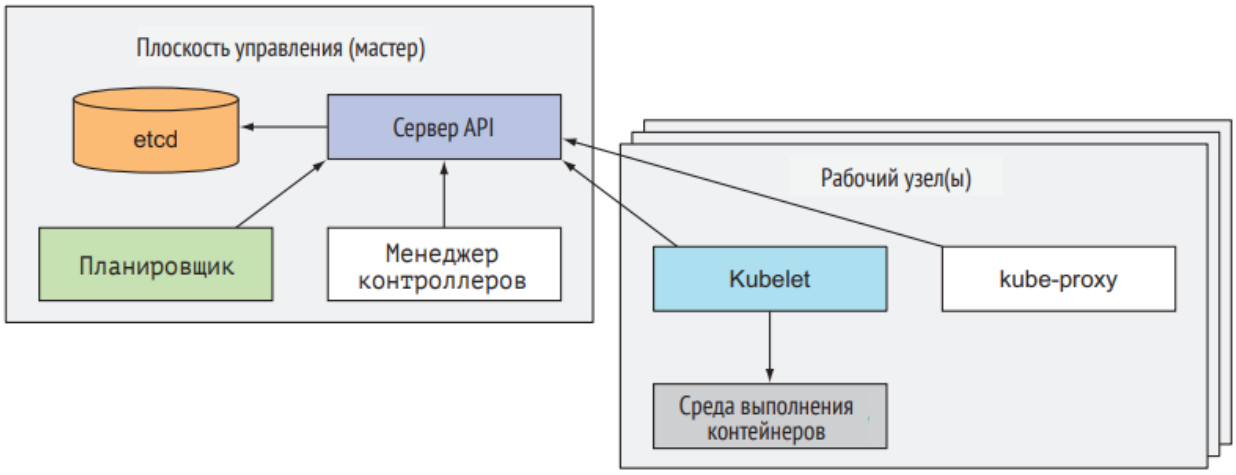

Рисунок 18 – Компоненты кластера Kubernetes

Плоскость управления – это то, что управляет кластером и заставляет его функционировать. Она состоит из нескольких компонентов, которые могут работать на одном ведущем узле либо быть распределены по нескольким узлам и реплицированы для обеспечения высокой доступности.

Рабочие узлы – это машины, на которых выполняются контейнеризированные приложения.

Для запуска приложения в Kubernetes необходимо упаковать его в один или несколько образов контейнеров, отправить эти образы в хранилище образов, а затем опубликовать описание приложения на сервере API Kubernetes.

Описание содержит такие сведения, как образ или образы контейнеров, содержащие компоненты приложения, как эти компоненты связаны друг с другом и какие из них должны выполняться совместно, а какие нет.

2.3.2 Процесс развертывания компонентов программной системы в Kubernetes

Когда сервер API обрабатывает описание приложения, планировщик назначает указанные группы контейнеров доступным рабочим узлам, исходя из вычислительных ресурсов, требуемых каждой группой, и нераспределенных ресурсов на каждом узле в данный момент. Агент Kubelet

на этих узлах затем поручает среде выполнения контейнеров (например, Docker) извлечь из хранилища требуемые образы контейнеров и запустить контейнеры.

После отправки дескриптора в Kubernetes он запланирует использование указанного количества реплик каждого модуля на доступных рабочих узлах. Затем агенты Kubelet на узлах поручают Docker извлечь образы контейнеров из реестра образов и запустить контейнеры

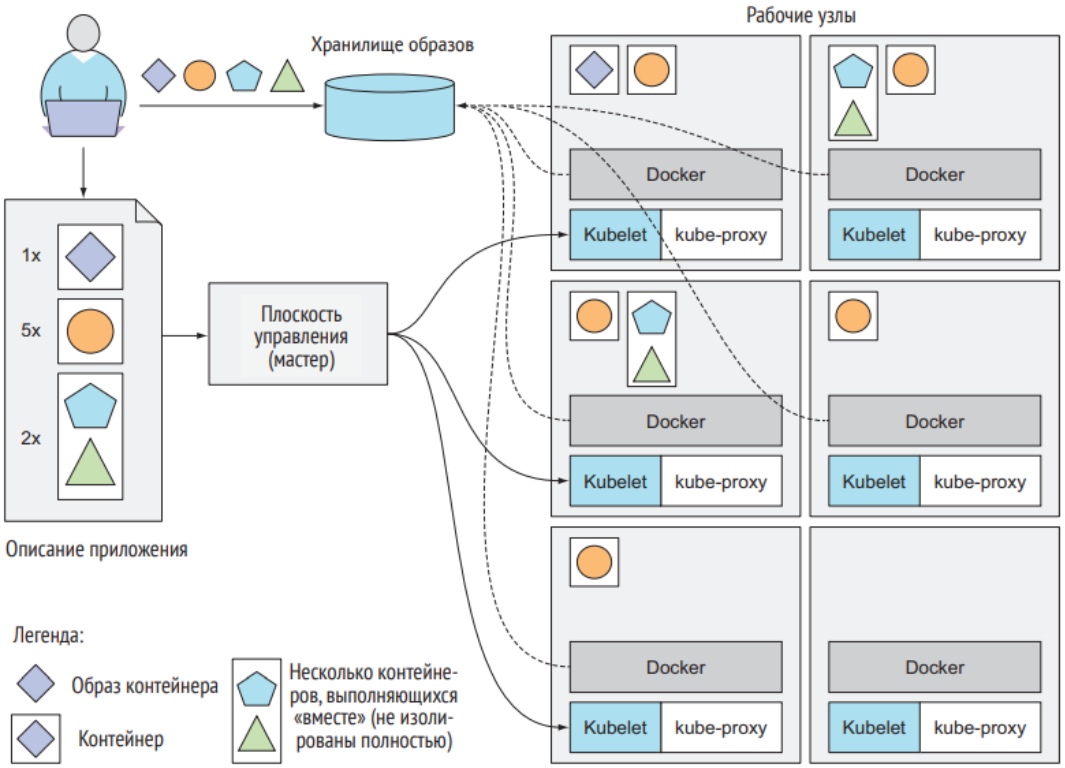

Рисунок 19 – Процесс развертывания приложения в Kubernetes

2.3.3 Преимущества использования Kubernetes

1) Упрощенное развертывание приложений. Kubernetes обеспечивает доступ ко всем своим рабочим узлам как к единой платформе развертывания. По сути, все узлы теперь представляют собой единую группу вычислительных ресурсов, которые ждут, когда приложения будут их потреблять.

2) Повышение эффективности задействования оборудования. Настройка кластера Kubernetes позволяет отделить программную систему от инфраструктуры. Компоненты программной системы могут свободно

запускаться и перемещаться между доступными узлами в кластере, что позволяет эффективно использовать аппаратные ресурсы.

3) Проверка здоровья и самолечение. Kubernetes отслеживает компоненты приложения и узлы, на которых они выполняются, и автоматически переносит их на другие узлы в случае аварийного сбоя узла. Также в случае аварийного завершения работы одного из рабочих физических узлов Kubernetes, работавшие до этого на нем модули будут автоматически переназначены на другие узлы, тем самым обеспечивая бесперебойность работы всей системы.

4) Автоматическое масштабирование. Если Kubernetes работает в облачной инфраструктуре, где добавлять дополнительные узлы так же просто, как запрашивать их через API поставщика облака, Kubernetes даже может автоматически масштабировать размер всего кластера вверх или вниз в зависимости от потребностей развернутых приложений.

5) Упрощение разработки приложений. Системы позволяет быстро разворачивать новые версии программных компонентов. Kubernetes позволяет также настроить процесс постепенного обновления программных компонентов, сделав его наиболее безопасным и плавным. В случае аварийной ситуации Kuberenetes может прервать процесс выкладки и вернуться к предыдущей стабильной конфигурации кластера.

2.4 Проектирования программной системы с использованием Kubernetes и Docker

Для проектирования программной системы с использованием технологии контейнеризации и системы оркестровки контейнерами Kubernetes необходимо выделить компоненты разрабатываемой системы, а также взаимосвязи между ними, после чего интегрировать их в кластер Kubernetes, используя предоставляемые им инструменты и сервисы.

2.4.1 Выбор базовых программных компонентов проектируемой системы

Поскольку разрабатываемая система представляет из себя веб-приложения, доступ к которому осуществляется посредством использования сети Интернет, то для отображения графического интерфейса будет использоваться браузер.

Минимальная реализация подобной программной системы должна включать в себя минимум три компонента: сервис для отдачи веб-приложения браузеру, сервис для обработки запросов от веб-приложения и хранилище данных.

Для реализации данных программных компонентов разрабатываемой системы были выбраны следующие языки программирования и технологии:

– для отдачи веб-приложения на браузер пользователя будет использоваться frontend-сервер Node.js;

– для обработки действия пользователя, отдачи данных и обработки запросов будет использоваться веб-сервис, реализованный на языке программирования GO;

– для хранения данных пользователей и служебных данных разрабатываемой системы, планируется использовать реляционную базу данных Postgres 10.1.

2.4.2 Простейший вариант программной архитектуры проектируемой системы

Наиболее простой вариант программной архитектуры с использованием frontend-backend подхода представлен на рисунке 20.

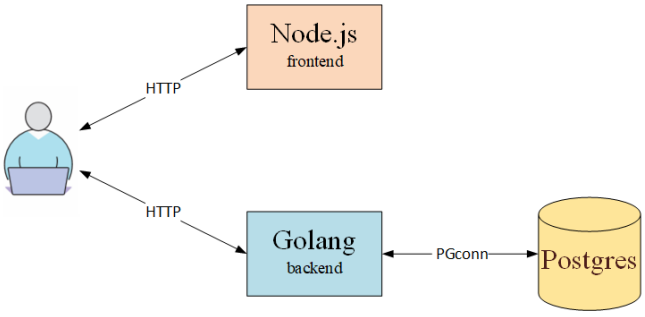

Рисунок 20 – Простой вариант программной архитектуры веб-приложения

Данный подход к проектированию программной системы имеет как плюсы, так и недостатки.

Плюсы:

– при такой архитектуре количество разнородных программных компонентов не велико.

– нет накладных расходов на дополнительные сервисы, обеспечивающие инфраструктуру приложения (балансировки нагрузки, прокси-серверы, регистраторы сервисов и другие)

Минусы:

– отсутствует единая точка входа для пользователя в программную систему

– масштабируемость для каждого из компонентов системы требует изменение конфигурации других элементов системы

– компоненты системы представляют из себя большие монолитные программы, некоторый функционал из которых может быть вынесен в отдельные сервисы

– отсутствуют компоненты программной системы, позволяющие равномерно распределить нагрузки между однородными компонентами системы

– выход из строя одного из компонентов приводит к отказу всей системы

С учётом всех плюсов и недостатков описанная выше архитектура может быть использована в небольших проектах, но для высоконагруженных систем она не годится в связи с громоздкостью отдельных компонентов и потенциальными проблемами при масштабировании.

2.4.3 Масштабируемый вариант архитектуры проектируемой системы

На рисунке 21 представлен вариант архитектуры программной системы с учетом необходимости в масштабировании отдельных компонентов и требованием к отказоустойчивости.

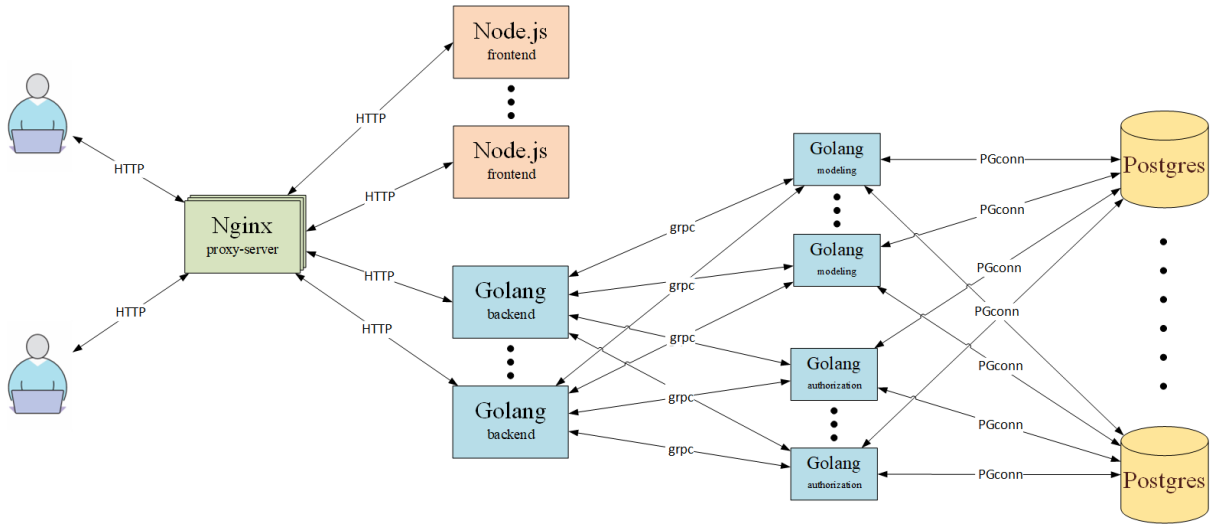

Рисунок 21 – Программная система с масштабируемой архитектурой

Заметна существенна разница между первым вариантом архитектуры и новым. Можно также выделить в данной архитектуре список достоинств и недостатков

Достоинства:

– каждый компонент проектируемой системы может быть горизонтально отмасштабирован

– нет единой точки отказа, выход из строя одного из компонентов системы не ведет к потере ее работоспособности

– для добавления дополнительного frontend или backend сервера достаточно только изменить конфигурацию прокси-сервера nginx.

Недостатки:

– в данной системе добавление компонентов при масштабировании все еще требует изменению конфигурации других компонентов

– отсутствует инфраструктура для автоматического масштабирования в зависимости от текущей нагрузки

– данная архитектура требует от администратора системы дополнительных временных затрат времени из-за в целом более высокой сложности по сравнению с предыдущим вариантом

Всё выявленные недостатки в данной архитектуре можно исправить при помощи использования системы оркестровки контейнеров Kubernetes.

2.4.4 Реализация проектируемой системы в кластере Kubernetes

Для переноса программных компонентов системы необходимо прежде всего использовать технологию контейнеризации Docker. За счет этого каждый программный компонент будет работать изолированно от других запущенных на физической машине контейнеров и независимо от самой ОС и конфигурации этой самой машины.

Для построения инфраструктуры для проектируемой программной системы в кластере Kubernetes потребуются перечень инструментов Kubernetes, представленный в таблице 5.

Таблица 5 – Используемые средства Kubernetes

Название средства Kubernetes	**Тип**	**Выполняемая функция**
Deployment	Контроллер развертывания	Развертывание экземпляров одного из компонентов программной системы
Service	Сервис	Обеспечение доступа между компонентами программной системы и из вне
Vault	Хранилище	Создание временных и постоянных хранилищ данных
Secret	Чувствительная информация	Хранения паролей и секретной информации для обеспечения контроля доступа

Итоговая архитектура представлена на рисунке ниже.

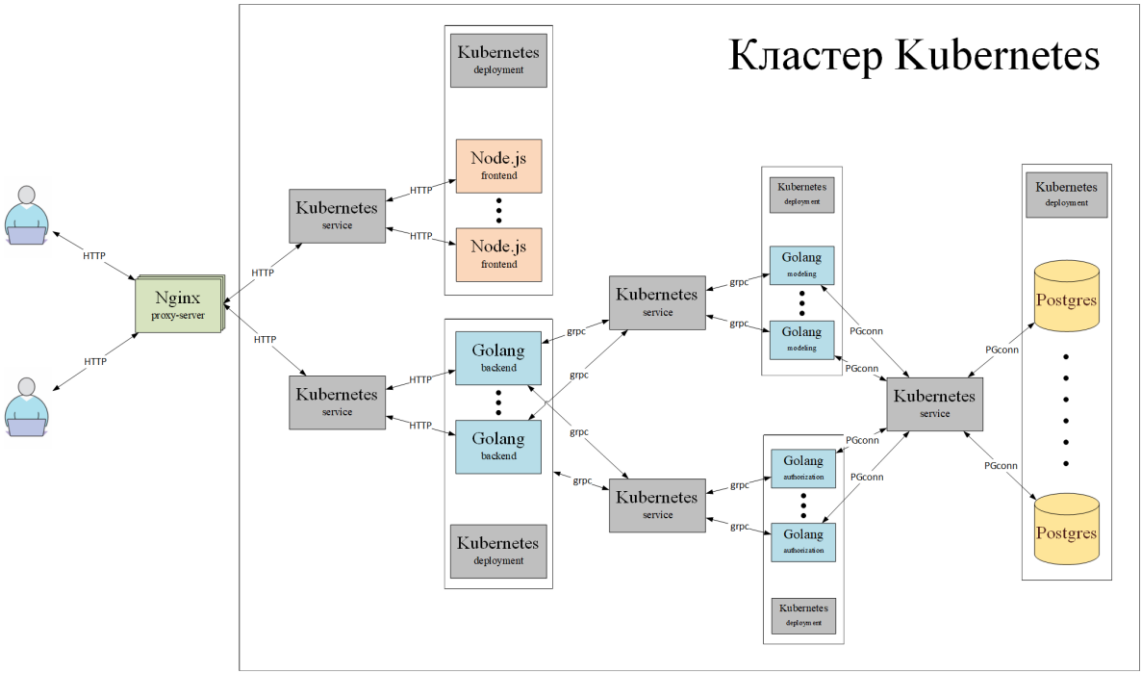

Рисунок 22 – Архитектура программной системы в Kubernetes

Такие средства Kubernetes, как vault и secret создаются и настраиваются посредством конфигурации соответствующих опций в ресурсах deployment. Кластер Kubernetes самостоятельно определит машины для размещения хранилища данных (vault) и паролей (secret).

Ресурсы deployment позволяют в автоматическом режиме разворачивать экземпляры программных компонентов и увеличивать/уменьшать их количество в зависимости от текущей нагрузки (если соответствующих критерии были определены администратором).

В случае использования такого ресурса kubernetes, как service, больше нет необходимости менять конфигурацию компонентов программной системы в случае изменения количества работающих элементов. Сервисы самостоятельно отслеживают запуск новых экземпляров программных компонентов и обеспечивают равномерное распределение нагрузки.

Kubernetes использует декларативный подход. Это значит, что администратору достаточно только описать желаемое состояние кластера, а дальше сама система уже сделает все необходимое, чтобы прийти к заданному состоянию. Это значительно облегчает расходы на администрирование и обслуживание подобных систем

3 Технологическая часть