Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра Автоматизации систем вычислительных комплексов

Чупахин Андрей Андреевич

Планирование вычислений в центрах обработки данных с построением плана миграции

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Научный руководитель:
в.н.с., к.т.н.
В.А. Костенко

Москва, 2016

Аннотация

В рамках выпускной квалификационной работы разработан и реализован алгоритм распределения ресурсов в центрах обработки данных с построением плана миграции.

Данный алгоритм является модификацией уже существующего алгоритма распределения ресурсов в центрах обработки данных с единым планировщиком для различных типов ресурсов. Модифицированный алгоритм получает не только отображение запросов на физические ресурсы ЦОД, но и строит план миграции выполняемых виртуальных ресурсов, если они в новых отображениях должны выполняться на других физических ресурсах. Миграция проводится с учетом гарантии выполнения соглашений о качестве сервиса размещенных в ЦОД запросов. Построенный план удовлетворяет заданному ограничению на время выполнения.

Содержание

Введение

В настоящее время центры обработки данных приобретают все большую популярность. Центры обработки данных(ЦОД) состоят из совокупности вычислительных ресурсов, систем хранения данных, физических каналов связи и коммутационного оборудования работающего под управлением облачной платформы. Всеми этими ресурсами нужно эффективно управлять. Для этой цели разрабатываются специальные алгоритмы распределения ресурсов. Одним из важнейшим свойств алгоритмов является возможность построения такого распределения, при котором утилизация ресурсов в ЦОД достаточно высокая, также немаловажным является время работы алгоритма, поэтому актуально построение алгоритма, в котором можно задавать баланс между качеством получаемого решения и вычислительной сложностью алгоритма.

Существует три различных режима работы ЦОД : IaaS (Infrastructure as a Service), PaaS (Platform as a Service), SaaS (Software as a Service). В моей работе рассматривается ЦОД работающий в режиме Infrastructure-as-a-Service (IaaS), т.е. в качестве предлагаемых ресурсов предоставляются вычислительные ресурсы, системы хранения данных (СХД) и сетевые ресурсы.

Основной задачей ЦОД является предоставление ресурсов пользователям. Пользователь может сделать запрос в ЦОД на получение некоторого набора ресурсов.

Ресурсным запросом в ЦОД является запрос на предоставление ресурсов, а именно: виртуальных машин (VM), виртуальных систем хранения данных (storage-элементов) и виртуальных каналов связи между ними. Фактически пользователь должен указать топологию желаемой сети: узлы - виртуальные машины или storage-элементы и связи между узлами — виртуальные каналы.

При задании ресурсного запроса клиент ЦОД может указать конкретные характеристики запрашиваемых ресурсов, например, есть возможность указать объем оперативной памяти, количество ядер у виртуальной машины, объем памяти у storage-элементов, пропускную способность виртуального канала связи.

После получения запроса планировщик ЦОД, в котором реализуется тот или иной алгоритм распределения ресурсов, пытается разместить его элементы на физические ресурсы.

Если запрос будет размещен, то ЦОД гарантирует, что в течение всего времени “жизни” запроса элементы запроса будут иметь запрошенные характеристики. Таким образом, в

данной выпускной работе рассматривается случай, когда для ресурсного запроса имеется возможность задания SLA (Service Level Agreement, возможность гарантированного обеспечения требуемого качество сервиса), т.е. соглашения о качестве предоставляемых услуг для всех типов ресурсов ЦОД.

Во время размещения в ЦОД запроса может возникнуть необходимость перемещения элементов уже размещенных запросов (миграция элементов запросов). Это может произойти, если при попытке назначения какого-нибудь элемента запроса не находится ни один физический ресурс, на который можно было бы поместить этот элемент, но суммарно физических ресурсов в ЦОД для размещения такого элемента хватает. Это связано с проблемой фрагментации ресурсов, которая возникает в результате удаления старых и размещения новых элементов запросов в ЦОД.

С фрагментацией нужно каким-то образом бороться, иначе ЦОД начнет деградировать - физические ресурсы будут простаивать. Поэтому в ЦОД должна проводится дефрагментация физических ресурсов. Под дефрагментацией можно понимать более компактное расположение запросов в ЦОД. Чтобы это делать, необходимо уметь проводить миграцию уже размещенных виртуальных элементов. Из требования гарантированного обеспечения запрошенного качества сервиса следует, что миграция виртуальных элементов должна проводится без их останова. Таким образом, миграция элементов запросов должна помочь уменьшить фрагментацию. Но процедура миграции достаточно затратная операция: она использует сетевые ресурсы ЦОД, также время окончания миграции сложно прогнозировать, из-за этого в некоторых случаях лучше не проводить миграцию(даже если после нее утилизация ресурсов увеличилась бы), так как миграция может не завершиться или продолжаться так долго, что затраты на нее не окупятся полученным результатом.

Из-за непредсказуемости времени миграции возникает задача распределения ресурсов ЦОД с возможностью построения плана миграции выполняющихся виртуальных ресурсов в ЦОД, который удовлетворяет заданным ограничениям на время выполнения.

1 Цели выпускной квалификационой работы

Целями выпускной квалификационной работы являются разработка алгоритма распределения ресурсов в ЦОД с построением плана миграции виртуальных элементов, применение которого позволит уменьшить фрагментацию физических ресурсов ЦОД.

Для достижения указанной цели должны быть решены следующие задачи:

1. Изучить существующие алгоритмы размещения запросов в ЦОД и провести анализ возможности их расширения для построения плана миграции.
2. Разработать алгоритм размещения запросов в ЦОД с построением плана миграции, удовлетворяющего заданному ограничению на время выполнения.
3. Провести исследование свойств разработанного алгоритма.

2 Обзор предметной области

Целью написания данной главы является анализ существующих алгоритмов планирования, строящих одновременно с отображением запросов на физические ресурсы план миграции работающих вычислительных ресурсов и анализ возможности расширения существующих алгоритмов для построения плана миграции. С облачными вычислениями [1] связано много задач от чисто математических, алгоритмических до сугубо прикладных. В данной главе рассматривается только задача распределения ресурсов в центрах обработки данных и проблема миграции работающих виртуальных ресурсов.

2.1 Основные понятия, термины

Центр обработки данных

Под центром обработки данных(ЦОД) в данной работе понимается совокупность вычислительных узлов (физических серверов), хранилищ данных, сетевых ресурсов (коммутаторов, маршрутизаторов, сетевых кабелей). Все перечисленные сущности ЦОД связаны между собой и образуют некоторую **физическую топологию**. Рассмотрим основные **топологии**, которые используются при создании ЦОД [3]:

1. Conventional data center network topology [2]. Это трехуровневая топология:
    1. Core Layer - корневой уровень, который содержит корневые маршрутизаторы, имеющие доступ в интернет
    2. Agregation Layer -уровень агрегации, который состоит из коммутаторов
    3. Access Layer - уровень доступа, состоящий из серверов

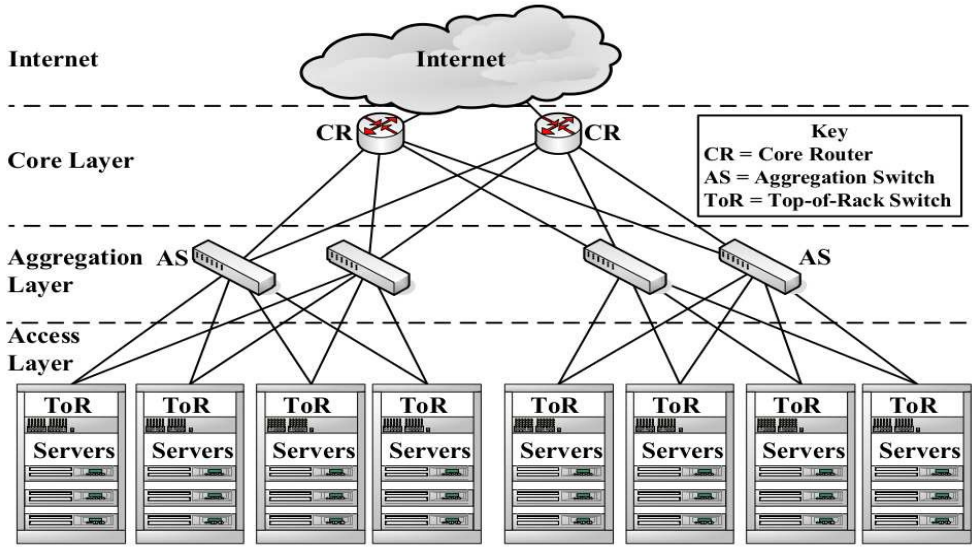

Рисунок 2.1

2. Clos topology [4]

Иерархическая система из коммутаторов, в которой могу отсутствовать блокировки. Это означает, что свободный вход входящего коммутационного элемента всегда может быть соединён со свободным выходом исходящего коммутационного элемента без необходимости перекоммутации уже существующих соединений.

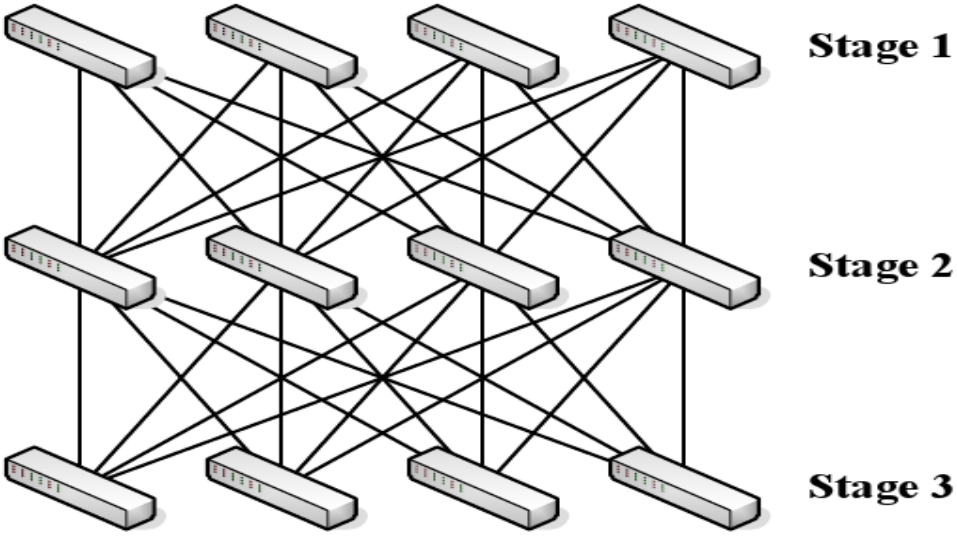

Рисунок 2.2

3. Fat-tree topology

Древовидная топология. Это специальный вариант Clos-топологии. Данная топология состоит из трех уровней: (K/2)^2 core-коммутаторов, K^2 aggregation-коммутаторов и K*(K/2)^2 серверов. Средний уровень состоит из K-портовых коммутаторов, которые объединены в K групп. В каждой группе K/2 коммутаторов являются aggregation-коммутаторы, которые связаны с верхним уровнем, другие K/2 коммутаторов – edge-коммутаторы, т.е. граничные, каждый из которые соединен с K/2 серверами. Верхний уровень, состоящий из core-коммутаторов, каждый i-ый порт которого связан с i-ой группой коммутаторов среднего уровня.

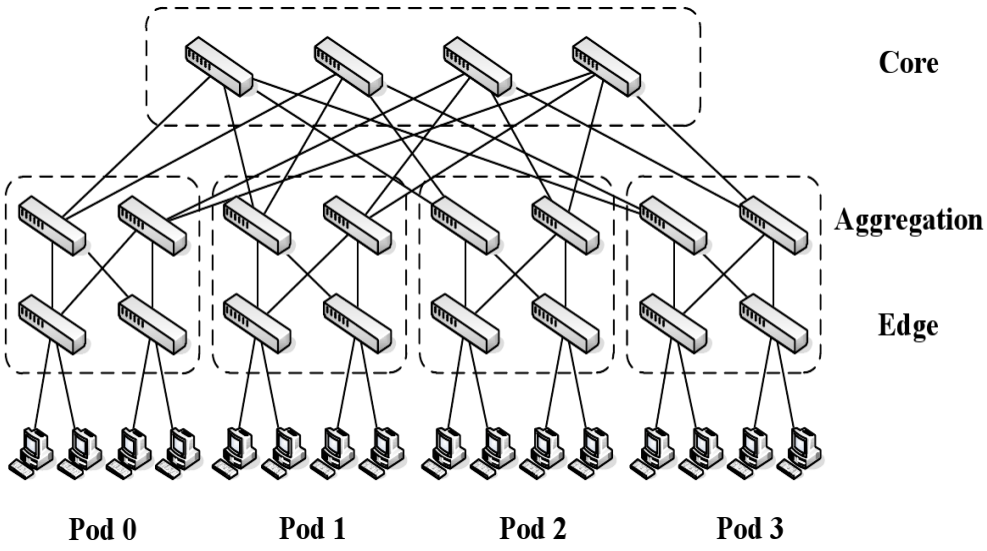

Рисунок 2.3

Виртуальный центр обработки данных

Виртуальный центр обработки данных (ВЦОД) – это виртуальная инфраструктура, которая позволяет создавать виртуальные машины, виртуальные хранилища данных, виртуальные коммутаторы и маршрутизаторы, а также виртуальные каналы связи. ВЦОД является абстракцией над физическим ЦОД. Для того чтобы создать ВЦОД в ЦОД применяется технология виртуализации. ВЦОД позволяет повысить уровень абстракции и оперировать не понятиями физических серверов и хранилищ, а работать с виртуальными сущностями. Такой слой виртуализации позволяет скрывать особенности физических машин и самое главное представлять ресурсы, как единое целое. Основная задача ВЦОД - принимать запросы от пользователей и размещать их с помощью средств виртуализации на физической топологии. Эту задачу выполняет облачная платформа.

В реальной жизни ответственность за работоспособность физических сущностей и виртуальных может разделяться[3] между поставщиком инфраструктуры (Infrastructure Provider, InP) и поставщиком сервисов (Service Provider, SP). InP следит за тем, чтобы сервера работали практически без сбоев, SP предоставляет сервисы, различные приложения, конечным пользователям и также следит за их работоспособностью.

Таким образом, ЦОД – это совокупность физических сущностей, ВЦОД – совокупность виртуальных сущностей.

Тенант

Тенант – это запрос пользователей в ВЦОД, который состоит из виртуальных машин,

виртуальных хранилищ данных, виртуальных коммутаторов и маршрутизаторов, а также виртуальных каналов связи. У каждой виртуальной сущности есть свои параметры. Например, у виртуальной машины – это количество ядер, RAM и объем жесткого диска. У виртуального хранилища свои параметры: объем, тип. Виртуальные коммутаторы и маршрутизаторы обычно не рассматривают как часть запроса, они являются программными и с точки зрения распределения ресурсов могут рассматриваться как виртуальные машины. Но, например, в работе [3] в качестве запрашиваемых ресурсов могли служить виртуальные коммутаторы. Виртуальные каналы зачастую имеют один параметр – это пропускная способность.

Виртуализация

Виртуализация [5, 6] — предоставление набора вычислительных ресурсов или их логического объединения, абстрагированное от аппаратной реализации, и обеспечивающее при этом логическую изоляцию вычислительных процессов, выполняемых на одном физическом ресурсе.

Облако

Совокупность ВЦОД и связанных с ними облачных платформ. Облака могут быть могут быть федеративными или централизованными [7].

Облачная платформа

Программная компонента ВЦОД, которая отвечает за принятие, обработку и исполнение запросов. Основные модули - это оркестратор ВЦОД и планировщик ресурсов ВЦОД. Также могут существовать и другие модули [8] — например, модуль мониторинга состояния виртуальных элементов и физических элементов.

Оркестратор

Оркестратор – это часть облачной, которая занимается принятием и исполнением запросов пользователей. Оркестратор должен знать текущее состояние ЦОД: утилизацию вычислительных и сетевых ресурсов, доступность серверов, иметь актуальную информацию от системы мониторинга. Когда оркестратор получает запросы, он должен понять, как отобразить эти запросы на физическую топологию ЦОД. Чтобы сделать это, ему нужен планировщик ресурсов, о котором будет сказано ниже. Именно он строит отображение запросов на физические ресурсы по определенному алгоритму. Отображение запроса в ЦОД

– это создание виртуальных сущностей запроса и размещение их на физической топологии. Виртуальные сущности создаются строго с теми параметрами, которые указал пользователь. Таким образом, при размещении должны соблюдаться SLA – service level agreement.

Планировщик ресурсов ВЦОД

В ВЦОД есть ограничения на размещение запросов. Построение отображения запросов, которое удовлетворяет этим ограничениям, осуществляет планировщик ресурсов ВЦОД. Входные данные о поступивших запросах, работающих виртуальных ресурсах и состоянии физических ресурсов ВЦОД планировщик получает от оркестратора.

Миграция

При очередной попытке размещения виртуального элемента в ВЦОД может возникнуть следующая ситуация: суммарное количество ресурсов хватает для размещения элемента, но не существует физического элемента, который бы полностью смог вместить этот элемент. Такая ситуация называется фрагментация ресурсов, т.е. неэффективное использование ресурсов. Она возникает при такой работе ЦОД, когда ресурсы постоянно выделяются и освобождаются. Чтобы ее избежать, нужно проводить дефрагментацию ресурсов, для этого необходима возможность миграции виртуальных машин и виртуальных хранилищ. Миграция достаточно затратный процесс, так как она задействует сетевые ресурсы ЦОД. Также миграция может не завершиться совсем, если мигрируют виртуальные машины, которые интенсивно работают с памятью. Поэтому планировщик облачной платформы должен уметь оценивать затраты на миграцию и, если она может завершиться за приемлемое время, строить план миграций.

2.2 Неформальная постановка задачи распределения ресурсов в ЦОД

Основываясь на терминах, введенных в предыдущем пункте, рассмотрим задачу распределения ресурсов в ЦОД. При получении запросов пользователей, облачная платформа должна разместить тенанты из запросов в ВЦОД таким образом, чтобы на протяжении всей жизни тенантов SLA не нарушались. Наиболее часто рассматривается задача, когда требуется разместить максимальное число запросов, но в ряде работ рассматриваются и другие постановки, например, минимизация нагрузки на сетевые элементы, минимизация потребляемой энергии, гарантия пропускной способности у каналов, утилизация вычислительных ресурсов. Более подробно различные постановки задачи распределения ресурсов в ЦОД рассмотрены в пункте 2.4. Задача распределения ресурсов в ЦОД является

NP-трудной задачей. К ней сводится задача об упаковке в контейнеры. В статье [9] NP-трудность задачи распределения ресурсов доказывается сведением к ней задачи о неделимом потоке с одним источником (single-source unsplittable flow [10]).

Размещение запросов - это NP-трудная задача многокритериальной оптимизации. Поэтому для ее решения в первую очередь используются различные эвристические алгоритмы: генетический, муравьиный алгоритм [11]. Также применяются жадные алгоритмы [12,13], алгоритм имитации отжига [14]. .

2.3 Критерий выбора алгоритма

Основной задачей данной работы является разработка алгоритма, который не только строит отображение тенантов пользователей, но и строит план миграции работающих виртуальных ресурсов. Миграция – один из способов увеличения утилизации ресурсов. Также миграция способствует снижению потребляемой энергии в ЦОД: сервера можно полностью освободить и отключить. Но миграция имеет много минусов, например, время миграции сложно предсказать, миграция сильно нагружает сеть.

В литературе пока не встречаются алгоритмы, решающие задачу, представленную в моей дипломной работе. Наиболее полный обзор различных классов алгоритмов распределения ресурсов ЦОД приведен в работе [23], но в данной выпускной работе мы ограничим класс рассматриваемых алгоритмов алгоритмами, которые наиболее «близки» к рассматриваемой в работе задаче по критериям: планируемыми ресурсами должны являться вычислительные ресурсы, хранилища данных и обязательно сетевые ресурсы, так как план миграции будет строиться таким образом, чтобы удовлетворять заданному ограничению по времени.

Также важным пунктом при обзоре алгоритмов – это возможность расширения алгоритма для построения плана миграции, т.е. в самом алгоритме должно быть заложено понятие миграции — это означает, что во время работы алгоритма виртуальные элементы могут менять свое назначение.

2.4 Обзор существующих алгоритмов

1. Рассмотрим алгоритм, представленный в [16]

Постановка задачи:

Разместить на минимальном количестве серверов виртуальные машины так, чтобы

количество мигрирующих виртуальных ресурсов было минимально. Также важным является поддержание утилизации ресурсов в некотором заданном диапазоне [min, max].

Алгоритм:

DRMA - Dynamic Resource Management Algorithm.

Применяется модификация алгоритма упаковки в контейнеры. В чистом виде алгоритм упаковки в контейнеры - минимизирует количество серверов, на которых располагаются задачи. Но он делает это за счет большого количества миграции. В данной работе предложен алгоритм, который позволяет снизить количество миграций.

Вывод:

В алгоритме рассматриваются две сущности: виртуальные машины и сервера. Данный алгоритм можно модифицировать так, чтобы и виртуальные хранилища тоже рассматривались в качестве планируемых ресурсов. Но в данной работе виртуальные каналы связи как планируемый ресурс не рассматриваются, следовательно, невозможно учесть ограничение на время миграции.

2. Рассмотрим алгоритм, представленный в [9].

Постановка задачи:

Основная задача - обеспечить максимальную утилизацию сетевых ресурсов.

Алгоритм:

В данной работе рассматривается эвристический алгоритм. Основная идея алгоритма - группировать соседние сервера и размещать их в подходящем кластере. При отображении тенанта ищется нужный кластер, в который можно его разместить. Этот подход приведет к компактному расположению запроса. Для отображения виртуальных машин на сервера применяется алгоритм нахождения максимального потока в транспортной сети [33].

Вывод:

В данной работе сетевые ресурсы не рассматриваются в качестве планируемых.

3. Рассмотрим алгоритм, представленный в [17].

Постановка задачи:

Основной задачей является минимизация коммуникаций в ЦОД между виртуальными машинами.

Алгоритм:

Минимизировать нагрузку на сеть можно при правильном размещении виртуальных машин, что и предлагается сделать в данной статье. Рассматривается задача - Traffic-Aware

Virtual Machine Placement Problem (TVMPP). Нужно помещать машины, которые интенсивно взаимодействуют, как можно ближе друг к другу. Такая стратегия способствует уменьшению количества узких каналов в ЦОД(bottleneck).

Вывод:

Ограничения предложенного алгоритма - трафик должен быть известен заранее. В реальности виртуальные машины могут принадлежать различным тенантам, на них запущены различные приложения, чей трафик неизвестен до развертывания. Также сеть и хранилища данных не рассматриваются в качестве планируемых ресурсов.

4. Рассмотрим алгоритм, представленный в [15].

Постановка задачи:

Максимизировать количество размещенных запросов.

Алгоритм:

Алгоритм состоит из двух стадий: размещение узлов, размещение каналов связи. Эти стадии не являются независимыми, так как иначе, по словам авторов, упадет качество получаемого решения. Проблема размещения тенантов сформулирована, как задача смешанного целочисленного программирования. Чтобы получить результат за полиномиальное время, исходная задача сводится к задаче линейного программирования.

Вывод:

В данной работе рассматриваются только виртуальные машины и каналы связи. У виртуальных машин только один параметр - количество ядер.

5. Рассмотрим алгоритм, представленный в [3].

Постановка задачи:

Автор рассматривает в качестве физических ресурсов сервера, коммутаторы и каналы связи. У каждого сервера есть несколько параметров: CPU, RAM, GPU, fast SRAM, RootDisk. Как утверждает автор, одной из нововведение его модели является введение в качестве планируемого ресурса коммутатора. Запрос состоит из виртуальных машин, виртуальных коммутаторов и виртуальных каналов связи. Основной задачей является максимизация количества размещенных запросов.

Алгоритм:

Так как задача является NP-трудной, то предлагается трехшаговый эвристический алгоритм: назначение виртуальных машин, назначение коммутаторов и назначение каналов. У данного эвристического подхода каждый следующий шаг сильно зависит от результатов

работы предыдущего. Поэтому автором был предложен следующее решение, возврат на предыдущий этап, если один этап не может быть выполнен. Основная идея эвристического алгоритма: минимизировать фрагментацию ресурсов серверов, минимизировать стоимость коммуникаций между виртуальными машинами и виртуальными коммутаторами, увеличить остаточную пропускную способность сети и предложить алгоритм балансировки нагрузки сети. При назначении коммутаторов и виртуальных машин используется алгоритм нахождения максимального потока в двудольном графе.

Вывод:

В данной работе не рассматривается миграция виртуальных элементов. Если виртуальные элементы уже размещены в ЦОД, то они не могут поменять свое расположение.

6. Рассмотрим алгоритм, представленный в [18].

Постановка задачи:

При планировании учитываются сетевые и вычислительные ресурсы, но каждая виртуальная машина и виртуальный канал связи не имеют параметров. В данной работе рассматривается задача поиска такого отображения запросов, при котором минимизируется заполнение физических каналов и узлов, т.е. запрос максимально распределяется по ЦОД.

Алгоритм:

Основной идеей алгоритма является разбиение исходного запроса на небольшие подзапросы в форме звезды и дальнейшее их размещение.

Вывод:

Главным недостатком предложенного алгоритма является отсутствие параметров у виртуальных элементов и ,как следствие, отсутствие SLA.

7. Рассмотрим алгоритм, представленный в [8].

Постановка задачи:

При планировании учитываются и вычислительные, и сетевые ресурсы. Основной задачей является поиск такого отображения виртуальных элементов на физические, при котором количество размещенных запросов было бы максимально.

Алгоритм:

Для назначения виртуальных машин используется эвристический алгоритм упаковки в контейнеры.

Вывод:

В данной работе не рассматривается миграция виртуальных элементов. Если виртуальные элементы уже размещены, то они не могут поменять свое расположение.

8. Рассмотрим алгоритм, представленный в [11].

Постановка задачи:

Максимизация количества размещенных запросов.

Алгоритм:

В данной работе рассматривается алгоритм на основе схемы муравьиных колоний.

Вывод:

В качестве планируемых ресурсов рассматриваются и вычислительные ресурсы, и сетевые ресурсы, но алгоритм обладает высокой вычислительной сложностью.

9. Рассмотрим алгоритм, представленный в [12].

Постановка задачи:

Максимизация количества размещенных запросов.

Алгоритм:

В данной работе рассматривается жадный алгоритм с ограниченным перебором. В работе рассматривается двухпроходный алгоритм: сначала назначаются виртуальные узлы, после производится назначение виртуальных каналов.

Вывод:

В качестве планируемых ресурсов рассматриваются и вычислительные ресурсы, и сетевые ресурсы. Не учитывается компактность расположения, что может сильно повлиять на результат прокладки виртуальных каналов.

10. Рассмотрим алгоритм, представленный в [13].

Постановка задачи:

Максимизация количества размещенных запросов.

Алгоритм:

В данной работе рассматривается жадный алгоритм с ограниченным перебором. Применяется алгоритм с общим планировщиком для всех типов ресурсов: виртуальные узлы и связанные с ним каналы планируются одновременно. Алгоритм строит компактное расположение запросов.

Вывод:

Данный алгоритм подходит по всем критериям. Достаточно легко можно встроить процедуру построения плана миграции.

11. Рассмотрим алгоритм, представленный в [19].

Постановка задачи:

Максимизация количества размещенных запросов

Алгоритм:

В статье для решения задачи предлагается использовать жадный алгоритм назначения запросов. Учитываются только виртуальные машины. Сеть не планируется, считается, что в запросах каналы все имеют нулевую пропускную способность. Алгоритм пытается разместить тенант в минимальное поддерево топологии ЦОД.

Вывод:

Алгоритм применим только для древовидной физической топологии. Не планируются сетевые ресурсы.

12. Рассмотрим алгоритм, представленный в [14].

Постановка задачи:

Максимизация количества размещенных запросов.

Алгоритм:

В данной работе решаются следующие задачи: назначение виртуальных узлов и виртуальных каналов. Используется алгоритм имитации отжига.

Вывод:

В данной работе не рассматривается миграция виртуальных элементов. Если виртуальные элементы уже размещены, то они не могут поменять свое расположение.

13. В работах [20,21,22] в качестве планируемых ресурсов рассматриваются только сетевые ресурсы. Главной задачей является построение такого отображения виртуальных элементов на физические, при котором гарантированно бы не нарушались заданные пропускные способности для виртуальных каналов связи.

2.5 Обобщение найденных подходов

Обзор алгоритмов распределения ресурсов ЦОД показал, что введенным в разделе 2.3 критериям удовлетворяет три алгоритма: алгоритм, основанный на схеме муравьиных колоний, алгоритм с раздельными планировщиками ресурсов и алгоритм с единым планировщиком для всех типов ресурсов.

Результаты обзора рассмотренных алгоритмов сведены в таблицу и представлены в Приложении А.

2.6 Проблема миграции виртуальных машин

В статье [23] указываются факторы, влияющие на миграцию. Накладные расходы в виде: загрузка диска, энергетические расходы, длительность миграции, захват сетевых и вычислительных ресурсов – исследовались в статье [24]. В работе [25] показано, что восстановление работающего приложения может занять больше времени, чем миграция виртуальной машины, на котором развернуто это приложение. Еще один фактор, влияющий на миграцию: загрузка самой виртуальной машины, ее активность, например, интенсивность записи в оперативную память, также важным фактором является загрузка сети – [26]. Различные технологические подходы, как проводить миграцию, рассматриваются в статьях: [25]. Можно проводить параллельную миграцию, которая поможет сократить время, но повышаются накладные расходы. В статье [26] исследуются некоторые методы уменьшения накладных расходов при миграции. Существует два подхода для передачи оперативной памяти виртуальной машины при миграции:

1. Миграция памяти перед началом копирования всей виртуальной машины. Оперативная память копируется, когда виртуальная машина все еще работает на исходном месте назначения. Если скопированные участки памяти поменялись, то происходит их повторное копирование. После истечении определенного времени виртуальная машина останавливается и полностью копируется на место нового назначения.
2. Миграция памяти после копирования виртуальной машины. В самом начале процедуры миграции виртуальная машина “замирает” - переходит в состояние suspend. Далее происходит копирование минимального количества информации, описывающего состояние виртуальной машины – состояние центрального процессора, значения регистров, иногда содержимое оперативной памяти. После этого на новом

месте назначения поднимается виртуальная машина с данными характеристиками и одновременно с этим запускается копирование оставшейся информации.

Описанные два подхода исследуется в работе [27]. Есть работы, в которых рассматривают миграцию только одной машины в конкретный момент времени [28]. В работах [26, 29] рассматривают миграцию нескольких машин в один и тот же момент времени. В работе [30] исследовался вопрос о времени миграции виртуальной машины в зависимости от взаимного влияния виртуальных машин, сетевой топологии и состояния каналов связи.

Во всех рассмотренных выше статьях в основном обсуждаются технологические проблемы миграции. Но ни в одной из статей не был предложен алгоритм построения плана миграции. Время окончания миграции сложно предсказать, это делает задачу построения плана миграции еще сложнее.

В работе [31] предлагается эвристический алгоритм для построения плана миграции. Но в данной работе не учитывается время, затрачиваемое на миграцию. Алгоритм строит последовательность перемещений, которые нужно совершить. Перемещение представляется в виде тройки <v, s, d>, где v — перемещаемый виртуальный элемент, s — исходное место назначения виртуального элемента, d — новое место назначения виртуального элемента. Задача: найти оптимальную последовательность перемещений вида <v, s, d>. В данной работе показывается, что задача построения такого плана миграции является NP-трудной задачей. К задаче построения плана миграции сводится задача планирования — Sequence Planning Problem [32]. В работе [31] приводится два алгоритма для решения задачи построения плана миграции. Первый алгоритм является переборным алгоритмом, который рассматривает все возможные перестановки перемещений, который необходимо совершить. Второй алгоритм имеет квадратичную сложность от количества перемещений и состоит в следующем: сначала генерируем произвольную последовательность перемещений, потом просматриваем все тройки <v, s, d>, вычисляем для каждой тройки число возможных миграции, которые можно будет провести после перемещения v. После сортируем по убыванию этого числа все эти тройки. Полученная отсортированная последовательность и будет планом миграции.

В данной статье были представлены результаты сравнения качества получаемых решений следующих алгоритмов: случайного алгоритма(произвольная последовательность перемещений), предложенного в статье эвристического алгоритм и оптимального решение.

В экспериментах отмечено, что оптимальное решение и решение, полученное описанным в статье эвристическим алгоритмом, всегда близки друг к другу, что говорит в

пользу эвристического алгоритма.

2.7 Выводы

В данной главе был проведен аналитический обзор существующих алгоритмов. Из таблицы, приведенной в Приложении А, видно, что три алгоритма: алгоритм с единым планировщиком, с раздельным планировщиком ресурсов и алгоритм, основанный на схеме муравьиных колоний — подходят по всем критериям, которые были сформулированы в пункте 2.3. В моей работе за основу был взят алгоритм распределения ресурсов с единым планировщиком, который был в значительной мере переработан: была изменена общая схема алгоритма, были добавлены новые жадные критерии и изменена процедура планирования сетевых ресурсов.

Остальные два алгоритма имеют ряд недостатков. Муравьиный алгоритм имеет большую вычислительную сложность, чем жадный алгоритм. Алгоритм с раздельными планировщиками не размещает компактно запросы, из-за этого нагрузка на сеть возрастает.

3 Задача построения распределения ресурсов в ЦОД с построением плана миграции

3.1 Математическая постановка задачи построения распределения ресурсов ЦОД

Математическую постановку задачи распределения ресурсов в ЦОД возьмем из работы [4] и расширим ее для задачи распаределения ресурсов в ЦОД с построением плана миграции.

Модель физических ресурсов ЦОД будем задавать графом $H=(P\cup M\cup K,L)$, где P – множество вычислительных узлов, M – множество хранилищ данных, K – множество коммутационных элементов сети обмена ЦОД, L – множество физических каналов передачи данных. На множествах P, M, K и L определены векторные функции скалярного аргумента, задающие соответственно характеристики вычислительных узлов, хранилищ данных, коммутационных элементов и каналов передачи данных:

$$\begin{aligned}
&(ph_1,ph_2,\ldots,ph_{n1})=fph(p),\\
&(mh_1,mh_2,\ldots,mh_{n2})=fmh(m),\\
&(kh_1,kh_2,\ldots,kh_{n3})=fkh(k),\\
&(lh_1,lh_2,\ldots,lh_{n4})=flh(l).
\end{aligned} \quad (3.1)$$

В данной работе предполагается, что для коммутационных элементов и каналов передачи данных задаются одинаковые характеристики.

Ресурсный запрос будем задавать графом $G=(W\cup S,E)$, где W – множество виртуальных машин, используемых приложениями, S – множество виртуальных хранилищ данных (storage-элементов), E – множество виртуальных каналов передачи данных между виртуальными машинами и storage-элементами запроса. На множествах W, S и E определены векторные функции скалярного аргумента, задающие характеристики запрашиваемого виртуального элемента (SLA):

$$\begin{aligned}
&(wg_1,wg_2,\ldots,wg_{n1})=fwg(w),\\
&(sg_1,sg_2,\ldots,sg_{n2})=fsg(s),
\end{aligned} \quad (3.2)$$

$(eg_1, eg_2, \ldots, eg_{n4}) = feg(e)$.

Характеристики SLA элемента запроса совпадают с характеристиками соответствующего ему физического ресурса.

Назначением запроса называется отображение *B*:

$B : G \rightarrow H \cup \{Ø\} = \{W \rightarrow P \cup \{Ø\}, S \rightarrow M \cup \{Ø\}, E \rightarrow \{K, L\} \cup \{Ø\}\}$ (3.3)

Запрос *G* называется *назначенным полностью*, если

$\forall g \ G : B(g) != Ø$ (3.4)

Запрос *G* называется **назначенным частично**, если

$\exists\, g1 \in G: B(g1) != Ø \ и \ \exists\, g2 \in G: B(g2) = Ø$ (3.5)

Множество всех запросов обозначим $G = \cup\{Gi\}$ (назначенных, частично назначенных и ожидающих назначения).

Совокупность назначений запросов будем обозначать *A*.

Выделим три типа отношений между характеристиками запросов и соответствующими характеристиками физических ресурсов. Обозначим через *x* характеристику запроса и через *y* соответствующую ей характеристику физического ресурса. Тогда эти отношения можно записать следующим образом:

1. Недопустимость перегрузки емкости физического ресурса:

$\sum_{i \in R_j} x_i \le y_j$, здесь R_j - множество запросов, назначенных на выполнение на физическом ресурсе *j*.

2. Соответствие типов у запрашиваемого ресурса и физического ресурса:

$x = y$.

3. Наличие требуемых характеристик у физического ресурса:

$x \le y$.

Отображение $A : G \rightarrow H = \{W \rightarrow P, S \rightarrow M, E \rightarrow \{K, L\}\}$ будем называть **корректным**, если для всех физических ресурсов и всех их характеристик выполняются отношения 1-3.

Остаточным графом доступных ресурсов называется граф H_{res}, получаемый из графа

физических ресурсов H, для которого переопределены значения функций по характеристикам, которые должны удовлетворять отношению корректности 1:

$$fph_{res}(p) = fph(p) - \sum_{w \in W_p} fwg(w), \quad fmh_{res}(m) = fmh(m) - \sum_{s \in S_m} fsg(s),$$

$$fkh_{res}(k) = fkh(k) - \sum_{e \in E_l} feg(e), \quad flh_{res}(l) = flh(l) - \sum_{e \in E_k} feg(e) \qquad (3.6)$$

Здесь W_p - множество виртуальных машин, назначенных на выполнение на вычислительном узле p, E_l - множество виртуальных каналов, отображенных на физический канал l, E_k - множество виртуальных каналов, проходящих через коммутационный элемент k, S_m - множество storage-элементов, размещенных в хранилище данных m.

В качестве **исходных данных** задачи назначения ресурсных запросов на физические ресурсы заданы:

1. множество запросов (назначенных, частично назначенных и ожидающих назначения) $Z = \{Gi\}$, поступивших планировщику;
2. остаточный граф доступных ресурсов $H_{res} = (P \cup M \cup K, L)$.

***Требуется*: из множества *Z* разместить на выполнение в ЦОД максимальное число запросов таких, что отображение *A* является корректным.**

3.2 Математическая постановка задачи построения плана миграции

Для построения плана миграции в ЦОД требуется расширить математическую модель задачи планирования. Построенный план должен удовлетворять ранее приведенным требованиям корректности (все ресурсы, привлеченные к миграции элемента запроса, должны удовлетворять отношениям корректности). Это означает, что во время миграции виртуальных элементов SLA запросов не нарушаются.

Расширим модель ЦОД, приведенную в работе [4], введя следующие понятия: **задание на перемещение** и **перемещение** виртуальных элементов в ЦОД.

Задание на перемещение – это тройка $T = (e, s, d)$, где $e \in G$; $s, d \in H$, т.е. e — это виртуальный элемент: VM или storage-элемент; s и d — это физические ресурсы, которые соответствуют старому и новому назначению виртуального элемента e; $s != d$. **Задание на перемещение** T показывает, что виртуальный элемент e нужно переместить

с физического ресурса *s* на физический ресурс *d*.

Перемещением будем называть пятерку $TR = (T, ts, tp = \tau(T, H), \{Li\}, tun)$, где T — задание на перемещение, ts — время начала перемещения, tp — время выполнения задания на перемещение, зависящее от задания и текущего состояния ресурсов ЦОД, tun — виртуальный канал миграции, пропускная способность которого зависит от интенсивности миграции в ЦОД, $\{Li\}$ — множество физических каналов, по которым проходит виртуальный канал миграции.

План миграции — это последовательность перемещений, которая удовлетворяет следующим условиям корректности:

1. все перемещения укладываются в заданный директивный интервал
2. при перемещении виртуальных элементов SLA не нарушаются

В качестве **исходных данных** задачи построения плана миграции виртуальных элементов:

1. Множество заданий на перемещение $\{Ti\}$
2. Активность перемещаемых виртуальных элементов. Активность виртуальной машины — это интенсивность изменения данных в памяти, измеряется в Мб/с.
3. Ограничение на время выполнения плана миграции — $Tdir$

***Требуется*: для заданного множества заданий на перемещение построить корректный план миграции.**

3.3 Математическая постановка задачи распределения ресурсов ЦОД с построением плана миграции

Введенные в разделах 3.1. и 3.2 понятия позволяют сформулировать задачу распределения ресурсов в ЦОД с построением плана миграции.

В качестве **исходных данных** задано:

1. Множество поступивших запросов $Z = \{Gi\}$
2. Множество выполняемых запросов $M = \{Gi\}$ и их отображение $AM:M \rightarrow H$. Для каждой виртуальной машины известна ее интенсивность работы.
3. Граф остаточных ресурсов ЦОД – $Hres$
4. Ограничение на время миграции – $Tdir$

Требуется:

1. Отобразить максимальное число запросов L из Z, которые можно разместить не нарушая SLA (множество $L \subseteq Z$), и построить отображения:

1. $AL: L \in Z \rightarrow Hres$
2. $A^{*}M: M \rightarrow Hres$

2. Построить план миграции для множества заданий на перемещений {Ti}, которое строится для множества ранее назначенных в ЦОД виртуальных элементов, но поменявших свое назначение в новом отображении. Время выполнения данного плана миграции не должно выходить за рамки директивного интервала — [0, Tdir]

4 Описание алгоритма распределения ресурсов в ЦОД с построением плана миграции

В данной главе приведено описание разработанного алгоритма распределения ресурсов ЦОД с построением плана миграции. Данный алгоритм представляет собой двухэтапный алгоритм: сначала назначаются виртуальные узлы запроса, после этого прокладываются виртуальные каналы связи между ними. В Приложении Б представлено детальное описание алгоритма на псевдокоде.

4.1 Общее описание алгоритма

Алгоритм осуществляет назначение элементов запроса в соответствии с жадной схемой, в случае невозможности назначения очередного элемента вызывается процедура ограниченного перебора. Для процедуры ограниченного перебора задается параметр, ограничивающий глубину перебора, что позволяет задавать требуемый баланс между вычислительной сложностью и точностью алгоритма. Если в процедуре ограниченного перебора виртуальные элементы, которые принадлежат выполняющимся запросам, поменяли свое назначение, то запускается процедура построения плана миграции.

Алгоритм состоит из следующих последовательно выполняемых шагов:

1. Рассмотрим множество вновь пришедших ресурсных запросов $\{G_i\}$.
2. Если множество $\{G_i\}$ не пусто, то выбрать очередной запрос G_i в соответствии с жадным критерием K_G, в противном случае – завершение работы алгоритма.
3. Сформировать из элементов запроса G_i множество виртуальных узлов $U = \{W \cup S\}$
4. Провести назначение виртуальных узлов U. В случае неуспеха перейти к **шагу 6**
5. Провести назначение виртуальных каналов E запроса G_i. В случае успеха перейти к **шагу 2**.
6. Провести снятие всех назначенных элементов запроса G_i , удалить запрос G_i из множества запросов $\{G_i\}$ и перейти к **шагу 2**

Ниже детально опишем шаги 4 и 5.

4.2 Процедура назначения виртуальных узлов на физические ресурсы

Виртуальные узлы $U = \{W \cup S\}$ обрабатываемого запроса G назначаются по следующему алгоритму:

1. Выбрать очередной элемент N из множества U в соответствии с жадным критерием K_V, поместить N в очередь Q элементов, ожидающих назначения.
2. Выбрать из Q элемент N и провести следующие действия:
   a) Сформировать множество физических узлов Ph, на которые может быть назначен данный элемент N, то есть для которых выполнены отношения корректности отображения в случае назначения запроса N на любой из этих физических узлов. Если данное множество пусто, то вызвать процедуру ограниченного перебора. При неуспешном завершении процедуры вернуть **НЕУСПЕХ**.
   b) Выбрать физический ресурс ph из множества Ph в соответствии с жадным критерием K_P. Назначить N на ph и переопределить значения характеристик физических ресурсов в соответствии с функциями (3.6).
   c) Выбрать все виртуальные каналы E, связывающие элемент N с еще не назначенными элементами запроса G_i.
   d) Отсортировать множество каналов E по величине пропускной способности в невозрастающем порядке
   e) Удалить N из Q и U
   f) Добавить в Q виртуальные узлы, связанные с N, согласно порядку связывающих их виртуальных каналов из отсортированного множества E.
   g) Если Q не пусто перейти к **шагу 2**
   h) Если множество U не пусто, то перейти к **шагу 1**, иначе вернуть **УСПЕХ**.

Процедура ограниченного перебора будет описана ниже в разделе 4.4.

4.3 Процедура назначения виртуальных каналов на физические ресурсы

Виртуальные каналы связи E обрабатываемого запроса G назначаются по следующему алгоритму:

1. Отсортировать все виртуальные каналы E согласно их пропускной способности в невозрастающем порядке
2. Выбрать из отсортированного множества E виртуальный канал L
3. При помощи алгоритма поиска в ширину [34] попытаться найти кратчайший маршрут для виртуального канала L. Если не получается проложить маршрут, то вернуть **НЕУСПЕХ**, в противном случае провести назначение виртуального канала на физические ресурсы и переопределить значения характеристик физических ресурсов в соответствии с функциями (3.6).
4. Если множество E не пусто, то перейти к **шагу 2**, иначе вернуть **УСПЕХ**

При поиске маршрута алгоритмом поиска в ширину рассматриваются только те каналы, в которых достаточно пропускной способности, чтобы разместить рассматриваемых виртуальный канал.

4.4 Процедура ограниченного перебора.

Процедура ограниченного перебора взята из работы [13]. Процедура ограниченного перебора вызывается при отсутствии возможности назначить очередной виртуальный узел *N* из запроса *G* ни на один физический ресурс *ph* (См. Раздел 4.2, процедура назначения виртуальных узлов, шаг 2). Процедура рассматривает некоторые подмножества множества физических узлов графа физических ресурсов. Мощность подмножеств – это глубина перебора *d*. Количество просматриваемых подмножеств ограничивается числом *MA*. Просматриваются только те подмножества, у которых суммарное количество остаточных ресурсов у узлов достаточно для назначения текущего элемента *N*.

Общая схема процедуры ограниченного перебора:

I. Для заданного *d* сформировать все подмножества *Cs* множества физических узлов физического графа ресурсов из *d* физических ресурсов.
II. Если для некоторого множества *C* из *Cs* суммарная остаточная емкость ресурсов достаточна для назначения текущего элемента:
    1. Сохранить назначение виртуальных узлов и каналов
    2. Произвести снятие элементов *V*, которые назначены на физические ресурсы из множества C и для которых разрешена миграция.
    3. Произвести снятие всех виртуальных каналов, связанных с элементами из множества *V*.
    4. Производим попытку назначить элементы множества *{V∪N}* с использованием жадной схемы на физические узлы из множества *C* (См. Раздел 4.2, процедура назначения виртуальных узлов без вызова процедуры ограниченного перебора). Если не удалось назначить элементы, то переходим на **шаг 8**
    5. Пытаемся проложить виртуальные каналы для тех виртуальных узлов из V, которые не принадлежат запросу *G (N* принадлежит *G)*. Если каналы проложить не удалось, то переходим на **шаг 8**
    6. Вызываем процедуру построения плана миграции. Если она возвращает неуспех,

то переходим на **шаг 8**

7. В случае успешного построения плана миграции вернуть **УСПЕХ**.
8. Восстанавливаем старое назначение узлов *V* и связанных с ними каналов. Удаляем множество *C* из *Cs*. Рассматриваем следующее множество *C* - переходим к **шагу II**, если множество *Cs* пусто, то возвращаем **НЕУСПЕХ**.

4.5 Процедура построения плана миграции виртуальных элементов в ЦОД.

Приведем описание алгоритма построения плана миграции. **Входными данными** алгоритма являются:

1. Граф остаточных физических ресурсов – H_{res}
2. Множество виртуальных элементов *U*, для которых нужно провести миграцию.
3. Старое – A_{old} и новое – A_{new} отображение мигрирующих виртуальных элементов.
4. Размер директивного интервала T_{dir}, в рамках которого должна выполняться миграция.

Выходными данными является построенный план миграции, удовлетворяющий условиям корректности: SLA не должны нарушаться, все перемещения обязаны укладываться в директивный интервал. Если такой план миграции построить не удалось, то возвращается **НЕУСПЕХ**.

Для объяснения работы алгоритма вводятся следующие обозначения:

1. *{Ti}* — множество заданий на перемещение. $|\{Ti\}| = k$ (*k* заданий на перемещение вида $T = < ei, si, di >$).
2. *T* - произвольное задание на перемещение вида $< e, s, d >$.
3. *V* (или *Vi*) - размер виртуального элемента. Для storage-элемента *V* определяется его размером в некоторых условных еденицах (Mb, Gb), для виртуальной машины — суммой оперативной памяти, дисковой памяти.
4. *A = activity* (или *Ai*) - интенсивность работы виртуального элемента; показывает, какой объем данных в условных единицах(Mb, Gb) изменяется в единицу времени (second, minute, hour). *Activity* прямо пропорциональна интенсивности записи в память storage-элемента или VM. Чем выше интенсивность работы элемента, тем больше данных нужно будет передать в момент его миграции. Пусть некоторая часть данных уже переместилась с физического ресурса *s* на ресурс *d*, но если интенсивность работы ненулевая

может возникнуть ситуация, когда эта же часть данных изменится и ее нужно будет опять перемещать. Поэтому интенсивность работы виртуального элемента во время миграции является важной характеристикой и ее нужно учитывать, чтобы оценивать время, которое будет затрачено на перемещение.

5. *{TRi}* - план перемещения. *|{TRi}| = k. TRi = <Ti, tsi, tpi, {Li}, tun >. TRi* - перемещение виртуального элемента – *ei*.
6. *tc* – время конца миграции. Значение *tc* лежит в полуинтервале *[0, T_{dir})*.
7. *tfi = tsi + tpi* .Время окончания миграции виртуального элемента *ei*
8. *Q* - упорядоченная очередь заданий на перемещение.
9. $H_{res}(t)$, где *t* - это время. Граф остаточных ресурсов $H_{res}(t)$ рассчитывается для конкретного момента времени таким образом:
    a) До начала построения плана миграции, у нас есть граф остаточных ресурсов $H_{res}(0)$, который принимается как входные данные.
    b) Дальше для любого момента времени *t*, чтобы определить $H_{res}(t)$ просматриваем все перемещения – *{TRi}* (в этом множестве в момент времени *t<tc* может находиться только часть перемещений) и захватываем в $H_{res}(0)$ те физические каналы *Li*, для которых выполнено: *tsi <= t && t < tfi* , т.е. захватываем те каналы, которые в момент времени t заняты. Также нужно в $H_{res}(t)$ захватить временные реплики перемещаемых элементов, т.е. для *TRi*, выполняющегося в момент *t*, захватить ресурсы под элемент *ei* и на физическом элементе *s*, и на элементе *d*.
10. Перемещения могут производиться параллельно, поэтому сетевые ресурсы не отдаются полностью под миграцию какого-либо виртуального элемента, а делятся пропорционально их интенсивности. Для нахождения пропускной способности виртуального канала миграции tun для виртуального элемента *ei*, проводится следующая процедура: во множестве *{Li}* находится канал с самой маленькой пропускной способностью – *throughput*, после пропускная способность канала tun высчитывается по формуле:

$$tun.throughput = throughput * \left(\frac{V_i * (A_i + 1)}{\sum_{j=1}^{k} V_j * (A_j + 1)} \right) \quad (4.1)$$

Пропускную способность канала миграции можно высчитывать следующим образом

$$tun.throughput = throughput \quad (4.2)$$

Эти два варианта будут исследоваться далее в разделе с экспериментальн

11. Оценка времени миграции виртуального элемента является непростой задачей, поэтому в данной курсовой время миграции считается прямо пропорциональным отношению перемещаемых данных к пропускной способности канала передачи данных:

$$tp = \frac{V}{tun.throughput - Activity} \quad (4.3)$$

Если *Activity* больше, чем выделенная пропускная способность канала миграции *tun.throughput*, то соответствующее перемещение будет запрещено.

Заранее неизвестно, сколько времени потребуется на каждое перемещение, следовательно, задача построения плана миграции виртуальных элементов не является классической задачей построения расписания. Поэтому для решения данной проблемы был предложен алгоритм, описанный ниже:

A) Используя множество *U = {ei}, 1<=i<=k*, формируем множество заданий на перемещение *{Ti}*.
B) Упорядочиваем множество *{Ti}* по уменьшению произведения *Vi*(Ai+1)* и формируем из него очередь *Q*. Это делается для того, чтобы сначала попробовать переместить самые “трудные” элементы. Если у элемента интенсивность высокая, следовательно, и передаваемых данных будет много, так что даже с небольшим по *V* элементом, но с очень большой интенсивностью могут возникнуть сложности при перемещении.
C) Извлекаем очередной элемент из очереди *Q*.
D) Инициируем нулем переменную *observeTime*, которая отвечает за рассматриваемое в данный момент точку в расписании.
E) Если текущее время равно *tc*, переходим к пункту (F). Иначе пытаемся поставить перемещение в текущее время – *observeTime*:
    i. Формируем $H_{res}(observeTime)$.
    ii. Ищем путь минимальный по заданной стоимости в графе $H_{res}(observeTime)$ от вершины *si* до вершины *di*. Поиск производится при помощи алгоритма Дейкстры [35]. Стоимость ребра (физического канала) в графе – это занятая пропускная способность. Если не удалось найти путь, переходим к подпункту (iv).

iii. Проверяем, превысили ли мы время миграции: если превысили, то переходим к пункту (iv), иначе проверяем, мешаем ли каким-нибудь перемещениям. Если не мешаем никаким перемещениям, то к пункту (H), иначе переходим к подпункту (iv).

iv. Обновляем *observeTime* – пропускаем ближайшее к текущему времени перемещение. Переходим к подпункту (i).

F) Производим попытку переместить элемент *ei*. Ищем путь от *si* до *di* в графе $H_{res}(0)$. Если неуспех, то кладем обрабатываемое *Ti* в конец очереди и переходим к пункту (I).

G) Проверяем, превысили ли мы время миграции: если превысили, то кладем обрабатываемое *Ti* в конец очереди и переходим к пункту (I).

H) Кладем сформированное перемещение *TRi* в множество уже обработанных перемещений *{TRj}*. Обновляем время *tc*.

I) Если очередь *Q* зациклилась, то алгоритм возвращает неуспех.

J) Если очередь *Q* пуста, следовательно, план миграции построен – вернуть успех.

K) Вернуться к пункту (C).

Опираясь на выше описанный *алгоритм*, опишем *процедуру построения плана миграции*.

Пусть уже имеется план миграции *S* – он может быть пустым. Элементы, расположенные в ЦОД до запуска алгоритма распределения ресурсов и для которых разрешена миграция – *OLDm*. Пусть в процедуре ограниченного перебора элементы подмножества $OLDm^*$ множества *OLD* поменяли свое назначение. Если в S нет перемещений элементов из множества $OLDm^*$, следовательно, производится попытка дополнить план миграции *S* согласно выше описанному алгоритму – шаги A-K для множества $OLDm^*$. Если в плане миграции *S* есть перемещения элементов из $OLDm^*$, то старый строится новый план миграции для виртуальных элементов из *S* и $OLDm^*$. Если при дополнении плана миграции или при построении нового плана будет нарушен директивный срок, то производится откат к предыдущему корректному плану миграции или к пустому плану миграции и процедура построения плана миграции возвращает **НЕУСПЕХ**.

4.6 Жадные критерии

Качество работы жадного алгоритма существенно зависит от выбора жадных критериев. Критерии задаются для выбора очередного запроса – K_G, для выбора виртуального

узла - K_V и физического узла – K_P.

Критерий выбора очередного запроса для назначения K_G выбирает запрос, количество запрашиваемых виртуальных элементов в котором максимально.

Критерий K_V и K_P основан на критериях, рассматриваемом в работах [12, 13]. В рассматриваемых работах критерий основывается на функции стоимости, которая вычисляется, как взвешенная сумма требуемых характеристик с учетом их дефицита.

Рассмотрим виртуальный узел e. У этого узла есть параметры → $(r_{e,1}, r_{e,2}, \ldots, r_{e,n})$

Дефицит для i-го параметра вычисляется следующим образом →

$$d(i) = \frac{\sum_{G} \sum_{e \in E, E \in G} r_{e,i}}{\sum_{ph \in Ph} r_{ph,i}} \quad (4.4),$$

где G – граф запроса, E – виртуальные узлы запроса, e и v – виртуальные элементы запроса, Ph – физические узлы графа физических ресурсов, ph – физический узел.

Функция стоимости для виртуального элемента e →

$$r(e) = \sum_{i=1..n} d(i) * \left(\frac{r_{e,i}}{max r_{v,i}} \right)_{G, v \in G} \quad (4.5)$$

В работах [12, 13] K_V выбирает виртуальный элемент, у которого функция стоимости максимальна — делается это для того, чтобы сначала попробовать назначить самые тяжелые и дефицитные по ресурсам элементы. K_P выбирает физический элемент, у которого функция стоимости минимальна, чтобы достичь максимальной утилизации вычислительных ресурсов. Данные критерии не учитывают подходящие каналы к узлам. Критерий выбора физического узла не учитывает его положения в графе физических ресурсов.

В моей дипломной работе были предложены новые жадные критерии, для того чтобы исправить эти недостатки:

- Новый жадный критерий выбора виртуального узла e – выбирается узел с максимальным весом — *weight:*

$$weight(e) = r(e) * \sum_{l} Throughput(l) \quad (4.6)$$

l – виртуальные каналы, связанные с узлом e

- Новый жадный критерий выбора физического узла p – выбирается узел с минимальным весом — *weight:*

$$weight(p) = r(p) * \sum_{l} Throughput(l) * C(p) \quad (4.7)$$

l – физические каналы связи, связанные с узлом *p*

Throughput – текущая пропускная способность канала

$$C(p) = \sum_{e} r(e) * \left(\frac{min(bw(p,e))}{d(p,e)} \right) \quad (4.8)$$

e – физический узел в графе, до которого существует путь от узла *p*

min(bw(e,p)) – канал с минимальной пропускной способностью в кратчайшем пути от узла *p* до *e*

d(p,e) – реберное расстояние в кратчайшем пути от узла *e* до узла *p*

4.7 Выводы

В данной главе описан разработанный алгоритм распределения ресурсов ЦОД. Алгоритм состоит из двух последовательных шагов: назначения виртуальных узлов и назначения виртуальных каналов.

Назначение узлов производится по жадному алгоритму. В случае, когда очередной виртуальный узел не может быть размещен, вызывается процедура ограниченного перебора. В случае успешного завершения процедуры ограниченного перебора, вызывается процедура построения плана миграции, которая пытается построить план миграции, который удовлетворяет условиям корректности: во время миграции виртуальных элементов SLA не нарушаются, никакое перемещение в плане миграции не должно нарушать директивный срок. Поиск маршрутов для виртуальных каналов связи выполняется при помощи алгоритма поиска в ширину [34].

5 Описание программного модуля

Реализовано два программных модуля: модуль построения отображения запросов на физические ресурсы и модуль построения плана миграции. Реализованные программные модули являются расширением существующей реализации алгоритма распределения ресурсов ЦОД, описанном в работе [13].

Исходный программный модуль написан на языке C++, он же использовался в качестве основного языка программирования для реализации предложенного в данной работе алгоритма.

Модифицированный модуль поддерживает создание 12 алгоритмов планирования ресурсов ЦОД:

1. 3 различных алгоритма планирования ресурсов в ЦОД: алгоритм распределения ресурсов ЦОД с единым планировщиком ресурсов [13], с единым планировщиком и наивным поиском в ширину при прокладке виртуальных каналов [13] и разработанный алгоритм
2. Новый и старый жадный критерии
3. Включение или отключение процедуры построения плана миграции

5.1 Оценка вычислительной сложности реализованного алгоритма

Пусть N – число вычислительных узлов, S – число хранилищ данных, SW – число коммутаторов в сети передачи данных , L – число физических каналов передачи данных. Пусть далее RV – общее число виртуальных машин всех ресурсных запросов, RS – общее число виртуальных хранилищ данных, RL – общее число вируальных каналов. Через m обозначим глубину перебора в процедуре ограниченного перебора.

I. Процедура назначения виртуальных узлов. В рамках процедуры принимаются следующие действия:
   1. Формирование множества элементов для назначения и выбор очередного элемента для назначения (См. Раздел 4.2, шаги 1, 2a). Сложность этапа эквивалентна сложности сортировки массива и равна $C(W_V)=O((RV+RS)\log(RV+RS))$ (5.1)
   2. Введем обозначение $T=max(N,S)$. Шаг 2b алгоритма – подсчет весов для физических узлов имеет сложность $C(W_{ph}^T)$. При вычислении веса для нахождения кратчайших путей используется алгоритм Дейкстры(См. Раздел 4.6), поэтому

сложность данного шага равна: $C(W_{ph}^{T})=O((N+S+SW)^2)T$ (5.2)

3. Процедура ограниченного перебора. Количество перебираемых сочетаний вычисляется как $\max(C_N^m, C_S^m)$. Эту величину можно ограничить заранее заданным числом MA, чтобы не проводить полный перебор. На каждом этапе проводится три шага:
   a) Жадный алгоритм переназначения(См раздел 4.4). Пусть нужно переназначить t виртуальных элементов, тогда сложность жадного алгоритма не будет превышать $C_g=C(W_V)+C(W_{ph}^m)+t(m+C(W_{ph}^m))$
   b) Процедура построения плана миграции(См. Раздел 4.5). Количество заданий на перемещение не может превышать t. Размер очереди Q в каждый момент времени $q \le t$. На каждом шаге перемещение может удалиться из очереди или поместиться в ее конец. Худший случай — каждое перемещение находится в очереди $q-1$ шагов, но потом все-таки удаляется из очереди(иначе бы очередь зациклилась). Следовательно, всего шагов алгоритма не более, чем t^2. Вычислим теперь сложность одного шага алгоритма. Основной по сложности пункт — это п.(E). В этом пункте есть цикл, который будет выполняться не более, чем t раз, так как в интервале [0, tc) выполняется не более t перемещений. Сложность тела цикла формируется из трех частей: C1 — сложность формирования $H_{res}(t)$, C2 — алгоритм Дейкстры, C3 — проверка, мешает ли обрабатываемое перемещение уже назначенным. C1 не больше, чем $(O(t)+O(E))$ — производится поиск перемещений за $O(t)$ и назначение на физические каналы виртуальных каналов для миграции за $O(E)$. C2 — сложность алгоритма Дейкстры — $O((N+S+SW)^2)$. C3 — перемещений, которые могут мешать текущему не более, чем N; для каждого такого перемещения строится H_{res} (сложность C1), потом проверяется возможность миграции рассматриваемого перемещения — $O(E)$. Следовательно, общая сложность $C3=t(O(t)+O(E))$.

Итоговая сложность в наихудшем случае алгоритма построения плана миграции:

$$C_m = t^2((t+1)(O(t)+O(E))+O((N+S+SW)^2)) \quad (5.3)$$

c) Процедура переназначения каналов. В каждого переназнчаенного элемента нужно заново проложить виртуальные каналы – их не более, чем RL. Сложность данного шага $C_l = O(N+S+W+L)RLt$ (5.4)

4. Процедура ограниченного перебора может быть инициирована в процессе назначения любого элемента запроса. Принимая во внимание данный факт и вводя обозначение $Q = 2\max(RV, RS)$, получаем следующую оценку временной сложности процедуры назначения виртуальных узлов в наихудшем случае:

$$MA\{O(Q^4)+O(mQ^3)+O(2T+SW+L)RLQ+O((2T+SW)^2)(Q^2+mQ+m+T)+mQ\} \quad (5.5)$$

II. II. Процедура назначения виртуальных каналов. В наихудшем случае нужно назначить все RL виртуальных каналов. Для каждого канала маршрут строится с помощью алгоритма поиска в ширину [34]. Итоговая сложность процедуры в наихудшем случае: $O(N+S+SW+L)RL$ (5.6)

Итоговая сложность алгоритма распределения ресурсов ЦОД с построением плана миграции в наихудшем случае:

$$MA\{O(Q^4)+O(mQ^3)+O(2T+SW+L)RLQ+O((2T+SW)^2)(Q^2+mQ+m+T)+mQ\} \quad (5.7)$$

5.2 Модуль задачи планирования вычислений в ЦОД с построением плана миграции

На рисунке 5.1 приведена UML-диаграмма [36] классов модифицированного программного модуля для решения задачи планирования вычислений в ЦОД с построением плана миграции, который удовлетворяет заданному ограничению по времени. Красным отмечены разработанные и модифицированные классы.

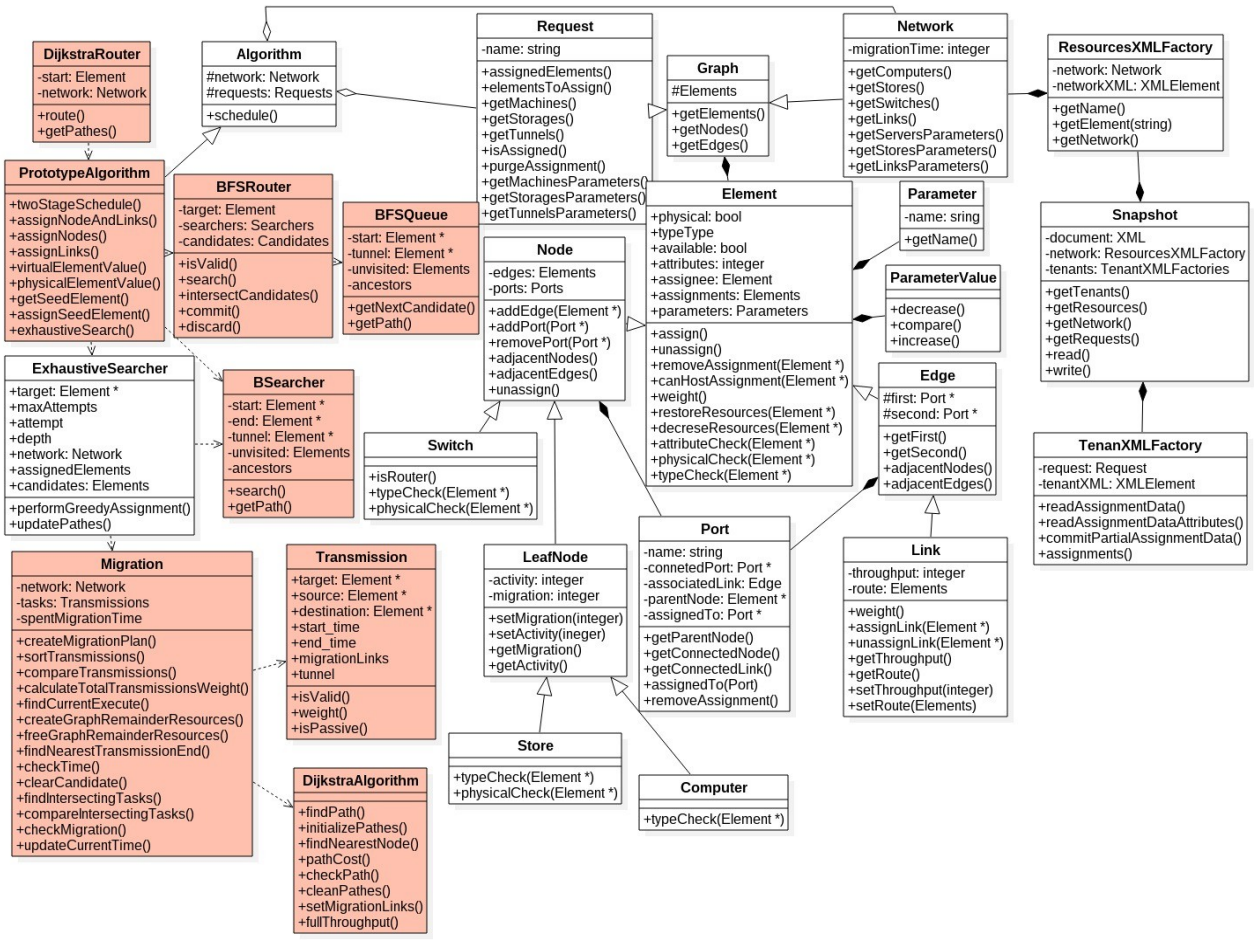

Рисунок 5.1 Диаграмма классов модуля задачи планирования вычислений в ЦОД с построением плана миграции

Разработанный класс *PrototypeAlgorithm* реализует двух шаговую схему назначения виртуальных ресурсов. Классы *BFSRouter*, *BFSQueue* - реализуют алгоритм поиска в ширину с пересечением кандидатов, описанный в работе [13], они были модифицированы таким образом, чтобы в них происходил захват сетевых ресурсов,. *Bsearcher* реализует алгоритм поиска в ширину, он тоже был модифицирован, чтобы происходил захват сетевых ресурсов. В разработнном классе *DijkstraRouter* реализуется алгоритм Дейкстры, который применяется для вычисления веса виртуального и физического элемента. Оставшиеся реализованные три

класса относятся к процедуре построения плана миграции: класс *Migration* реализует алгоритм построения плана миграции, класс *DijlstraAlgorithm* – алгоритм Дейкстры, используемый в алгоритме построения плана миграции(см. Раздел 4.5) и класс *Transmission* – реализация понятия перемещения, описанного в Разделе 3.2.

6 Экспериментальное исследование алгоритма

Сформулируем цели экспериментального исследования разработанного алгоритма:

1. Сравнить эффективность работы алгоритмов распределения ресурсов с единым планировщиков [13], с единым планировщиком и наивным поиском в ширину [34] при прокладке виртуальных каналов [13] и разработанного алгоритма.
2. Сравнить эффективность работы процедуры миграции при различном выборе пропускной способности канала миграции и при различной начальной загруженности ЦОД.
3. Сравнение на реальных данных работы алгоритмов распределения ресурсов с единым планировщиков [13], с единым планировщиком и наивным поиском в ширину при прокладке виртуальных каналов [13] и разработанного алгоритма.

Краткое описание используемой вычислительной системы:

1. Размер оперативной памяти – 32 Гб
2. Тип процессора – Intel Xeon CPU E5-2609, 2,40GHz, 8 cores
3. Версия операционной системы – Ubuntu 14.04.4 LTS, Trusty Tahr

6.1 Сравнение эффективности работы алгоритмов распределения ресурсов ЦОД

6.1.1 Исходные данные

Для проведения исследования эффективности работы алгоритмов распределения ресурсов с единым планировщиков [13], с единым планировщиком и наивным поиском в ширину при прокладке виртуальных каналов [13] и разработанного алгоритма в качестве физической топологии рассматривались топологии “толстого” дерева и древовидная топология. Граф запроса может иметь следующие топологии: дерева, кольца, звезды.

При генерации тестов параметры физической топологии могли меняться следующим образом:

1. Количество физических узлов - от 20 до 256
2. Параметры физических узлов:
   a) Количество ядер – от 32 до 64
   b) Количество RAM – от 32 до 64 Гб
3. Пропускные способности физических каналов связи – от 50 до 200 Мб/с

Параметры запросов могли меняться следующим образом:

1. Количество запросов – от 10 до 200
2. Количество виртуальных машин в запросе – от 15 до 30
3. Параметры виртуальных машин:
   a) Количество ядер – от 2 до 8
   b) Количество RAM – от 4 до 16 Гб
4. Пропускные способности виртуальных каналов – от 5 до 15 Мб/с

Все тесты были поделены на 6 классов – 2 варианта физической топологии и 3 варианта виртуальной топологии. Каждый класс содержит 6 тестов. В совокупности было проведено 36 тестов. Подробное описание исходных данных можно найти в Приложении В.

6.1.2 Результаты экспериментального исследования

Результаты сравнительного экспериментального исследования алгоритмов распределения ресурсов с единым планировщиком [13], с единым планировщиком и наивным поиском в ширину [34] при прокладке виртуальных каналов [13] и разработанного алгоритма представлены на рис. 6.1-6.6. Абсолютные данные, на которых основаны графики, приведены в табл. В1-В18.

Обозначения на рисунках:

1. Алгоритм распределения ресурсов с единым планировщиком и наивным поиском в ширину при прокладке виртуальных каналов [13] – Алгоритм 1
2. Разработанный в данной работе алгоритм – Алгоритм 2
3. Алгоритм распределения ресурсов с единым планировщиком [13] – Алгоритм 3
4. RAM обозначается синим цветом, CPU – красным цветом, сетевые ресурсы – желтым цветом
5. Утилизация ресурса конкретного типа – это отношение суммарного количества используемого ресурса к суммарному количеству имеющегося ресурса этого типа в ЦОД. Утилизация представляет собой число от 0 до 1
6. Для каждого теста для Алгоритмов 1, 2, 3 изображены результаты работы – утилизация RAM, утилизация CPU, утилизация сетевых ресурсов

На рис.6.1 показана утилизация ресурсов для алгоритмов 1, 2, 3 в случае топологии “толстого дерева” и запросов в форме дерева. График построен на основе табл. B1.

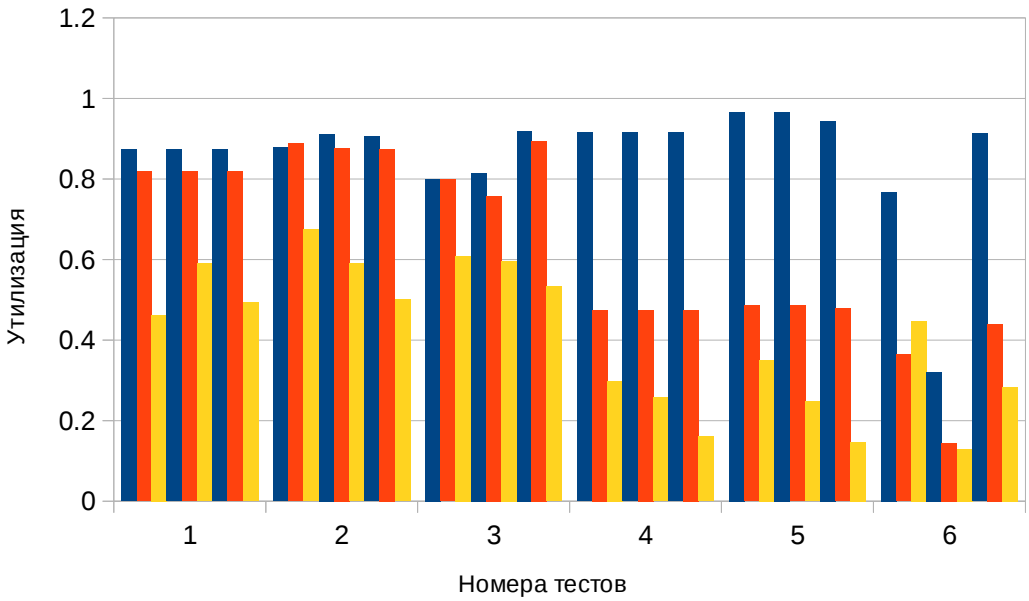

Рисунок 6.1. Утилизация ресурсов для Алгоритмов 1, 2, 3 в случае топологии “толстого дерева” и запросов в форме дерева.

На рис.6.2 показана утилизация ресурсов для алгоритмов 1, 2, 3 в случае топологии “толстого дерева” и запросов в форме кольца. График построен на основе табл. B2.

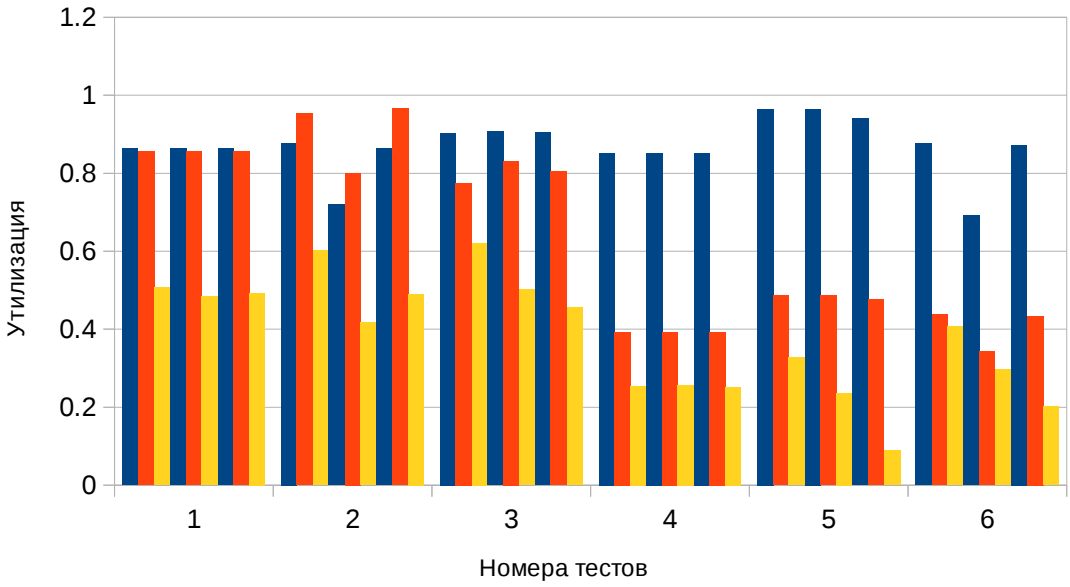

Рисунок 6.2. Утилизация ресурсов для Алгоритмов 1, 2, 3 в случае топологии “толстого дерева” и запросов в форме кольца.

На рис.6.3 показана утилизация ресурсов для алгоритмов 1, 2, 3 в случае топологии “толстого дерева” и запросов в форме звезды. График построен на основе табл. В3.

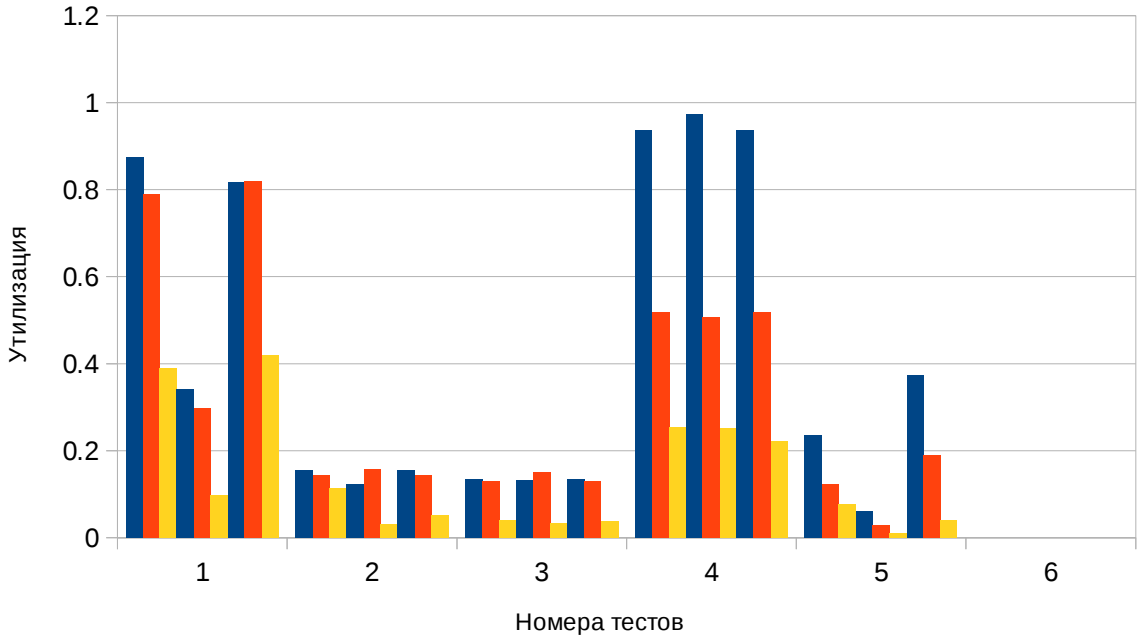

Рисунок 6.3. Утилизация ресурсов для Алгоритмов 1, 2, 3 в случае топологии “толстого дерева” и запросов в форме звезды

На рис.6.4 показана утилизация ресурсов для алгоритмов 1, 2, 3 в случае древовидной топологии и запросов в форме дерева. График построен на основе табл. В4.

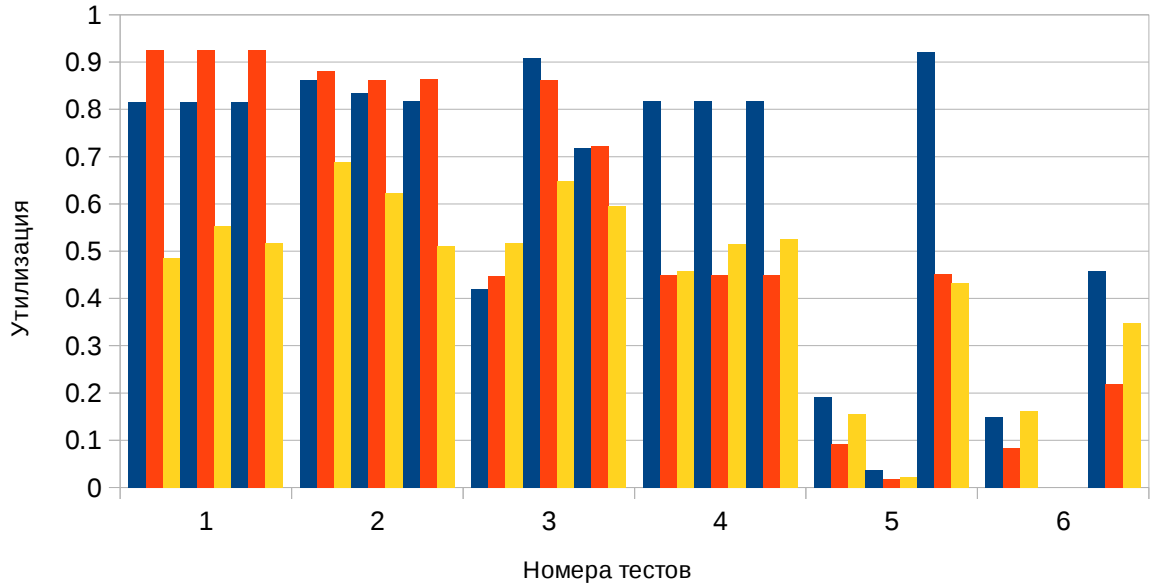

Рисунок 6.4. Утилизация ресурсов для Алгоритмов 1, 2, 3 в случае древовидной топологии и запросов в форме дерева.

На рис.6.5 показана утилизация ресурсов для алгоритмов 1, 2, 3 в случае древовидной топологии и запросов в форме кольца. График построен на основе табл. B5.

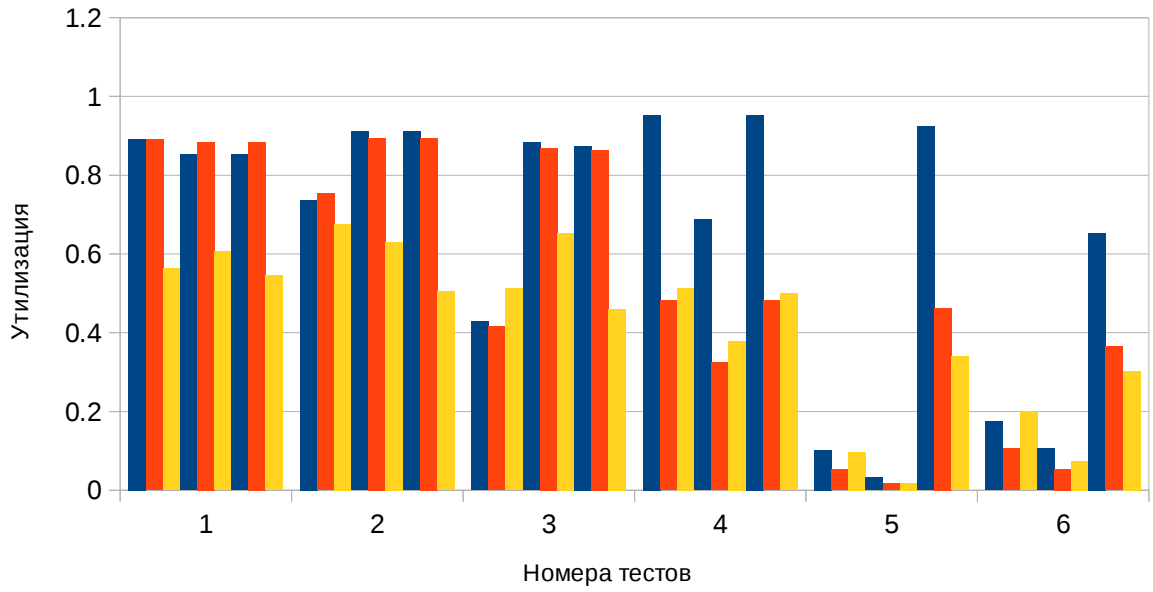

Рисунок 6.5. Утилизация ресурсов для Алгоритмов 1, 2, 3 в случае древовидной топологии и запросов в форме кольца.

На рис.6.6 показана утилизация ресурсов для алгоритмов 1, 2, 3 в случае древовидной топологии и запросов в форме звезды. График построен на основе табл. B6.

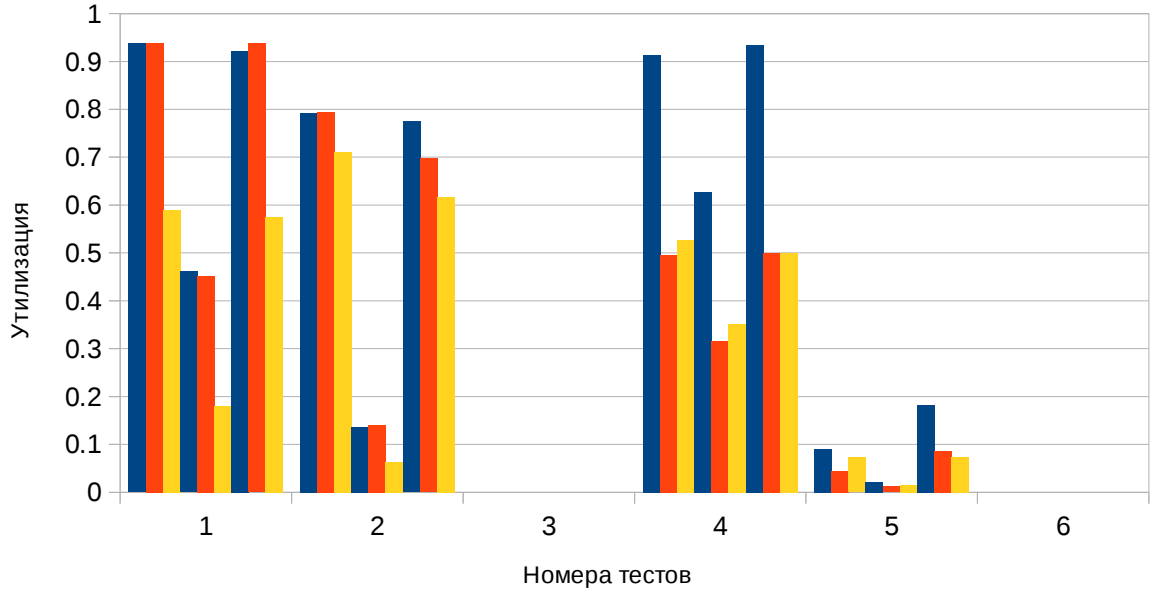

Рисунок 6.6. Утилизация ресурсов для Алгоритмов 1, 2, 3 в случае древовидной топологии и запросов в форме звезды.

6.2 Сравнение эффективности работы процедуры миграции при различном выборе канала миграции и при различной начальной загруженности ЦОД

6.2.1 Исходные данные

Все тесты проводились на топологии, приведенной в Приложении Е. Рассматривается два состояния ЦОД : при небольшой загрузке каждого узла(до 50%) и сильно загруженный ЦОД(выше 70%). Запросы состоят только из виртуальных машин. В ЦОД могут быть два типа запросов:

- легковесный(light) — 5 CPU, 400 условный единиц RAM
- тяжеловесный(heavy) — 20 CPU, 1000 условный единиц RAM

В ходе эксперимента анализировалось 8 различный ситуаций работы ЦОД:

- Два варианта загрузки ЦОД:
  - сильно загруженный — все узлы загружены более, чем на 50%
  - слабо загруженный — все узлы загружены менее, чем на 50%
- Два варианта расположения запросов в ЦОД до начала планирования — преобладают легковесные или тяжеловесные запросы.
- Два варианта новых запросов в ЦОД — преобладают легковесные или тяжеловесные запросы.

Введем следующие обозначения:

- I — ситуация со слабо загруженным ЦОД
- II — ситуация с сильно загруженным ЦОД
- III — ситуация с преобладанием тяжеловесных запросов
- IV — ситуация с преобладание легковесных запросов
- Пара (римская цифра№1, римская цифра№2) означает, что ЦОД находится в ситуации соответствующей римской цифре №1, а вновь пришедшие запросы соответствуют римской цифре №2

6.2.2 Результаты экспериментального исследования

Результаты тестов при различной начальной загруженности ЦОД приводятся в Приложении Г. Результаты тестов при различном выборе пропускной способности канала миграции приводятся в Приложении Д. Топология для проведения экспериментов приводится в Приложении Е. В разделе 4.5 было указано, что канал для миграции можно

выбрать двумя способами.

Способ №1:

$$tun.throughput = throughput * \left(\frac{V_i * (A_i + 1)}{\sum_{j=1}^{k} V_j * (A_j + 1)} \right) \quad (6.1)$$

Способ №2:

$$tun.throughput = throughput \quad (6.2)$$

Ниже приведена диаграмма, показывающая зависимость времени миграции от выбора способа, задающего пропускную способность канала миграции. Время миграции измеряется в секундах.

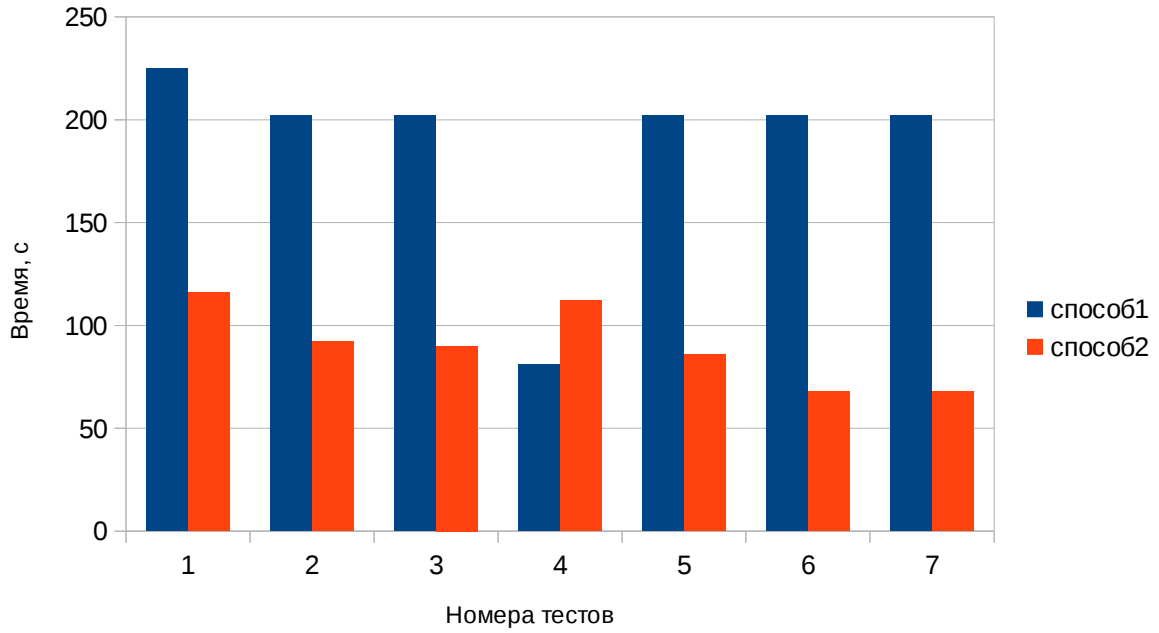

Рисунок 6.7. Время миграции при различном выборе пропускной способности канала и различной загруженности ЦОД

6.3 Сравнение эффективности работы алгоритмов распределения ресурсов ЦОД на реальных данных

6.3.1 Исходные данные

В качестве исходных данных были взяты данные, сгенерированные на основе реальных данных, предоставленных компанией Huawei. Топология ЦОД в рассматриваемых тестах

представляет собой связанные между собой кластеры из виртуальных машин. В каждом кластере есть только один коммутатор, к которому присоединены виртуальные машины, виртуальные машины не связаны друг с другом напрямую. Коммутаторы из каждого кластера соединяются между собой, связывая кластеры в единую топологию. Физические каналы между кластерами имеют примерно в 4 раза меньшую пропускную способность по сравнению с пропускной способностью каналов внутри кластера. Графы запросов представляют собой графы произвольной структуры. В тестах 1 и 2 присутствует много запросов, графы которых сильносвязны. Подробное описание исходных данных можно найти в Приложении Ж.

6.3.2 Результаты экспериментального исследования

На рис.6.8 показана утилизация ресурсов для Алгоритмов 1, 2, 3, запущенных на реальных тестах. График построен на основе табл. Ж1-Ж3.

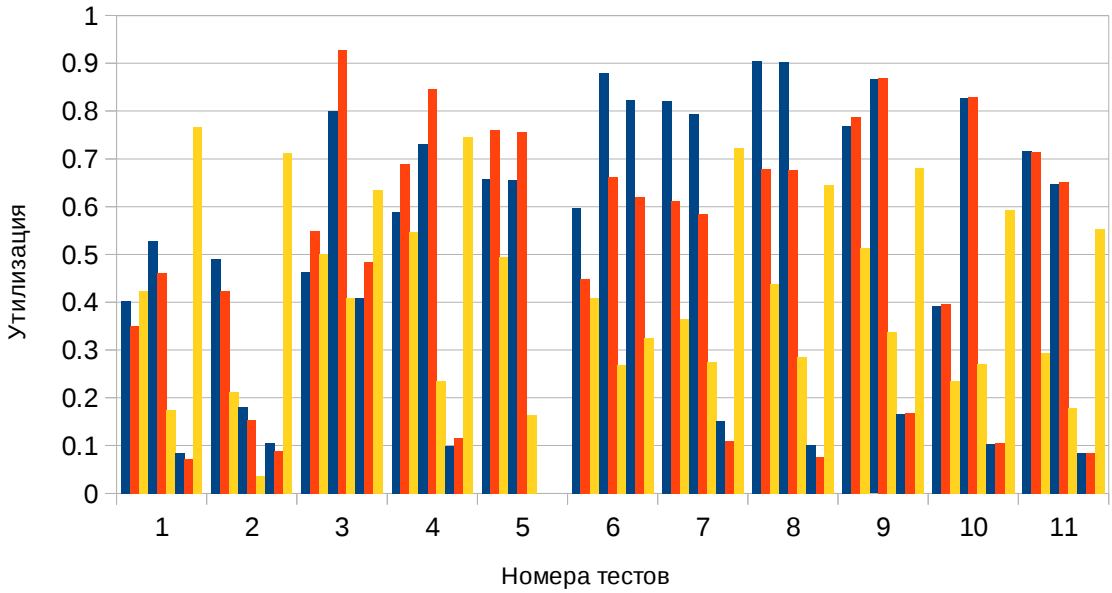

Рисунок 6.8. Утилизация ресурсов для Алгоритмов 1, 2, 3 при тестировании на реальных данных.

6.4 Выводы

Проведенное исследование показало, что качество получаемых решений разработанного алгоритма сравнимо с существующими алгоритмами. Также следует отметить, что разработанный алгоритм показал на реальных данных результаты лучше, чем

остальные алгоритмы. Проведенные эксперименты позволяют утверждать, что новый жадный критерий позволяет получать приемлемые результаты на более широком классе входных данных, чем существующий алгоритм [13]. Также разработанный алгоритм кроме построения отображения виртуальных элементов на физические, оценивает затраты на миграцию работающих виртуальных машин и строит план миграции для них в случае, если время миграции укладывается в директивный срок.

Заключение

1. Разработан алгоритм распределения ресурсов в ЦОД с построением плана миграции
2. Разработана процедура построения плана миграции с учетом требования SLA. Построенный план миграции удовлетворяет заданному ограничению на время выполнения.
3. Расширен набор жадных критериев, которые позволяют решать более эффективно по точности ряд новых задач.
4. Проведено сравнение разработанного алгоритма и существующих алгоритмов, показано, что разработанный алгоритм не только получает сравнимые с ними решения, но и оказался лучше существующих алгоритмов на реальных данных.

Список использованных источников

1. M. Armbrust; A. Griffith; A. D. Joseph; R. Katz; A. Konwinski; G. Lee; D. Patterson; A. Rabkin; I. Stoica; M, Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. —Berkeley, CA, USA, 2009 —25c. [Электронный ресурс]. URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf (дата обращения: 10.04.2016).
2. Data Center: Load Balancing Data Center Services SRND; — San Jose, CA, USA, 2004 — 94c. [Электронный ресурс]. URL: http://docslide.us/documents/data-center-load-balancing-data-center-services-srnd.html (дата обращения: 10.04.2016).
3. Md Rabbani. Resource Management in Virtualized Data Center. —Waterloo, Ontario, Canada, 2014 —72c. [Электронный ресурс]. URL: https://uwspace.uwaterloo.ca/bitstream/handle/10012/8280/Rabbani_Md.pdf?sequence=3 (дата обращения: 10.04.2016).
4. Clos C. A Study of Non-Blocking Switching Networks // Bell System Technical Journal, — March 1953, — P. 406-424
5. Виртуализация // Википедия. —2016, [Электронный ресурс]. URL: http://ru.wikipedia.org/?oldid=77889318 (дата обращения: 10.04.2016).
6. Bhanu P. Tholeti. Hypervisors, virtualization, and the cloud: Learn about hypervisors, system virtualization, and how it works in a cloud environment. © Copyright IBM Corporation 2011 — 8c. [Электронный ресурс]. URL: https://www.ibm.com/developerworks/cloud/library/cl-hypervisorcompare/ (дата обращения: 10.04.2016).
7. Rajkumar Buyya, Rajiv Ranjan, Rodrigo N. Calheiros. InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. — Melbourne, Australia —2010, 20c [Электронный ресурс]. URL: https://arxiv.org/ftp/arxiv/papers/1003/1003.3920.pdf (дата обращения: 11.04.2016).
8. Theophilus Benson, Aditya Akella Anees Shaikh, Sambit Sahu .CloudNaaS: A Cloud Networking Platform for Enterprise Applications.— Madison, WI,USA — 2011, 13c. [Электронный ресурс]. URL: http://pages.cs.wisc.edu/~tbenson/papers/CloudNaaS.pdf (дата обращения: 11.04.2016).
9. C. Guo, G. Lu, H.J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, Y. Zhang.Second-Net: A Data Center Network Virtualization Architecture with Bandwidth Guarantees. // Co-NEXT '10 Proceedings of the 6th International Conference , Article No. 15. New York, NY, USA, — 2010
10. S. Kolliopoulos, C. Stein. Improved approximation algorithms for unsplittable flow problems.

In Foundations of Computer Science, — 1997. —P.426 - 436

11. Kostenko V.A., Plakunov A.V. An Algorithm for Constructing Single Machine Schedules Based on Ant Colony Approach // J. of Computer and Systems Sciences Intern. 2013. V. 52. № 6. P. 928–937
12. Вдовин П. М., Костенко В. А. Алгоритм распределения ресурсов в центрах обработки данных с раздельными планировщиками для различных типов ресурсов // Известия Российской академии наук. Теория и системы управления. — 2014. — № 6. — С. 80–93.
13. Зотов И. А., Костенко В. А. Алгоритм распределения ресурсов в центрах обработки данных с единым планировщиком для различных типов ресурсов // Известия Российской академии наук. Теория и системы управления. — 2015. — № 1. — Р. 61–71.
14. R. Ricci, C. Alfeld, J. Lepreau: A Solver for the Network Testbed Mapping problem. //ACM SIGCOMM Computer Communication Review — 2003 — 33, — P. 65-81.
15. M. Chowdhury, M.R. Rahman, R. Boutaba. Vineyard: Virtual network embedding algorithms with coordinated node and link mapping. Networking.// IEEE/ACM Transactions on, — feb, 2012— P. 206 –219.
16. A. Ngenzi, Selvarani R, S. Nair, Dynamic Resource Management In Cloud Data Centers For Server Consolidation — Bangalore-India —2015 — 8c. [Электронный ресурс]. URL:https://arxiv.org/ftp/arxiv/papers/1505/1505.00577.pdf (дата обращения: 12.04.2016).
17. X. Meng, V. Pappas, Li Zhang: Improving the scalability of data center networks with traffic-aware virtual machine placement.// In INFOCOM, Proceedings IEEE,. — march 2010— P. 1 – 9.
18. Algorithms for Assigning Substrate Network Resources to Virtual Network Components / Yong Zhu and Mostafa Ammar — Atlanta, Georgia, USA, 2006 —12c. . [Электронный ресурс]. URL: http://www.cc.gatech.edu/fac/Mostafa.Ammar/papers/INFOCOM-YONG.pdf (дата обращения: 12.04.2016).
19. H. Ballani, P. Costa, T. Karagiannis, A. I. T. Rowstron. Towards predictable datacenter networks. // SIGCOMM’11, — Toronto, Ontario, Canada. 2011, — P.15–19 [Электронный ресурс]. URL:http://conferences.sigcomm.org/sigcomm/2011/papers/sigcomm/p242.pdf (дата обращения: 12.04.2016).
20. T. Lam, S. Radhakrishnan, A. Vahdat, and G. Varghese. NetShare: Virtualizing DataCenter Networks across Services, — San Diego,CA,USA, 2010
21. A. Shieh, S. Kandulaz, A. Greenberg, C. Kim, B. Saha. Sharing the Data Center Network.// USENIX'11 NSDI, — March 2011. [Электронный ресурс]. URL: https://www.usenix.org/legacy/ event/nsdi11/tech/full_papers/Shieh.pdf (дата обращения:

12.04.2016).

22. H. Rodrigues, J. R. Santos, Y. Turner, P. Soares, and D. Guedes. Gatekeeper: Supporting Bandwidth Guarantees for Multi-tenant Datacenter Networks. // Workshop on I/O Virtualization (WIOV'11), Portland, OR,USA, — 2011. [Электронный ресурс]. URL: https://pdfs.semanticscholar.org/a957/f95e7b95be34105d7ab5e6cfc2d17ce26e0c.pdf (дата обращения: 12.04.2016).
23. Jiangtao Zhang, Hejiao Huang, XuanWang. Resource provision algorithms in cloud computing: A survey. //Journal of Network and Computer Applications —2016, V.64 Issue C, —P. 23-42
24. A. Sallam, K. Li .A multi-objective virtual machine migration policy in cloud systems. // Comput Journal — 2014; 57(2) , —P. 195–204. [Электронный ресурс]. URL: http://comjnl.oxfordjournals.org/content/early/2013/02/28/comjnl.bxt018 (дата обращения: 13.04.2016).
25. Zhao M,Figueiredo RJ. Experimental study of virtual machine migration in support of reservation of cluster resources. //VTDC'07, 2nd international Workshop on Virtualization technology in distributed computing. — 2007, — p. 5. [Электронный ресурс]. URL: http://dl.acm.org/citation.cfm?id=1408659 (дата обращения: 13.04.2016).
26. Sotomayor B, Keahey K, FosterI. Combining batch execution and leasing using virtual machines. // HPDC'08, Boston, Massachusetts, USA, — June 2008, — p.87–96. [Электронный ресурс]. URL: http://www.nimbusproject.org/files/hpdc242s-sotomayor.pdf (дата обращения: 13.04.2016).
27. Yin F, Liu W, Song J. Live virtual machine migration with optimized three-stage memory copy./Future information technology. Springer; — Beijing, China, 2014, — P.69–75. [Электронный ресурс]. URL: http://www.hindawi.com/journals/jam/2014/297127/ (дата обращения: 13.04.2016).
28. Han P, Gong Q. Optimized live vm migration with eviction-free memory approach.// In: Proceedings of the 9th international symposiumon linear drives for industry applications, vol.1.Springer;2014. — P.185–191.
29. Ye K, Jiang X, Ma R, Yan F. Vc-migration: live migration of virtual clusters in the cloud.//ACM/IEEE 13th international conference on grid computing (GRID); 2012. — P;209–218. [Электронный ресурс]. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6319172 &filter%3DAND%28p_IS_Number%3A6319148%29%26pageNumber%3D2 (дата обращения: 13.04.2016).
30. Sarker T K, Tang M. Performance-driven live migration of multiple virtual machines in datacenters.// IEEE international conference on granular computing (GrC).IEEE;2013. —

P.253–258.

31. Walk the Line: Consistent Network Updates with Bandwidth Guarantees // HotSDN'12, — Helsinki, Finland, 2012, p.6. [Электронный ресурс]. URL: http://web.engr.illinois.edu/~caesar/papers/hot18-ghorbani.pdf (дата обращения: 13.43.2016).
32. B. G. Jozsa and M. Makai. On the solution of reroute sequence planning (RSP) problem in mpls networks.// Computer Networks, 42(2) , 2003 — P.199 – 210. [Электронный ресурс]. URL: http://www.cs.elte.hu/~marci/papers/reroute.pdf (дата обращения: 13.04.2016).
33. Алгоритм Форда-Фалкерсона нахождения максимального потока в транспортной сети // Википедия. —2016, [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Алгоритм_Форда_—_Фалкерсона (дата обращения: 15.04.2016).
34. Алгоритм поиска в ширину// Википедия. —2016, [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Поиск_в_ширину (дата обращения: 15.04.2016).
35. Алгоритм Дейкстры // Википедия. —2016, [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Алгоритм_Дейкстры (дата обращения: 15.04.2016).
36. Буч Г., Рамбо Д., Джекобсон А. Язык UML. Руководство пользователя. М.: ДМК Пресс, 2004. С. 432.

Приложение А. Сравнительная таблица алгоритмов распределения ресурсов в ЦОД

Таблица А1. Сравнения различных алгоритмов распределения ресурсов ЦОД

Алгоритм	Критерий планирования	Планирование VM или Storage	Планирование сетевых ресурсов	Миграция
Dynamic Resources Management in Cloud Data Centers for Server Consolidation [16]	Максимизировать количество размещенных запросов	+	-	-
Second-Net: A Data Center Network Virtualization Architecture with Bandwidth Guarantees [9]	Максимизация утилизации сетевых ресурсов	+	-	-
Improving the scalability of data center networks with traffic-aware virtual machine placement [17]	Минимизация коммуникаций в ЦОД между виртуальными машинами.	+	-	-
Vineyard: Virtual network embedding algorithms with coordinated node and link mapping [15]	Максимизировать количество размещенных запросов	+	+	-
Resource Management in Virtualized Data Center [3]	Максимизировать количество размещенных запросов	+	+	-
Algorithms for Assigning Substrate Network Resources to Virtual Network Components [18]	Минимизируется заполнение каналов и узлов, т.е. максимально распределяется запрос	+	+	-
CloudNaaS: A Cloud Networking Platform for Enterprise Applications [8]	Максимизировать количество размещенных запросов	+	+	-

Алгоритм назначения ресурсов в центрах обработки данных, основанный на схеме муравьиных колоний [11]	Максимизировать количество размещенных запросов	+	+	+
Алгоритм распределения ресурсов в центрах обработки данных с раздельными планировщиками для различных типов ресурсов [12]	Максимизировать количество размещенных запросов	+	+	+
Алгоритм распределения ресурсов в центрах обработки данных с единым планировщиками для различных типов ресурсов [13]	Максимизировать количество размещенных запросов. Построение компактного отображения.	+	+	+
*Towards predictable datacenter networks [19]	Максимизировать количество размещенных запросов	+	-	-
Solver for the Network Testbed Mapping problem [14]	Максимизировать количество размещенных запросов	+	-	+
NetShare[20] Seawall[21] Gatekeeper[22]	Гарантия пропускной способности	+	+	-

Приложение Б. Описание алгоритма на псевдокоде

Ниже представлено детальное описание на псевдокоде алгоритма распределения ресурсов в ЦОД с построением плана миграции. Все ссылки будут расшифрованы ниже, в разделах **Вспомогательные процедуры** и **Пояснения к алгоритму**.

Algorithm

Sort all Requests by their size[1] *in decreasing order*

for Each *Request* in the order **do**

    Consider *Nodes* from *Request*

    if not Assign *Nodes* **then**

        unassign all elements from *Request*

        goto to the end of loop

    end if

    Consider *Links* from *Request*

    if not Assign *Links* **then**

        unassign all elements from *Request*

        goto to the end of loop

    end if

end for

Nodes assignment

while *Nodes* is not empty yet **do**

    Push in deque *Queue* virtual node with the largest weight[2]

    while *Queue* is not empty yet **do**

        Pop *Node* from *Queue*

        Choose *Physical Node* with the smallest weight[3]

        if not Assign *Node* to *Physical Node* **then**

            if not Exhaustive search for *Node* and *Physical Nodes*[1] **then**

                return false

            end if

        end if

        Find all virtual channel from *Node* to adjacent unassigned nodes

        Sort virtual channel by their weight[4] in descending order

Add to the *Queue* virtual nodes in order of linked channel

delete *Node* from *Nodes*

end while

end while

return true

Links assignment

Sort virtual *Links* from *Request* by their weight in descending order

for each *Link* in the order **do**

if not find path for *Link* using BFS[II] **then**

return false

else

assign *Link*

end if

end for

Вспомогательные процедуры

(I) Процедура ограниченного перебора. Параметрами процедуры являются: depth – глубина перебора, MaxAttempt – максимальное число просматриваемых сочетаний физических элементов в количестве depth штук.

for attempt **in** MaxAttempt **do**

Get next cortege[5] with *depth* size

Save old pathes and old location for *Virtual Nodes* in cortege

Unassign *Virtual Nodes*

if Perform greedy assignment for *Node* and *Cortege* **then**

if Update Pathes for *Virtual Nodes* **then**

if Migration Plan[III] *Virtual Nodes* **then**

return true

end if

end if

end if

Unassign *Virtual Nodes*

Restore old location and pathes

return false

end for

(II) В алгоритме поиска в ширину рассматриваются только те физические каналы, на которых можно разместить соответствующий виртуальный канал.

(III) Описания алгоритма построения плана миграции можно посмотреть в разделе 4.5.

Пояснения к алгоритму

Параметры виртуального или физического элемента $e \rightarrow$ $(r_{e,1}, r_{e,2}, \ldots, r_{e,n})$

Дефицит по i-ому параметру $\rightarrow$ $$d(i) = \frac{\sum_{G} \sum_{e \in E, E \in G} r_{e,i}}{\sum_{ph \in Ph} r_{ph,i}}$$

Функция стоимости для эемента $e \rightarrow$ $$r(e) = \sum_{i=1..n} d(i) * \left(\frac{r_{e,i}}{maxr_{v,i}} \right)_{G, v \in G}$$

E – все виртуальные элементы запроса

G – граф запроса

e, v – конкретные виртуальные элемнеты запроса

Ph – все физический элементы в графе физических ресурсов

ph – конкретный физический элемент

(1) Размер запроса – это сумма количества узлов и ребер в графе запроса

(2) Вес виртуального элемента – $$weight(e) = r(e) * \sum_{l} Throughput(l)$$

l – каналы, инцидентные элементу e

(3) Вес физического элемента – $$weight(p) = r(p) * \sum_{l} Throughput(l) * C(p)$$

l – каналы, инцидентные элементу e

Throughput – доступная пропускная способность физического канала связи

$$C(p) = \sum_{e} r(e) * \left(\frac{\min(bw(p,e))}{d(p,e)} \right)$$

e – физический узел, до которого есть путь от элемента p

min(bw(e,p)) величина самого тонкого каналы связи в кратчайшем пути от узла *e* до *p*

d(p,e) – реберное расстояние в кратчайшем пути от узла e до узла p

(4) Вес для виртуального канала – это его пропускная способность

(5) *Cortege* – это подмножество физических ресурсов мощности *depth*.

Приложение В. Подробное описание результатов

В данном приложении приведены таблицы с результатами работы алгоритмов распределения ресурсов ЦОД. Алгоритм 1 – алгоритм распределения ресурсов с единым планировщиком, Алгоритм 2 – с единым планировщиком и наивным поиском в ширину при прокладке виртуальных каналов, Алгоритм 3 – разработанный в данной дипломной работе алгоритм. Утилизация физических ресурсов представляет собой число от 0 до 1 и является отношением суммарного количество занятых ресурсов к общему числу ресурсов заданнного типа. RAM измеряется в Гб, VCPUs – количество ядер, сетевые ресурсы – суммарная пропускная способность каналов связи, измеряется в Мб/с, время работы алгоритмов измеряется в секундах.

Таблица В1. Результаты работы Алгоритма 1 в случае топологии “толстого дерева” и запросов в форме дерева

Номера тестов	1	2	3	4	5	6
Общее количество запросов	20	20	20	20	200	10
Количество размещенных запросов	6	6	5	3	26	5
Утилизация физических ресурсов						
RAM	0,873	0,879	0,80	0,916	0,966	0,768
VCPUs	0,818	0,889	0,799	0,475	0,486	0,365
Сетевые ресурсы	0,463	0,674	0,609	0,298	0,351	0,448
Параметры запроса						
RAM	3075	2974	3023	2960	44239	3072
CPU	1478	1466	1533	1590	22253	1491
Сетевые ресурсы	2098	3472	2050	2125	31858	3758
Параметры ЦОД						
RAM	1024	1024	1024	512	8192	2048
CPU	512	512	512	512	8192	2048
Сетевые ресурсы	2964	6400	3200	6400	204800	38400
Время работы в секундах	1	8	6	1	7	4

Таблица В2. Результаты работы Алгоритма 2 в случае топологии “толстого дерева” и запросов в форме дерева

Номера тестов	1	2	3	4	5	6
Общее количество запросов	20	20	20	20	200	10
Количество размещенных запросов	6	6	5	3	26	2
Утилизация физических ресурсов						
RAM	0,873	0,911	0,814	0,916	0,966	0,321
VCPUs	0,818	0,877	0,758	0,475	0,486	0,145
Сетевые ресурсы	0,591	0,590	0,597	0,258	0,248	0,130
Параметры запроса						
RAM	3075	2974	3023	2960	44239	3072
CPU	1478	1466	1533	1590	22253	1491
Сетевые ресурсы	2098	3472	2050	2125	31858	3758
Параметры ЦОД						
RAM	1024	1024	1024	512	8192	2048
CPU	512	512	512	512	8192	2048
Сетевые ресурсы	2964	6400	3200	6400	204800	38400
Время работы в секундах	3	12	7	1	80	3

Таблица В3. Результаты работы Алгоритма 3 в случае топологии “толстого дерева” и запросов в форме дерева

Номера тестов	1	2	3	4	5	6
Общее количество запросов	20	20	20	20	200	10
Количество размещенных запросов	6	6	3	3	25	6
Утилизация физических ресурсов						
RAM	0,873	0,905	0,919	0,916	0,944	0,913
VCPUs	0,818	0,875	0,895	0,475	0,479	0,440
Сетевые ресурсы	0,494	0,501	0,534	0,161	0,148	0,283
Параметры запроса						
RAM	3075	2974	3023	2960	44239	3072
CPU	1478	1466	1533	1590	22253	1491
Сетевые ресурсы	2098	3472	2050	2125	31858	3758
Параметры ЦОД						
RAM	1024	1024	1024	512	8192	2048
CPU	512	512	512	512	8192	2048
Сетевые ресурсы	2964	6400	3200	6400	204800	38400
Время работы в секундах	1	3	2	1	6	3

Таблица В4. Результаты работы Алгоритма 1 в случае топологии “толстого дерева” и запросов в форме кольца

Номера тестов	1	2	3	4	5	6
Общее количество запросов	20	20	20	20	200	10
Количество размещенных запросов	6	6	6	3	27	9
Утилизация физических ресурсов						
RAM	0,863	0,878	0,903	0,852	0,965	0,876
VCPUs	0,855	0,953	0,775	0,393	0,486	0,438
Сетевые ресурсы	0,507	0,602	0,621	0,254	0,327	0,407
Параметры запроса						
RAM	2958	3020	3041	2956	43827	1993
CPU	1505	1490	1402	1503	22211	997
Сетевые ресурсы	2256	3743	2269	2266	33257	2475
Параметры ЦОД						
RAM	1024	1024	1024	512	8192	2048
CPU	512	512	512	512	8192	2048
Сетевые ресурсы	6400	6400	3200	6400	204800	38400
Время работы в секундах	1	1	4	1	9	1

Таблица В5. Результаты работы Алгоритма 2 в случае топологии “толстого дерева” и запросов в форме кольца

Номера тестов	1	2	3	4	5	6
Общее количество запросов	20	20	20	20	200	10
Количество размещенных запросов	6	5	6	3	27	7
Утилизация физических ресурсов						
RAM	0,863	0,720	0,906	0,852	0,965	0,692
VCPUs	0,855	0,800	0,832	0,393	0,486	0,343
Сетевые ресурсы	0,484	0,417	0,503	0,256	0,237	0,296
Параметры запроса						
RAM	2958	3020	3041	2956	43827	1993
CPU	1505	1490	1402	1503	22211	997
Сетевые ресурсы	2256	3743	2269	2266	33257	2475
Параметры ЦОД						
RAM	1024	1024	1024	512	8192	2048
CPU	512	512	512	512	8192	2048
Сетевые ресурсы	6400	6400	3200	6400	204800	38400
Время работы в секундах	2	1	1	2	80	2

Таблица В6. Результаты работы Алгоритма 3 в случае топологии “толстого дерева” и запросов в форме кольца

Номера тестов	1	2	3	4	5	6
Общее количество запросов	20	20	20	20	200	10
Количество размещенных запросов	6	6	6	3	26	9
Утилизация физических ресурсов						
RAM	0,863	0,863	0,905	0,852	0,940	0,872
VCPUs	0,855	0,967	0,805	0,393	0,476	0,434
Сетевые ресурсы	0,493	0,491	0,456	0,252	0,089	0,203
Параметры запроса						
RAM	2958	3020	3041	2956	43827	1993
CPU	1505	1490	1402	1503	22211	997
Сетевые ресурсы	2256	3743	2269	2266	33257	2475
Параметры ЦОД						
RAM	1024	1024	1024	512	8192	2048
CPU	512	512	512	512	8192	2048
Сетевые ресурсы	6400	6400	3200	6400	204800	38400
Время работы в секундах	1	1	3	1	5	1

Таблица В7. Результаты работы Алгоритма 1 в случае топологии “толстого дерева” и запросов в форме звезды

Номера тестов	1	2	3	4	5	6
Общее количество запросов	20	20	20	20	200	10
Количество размещенных запросов	5	1	1	3	12	0
Утилизация физических ресурсов						
RAM	0,874	0,155	0,136	0,936	0,235	0
VCPUs	0,789	0,145	0,129	0,518	0,122	0
Сетевые ресурсы	0,389	0,115	0,040	0,253	0,076	0
Параметры запроса						
RAM	3235	3222	3137	3110	45932	2028
CPU	1601	1557	1647	1654	23129	1040
Сетевые ресурсы	2248	3789	2219	2198	204800	2455
Параметры ЦОД						
RAM	1024	1024	1024	512	8192	2048
CPU	512	512	512	512	8192	2048
Сетевые ресурсы	6400	6400	3200	6400	204800	38400
Время работы в секундах	3	4	2	1	121	5

Таблица В8. Результаты работы Алгоритма 2 в случае топологии “толстого дерева” и запросов в форме звезды

Номера тестов	1	2	3	4	5	6
Общее количество запросов	20	20	20	20	200	10
Количество размещенных запросов	2	1	1	3	3	0
Утилизация физических ресурсов						
RAM	0,341	0,123	0,132	0,975	0,062	0
VCPUs	0,299	0,156	0,150	0,506	0,030	0
Сетевые ресурсы	0,098	0,032	0,034	0,251	0,011	0
Параметры запроса						
RAM	3235	3222	3137	3110	45932	2028
CPU	1601	1557	1647	1654	23129	1040
Сетевые ресурсы	2248	3789	2219	2198	204800	2455
Параметры ЦОД						
RAM	1024	1024	1024	512	8192	2048
CPU	512	512	512	512	8192	2048
Сетевые ресурсы	6400	6400	3200	6400	204800	38400
Время работы в секундах	1	1	1	1	935	2

Таблица В9. Результаты работы Алгоритма 3 в случае топологии “толстого дерева” и запросов в форме звезды

Номера тестов	1	2	3	4	5	6
Общее количество запросов	20	20	20	20	200	10
Количество размещенных запросов	5	1	1	3	19	0
Утилизация физических ресурсов						
RAM	0,817	0,155	0,136	0,936	0,373	0
VCPUs	0,818	0,145	0,129	0,518	0,190	0
Сетевые ресурсы	0,420	0,052	0,037	0,222	0,040	0
Параметры запроса						
RAM	3235	3222	3137	3110	45932	2028
CPU	1601	1557	1647	1654	23129	1040
Сетевые ресурсы	2248	3789	2219	2198	204800	2455
Параметры ЦОД						
RAM	1024	1024	1024	512	8192	2048
CPU	512	512	512	512	8192	2048
Сетевые ресурсы	6400	6400	3200	6400	204800	38400
Время работы в секундах	3	4	2	1	31	9

Таблица В10. Результаты работы Алгоритмов 1 в случае древовидной топологии и запросов в форме дерева

Номера тестов	1	2	3	4	5	6
Общее количество запросов	20	20	20	20	200	10
Количество размещенных запросов	6	6	3	3	5	1
Утилизация физических ресурсов						
RAM	0,816	0,862	0,419	0,818	0,191	0,149
VCPUs	0,926	0,881	0,447	0,449	0,093	0,084
Сетевые ресурсы	0,486	0,690	0,516	0,457	0,157	0,162
Параметры запроса						
RAM	2875	3025	3067	3005	59568	3091
CPU	1599	1505	1415	1504	5944	1502
Сетевые ресурсы	2110	3499	2117	2095	29970	3771
Параметры ЦОД						
RAM	1024	1024	1024	512	8192	2048
CPU	512	512	512	512	8192	2048
Сетевые ресурсы	4659	4358	1500	3000	51000	12600
Время работы в секундах	1	7	5	1	177	9

Таблица В11. Результаты работы Алгоритмов 2 в случае древовидной топологии и запросов в форме дерева

Номера тестов	1	2	3	4	5	6
Общее количество запросов	20	20	20	20	200	10
Количество размещенных запросов	6	6	6	3	1	0
Утилизация физических ресурсов						
RAM	0,816	0,834	0,908	0,818	0,037	0
VCPUs	0,926	0,861	0,861	0,449	0,017	0
Сетевые ресурсы	0,554	0,624	0,648	0,514	0,022	0
Параметры запроса						
RAM	2875	3025	3067	3005	59568	3091
CPU	1599	1505	1415	1504	5944	1502
Сетевые ресурсы	2110	3499	2117	2095	29970	3771
Параметры ЦОД						
RAM	1024	1024	1024	512	8192	2048
CPU	512	512	512	512	8192	2048
Сетевые ресурсы	4659	4358	1500	3000	51000	12600
Время работы в секундах	3	10	7	1	1020	4

Таблица В12. Результаты работы Алгоритмов 3 в случае древовидной топологии и запросов в форме дерева

Номера тестов	1	2	3	4	5	6
Общее количество запросов	20	20	20	20	200	10
Количество размещенных запросов	6	6	5	3	24	3
Утилизация физических ресурсов						
RAM	0,816	0,817	0,718	0,818	0,922	0,459
VCPUs	0,926	0,865	0,723	0,450	0,453	0,219
Сетевые ресурсы	0,516	0,510	0,595	0,526	0,433	0,348
Параметры запроса						
RAM	2875	3025	3067	3005	59568	3091
CPU	1599	1505	1415	1504	5944	1502
Сетевые ресурсы	2110	3499	2117	2095	29970	3771
Параметры ЦОД						
RAM	1024	1024	1024	512	8192	2048
CPU	512	512	512	512	8192	2048
Сетевые ресурсы	4659	4358	1500	3000	51000	12600
Время работы в секундах	1	5	3	1	42	3

Таблица В13. Результаты работы Алгоритмов 1 в случае древовидной топологии и запросов в форме кольца

Номера тестов	1	2	3	4	5	6
Общее количество запросов	20	20	20	20	200	10
Количество размещенных запросов	6	5	3	3	3	2
Утилизация физических ресурсов						
RAM	0,891	0,737	0,429	0,953	0,102	0,174
VCPUs	0,890	0,754	0,416	0,482	0,053	0,106
Сетевые ресурсы	0,564	0,677	0,512	0,513	0,097	0,200
Параметры запроса						
RAM	3051	3083	3026	3228	44245	2013
CPU	1496	1518	1499	1533	22264	1058
Сетевые ресурсы	2301	3740	2265	2227	33353	2486
Параметры ЦОД						
RAM	1024	1024	1024	512	8192	2048
CPU	512	512	118000	512	8192	2048
Сетевые ресурсы	4585	4464	512	3000	51000	12600
Время работы в секундах	3	8	5	1	191	5

Таблица В14. Результаты работы Алгоритмов 2 в случае древовидной топологии и запросов в форме кольца

Номера тестов	1	2	3	4	5	6
Общее количество запросов	20	20	20	20	200	10
Количество размещенных запросов	6	6	6	2	1	1
Утилизация физических ресурсов						
RAM	0,853	0,911	0,883	0,688	0,032	0,106
VCPUs	0,885	0,895	0,869	0,324	0,0177	0,055
Сетевые ресурсы	0,606	0,631	0,653	0,379	0,018	0,074
Параметры запроса						
RAM	3051	3083	3026	3228	44245	2013
CPU	1496	1518	1499	1533	22264	1058
Сетевые ресурсы	2301	3740	2265	2227	33353	2486
Параметры ЦОД						
RAM	1024	1024	1024	512	8192	2048
CPU	512	512	118000	512	8192	2048
Сетевые ресурсы	4585	4464	512	3000	51000	12600
Время работы в секундах	6	9	7	1	980	2

Таблица В15. Результаты работы Алгоритмов 3 в случае древовидной топологии и запросов в форме кольца

Номера тестов	1	2	3	4	5	6
Общее количество запросов	20	20	20	20	200	10
Количество размещенных запросов	6	6	6	3	26	7
Утилизация физических ресурсов						
RAM	0,853	0,911	0,874	0,953	0,924	0,654
VCPUs	0,885	0,895	0,863	0,482	0,462	0,365
Сетевые ресурсы	0,545	0,505	0,460	0,500	0,339	0,303
Параметры запроса						
RAM	3051	3083	3026	3228	44245	2013
CPU	1496	1518	1499	1533	22264	1058
Сетевые ресурсы	2301	3740	2265	2227	33353	2486
Параметры ЦОД						
RAM	1024	1024	1024	512	8192	2048
CPU	512	512	118000	512	8192	2048
Сетевые ресурсы	4585	4464	512	3000	51000	12600
Время работы в секундах	3	1	3	1	22	1

Таблица В16. Результаты работы Алгоритмов 1 в случае древовидной топологии и запросов в форме звезды

Номера тестов	1	2	3	4	5	6
Общее количество запросов	20	20	20	20	200	10
Количество размещенных запросов	6	5	0	3	4	0
Утилизация физических ресурсов						
RAM	0,938	0,792	0	0,914	0,091	0
VCPUs	0,939	0,795	0	0,496	0,044	0
Сетевые ресурсы	0,590	0,712	0	0,527	0,073	0
Параметры запроса						
RAM	3120	3225	3173	3124	46336	2169
CPU	1599	1568	1578	1588	23122	1023
Сетевые ресурсы	2196	3820	2262	2219	33355	2469
Параметры ЦОД						
RAM	1024	1024	1024	512	8192	2048
CPU	512	512	512	512	8192	2048
Сетевые ресурсы	4235	4578	1500	3000	51000	12600
Время работы в секундах	5	8	3	1	48	5

Таблица В17. Результаты работы Алгоритмов 2 в случае древовидной топологии и запросов в форме звезды

Номера тестов	1	2	3	4	5	6
Общее количество запросов	20	20	20	20	200	10
Количество размещенных запросов	3	1	0	2	1	0
Утилизация физических ресурсов						
RAM	0,461	0,137	0	0,627	0,022	0
VCPUs	0,451	0,141	0	0,316	0,012	0
Сетевые ресурсы	0,181	0,063	0	0,351	0,015	0
Параметры запроса						
RAM	3120	3225	3173	3124	46336	2169
CPU	1599	1568	1578	1588	23122	1023
Сетевые ресурсы	2196	3820	2262	2219	33355	2469
Параметры ЦОД						
RAM	1024	1024	1024	512	8192	2048
CPU	512	512	512	512	8192	2048
Сетевые ресурсы	4235	4578	1500	3000	51000	12600
Время работы в секундах	1	1	1	1	1000	2

Таблица В18. Результаты работы Алгоритмов 3 в случае древовидной топологии и запросов в форме звезды

Номера тестов	1	2	3	4	5	6
Общее количество запросов	20	20	20	20	200	10
Количество размещенных запросов	6	5	0	3	9	0
Утилизация физических ресурсов						
RAM	0,922	0,774	0	0,934	0,183	0
VCPUs	0,938	0,697	0	0,5	0,086	0
Сетевые ресурсы	0,575	0,616	0	0,499	0,073	0
Параметры запроса						
RAM	3120	3225	3173	3124	46336	2169
CPU	1599	1568	1578	1588	23122	1023
Сетевые ресурсы	2196	3820	2262	2219	33355	2469
Параметры ЦОД						
RAM	1024	1024	1024	512	8192	2048
CPU	512	512	512	512	8192	2048
Сетевые ресурсы	4235	4578	1500	3000	51000	12600
Время работы в секундах	1	6	2	1	41	1

Приложение Г. Результаты работы процедуры построения плана миграции при различной начальной загруженности ЦОД

Таблица Г.1.

	I				II			
	III,III	III,IV	IV,III	IV,IV	III,III	III,IV	IV,III	IV,IV
VM	23	35	35	38	29	29	28	36
migrate all	9	16	17	19	16	12	15	21
migrate light	8	12	15	18	13	9	13	20
migrate heavy	1	4	2	1	3	3	2	1
З.ЦОД[1]	0.500	0.500	0.500	0.500	0.785	0.785	0.710	0.857
ИМ[2]	0.391	0.457	0.485	0.500	0.551	0.413	0.535	0.583
ИМЛ[3]	0.666	0.428	0.535	0.5625	0.65	0.409	0.464	0.555
ИМТ[4]	0.250	0.571	0.285	0.166	0.333	0.428	0.285	0.200
ДТМ[5]	0.111	0.250	0.133	0.055	0.187	0.250	0.133	0.047
ДЛМ[6]	0.888	0.750	0.866	0.944	0.812	0.750	0.866	0.952

(1)З.ЦОД - загрузка ЦОД; вычисляется как отношение занятых физических ресурсов к общему числу физических ресурсов в ЦОД.

(2)ИМ - интенсивность миграции; вычисляется как отношение числа мигрирующих VM к общему числу VM в ЦОД.

(3)ИМЛ - интенсивность миграции легковесных VM; вычисляется как отношение числа легковесных мигрирующих VM к общему числу легковесных VM в ЦОД.

(4)ИМТ – интенсивность миграции тяжеловесных VM; вычисляется как отношение числа тяжеловесных мигрирующих VM к общему числу тяжеловесных VM в ЦОД.

(5)ДТМ – отношение количества тяжеловесных мигрирующих VM в общему числу мигрирующих VM.

(6)ДЛМ – отношение количества легковесных мигрирующих VM в общему числу мигрирующих VM.

Пара (римская цифра№1, римская цифра№2) означает, что ЦОД находится в ситуации

соответствующей римской цифре №1, а вновь пришедшие запросы соответствуют римской цифре №2.

Таблица Г.2.

Тесты	Размещенные в ЦОД запросы		Поступившие в ЦОД запросы	
	heavy	light	heavy	light
I,III,III	4	12	7	0
I,III,IV	4	12	3	16
I,IV,III	3	16	4	12
I,IV,IV	3	16	3	16
II,III,III	6	20	3	0
II,III,IV	6	20	1	2
II,IV,III	4	24	3	4
II,IV,IV	4	32	1	4

Приложение Д. Результаты работы процедуры построения плана миграции при различном выборе пропускной способности канала миграции

Способ №1:

$$tun.throughput = throughput * \left(\frac{V_i * (A_i + 1)}{\sum_{j=1}^{k} V_j * (A_j + 1)} \right) \text{(Д.1)}$$

Способ №2:

$$tun.throughput = throughput \text{ (Д.2)}$$

Время миграции измеряется в секундах

Таблица.Д.1. Время, затраченное на миграцию виртуальных машин в зависимости от выбора пропускной способности виртуального канала для миграции

Тесты	Способ №1	Способ №2
I,III,III	0	0
I,III,IV	225	116
I,IV,III	202	92
I,IV,IV	202	90
II,III,III	81	112
II,III,IV	202	86
II,IV,III	202	68
II,IV,IV	202	68

Приложение Е. Топология для проведения экспериментов с исследованием процедуры построения плана миграции

Топология ЦОД для проведения экспериментов

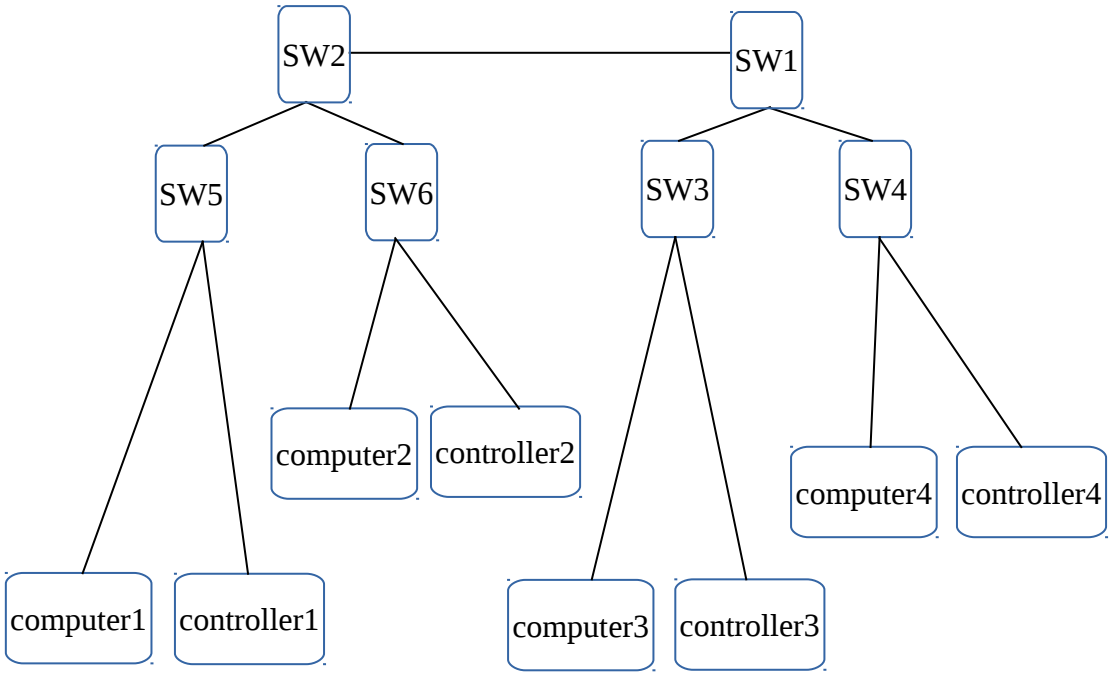

Таблица Е.1.Характеристики вычислительных узлов

Compute Nodes	CPU's	RAM
Computer1	80	8000
Computer2	60	10000
Computer3	80	8000
Computer4	60	10000

Приложение Ж. Результаты работы алгоритмов распределения ресурсов в ЦОД на реальных данных

В приведенных таблицах представлены результаты экспериментального исследования на реальных данных алгоритмов распределения ресурсов в ЦОД: алгоритма с единым планировщиком и наивным поиском в ширину при прокладке виртуальных каналов(Алгоритм 1), разработанного в данной дипломной работе алгоритма(Алгоритм 2) и алгоритма с единым планировщиком (Алгоритм 3). Время работы агоритм измеряется в часах(hours), минутах(mins) и секундах(secs).

Таблица Ж1. Результаты работы Алгоритма 1 на реальных данных.

№ теста	1	2	3	4	5	6	7	8	9	10	11
Число запросов	240	520	120	520	1100	240	520	1100	120	520	1100
Размещенные запросы	87	314	40	257	611	106	392	802	75	210	740
Утилизация ресурсов											
RAM	0,40	0,49	0,464	0,59	0,66	0,6	0,82	0,91	0,77	0,4	0,72
VCPUs	0,35	0,42	0,549	0,69	0,76	0,45	0,61	0,68	0,79	0,4	0,72
Сетевые ресурсы	0,42	0,21	0,5	0,55	0,5	0,4	0,37	0,44	0,52	0,24	0,3
Параметры запроса											
RAM	25156	54862	13156	55736	116627	25935	57083	114198	12847	54639	116922
CPU	12610	27416	6607	27874	58258	12940	28661	57020	6466	27319	58499
Сетевые ресурсы	9806	21754	5592	22549	46068	10372	23100	43732	5374	21643	46455
Параметры ЦОД											
RAM	21073	52317	10537	52236	104403	18074	44996	89850	9067	44963	90101
CPU	12046	29997	4528	22444	45146	12024	30179	59941	4483	22411	45121
Сетевые ресурсы	19581	54677	14699	69821	146019	24839	69893	144761	11748	55806	114836
Время работы	10 mins 55 secs	16 mins 25 secs	5 mins 4 secs	14 mins 47 secs	1 hours 34 mins 39 escs	10 min 14 secs	44 mins 10 secs	4 hours 18 mins 15 secs	3 mins 24 secs	55 mins 42 secs	4 hours 12 mins 18 secs

Таблица Ж2. Результаты работы Алгоритма 2 на реальных данных.

№ теста	1	2	3	4	5	6	7	8	9	10	11
Число запросов	240	520	120	520	1100	240	520	1100	120	520	1100
Размещенные запросы	137	115	86	413	794	176	350	817	87	412	681
Утилизация ресурсов											
RAM	0,529	0,182	0,801	0,730	0,657	0,881	0,794	0,904	0,866	0,826	0,648
VCPUs	0,462	0,153	0,929	0,846	0,757	0,662	0,585	0,677	0,868	0,829	0,651
Сетевые ресурсы	0,174	0,037	0,408	0,235	0,165	0,269	0,275	0,285	0,337	0,272	0,180
Параметры запроса											
RAM	25156	54862	13156	55736	116627	25935	57083	114198	12847	54639	116922
CPU	12610	27416	6607	27874	58258	12940	28661	57020	6466	27319	58499
Сетевые ресурсы	9806	21754	5592	22549	46068	10372	23100	43732	5374	21643	46455
Параметры ЦОД											
RAM	21073	52317	10537	52236	104403	18074	44996	89850	9067	44963	90101
CPU	12046	29997	4528	22444	45146	12024	30179	59941	4483	22411	45121
Сетевые ресурсы	19581	54677	14699	69821	146019	24839	69893	144761	11748	55806	114836
Время работы	3 mins 15 secs	37 mins 21 secs	37 secs	36 mins 20 secs	6 hours 32 mins	2 mins 17 secs	39 mins 42 secs	4 hours 8 mins 36 secs	1 mins 1 secs	35 mins 30 secs	3 hours 24 mins 44 secs

Таблица Ж3. Результаты работы Алгоритма 3 на реальных данных.

№ теста	1	2	3	4	5	6	7	8	9	10	11
Общее количество запросов	240	520	120	520	1100	240	520	1100	120	520	1100
Количество размещенных запросов	21	70	34	56		167	72	109	17	60	98
Утилизация физических ресурсов											
RAM	0,084	0,105	0,409	0,100		0,822	0,150	0,101	0,166	0,103	0,084
VCPUs	0,073	0,089	0,484	0,117		0,620	0,110	0,075	0,169	0,105	0,084
Сетевые ресурсы	0,766	0,713	0,636	0,746		0,325	0,722	0,645	0,682	0,593	0,554
Параметры запроса											
RAM	25156	54862	13156	55736	116627	25935	57083	114198	12847	54639	116922
CPU	12610	27416	6607	27874	58258	12940	28661	57020	6466	27319	58499
Сетевые ресурсы	9806	21754	5592	22549	46068	10372	23100	43732	5374	21643	46455
Параметры ЦОД											
RAM	21073	52317	10537	52236	104403	18074	44996	89850	9067	44963	90101
CPU	12046	29997	4528	22444	45146	12024	30179	59941	4483	22411	45121
Сетевые ресурсы	19581	54677	14699	69821	146019	24839	69893	144761	11748	55806	114836
Время работы	3 mins 36 secs	13 mins 4 secs	4 mins 30 secs	35 mins 45 secs	Не завершиился за 15 часов	5 mins 43 secs	47 mins 42 secs	6 hours 12 mins 28 secs	2 mins 12 secs	15 mins 16 secs	47 mins 36 secs