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

Чупахин Андрей Андреевич

Планирование вычислений в центре обработки данных с учетом политик размещения виртуальных ресурсов

КУРСОВАЯ РАБОТА

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

Москва, 2017

Аннотация

В рамках курсовой работы разработан и реализован алгоритм распределения ресурсов в центрах обработки данных с учетом политик размещения виртуальных ресурсов.

Данный алгоритм является модификацией уже существующего алгоритма распределения ресурсов в центрах обработки данных с раздельными планировщиками для различных типов ресурсов. Модифицированный алгоритм получает отображение запросов на физические ресурсы центра обработки данных, учитывая тот факт, что некоторые физические сервера в центре обработки данных могут иметь архитектуру с неоднородным доступом к памяти. Кроме этого разработанный алгоритм при размещении запросов в центре обработки данных учитывает политики размещения виртуальных ресурсов, которые задаются инициатором запроса.

Содержание

Введение

По мере развития облачных технологий [1-4] остро встал вопрос о построении систем управления ресурсами центра обработки данных (ЦОД). ЦОД состоит из совокупности вычислительных ресурсов, систем хранения данных, физических каналов связи и коммутационного оборудования. Всеми этими ресурсами нужно эффективно управлять, поэтому одной из главных задач системы управления ресурсами ЦОД должно быть повышение эффективности использования ресурсов. Для решения этой задачи в таких системах используются различные алгоритмы распределения ресурсов.

Одним из важнейших свойств таких алгоритмов является возможность построения такого распределения, при котором загрузка ресурсов в ЦОД была бы достаточно высокой, 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 в зависимости от предоставляемых ресурсов: IaaS (Infrastructure as a Service), PaaS (Platform as a Service), SaaS (Software as a Service). В данной 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, объем жесткого диска у виртуальной машины, объем памяти у виртуальной системы хранения данных, 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, 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. В данной работе рассматриваются только статические SLA: параметры виртуальной машины, параметры хранилища данных, пропускная способность виртуального канала связи. SLA, связанные с контролем задержки в канале связи, нагрузка на коммутационное оборудование, не рассматриваются.

В реальной жизни может возникнуть необходимость влиять на размещение виртуальных машин, т.е задавать политики размещения. В данной работе рассматриваются следующие возможности:

1. Каждой виртуальной машине можно поставить в соответствие набор физических серверов, на одном из которых она должна быть размещена – VM-to-PM affinity,
2. Каждой виртуальной машине можно поставить в соответствие набор физических серверов, на которых она не может быть размещена – VM-to-PM anti-affinity,
3. Можно задать набор виртуальных машин, который должен быть размещен на одном физическом сервере – VM-to-VM affinity,
4. Можно задать набор виртуальных машин, который должен быть размещен на разных физических серверах – VM-to-VM anti-affinity.

Физические сервера, на которых размещаются виртуальные машины клиентов, имеют неоднородную структуру. В серверах присутствует более одной системной шины. Каждая системная шина обслуживает небольшое количество процессоров. Каждая группа процессоров имеет свою собственную память, а также может иметь свои собственные каналы ввода-вывода. Кроме этого каждый процессор имеет доступ к памяти, которая связана с другими группами. Каждая такая группа называется NUMA(Non-Uniform Memory Access)-узлом или NUMA-блоком. Количество процессоров в NUMA-блоке может быть разным в зависимости от оборудования. Но самое главное, что доступ процессора к локальной памяти NUMA-блока происходит быстрее, чем к памяти другого NUMA-блока. Поэтому такая структура памяти

получила название — архитектура неоднородного доступа к памяти. Более подробно данная архитектура будет рассмотрена в главе 2.1 «Основные понятия и термины».

Алгоритм планирования ресурсов ЦОД должен учитывать особенности этой архитектуры. В данной курсовой работе мы рассматриваем физические сервера, у которых задано число NUMA-блоков и их параметры. Планирование использования физических ресурсов проводиться с учетом такой архитектуры серверов.

Большинство существующих алгоритмов планирования ресурсов в ЦОД не позволяют учитывать политики размещения и архитектуру серверов, в частности NUMA-архитектуру. Если не учитывать NUMA-архитектуру, это приводит к снижению производительности виртуальных машин [26].

В данной работе предлагается алгоритм, который устраняет данные недостатки. Его характерными особенностями являются: возможность планировать все типы ресурсов (виртуальные машины, виртуальные хранилища данных, виртуальные каналы связи), возможность гибкой настройки алгоритма для достижения баланса между скоростью работы и качеством получаемого решения. Также предложенный алгоритм поддерживает возможность задания политик на размещение виртуальных машин (расположение группы виртуальных машин на одном сервере или на разных серверах, запрет или четкое указание набора физических узлов для размещения) и поддерживает работу с серверами, которые имеют NUMA-архитектуру.

1 Цели курсовой работы

Целями курсовой работы является разработка алгоритма распределения ресурсов в ЦОД с учетом политик размещения виртуальных ресурсов и учетом NUMA-архитектуры физических серверов.

Для достижения указанной цели должны быть решены следующие задачи:

1. Провести анализ возможности расширения существующих алгоритмов распределения ресурсов в ЦОД для учета политик размещения виртуальных ресурсов и NUMA-архитектуры серверов.
2. Разработать алгоритм распределения ресурсов ЦОД, который должен обладать следующими особенностями:
   a) вычислительный ресурсы, системы хранения данных, сетевые ресурсы являются планируемыми ресурсами,
   b) возможность указывать правила размещения виртуальных машин,
   c) возможность эффективного планирования ресурсов для серверов, имеющих NUMA-архитектуру.
3. Провести исследование свойств разработанного алгоритма

2 Обзор предметной области

Целью написания данной главы является анализ существующих алгоритмов планирования, которые строят отображение запросов на физические ресурсы с учетом политик размещения виртуальных ресурсов и с учетом NUMA-архитектуры серверов ЦОД. С облачными вычислениями [25] связано много задач от чисто математических, алгоритмических, до сугубо прикладных. В данной главе рассматривается только задача распределения ресурсов в центрах обработки данных и задача размещения виртуальных машин на сервере с NUMA-архитектурой.

2.1 Основные понятия и 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 [5]:

1. Традиционная топология ЦОД (Conventional data center network topology [6]). Это трехуровневая древовидная топология, которая состоит из:
   a) Уровень ядра (Core Layer) — корневой уровень, который содержит корневые маршрутизаторы, имеющие доступ в интернет,
   b) Уровень агрегации (Agregation Layer) — уровень агрегации, который состоит из коммутаторов, подключенных к одному или нескольким маршрутизаторам уровня ядра.
   c) Уровень доступа (Access Layer) — уровень доступа, состоящий из серверов и систем хранения данных.

2. Многоярусная сеть Клоза (Clos topology [7]).

Иерархическая система из коммутаторов, которую можно построить таким образом, чтобы в ней отсутствовали 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.

3. Топология «толстое» дерево (Fat-tree topology).

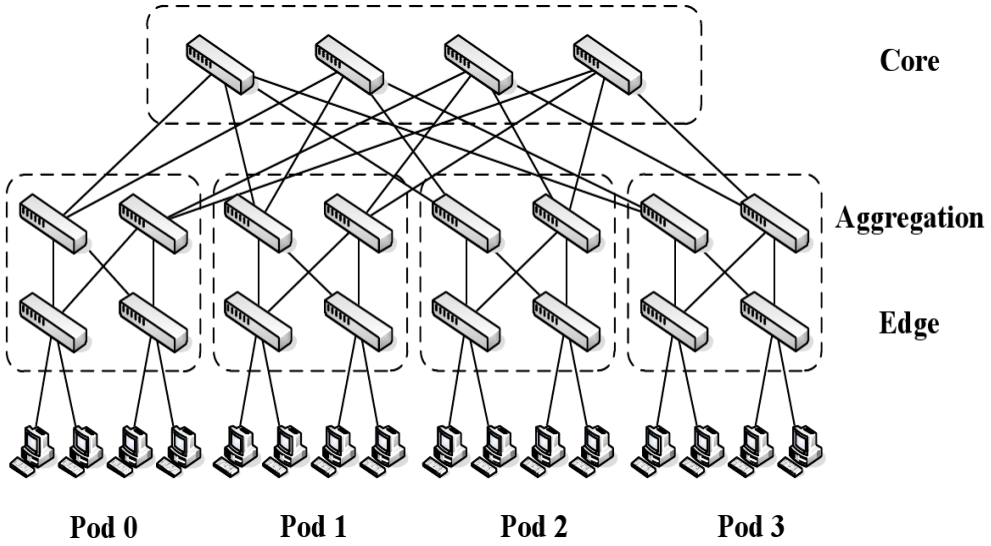

Рисунок 2.1

Древовидная топология. Это специальный вариант сети Клоза. Данная топология состоит из трех уровней: $\left(\frac{K}{2}\right)^2$ маршрутизаторов уровня ядра, K^2 коммутаторов уровня агрегации и $K\left(\frac{K}{2}\right)^2$ серверов. Средний уровень состоит из K-портовых коммутаторов, которые объединены в K групп. В каждой группе $\frac{K}{2}$ коммутаторов являются aggregation-коммутаторы, которые связаны с верхним уровнем, другие $\frac{K}{2}$ коммутаторов – edge-коммутаторы, т.е. граничные, каждый из которые соединен с $\frac{K}{2}$ серверами. Верхний уровень, состоящий из core-коммутаторов, каждый i-ый порт которого связан с i-ой группой коммутаторов среднего уровня.

Древовидная топология и топология «толстого» дерева часто применяется при проектировании сети в ЦОД, поэтому разработчики алгоритмов распределения ресурсов в ЦОД проверяют эффективность своих алгоритмов именно на этих топологиях.

Виртуализация

Виртуализация [8] — 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

Облако — это ЦОД и облачная платформа, которая управляет ресурсами ЦОД. Облака могут быть могут быть федеративными или централизованными [9].

Облачная платформа

Программная компонента ЦОД, которая отвечает за принятие, обработку и исполнение запросов. Основные модули — это оркестратор ЦОД и планировщик ресурсов ЦОД. Также могут существовать и другие модули [10] — например, модуль мониторинга состояния виртуальных элементов и физических элементов.

Оркестратор

Оркестратор – это часть облачной платформы, которая занимается приемом 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 определить, как отобразить эти запросы на физическую топологию ЦОД. Чтобы сделать это, ему нужен планировщик ресурсов. Именно планировщик 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.

Планировщик ресурсов ЦОД

Инициатор запроса в ЦОД на получение некоторого набора ресурсов задет некоторые характеристики элементов запроса. Построение отображения запросов, которое удовлетворяет этим характеристикам, осуществляет планировщик ресурсов ЦОД. Входные данные о поступивших запросах, работающих виртуальных ресурсах и состоянии физических ресурсов ЦОД планировщик получает от оркестратора.

Архитектура с однородным доступом к памяти

UMA(Uniform Memory Access)-мультипроцессор [26, Глава 8.1] — это мультипроцессор с общей памятью, у которого время доступа к любому слову памяти одинаково. Простейшие UMA-мультипроцессоры основаны на использовании общей шины. В этом случае два и более процессора используют одну и туже шину для доступа к памяти. Отсюда сразу следует ограничение: процессоры одновременно не могут обращаться к памяти. Это ограничение влияет на масштабируемость системы: при большом количестве процессоров возникнет большое количество коллизий при обращении к памяти.

Существуют более продвинутые версии UMA-мультипроцессоров: использующие координатные коммутаторы или многоступенчатые схемы коммутации.

UMA-мультипроцессоры на координатных коммутаторах нельзя сделать очень большими, так как размер координатного коммутатора квадратично зависит от количества процессоров.

UMA-мультипроцессоры, основанные на многоступенчатой коммутации, используют $\frac{N}{2}\log(N)$ коммутаторов с двумя входами и выходами. Но в полученной многоступенчатой коммутационной сети могут возникать блокировки.

Архитектура с неоднородным доступом к памяти

Число процессоров для UMA-мультипроцессора с общей шиной ограничено несколькими десятками, мультипроцессоры с координатной и многоступенчатой коммутацией нуждаются в большом количестве дорогостоящих коммутаторов, а также количество процессоров в них ненамного больше. Чтобы использовать более сотни процессоров была предложена новая архитектура — NUMA(Non-Uniform Memory Access) [26]. NUMA-мультипроцессоры лучше

масштабируется, благодаря снижению потенциальных параллельных обращений к памяти путем группирования процессоров и выделения для каждой группы своей локальной памяти.

NUMA-мультипроцессор — это мультипроцессор с общей памятью, у которого доступ к локальным модулям памяти производится быстрее, чем к удаленным модулям. Все программы, написанные для UMA-архитектуры будут работать без изменений на NUMA-архитектуре, но производительность их будет хуже, чем на UMA-мультипроцессорах. Более подробно проблема производительности будет рассмотрена в главе 2.3 «Архитектура с неоднородным доступом к памяти».

2.2 Содержательная постановка задачи распределения ресурсов в центрах обработки данных с учетом политик размещения виртуальных ресурсов

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 [27]. Задача распределения ресурсов в ЦОД является NP-трудной задачей, так как к ней сводится задача об упаковке в контейнеры. В статье [11] NP-трудность задачи распределения ресурсов доказывается сведением к ней задачи о неделимом потоке с одним источником (single-source unsplittable flow [12]). Так как размещение запросов это NP-трудная задача, для ее решения в первую очередь используются различные эвристические алгоритмы: генетический, муравьиный алгоритм [13]. Также применяются жадные алгоритмы [13,14,28], алгоритм имитации отжига [15].

В данной работе рассматривается следующая постановка задачи распределения ресурсов ЦОД. Из вновь поступивших запросов нужно разместить максимальное количество. При этом учитывается возможность указания политик на размещение виртуальных машин:

1. Каждой виртуальной машине можно поставить в соответствие набор физических серверов, на одном из которых она должна быть размещена – VM-to-PM affinity,
2. Каждой виртуальной машине можно поставить в соответствие набор физических серверов, на которых она не может быть размещена – VM-to-PM anti-affinity,
3. Можно задать набор виртуальных машин, который должен быть размещен на одном физическом сервере – VM-to-VM affinity,
4. Можно задать набор виртуальных машин, который должен быть размещен на разных физических серверах – VM-to-VM anti-affinity.

Политики задаются для каждого запроса независимо. Корректность политик размещения не проверяется. Под корректностью понимается непротиворечивость политик, например, для одной и той же машины не должно существовать сервера, на котором она обязана и не обязана разместиться вследствие существования двух противоречивых политик.

При размещении виртуальных машин на серверах ЦОД учитывается наличие серверов с NUMA-архитектурой. Если не учитывать NUMA-архитектуру серверов, производительность машин снизится.

2.3 Архитектура с неоднородным доступом к памяти

В главе 2.1 было введено понятие архитектуры с неоднородным доступом к памяти, или NUMA-архитектуры. Особенностью NUMA-архитектуры является разделение памяти на локальную и удаленную для каждого процессора. Время доступа к удаленной памяти больше, чем время доступа к локальной памяти.

UMA-архитектура, описанная в **главе 2.1**, имеет существенное ограничение на количество процессоров; только несколько десятков. Это ограничение возникает из-за того, что все процессоры, а также оперативная память, подключены к одной шине. При большом количестве процессоров шина становится «бутылочным» горлышком системы. Чтобы обойти это ограничение была предложена NUMA-архитектура.

NUMA-мультипроцессор состоит из двух и более NUMA-блоков. NUMA-блоки соединены между собой при помощи каналов связи. Каждый NUMA-блок в свою очередь состоит из одного и более процессоров. Все процессоры в NUMA-блоки подключены к одной локальной системной шине, к этой же шине подключена и локальная оперативная память.

Процессоры могут обращаться как к своей локальной памяти через быструю локальную шину, так и к удаленной памяти через более медленную, чем локальная шина, коммуникационную среду. Таким образом, неоднородность доступа к памяти — это следствие разбиения процессоров по группам. Положительной особенностью данной архитектуры является возможность иметь в составе NUMA-мультипроцессора более сотни процессоров. Но также важно понимать, что данная архитектура была предложена для улучшения производительности памяти. Если программа работает в основном с памятью, то правильное использование NUMA-архитектуры даст улучшение производительности. Если приложение работает с диском или с сетевыми устройствами, то улучшения не будет. Но использование данной архитектуры влечет за собой накладные расходы. Во-первых, так как каждый процессор в такой системе имеет кэш-память, то могут возникать ситуации «неодинакового» видения памяти. Пусть один процессор изменил значение некоторой переменной в своем кэше, а процессор из другого NUMA-блока в это же время прочитал эту же переменную из памяти. В итоге получится, что два процессора имеют разное видение одной и той же переменной. Такая проблема решается за счет системы согласованности кэш-памяти. NUMA-мультипроцессоры с такой системой называются CC-NUMA [26] (Cache-Coherent NUMA — NUMA с согласованным кэшированием). Наличие такой системы согласования ухудшает общую производительность системы, но это вынужденная мера, так как иначе программирование на таких NUMA-системах было бы очень сложной задачей из-за неодинакового видения памяти у каждого процессора NUMA-системы.

Таким образом, у NUMA-мультипроцессоров есть две ключевые особенности, которые нужно принимать во внимание, иначе невозможно будет получить преимущества от использования таких систем. Первая особенность — наличие системы согласования кэшей, вторая особенность — неоднородный доступ к памяти. Систему согласования кэшей можно реализовать на аппаратном уровне, это снизит издержки на поддержку согласованности. Кроме этого при разработке программного обеспечения можно избегать таких ситуаций, когда несколько процессоров работают с одним и тем же участком памяти.

Вторая особенность тоже очень важна. Если задача, работающая на определенном процессоре, постоянно обращается к удаленной памяти, это приводит к недоиспользованию процессора, так как он будет вынужден простаивать во время обработки запроса к памяти. В UNIX-подобных системах есть поддержка NUMA в компиляторах и программных библиотеках [29-34]. Программы, написанные не под NUMA-архитектуру, нужно переписывать с

использованием таких библиотек, что может потребовать существенных вложений времени и средств. Идеальной является ситуация, когда поддержка NUMA-архитектуры не требует никаких вмешательств со стороны человека. Такую поддержку обеспечивают некоторые операционные системы [35-37].

Операционные системы, которые могут работать с NUMA-архитектурой, действуют следующим образом. При размещении очередной задачи на процессор, операционная система пытается выделить память для этой задачи из локальной памяти. Если это не получается сделать, то производится попытка перемещения уже работающих задач на процессорах. При перемещении задачи также перемещаются страницы памяти, которые ассоциированы с данной задачей[38].

Все задачи, выполняющиеся на мультипроцессорах, можно поделить на две группы: задачи интенсивно работающие с памятью и не требующие большого количества вычислительных ресурсов; задачи, которые не работают с памятью так интенсивно, но интенсивно использующие вычислительные ресурсы. Например, в [39] рассматривается проблема размещения задач из этих групп на NUMA-мультипроцессорах.

В данной курсовой работе рассматривается NUMA-мультипроцессор следующего вида. Он состоит из нескольких NUMA-блоков. NUMA-блок представляет собой один многоядерный процессор. Каждый NUMA-блок имеет одинаковое количество ядер и одинаковый объем локальной оперативной памяти. NUMA-блоки обращаются к жесткому диску через один общий контроллер.

В данной работе виртуальной машине разрешено размещаться на физическом сервере с NUMA-архитектурой только на одном NUMA-блоке. При размещении виртуальной машины на нескольких NUMA-блоках ее производительность может упасть. Поэтому возможность размещения виртуальных машины на нескольких NUMA-блоках в данной курсовой работе не рассматривается. Кроме снижения производительности возможность размещения таких машин накладывает существенные требования к среде развертывания виртуальных машин. Но такие гипервизоры существуют, например [35].

Таким образом учет NUMA-архитектуры в том виде, в котором она описана выше, при построении распределения ресурсов в ЦОД дает следующие плюсы и минусы.

Плюсы:

1. Производительность виртуальных машин вырастет, так как будет меньше виртуальных машин, которые разместились на нескольких NUMA-блоках.

Минусы:

1. Ухудшение планируемости, т.е.е количество размещенных в ЦОД запросов будет меньше.
2. Ограничение на максимальный размер виртуальной машины. Размер виртуальной машины будет ограничен размером NUMA-блока, а не размером сервера.

2.4 Критерий выбора алгоритма

Основной задачей курсовой работы является разработка алгоритма, который строит отображение запросов пользователей с учетом политик размещения виртуальных машин и виртуальных хранилищ данных. Кроме этого алгоритм должен учитывать NUMA-архитектуру физических серверов.

В литературе не было найдено алгоритмов, которые решают задачу, представленную в моей курсовой работе. Наиболее полный обзор различных алгоритмов планирования вычислений в ЦОД приведен в работе [40], но в данной курсовой работе мы ограничим класс рассматриваемых алгоритмов алгоритмами, которые наиболее «близки» к рассматриваемой в работе задаче по критериям:

1. Вычислительные ресурсы, хранилища данных являются планируемыми ресурсами,
2. Сетевые ресурсы являются планируемыми ресурсами.
3. Возможность расширения алгоритма для учета политик размещения виртуальных ресурсов и учета NUMA-архитектуры.

2.5 Обзор существующих алгоритмов

В ходе обзора были рассмотрены алгоритмы с различными постановками задач:

1. Максимизация количества размещенных запросов [5,10,13-16,18,20,21,27]
2. Максимизация утилизациия сетевых ресурсов [11]
3. Снижение нагрузки на сетевые ресурсы за счет минимизации коммуникаций в ЦОД между виртуальными машинами. [17,19]
4. Гарантия пропускной способности в каналах [22-24]

Рассмотренные алгоритмы представляют широкий спектр алгоритмов, которые могут применяться в задаче распределения ресурсов в ЦОД. Применяются жадные [14,15,20,27], эвристические алгоритмы [5,9,10], точные алгоритмы [5,18]. В частности, применяются модифицированные алгоритмы упаковки в контейнеры [16]. В некоторых работах задача распределения ресурсов в ЦОД сводится к задаче линейного программирования [18]. Применяются муравьиные алгоритмы[13], алгоритмы имитации отжига[21].

Обзор алгоритмов распределения ресурсов в ЦОД показал, что введенным в разделе 2.4 критериям удовлетворяют четыре алгоритма: алгоритм, основанный на схеме муравьиных колоний [13], алгоритм с раздельными планировщиками ресурсов [14], алгоритм с единым планировщиком для всех типов ресурсов [15], а также алгоритм, представленный в работе [27].

Результаты обзора рассмотренных алгоритмов представлены в таблице 2.1. Критерии анализа алгоритмов были представлены в главе 2.5 «Критерии выбора алгоритма». Данные критерии располагаются в столбцах таблицы 2.1.

Таблица 2.1. Сравнения различных алгоритмов распределения ресурсов ЦОД

Алгоритм	Критерий планирования	Планирование VM или Storage	Планирование сетевых ресурсов	Учет Политик	Учет NUMA-архитектуры
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 [11]	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 [18]	Максимизировать количество размещенных запросов	+	+	-	-
Resource Management in Virtualized Data Center [5]	Максимизировать количество размещенных запросов	+	+	-	-
Algorithms for Assigning Substrate Network Resources to Virtual Network Components [19]	Минимизируется заполнение каналов и узлов, т.е. максимально распределяется запрос	+	+	-	-
CloudNaaS: A Cloud Networking Platform for Enterprise Applications [10]	Максимизировать количество размещенных запросов	+	+	-	-
Алгоритм назначения ресурсов в центрах обработки данных, основанный на схеме муравьиных колоний [13]	Максимизировать количество размещенных запросов	+	+	+	+
Алгоритм распределения ресурсов в центрах обработки данных с раздельными планировщиками для различных типов ресурсов [14]	Максимизировать количество размещенных запросов	+	+	+	+

Алгоритм распределения ресурсов в центрах обработки данных с единым планировщиками для различных типов ресурсов [15]	Максимизировать количество размещенных запросов. Построение компактного отображения.	+	+	+	+
Алгоритм, представленный в дипломной работе [27]	Максимизировать количество размещенных запросов.	+	+	+	+
Towards predictable datacenter networks [20]	Максимизировать количество размещенных запросов	+	-	-	-
Solver for the Network Testbed Mapping problem [21]	Максимизировать количество размещенных запросов	+	-	-	-
NetShare[22] Seawall[23] Gatekeeper[24]	Гарантия пропускной способности	+	+	-	-

2.6 Выводы

В данной главе введены основные понятия и термины, связанные с центрами обработки данных. Кроме этого в главе 2.3 была подробно рассмотрена архитектура с неоднородным доступом к памяти: были выделены плюсы и минусы учета данной архитектуры при распределении ресурсов в центрах обработки данных. В главе 2.5 был представлен обзор существующих алгоритмов. Существующие алгоритмы сравнивались по следующим критериям:

1. Возможность планирования вычислительных ресурсов и хранилищ данных,
2. Возможность планирования сетевых ресурсов,
3. Возможность расширения алгоритма для учета политик на размещение виртуальных ресурсов,

4. Возможность расширения алгоритма для учета NUMA-архитектуры в серверах центров обработки данных.

Из таблицы 2.1, приведенной в главе 2.5 видно, что четыре алгоритма: алгоритм, основанный на схеме муравьиных колоний [13], алгоритм с раздельными планировщиками ресурсов [14], алгоритм с единым планировщиком для всех типов ресурсов [15], а также алгоритм, представленный в работе [27] — подходят по всем критериям, которые были сформулированы в пункте 2.4.

В моей работе за основу был взят алгоритм из [27]. В данной курсовой работе представлена модификация алгоритма [27]. Была модифицирована общая схема работы алгоритма, добавлена процедура учета политик на размещение виртуальных ресурсов, была модифицирована процедура назначения узлов, добавлена процедура учета NUMA-архитектуры.

Причины выбора данного алгоритма для мофикации следующие:

1. Выбранный алгоритм — это алгоритм, сочетающий жадные стратегии и ограниченный перебор. Этот алгоритм работает быстрее, чем муравьиный алгоритм [41]. Муравьиный алгоритм [13] имеет большую вычислительную сложность, чем алгоритмы [14,15,27].
2. Наличие процедуры ограниченного перебора позволяет задавать баланс между точностью получаемого решения и временем работы алгоритма. Этот баланс достигается благодаря изменению глубины ограниченного перебора.
3. Выбранный за основу алгоритм показывает результаты сравнимые с алгоритмами [14,15, 27], а на некоторых данных результаты даже лучше.
4. Выбранный алгоритм достаточно легко расширяем для учета политик размещения виртуальных машин и учета NUMA-архитектуры.

3 Математическая постановка задачи построения распределения ресурсов в центре обработки данных с учетом политик размещения виртуальных ресурсов

Математическую постановку задачи распределения ресурсов в ЦОД возьмем из работы [13] и расширим ее для задачи распределения ресурсов в ЦОД с учетом политик размещения виртуальных ресурсов и учетом NUMA-архитектуры.

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 и L определены векторные функции скалярного аргумента, задающие соответственно характеристики вычислительных узлов, хранилищ данных, коммутационных элементов и каналов передачи данных:

$$\begin{gathered}(ph_1,ph_2,\dots,ph_{n1})=fph(p),\\(mh_1,mh_2,\dots,mh_{n2})=fmh(m),\\(lh_1,lh_2,\dots,lh_{n3})=flh(l).\end{gathered} \quad (3.1)$$

В данной работе предполагается, что у коммутационных элементов нет характеристик, они нужны только как элементы, через которые может проходить маршрут в коммуникационной среде. На физических серверах и системах хранения данных маршрут может либо начинаться, либо заканчиваться.

В данной курсовой работе у элемента из множества P рассматриваются только три характеристики ($n_1=3$):

1. ph_1 — количество ядер,
2. ph_2 — размер оперативной памяти,
3. ph_3 — объем жесткого диска.

NUMA-блок — это составляющая часть любого сервера, т.е. элемента $p\in P$. Для каждого элемента из множества P определена функция, которая показывает, какие NUMA-блоки находятся в его составе:

$$nb(p) = \{nb_1, nb_2, \ldots, nb_{np}\} = NB_p, \ |NB_p| = np$$

Для каждого элемента из множества $nb \in NB_p$ определена векторная функция скалярного аргумента, задающая характеристики NUMA-блока:

$$fnb(nb) = (nbh_1, nbh_2, \ldots, nbh_{n4})$$

У NUMA-блока в данной курсовой рассматриваются две характеристики ($n_4 = 2$):

1. nbh_1 — количество ядер,
2. nbh_2 — объем оперативной памяти.

Для каждого элемента $p \in P$ выполнено следующее соотношение:

$\sum_{nb \in NB_p} fnb(nb) = (ph_1, ph_2)$, где $ph_1, \ ph_2$ определяются из $(ph_1, ph_2, \ldots, ph_{n1}) = fph(p)$

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

$$\begin{gathered} (wg_1, wg_2, \ldots, wg_{n1}) = fwg(w), \\ (sg_1, sg_2, \ldots, sg_{n2}) = fsg(s), \\ (eg_1, eg_2, \ldots, eg_{n4}) = feg(e). \end{gathered} \quad (3.2)$$

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

Политики размещения виртуальных ресурсов задаются для каждого запроса G . В данной работе рассматриваются политики только для виртуальных машин. Политики могут быть 4-ех типов:

1. VA (VM-to-VM affinity). Для каждой виртуальной машины w_i из запроса определена функция $VA(w_i) = (w_1, w_2, \ldots, w_m)$, где $(w_1, w_2, \ldots, w_m)$ — это некоторые виртуальные

машины из запроса G . Данная функция показывает, какие виртуальные машины нужно разместить на одном и том же сервере.

2. VNA (VM-to-VM anti-affinity). Для каждой виртуальной машины w_i из запроса определена функция $VNA(w_i)=(w_1,w_2,\dots,w_k)$, где $(w_1,w_2,\dots,w_k)$ — это виртуальные машины из запроса G . Данная функция показывает, какие виртуальные машины нужно разместить на разных серверах.
3. PA (VM-to-PM affinity). Для каждой виртуальной машины w_i из запроса определена функция $PA(w_i)=(p_1,p_2,\dots,p_l)$, где $(p_1,p_2,\dots,p_l)$ — это физические сервера из множества P . Данная функция показывает, на каких физических серверах нужно размещать виртуальную машину w_i .
4. PNA (VM-to-PM anti-affinity).Для каждой виртуальной машины w_i из запроса определена функция $PNA(w_i)=(p_1,p_2,\dots,p_s)$, где $(p_1,p_2,\dots,p_s)$ — это физические сервера из множества P . Данная функция показывает, на каких физических серверах нельзя размещать виртуальную машину w_i .

Множество виртуальных машин, для которых заданы политики размещения $W_R=W_{VA}\cup W_{VNA}\cup W_{PA}\cup W_{PNA}$. W_{VA} — множество виртуальных машин, для которых заданы политики VM-to-VM affinity; W_{VNA} — множество виртуальных машин, для которых заданы политики VM-to-VM anti-affinity; W_{PA} — множество виртуальных машин, для которых заданы политики VM-to-PM affinity; W_{PNA} — множество виртуальных машин, для которых заданы политики VM-to-PM anti-affinity.

Множество политик для запроса G:

$Rs_{VA}=\{VA(w_i)|w_i\in W_{VA}\}$, $Rs_{VNA}=\{VNA(w_i)|w_i\in W_{VNA}\}$, $Rs_{PA}=\{PA(w_i)|w_i\in W_{PA}\}$

$Rs_{PNA}=\{PNA(w_i)|w_i\in W_{PNA}\}$, $Rs(G)=Rs_{VA}\cup Rs_{VNA}\cup Rs_{PA}\cup Rs_{PNA}$

Назначением запроса называется отображение *B*:

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

Запрос *G* называется **назначенным полностью**, если

$\forall g \in G : B(g) \neq \varnothing$ (3.4)

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

$\exists g_1 \in G : B(g_1) \neq \varnothing$ и $\exists g_2 \in G : B(g_2) = \varnothing$ (3.5)

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

Совокупность назначений запросов будем обозначать *A*.

Выделим пять типов 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: $\sum_{i \in R_j} x_i \leq y_j$, здесь R_j - множество виртуальных ресурсов, 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: $x \leq y$.
4. (Для серверов). Недопустимость перегрузки NUMA-блоков у физических серверов: $\sum_{i \in Q_j} x_i \leq z_j$, здесь Q_j - множество виртуальных ресурсов, назначенных на выполнение на NUMA-блоке физического ресурса *j*. z — это соответствующая x характеристика NUMA-блока.
5. (Для серверов). Каждая виртуальная машина может быть назначена только на один NUMA-блок одного конкретного сервера.

Отображение $A : G \to H = \{W \to P, S \to M, E \to \{K, L\}\}$ будем называть **корректным**, если для всех физических ресурсов и всех их характеристик выполняются отношения 1-5, а также для каждого запроса G выполнены политики $Rs(G)$.

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:

$$fph_{res}(p) = fph(p) - \sum_{w \in W_p} fwg(w)$$
$$fmh_{res}(m) = fmh(m) - \sum_{s \in S_m} fsg(s) \quad (3.6)$$
$$flh_{res}(l) = flh(l) - \sum_{e \in E_l} feg(e)$$

Здесь W_p — множество виртуальных машин, назначенных на выполнение на вычислительном узле p, E_l - множество виртуальных каналов, отображенных на физический канал l, S_m — множество виртуальных хранилищ данных, размещенных в системе хранения 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.**

4 Описание алгоритма распределения ресурсов в центре обработки данных с учетом политик размещения виртуальных ресурсов

В данной главе приведено описание алгоритма распределения ресурсов в ЦОД. Данный алгоритм является расширением алгоритма [27] в части удовлетворения политик размещения виртуальных ресурсов и учета особенностей NUMA-архтектуры. Данный алгоритм состоит из трех этапов: сначала вызывается процедура учета политик размещения виртуальных ресурсов, далее назначаются виртуальные узлы запроса, после этого прокладываются виртуальные каналы связи между ними. Так как алгоритм, представленный в данной курсовой работе, является модификацией алгоритма из [27], то в данной главе приведены только модифицированные этапы алгоритма. Те этапы алгоритма, которые важны для понимания алгоритма в целом, но не были разработаны в рамках данной курсовой работы, приведены в Приложении А. Детальное описание алгоритма, на котором базируется решение, предложенное в данной работе, можно посмотреть в [27].

4.1 Схема работы 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.

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 $V = \{W \cup S\}$
4. Провести назначение виртуальных машин W с учетом политик на их размещение. В случае неуспеха прейти к **шагу 7**.
5. Провести назначение виртуальных узлов U. В случае неуспеха перейти к **шагу 7**
6. Провести назначение виртуальных каналов E запроса G_i. В случае успеха перейти к **шагу 2**.

7. 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. Описание шага 6 приведено в Приложении А и в [27].

4.2 Процедура удовлетворения политик размещения виртуальных машин

Сформируем множество виртуальных машин, для которых заданы политики размещения $W_R = W_{VA} \cup W_{VNA} \cup W_{PA} \cup W_{PNA}$ (См. Главу 3).

Виртуальные машины из множества W_{PA} назначаются в соответствии с жадным алгоритмом (См. Раздел 4.3, вызывается процедура назначения виртуальных узлов без вызова процедуры ограниченного перебора). Если не удалось назначить виртуальные машины, то вернуть **НЕУСПЕХ**.

Соблюдение политик для виртуальных машин из множества W_{PNA} контролируется в процессе работы алгоритма. При назначении виртуальной машины $w \in W_{PNA}$ на $p \in P$ проверяется принадлежность p множеству W_{PNA}.

Далее рассмотрим процедуру удовлетворения политик для множеств W_{VA} и W_{VNA}. Сформируем множество $Rs_v = Rs_{VA} \cup Rs_{VNA}$. Каждый элемент этого множества является множеством из виртуальных машин, которые связаны между собой некоторой политикой размещения. Процедура удовлетворения политик выглядит следующим образом.

1. Если множество Rs_v не пусто, то выбираем любой элемент этого множества $R \in Rs_v$, иначе перейти к **шагу 4**.
2. Если $R \in R_{VNA}$, то провести следующие действия:
   a) Выбираем виртуальную машину *N* из множества *R* в соответствии с жадным критерием K_V. Если множество *R* пусто перейти к **шагу 2g**.
   b) Если виртуальная машина *N* имеет в графе запроса *G* связанные с ней виртуальные машины, и некоторые из этих машин уже назначены на некоторые физические сервера, то выполняем процедуру поиска в ширину с пересечением кандидатов [15], иначе переходим к **шагу 2c**. Если процедура завершилась успешно, то перейти к **шагу 2a**, иначе перейти к **шагу 2c**.

c) 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 вернуть **НЕУСПЕХ**.

d) Выбрать физический ресурс *ph* из множества *Ph* в соответствии с жадным критерием K_P. Если множество *Ph* пусто и *N* не назначен, то вернуть **НЕУСПЕХ**.

e) Если на данном сервере не располагаются виртуальные машины из *R*, то назначить *N* на *ph* и переопределить значения характеристик физических ресурсов в соответствии с функциями (3.6). Иначе перейте к **шагу 2d**.

f) Удалить *N* из *R*. Перейти к **шагу 2a**.

g) Перейти к **шагу 1**.

3. Если $R \in R_{VA}$, то провести следующие действия:

a) Сортируем по неубыванию виртуальные машины из множества *R* в соответствии с жадным критерием K^*_V. *N* — это самая первая виртуальная машина в отсортированоом множестве.

b) 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 вернуть **НЕУСПЕХ**.

c) Выбрать физический ресурс *ph* из множества *Ph* в соответствии с жадным критерием K_P. Если множество *Ph* пусто и виртуальные машины из множества *R* не назначены, то вернуть **НЕУСПЕХ**.

d) Если на данном сервере могут расположиться все виртуальные машины из *R*, то назначить их на *ph* и переопределить значения характеристик физических ресурсов в соответствии с функциями (3.6). Иначе перейте к **шагу 3c**.

e) Для всех виртуальных машин запретить миграцию. Перейти к **шагу 1**.

4. Вернуть **УСПЕХ**.

При назначении виртуальной машины на сервер вызывается процедура назначения виртуальной машины внутри физического сервера (См. Главу 4.3, пункт 2b; См. Главу 4.4).

4.3 Процедура назначения виртуальных узлов на физические ресурсы центра обработки данных

Виртуальные узлы $U = \{W \backslash W_R \cup S\}$ 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, и для которых выполнены отношения корректности отображения 1-3 (См. Главу 3) 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. Если ph является физическим сервером, то вызвать процедуру назначения виртуальной машины внутри физического сервера. При неуспешном завершнении процедуры вернуть **НЕУСПЕХ**.
   c) Назначить N на ph и переопределить значения характеристик физических ресурсов в соответствии с функциями (3.6).
   d) Выбрать все виртуальные каналы E, связывающие элемент N с еще не назначенными элементами запроса G_i.
   e) Отсортировать множество каналов E по величине пропускной способности в невозрастающем порядке
   f) Удалить N из Q и U
   g) Добавить в Q виртуальные узлы, связанные с N, согласно порядку их следования в отсортированном множестве E.
   h) Если Q не пусто перейти к **шагу 2**
   i) Если множество U не пусто, то перейти к **шагу 1**, иначе вернуть **УСПЕХ**.

Описание процедуры ограниченного перебора приведено в Приложении А и в [27]. Процедура назначения виртуальных машин на NUMA-блоки приведена ниже в главе 4.4.

4.4 Процедура назначения виртуальной машины внутри физического сервера центра обработки данных

Рассмотрим процедуру назначения виртуальной машины *N* на физический сервер *P* с учетом NUMA-архитектуры сервера *P*:

1. Сформировать для *P* множество *NB* NUMA-блоков, на которые может быть назначен данный элемент *N*, и для которых выполнены отношения корректности отображения 4-5 (См. Главу 3). Если данное множество пусто, то вызвать процедуру ограниченного перебора внутри физического сервера. При неуспешном завершении процедуры вернуть **НЕУСПЕХ**.
2. Выбрать NUMA-блок *nb* из множества *NB* в соответствии с жадным критерием K_{NB}. Назначить *N* на *nb* и переопределить значения характеристик физических ресурсов в соответствии с функциями (3.6).
3. Вернуть **УСПЕХ**.

4.5 Процедура ограниченного перебора внутри физического сервера центра обработки данных

Процедура ограниченного перебора внутри физического сервера является модификацией процедуры ограниченного перебора из работы [27]. Процедура ограниченного перебора вызывается при отсутствии возможности назначить очередную виртуальную машину *N* из запроса *G* ни на один NUMA-блок физического сервера *P* (См. Раздел 4.4, процедура назначения виртуальных машины внутри физического сервера, шаг 1). Процедура рассматривает некоторые подмножества множества NUMA-блоков физического сервера *P*. Мощность подмножеств – это глубина перебора d_{NB}. Количество просматриваемых подмножеств ограничивается числом MA_{NB}. Просматриваются только те подмножества, у которых суммарное количество остаточных ресурсов у узлов достаточно для назначения текущего элемента *N*.

Общая схема процедуры ограниченного перебора:

1. Для заданного d_{NB} сформировать все подмножества *Cs* множества NUMA-блоков физического сервера *P* из d_{NB} NUMA-блоков.

2. Если для некоторого множества *C* из *Cs* суммарная остаточная емкость ресурсов достаточна для назначения рассматриваемой виртуальной машины:
   a) Сохранить назначение виртуальных машин
   b) Произвести снятие элементов *V*, которые назначены на физические ресурсы из множества C.
   c) Производим попытку назначить элементы множества *{V∪N}* на физические ресурсы из множества *C* с использованием жадной схемы (См. Раздел 4.3, шаги 1-2a-2-c без вызова процедуры ограниченного перебора и процедуры назначения виртуальной машины внутри сервера). Если удалось назначить элементы, то возвращаем **УСПЕХ**.
   d) Восстанавливаем старое назначение узлов *V*. Удаляем множество *C* из *Cs*. Рассматриваем следующее множество *C* - переходим к **шагу 2**, если множество *Cs* пусто, то возвращаем **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, для выбора NUMA-блока — K_{NB}. Описание данных критериев представлено в Приложении Б.

4.7 Выводы

В данной главе описан разработанный алгоритм распределения ресурсов ЦОД. Алгоритм состоит из трех последовательных шагов: процедуры учета политик на размещение виртуальных ресурсов, назначения виртуальных узлов и назначения виртуальных каналов.

Процедура назначения политик рассматривает последовательно множества виртуальных машин, которые связаны некоторой общей политикой.

Назначение узлов производится по жадному алгоритму. При назначении виртуальной машины вызывается процедура назначения виртуальной машины с учетом внутреннй архитектуры сервера. В случае, когда очередной виртуальный узел не может быть размещен, вызывается процедура ограниченного перебора.

Поиск маршрутов для виртуальных каналов связи выполняется при помощи алгоритма поиска в ширину [42].

5 Реализация разработанного алгоритма

Разработанный в данной работе алгоритм является модицикацией алгоритма из работы [27], поэтому для его реализации был модифицирована реализация алгоритма из [27]. Был реализован программный модуль учета NUMA-архитектуры, модифицирована схема работы алгоритма и добавлена процедура учета политик на размещение виртуальных ресурсов. Исходный программный модуль написан на языке C++, он же использовался в качестве основного языка программирования для реализации предложенных в данной работе изменений модификаций.

Модифицированная реализация алгоритма позволяет строить отображение запросов на физические ресурсы с учетом политик размещения виртуальных ресурсов и с учетом NUMA-архитектуры физических серверов ЦОД.

5.1 Описание программного модуля

На рисунке 5.1 приведена UML-диаграмма [43] классов модифицированного программного модуля для решения задачи планирования вычислений в ЦОД с учетом политик на размещение виртуальных ресурсов и учетом NUMA-архитектуры. Зеленым цветом отмечены разработанные и модифицированные классы.

Разработанный класс *PrototypeAlgorithm* реализует трехшаговую схему назначения виртуальных ресурсов, описанную в главе 4.1. Класс *Element* был модифицирован: в него добавились проверки политик размещения виртуальных ресурсов. Также был модифицирован класс *Computer*. Теперь он хранит внутри себя NUMA-блоки, может проводить назначение виртуальных машин согласно алгоритму, описанному в главе 4.3 и главе 4.4, а также при необходимости вызывать процедуру ограниченного перебора, описанную в главе 4.5. Класс *Snapshot*, который отвечает за считывание входных данных, был изменен для того, чтобы была возможность считывать политики на размещение виртуальных ресурсов. Класс *NumaBlock* представляет собой абстракцию NUMA-блока в физическом сервере.

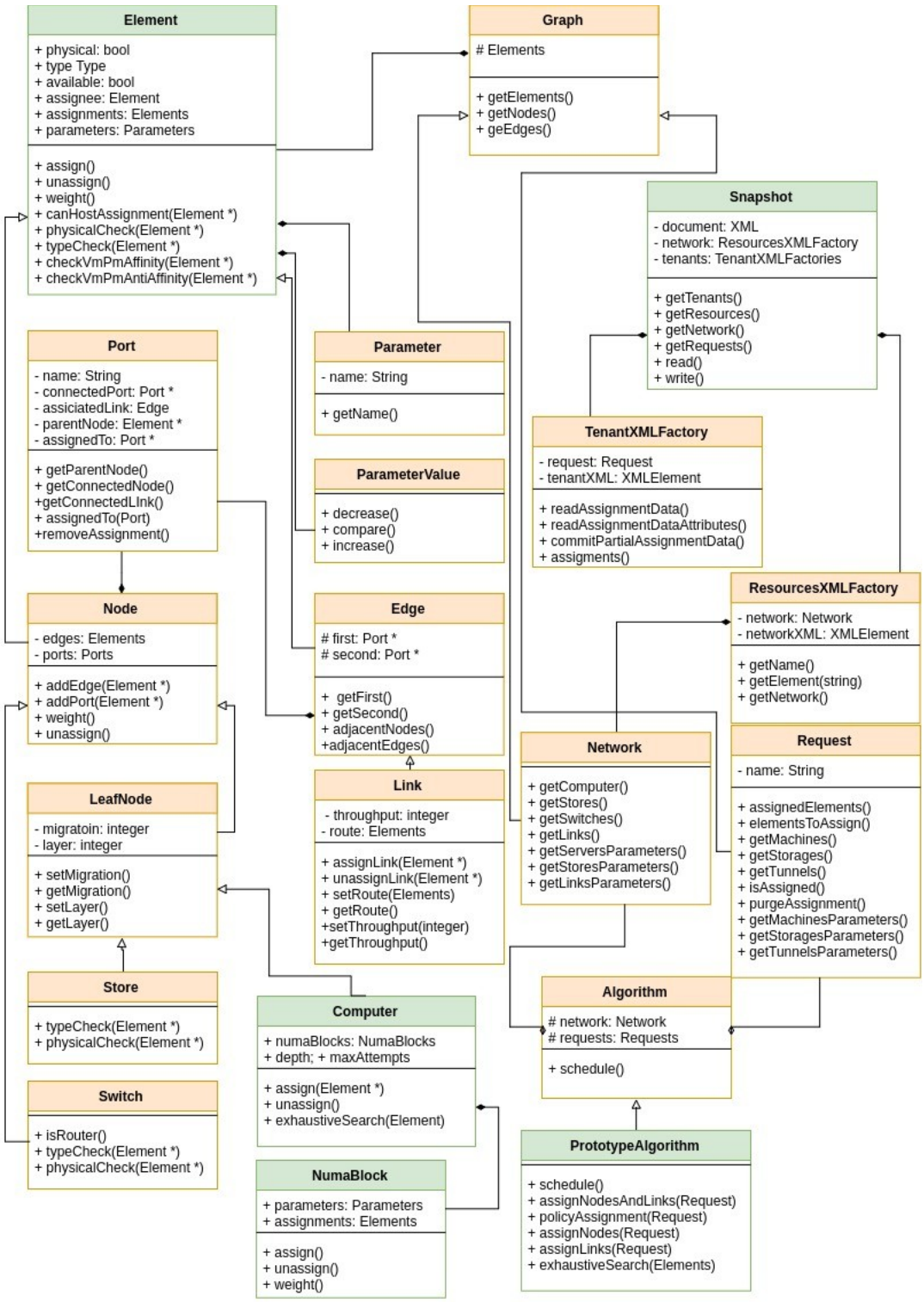

Рисунок 5.1 Диаграмма классов алгоритма планирования вычислений в ЦОД с учетом политик размещения виртуальных ресурсов и учетом NUMA-архитектуры серверов

6 Экспериментальное исследования разработанного алгоритма

Сформулируем цели экспериментального исследования разработанного алгоритма:

1. Исследовать влияние учета NUMA-архитектуры на количество размещенных запросов в центре обработки данных.
2. Исследовать влияние учета NUMA-архитектуры на производительность виртуальных машин.
3. Сравнение результатов работы разработанного алгоритма с существующим алгоритмом при учете политик на размещение виртуальных ресурсов.

Описание используемой вычислительной системы:

1. Размер оперативной памяти — 4 Gb
2. Тип процессора — Intel(R) Core(TM) i5-4210U CPU 1.70GHz, 4 ядра
3. Версия операционной системы — Linux Mint 18.1 Serena

6.1 Влияние учета неоднородной архитектуры физических серверов на качество получаемого решения

В главе 2.3 были приведены плюсы и минусы учета NUMA-архитектуры физических серверов в алгоритмах планирования. Главный минус учета NUMA-архитектуры — это снижение количества размещенных запросов в ЦОД. Для того чтобы проверить насколько сильно снизится качество получаемого решения, был проведен эксперимент.

6.1.1 Исходные данные

Для проведения исследования в качестве физической топологии рассматривается древовидная топология. Физическая топология состоит из серверов и коммутационного оборудования. Топологии запроса могут быть различными: дерево, кольцо, звезда.

При генерации тестов параметры физической топологии были следующими :

1. Количество физических серверов — 63, количество коммутаторов — 63.
2. Параметры физических узлов:

a) Количество ядер — 16
b) Количество RAM — 64 Гб

3. Пропускные способности физических каналов связи – от 200 до 300 Мб/с

При генерации тестов параметры запросов были следующими:

1. Количество запросов — 100
2. Количество виртуальных машин в запросе — 6
3. Параметры виртуальных машин:
   a) Количество ядер — от 1 до 6
   b) Количество RAM — от 8 до 16 Гб
4. Пропускные способности виртуальных каналов – от 2 до 8 Мб/с
5. Топология выбиралась случайным образом из множества {кольцо, дерево, звезда}.

Всего было сгенерировано 15 тестов. Граф физической инфраструктуры ЦОД во всех тестах был одинаковым. Все тесты можно поделить на 5 групп по 3 теста в каждой. Тесты в одной группе отличаются только в количестве NUMA-блоков в физической топологии. В каждой группе есть один тест, в котором все физические сервера имеют лишь один NUMA-блок (это равносильно отсутствию NUMA-блоков; таким образом сейчас представляются физические сервера во всех существующих алгоритмах распределения ресурсов в ЦОД), тест, где все сервера имеют два NUMA-блока и тест, в котором сервера имеют 4 NUMA-блока.

В совокупности было проведено 15 тестов. Подробное описание исходных данных можно найти в Приложении В.

6.1.2 Результаты экспериментального исследования

Результаты экспериментального исследования представлены на рис. 6.1-6.2. Данные, на основе, которых строились приведенные рисунки, приведены в табл. В1-В3.

Обозначения на рисунке 6.2:

1. Загрузка RAM обозначается красным цветом, CPU — синим цветом, сетевые ресурсы — желтым цветом

2. Загрузка ресурса конкретного типа — это отношение суммарного количества используемого ресурса к суммарному количеству имеющегося ресурса этого типа в ЦОД. Загрузка представляет собой число от 0 до 1
3. Для каждого теста три столбца (синий, красный, желтый) относятся к ситуации с одним NUMA-блоком, другие три столбца для ситуации с двумя NUMA-блоками и оставшиеся столбцы относятся к загрузке ресурсов в ЦОД с физическими серверами, которые имеют 4 NUMA-блока в своем составе.

На рис. 6.1 показано количество размещенных запросов в ЦОД в зависимости от количества NUMA-блоков в серверах.

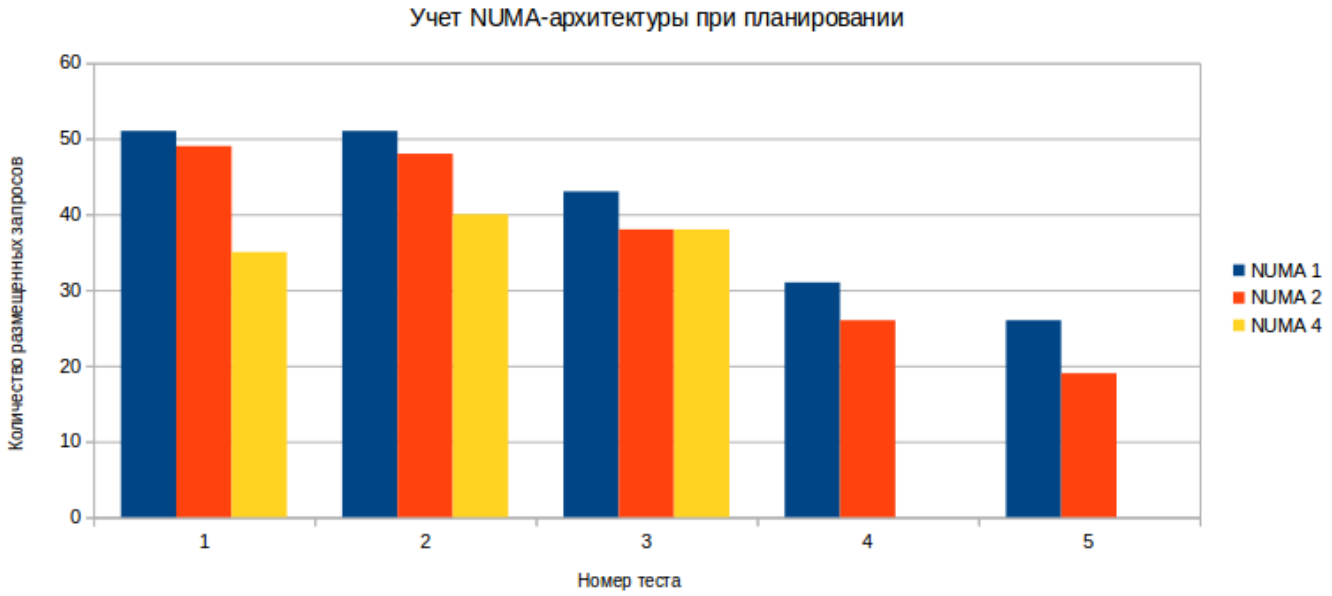

Рисунок 6.1. Количество размещеных запросов

На рис. 6.2 показана загрузка вычислительных и сетевых ресурсов в зависимости от количества NUMA-блоков в составе физических серверов ЦОД. На тестах 4 и 5 алгоритм не разместил ни одного запроса в ситуации, когда NUMA-блоков в сервере 4, так как виртуальные машины имели размер больше, чем размер NUMA-блока. Исходя из представленнных диаграмм, видно, что при переходе от одного NUMA-блока к двум NUMA-блокам качество получаемого решения снижается не слишком сильно по сравнению с переходом от двух NUMA-блоков к четырем NUMA-блокам.

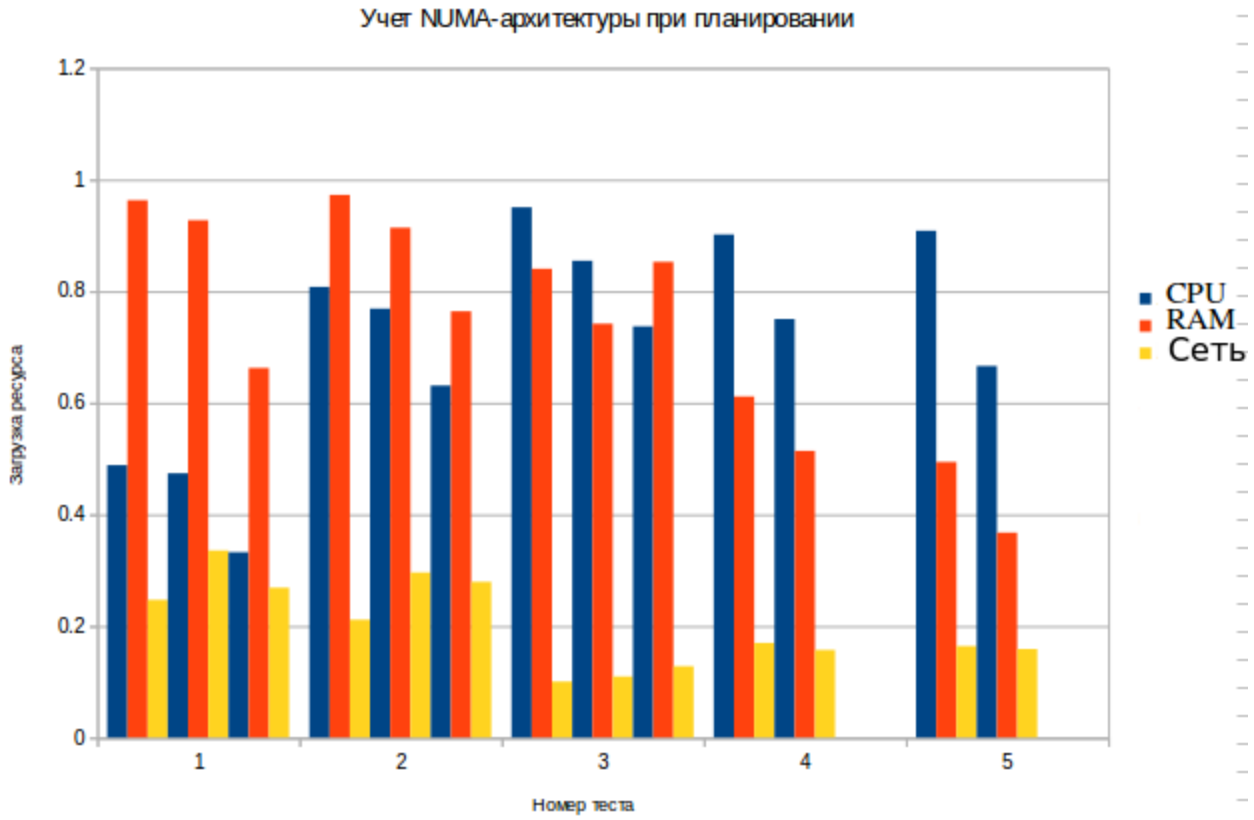

Рисунок 6.2. Загрузка физических ресурсов ЦОД

6.2 Влияние учета неоднородной архитектуры физических серверов на производительность виртуальных машин

В главе 2.3 были приведены плюсы и минусы учета NUMA-архитектуры физических серверов в алгоритмах планирования. Снижение количества размещенных запросов в ЦОД исследовалось в главе 6.1. Главным плюсом учета NUMA-архитектуры — это повышение производительности виртуальных машин за счет того, что они полностью располагаются в одном NUMA блоке и не существует так называемых «Wide-VM» [35]. В терминологии Vmware [35] это такие виртуальные машины, которые располагаются на нескольких NUMA-блоков, вследствие чего у них падает производительность. В данной главе представлены результаты исследования зависимости производительности виртуальных машин от количества NUMA-блоков на физическом сервере.

6.2.1 Исходные данные

Исходные данные в данном эксперименте такие же, как и в эксперименте, описанном в главе 6.1. Было проведено 15 запусков алгоритма. Алгоритм при размещении виртуальной машины на физическом сервере не учитывал NUMA-архитектуру, а размещал таким образом, как будто сервер представляет собой один большой процессор. Работа алгоритма в таком режиме показывает каким образом работают сейчас существующие алгоритмы (алгоритмы, которые не учитывают NUMA-архитектуру).

После каждого запуска вычислялось отношение количества виртуальных машин, которые не уместились целиком на одном NUMA-блоке (данные виртуальные машины являются низкопроизводительными), к общему количеству размещенных в ЦОД виртуальных машин. Данная характеристика принималась за оценку производительности виртуальных машин в ЦОД. Чем ближе данная характеристика к 0, тем больше высокопроизводительных виртуальных машин, чем ближе к 1 — тем больше низкопроизводительных.

6.2.2 Результаты экспериментального исследования

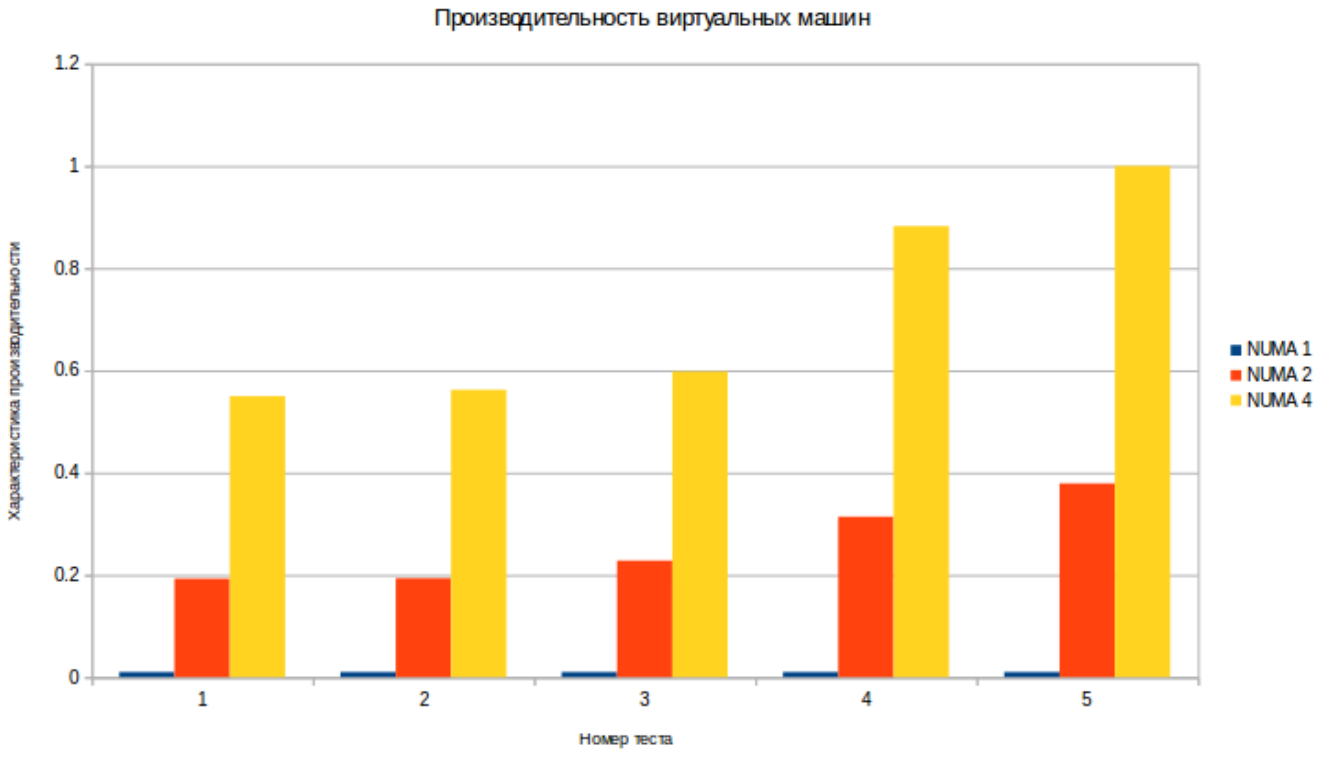

Рисунок 6.3. Снижение производительности виртуальных машин при неучете NUMA-архитектуры физических серверов

На рис. 6.3 представлена диаграмма с результатами эксперимента. На диаграмме видно, как сильно снижается производительность виртуальных машин при неучете NUMA-архитектуры. От 20 до 40% виртуальных машин в тестах 1-5 имеют низкую производительность из-за того, что располагаются на нескольких NUMA-блоках. Если у сервера 4 NUMA-блока, то производительность падает у 50-100% машин.

Таким образом, с одной стороны, учет NUMA-архитектуры снижает количество размещенных запросов, с другой — уменьшает производительность виртуальных машин. Между двумя этими показателями можно искать баланс. Но эта задача дальнейших исследований.

На рисунке 6.2 показано, что в ситуации с двумя NUMA-блоками в каждом сервере, загрузка вычислительных ресурсов снижается не очень сильно, количество размещенных запросов тоже снижается не сильно (См. Рис. 6.1). В это же время производительность может упасть у 20% виртуальных машин (См. Рис 6.3), если не учитывать NUMA-архитектуру. Этот пример показывает, что учет NUMA-архитектуры не всегда приводит к существенному снижению качества получаемого планировщиком решения.

6.3 Сравнение результатов работы разработанного алгоритма с существующим алгоритмом при учете политик на размещение виртуальных ресурсов

В данной главе представлено сравнению разработанного алгоритма с коммерческим алгоритмом. Сравнивалось количество размещенных запросов, количество выполненных политик, а также загрузка физических ресурсов.

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 физических серверов. В каждом кластере есть только один коммутатор, к которому присоединены физические сервера, физические сервера 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.

Всего было сгенерировано 12 тестов.

6.3.2 Результаты экспериментального исследования

На рис. 6.4-6.6 представлены результаты экспериментального исследования.

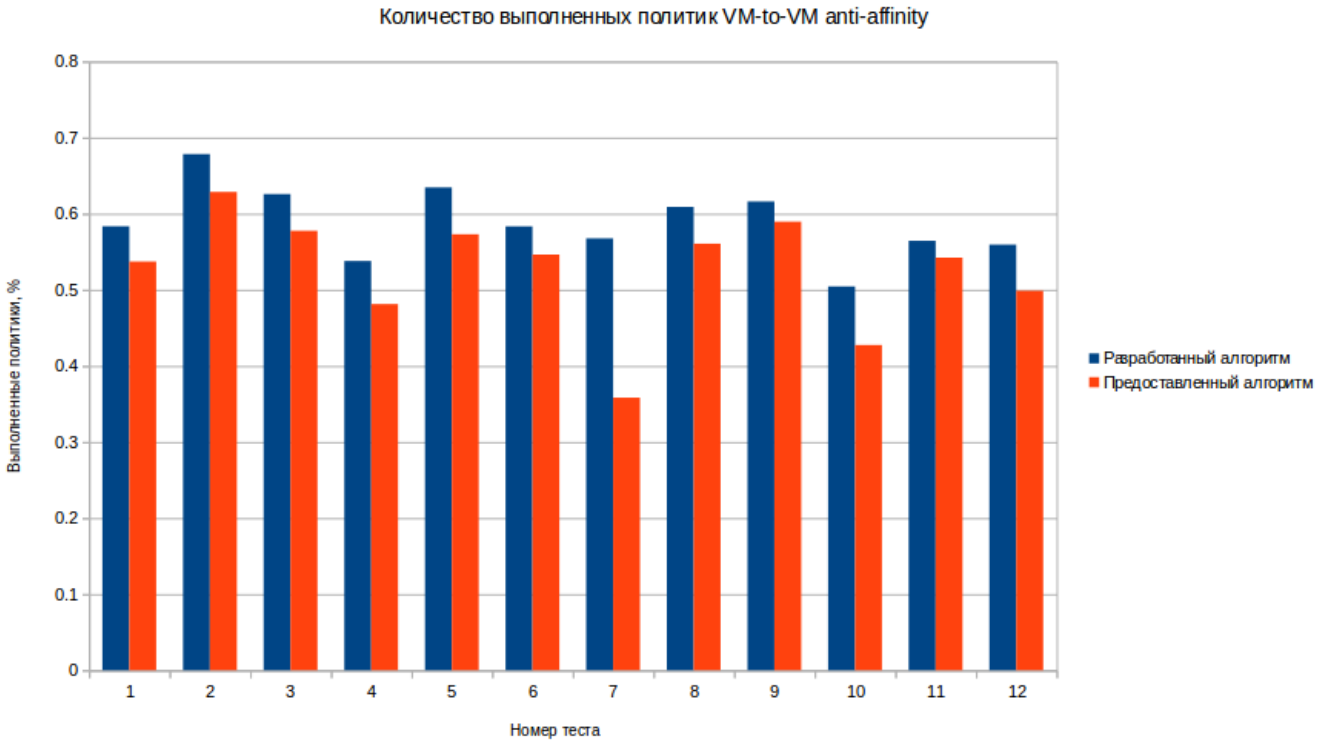

Рисунок 6.4. Количество выполненных политик размещения виртуальных машин

На рис. 6.4 изображено количество выполненных политик VM-to-VM anti-affinity (См. Главу 3). Разработнный алгоритм на всех тестах показал результаты лучше, чем предоставленный алгоритм.

На рис. 6.5 показан процент размещенных запросов. Из диаграммы видно, что разработанный алгоритм разместил больше запросов, чем предоставленный алгоритм.

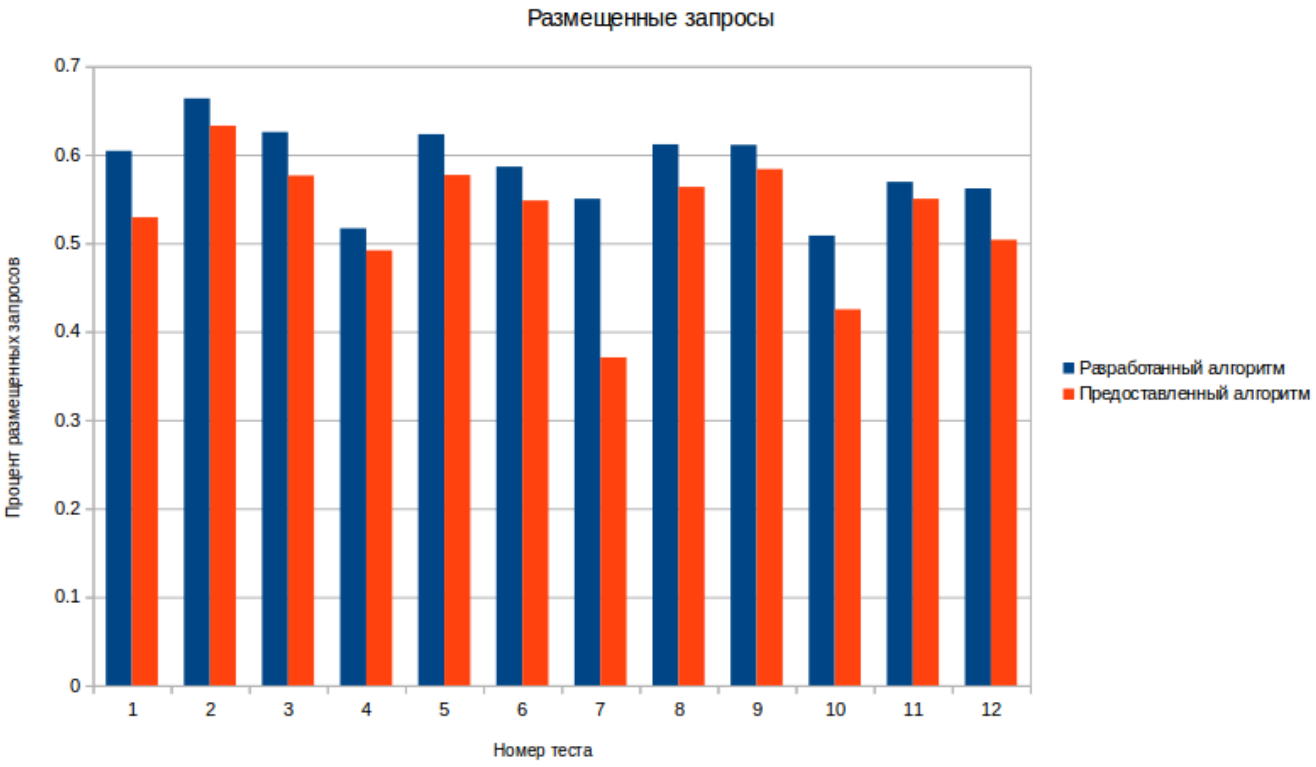

Рисунок 6.5. Процент размещенных запросов в ЦОД

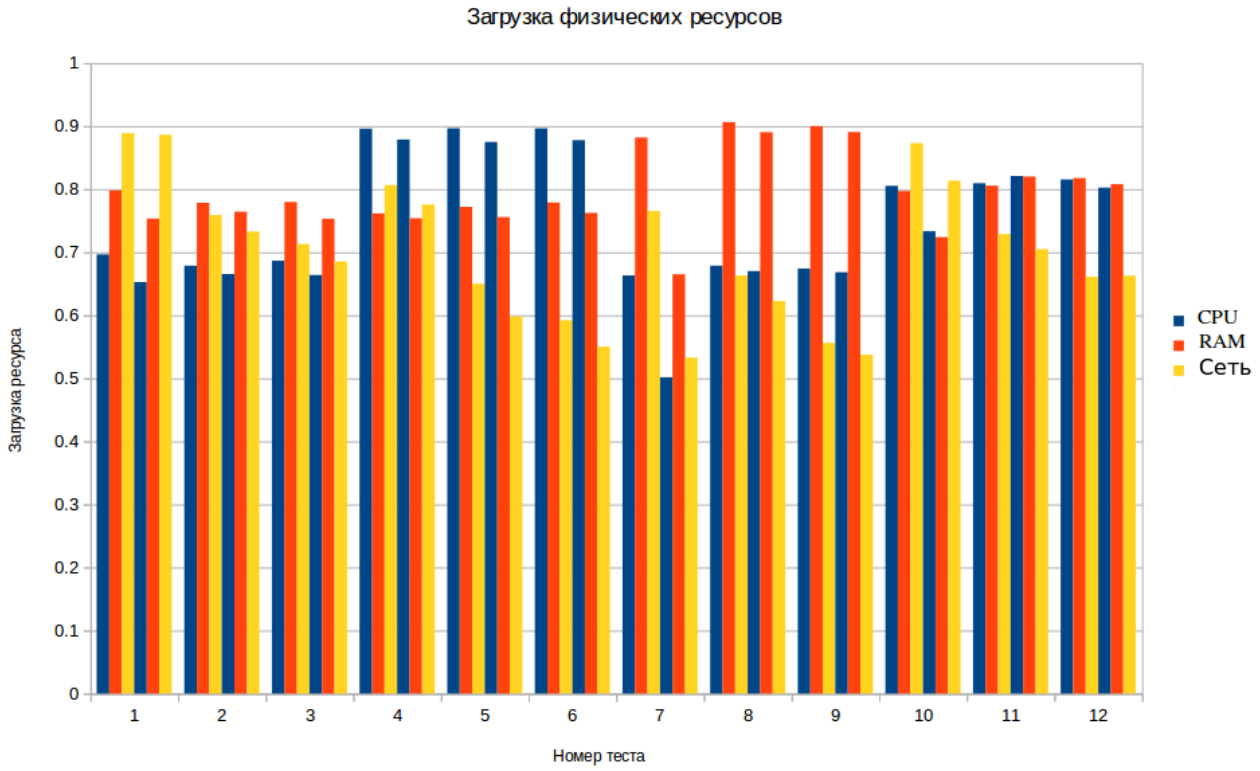

Рисунок 6.6. Загрузка физических ресурсов в ЦОД

Обозначения на рисунке 6.6:

1. Загрузка RAM обозначается красным цветом, CPU — синим цветом, сетевые ресурсы — желтым цветом
2. Загрузка ресурса конкретного типа — это отношение суммарного количества используемого ресурса к суммарному количеству имеющегося ресурса этого типа в ЦОД. Загрузка представляет собой число от 0 до 1
3. Для каждого теста первые три столбца (синий, красный, желтый) относятся к разработанному в данной курсовой работе алгоритму, остальные столбцы относятся к предоставленному алгоритму.

   Из рис. 6.6 видно, что разработанный алгоритм более эффективно использует ЦОД.

6.4 Выводы

Проведенное исследование показало, что учет NUMA-архитектуры снижает качество получаемого решения (См. Рис 6.1-6.2). Но не учитывать NUMA-архитектуру нельзя, как показано во втором эксперименте (См. Главу 6.2). При игнорировании NUMA-архитектуры у виртуальных машин падает производительность. На рис. 6.3 показано, что от 20% до 100% виртуальных машин могут стать низко производительными, если алгоритм планирования при размещении виртуальных машин не учитывает NUMA-архитектуру физических серверов ЦОД.

Получается, что можно выделить два противоположный критерия планируемости: максимизация количества размещенных запросов и минимизация низко производительных виртуальных машин. Построение алгоритма, который учитывает эти два критерия, является актуальной задачей. В можно выделить решение, которое удовлетворяет этим двум критериям. Если рассматривать тесты, в которых физическая топология состоит из серверов, в которых два NUMA-блока, то снижение количества размещенных запросов не такое сильное. А учет NUMA-архитектуры позволяет избежать появления большого количества низко производительных виртуальных машин.

Кроме экспериментов, связанных с учетом NUMA-архитектуры, был проведен эксперимент с учетом политик размещения виртуальных машин (См. Главу 6.3). В данном

эксперименте разработанный в данной курсовой алгоритм сравнивался с предоставленным компанией Huawei алгоритмом по следующим критериям:

1. Количество размещенных запросов,
2. Количество выполненных политик,
3. Загрузка физических ресурсов.

Разработанный алгоритм показал результаты лучше, чем предоставленный алгоритм, по всем трем критериям.

Заключение

В рамках курсовой работы были выполнены следующие задачи:

1. Разработан алгоритм распределения ресурсов в центрах обработки данных, который рассматривает вычислительные ресурсы, ресурсы хранения данных и сетевые ресурсы, как планируемые ресурсы, кроме этого алгоритм учитывает политики размещения виртуальных машин и учитывает NUMA-архитектуру физических серверов.
2. Проведено исследование влияния учета NUMA-архитектуры на качество получаемого решения. Было показано, что учет NUMA-архитектуры очень важен, так как без него при планировании возникает большое количество низко производительных виртуальных машин.
3. Проведено сравнение разработанного алгоритма с существующим алгоритмом.

Список использованных источников

1. Amazon Web Services. [Электронный ресурс]. URL https://aws.amazon.com/ (дата обращения: 10.04.2017)
2. Microsoft Azure. [Электронный ресурс]. URL https://azure.microsoft.com/en-us/ (дата обращения: 10.04.2017)
3. IBM Cloud. [Электронный ресурс]. URL https://www.ibm.com/cloud-computing/ (дата обращения: 10.04.2017)
4. Google App Engine. [Электронный ресурс]. URL http://code.google.com/appengine (дата обращения: 10.04.2017)
5. 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).
6. 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).
7. Clos C. A Study of Non-Blocking Switching Networks // Bell System Technical Journal, — March 1953, — P. 406-424
8. 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).
9. 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).
10. 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).

11. 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
12. S. Kolliopoulos, C. Stein. Improved approximation algorithms for unsplittable flow problems. In Foundations of Computer Science, — 1997. —P.426 - 436
13. 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
14. 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.
15. 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.
16. A. Ngenzi, Selvarani R, S. Nair, Dynamic Resource Management In Cloud Data Centers For Server Consolidation — Bangalore-India —2015 — 8с. [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. 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.
19. Algorithms for Assigning Substrate Network Resources to Virtual Network Components / Yong Zhu and Mostafa Ammar — Atlanta, Georgia, USA, 2006 —12с. . [Электронный ресурс]. URL: http://www.cc.gatech.edu/fac/Mostafa.Ammar/papers/INFOCOM-YONG.pdf (дата обращения: 12.04.2016).
20. 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).

21. R. Ricci, C. Alfeld, J. Lepreau: A Solver for the Network Testbed Mapping problem. //ACM SIGCOMM Computer Communication Review — 2003 — 33, — P. 65-81.
22. T. Lam, S. Radhakrishnan, A. Vahdat, and G. Varghese. NetShare: Virtualizing DataCenter Networks across Services, — San Diego,CA,USA, 2010
23. 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).
24. 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).
25. 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).
26. Танненбаум Э., Бос Х. Современные операционные системы. 4-ое изд. — Спб.: Питер, 2015. — 1120 с.: ил. — (Серия “Классика computer science”).
27. Чупахин А.А., Планирование вычислений в центрах обработки данных с построением плана миграции // Выпускная квалификационна работа, ВМК, МГУ, 2016.
28. Вдовин П.М. Жадные алгоритмы и стратегии ограниченного перебора для планирования вычислений в системах с жесткими требованиями к качеству обслуживания. дисс. к. ф-м. н.МГУ им. М.В. Ломоносова 2016 год
29. Optimizing Application for NUMA. [Электронный ресурс]. URL: https://software.intel.com/sites/default/files/m/d/4/1/d/8/3-5-MemMgt_-_Optimizing_Applications_for_NUMA.pdf (дата обращения 12.04.2017)
30. TCMalloc NUMA Aware. [Электронный ресурс]. URL: https://developer.amd.com/wordpress/media/2012/10/NUMA_aware_heap_memory_manager_article_final.pdf (дата обращения 12.04.2017)

31. TCMalloc : Thread-Caching Malloc. [Электронный ресурс]. URL: http://goog-perftools.sourceforge.net/doc/tcmalloc.html (дата обращения 12.04.2017)
32. Chistopher Cantalupa, Vishwanath Venkatesan, Jeff R. Hammond, Krzysztof Czurylo, Simon Hammond. User Extensible Heap Manager for Heterogeneous Memory Platform and Mixed Memory Polices. March 18, 2015. [Электронный ресурс]. URL: http://memkind.github.io/memkind/memkind_arch_20150318.pdf (дата обращения 12.04.2017)
33. Nakul Manchanda and Karan Anand. Non-Uniform Memory Access (NUMA) // New York University.
34. Renaud Lachaize, Baptiste Lepers, Vivien Quema. MemProf: a Memory Profiler for NUMA Multicore Systems. [Электронный ресурс]. URL: https://www.usenix.org/system/files/conference/atc12/atc12-final229.pdf (дата обращения 12.04. 2017)
35. Performance Best Practices for Vmware vSphere 5.5. [Электронный ресурс]. URL: https://www.vmware.com/pdf/Perf_Best_Practices_vSphere5.5.pdf (дата обращения: 12.04.2017).
36. NUMA Support by Windows. [Электронный ресурс]. URL: https://msdn.microsoft.com/ru-ru/library/windows/desktop/aa363804(v=vs.85).aspx (дата обращения 12.04.2017)
37. Ulrich Drepper. What Every Programmer Should Know About Memory. // Red Hat Inc., November 21, 2007
38. VMware NUMA Optimization Algorithms ans Settings. [Электронный ресурс]. URL: https://pubs.vmware.com/vsphere-4-esx-vcenter/index.jsp?topic=/com.vmware.vsphere.resourcemanagement.doc_41/using_numa_systems_with_esx_esxi/c_vmware_numa_optimization_algorithms.html (дата обращения 12.04.2017)
39. Mei-Ling Chiang, Chieh-Jui Yang , Shu-Wei Tu. Kernel mechanisms with dynamic task-aware scheduling to reduce resource contention in NUMA multi-core systems. // The Journal of System and Software. 2016
40. 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
41. Вдовин П.М, Зотов И.А, Костенко В.А., Плакунов А.В., Смелянский Р.Л. Сравнение различных подходов к распределению ресурсов в центрах обработки данных // Изв. РАН.

ТиСУ. 2014. № 4. С. 71-83; Vdovin P.M., Zotov I.A., Kostenko V.A., Plakunov A.V., Smelyansky R.L. Comparing Various Approaches to Resource Allocating in Data Centers // J. of Computer and Systems Sciences Intern. 2014. V.53. No 4. P. 689-701.

42. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. В кн: Алгоритмы: построение и анализ., 3-е изд. : Пер. С англ. - М. : ООО «И. Д. Вильямс», 2013. - 1328 с. : ил. - Парал. Тит. Англ.
43. Буч Г., Рамбо Д., Джекобсон А. Язык UML. Руководство пользователя. М.: ДМК Пресс, 2004. С. 432.

Приложение А

Процедура назначения виртуальных каналов на физические ресурсы

Процедура назначения виртуальных каналов на физические взята из работы [27].

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 [42] 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.

Процедура ограниченного перебора

Процедура ограниченного перебора взята из работы [15]. 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 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:

1. Для заданного *d* сформировать все подмножества *Cs* множества физических узлов физического графа ресурсов из *d* физических ресурсов.

2. Если для некоторого множества *C* из *Cs* суммарная остаточная емкость ресурсов достаточна для назначения текущего элемента:
   a) Сохранить назначение виртуальных узлов и каналов
   b) Произвести снятие элементов *V*, которые назначены на физические ресурсы из множества C и для которых разрешена миграция.
   c) Произвести снятие всех виртуальных каналов, связанных с элементами из множества *V.*
   d) 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 2f**
   e) 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 2f**
   f) Восстанавливаем старое назначение узлов *V* и связанных с ними каналов. Удаляем множество *C* из *Cs*. Рассматриваем следующее множество *C* - переходим к **шагу 2**, если множество *Cs* пусто, то возвращаем **НЕУСПЕХ**.

Приложение Б

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, для выбора NUMA-блока — K_{NB}.

Критерий выбора очередного запроса для назначения K_G выбирает запрос, у которого максимальна взвешенная сумма, состоящая из суммарного количества запрашиваемых ресурсов в запросе и количеств политик на размещение виртуальных ресурсов.

Критерий K_V и K_P основан на критериях, рассматриваемом в работах [14, 15]. 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 → $$d(i) = \frac{\sum_{G} \sum_{e \in E, E \in G} r_{e,i}}{\sum_{ph \in Ph} r_{ph,i}},$$

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

В работах [14, 15] 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 [27] были предложены новые жадные критерии, для того чтобы исправить эти недостатки:

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

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

l – виртуальные каналы, связанные с узлом *e*

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

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

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

Throughput – текущая пропускная способность канала

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

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*

3. Жадный критерий выбора K_{NB}. Выбирается NUMA-блок с минимальной стоимостью *r(nb).*

Приложение В

Таблица В1. Исходный данные для экспериментального исследования алгоритма.

	A1			A2		
Номер теста	1			2		
Тип физической Топологии		Дерево			Дерево	
Тип запросов		Топология произвольная: Кольцо, дерево, звезда			Топология произвольная: Кольцо, дерево, звезда	
Параметры генерации Физической топологии		Сервера – 63 Коммутаторы – 63 CPU – 16 RAM – 64 Каналы – 200…300			Сервера – 63 Коммутаторы – 63 CPU – 16 RAM – 64 Каналы – 200…300	
Параметры генерации Запросов		Количество виртуальных Машин – 6 CPU – 1..2 RAM – 8..16 Каналы – 2..8			Количество виртуальных Машин – 6 CPU – 2..3 RAM – 8..16 Каналы – 2..8	
Общее Количество запросов	100	100	100	100	100	100
Размещенные Запросы	51	49	35	51	48	40
VCPU	0.488281	0.473633	0.332031	0.807617	0.768555	0.630859
RAM	0.962891	0.927002	0.662354	0.972412	0.913574	0.763916
Сетевые ресурсы	0.246788	0.335339	0.26866	0.21126	0.295865	0.27921
NUMA	1	2	4	1	2	4
Время работы, с	7	33	252	4	62	134

Таблица В2. Исходный данные для экспериментального исследования алгоритма.

	A3			A4			A5		
Номер теста	3			4			5		
Тип физической Топологии		Дерево			“Толстое дерево”			Дерево	
Тип запросов		Топология произвольная: Кольцо, дерево, звезда			Топология произвольная: Кольцо, дерево, звезда			Топология произвольная: Кольцо, дерево, звезда	
Параметры генерации Физической топологии		Сервера – 63 Коммутаторы – 63 CPU – 16 RAM – 64 Каналы – 200…300			Сервера – 63 Коммутаторы – 63 CPU – 16 RAM – 64 Каналы – 200, 400, 800, 1600			Сервера – 63 Коммутаторы – 63 CPU – 16 RAM – 64 Каналы – 200..300	
Параметры генерации Запросов		Количество виртуальных Машин – 6 CPU – 3..4 RAM – 8..16 Каналы – 2..8			Количество виртуальных Машин – 6 CPU – 4..5 RAM – 8..16 Каналы – 2..8			Количество виртуальных Машин – 6 CPU – 5..6 RAM – 8..16 Каналы – 2..8	
Общее Количество запросов	100	100	100	100	100	100	100	100	100
Размещенные Запросы	43	38	38	31	26	0	26	19	0
VCPU	0.950195	0.854492	0.736816	0.901367	0.75	0	0.908203	0.666016	0
RAM	0.840088	0.741699	0.852539	0.611328	0.513672	0	0.493896	0.367188	0
Сетевые ресурсы	0.100395	0.109474	0.127895	0.169683	0.15702	0	0.163975	0.158602	0
NUMA	1	2	4	1	2	4	1	2	4
Время работы, с	3	45	70	4	44	14	3	37	12