А.А. Спицын, Д.И. Мутин, О.Я. Кравец

ПРОЦЕССЫ МИГРАЦИИ ВИРТУАЛЬНЫХ МАШИН В ОБЛАЧНЫХ СРЕДАХ

Монография

Воронеж
Издательство «Научная книга»
2024

УДК 004.5
ББК 32.973
С 72

Рецензенты:

Ломакина Л.С., доктор технических наук, профессор, ФГБОУ ВО «Нижегородский государственный технический университет им. Р.Е. Алексеева», профессор кафедры «Вычислительные системы и технологии»;

Перепёлкин Д.А., доктор технических наук, профессор, ФГБОУ ВО «Рязанский государственный радиотехнический университет им. В.Ф. Уткина», профессор кафедры систем автоматизированного проектирования вычислительных средств

С 72 **Спицын, А.А.** Процессы миграции виртуальных машин в облачных средах: Монография/ А.А. Спицын, Д.И. Мутин, О.Я. Кравец. – Воронеж: Издательство «Научная книга», 2024. – 148 с.

ISBN 978-5-907328-32-7

Монография посвящена проблеме миграции виртуальных машин в облачных средах, связанной с реализацией иерархической стратегии балансировки нагрузки. Особенностью эвристического алгоритма планирования задач в облаке с полиномиальной по времени сложностью является соблюдение регламентных сроков исполнения задач.

Монография предназначена для научных и инженерных работников, интересующихся вопросами технической диагностики, а также может быть полезна студентам, магистрантам и аспирантам направления «Информатика и вычислительная техника».

Рис. 31. Табл. 6. Библиогр.: 228 назв.

УДК 004.5
ББК 32.973
С 72

ISBN 978-5-907328-32-7

© Спицын А.А., Мутин Д.И., Кравец О.Я., 2024

Оглавление

Общая характеристика работы

Актуальность темы. Стремительное развитие облачных сервисов породил множество методов управления ими. Большой вклад в развитие методов управления внесли Крупин А., Черняк Л., Li C., Wang Y., Mateusz G. В теоретическом плане ряд методов управления облачными сервисами сводится к оптимизации выбора физических исполнителей и порядка исполнения задач в распределенной системе внутри облака. Разумной идеей кажется модифицировать архитектуру системы управления облачными средами, расширив ее на случай нескольких уровней с учетом дополнительного миграционного слоя с использованием концепции контейнеризации.

Несмотря на глубоко развитую теорию назначений, существуют проблемы в облачных средах, связанные с алгоритмами распределения задач и виртуальных машин между физическими исполнительными устройствами в облаке. В данной области ведущие исследователи Романченко И., Самойленко А., Beegom A.A., Jennings N.R. развили множество походов, ориентированных на классические задачи распределения процессов, обладающие как правило экспоненциальной сложностью. Ключевые показатели для этих подходов, к сожалению, не учитывают то, что задачи планирования в облаке происходят в реальном масштабе времени, и алгоритмы с экспоненциальной сложностью неприемлемы. Несмотря на значительные вычислительные мощности, процессы в облаке происходят быстрее, чем решается задачи их распределения. Необходимы алгоритмы, основанные на декомпозиции больших задач на подзадачи по специальному критерию с полиномиальной по времени сложностью.

Облачные среды часто проходит через периоды локальной перегрузки после неожиданного поступления пользовательских задач. В такие периоды задачи часто пропускают свои предельные сроки. Следовательно,

необходимы иерархические или иные кластер-ориентированные стратегии балансировки нагрузки для облачных многокластерных центров обработки данных с приоритизацией локальной балансировки нагрузки сначала внутри кластера, а затем внутри центра обработки данных.

Таким образом, актуальность исследования продиктована необходимостью дальнейшего развития средств математического и программного обеспечения управления миграцией виртуальных машин в облачных средах на основе иерархической стратегии балансировки нагрузки.

Целью работы является разработка математического и программного обеспечения управления процессами миграции виртуальных машин в облачных средах на основе иерархической стратегии балансировки нагрузки на виртуальные машины.

Задачи исследования. Для достижения поставленной цели необходимо решить следующие задачи:

1. Модифицировать архитектуру системы управления облачными средами, расширив ее на случай нескольких уровней с учетом дополнительного миграционного слоя с использованием концепции контейнеризации.

2. Разработать эвристический алгоритм планирования задач в облаке, основанный на декомпозиции больших задач на подзадачи по специальному критерию с полиномиальной по времени сложностью.

3. Создать иерархическую стратегию балансировки нагрузки для облачных многокластерных центров обработки данных с приоритизацией локальной балансировки нагрузки сначала внутри кластера, а затем внутри центра обработки данных.

4. Разработать оптимизационный алгоритм решения многокритериальной задачи управления распределением задач в облачных средах на основе алгоритма роя частиц со сверточной фитнес-функцией.

5. Создать программный прототип системы управления распределением заданий в облачных средах.

Объект исследования: системы облачных вычислений.

Предмет исследования: математическое и программное обеспечение управления миграцией виртуальных машин в облачных средах на основе иерархической стратегии балансировки нагрузки.

Научная новизна работы. В работе получены следующие результаты, характеризующиеся научной новизной:

1. Модифицированная многоуровневая архитектура системы управления облачными средами, отличающаяся наличием дополнительного миграционного слоя с использованием концепции контейнеризации, обеспечивающая сокращение времени задержки, вызванной осуществлением миграции.

2. Эвристический алгоритм планирования задач в облаке с полиномиальной по времени сложностью, отличающийся декомпозицией больших задач на подзадачи по специальному критерию и обеспечивающий соблюдение регламентных сроков исполнения задач.

3. Иерархическая стратегия балансировки нагрузки для облачных многокластерных центров обработки данных, отличающаяся приоритизацией локальной балансировки нагрузки сначала внутри кластера, а затем внутри центра обработки данных и обеспечивающая уменьшение среднего времени отклика и служебных издержек межкластерной коммуникации.

4. Оптимизационный алгоритм решения многокритериальной задачи управления распределением задач в облачных средах, отличающийся применением алгоритма роя частиц со сверточной фитнес-функцией и обеспечивающий оптимальное время выполнения и надежность как компьютерных ресурсов, так и сетевых связей.

5. Структура программного прототипа системы управления распределением заданий в облачных средах, отличающаяся наличием подсистем обслуживания дополнительного миграционного слоя и декомпозицией больших задач на подзадачи, обеспечивающая выполнение требований к качеству обслуживания.

Теоретическая и практическая значимость исследования заключается в разработке математического и программного обеспечения управления миграцией виртуальных машин в облачных средах на основе иерархической стратегии балансировки нагрузки, а также информационного и программного обеспечения для экспериментальной оценки качества разработанных методов и алгоритмов.

Теоретические результаты работы могут быть использованы в проектных и научно-исследовательских организациях, занимающихся проектированием платформенно-инвариантных систем управления облачными средами в условиях штатной или нерегламентированной внешней нагрузки.

Основное содержание работы

В первой главе исследуются особенности разработки математического и программного обеспечения управления миграцией виртуальных машин в облачных средах на основе иерархической стратегии балансировки нагрузки, и анализируется современное состояние проблемы управления ими. Отмечено, что повысить эффективность управления облачными средами можно путем модификации архитектур системы управления, создания алгоритмов планирования размещением задач в виртуальных машинах и распределения виртуальных машин по физическим исполнительным устройствам. Потребовалась формализации данных задач, а также алгоритмизации их решения с учетом особенностей, отраженных на рис. 0.1. Сформулирована цель и задачи исследования.

Вторая глава посвящена разработке механизмов оптимизации миграции виртуальных машин в облачных средах с помощью эффективного алгоритма размещения.

Миграция виртуальных машин осуществляется планировщиком, установленным в вычислительных узлах с основной целью балансировки нагрузки на физические серверы. Миграция осуществляется на основе внут-

реннего дизайна облачной вычислительной системы. Время отклика, согласованное с пользователем и содержащееся в SLA, скомпрометировано из-за тяжелой обработки, необходимой для принятия решения о миграции и фактического осуществления миграции.

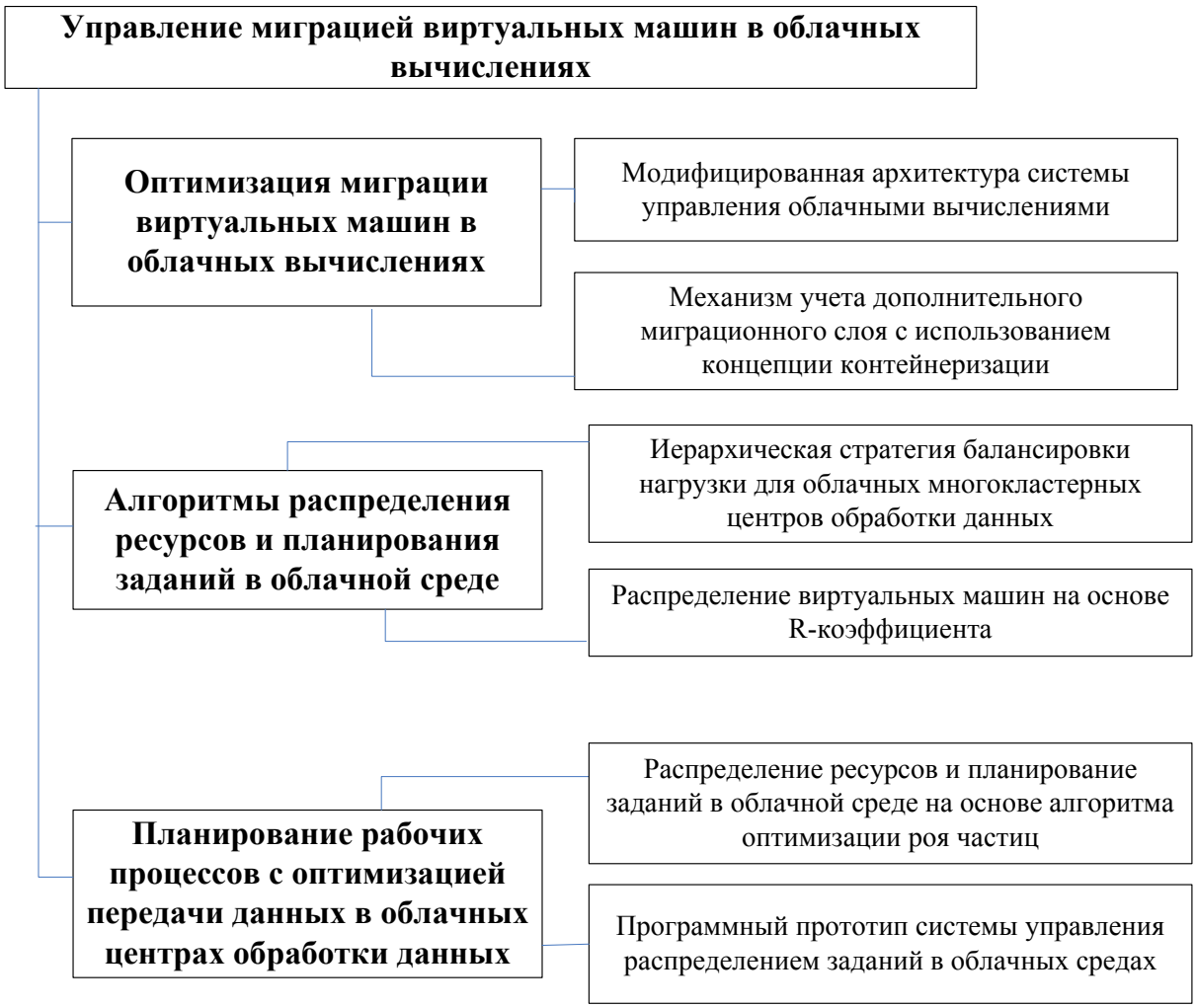


Рис. 0.1. Дизайн исследования

Алгоритм планирования вызывает программное обеспечение, расположенное на уровне OML, которое проверяет правильность запроса, и фактическая миграция осуществляется в бизнес-фазе, в которой реализуется эффективный алгоритм миграции/размещения. Здесь реализован алгоритм размещения Squirrel Whale Optimization Algorithm (S-WOA). Перенос выполняется на основе пяти компонент, которые включают перенос виртуальной машины на физическую машину (PM); контейнер для VM под та-

ким же индексом; и контейнер для VM под разными PM задач в виртуальных машин в контейнере; контейнер задачи VM. Фитнес-функция разработана и реализована для поиска наилучшего метода миграции. Фитнес-функция разработана с использованием нескольких параметров, которые включают пропускную способность, использование ресурсов и нагрузку.

На этапе трансформации осуществляется переход от виртуальной машины к контейнеру и наоборот. Производительность стратегии миграции в облаке на основе S-WOA оценивается с точки зрения количества экземпляров виртуальных машин, загрузки процессора, использования памяти, количества активированных PM и времени.

Третья глава посвящена описанию модифицированной многоуровневой архитектуры системы управления облачными средами, отличающейся наличием дополнительного миграционного слоя с использованием концепции контейнеризации, обеспечивающей сокращение времени задержки, вызванной осуществлением миграции.

При решении проблемы минимизации затрат, связанных с выполнением приложений в облачных ресурсах, создан новый алгоритм разделения задач под названием Оптимизированный по стоимости эвристический алгоритм (COHA) для облачного планировщика для оптимизации стоимости выполнения задач рабочего процесса. В этом алгоритме большие задачи, которые, скорее всего, не будут соответствовать заданным пользователем срокам, разбиваются на подзадачи, чтобы сократить продолжительность планирования. Это делается для того, чтобы все задачи выполнялись в установленные сроки.

Для сравнения результатов реализован известный алгоритм HSLJF и проведено его сравнение и предложенного COHA с использованием трех реальных приложений рабочего процесса Montage, CyberShake и Sipht. Можно сделать вывод, что сложность предложенного метода является полиномиальной. Экспериментальные результаты, полученные в результате

моделирования, показывают, что COHA может сократить продолжительность планирования всех задач по сравнению с JSLJF, тем самым снижая стоимость выполнения на 32,5% для SIPHT (рис. 3.3), 3,9% для Montage и 1,2% для CyberShake.

Облачная архитектура представляет собой набор центров обработки данных, и каждый центр обработки данных представляет собой конечное множество G кластеров Ck, связанных между собой, где каждый кластер содержит одну или несколько физических машин (PM), соединенных между собой коммутаторами, и в каждом PM работает несколько виртуальных машин (VM). Физические машины в кластерах неоднородны, поэтому нагрузка в одном кластере может быть очень высокой, в то время как в других кластерах может ничего не работать. Более высокая пропускная способность может быть достигнута, если добавить балансировку нагрузки между кластерами, а также балансировку нагрузки между физическими машинами внутри кластера.

Разработана иерархическая стратегия на двух уровнях: реализуется внутрикластерная и межкластерная балансировка: внутрикластерная балансировка нагрузки - запускается только тогда, когда некоторые менеджеры VM не могут локально сбалансировать перегрузку своих VM. Зная глобальное состояние каждого PM, менеджер может равномерно распределить глобальную перегрузку между своими физическими машинами.

Межкластерная балансировка нагрузки: на этом втором уровне выполняется глобальная балансировку нагрузки между всеми кластерами облачного центра обработки данных только в том случае, если на другом уровне не удается достичь полного баланса нагрузки. Важно отметить, что на этом уровне алгоритм всегда успешно балансирует нагрузку на все эти кластеры.

В главе 4 представлены особенности алгоритмической и программной реализации механизмов управления исполнением рабочих процессов в

облачной инфраструктуре, основанные на результатах предшествующих глав.

Цель - оптимизировать время передачи данных при минимизации времени выполнения и повышении надежности. Подход предлагается в контексте облака IaaS с несколькими центрами обработки данных, где приложения рабочего процесса должны быть запланированы для выполнения в разных центрах обработки данных.

Для оценки эффективности предложенного подхода разработана имитационная модель, основанная на инструменте CloudSim для моделирования облака с несколькими платформами центров обработки данных. Решение оценено в различных конфигурациях платформ (3dcx3vm, 3dcx5vm, 5dcx3vm и 5dcx5vm) с использованием четырех известных приложений. Время выполнения, переданные данные и общая надежность были сопоставлены с результатами, полученными алгоритмом RHEFT. Экспериментальные результаты представляют собой время выполнения для разного количества задач. Использовано четыре приложения рабочего процесса с 30 (25 для Montage), 50 (60 для SIPHT) и 100 задачами. Улучшение времени для RDPSO составляет 28-30%, уменьшение вероятности отказов – от 20% и более.

Создан оптимизационный алгоритм решения многокритериальной задачи управления распределением задач в облачных средах, отличающийся применением алгоритма роя частиц со сверточной фитнес-функцией и обеспечивающий оптимальное время выполнения и надежность как компьютерных ресурсов, так и сетевых связей.

Разработан прототип системы диспетчеризации заданий в облачных средах. Программная реализация прошла государственную регистрацию в ФИПС.

В заключении представлены основные результаты работы.

1. Проблемы управления заданиями на серверах в облачных средах

1.1. Проблемы миграции виртуальных машин

Облачные вычисления стали основной сервис-ориентированной архитектурой. Однако масштабное применение облачных вычислений предполагает огромное количество рабочих нагрузок и все большее количество задач [2.1]. Из-за существования изменяющихся нагрузок при этом некоторые вычислительные узлы перегружены, а некоторые недоиспользуются, что приводит к несбалансированному распределению нагрузки [2.2]. Следовательно, очень важно распределить нагрузки между вычислительными узлами для полного использования преимуществ облачной вычислительной системы с улучшенной удовлетворенностью пользователей [2.3].

В качестве новой преобладающей коммерческой парадигмы облачным вычислениям уделяется большое внимание как в промышленном, так и в академическом сообществах. Благодаря передовому совершенствованию облачных вычислений нескольким предприятиям и частным лицам разрешено передавать значительные данные на аутсорсинг в облако, несмотря на поддержание и строительство локальных центров обработки данных. Кроме того, пользователи облака также пользуются несколькими видами вычислительных услуг, предоставляемых публичным облаком [2.4].

Облачные вычисления определяются NIST как модель обеспечения доступа к сети по требованию, которая удобна для общего пула вычислительных ресурсов, таких как серверы, сети, приложения, хранилища и службы, которые быстро освобождаются и предоставляются с уменьшением усилий облачного провайдера или управления взаимодействием. Кроме того, облачные вычисления рассматриваются как тогдашняя парадигма ovel computing, поскольку она позволяет использовать вычислительную инфраструктуру на более чем одном уровне абстракции по другой компь-

ютерной сети или Интернету. Из-за последствий для более высокой гибкости и доступности при более низких затратах облачные вычисления являются предметом, которому уделяется большое внимание [2.5].

Услуги, предоставляемые через облачную вычислительную систему, совместно используют сеть и инфраструктурные объекты, такие как хранилища, Физические серверы и т.д. Ключевым аспектом эффективности и производительности является фактическое размещение сервисных функций, распределение памяти, пути потока данных и потоки сетевой маршрутизации [2.6].

Задача распределения облачных сервисов (CSDP) [2.7] разработана с основной целью определения размещения виртуальных функций и принятия решения о маршрутизации сетевых потоков, удовлетворяющей ресурсным возможностям, требованиям QoS и снижающей общую стоимость инфраструктуры. Этот метод не помог, когда облачные вычислительные сервисы предоставляются в качестве бэкэнд-сервисов приложению на основе Интернета Вещей, которое рассматривает отдельный уровень устройств и многие другие слои, подключенные к слоям облачных вычислений [2.8, 2.9].

На протяжении многих лет были разработаны различные методы повышения производительности облачных вычислительных систем, которые включают приоритетную консолидацию работ [2.10], синхронизацию процессов с использованием имитационных экземпляров [2.11], совместное использование ресурсов между параллельными заданиями на основе подхода группового планирования [2.12] и использование общей очереди событий между облачными многоядерными системами. Однако эти подходы приводят к серьезным узким местам, особенно к необходимости обрабатывать огромное количество потоков для моделирования работающего облака.

В литературе представлена методика federationenabled, предназначенная для повышения производительности крупномасштабного центра обработки данных. В этом случае для оптимизации размещения виртуаль-

ных машин используются генетические алгоритмы. Кроме того, алгоритмы упорядочения используются для создания размещения виртуальных машин, а методы прогнозирования вводятся для решения изменяющихся во времени прогнозируемых требований. Кроме того, динамический способ размещения виртуальных машин вводится для борьбы с перегрузкой VM. Методы размещения виртуальных машин делятся на методы обеспечения качества обслуживания (QoS) и методы, основанные на мощности.

Существующие подходы к размещению виртуальных машин разделяются на динамические и статические. Кроме того, алгоритмы прогнозирования должны быть разработаны с учетом нескольких поисковых пространств для прогнозирования. При принятии решений о миграции с использованием генетического алгоритма требуется больше времени выполнения. Большинство алгоритмов были верифицированы в режиме моделирования, и производительность этих алгоритмов в реальной среде облачных вычислений никогда не тестировалась. Интеграция новых алгоритмов с существующей системой облачных вычислений была отмечена как большое узкое место.

Существует множество миграционных стратегий, в которых вопрос контейнеризации широко открыт. Таким образом, существует требование выбора наилучшей стратегии миграции и последующего ее осуществления на основе стратегии размещения, сохраняя при этом возможность выполнения условия SLA путем выбора правильной функции установки.

Были рассмотрены восемь классических стратегий, основанных на миграции в облачных средах, а также их ограничения. Мохаммед К. Hussein et al. [2.13] разработали архитектуру размещения контейнера как сервиса (CaaS) в облаке. Эта архитектура использовала эвристику планирования, такую как Max Fit (MF) и Best Fit (BF). Здесь BF и MF были вычислены с использованием функции пригодности, которая одновременно вычисляла оставшиеся потери ресурсов как VM, так и PM. Кроме того, был вве-

ден метаэвристический подход размещения, который использовал Оптимизацию муравьиной колонии на основе наилучшего соответствия (ACO-BF). Этот метод был высокоэффективен с точки зрения использования VM и PM и минимизации числа активных PM и включенных VM, но не учитывал дополнительные вычислительные ресурсы, такие как процессорные ядра, память, хранилище и передача данных.

Rong Zhang et al. [2.14] разработали архитектуру Container-VM-PM для размещения контейнеров Docker. Проблема размещения контейнеров Docker была сосредоточена в рамках архитектуры CVP путем одновременного рассмотрения трех вовлеченных сущностей. Кроме того, рассматривалась функция пригодности для выбора PM и VM. Этот метод был более эффективным в использовании ресурса, однако консолидация контейнеров и балансировка нагрузки в рамках архитектуры CVP не рассматривались для повышения производительности системы.

Li Chunlin et al. [2.15] представили динамическое многоцелевое оптимизированное размещение реплик и методы миграции SaaS-приложений в пограничном облаке. Здесь проблема размещения реплик была решена на основе генетического алгоритма быстрой сортировки без доминирования. Кроме того, была введена модель миграции реплик для горячих точек доступа для получения отношения миграции спаривания от целевого узла к исходному. Таким образом, метод сократил время миграции и время отклика и улучшил использование сетевых ресурсов. Однако этот метод не учитывал дополнительные накладные расходы на ввод-вывод, вызванные процессом миграции.

SaadZaheer et al. [2.16] разработали подход для облегчения реализации и оценки процесса размещения алгоритмов в трехуровневом облачном центре обработки данных. Кроме того, был введен классический метод кластеризации для тестирования различных стратегий размещения процессов. Затем был установлен критерий, учитывающий локальность, чтобы

автоматически реструктурировать базовую сеть для эффективного процесса размещения. Метод не учитывал динамическую балансировку нагрузки, консолидацию процессов и адаптивные методы обучения для повышения производительности. Ruiting Zhou et al. [2.17] представил онлайн-алгоритм максимизации совокупной стоимости всех обслуживаемых кластеров. Здесь был введен одноразовый подход для определения схемы размещения данного контейнерного кластера (КК). Кроме того, был создан алгоритм первичного двойного онлайн-размещения, который использовал одноразовый подход в качестве строительного блока для принятия решений по прибытии каждого запроса СС. Этот метод достиг большей вычислительной и экономической эффективности, но все же решения на месте не принимались без опоры на знание будущих поступающих запросов.

Бо Лю и др. [2.18] разработали подход, названный мультиобъективным контейнерным планированием, основанным на многоцелевой оптимизации. Эта структура учитывает использование памяти каждого узла, использование процессора каждого узла, потребление времени передачи изображений по сети, связь между контейнерами и узлами, кластеризацию контейнеров, которые влияют на производительность приложения в контейнерах. Затем был выбран соответствующий узел для развертывания контейнеров, необходимых для выделения в процессе планирования. Следовательно, для каждого ключевого фактора был определен метрический метод, а затем для каждого фактора была установлена скоринговая функция. Наконец, объедините выходные данные каждого фактора в составную функцию для повышения производительности системы. Однако отказоустойчивость контейнеров не учитывалась в процессе обучения.

Камран и Бабар Назир [2.19] разработали подход к размещению и миграции виртуальных машин, рассмотрев требования к качеству обслуживания нескольких пользователей для снижения нарушений SLA и энергопотребления из-за недоиспользования центров обработки данных. Кроме

того, был внедрен эвристический подход к распределению энергоинформационных ресурсов для распределения задач пользователя в облачных ресурсах в облачные ресурсы, что дает минимальную энергию. Аспект отказоустойчивой миграции виртуальных машин не рассматривался для достижения лучшей производительности системы.

Anurag Satpathy et al. [2.20] представили двухуровневый подход к размещению виртуальных машин в облачных центрах обработки данных с живой миграцией. Изначально структура массового обслуживания была введена для управления и планирования огромного набора виртуальных машин. После этого для снижения энергопотребления и потерь ресурсов в центрах обработки данных был создан многоцелевой подход к размещению виртуальных машин, получивший название crow search-enabled VM placement (CSAVMP). Метод подвергается управлению центром обработки данных, что является сложной и трудоемкой задачей.

Ряд рекомендаций касался миграции виртуальных машин, но большинство рекомендаций не рассматривали различные стратегии фитнес-функции, построенной с использованием различных условий SLA.

1.2. Метаэвристические алгоритмы оптимизации в облачных средах

Облако - это распределенная и параллельная вычислительная система, состоящая из набора виртуализированных и взаимосвязанных ПК, которые динамически представляются и подготавливаются как более чем один унифицированный вычислительный ресурс на основе SLA (Service level agreement), устанавливаемого путем обсуждения между пользователями и поставщиками облачных услуг [3.1].

Облачные вычисления [3.2] - это крупномасштабная модель распределенных вычислений, которая опирается на экономический размер облачного оператора, который является динамичным, виртуализированным и

абстрактным. Основная информация об облачных средах – это способность справиться с объемом вычислений, хранилищ и различных видов услуг и платформ, которые выделяются внешним пользователям по требованию через интернет. Облачные вычисления [3.5] - это быстро развивающаяся парадигма вычислений с целью освобождения пользователей облака от управления аппаратным и программным обеспечением, информационными ресурсами и сетями и переноса этого бремени на облачный сервис.

Распределенная и параллельная система состоит из набора виртуальных и взаимосвязанных ПК, которые динамически подготавливаются и представляются как более чем один унифицированный ресурс вычислений на основе SL, настроенного путем переговоров между клиентами и поставщиком услуг. Существует [3.6] в основном 3 вида услуг, предлагаемых облаком. Во-первых, это IaaS (инфраструктура как услуга), которая предлагает пользователям облачную инфраструктуру - это инфраструктура для различных целей, а именно вычислительные ресурсы и система хранения данных. Во-вторых, платформа как услуга (PaaS), которая предлагает платформу клиентам, чтобы они могли делать свои приложения на этой платформе. В-третьих, программное обеспечение как услуга (SaaS), которая предоставляет программное обеспечение пользователям, так что пользователям не требуется устанавливать программное обеспечение на свои машины, и они могут использовать программное обеспечение непосредственно из облака.

Планирование задач [3.7] является одной из наиболее важных и существенных проблем в облачных средах, и многие исследователи пытались предсказать оптимальное решение для планирования задач на существующих ресурсах в облачном окружении. Но проблема планирования задач - это NP-полная задача. В настоящее время эвристические алгоритмы оптимизации широко используются при решении NP-полных задач. Эволюционный алгоритм используется для предсказания субоптимального

решения проблемы за значительное время вычислений. Для быстрого достижения решения используется несколько эвристических методов.

Планирование задач играет важную роль в развитии надежности и гибкости облачных систем. На рис. 1.1 показан процесс планирования задач.

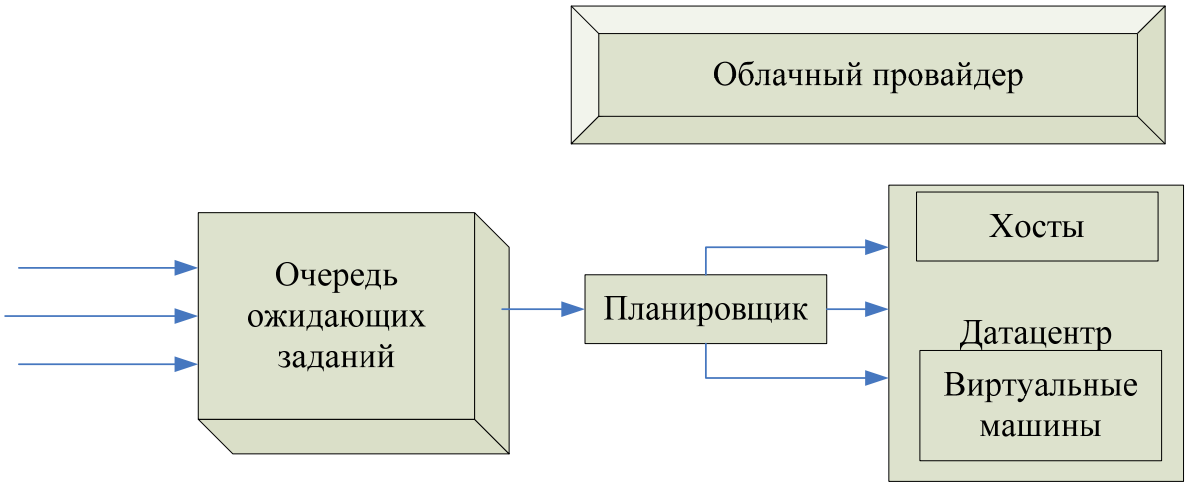


Рис. 1.1. Процесс планирования задач

Основная причина планирования задач для ресурсов в соответствии с заданным периодом времени включает в себя прогнозирование наилучшей и полной последовательности, в которой различные задачи могут быть выполнены для обеспечения удовлетворительного и наилучшего результата для пользователя. Ресурсы в любом типе облачных вычислений часто распределяются динамически в соответствии с потребностями и последовательностью выполнения задачи, что приводит к планированию задач в облаке быть динамической проблемой, где последовательность может быть полезна во время обработки задач.

Планирование задач должно быть динамическим, чтобы поток задач и пути их выполнения были неопределенными, и в то же время ресурсы были неопределенными, потому что существует несколько задач, присутствующих для совместного использования задач одновременно в одно и то же время.

Проблема планирования задач состоит [3.8] из M машин и N задач.

Каждая задача должна быть обработана одной из виртуальных машин M таким образом, чтобы в конечном итоге вся продолжительность планирования была бы сокращена.

Алгоритм планирования концентрируется на параметрах качества, а именно стоимость выполнения задач, период решения. Каждая задача может быть выполнена на одном ресурсе и не может быть приостановлена до конца. Поскольку алгоритм планирования статичен, ожидаемое время выполнения задачи j на I ресурсах считается заранее заданным. Цель алгоритма планирования состоит в том, чтобы представить каждое задание каждому ресурсу, чтобы сократить время потока и продолжительность выполнения заданий.

Выделение физических и/или виртуальных ресурсов [3.9] - это основной тип управления в облачной среде, поскольку его эффективность напрямую влияет на стоимость и производительность всей системы. Таким образом, неэффективное распределение ресурсов имеет отрицательный эффект влияния на стоимость и производительность. Основная цель распределения ресурсов состоит в том, чтобы наилучшим образом использовать ресурсы инфраструктуры и интегрировать их для достижения большей пропускной способности при решении проблем крупномасштабных вычислений.

Ресурсы вычислений в таких средах распределяются, когда потребитель посылает запрос со своими потребностями. В [3.11] исследовано три различных эвристических алгоритма, генетический алгоритм, поиск блокировок и имитационный отжиг. Существующие решения в каждом поколении оцениваются индивидами и функцией пригодности, которые имеют лучшую ценность пригодности, генерируя новые решения с помощью операторов кроссовера и мутации. Одним из новейших эвристических алгоритмов является оптимизация роя частиц.

Частицы обозначаются как группа виртуальных машин, выделенных

для выполнения задач. Каждая частица в поведении роя имеет 2 главных признака, а именно скорость v, указывающая на скорость движения, и положение x, обозначающее рекомендуемое местоположение.

Для частицы p лучшее решение известно как лучший личный опыт (pbest), в то время как среди всех частиц популяции лучшим решением является лучший групповой опыт (gbest). В любой момент времени на положение частицы влияет ее личное лучшее положение и положение другой лучшей частицы в глобальном проблемном пространстве.

Метаэвристический метод PSO (Particle Swarm Optimization) с собственным адаптивным поиском на основе методов оптимизации предложен в [3.3]. Алгоритм PSO не похож [3.4] на другие алгоритмы, основанные на популяции, а именно на генетические алгоритмы, которые не имеют прямой индивидуальной рекомбинации популяции. Алгоритм оптимизации роя частиц концентрируется на снижении общей вычислительной стоимости приложения рабочего процесса. В качестве показателя производительности используется полная стоимость выполнения приложения.

Основная цель - снизить общую стоимость выполнения рабочих процессов приложений в среде облачных вычислений. Оптимизация роя частиц на основе отображения ресурса задачи может обеспечить трехкратную экономию затрат в отличие от BRS (Лучший выбор ресурсов) на основе сопоставления для приложения рабочего процесса. Кроме того, оптимизация роя частиц уравновешивает нагрузку на вычислительные ресурсы, распределяя задачи по возможным ресурсам. Таким образом, можно сделать вывод, что планирование задач и управление ресурсами должны быть тщательно проанализированы и оптимизированы в облачной среде для достижения снижения затрат и времени, что приводит к повышению качества и надежности.

Технология виртуализации обычно используется для виртуализации отдельных серверов в различные серверы, что не только создает операци-

онную среду для платформы облачных вычислений на базе виртуальных машин, но и существенно повышает ее эффективность.

В [3.16, 3.17] исследовано QoS (качество обслуживания) на основе GHPSO (генетическая гибридная частица Swarm Optimization) для планирования приложений облачных активов. В генетическом гибридном рое частиц оптимизация мутации и гибрида наследственного алгоритма вставляется в алгоритм PSO. Результаты моделирования показывают, что генетическая гибридная оптимизация роя частиц обеспечивает более высокую производительность по сравнению со стандартным алгоритмом PSO, используемым при предельных затратах в течение заданного времени выполнения.

В [3.18, 3.19] предложен улучшенный алгоритм для выполнения субоптимизации или оптимизации планирования облака. MGA (улучшенный генетический алгоритм) используется для автоматизированного подхода к бронированию. Тесты подтверждают, что скорость MGA почти вдвое превышает традиционную.

Генетический алгоритм часто лучше формирует стратегии планирования и использования стоимости активов, чем инфраструктура с открытым исходным кодом в качестве облачных фреймворков сервиса. В [3.20] усовершенствован алгоритм планирования, основанный на затратах, для эффективного представления задач возможным исполнителям в облаке. Этот алгоритм планирования измеряет как производительность вычислений, так и стоимость активов, а также повышает соотношение коммуникации и вычислений, собирая клиентские задачи в соответствии с возможностями обработки конкретного облачного актива и отправляя собранные задания в актив.

В [3.21] предложена оптимизация роя частиц для построения алгоритма решения проблемы балансировки нагрузки в виртуальной среде. Целью данного исследования является ограничение времени выполнения за-

нятий. В [3.22, 3.23] оптимизация роя частиц предлагается для уменьшения среднего времени выполнения работ и повышения коэффициента доступности активов.

В [3.24, 3.25] адаптивный алгоритм управления заданиями предлагается использовать для расположения экземпляров виртуальных машин на расположении облачных физических активов и уменьшать деградацию системы.

В [3.26] предложен планировщик для регистрации размещения виртуальной машины в соответствии с текущей нагрузкой физических активов и ограничения используемых ресурсов. Такое ограничение может привести к массовому снижению затрат. Кроме того, чем больше потребляемая мощность, тем больше рассеивается тепло, и таким образом возрастает вероятность аппаратных сбоев.

Оптимизация роя частиц используется для формирования групповых и глобальных задач оптимизации. Динамически варьируемая оптимизация роя частиц инерционного веса быстро создает и удовлетворяет почти оптимальным решениям. Преимущества оптимизации роя частиц заключаются в быстрой сборке и большей точности выходных данных. Оптимизация роя частиц лучше всего применима к глобальному поиску.

1.3. Оптимизация по стоимости как один из подходов к управлению облачными системами

Облачные вычисления - это интернет-платформа, которая обеспечивает предоставление цифровых услуг различными поставщиками облачных услуг по требованию, что зависит от ограничений качества обслуживания (QoS) [3.31-3.33]. В последние годы наблюдается стремительный рост развития облачных технологий. Это связано с их огромными преимуществами, включая 1) снижение первоначальных капитальных затрат на

оборудование, программное обеспечение, хостинг и развертывание; 2) доступность; 3) независимость от местоположения; 4) гибкость и 5) гибкость рынка [3.34]. Облако обеспечивает:

- программные услуги, которые доступны через сторонние через Интернет сервисы, называют как «программное обеспечение как услуга» (SaaS);

- хранения данных, сетевой инфраструктуры и виртуализации - известны как «инфраструктура как услуга» (IaaS);

- аппаратные и программные средства, доступные через интернет, которые обычно называют «платформа как услуга» (PaaS) [3.35].

Выгоды от облачных вычислений породили множество изменений, приведших к развертыванию высокотехнологичных платформ, таких как интернет вещей и мобильные периферийные вычисления. Эти платформы с помощью облачных вычислений способны создать умную среду, которая обеспечивает умное здравоохранение, города, транспорт, жилье, энергию, жизнь и многое другое, чтобы облегчить образ жизни. Облачные вычисления могут моделировать различные приложения как набор задач рабочего процесса [3.36].

Эти задачи рабочего процесса имеют набор ребер, которые представляют зависимости данных между рабочими процессами. Другими словами, ребро указывает, что последующая задача рабочего процесса не может быть запущена до тех пор, пока не будет завершена предыдущая задача рабочего процесса [3.37-3.40].

С помощью рабочих процессов пользователи могут представлять свои вычислительные потребности удобно для поиска, переформатирования и анализа данных [3.41].

Рабочие процессы используются в различных областях, таких как биоинформатика, астрофизика, моделирование и прогнозирование катастроф [3.42-3.44]. Многие исследователи разработали различные виды науч-

ных алгоритмов планирования рабочих процессов для различных сред: от однородных кластеров с ограниченным набором ресурсов до крупномасштабных сетей сообществ и до самой последней парадигмы ресурсоемких облачных вычислений [3.45].

Эти алгоритмы разработаны с целью повышения пропускной способности и производительности системы.

Существуют «бесконечные «облачные ресурсы, доступные через сеть облачным пользователям с оплатой за использование [3.46, 3.47]. Однако ежедневно растущее число облачных пользователей и новые технологии в облаке затрудняют обеспечение оптимального использования этих ресурсов [3.31]. Кроме того, приложения рабочего процесса являются приложениями больших данных и часто требуют много часов, чтобы закончить выполнение из-за их природы и размера данных. Много работы было сделано другими исследователями, чтобы найти оптимальное решение этой проблемы с помощью эвристических алгоритмов. Тем не менее, проблема все еще существует. Это связано с тем, что большинство эвристик в значительной степени зависят от приоритета задания без учета продолжительности планирования задания. Следовательно, очень трудно достичь оптимального решения с помощью современных эвристических алгоритмов.

Основное внимание уделено решению проблемы планирования рабочего процесса. Представлен эвристический алгоритм планирования под названием Cost Optimized Heuristic Algorithm (COHA) для планирования рабочих процессов в облачных средах. В этом алгоритме решаются большие задачи, которые, вероятно, не укладываются в требуемые сроки из-за их размера. Как следствие, данные разбиваются на подзадачи и ставятся на Виртуальные машины для исполнения. Целью предложенного алгоритма является снижение приоритетности планирования в предыдущих эвристических методах и минимизация затрат на выполнение задач, а также возможность завершения всех задач раньше установленных сроков.

Планирование рабочих процессов в облачных средах является сложной задачей. Она была определена многими исследователями как NP-полная задача [3.32, 3.38-3.43]. Эта проблема была исследована многими исследователями, чтобы найти оптимальное решение.

Тем не менее проблема все еще существует. Например, в [3.44] была описана NP-полная задача. Исследователи трансформируют невыпуклое ограничение для многих линейных ограничений с использованием линеаризации и переформулировки на основе эвристических методов.

В [3.47] нечеткая доминирующая сортировка основана на гетерогенном подходе. Представлен алгоритм early-Finish-Time (FDHEFT). Этот подход состоит из двух фаз, приоритизации задач и выбора экземпляра. На этапе приоритизации задач алгоритм вычисляет приоритеты каждой задачи и ставит их в очередь в невозрастающем порядке, причем большие значения ранжируются первыми. Этап выбора экземпляра сортирует и выбирает все задачи на основе их нечеткого доминирования приоритетов, минимизации затрат и периода исполнения.

Кроме того, в [3.55] предложен общий рамочный эвристический алгоритм для многоцелевого статического планирования рабочих процессов в гетерогенных вычислительных средах, называемый многоцелевым алгоритмом планирования списков (MOLS). Алгоритм пытается найти подходящее решение Парето, развертывая две стратегии, то есть максимизируя расстояние до вектора ограничений пользователя для доминирующих решений и минимизируя его в противном случае.

Хотя предложенный алгоритм способен производить лучшие результаты в стоимости, он не эффективен при работе с небольшим количеством задач и процессоров.

Кроме того, в [3.51] определена задержка занятия виртуальных машин как одна из основных проблем планирования рабочих процессов в среде облачных вычислений. Предложена экономически эффективная стратегия

планирования с учетом крайних сроков (CEDA), которая позволяет оптимизировать общее время выполнения задач рабочего процесса и экономические затраты при соблюдении крайних сроков. Метод выбирает задачу рабочего процесса с самым высоким значением возрастающего ранга на каждом шаге и отправляет ее в самый дешевый экземпляр для уменьшения стоимости. Кроме того, время ожидания использовалось для планирования других задач, чтобы еще больше снизить накладные расходы.

Другие исследователи [3.56] представили набор экономически эффективных алгоритмов планирования рабочего процесса. Были представлены методы обработки приложений рабочего процесса больших данных для удовлетворения ограничений крайних сроков при снижении затрат. Предложенные методы использовали взвешенные значения ресурсов для снижения затрат на выполнение рабочего процесса. Однако рассматривался только процессор как единственная стоимость обработки задач, и игнорировались другие не менее важные ресурсы, такие как стоимость хранения, передачи и т.д. Игнорирование этих важных элементов затрат не даст реальной стоимости планирования задачи рабочего процесса в распределенной облачной среде.

В [3.57] представлена гибридная схема планирования задач под названием HSLJF. Предложенная схема представляет собой комбинацию кратчайшего первого задания (SJF) и алгоритма Longest Job First (LJF). Она учитывает продолжительность задачи и нагрузку на ресурсы. Схема сортирует задачи в две очереди. Первая очередь для коротких задач, а вторая - для длительных задач. Использование ресурсов максимизируется. Однако метод не учитывает стоимость выполнения задач на облачных ресурсах.

Можно сделать вывод, что планирование рабочих процессов приложений на облачных ресурсах для снижения затрат на выполнение является сложной задачей. Для оптимизации затрат на выполнение требуются более надежные алгоритмы планирования.

1.4. Использование двухуровневых динамических моделей распределения нагрузки с глобальным и локальным менеджерами

Основываясь на пространственном распределении узлов, мы можем найти три типа алгоритмов, которые определяют, какой узел отвечает за балансировку нагрузки в среде облачных вычислений.

В технике централизованной балансировки нагрузки все решения о распределении и планировании принимаются одним узлом, в технике распределенной балансировки нагрузки ни один узел не отвечает за принятие решения о выделении ресурсов или планировании задач; каждый узел в сети поддерживает локальную базу знаний для обеспечения эффективного распределения задач в статической среде и перераспределения в динамической среде, и, наконец, тип алгоритма, принятого здесь, иерархическая балансировка нагрузки включает различные уровни облака в решение о балансировке нагрузки. Такие методы балансировки нагрузки в основном работают в режиме master slave. Они могут быть смоделированы с использованием древовидной структуры данных, в которой каждый узел в дереве сбалансирован под наблюдением своего родительского узла. Мастер или менеджер могут использовать легкий процесс агента для получения статистики подчиненных узлов или дочерних узлов. На основе информации, собранной родительским узлом, принимается решение о подготовке или планировании [3.73].

Одной из проблем алгоритмов балансировки нагрузки являются накладные расходы, которые определяют объем накладных расходов, связанных с реализацией системы балансировки нагрузки. Он состоит из накладных расходов, связанных с затратами на миграцию виртуальных машин или коммуникационными затратами [3.85]. Хорошо разработанный алгоритм балансировки нагрузки должен снизить накладные расходы. Основной целью методов балансировки нагрузки является ускорение выполнения приложений на ресурсах, рабочая нагрузка которых изменяется во

время выполнения непредсказуемым образом [3.86]. Следовательно, важно определить метрики для измерения рабочей нагрузки ресурсов. В литературе было предложено несколько индексов нагрузки, таких как длина очереди ЦП, средняя длина очереди ЦП, загрузка ЦП и т. д.

Успех алгоритма балансировки нагрузки зависит от стабильности количества сообщений (небольшие накладные расходы), среды поддержки, низкой стоимости обновления рабочей нагрузки и короткого среднего времени отклика, которое является значимым измерением для пользователя [3.87]. Также важно измерить стоимость связи, вызванную операцией балансировки нагрузки.

Существует довольно много исследований, проведенных в области решения балансировки нагрузки. Большинство из них сосредоточены на балансировке нагрузки для одного уровня. Одной из сложных проблем планирования в облачных центрах обработки данных является учет рабочей нагрузки в облачной архитектуре различного уровня и интегральных характеристик кластеров и хостинговых физических машин, входящих в ее состав. В отличие от традиционных алгоритмов планирования балансировки нагрузки, которые рассматривают только виртуальные машины с одним фактором, таким как загрузка ЦП, наша предлагаемая модель рассматривает ЦП и память, интегрированные как для физических машин (PM), так и для кластеров.

В работе [3.75] введен динамический и интегрированный алгоритм планирования ресурсов (DAIRS) для балансировки виртуальных машин в облаке. Этот алгоритм рассматривает процессор, память и пропускную способность сети как интегрированный ресурс с весами. Они также разработали новую метрику, средний уровень дисбаланса всех хостов, чтобы оценить производительность при планировании нескольких ресурсов. DAIRS - один из самых ранних алгоритмов, который исследовал множество типов ресурсов и рассматривал их как интегрированную ценность.

Главный недостаток DAIRS заключается в том, что он игнорирует коммуникационные издержки миграции. Малхотра и др. [3.76] предложил фреймворк, основанный на интеллектуальных агентах на двух уровнях модели облачных вычислений. Первый из них находится на уровне центра обработки данных, а второй - на глобальном уровне. Хао и др. [3.77] предложил схему балансировки нагрузки, работающую на трех уровнях: Центр Обработки Данных, хост и обрабатывающие элементы. Балансировка облака контролируется физическим ресурсным объектом, включая центры обработки данных, хосты и PEs, используя минимальное значение стандартного отклонения. Стандартное отклонение нагрузки может быть больше или меньше, чем до планирования. Ван и др. [3.78] предложил балансировку нагрузки в трехуровневой архитектуре облачных вычислений. Третий уровень - это сервисный узел, который используется для выполнения подзадачи. Второй уровень - это Диспетчер служб, который обычно делит задачу на несколько логически независимых подзадач. Алгоритм планирования баланса нагрузки Min-Min (LBMM) берет за основу характеристику алгоритма планирования Min-Min. Но самая большая слабость алгоритма планирования Min-Min заключается в том, что он учитывает только время завершения каждой задачи на узле, но не учитывает рабочую нагрузку каждого узла. Таким образом, LBMM улучшит несбалансированность нагрузки Min-Min и эффективно сократит время выполнения каждого узла. Многоуровневая иерархическая топология сети может снизить стоимость хранения данных. Другой кластерный алгоритм балансировки нагрузки в облачных средах предложен в работе [3.79] и предназначен для нескольких ведущих и нескольких ведомых архитектур (MMMS), где сеть делится на кластеры с использованием алгоритма кластеризации таким образом, что каждый узел (ведущий) в сети принадлежит ровно одному кластеру. Новые узлы, входящие в сеть, назначаются существующему кластеру или новому кластеру в случае, если кластер полностью занят на основе

используемой стратегии кластеризации.

Каждый кластер имеет по крайней мере один узел Межкластерной связи (ICC). Таким образом, масштабируемость архитектуры зависит от метода кластеризации. Кластеризация выполняется на начальных этапах инициализации сети. Типы узлов и их функции следующие. Подчиненный- это вычислительный элемент сети. Обработка задач осуществляется в этих элементах. Каждый исполнитель напрямую связан только с одним задающим узлом. Предлагаемый алгоритм балансировки нагрузки делится на две части в зависимости от типа узлов и связи между ними. Эти части называются: (1) распределение нагрузки между ведущими устройствами; и (2) распределение нагрузки от ведущего устройства к ведомому. Ведущий - это вычислительная машина, которая решает политику балансировки нагрузки задач между подчиненными и выбирает подчиненный узел для выполнения задач. В данной работе предлагается модель облачной вычислительной среды, в которой разрабатывается иерархическая стратегия балансировки нагрузки. Характеристики предлагаемой стратегии можно резюмировать следующим образом:

1) это стратегия балансировки на уровне физических машин в первую очередь;

2) переход к балансировке нагрузки на уровне кластера, чтобы снизить затраты на связь, если система находится в состоянии насыщения.

Предлагается динамическая модель с глобальным менеджером на более высоком уровне и локальным менеджером на следующем. Это метод, который может быть использован для повышения производительности облачных вычислений путем балансировки рабочей нагрузки между всеми узлами облака с максимальным использованием ресурсов, что, в свою очередь, снижает накладные расходы и затраты на связь.

1.5. Проблемы планирования рабочих процессов с оптимизацией передачи данных в облачных центрах обработки данных

Облака типа «Infrastructure as a service» (IaaS) предлагают огромные возможности для решения масштабных научных задач. Выполнение рабочих процессов в таких средах может быть дорогостоящим по времени, если они не запланированы должным образом. Хотя планирование рабочих процессов в облаке широко изучено, большинство подходов фокусируется на двух требованиях пользователя к качеству обслуживания, а именно на времени выполнения и затратах. Необходимо учитывать и другие важные особенности облачных вычислений, гетерогенные облачные центры обработки данных. Цель - оптимизировать время передачи данных при минимизации времени выполнения и повышении надежности. Основываясь на моделировании, результаты показывают значительное улучшение с точки зрения времени выполнения, передаваемых данных и надежности по сравнению с проверенным методом HEFT (гетерогенное самое раннее время выполнения) для эталонных рабочих процессов.

Требования к рабочим процессам приложений в области данных и вычислений предполагают более высокую производительность вычислительной среды для того, чтобы выполнять их в разумные сроки и без сбоев. Проблема планирования рабочего процесса является NP-полной [28] и на протяжении многих лет широко изучалась в контексте распределенных систем, главным образом в грид-средах. Однако с появлением инфраструктуры облачных вычислений планирование научных рабочих процессов по-прежнему остается фундаментальной проблемой. Облачные вычисления - это модель обеспечения доступа к динамической сети по требованию к пулу общих ресурсов, который может быть быстро обеспечен [4.19] и который затрудняет удовлетворение некоторых ограничений качества обслуживания (QoS). Ограничения качества обслуживания используются в задачах планирования рабочих процессов в рамках сети [4.2, 4.5, 4.16, 4.20] или

облачной системы [4.6, 4.7, 4.13, 4.21, 4.23]. Среди всех этих ограничений - время выполнения и стоимость - это два основных ограничения, которые широко изучаются в [4.31].

Однако другие ограничения QoS, такие как надежность, могут быть полезны из-за их влияния на проблемы планирования рабочих процессов для облачных систем для достижения приемлемой производительности.

В облачных системах пользователи платят за использование облачных ресурсов. Таким образом, они более требовательны к производительности и надежности. Однако вычислительные ресурсы и сети не являются безотказными, и любой тип сбоя может иметь решающее значение для выполнения приложений.

Использование надежного алгоритма планирования может справиться с такими сбоями, но известно, что такое планирование – это NP-полная задача [4.12]. Именно по этой причине были предложены для решения этой проблемы многие эвристики [4.8, 4.9, 4.10, 4.12, 4.17] и метаэвристики, такие как генетический алгоритм (GA) [4.23, 4.25], оптимизация роя частиц (PSO) [4.2, 4.6] и оптимизация муравьиным алгоритмом (ACO) [4.5, 4.16].

К сожалению, большинство исследований основано только на модели надежности ресурсов, которая не подходит для реальных облачных систем и, следовательно, пренебрегает неоднородностью ресурсов и надежностью сетевых связей в основном в нескольких облачных центрах обработки данных. Более того, большинство подходов не учитывают дополнительное время, затрачиваемое на передачу данных между задачами, которое может повлиять на производительность всего приложения. Действительно, в нескольких облачных центрах обработки данных ресурсы развертываются в разных регионах, и поэтому время передачи данных становится значительным и непосредственно влияет на время отклика приложений рабочего процесса.

Фактически облачные сервисы, которые полагаются на виртуальные машины (VM), расположены в нескольких распределенных центрах обработки данных, что требует учета времени передачи данных и надежности сети.

В работе предлагается стратегия, ориентированная на надежность, основанная на дискретном PSO (DPSO) для планирования рабочих процессов в нескольких облачных центрах обработки данных, где надежность определяется совместно вычислительными ресурсами и сетевыми связями. Подход уделяет внимание минимизации времени выполнения рабочего процесса, включая как время выполнения задачи, так и время передачи данных, одновременно повышая общую надежность приложений.

Предыдущие работы были выполнены для оптимизации передачи данных и надежности в распределенных средах, таких как сети. Рассмотрим существующие исследования, связанные с оптимизацией передачи данных и повышением ограничений надежности в распределенных вычислительных системах, таких как грид- и облачные инфраструктуры.

В работах [4.8, 4.9, 4.12] рассматривается проблема оптимизации надежности с учетом времени передачи данных. В [4.12] используется список планирования, эвристика называется BSA (Bi-objective Scheduling Algorithm) - алгоритм планирования параллельных приложений на гетерогенных распределенных вычислительных системах, учитывающий не только время выполнения, но и вероятность отказа приложения. В [4.8] представлены две двухцелевые эвристики планирования для минимизации времени и повышения надежности приложений в гетерогенных вычислительных системах. Первой из них было усовершенствование традиционного алгоритма динамического планирования уровня (DLS) [4.24], называемого двухобъектным динамическим планированием уровня (BDLS). Второй был основан на генетических алгоритмах и называется двухобъектным генетическим алгоритмом (BGA), оптимизирующим те же критерии, что и

первый. Для независимых задач был представлен двухкритериальный алгоритм планирования [4.9], оптимизирующий время выполнения и надежность в гетерогенных системах. Авторы показали, что задачи должны быть сопоставлены с ресурсом таким образом, чтобы произведение времени выполнения задачи и частоты отказов ресурса можно было свести к минимуму. Затем эта идея была использована для улучшения алгоритма HEFT [4.27], чтобы рассмотреть надежность, дающую эвристику reliable-HEFT.

В [4.5] рассмотрены надежность, время и стоимость в качестве параметров QoS при планировании рабочих процессов в сетях. Предложенный алгоритм оптимизировал один параметр, удовлетворяя ограничениям двух других. Алгоритм основан на ACO, где были определены и выбраны семь эвристик по значениям феромонов. После этого в [4.6] представлен набор базовых версий дискретного PSO (S-PSO) для планирования облачных рабочих процессов с учетом следующих ограничений QoS пользователей: крайний срок, бюджет и надежность. Эти ограничения оптимизируются отдельно, поскольку пользователям рекомендуется указать один предпочтительный QoS в качестве цели оптимизации. Однако в этой работе не было дано никакой спецификации облачной модели. Кроме того, не было смоделировано ограничение надежности. Оно было просто указано как пороговое значение (например, быть не меньше определяемой пользователем переменной MinReliability), и поэтому надежность передачи данных не была определена. Напротив, в [4.13] был использован алгоритм PSO для планирования рабочих процессов в облаке для повышения надежности там, где рассматривается надежность передачи данных. Но, к сожалению, надежность облачных ресурсов и передачи данных были выражены в виде дискретных уровней, которым недостает точности (значения в интервале [1, 5]).

В [4.18] исследовано влияние надежности выполнения рабочего процесса на его стоимость в распределенных вычислительных системах, а за-

тем предложен алгоритм под названием reliability for profit assurance (RPA) для планирования рабочего процесса, который включает в себя схему репликации с учетом затрат для повышения надежности. Подходы к репликации задач часто используются для борьбы с отказами ресурсов. Однако реплики, созданные для повышения надежности, часто приводят к неоправданной трате ресурсов. Принятые решения не учитывали надежность каналов связи.

Основанный на репутации генетический алгоритм look-ahead (LAGA) был представлен в [4.29] для крупномасштабных распределенных систем. LAGA был оптимизирован с точки зрения времени выполнения и частоты отказов. Время выполнения использовалось для получения порядка задач в каждом поколении, а затем учитывалась наименьшая частота отказов для выбора ресурса в операции мутации. Авторы LAGA сосредоточились на интенсивных компьютерных приложениях, где надежность связи и время связи между задачами не были смоделированы.

В некоторых работах поддержание надежности связано с потреблением энергии. В [4.10] предложен метод многоцелевого планирования списков (MOLS) для выполнения рабочих процессов в распределенных вычислительных системах. Использована оптимальность по Парето для достижения четырех целей: время выполнения, финансовые затраты, энергопотребление и надежность. В [4.11] предложен алгоритм улучшения надежности на основе времени (DRB-FTSA-E), который оптимизирует энергию и надежность при соблюдении временных ограничений в гетерогенной системе.

Управление надежностью систем привлекает внимание многих исследователей [4.16, 4.21, 4.23, 4.30, 4.32, 4.34]. В [4.16] предложен алгоритм планирования рабочего процесса на основе ограничений для максимизации надежности. Созданы три новые эвристики планирования рабочего процесса, основанные на муравьином алгоритме. Совокупность этих эв-

ристик сводила к минимуму нарушения ограничений по времени, стоимости и надежности. Для простоты предполагалось, что сеть надежна, но это нереально в среде нескольких облачных центров обработки данных. В [4.30] представлен алгоритм развертывания приложений рабочего процесса в объединенных облаках. Алгоритм способен оценить наиболее надежную схему, отвечающую требованиям безопасности приложения и оптимизирующую его стоимость. В [4.21] использован адаптивный и своевременный алгоритм планирования для поддержки отказоустойчивости. Надежность сети не моделировалась. Генетический алгоритм был предложен в [4.34] для оптимизации энергопотребления и надежности параллельного приложения, выполняемого на гетерогенных процессорах кластера, но надежность каналов связи не учитывалась. В [4.23] используется NSGA-II с метаэвристическим алгоритмом для многокритериального планирования рабочего процесса в облаке. Облачная модель представляет собой единый сайт, где пропускная способность должна быть равномерной, а надежность ссылок полностью игнорировалась. В [4.32] предложена эвристика максимизации надежности с энергетическим ограничением (MREC) для каждой задачи параллельного приложения на гетерогенных распределенных системах. Надежность была выражена на основе частоты отказов процессора, на который была возложена задача, но надежность сети не рассматривалась, несмотря на то, что была принята ресурсная модель для распределенных систем.

Наконец, многие исследования были сосредоточены на передаче данных независимо от ограничения надежности. Например, в [4.1] предложена адаптивная эвристика временных затрат для обработки базовых рабочих процессов с наименьшими затратами. Стратегия размещения данных на основе кластеризации k-средних в сочетании с подходом многоуровневой репликации задач используется в [4.33]. Цель состояла в том, чтобы свести к минимуму общую передачу данных между несколькими

центрами обработки данных облака без учета ограничения надежности. Впоследствии в [4.4] предложен подход, который разделял графы для выполнения интенсивного рабочего процесса в распределенных центрах обработки данных, что оптимизировало общую стоимость передачи данных.

За исключением [4.6, 4.21, 4.23], все исследования были реализованы в едином облачном центре обработки данных. Другие исследования были выполнены либо в распределенных компьютерных системах, кластерах или в сети платформ, а предлагаемый подход смоделирован и проверен на нескольких облачных центрах обработки данных.

1.6. Выбор инструментальных средств исследования и реализации

Разработка программы ведётся на языке Java с использованием комплекта разработчика приложений Oracle Open Java Developer Kit 14. Для реализации используются среда выполнения Apache NetBeans 12, а также библиотеки JavaFX, CloudSim и Apache Maven.

Поскольку использование реальных испытательных стендов ограничивает эксперименты до масштабов испытательного стенда и делает воспроизведение результатов чрезвычайно трудным делом, альтернативные подходы к тестированию и экспериментам используют разработку новых облачных технологий.

Подходящей альтернативой является использование инструментов моделирования, которые открывают возможность оценки гипотезы до разработки программного обеспечения в среде, где можно воспроизводить тесты. На стороне поставщика среды моделирования позволяют оценивать различные виды сценариев аренды ресурсов при различных распределениях нагрузки и цен. Такие исследования могут помочь поставщикам оптимизировать стоимость доступа к ресурсам с акцентом на повышение прибыли. В отсутствие таких платформ моделирования клиенты и поставщики

облачных услуг вынуждены полагаться либо на теоретические и неточные оценки, либо на методы проб и ошибок, которые приводят к неэффективной производительности услуг и получению прибыли.

Основная цель CloudSim - предоставить обобщенную и расширяемую среду моделирования, которая позволяет беспрепятственно моделировать, моделировать и экспериментировать с появляющимися инфраструктурами облачных вычислений и службами приложений. Используя CloudSim, исследователи и отраслевые разработчики могут сосредоточиться на конкретных проблемах проектирования систем, которые они хотят исследовать, не беспокоясь о деталях низкого уровня, связанных с облачными инфраструктурами и службами.

Среди функций CloudSim:

– поддержка моделирования и симуляции крупномасштабных центров обработки данных облачных вычислений;

– поддержка моделирования и симуляции хостов виртуализированных серверов с настраиваемыми политиками для предоставления ресурсов хоста виртуальным машинам;

– поддержка моделирования и симуляции контейнеров приложений;

– поддержка моделирования и симуляции энергозависимых вычислительных ресурсов;

– поддержка моделирования и симуляции топологий сетей центров обработки данных и приложений для передачи сообщений;

– поддержка моделирования и симуляции федеративных облаков;

– поддержка динамического ввода элементов моделирования, остановки и возобновления моделирования;

– поддержка пользовательских политик для выделения хостов виртуальным машинам и политик для выделения ресурсов хоста виртуальным машинам.

1.7. Постановка задач работы

Проведенный анализ позволил сформулировать цель исследования.

Целью работы является разработка математического и программного обеспечения управления миграцией виртуальных машин в облачных средах на основе иерархической стратегии балансировки нагрузки.

Для достижения поставленной цели необходимо решить задачи:

1. Модифицировать архитектуру системы управления облачными средами, расширив ее на случай нескольких уровней с учетом дополнительного миграционного слоя с использованием концепции контейнеризации.

2. Разработать эвристический алгоритм планирования задач в облаке, основанный на декомпозиции больших задач на подзадачи по специальному критерию с полиномиальной по времени сложностью.

3. Создать иерархическую стратегию балансировки нагрузки для облачных многокластерных центров обработки данных с приоритизацией локальной балансировки нагрузки сначала внутри кластера, а затем внутри центра обработки данных.

4. Разработать оптимизационный алгоритм решения многокритериальной задачи управления распределением задач в облачных средах на основе алгоритма роя частиц со сверточной фитнес-функцией.

5. Создать программный прототип системы управления распределением заданий в облачных средах.

2. Оптимизация миграции виртуальных машин в облачных средах с помощью эффективного алгоритма размещения

2.1. Проблемы оптимизации и ее особенности

Как правило, архитектура системы облачных вычислений содержит несколько уровней, которые включают аппаратный уровень, уровень инфраструктуры, уровень платформы и уровень услуг.

Каждый слой выполняет различные виды операций и предоставляет различные виды услуг.

Миграция виртуальных машин осуществляется планировщиком, установленным в вычислительных узлах с основной целью балансировки нагрузки на физические серверы. Миграция осуществляется на основе внутреннего дизайна облачной вычислительной системы. Время отклика, согласованное с пользователем и содержащееся в SLA, скомпрометировано из-за тяжелой обработки, необходимой для принятия решения о миграции и фактического осуществления миграции.

В отличие от традиционной виртуальной машины (VM), контейнер-это новая облегченная технология виртуализации, которая работает на уровне операционной системы для инкапсуляции задачи и ее библиотечных зависимостей для выполнения.

В данной работе представлена модифицированная многоуровневая архитектура, рассматривающая добавление еще одного слоя OML (Оптимизированного миграционного слоя), отвечающего за оптимизацию процесса миграции с акцентом на сокращение времени задержки, вызванной выполнением миграции. Дополнительный слой использует концепцию контейнеризации для оптимизации и сокращения времени задержки, вызванной осуществлением миграции. В слое OML реализован эффективный алгоритм размещения. OML включает в себя определенные компоненты, такие как проверка, бизнес-процесс, трансформация и фаза развертывания.

Компонент, ответственный за выполнение миграции, помещается на карантин путем внесения изменений в файлы конфигурации. OML разработан с использованием четырех фаз, которые включают в себя валидацию, бизнес-фазу и фазу трансформации.

Алгоритм планирования вызывает программное обеспечение, расположенное на уровне OML, которое проверяет правильность запроса, и фактическая миграция осуществляется в бизнес-фазе, в которой реализуется эффективный алгоритм миграции/размещения. Здесь реализован алгоритм размещения, получивший название Squirrel Whale Optimization Algorithm (S-WOA). Перенос выполняется на основе пяти условий, которые включают виртуальной машины на физическую машину (PM), контейнер для VM под таким же гостем, и контейнер для VM под разными PM задач в виртуальных машин в контейнере, и контейнер задачи VM. Фитнес-функция разработана и реализована для поиска наилучшего метода миграции. Фитнес-функция разработана с использованием нескольких параметров, которые включают пропускную способность, использование ресурсов и нагрузку.

На этапе трансформации осуществляется переход от виртуальной машины к контейнеру и наоборот. Производительность стратегии миграции в облаке на основе S-WOA оценивается с точки зрения количества экземпляров виртуальных машин, загрузки процессора, использования памяти, количества активированных PMS и времени. Предложенный метод S-WOA обеспечивает максимальную загрузку процессора 0,483, максимальную память 0,523 и минимальное время 4,99 с.

Проблемы, с которыми сталкиваются традиционные стратегии, рассматриваются ниже:

Метод Container as a Service (CaaS) становится очень заметным видом модели облачного сервиса. Однако размещение контейнера на виртуальной машине - это классическая проблема планирования. Существую-

щие исследования отдельно сосредоточены либо на размещении VM на PM или контейнере, либо только на задачах без контейнеризации, размещении на VM. Хотя этот метод приводит к чрезмерно используемым или недоиспользуемым PM, а также к чрезмерно используемым или недоиспользуемым VM [2.13].

В [2.14] для размещения VM разработан метод размещения контейнеров. Здесь Docker - это зрелая схема контейнеризации, используемая для выполнения виртуализации на уровне операционной системы. Одна из открытых проблем в облаке заключается в том, как правильно выбрать виртуальную машину для инициализации контейнера, что аналогично обычному вопросу размещения виртуальной машины в направлении PMs. Тем не менее, основная причина рассеянного распределения контейнеров в центре обработки данных, что приводит к худшему использованию физических ресурсов.

При размещении реплик необходимо выбрать подходящее место для поддержания балансировки нагрузки системы и для повышения производительности системы. Из-за огромного количества гетерогенных узлов в различных местах пограничной облачной системы неэффективный алгоритм размещения реплик влияет на балансировку нагрузки системы. Поэтому основной проблемой является размещение реплик в пограничной облачной среде [2.15].

В традиционных облачных сервисах основные проблемы подготовки контейнерного кластера заключаются в оптимальном размещении контейнеров при рассмотрении межконтейнерного трафика в контейнерном кластере. Ключевая проблема еще больше обостряется, когда контейнерный кластер подготавливается в режиме онлайн [2.16].

Миграция виртуальных машин - это процесс переноса виртуальных машин с одного физического сервера на другой. Он предоставляет различные преимущества центру обработки данных в нескольких сценариях,

включая отказоустойчивость, управляемость балансировкой нагрузки, повышение производительности и управление питанием, но миграция виртуальной машины приводит к нарушениям соглашения об уровне обслуживания (SLA) и снижению производительности, которые нельзя игнорировать, особенно если необходимо достичь критических бизнес-целей.

Вопрос контейнеризации практически не рассматривался, особенно учитывая проблему миграции виртуальных машин. Время, затраченное на миграцию виртуальной машины, зависит от ее размера. Чем меньше размер виртуальной машины, тем быстрее происходит миграция. Можно иметь несколько вариантов миграции виртуальной машины с одного физического компьютера, рассматривая только один вариант перемещения виртуальной машины на физическую машину и игнорируя другие альтернативы, которые могут быть рассмотрены для осуществления миграции виртуальной машины. Выбор фитнес-функции, которая включает в себя все вопросы производительности, также никогда не рассматривается.

Таким образом, задача состоит в том, чтобы определить наилучшую стратегию миграции, учитывающую функцию полезности, в которую включены все требования к производительности и использованию ресурсов.

2.2. Расширенная архитектурная модель облачных вычислений

Основная цель исследования - разработать подход к миграции в облаке путем добавления слоя, который обеспечивает эффективную миграцию виртуальных машин. На этом уровне реализованы алгоритмы планирования и миграции. В архитектуре облачных вычислений существует четыре уровня, такие как аппаратный уровень, инфраструктурный уровень, уровень платформы и прикладной уровень. Кроме того, OML недавно появился в облачной архитектуре, которая предназначена для выполнения миграции. В OML алгоритм размещения рассматривает различные варианты размещения и находит лучшие, а затем инициирует миграцию, которая вы-

брана как лучшая. Рассматриваются 5 различных стратегических вариантов, которые включают миграцию виртуальной машины на физическую машину, преобразование виртуальной машины в контейнер, который затем переносится на физическую машину, а затем виртуальная машина строится обратно, преобразование контейнера в виртуальную машину и затем миграцию на другую машину, перенос контейнера на другую физическую машину и последующее преобразование его обратно в физическую машину. OML проходит четыре этапа, которые включают проверку, бизнес, трансформацию и развертывание для осуществления миграции виртуальных машин в облаках.

Планировщик облачной вычислительной системы инициирует процесс осуществления миграции виртуальных машин путем взаимодействия с программой валидации, расположенной на уровне OML. Как только запрос будет признан подлинным, фактическая миграция осуществляется через другую программу, называемую бизнес-программой. Деловая программа предназначена для определения наилучшего варианта миграции. Преобразование VM в контейнер или контейнера в VM осуществляется отдельной программой, на которую воздействуют на основе стратегического решения, определенного бизнес-программой. Взаимодействие между различными программами в слое OML показано на рис. 2.1.

Миграционный подход, основанный на алгоритме оптимизации S-WOA в платформе облачных вычислений, показан на рис. 2.1, где представлена системная модель миграции в облаке.

В последние несколько десятилетий облачным средам уделяется большое внимание в области информатики. Модель облачных вычислений обеспечивает гибкий и самый простой способ управления и извлечения файлов и данных. Однако облачная модель состоит из нескольких PM для решения запросов от пользователей, и PM собирает виртуальные машины для динамической обработки задач. Виртуальная машина, доступная в об-

лаке, создается динамическим способом, чтобы уменьшить узкое место и проблему визуализации. Кроме того, контейнер - это новая облегченная технология виртуализации, которая не требует монитора виртуальных машин (VMM). Контейнер предлагает изолированную виртуальную среду на уровне операционной системы.

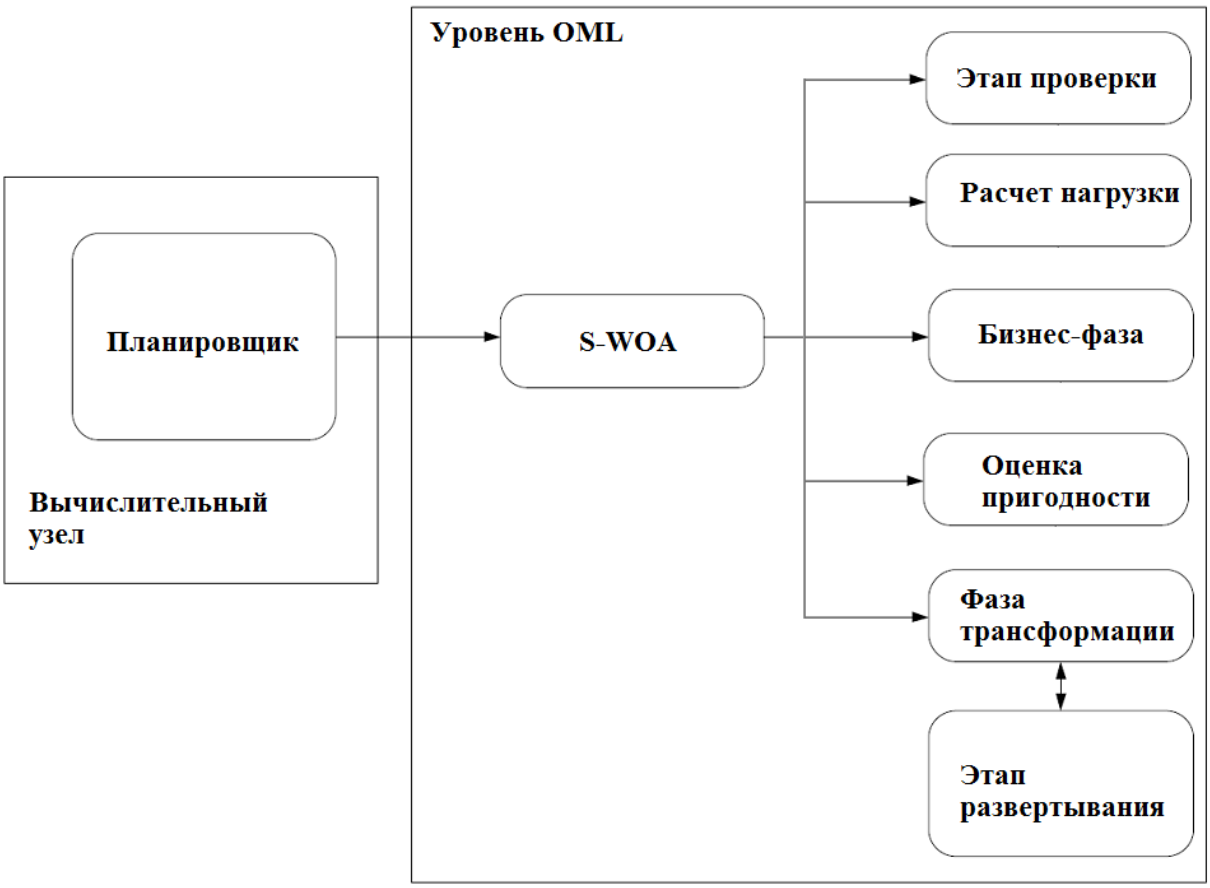


Рис. 2.1. Многоуровневая архитектура OML

Однако различные контейнеры, выполняющиеся в конкретной операционной системе, используют одно и то же ядро операционной системы, и тогда контейнер не нуждается в запуске операционной системы. Однако миграция выполняется на виртуальной машине в контейнер, или контейнер в виртуальную машину, или кластер виртуальных машин в кластер контейнеров.

2.3. Пятислойная архитектура и модифицированный алгоритм

В этом разделе представлен предложенный алгоритм S-WOA, использующий оптимизированную архитектуру Migration Layer (OML) в облачных средах для миграции. В общем случае архитектура облачных вычислений может быть разделена на несколько уровней, таких как аппаратный уровень, инфраструктурный уровень, платформенный уровень и прикладной уровень. Кроме того, OML разрабатывается недавно и размещается над прикладным уровнем с использованием определенных компонентов, таких как фаза проверки, бизнес-фаза, фаза преобразования и фаза развертывания. На этапе проверки сначала проверяется запрос на миграцию, а затем состояние виртуальной машины/контейнера. После этого метрики виртуальной машины/контейнера проверяются для миграции в облаке. На бизнес-этапе проверяется фактическая информация. Здесь алгоритм механизма размещения используется с использованием недавно разработанного алгоритма оптимизации, а именно S-WOA для выбора подходящего сервера (физической машины) для миграции. Миграция выполняется на виртуальной машине в контейнер, или контейнер в виртуальную машину, или кластер виртуальных машин в кластер контейнеров.

Предлагаемый S-WOA разработан путем интеграции SSA [2.40] и WOA [2.41]. Кроме того, была разработана фитнес-функция, основанная на пропускной способности, использовании ресурсов и нагрузке. На этапе преобразования выполняется преобразование информации виртуальной машины в информацию контейнера или наоборот в нотации объекта миграции (MON). Наконец, на этапе развертывания алгоритм размещения, адаптированный на бизнес-этапе, используется в месте назначения для миграции, а на этапе преобразования доступен преобразованный код, который развертывается в соответствии с требуемой стратегией. После завершения развертывания исходный ресурс удаляется.

2.3.1. Этап проверки

Определим облачную модель с различными виртуальными машинами, PM и контейнерами. Облачная модель, содержащая *n* PM, представлена в виде $J=\{J_1,\ldots,J_i,\ldots,J_n\}$, $1\leq i\leq n$, и несколько виртуальных машин доступны в каждой PM. Кроме того, *V* виртуальных машин определено как $V=\{V_1,\ldots,V_j,\ldots,V_m\}$, $1\leq j\leq m$, где общее количество виртуальных машин обозначается как V_m. Кроме того, количество *C* доступных контейнеров определено как $C=\{C_1,\ldots,C_k,\ldots,C_p\}$, $1\leq k\leq p$, где общий объем контейнеров обозначается как C_p. Таким образом, на этапе проверки всякий раз, когда возникает необходимость/ запрос на миграцию, сначала проверяется истинность следующего: а) VM/Container существует, активен и работает, б) метрики VM/Container корректны. Однако каждый контейнер состоит из нескольких параметров, таких как загрузка процессора, память, производительность в MIPS, количество обрабатывающих элементов и частота, и уравнение выражается следующим образом,

$$C_k=\{ D_k, H_k, B_k, A_k, I_k\} \quad (2.1)$$

где D_k обозначает загрузку процессора в k-м контейнере, а H_k относится к количеству памяти, используемой k-м контейнером. Общее количество MIPS, используемых k-м контейнером, обозначается как B_k, A_k обозначает общее количество обрабатывающих элементов, используемых k-м контейнером, а I_k есть частота k-го контейнера. Для миграции следует учитывать 5 ситуаций:

- от VM к PM;
- контейнер для VM с той же PM;
- контейнер для VM с разными PM;
- миграция задач из VM в контейнер;
- миграция задач контейнера на виртуальную машину.

Коэффициент качества задач (TQR) вычисляется с учетом количества задач с первоначальным крайним сроком выполнения задач и времени,

затраченного на их выполнение. Таким образом, TQR вычисляется с использованием уравнения

$$TQR = \frac{1}{|t|}\sum_{r=1}^{|t|}\left(\frac{P_{ori} - P_{comp}}{G}\right) \tag{2.2}$$

где $|t|$ - число задач, P_{ori} - первоначальный срок выполнения задач, а P_{comp} – полное время выполнения задач. Далее:

- если $TQR>T$, оценить загрузку облака, удовлетворяющую трем условиям;

- если $TQR \leq T$, а нагрузка VM больше порогового значения, выполнить условие-4;

- если TQR≤T , а нагрузка VM меньше или равна порогу, выполнить условие-5.

2.3.2. Расчет нагрузки

Значение нагрузки рассчитывается на основе используемых контейнером ресурсов для выполнения задачи, поставленной пользователем.

Память, процессор, количество обрабатывающих элементов, MIPS и частота - это параметры, учитываемые при вычислении нагрузки в облачной платформе. Поэтому нагрузка контейнера вычисляется с помощью выражения:

$$L_k = \frac{1}{G_1}\sum_{q=1}^{r}\frac{\left(G^D + G^H + G^B + G^A + G^I\right)y}{\left(max\left(G^D\right) + max\left(G^H\right) + max\left(G^B\right) + max\left(G^A\right) + max\left(G^I\right) + \right)e} \tag{2.3}$$

где G^D - общее количество процессоров, используемых в k-м контейнере, G^H общее количество памяти, используемой в k-м контейнере, G^B MIPS в k-м контейнере, G^A - количество обрабатывающих элементов в k-м контейнере, G^I - количество частот в k-м контейнере, r - к число задач, а G_1 - нормализующий коэффициент. Если q-я задача выполняется k-м контейнером, то параметр y=1 или иначе y становится 0.

$$y = \begin{cases} 1, \text{если } q-я \text{ задача выполняется } k-м \text{ контейнером} \\ 0, в \text{ противном случае} \end{cases} \quad (2.4)$$

Нагрузка VM вычисляется с учетом нагрузки контейнера и нормализующего коэффициента:

$$L_j = \frac{1}{G_2} \sum_{k=1}^{p} L_k \quad (2.5)$$

где L_k - нагрузка k-го контейнера, а $\frac{1}{G_2}$ - нормализующий коэффициент. Затем нагрузка PM вычисляется с учетом нагрузки VM с нормирующим коэффициентом, а выражение для нахождения нагрузки PM задается (2.6):

$$L_i = \frac{1}{G_3} \sum_{j=1}^{m} L_j \quad (2.6)$$

где L_j - нагрузка j-й VM, а $\frac{1}{G_3}$ - нормирующий коэффициент. Затем вычисление нагрузки в облаке следует трем условиям:

- Если нагрузка PM больше порогового значения, то условие-1, в противном случае миграции не будет.

- Если нагрузка VM превышает пороговое значение, то условие-2.

- Если нагрузка PM и VM превышает пороговое значение, то условие-3.

Здесь упомянутые условия-1, 2 и 3 есть кодирование решения. После этого, если TQR≤T, то:

- Если нагрузка VM превышает пороговое значение, то условие-4.

- Если нагрузка VM является меньшей, чем пороговое значение, проанализировать состояние-5.

2.3.3. Бизнес-фаза

Фактическая информация проверяется на бизнес-фазе, и для всех условий разрабатывается алгоритм размещения, а именно S-WOA для выбо-

ра подходящего сервера (PM) для миграции. Кодирование решения, вычисление функции пригодности и алгоритмическая процедура предложенного S-WOA проиллюстрированы в следующем разделе.

Кодирование решения

Кодирование решения - это представление решения, которое должно быть определено с помощью предложенного алгоритма. Здесь для кодирования решения подходят пять условий для лучшей миграции в облаке.

Условие-1

На рис. 2.2 показано решение, кодирующее разработанную модель для условия-1. Здесь миграция выполняется из VM в PM. Здесь предлагаемый S-WOA выбирает виртуальную машину, которая подходит для миграции. Будем считать, что существует *n* PM. Первая VM работает на первой PM, третья VM работает на второй PM и четвертая VM работает на *m*-th PM. Здесь PM представлена как {1,2,..., n}, где n - число PM. Индекс VM представлен как {1,3,...,4}.

Условие-2

На рис. 2.3 представлено решение, кодирующее разработанную модель для условия-2. В этом состоянии миграция выполняется из контейнера в виртуальную машину в той же PM. Здесь предлагаемый S-WOA выбирает контейнер, который подходит для миграции. Предположим, что число виртуальных машин в соответствующей PM обозначается как *l*. Первый контейнер работает на первой VM в соответствующей PM, третий контейнер работает на второй VM в соответствующей PM, а четвертый контейнер работает под *y*-й VM в соответствующей PM. Количество виртуальных машин в соответствующих PM представляется как {1,2,..., l}, где l - число виртуальных машин в соответствующей PM. Индекс контейнера представлен как {1,3,...,4} в соответствующей PM.

Состояние-3

Кодирование решения разработанной модели для условия-3 проил-

люстрировано на рис. 2.4. В этом случае миграция выполняется из контейнера в виртуальную машину в разных PM. Таким образом, разработанный S-WOA выбирает конкретный контейнер для миграции. Пусть m - число VM. Здесь первый контейнер работает на первой виртуальной машине, третий контейнер работает на второй виртуальной машине, а четвертый контейнер работает на p-й виртуальной машине на разных PM.

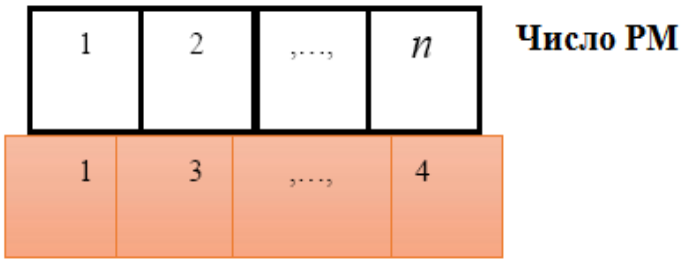


Рис. 2.2. Кодирование решения для условия-1

Рис. 2.3. Кодирование решения для условия-2

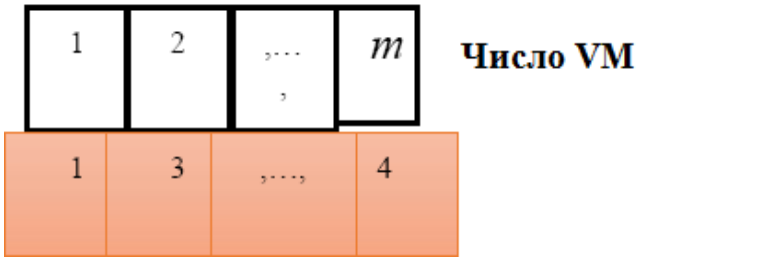


Рис. 2.4. Кодирование решения для условия-3

Вышеупомянутые три условия выполняются, когда TQR больше порогового значения.

Состояние-4

Кодирование решения разработанной модели для условия-4 иллюстрирует рис. 2.5. Здесь миграция осуществляется из задач в VM в контейнер. Здесь предлагаемый S-WOA выбирает задачи в VM, которые подходят для миграции. Пусть 1×k – число задач в VM, где k - количество контейне-

ров, первая задача в VM работает с первым контейнером, вторая задачя - со вторым контейнером, четвертая задача - с третьим контейнером, а пятая задача в VM работает с контейнером 1×k. Здесь число контейнеров 1×k представлено как {1,2,3,..., 1×k}. Индекс задач в VM обозначается как {1,2,...,5}.

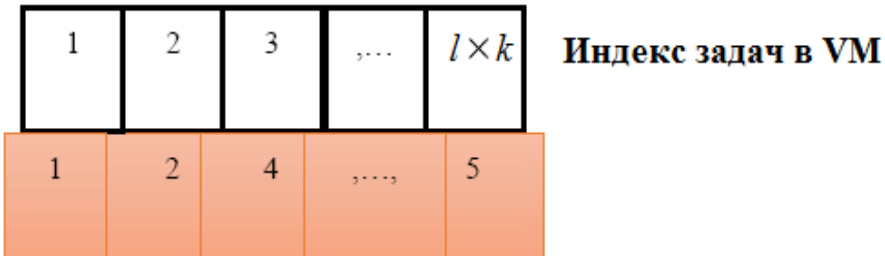


Рис. 2.5. Кодирование решения для условия-4

Состояние-5

Кодирование решения для условия-5 с использованием разработанной модели представлено на рис. 2.6. Здесь миграция выполняется из задач контейнера в виртуальную машину.

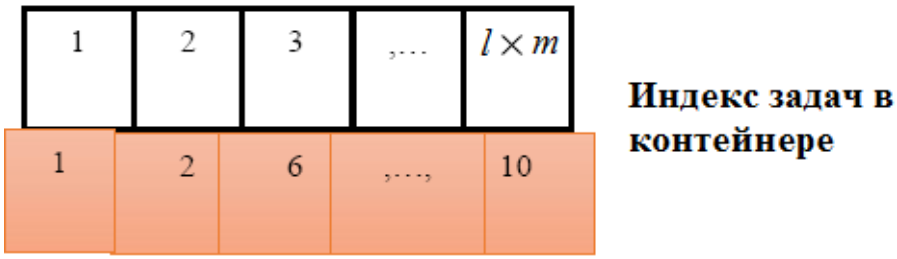


Рис. 2.6. Кодирование решения для условия-5

Кроме того, предлагаемый S-WOA выбирает контейнерные задачи, которые подходят для миграции. Пусть 1×m - число VM. Первая задача из контейнера выполняется в первой VM, вторая задачи задача из контейнера выполняется во второй VM, шестая задача из контейнера выполняется в третьей VM и десятая задача из контейнера выполняется в 1×m VM. Здесь число виртуальных машин 1×m представлено как {1,2,3,..., 1×m }. Индекс задач в контейнере обозначается {1,2,6,...,10}.

Условие-4 и 5 следует соблюдать, когда TQR меньше или равен по-

рогу, а нагрузка VM больше, или меньше, или равна порогу.

2.3.4. Оценка пригодности

Фитнес-функция оценивается для определения оптимального решения из множества решений. Пригодность S-WOA оценивается с использованием пропускной способности, использования ресурсов и нагрузки. Фитнес с минимальным значением - это оптимальное решение для миграции в облако. Здесь функция пригодности вычисляется для трех условий и приводится ниже. Фитнес-функция для первого условия, выражается следующим образом:

$$F_1 = \frac{1}{n}\sum_{i=1}^{n} S_i \tag{2.7}$$

где F_1 – фитнес-функция для условия-1, n – число PM.

$$S_i = \begin{cases} \frac{1}{2}\sum\limits_{\substack{j=1 \\ j\in i}}^{m}\left(L_j + M_j\right), \textit{если i не загружено} \\ 1, \textit{если перегружено} \end{cases} \tag{2.8}$$

где L_j - нагрузка j-й VM, а M_j - стоимость миграции:

$$M_j = \frac{1}{n}\sum_{i=1}^{n}\frac{u_i}{cm} \tag{2.9}$$

где u_i - число миграций, а c - постоянная. Тогда для условий-2 и 3 выражение фитнес-функции представляется в виде

$$F_2 = \frac{1}{n}\sum_{i=1}^{n} S_i \tag{2.10}$$

где $S_i = \begin{cases} \frac{1}{2}\sum\limits_{\substack{j=1 \\ j\in i}}^{m}\left(L_{ij} + M_{ij}\right), \textit{если i не загружено} \\ 1, \textit{если перегружено} \end{cases}$

и стоимость миграции выражается как

$$M_{ij} = \frac{1}{m}\sum_{j=1}^{m}\frac{u_j^*}{cp} \quad (2.11)$$

где u_j^* - количество миграций из контейнера в VM. Для условий-4 и 5 следует следовать тому же уравнению фитнес-функции условия-1.

2.3.5. Алгоритмические процедуры в методе S-WOA

В этом разделе представлен подход к миграции с использованием предложенного S-WOA для выбора подходящего сервера PM для миграции в облачную вычислительную платформу. S-WOA разработан путем объединения SSA с WOA [3]. Интегрируя WOA с SSA, параметрические характеристики обеих оптимизаций наследуются, что повышает производительность точности классификации. Алгоритмическая процедура миграции в облаке:

Шаг 1. Инициализация

Инициация популяции, которая дается как

$$X=\{X_1, X_2, \ldots, X_u, \ldots, X_v\};\ 1\leq u\leq v \quad (2.12)$$

где v - общая численность популяции, а X_u – u-е решение.

Шаг 2. Оценка фитнес-функции

Пригодность для каждого решения вычисляется на основе фитнес-функции, представленной в (2.5) и (2.8) для условий 1, 2 и 3. Фитнес-функция принимается как функция максимизации, а решение, обеспечивающее максимальную пригодность, рассматривается как лучшее решение.

Шаг 3. Обновление позиции с использованием алгоритма мутации

Алгоритм WOA обеспечивает лучшую точность и эффективность выбора наилучших последовательностей. Согласно алгоритму WOA [3], уравнение мутации задается формулой

$$X(h+1)=X^*(h)-RS \quad (2.13)$$

где h – номер текущей итерации, R – вектор коэффициентов, X^* - позиция

вектора с лучшим решением, S – окрестность лучшего поиска.

$$X(h+1)=X^{*}(h)-R|KX^{*}(h)-X(h)| \quad (2.14)$$

где K – вектор коэффициентов. Считая, что $KX^{*}(h)$ больше, получаем

$$X(h+1)=X^{*}(h)(1-RK)+RX(h) \quad (2.15)$$

Для получения глобального оптимума в задаче миграции, применен метод SSA [2.4]. Отсюда

$$X(h+1)=X(h)+f_d T_y(X^{*}(h)-X(h))Q_1 \quad (2.16)$$

$$X(h+1)=X(h)+f_d T_y X^{*}(h)Q_1-f_d T_y X(h)Q_1 \quad (2.17)$$

$$X^{*}(h)=\frac{X(h+1)-X(h)+f_d T_y X(h)Q_1}{f_d T_y Q_1} \quad (2.18)$$

Подставляя (2.18) в (2.15), получаем:

$$X(h+1)=\frac{X(h+1)-X(h)+f_d T_y X(h)Q_1}{f_d T_y Q_1}(1-RK)+RX(h) \quad (2.19)$$

$$X(h+1)=\frac{X(h+1)(1-RK)}{f_d T_y Q_1}+\frac{X(h+1)(f_d T_y Q_1-1)}{f_d T_y Q_1}(1-RK)+RX(h) \quad (2.20)$$

$$X(h+1)\frac{X(h+1)(1-RK)}{f_d T_y Q_1}=\frac{X(h)(f_d T_y Q_1-1)}{f_d T_y Q_1}(1-RK)+RX(h) \quad (2.21)$$

$$X(h+1)\left[1-\frac{(1-RK)}{f_d T_y Q_1}\right]=\frac{X(h)(f_d T_y Q_1-1)}{f_d T_y Q_1}(1-RK)+RX(h) \quad (2.22)$$

$$X(h+1)\left[\frac{f_d T_y Q_1-1+RK}{f_d T_y Q_1}\right]=\frac{X(h)(f_d T_y Q_1-1)}{f_d T_y Q_1}(1-RK)+RX(h) \quad (2.23)$$

Таким образом, получаем финальное выражение для предложенного метода S-WOA миграции на платформе облачных вычислений:

$$X(h+1)=\frac{f_d T_y Q_1}{f_d T_y Q_1-1+RK}\left[\frac{X(h)(f_d T_y Q_1-1)}{f_d T_y Q_1}(1-RK)+RX(h)\right] \quad (2.24)$$

где случайное расстояние сглаживания обозначается как f_d, Q_1 - случайное число представляется в диапазоне от 0 до 1, h - текущая итерация. Значение T_y=1.9 и получено в результате экспертного анализа. R и K представляют

собой векторы коэффициентов, которые основаны на константах s и t:

$$R=2ts-t \quad (2.25)$$

$$K=2s \quad (2.26)$$

Константа t линейно уменьшается от 2 до 0, указывая на переключение между фазами исследования и эксплуатации, а константа s представляет собой случайный вектор, изменяющийся в диапазоне [0,1].

Шаг 4. Проверка осуществимости решения

Осуществимость решения вычисляется на основе фитнес-функции. Если новое сгенерированное решение лучше предыдущего, то оно заменяется новым решением.

Шаг 5. Остановка

Повторяем шаги до максимальных итераций до тех пор, пока не будут определены глобальные оптимальные наилучшие решения.

Таким образом, алгоритм оптимизации, рассмотренный в этом разделе, направлен на определение оптимальных весов для миграции в облаке. Псевдокод разработанного алгоритма S-WOA изображен в алгоритме 2.1, который демонстрирует пошаговое описание алгоритма.

Алгоритм 2.1: Псевдокод предложенного алгоритма S-WOA

Input: Популяция $X=\{X_1, X_2, \ldots, X_u, \ldots, X_v\}$; $1\leq u\leq v$

Output: X^* - лучшее решение

Procedure:

Begin

Инициализация популяции $X=\{X_1, X_2, \ldots, X_u, \ldots, X_v\}$; $1\leq u\leq v$

Вычисление фитнес-функции с использованием (2.5) и (2.8)

while ($h<h_{max}$)

for каждого возможного решения

If $l>0.5$

Обновить позицию S-WOA используя (2.19)

End if

End for

Убедиться в наличии возможного решения вне области поиска

Обновить X^* при наличии лучшего решения

h=h+1

End while

Return X^*

2.3.6. Фаза трансформации

На основе этапа валидации определяется условие и алгоритм размещения с использованием предложенного алгоритма S-WOA. Кроме того, оценивается пригодность для соответствующего состояния. Таким образом, выполняется преобразование информации виртуальной машины в информацию контейнера или наоборот в MON (обозначение объекта миграции).

2.3.7. Этап развертывания

С помощью алгоритма размещения в бизнес-фазе становится доступным место назначения для миграции, а с фазы преобразования - преобразованный код, который развертывается в соответствии с требуемой стратегией. В зависимости от условий миграция развертывается.

2.4. Эксперименты и сравнительный анализ

2.4.1. Экспериментальная установка

Исполнение разработанного алгоритма осуществлялось в OpenStack в PYTHON, используя ПК с ОС Windows 10, 2 ГБ оперативной памяти и процессором Intel i3 core.

2.4.2. Показатели оценки

Производительность предлагаемого S-WOA используется для анали-

за методов, использующих количество установленных виртуальных машин, загрузку процессора, использование памяти, количество активированных PM и время.

2.4.3. Расчеты и анализ данных

Эффективность разработанного метода анализируется путем сравнения разработанной модели с существующими методами, такими как ACO-BF [2.1], Container VM-PM [2.2] и многокритериальная оптимизация размещения реплик [2.3].

Анализ с количеством задач=100

Проведем сравнительный анализ метрик миграции в облаке с использованием 100 задач.

Анализ с использованием метрики количества экземпляров виртуальной машины с различным количеством контейнеров, заключается в следующем. Для 50 контейнеров число экземпляров VM, измеренное с помощью ACO-BF, Container VM-PM, многокритериальной оптимизации размещения реплик и предложенного S-WOA, равно 24, 24, 24 и 23. При container=80 число экземпляров VM, вычисленных с помощью ACO-BF, Container VM-PM и многокритериальной оптимизации размещения реплик и предложенного S-WOA, равно 24, 24, 24 и 23.

Анализ методов, использующих загрузку ЦП, заключается в следующем. Для 50 контейнеров, загрузка процессора вычисляется ACO-BF, Container VM-PM и многокритериальной оптимизации размещения реплик и предлагаемого S-WOA составляет 0.259, 0.273, 0.280, и 0.283. При container=80 загрузка ЦП, вычисленная с помощью ACO-BF, Container VM-PM, многокритериальной оптимизации размещения реплик и предложенного S-WOA, составляет 0,407, 0,408, 0,420 и 0,460.

Анализ методов, использующих параметр использования памяти, заключается в следующем. Для 50 контейнеров использование памяти, вы-

численное с помощью ACO-BF, Container VM-PM, многокритериальной оптимизации размещения реплик и предложенного S-WOA, составляет 0,256, 0,274, 0,300 и 0,305. При container=80 использование памяти, вычисленное с помощью ACO-BF, Container VM-PM и многокритериальной оптимизации размещения реплик и предложенного S-WOA, составляет 0,309, 0,323, 0,372 и 0,377.

Анализ методов, использующих метрику числа активированных PM, заключается в следующем. Для 50 контейнеров количество активированных PM, вычисленное с помощью ACO-BF, Container VM-PM, многокритериальной оптимизации размещения реплик и предложенного S-WOA, составляет 3, 3, 3 и 4. При container=80 число активированных PM, вычисленное с помощью ACO-BF, Container VM-PM, многокритериальной оптимизации размещения реплик и предложенного S-WOA, равно 3, 3, 4 и 4.

Анализ методов, использующих метрику времени, заключается в следующем. Для 50 контейнеров время, вычисленное ACO-BF, Container VM-PM, многокритериальной оптимизацией размещения реплик и предложенным S-WOA, составляет 19,44 с, 18,99 с, 9,96 с и 4,99 с. Когда контейнер=80, время, вычисленное с помощью ACO-BF, Container VM-PM, многокритериальной оптимизации размещения реплик и предложенного S-WOA, составляет 47,88 с, 35,19 с, 22,73 с и 12,56 с.

Проведем сравнительный анализ метрик миграции в облаке с использованием 100 задач.

Анализ с использованием метрики количества экземпляров виртуальной машины с различным количеством контейнеров, заключается в следующем. Для 50 контейнеров количество экземпляров VM, измеренное с помощью ACO-BF, Container VM-PM, многокритериальной оптимизации размещения реплик и предложенного S-WOA, составляет 25, 24, 24 и 22. Когда контейнер=80, количество создаваемых виртуальных машин вычисляется ACO-BF, Container VM-PM, многокритериальная оптимизация раз-

мещения реплик и предлагаемого S-WOA составляет 24, 23, 23, 23.

Анализ методов, использующих загрузку ЦП, заключается в следующем. Для 50 контейнеров, загрузка процессора, измеренная с помощью ACO-BF, Container VM-PM, многокритериальной оптимизации размещения реплик и предложенного S-WOA, составляет 0.298, 0.335, 0.341, 0.364. При container=80 загрузка ЦП, вычисленная с помощью ACO-BF, Container VM-PM, многокритериальной оптимизации размещения реплик и предложенного S-WOA, составляет 0,350, 0,455, 0,456 и 0,483.

Анализ методов, использующих параметр использования памяти, заключается в следующем. Для 50 контейнеров использование памяти, вычисленное с помощью ACO-BF, Container VM-PM, многокритериальной оптимизации размещения реплик и предложенного S-WOA, составляет 0,265, 0,285, 0,288 и 0,294. При container=80 использование памяти, вычисленное с помощью ACO-BF, Container VM-PM, многокритериальной оптимизации размещения реплик и предложенного S-WOA, составляет 0,410, 0,448, 0,505 и 0,523.

Анализ методов, использующих метрику числа активированных PM, заключается в следующем. Для 50 контейнеров количество активированных PM, вычисленное с помощью ACO-BF, Container VM-PM, многокритериальной оптимизации размещения реплик и предложенного S-WOA, равно 3, 3, 4 и 4. При container=80 число активированных PM, вычисленное с помощью ACO-BF, Container VM-PM, многокритериальной оптимизации размещения реплик и предложенного S-WOA, равно 3, 3, 3 и 4.

Анализ методов, использующих метрику времени, заключается в следующем. Для 50 контейнеров время, вычисленное ACOBF, Container VM-PM и многокритериальной оптимизацией размещения реплик и предложенным S-WOA, составляет 24,35 с, 13,45 с, 12,76 с и 6,53 с. Когда container=80, время, вычисленное ACO-BF, Container VM-PM, многокритериальной оптимизацией размещения реплик и предложенным S-WOA, со-

ставляет 63,67 с, 58,87 с, 34,49 с и 17,63 с.

2.4.4. Сравнительное обсуждение

Минимальное количество экземпляров виртуальных машин - 23, полученное с помощью предложенного S-WOA, в то время как существующие ACO-BF, Container VM-PM и многокритериальная оптимизация размещения реплик составляют 25, 24 и 24. Максимальная загрузка ЦП, вычисленная S-WOA, составляет 0,483, тогда как существующие ACO-BF, Container VM-PM и Многокритериальная оптимизация размещения реплик дают 0,350, 0,455 и 0,458 при числе входящих задач 200. Кроме того, максимальное значение использования памяти, измеренное S-WOA, составляет 0,523, тогда как существующие ACO-BF, Container VM-PM и многокритериальная оптимизация размещения реплик составляют 0,410, 0,448 и 0,505 при числе входящих задач 100. Минимальное значение времени, вычисленное S-WOA, составляет 4,99 с, тогда как существующие ACO-BF, Container VM-PM и многокритериальная оптимизация размещения реплик составляют 19,44 с, 18,99 с и 9,96 с.

2.5. Выводы

Представлен алгоритм размещения S-WOA для миграции в облаке. Как правило, архитектура облачных вычислений состоит из прикладного уровня, уровня платформы, уровня инфраструктуры и аппаратного уровня.

Аппаратный уровень используется для обработки физических ресурсов облака, включая физическое оборудование сетевых устройств и энергосистемы.

Инфраструктурный уровень также называется уровнем виртуализации, который разделяет аппаратное обеспечение и обеспечивает пул дисковых хранилищ и вычислительных ресурсов.

Следующий уровень - это уровень платформы, который широко охватывает операционные системы и фреймворки приложений в зависимости от каждой конкретной платформы. Таким образом, этот уровень используется для сокращения усилий по разработке, предоставляя разработчикам платформу разработки в качестве сервиса без установки каких-либо фреймворков или программного обеспечения на их компьютеры.

Уровень приложений предлагает конечным пользователям облачные приложения в качестве сервиса. OML вводится недавно и размещается над прикладным уровнем с использованием определенных компонентов, таких как фаза проверки, бизнес-фаза, фаза преобразования и фаза развертывания.

На бизнес-этапе проверяется фактическая информация. Здесь алгоритм размещения выполняется с использованием алгоритма оптимизации, а именно S-WOA для выбора подходящего сервера (Физической машины) для миграции, удовлетворяющего пяти условиям.

Предлагаемый S-WOA разработан путем интеграции SSA и WOA. Наряду с предложенным методом S-WOA рассматривается фитнес-функция с использованием полосы пропускания, использования ресурсов и нагрузки.

Предложенная S-WOA и фитнес-функция улучшили общую производительность сети для лучшей миграции, а затем фаза преобразования и развертывания выполняется в зависимости от состояния.

Эффективность предложенного S-WOA рассчитана по отношению к другим методам и показала эффективные результаты при максимальной загрузке процессора 0,483, максимальной памяти 0,523 и минимальном времени 4,99 с соответственно.

Работа может быть расширена путем определения решений для ограничений при обследовании миграции в облачных технологиях.

Список источников к главе 2

2.1. Mateusz G., Alicja G., Pascal B. Cloud brokering: Current practices and upcoming challenges// IEEE Cloud Comput., 2015, Vol. 2, no.2, p. 40–47.

2.2. Milani A.S., Navimipour N.J. Load balancing mechanisms and techniques in the cloud environments: Systematic literature review and future trends// J. Netw. Comput. Appl., 2016. Vol. 71, no. 1, p. 86–98.

2.3. Jiang Y.C. A survey of task allocation and load balancing in distributed systems// IEEE Trans. Parallel Distrib. Syst., 2016, Vol. 27, no. 2, p. 585–599.

2.4. Ning J., Cao Z., Dong X., Liang K., Ma H., Wei L. Auditable-Time Outsourced Attribute-Based Encryption for Access Control in Cloud Computing// IEEE Transactions on Information Forensics And Security, 2017.

2.5. Jansen W., Grance T. Guidelines on security and privacy in public cloud computing// NIST, SP.800-144, 2011, p. 800-144.

2.6. Barcelo M., Correa A., Llorca J., Tulino A.M., Vicario J.L., Morell A. IoT-cloud service optimization in next generation smart environments// IEEE J. on Selected Areas in Communications, 2016, vol. 34, no. 12, p. 4077-4090.

2.7. Barcelo M., Llorca J., Tulino A., Raman N. The cloud service distribution problem in distributed cloud networks// IEEE ICC 2015 SAC - Data Storage and Cloud Computing, London, United Kingdom, Jun. 2015.

2.8. Stojmenovic I., Wen S. The fog computing paradigm: Scenarios and security// Federated Conference Computer Science and Information Systems (FedCSIS), on, pp. 1–8, September, 2014. https://doi.org/10.15439/2014F503.

2.9. Jamshidi P., Ahmad A., Pahl C., Cloud migration research: a systematic review// IEEE Transactions on Cloud Computing, 2013, vol.1, no.2, p.142-157.

2.10. Liu X., Wang C., Zhou B.B., Chen J., Yang T., Zomaya A.Y. Priority-based consolidation of parallel workloads in the cloud// IEEE Trans Parallel DistribSyst, 2012, vol.24, no.9.

2.11. Yao F., Yao Y., Chen H., Li T., Lin M., Zhang X. An intelligent scheduling algorithm for complex manufacturing system simulation with frequent synchronizations in a cloud environment// Memetic Computing, 2019, vol. 11, p. 357–370.

2.12. Wiseman Y., Feitelson D.G. Paired gang scheduling// IEEE Trans Parallel DistribSyst, 2003, vol. 14, no .6, p.581–592.

2.13. Hussein M.K., Mousa M.H., Alqarni M.A. A placement architecture for a container as a service (CaaS) in a cloud environment// Journal of Cloud Computing, 2019, vol.8, no.1, p.7.

2.14. Zhang R., Zhong A.M., Dong B., Tian F., Li R. Container-VM-PM architecture: A novel architecture for docker container placement// International Conference on Cloud Computing, Springer, 2018, p. 128-140.

2.15. Li C., Wang Y., Tang H., Luo Y. Dynamic multiobjective optimized

replica placement and migration strategies for SaaS applications in edge cloud// Future Generation Computer Systems, 2019, vol.100, p.921-937.

2.16. Zaheer S., Malik A.W., Rahman A.U., Khan S.A. Locality-aware process placement for parallel and distributed simulation in cloud data centers// The Journal of Supercomputing, 2019, vol.1, p.23.

2.17. Zhou, R., Li, Z. and Wu, C., "An Efficient Online Placement Scheme for Cloud Container Clusters," IEEE Journal on Selected Areas in Communications, vol.37, no.5, pp.1046-1058, 2019.

2.18. Liu, B., Li, P., Lin, W., Shu, N., Li, Y. and Chang, V., "A new container scheduling algorithm based on multiobjective optimization," Soft Computing, vol.22, no.23, pp.7741-7752, 2018.

2.19. Nazir, B., "QoS-aware VM placement and migration for hybrid cloud infrastructure," The Journal of Supercomputing, vol.74, no.9, pp.4623-4646, 2018. https://doi.org/10.1007/s11227-017-2071-1

2.20. Satpathy, A., Addya, S.K., Turuk, A.K., Majhi, B. and Sahoo, G., "Crow search based virtual machine placement strategy in cloud data centers with live migration," Computers & Electrical Engineering, vol.69, pp.334-350, 2018.

2.25. JKRSastry, M TrinathBasu, Securing Multi-tenancy systems through user spaces defined within the database level, Jour of Adv Research in Dynamical & Control Systems, Volume 10, issue 7, Page 405-412, 2018

2.26. J. K. R. Sastry, K. Sai Abhigna, R. Samuel and D. B. K. Kamesh, Architectural models for fault tolerance within clouds at the infrastructure level, ARPN Journal of Engineering and Applied Sciences, VOL. 12, NO. 11, 2017, Pages 3463-3469

2.40. Jain, M., Singh, V. and Rani, A., "A novel natureinspired algorithm for optimization: Squirrel search algorithm," Swarm and evolutionary computation, vol.44, pp.148-175, 2019.

2.41. Mirjalili, S. and Lewis, A., "The whale optimization algorithm," Advances in engineering software, vol.95, pp.51-67, 2016.

3. Алгоритмизация распределения ресурсов и планирования заданий в облачных средах

3.1. Распределение ресурсов и планирование заданий в облачных средах на основе алгоритма оптимизации роя частиц и R-коэффициента

Разработан алгоритм PSO, а R-коэффициент и PSO объединены для эффективного распределения ресурсов и планирования в облачной среде.

3.1.1. Оптимизация роя частиц

Оптимизация роя частиц состоит из частиц Роя, каждая из которых указывает на решение проблемы. В этом исследовании частицы указывают на группу виртуальных машин, которые выделяются для выполнения задач. Каждая частица в поведении Роя имеет 2 основных символа, а именно позицию x, которая обозначает рекомендуемое место, и скорость v, которая указывает скорость движения. Оптимизация роя частиц использует адаптивное движение, которое преобразует положение частицы на каждой итерации. Шаги, используемые в алгоритме PSO, следующие:

(1) На первом шаге в алгоритме PSO производится инициализация параметров.

MAX_ST = 1000;

PF = 500;

NoT = 10;

PS = 100;

NoI = 40.

ST обозначает время моделирования, а PF - частоту паузы, когда на каждой паузе новый исполнитель должен отправлять облачные пакеты и виртуальные машины в возможный центр обработки данных. Неизвестно

количество задач, PS обозначает размер популяции.

(2) Следующим шагом в алгоритме PSO является передача начальных параметров функции полезности.

(3) после этого частицы инициализируются.

(4) Затем Рой инициализируется.

MinP (0);

MaxP (this.NoDC – 1);

MaxMinV (0.5);

setPt (this.Pt);

for (integer a = 0; I < NoI; a++)

this.swarm.evolve ();

if (a % 10 == 0) {

GB (Min Cost)

Makespan = this.ff.evaluate (bestPt.getGB)

P обозначает позицию, а NoDC – количество центров обработки данных. Pt обозначает частицы, а V - скорость. ff указывает на фитнес-функцию фитнеса, а GB - на глобальный оптимум.

Фитнес-функция:

(1) первым шагом в функции полезности является получение матрицы времени выполнения. Запись [a, b] содержит время выполнения задачи x в центре обработки данных y.

Возвращает матрицу времени выполнения.

(2) второй шаг состоит в том, чтобы получить матрицу времени взаимодействия. Запись [a, b] содержит время передачи задачи x в центр обработки данных y.

Возвращает матрицу времени взаимодействия.

(3) Третий шаг предназначен для текущего количества задач.

Makespan = Math.maximum (makespan, DCWT[DCId]);

(4) Четвертый шаг – инициализация матриц времени связи и выпол-

нения.

For (int a = 0; a < NoT; a++) {

For (int b = 0 ; b < NoDC; b++) {

ExecutionT [a] [b] = Math.random () _Max_ET;

CommunicationT [a] [b] = Math.random() _

Max_CT+20;

T - время, a и b- целые числа, ET - время выхода, а CT - время связи.

3.1.2. R-коэффициент

Основное назначение класса «R factor» заключается в выделении виртуальной машины для вычисления «R factor». Этот класс будет работать для распределения между двумя центрами данных.

Первый центр обработки данных будет рассматриваться как локальный центр обработки данных, а второй - как облачный / удаленный центр обработки данных. Облачный дата-центр будет использоваться только в том случае, если R-коэффициент будет выше некоторого порога в соответствии со следующим алгоритмом:

(1) Инициализация параметров

R = 2; инициирующее значение по умолчанию

L_Threshold= 1;

U_Threshold= 10;

NoH (используется для вычисления R-коэффициента) = 1;

NoM (используется для вычисления R-коэффициента) = 1;

L - нижний порог, U – верхний порог, NoH - количество просмотров, NoM – число потерь.

(2) Создание нового объекта APS виртуальной машины, где APS обозначает простую политику приложения.

(3) Выделение хоста для данной виртуальной машины.

Возвращает $true, если хост выделен; $false в противном случае

If виртуальная машина не была создана

Do (до тех пор, пока хост не будет найден или пока все они не будут перебраны)

Int morefree= Int.Min_

(4) Вычисление R-коэффициента.

R = (double) H/M;

Применить ограничения для выделения **/

If (R > L_Threshold && R < U_Threshold)

// разрешить свободное распределение

} иначе, если (P < L_Threshold) {

// принудительное выделение облака

Ida = 1;

Попытка создания виртуальной машины на определенном хосте

If (result) { //если виртуальная машина была успешно создана

на хосте

Фиксируем успешное значение R-коэффициента для локальных ресурсов

Результат = True;

Прерывание;

(5) Регистрация значения отказа R-коэффициента для локальных ресурсов и возврат результатов.

3.2. Анализ оптимизированного алгоритма эффективного распределения ресурсов и планирования в облачной среде

На рис. 3.1 показана блок-схема оптимизированного алгоритма эффективного распределения ресурсов и планирования в облачной среде.

Сначала инициализируем параметры виртуальной машины:

long size = 10000; // размер изображения (MB)

int ram = 4096; // память ВМ (MB)

int mips = 1000;

long bw = 4000;

int pesNumber = 4; // количество процессоров

String vmm = “Xen”; // имяVMM

Затем создаются виртуальные машины, в которых также создается контейнер для хранения:

long length = 40000;

long fileSize = 300;

long outputSize = 300;

int pesNumber = 1.

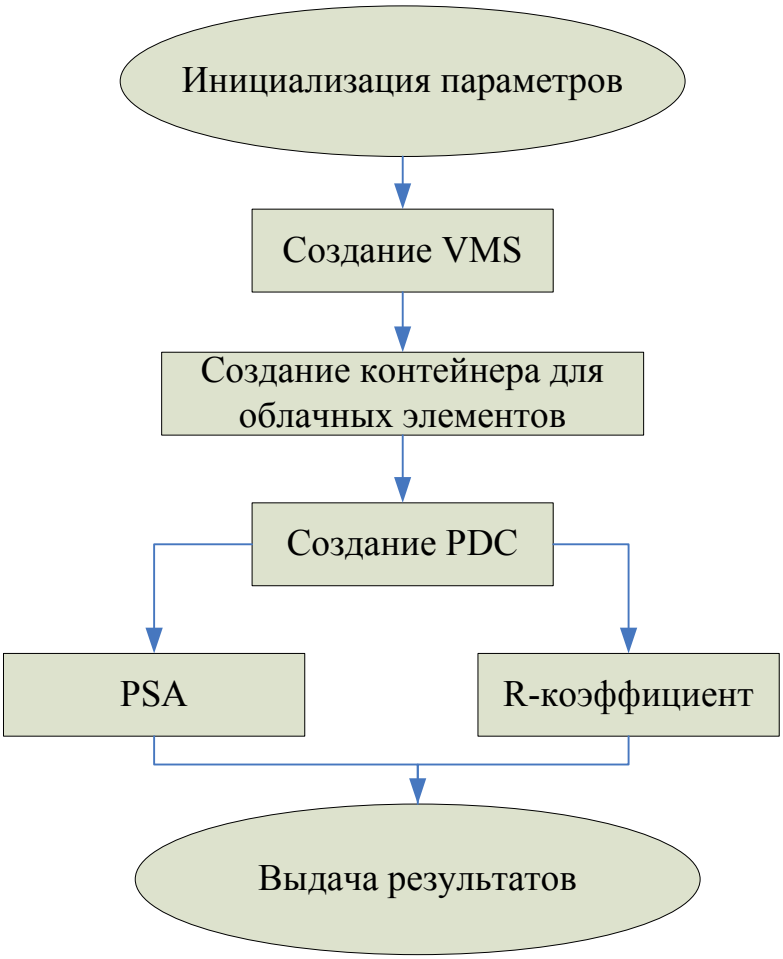


Рис. 3.1. Структурная схема алгоритма

Вслед за облаками создается центр обработки данных, который включает в себя более одного ядра. Центр обработки данных настроен для

использования с распределением виртуальных машин на основе R-коэффициента и этот алгоритм присутствует внутри имени файла VmAllocationPolicyRFactor.java. Этот класс будет функционировать для распределения между двумя центрами обработки данных. Первый центр обработки данных будет рассматриваться как локальный центр обработки данных, а второй - как удаленный/облачный центр обработки данных.

Облачный дата-центр будет использоваться только в случае, когда R-коэффициент превышает определенный порог. Кластеры в системе можно разделить на два типа, а именно облачный узел и локальный узел. Облачный узел строится с использованием ресурса облака, и ресурсы имеют отношение к функциональности локального узла. Локальные узлы строятся с использованием локальных ресурсов и являются одной из важных частей системы. Разница между локальным узлом и облачным узлом заключается в количестве узлов в облаке, которое меняется в зависимости от состояния загрузки системы и допустимого количества ресурсов. С точки зрения ресурсного обеспечения система в данном исследовании использует облачные ресурсе и локальные ресурсы. Есть и гибридный режим ресурса.

Часто существуют определенные требования, которых нет в кэше, но они должны быть отправлены на исходный сервер из-за особенностей кэша. Эти запросы называются miss request, и на время их ответа влияет состояние сети и пропускная способность исходного сервера.

Когда есть перегрузка сети или физический сервер перегружен, тогда miss request будут такими же, как если нагрузка в системе уменьшится.

Относительный индекс производительности R представляет собой отношение среднего времени обслуживания запросов к среднему времени отклика исходного запроса на пропуск сервера за данный период времени.

Алгоритм оптимизации роя частиц используется для определения того, какие облака будут распределены на какую виртуальную машину, так что алгоритм пытается снизить стоимость распределения, которая состав-

ляет время распределения. На рис. 3.2 показана блок-схема алгоритма PSO.

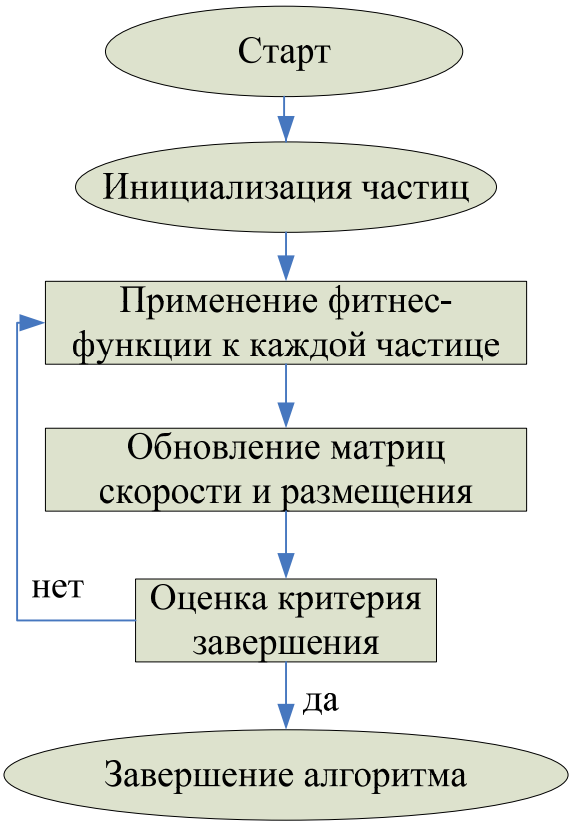


Рис. 3.2. Блок-схема алгоритма PSO

В PSO итерации используются для нахождения t позиции каждой частицы, которая называется личным лучшим (Pb), достигнутым частицей I, и глобальным лучшим (Pg). Это помогает в прогнозировании наилучшего решения за короткое время вычислений [3.13]. После того, как все задачи расположены в облачном окружении, алгоритм оптимизации используется для оценки минимальных значений времени ожидания заданий [3.14]. Эти минимальные значения используются для обеспечения правильного порядка выполнения задач, что, в свою очередь, сокращает общее время ожидания. В нашем исследовании после получения правильного оптимального порядка задач алгоритм, генерирующий очереди, используется для прогнозирования порога, а затем отправляет задачу в эту очередь. Затем расписание планирует задачи на соответствующий ресурс. Основная цель оптимизация роя частиц заключается [3.15] в выделении запроса

пользователя на соответствующий ресурс. В нашем исследовании для эффективного планирования задачи в облачном окружении процесс планирования задач нуждается в оптимальном алгоритме, который учитывает ресурсы и задачу. Алгоритм оптимизации роя частиц учитывает как задачу, так и ресурс и помогает сохранить ресурс занятым и сократить время обработки задач.

Для анализа было отобрано 500 задач с 20 виртуальными машинами для объединения R-коэффициента и частиц Роя. Первая задача находится между задачами и виртуальными машинами. При выполнении задач с использованием десяти виртуальных машин время отклика виртуальных машин становится стабильным через 40 с. При использовании 20 виртуальных машин среднее время отклика уменьшается до 20 с.

3.3. Оптимизированный по стоимости эвристический алгоритм для планирования рабочих процессов в облачной среде сервисов IaaS

3.3.1. Модель приложения рабочего процесса

Планирование рабочих процессов - это метод, используемый для сопоставления и управления выполнением нескольких связанных рабочих процессов в общий пул распределенных систем. Рабочий процесс - это представление набора задач рабочего процесса в виде направленного ациклического графа DAG:

$G=(W,E)$,

где $W = (Wt_1, Wt_2, Wt_3, \ldots, Wt_n)$;

Wt - набор из n задач рабочего процесса приложении,

E - множество ребер, представляющих поток зависимостей данных между задачей рабочего процесса Wt_i и задачей рабочего процесса Wt_j, которая обозначается через $E_{i,j}=(Wt_i, Wt_j)$.

Каждое ребро $E_{i,j}$ является представлением рабочего процесса с ограничением приоритета, которое указывает, что задача рабочего процесса Wt_j не может начаться до завершения Wt_i. В этом сценарии задача рабочего процесса Wt_i является предшественницей задачи рабочего процесса Wt_j, в то время как задача рабочего процесса Wt_j называется непосредственным преемником задачи рабочего процесса Wt_i. Все предшественнику задачи рабочего процесса на Wt_i представляется как $\mathrm{Pre}(Wt_i)$, а преемник задач рабочего процесса представляется как $\mathrm{Succ}(Wt_i)$. Поэтому задача рабочего процесса предшественника и задачи рабочего процесса преемников представлены следующим образом:

$$\mathrm{Pre}(Wt_i) = \{Wt_j \mid (Wt_j, Wt_i) \in D\} \quad (3.1)$$

$$\mathrm{Succ}(Wt_i) = \{Wt_i \mid (Wt_j, Wt_i) \in D\} \quad (3.2)$$

В каждом DAG есть задачи входа в рабочий процесс и задачи выхода из рабочего процесса. Пример DAG рабочего процесса из 12 задач с задачами рабочего процесса входа и выхода приведен на рис. 3.3.

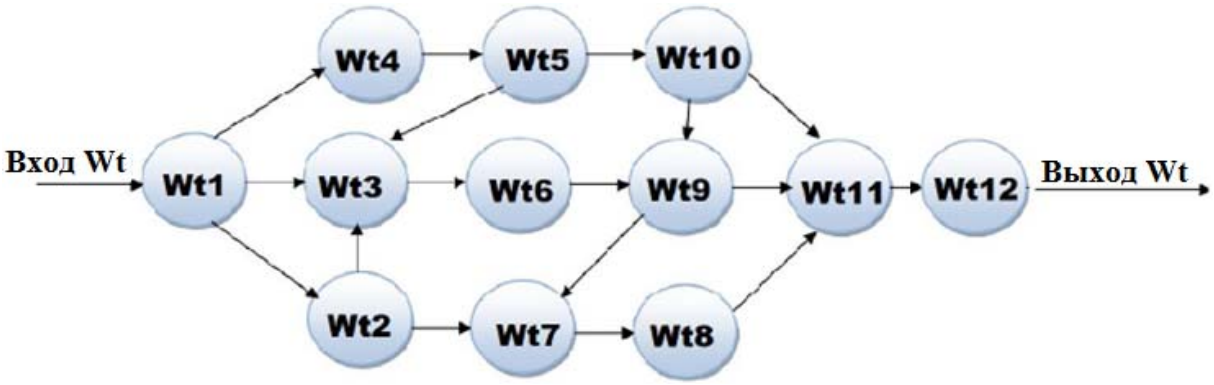


Рис. 3.3. Пример DAG рабочего процесса с задачами входа и выхода из рабочего процесса

Задача входа в рабочий процесс - это задача рабочего процесса без предшествующей задачи рабочего процесса, которая передается Wt_{entry}, как в (3.3):

$$\mathrm{Pre}(Wt_{entry}) = \varnothing \quad (3.3)$$

Задача выхода из рабочего процесса - это задача рабочего процесса без преемника, которая также передается Wt_{exit}, как в (3.4):

$$Succ(Wt_{exit})=Ø \quad (3.4)$$

3.3.2. Модель выполнения рабочего процесса

В основном рабочие процессы используются для управления потоком данных. Он моделируется как направленный ациклический граф G=(T,E), где T обозначает набор из n задач в рабочем процессе приложения и E - множество ребер, представляющих поток зависимостей данных [3.37, 3.39]. Выполнение рабочих процессов в основном связано с облачными средами, поскольку они влияют на нагрузку и производительность систем. Выполнение рабочего процесса состоит из двух основных фаз, которые включают в себя фазу предоставления ресурсов и фазу генерации и сопоставления задач [3.39]. Фаза подготовки ресурсов обнаруживает все доступные экземпляры облачных вычислений (как аппаратные, так и программные) и развертывает их, чтобы гарантировать успешное выполнение каждой входящей задачи. Фаза сопоставления задач, с другой стороны, представляет собой процесс, в котором все несопоставленные задачи в метаданных сопоставляются с различными Виртуальными машинами (VM) для выполнения. Планирование направлено на обеспечение эффективного, действенного и точно в срок планирования (EEJSP), что позволяет увеличить пропускную способность, уменьшить период выполнения и общую стоимость выполнения [3.43]. Модель выполнения рабочего процесса на рис. 3.4 аналогична системе управления рабочими процессами Pegasus в [3.58], и Workflowsim в [3.59].

Модель выполнения имеет несколько уровней с различными компонентами выполнения рабочего процесса, такими как пользователь, рабочий процесс Отображения, кластеризация, рабочий процесс управления и, местные Планировщики. Другие компоненты - это монитор отказов, генератор

отказов, рабочие узлы, планировщик, облачная система подготовки IaaS, облако IaaS, хранилище и облачные виртуальные машины. Компоненты модели выполнения рабочего процесса, используемые в данной работе:

Рис. 3.4. Модель выполнения задачи рабочего процесса

1) Пользователь: пользователь может быть отдельным облачным пользователем, группой лиц или организацией, которые отправляют запрос на отправку хосту для принятия решения о планировании.

На данный момент они, вероятно, будут представлять собой множество рабочих задач, некоторые из которых будут зависеть друг от друга [3.41]. Эти поступившие задачи рабочего процесса затем ставятся в очередь на основе инструкций алгоритма.

2) Сопоставитель рабочих процессов: сопоставитель отвечает за создание исполняемого сопоставления рабочих процессов для всех несопоставленных абстрактных рабочих процессов. Сопоставитель развернут для преобразования файлов DAG в расширяемый язык разметки (XML) для

создания списка задач и назначения.

3) Механизм кластеризации. При выполнении задач механизм кластеризации используется для объединения всех поступивших задач в группу задач для планирования в соответствии с конструкцией алгоритмов. Это делается для повышения эффективности системы и снижения накладных расходов [3.59, 3.60].

4) механизм рабочего процесса: как и большинство заданий или задач, которые могут иметь зависимости между собой, механизм рабочего процесса готовит и ставит задачи в очередь для выполнения на основе их зависимостей. Это означает, что дочерняя задача рабочего процесса не может быть запущена до тех пор, пока не будет завершена родительская задача рабочего процесса. Механизм рабочего процесса выпускает задачу в планировщик, когда все его родительские задачи выполняются безопасно.

5) место выполнения: Все поступившие задачи рабочего процесса ставятся в очередь в соответствии с разработанным алгоритмом и отправляются на место выполнения для принятия исполнительных решений.

Сайт выполнения отвечает за заботу обо всех поступивших задачах рабочего процесса, которые должны быть запланированы. Он включает в себя планировщик задач, который выполняет фактическое планирование, сопоставляя все несопоставленные задачи рабочим узлам с помощью алгоритмов, рабочую заметку, монитор отказов и генератор отказов [3.59].

3.3.3. Основные определения

Некоторые из основных терминов, используемых при реализации алгоритма:

1) Общая стоимость: поскольку облачные вычисления используют механизм оплаты по мере использования, то общая стоимость выполнения списка задач рабочего процесса имеет первостепенное значение для оценки системы представление. Уравнение (3.5) представляет собой математи-

ческое описание параметра

$$TC = \left(P_j \bullet PC + \left(\sum_{f \in Fin_j} Size(f) + \sum_{f \in Fout_j} Size(f) \right) \right) \bullet TrC \quad (3.5)$$

где TC - суммарные затраты,

P_J - время обработки работы J,

PC - стоимость обработки,

TrC - стоимость передачи входных файлов (Fin_j) и выходных файлов ($Fout_j$).

2) Время выполнения: это время, затраченное виртуальной машиной (VM) для выполнения задачи рабочего процесса. Время выполнения задачи Wt_i на VM_k вычисляется в (3.6):

$$ET = \frac{TL}{MIPS_{VM_k}} \quad (3.6)$$

где TL - размер задачи,

$MIPS_{VMk}$ – миллион инструкций в секунду виртуальной машины К.

3) ожидаемое время завершения рабочего процесса задачи Wt_i на VM_j вычисляется с помощью (3.7) [3.57]:

$$ECT = ET + loadVM_j \quad (3.7)$$

где ET - время выполнения задачи рабочего процесса Wt_i на VM_j,

$loadVM_j$ - рабочая нагрузка виртуальной машины j в данный момент времени. В табл. 3.1 представлены обозначения и их определения, используемые во всей работе.

Таблица 3.1

Аббревиатуры и их описания

Аббревиатуры	**Описание**
W	рабочий процесс, представленный (DAG)
E	набор ребер с ограничением зависимостей
Wt	Задачи Рабочего Процесса
N	количество задач
k	количество типов виртуальных машин

Аббревиатуры	Описание
ECT	ожидаемое время завершения
DL	Крайний Срок
Dl_{queue}	Крайний Срок Очереди
Wt_{queue}	очереди задач рабочего процесса
EL	Продолжительность выполнения
TC	общая стоимость
Pj	время обработки задания j
PC	Стоимость обработки
TrC	стоимость передачи входных и выходных файлов
Fin_j	передача входных файлов задания j
$Fout_j$	передача выходных файлов задания j
Wt_{entry}	Задачи рабочего процесса ввода
Wt_{exit}	задачи рабочего процесса выхода
TL	Длина задач
$MIPS_{VMk}$	миллион инструкция в секунду виртуальной машины К
COHA	Оптимизированный по стоимости эвристический алгоритм
ET	время выполнения
$loadVM_j$	рабочая нагрузка виртуальной машины j

3.4. Эвристический алгоритм оптимизации затрат (COHA)

3.4.1. Особенности алгоритма

Затраты, связанные с выполнением приложений в облачных ресурсах были предметом озабоченности многих исследователей в области облачных вычислений. При решении этой проблемы изучаются алгоритмы планирования оптимизации затрат в облаке и представляется новый алгоритм разделения задач под названием Оптимизированный по стоимости эвристический алгоритм (COHA) для облачного планировщика для оптимизации стоимости выполнения задач рабочего процесса. В этом алгоритме большие задачи, которые, скорее всего, не будут соответствовать заданным пользователем срокам, разбиваются на подзадачи, чтобы сократить продолжительность планирования. Это делается для того, чтобы все задачи выполнялись в установленные сроки. Псевдокод предложенного алгоритма

находится на рис. 3.5.

Алгоритм COHA

Input: DAG: G=(W,E); W - n рабочих потоков задач (wt_1, wt_2, …, wt_n); VM (vm_1, vm_2, …, vm_n)
Begin
wt_{queue} =(wt_1, wt_2, …, wt_n)
VM_{list}=(vm_1, vm_2, …, vm_n)
While $wt_{queue}\neq\varnothing$ **do**
Вычислить ECT
Создать Dl_{queue} - очередь задач, основанную на дедлайнах
* **Foreach** ECT для wt_i из vm_k **do**
Сравнить ECT для wt_i из vm_k с их дедлайнами
If $ECT_{wti}>Dl_{wti}$ **Then** разбить задачу на две
Else
If $ECT_{wti}\leq Dl_{wti}$
Сопоставить задачи с VM
** Обновить Dl_{queue}
While $Dl_{queue}\neq\varnothing$ **do**
Repeat шаги с * по **
End While
If $Dl_{queue}=\varnothing$
End Else
End Foreach
End While
End

Рис. 3.5. Псевдокод предложенного алгоритма COHA

Алгоритм начинается с определения всех прибывших рабочего процесса задачи в задачи рабочего процесса из очереди с набором задач рабочего процесса Wt=(Wt_1, Wt_2, Wt_3, ..., Wt_n) с соответствующими длинами исполнения (EL001, EL002, EL003, ..., EL00n). Эти задачи должны быть выполнены на облачных ресурсах (виртуальные машины), которые образуют множество виртуальных машин (VM_1, VM_2, ..., VM_K). Вычисляется ожидаемое время завершения (ECT) каждой задачи в очереди задач рабочего процесса, которая должна быть запланирована на виртуальной машине. Для того чтобы задача соответствовала своему крайнему сроку, время

не должно его превышать. Затем создается очередь крайних сроков (Dl_{queue}) для повторной очереди всех поступивших задач в очередь задач рабочего процесса (wt_{queue}) на основе их крайних сроков. После этого крайний срок для каждой задачи сравнивается с ее EL, и если EL Wt_i на VM_k больше крайнего срока, то задача будет разбита на подзадачи и отправлена в готовую очередь для решения проблемы сопоставления. Алгоритм завершается, когда все несопоставленные задачи рабочего процесса в очереди крайнего срока рассмотрены.

3.4.2. Среда моделирования, результаты и обсуждение

WorkflowSim [3.50] - это среда моделирования Java для моделирования и выполнения приложений рабочего процесса как в грид-среде, так и в среде облачных вычислений. Среда была использована многими исследователями для моделирования и оценки алгоритмов [3.40, 3.59, 3.61]. Workflowsim - это расширение CloudSim [3.62], который поддерживает DAG рабочего процесса. Она помогает предлагать среду исполнения и моделирования. Она может моделировать, выполнять, анализировать и оценивать производительность облачного планирования алгоритмы в среде облачных вычислений [3.40]. Она поддерживает кластеризацию через механизм кластеризации, который объединяет задачи в группу задач на основе конструкции алгоритма [3.59]. Удалось адаптировать Workflowsim для моделирования и оценки производительности предложенного алгоритма. Расширены и модифицированы коды базового алгоритма планирования в Workflowsim. Проанализированы три рабочих процесса, включая Montage, CyberShake и SIPHT. Рабочие процессы моделировались разных размеров: сверхмалые (25 или 30 задач), малые (50 или 60 задач), средние (100 задач) и большие (1000 задач) [3.63].

Эти рабочие процессы относятся к разным областям и входят в большие наборы данных, которые структурированы по-разному [3.46]. Ра-

бочие процессы изображены на рис. 3.6. Подробная спецификация параметров моделирования приведена в табл. 3.2.

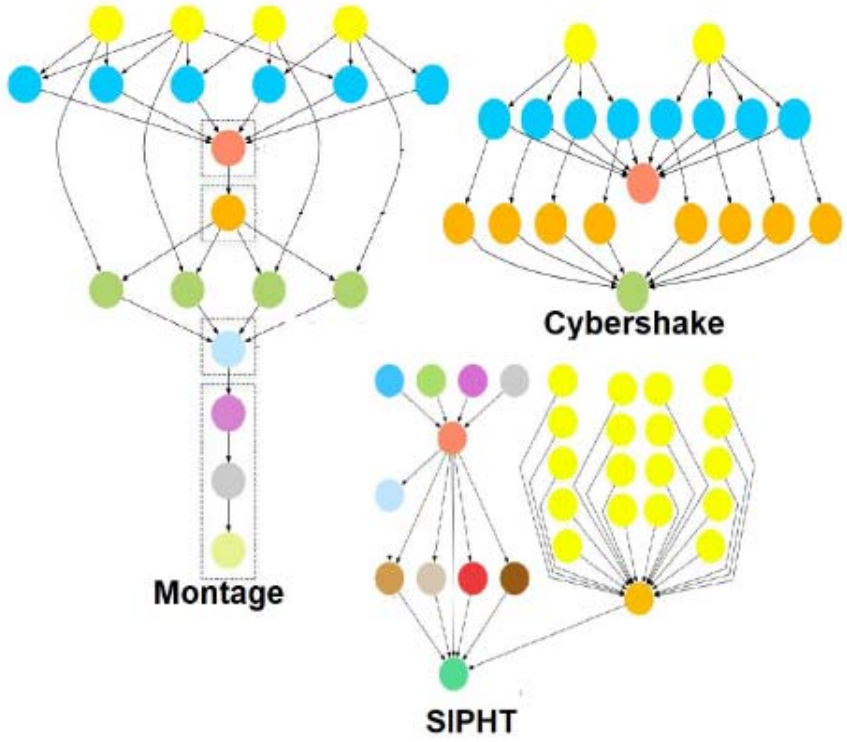


Рис. 3.6. Структура рабочих процессов, используемых в моделировании [3.65]

Для сравнения результатов реализован HSLJF алгоритм, предложенный в [3.57]. Проведено сравнение алгоритма HSLJF и предложенного COHA с использованием трех реальных приложений рабочего процесса, таких как Montage, CyberShake и Sipht. Сравнение проведено для оптимизации затрат на выполнение приложений документооборота на облачных ресурсах. На рис. 3.7 показаны результаты расчета стоимости исполнения как для HSLJF, так и для COHA, использующих рабочий процесс Montage. Предложенный метод эффективно эксплуатировал и использовал все пять виртуальных машин. Это снизило стоимость выполнения на 3,9%.

На рис. 3.8 показаны результаты моделирования стоимости выполнения алгоритмов COHA и HSLJF на CyberShake с четырьмя различными заданиями (30, 50, 100 и 1000) на пяти различных виртуальных машинах.

Среди этих двух алгоритмов COHA продемонстрировал свою способность распределять рабочие нагрузки на различные виртуальные машины равномернее, чем HSLJF, и тем самым снизил стоимость выполнения на 1,2%.

Таблица 3.2

Параметры планирования, используемые при моделировании

Объект, параметры		Значения
Рабочие процессы	Montage	25, 50, 100, 1000
	CyberShake	30, 50, 100, 1000
	SIPHT	30, 60, 100, 1000
Датацентр/ Виртуальная машина	К-во хостов	2
	Архитектура	x86
	ОС	Linux
	Память хоста (MB)	2048
	Хранилище датацентра	1 TB
	Пропускная способность хоста	100 Гбит / с
	Стоимость использования хранилища	0,001
	Стоимость использования процессора	0,3
	Стоимость использования памяти	0,05
	Память виртуальной машины (MB)	512

Наконец, на рис. 3.9 измерена производительность предложенного COHA и HSLJF. Результаты показывают, что COHA может решить проблему планирования задач гораздо лучше, чем алгоритм HSLJF, за счет снижения затрат на исполнение на 32,5% при использовании рабочего процесса SIPHT. Это происходит потому, что COHA реагирует на развертывание виртуальных машин и неопределенность прибытия задач.

Усовершенствование алгоритма заключается в том, что предлагаемый метод всегда разбивает большие задачи на подзадачи и переносит их на виртуальные машины, гарантируя, что все задачи будут завершены раньше срока.

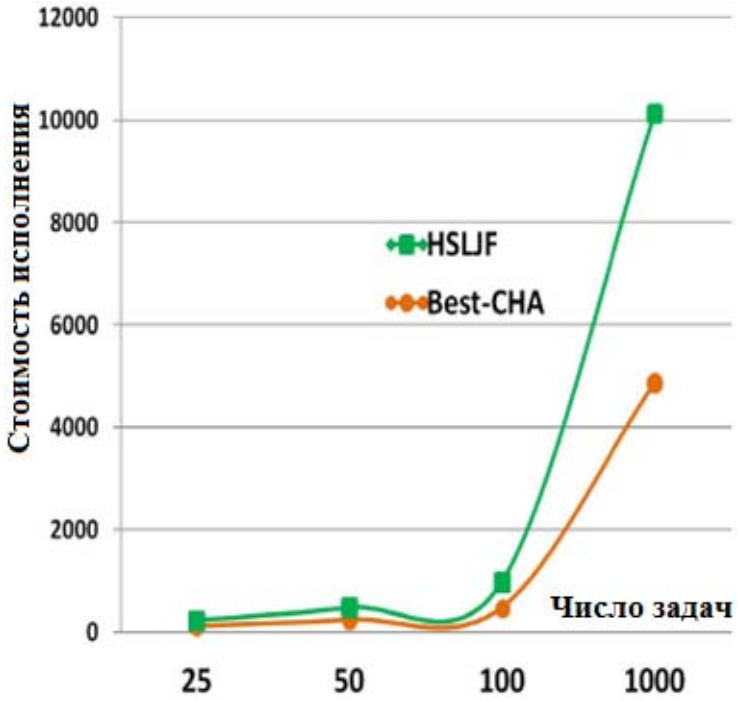


Рис. 3.7. Сравнение затрат на выполнение при различном количестве задач на рабочем процессе Montage

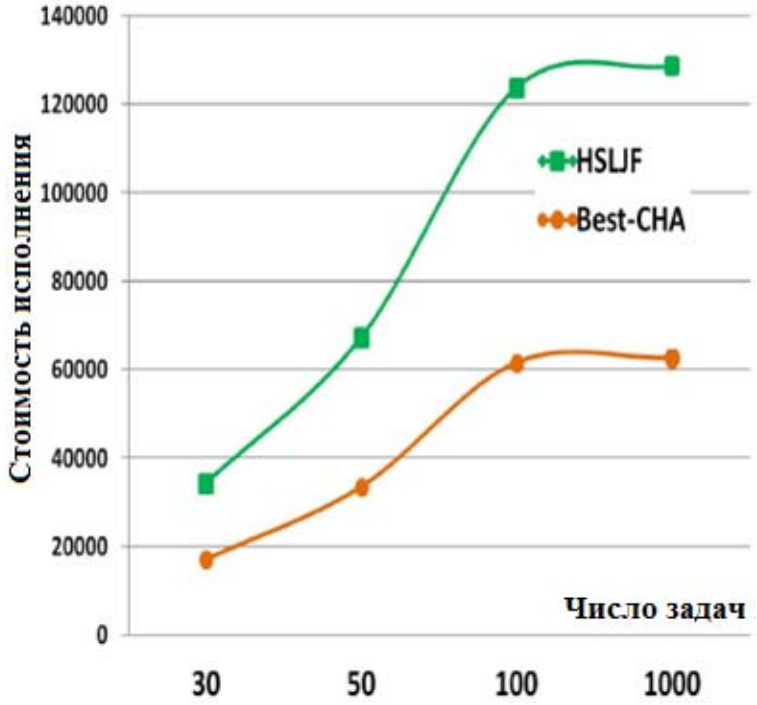


Рис. 3.8. Сравнение стоимости выполнения по различному количеству задач на рабочем процессе CyberShake

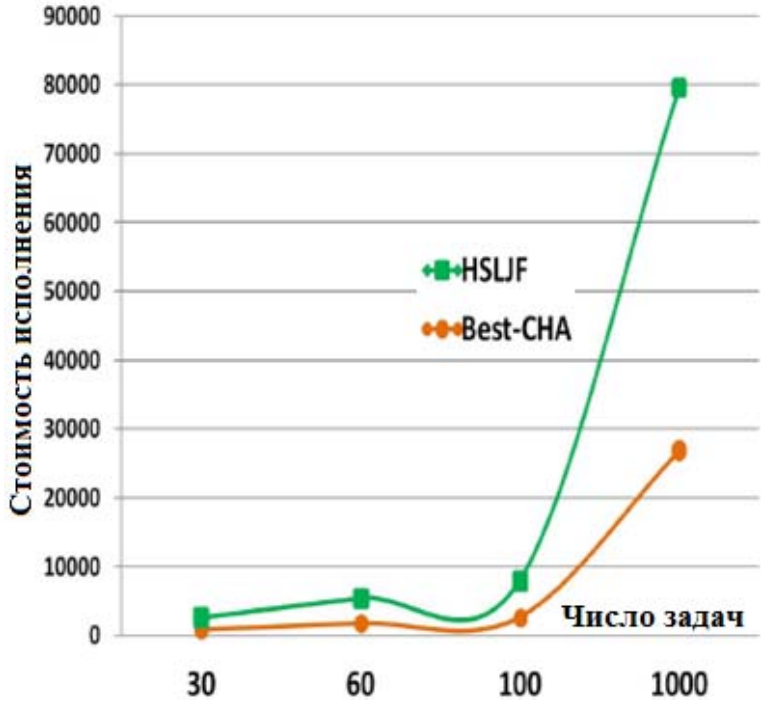


Рис. 3.9. Сравнение затрат на исполнение при различном количестве задач для рабочего процесса SIPHT

3.5. Подход к созданию иерархической стратегии балансировки нагрузки в облачных средах

Облачные вычисления - это предоставление вычислений в качестве услуги, а не продукта, в результате чего общие ресурсы, программное обеспечение и информация предоставляются компьютерам и другим устройствам в качестве утилиты через Интернет. Облачный центр обработки данных можно рассматривать как иерархическую систему по структуре, которая состоит из множества кластеров и каждый кластер содержит одну или несколько физических машин (PM), в каждой PM работает несколько виртуальных машин (VM). Виртуализация и оперативная миграция виртуальных машин между PM являются ключевыми коэффициентами эффективного распределения ресурсов в центрах обработки данных. Живая миграция виртуальной машины из одного PM в другой позволяет реагировать на изменяющиеся потребности в ресурсах виртуальных машин [3.71]. Ко-

гда потребность в ресурсах виртуальных машин увеличивается, они могут быть перенесены в другие PM с более низкой нагрузкой, что позволяет избежать нарушений SLA. По этим причинам миграция виртуальных машин является ключевым компонентом проблемы балансировки нагрузки [3.72]. С другой стороны, миграция виртуальных машин требует времени, создает накладные расходы и может негативно повлиять на выполнение SLA [3.71]. Миграция виртуальной машины может увеличить нагрузку как источника, так и приемника PM, создает дополнительную нагрузку на сеть и делает мигрирующую виртуальную машину менее реактивной во время миграции [3.82]. Поэтому важно поддерживать количество миграций виртуальных машин на разумном уровне. Понимание точного влияния миграции само по себе является сложной проблемой. Следовательно, при распределении VM необходимо найти оптимальный баланс между качеством услуг и стоимостью связи.

Балансировка нагрузки - это механизм распределения нагрузки между различными узлами любой системы [3.76]. Основной целью балансировки нагрузки является оптимальное использование имеющихся ресурсов. Благодаря виртуализации облачные центры обработки данных должны иметь возможность сбалансировать нагрузку на каждом уровне иерархическим образом. Такая гибкость становится ключевым коэффициентом в современных инфраструктурах облачных вычислений, направленных на эффективное распределение нагрузки между ресурсами.

3.5.1. Проблема балансировки нагрузки

Нагрузка - это абстрактное понятие, описывающее загруженность системы, а распределение балансировки нагрузки всех ресурсов параллельной системы называется балансировкой нагрузки. Балансировка нагрузки способствует обеспечению высокой эффективности алгоритма назначения задач, и он может регулировать распределение нагрузки в любое

время, чтобы держать все ресурсы в системе в сбалансированном состоянии. Существует два вида методов балансировки нагрузки всех ресурсов в системе: назначение нагрузки и миграция нагрузки. Назначение нагрузки заключается в правильном назначении пользовательских задач всем ресурсам, чтобы сделать нагрузку системы на все ресурсы примерно равной.

Миграция нагрузки заключается в переносе задач с тяжелонагруженных ресурсов на легконагруженные, чтобы сбалансировать нагрузку на систему и повысить общую производительность. Предложенный там алгоритм балансировки нагрузки используется при миграции нагрузки.

3.5.2. Проектирование системы

Архитектура

Как показано на рис. 3.10, облачная архитектура представляет собой набор центров обработки данных, и каждый центр обработки данных представляет собой конечное множество G кластеров Ck, связанных между собой, где каждый кластер содержит одну или несколько физических машин (PM), соединенных между собой коммутаторами, и в каждом PM работает несколько виртуальных машин (VM). Физические машины в кластерах неоднородны, поэтому нагрузка в одном кластере может быть очень высокой, в то время как в других кластерах может ничего не работать. Более высокая пропускная способность может быть достигнута, если добавить балансировку нагрузки между кластерами, а также балансировку нагрузки между физическими машинами внутри кластера.

Ссылаясь на структуру предложенной модели на рис. 3.10, мы предлагаем иерархическую стратегию балансировки нагрузки, основанную на двух менеджерах: локальном и глобальном.

Локальные менеджеры собирают информацию о нагрузке с физических машин PM и балансируют нагрузку в этой локальной области. Они уравновешивают нагрузку для физических машин.

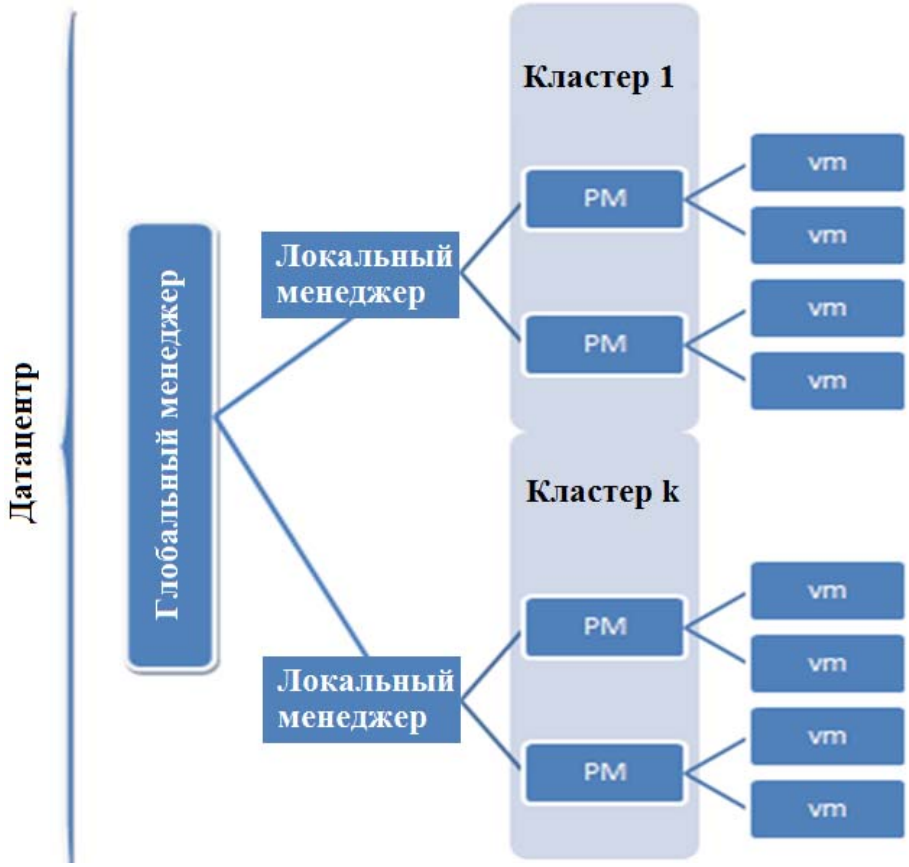


Рис. 3.10. Архитектура предлагаемой стратегии

Глобальный менеджер периодически собирает информацию о нагрузке из каждого кластера. Глобальный менеджер балансирует нагрузку для кластеров и периодически связывается с местными менеджерами для сбора информационной нагрузки каждого кластера.

Этот подход предназначен для улучшения общего использования ресурсов кластера и снижения накладных расходов на межпроцессную коммуникацию в одном кластере. Поэтому задачи этого алгоритма двояки: снижение нагрузки перегруженных машин за счет миграции нагрузки кластеров и снижение затрат на связь за счет сокращения числа миграций между различными федерациями, если система находится в состоянии насыщения.

3.5.3. Предлагаемая стратегия

Предлагаемая работа позволяет разработать иерархическую стратегию на двух уровнях и проектируется: внутрикластерная и межкластерная: внутрикластерная балансировка нагрузки: на этом уровне балансировка нагрузки запускается только тогда, когда некоторые менеджеры VM не могут локально сбалансировать перегрузку своих VM. зная глобальное состояние каждого PM, менеджер PMC может равномерно распределить глобальную перегрузку между своими физическими машинами.

Межкластерная балансировка нагрузки: на этом втором уровне он выполняет глобальную балансировку нагрузки между всеми кластерами облачного центра обработки данных. Он выполняется только в том случае, если на другом уровне не удается достичь полного баланса нагрузки. Важно отметить, что на этом уровне алгоритм всегда успешно балансирует нагрузку на все эти кластеры. Таким образом, бесполезно проверять, являются ли некоторые кластеры все еще несбалансированными.

Основное преимущество этой стратегии заключается в приоритизации локальной балансировки нагрузки сначала внутри кластера, а затем внутри центра обработки данных. Цель этой стратегии соседства состоит в том, чтобы уменьшить количество сообщений между физическими машинами. Как следствие этой цели, накладные расходы, вызванные нашей стратегией, уменьшаются.

Оценка нагрузки и политика обмена информацией

В идеале информация о нагрузке должна отражать текущую загрузку процессора, использование памяти и сетевой трафик узла. Традиционно нагрузка узла в данный момент времени описывалась просто загрузкой процессора. В предлагаемой работе для измерения нагрузки используются загрузка процессора и загрузка памяти.

Предположим, что один кластер k имеет n физических машин в виде $\{p_{mk},1 \dots p_{mk}, n\}$. Предположим, что нагрузка PM равна l_i, а $\overline{l}$ - средняя на-

грузка каждого PM. Когда мы вычисляем стандартное отклонение нагрузки, мы обращаем внимание на физические машины, которые относятся к кластеру.

Стандартное отклонение нагрузки в одном кластере задается так:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(l_i - \overline{l})^2} \tag{3.7}$$

Если значение стандартного отклонения невелико, это означает, что разница каждой нагрузки невелика. Небольшое стандартное отклонение говорит о том, что нагрузка всей системы сбалансирована. Чем меньше значение стандартного отклонения, тем более сбалансирована нагрузка системы.

Определение состояния кластера

Физические машины, входящие в состав одного кластера, могут стать перегруженными или насыщенными, даже если выполняется динамическая миграция виртуальных машин, чтобы избежать узких мест или перегрузки некоторых ресурсов; в этом случае кластер считается в насыщенном состоянии, поэтому уровень другого кластера считается выполняющим использование ресурсов и равномерно распределяющим рабочую нагрузку по всем машинам, входящим в состав центра обработки данных. Два пороговых значения T1 и T2 рассматриваются для определения политики передачи процесса и в соответствии с (3.7); состояние кластера является следующим:

```
If  σ  ≤ T1: Cluster is balanced
Else
      If  σ ≥ T1 and  σ  ≤ T2
      Cluster is overloaded.
        Else
      Cluster is saturated.
      End
End
```

Политика информационного обмена, выбранная для данного иссле-

дования, представляет собой периодическую политику с временным интервалом, который будет установлен в имитационных экспериментах.

Политика передачи процессов

В предлагаемой стратегии это определение включает два этапа. Во-первых, физические машины классифицируются в соответствии с их нагрузкой. Принимается решение, запускать ли процесс на другой машине. Во-вторых, кластеры классифицируются так, чтобы работать как приемник или источник в случае состояния насыщения одного из них.

Классификация нагрузки физической машины

В соответствии с (3.7) и при перегрузке кластера можно рассматривать запуск операции балансировки нагрузки. Определить, находится ли физическая машина в кластере в подходящем состоянии для участия в передаче процесса, как сервер или получатель задач, в зависимости от ее состояния доступности. Мы классифицируем физические машины на перегруженные (источники), нормальные (нейтральные) и недогруженные (приемники).

Классификация нагрузки кластера

Когда кластер находится в насыщенном состоянии, миграция виртуальных машин будет перенесена на другой уровень, поэтому кластеры классифицируются на перегруженные, нормальные и недогруженные, чтобы участвовать соответственно как нейтральные источники или приемники.

3.6. Выводы

В работе исследуется проблема планирования и управления ресурсами в облачных средах. Алгоритмы оптимизации роя частиц являются наиболее известными алгоритмами планирования задач в распределенных системах. Для повышения производительности алгоритма оптимизации роя частиц рекомендуется использовать модифицированный алгоритм оп-

тимизации роя частиц, в котором он объединяется для генерации начальной популяции. Эксперименты показывают, что как R-коэффициент, так и алгоритм PSO показывают правильные результаты. Этот алгоритм может быть использован в среде облачных вычислений для эффективного планирования задач на уже существующих ресурсах, чтобы сократить время выполнения задач. В будущих работах планируется сосредоточиться на развитии процесса разделения приложений или больших задач на небольшие подзадачи, разрабатывая процесс для большей скорости и точного распределения в зависимости от длины задачи. Это приведет к повышению точности и скорости распределения ресурсов и планирования задач, а тем самым и к повышению эффективности облачных вычислительных систем.

В исследовании разработан новый алгоритм планирования рабочего процесса для задачи планирования задач в облаке IaaS, основанный на эвристическом подходе, и названный оптимизированным по стоимости эвристическим алгоритмом (COHA). COHA может эффективно разделить большие задачи на подзадачи. Это позволяет всем задачам уложиться в установленные сроки без каких-либо задержек. Предложенный метод реализован и апробирован в Workflowsim. Мы сравниваем производительность предложенного алгоритма с алгоритмом HSLJF, используя три рабочих процесса, включая Montage, CyberShake и SIPHT. COHA может эффективно переносить все задачи на виртуальную машину с помощью гарантия того, что все задачи будут выполнены в установленные сроки.

Предложена иерархическая стратегия балансировки нагрузки для облачных центров обработки данных. Основная цель состоит в том, чтобы добиться значительного преимущества в среднем времени отклика и минимальных затрат на коммуникацию между кластерами. Поскольку эта работа является всего лишь концептуальной моделью, требуется еще больше работы для реализации алгоритма и решения проблем.

Список источников к главе 3

3.1. Buyya R., Yeo C.S., Venugopal S., Broberg J., Brandic I. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems. 2009. 25(6), pp.599–616.

3.2. Hayes B. Cloud computing. Communications of the ACM, 2008, 51(7), pp.9–11.

3.3. Kennedy J., Eberhart R. Particle swarms optimization. IEEE International Conference on Neural Networks, 1995. 4, pp.1942–1948.

3.4. Pandey S., Wu L., Guru S.M., Buyya R. A Particle Swarm Optimization-Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments. 2010 24th IEEE international Conference on Advanced Information Networking and Applications (AINA), IEEE. 2010. pp.400–407.

3.5. Liu P. Cloud computing definition and characteristics. China cloud computing. 2009. http://www.chinacloud.cn/2009-2-25.

3.6. Kumar P., Verma A. Independent task scheduling in cloud computing by improved genetic algorithm. International Journal of Advanced Research in Computer Science and Software Engineering, 2012. 2(5).

3.7. Roy P., Mejbah M., Das N. Heuristic based task scheduling in multiprocessor systems with genetic algorithm by choosing the eligible processor. International Journal of Distributed and Parallel Systems (IJDPS). 2012. 3(4).

3.8. Chalack S.A., Razavi S.N., Harounabadi A. Job scheduling on the grid environment using max-min firefly algorithm. International Journal of Computer Applications Technology and Research. 2014. 3(1), pp.63–67.

3.9. Vinothina V., Sridaran R., Ganapathi P. A survey on resource allocation strategies in cloud computing. International Journal of Advanced Computer Science and Applications, 2012. 3(6), pp.97–104.

3.10. Alkayal E.S., Jennings N.R., Abulkhair M.F. Efficient Task Scheduling Multi-Objective Particle Swarm Optimization in Cloud Computing. 2016 IEEE 41st Conference on Local Computer Networks Workshops (LCN Workshops). 2016. November; IEEE. pp.17–24.

3.11. Buyya A.R., Nath B. Nature's Heuristics for Scheduling Jobs on Computational Grids. 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000), 2000. India.

3.12. Zhang L., Chen Y., Yang B. Task Scheduling Based on PSO Algorithm in Computational Grid. 2006 Proceedings of the 6th International Conference on Intelligent Systems Design and Applications, Jinan, China. 2006. p.2.

3.13. Kalra M., Singh S. A review of metaheuristic scheduling techniques in cloud computing. Egyptian Informatics Journal, 2015. 16(3), pp.275–295.

3.14. Verma A., Kaushal S. Bi-criteria priority based particle swarm optimization workflow scheduling algorithm for cloud. 2014 Recent Advances in Engineering and Computational Sciences (RAECS). IEEE. 2014. pp.1–6.

3.15. Beegom A.A., Rajasree M.S. A Particle Swarm Optimization Based Pareto Optimal Task Scheduling in Cloud Computing. International Conference in Swarm Intelligence, Cham, Springer. 2014, pp.79–86.

3.16. Jun Xue S., Wu W. Scheduling workflow in cloud computing based on hybrid particle swarm algorithm. Indonesian Journal of Electrical Engineering, 2012. 10(7), pp.1560–1566.

3.17. Guo L., Zhao S., Shen S., Jiang C. Task scheduling optimization in cloud computing based on heuristic algorithm. Journal of Networks, 2012. (7), pp.547–553.

3.18. Varalakshmi P., Ramaswamy A., Balasubramanian A., Vijaykumar P. An optimal workflow based scheduling and resource allocation in cloud. Advances in Computing and Communications, First International Conference, ACC, 2011. pp.411–420.

3.19. Zhong H., Tao K., Zhang X. An Approach to Optimized Resource Scheduling Algorithm for Open-Source Cloud Systems. Fifth Annual China Grid Conference. 2010.

3.20. Selvarani S., Sadhasivam G. Improved Cost-Based Algorithm for Task Scheduling in Cloud Computing. IEEE International Conference on Computational Intelligence and Computing Research (ICCIC). 2010.

3.21. Liu Z., Wang X. A pso-based algorithm for load balancing in virtual machines of cloud computing environment. Advances in Swarm Intelligence, ser. Lecture Notes in Computer Science, Berlin, Heidelberg, Springer. 2012. Vol. 7331, pp.142–147.

3.22. Zhan S., Huo H. Improved PSO-based task scheduling algorithm in cloud computing. Journal of Information and Computational Science, 2012. 9(13), pp.3821–3829.

3.23. Liu J., Guo Luo X., Zhang X.M.F. Job scheduling algorithm for cloud computing based on particle swarm optimization. Advanced Materials Research, 2013. 662, pp.957–960.

3.24. Jeyarani R., Nagaveni N., Vasanth Ram R. Design and implementation of adaptive power-aware virtual machine provisioner (APA-VMP) using swarm intelligence. Future Generation Computer Systems, 2012. 28(5), pp.811–821.

3.25. Liu Y., Zhu H. A survey of the research on power management techniques for high-performance systems. Software Practice and Experience, 2010. 40(11), pp.943–964.

3.26. Feller E., Rilling L., Morin C. Energy-Aware Ant Colony Based Workload Placement in Clouds. 12th International Conference on Grid Computing, ser. Grid, IEEE Computer Society. 2011. Vol. 11, pp.26–33.

3.31. Patel, G., Mehta, R., and Bhoi, U., “Enhanced load balanced min-min algorithm for static meta task scheduling in cloud computing,” Procedia Computer Science, vol. 57, pp. 545–553, 2015.

3.32. Bitam, S., “Bees life algorithm for job scheduling in cloud comput-

ing," in Proceedings of The Third International Conference on Communications and Information Technology, 2012, pp. 186–191.

3.33. Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., and Stoica, I., "Above the clouds: A Berkeley view of cloud computing," Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS, vol. 28, no. 13, p. 2009, 2009.

3.34. InfoTech, "What is cloud computing," IBM Journal of Research and Development, vol. 60, no. 4, pp. 41–44, 2012.

3.35. Savu, L., "Cloud computing: Deployment models, delivery models, risks and research challenges," in 2011 International Conference on Computer and Management (CAMAN). IEEE, 2011, pp. 1–4.

3.36. Masdari, M., ValiKardan, S., Shahi, Z., and Azar, S. I., "Towards workflow scheduling in cloud computing: a comprehensive analysis," Journal of Network and Computer Applications, vol. 66, pp. 64–82, 2016.

3.37. Zhou, X., Zhang, G., Sun, J., Zhou, J., Wei, T., and Hu, S., "Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based heft," Future Generation Computer Systems, vol. 93, pp. 278–289, 2019.

3.38. Rodriguez, M. A. and Buyya, R., "Budget-driven scheduling of scientific workflows in IAAS clouds with fine-grained billing periods," ACM Transactions on Autonomous and Adaptive Systems (TAAS), vol. 12, no. 2, pp. 1–22, 2017.

3.39. Sahni, J. and Vidyarthi, D. P., "A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment," IEEE Transactions on Cloud Computing, vol. 6, no. 1, pp. 2–18, 2015.

3.40. Anwar, N. and Deng, H., "Elastic scheduling of scientific workflows under deadline constraints in cloud computing environments," Future Internet, vol. 10, no. 1, p. 5, 2018.

3.41. Nasr, A. A., El-Bahnasawy, N. A., Attiya, G., and El-Sayed, A., "Costeffective algorithm for workflow scheduling in cloud computing under deadline constraint," Arabian Journal for Science and Engineering, vol. 44, no. 4, pp. 3765–3780, 2019.

3.42. Manasrah, A. M. and Ba Ali, H., "Workflow scheduling using hybrid gapso algorithm in cloud computing," Wireless Communications and Mobile Computing, vol. 2018, 2018.

3.43. Liu, J., Pacitti, E., Valduriez, P., and Mattoso, M., "Parallelization of scientific workflows in the cloud," 2014.

3.44. Juve, G., Chervenak, A., Deelman, E., Bharathi, S., Mehta, G., and Vahi, K., "Characterizing and profiling scientific workflows," Future Generation Computer Systems, vol. 29, no. 3, pp. 682–692, 2013.

3.45. Sossa, M. A. R., "Resource provisioning and scheduling algorithms for scientific workflows in cloud computing environments," Ph.D. dissertation,

University of Melbourne, Department of Computing and Information Systems, 2016.

3.46. Li, Z., Ge, J., Hu, H., Song, W., Hu, H., and Luo, B., "Cost and energy aware scheduling algorithm for scientific workflows with deadline constraint in clouds," IEEE Transactions on Services Computing, vol. 11, no. 4, pp. 713–726, 2015.

3.47. Chawla, Y. and Bhonsle, M., "A study on scheduling methods in cloud computing," International Journal of Emerging Trends & Technology in Computer Science (IJETTCS), vol. 1, no. 3, pp. 12–17, 2012.

3.48. Garey, M. R., "Computers and intractability: A guide to the theory of np-completeness," Revista Da Escola De Enfermagem Da USP, vol. 44, no. 2, p. 340, 1979.

3.49. Awad, A., El-Hefnawy, N., and Abdel kader, H., "Enhanced particle swarm optimization for task scheduling in cloud computing environments," Procedia Computer Science, vol. 65, pp. 920–929, 2015.

3.50. Rimal, B. P. and Maier, M., "Workflow scheduling in multi-tenant cloud computing environments," IEEE Transactions on parallel and distributed systems, vol. 28, no. 1, pp. 290–304, 2016.

3.51. Haidri, R. A., Katti, C. P., and Saxena, P. C., "Cost effective deadline aware scheduling strategy for workflow applications on virtual machines in cloud computing," Journal of King Saud University-Computer and Information Sciences, 2017.

3.52. Elsherbiny, S., Eldaydamony, E., Alrahmawy, M., and Reyad, A. E., "An extended intelligent water drops algorithm for workflow scheduling in cloud computing environment," Egyptian informatics journal, vol. 19, no. 1, pp. 33–55, 2018.

3.53. Kalra, M. and Singh, S., "A review of metaheuristic scheduling techniques in cloud computing," Egyptian informatics journal, vol. 16, no. 3, pp. 275–295, 2015.

3.54. Jiang, Y., Huang, Z., and Tsang, D. H., "Towards max-min fair resource allocation for stream big data analytics in shared clouds," IEEE Transactions on Big Data, vol. 4, no. 1, pp. 130–137, 2016.

3.55. Fard, H. M., Prodan, R., Barrionuevo, J. J. D., and Fahringer, T., "A multi-objective approach for workflow scheduling in heterogeneous environments," in 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012). IEEE, 2012, pp. 300–309.

3.56. Zheng, W., Qin, Y., Bugingo, E., Zhang, D., and Chen, J., "Cost optimization for deadline-aware scheduling of big-data processing jobs on clouds," Future Generation Computer Systems, vol. 82, pp. 244–255, 2018.

3.57. Alworafi, M. A., Dhari, A., El-Booz, S. A., Nasr, A. A., Arpitha, A., and Mallappa, S., "An enhanced task scheduling in cloud computing based on hybrid approach," in Data Analytics and Learning. Springer, 2019, pp. 11–25.

3.58. Deelman, E., Vahi, K., Juve, G., Rynge, M., Callaghan, S., Maech-

ling, P. J., Mayani, R., Chen, W., Da Silva, R. F., Livny, M. et al., “Pegasus, a workflow management system for science automation,” Future Generation Computer Systems, vol. 46, pp. 17–35, 2015.

3.59. Cao, J., Wen, L., and Liu, X., Process-Aware Systems: First International Workshop, PAS 2014, Shanghai, China, October 17, 2014. Proceedings. Springer, 2015, vol. 495.

3.60. Chen, W. and Deelman, E., “Workflowsim: A toolkit for simulating scientific workflows in distributed environments,” in 2012 IEEE 8th International Conference on E-Science. IEEE, 2012, pp. 1–8.

3.61. Voevodin, V. and Sobolev, S., Supercomputing: Third Russian Supercomputing Days, RuSCDays 2017, Moscow, Russia, September 25–26, 2017, Revised Selected Papers. Springer, 2017, vol. 793.

3.62. Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., and Buyya, R., “Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” Software: Practice and experience, vol. 41, no. 1, pp. 23–50, 2011.

3.63. Rodriguez, M. A. and Buyya, R., “Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms,” Future Generation Computer Systems, vol. 79, pp. 739–750, 2018.

3.64. Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M.-H., and Vahi, K., “Characterization of scientific workflows,” in 2008 third workshop on workflows in support of large-scale science. IEEE, 2008, pp. 1–10.

3.65. Mehta, G., Juve, G., and Chen, W., “Workflow generator,” URL: https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator, 2009.

3.71. Amar, M., Anurag, K., Rakesh, K., Rupesh, K., Prashant, Y.: SLA driven load balancing for web applications in cloud computing environment. Inf. Knowl. Manage. 1(1), 5–13 (2011)

3.72. Mann, Z.Б.: Allocation of virtual machines in cloud data centers—a survey of problem models and optimization algorithms. ACM Comput. Surv. (CSUR), 48(1), 11 (2015)

3.73. Jung, G., Hiltunen, M.A., Joshi, K.R., Schlichting, R.D., Mistral, C.P.: Dynamically managing power, performance, and adaptation cost in cloud infrastructures. In: IEEE 30th International Conference on Distributed Computing Systems (ICDCS), pp. 62–73 (2010)

3.74. Katyal, M., Mishra, A.: A comparative study of load balancing algorithms in cloud computing environment. arXiv preprint arXiv:1403.6918 (2014)

3.75. Tian, W., Zhao, Y., Zhong, Y., Xu, M., Jing, C.: A dynamic and integrated load-balancing scheduling algorithm for cloud datacenters. In: 2011 IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS), pp. 311–315. IEEE (2011)

3.76. Malhotra, M., Singh, A.: Adaptive framework for load balancing to improve the performance of cloud environment. In: 2015 IEEE International Conference on Computational Intelligence & Communication Technology

(CICT), pp. 224–228. IEEE (2015)

3.77. Hao, Y., Liu, G., Lu, J.: Three levels load balancing on cloudsim. Int. J. Grid Distrib. Comput. 7(3), 71–88 (2014)

3.78. Wang, S.C., Yan, K.Q., Liao, W.P., Wang, S.S.: Towards a load balancing in a three-level cloud computing network In: 2010 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT), vol. 1, pp. 108–113. IEEE (2010)

3.79. Dhurandher, S.K., Obaidat, M.S., Woungang, I., Agarwal, P., Gupta, A., Gupta, P.: A clusterbased load balancing algorithm in cloud computing. In: 2014 IEEE International Conference on Communications (ICC), pp. 2921–2925. IEEE (2014)

3.80. Culler, D.E., Karp, R.M., Patterson, D.A., Sahay, A., Santos, E.E., Schauser, K.E., Subramonian, R., von Eicken, T.: LogP: a practical model of parallel computation (1993)

3.81. Culler, D.E., Karp, R.M., Patterson, D.A., Sahay, A., Santos, E.E., Schauser, K.E., Subramonian, R., von Eicken, T.: LogP: a practical model of parallel computation. Commun. ACM 39(11), 78–85 (1996)

3.82. Xu, G., Pang, J., Fu, X.: A load balancing model based on cloud partitioning for the public cloud. Tsinghua Sci. Technol. 18(1), 34–39 (2013)

3.83. Smith, J., Nair, R.: The architecture of virtual machines. Computer 38(50), 32–38 (2005)

3.84. Chaoqun, X., Yi, Z., Wei, Z.: A load balancing algorithm with key resource relevance for virtual cluster. Int. J. Grid Distrib. Comput. 6(5), 1–16 (2013)

3.85. Xu, M., Tian, W., Buyya, R.: A survey on load balancing algorithms for VM placement in cloud computing. arXiv preprint arXiv:1607.06269 (2016)

3.86. Eager, D.L., Lazowska, E.D., Zahorjan, J.: Adaptive load sharing inhomogeneous distributed systems. IEEE Trans. Softw. Eng. 12(5), 662–675 (1986)

3.87. Badidi, E.: Architecture and services for load balancing in object distributed systems. Ph.D. thesis, Faculty of High Studies, University of Montreal, Mai (2000)

4. Планирование рабочих процессов с оптимизацией передачи данных в облачных центрах обработки данных на основе оптимизации роя частиц

Облака типа «Infrastructure as a service» (IaaS) предлагают огромные возможности для решения масштабных научных задач. Выполнение рабочих процессов в таких средах может быть дорогостоящим по времени, если они не запланированы должным образом. Хотя планирование рабочих процессов в облаке широко изучено, большинство подходов фокусируется на двух требованиях пользователя к качеству обслуживания, а именно на времени выполнения и затратах. Необходимо учитывать и другие важные особенности облачных вычислений, гетерогенные облачные центры обработки данных. Цель - оптимизировать время передачи данных при минимизации времени выполнения и повышении надежности. Основываясь на моделировании, результаты показывают значительное улучшение с точки зрения времени выполнения, передаваемых данных и надежности по сравнению с проверенным методом HEFT (гетерогенное самое раннее время выполнения) для эталонных рабочих процессов.

4.1. Формализация задачи планирования

Расписание определяется с точки зрения отображения приложений, инфраструктуры ресурсов и критериев производительности. В данной работе критерием является время выполнения и надежность.

4.1.1. Контекст

Подход предлагается в контексте облака IaaS с несколькими центрами обработки данных, где приложения рабочего процесса должны быть запланированы для выполнения в разных центрах обработки данных. Чтобы достичь минимизации времени выполнения приложения при одновремен-

ном повышении надежности, моделируется время передачи данных при вычислении времени выполнения и учитываем надежность связи (надежность канала) для выражения надежности ресурсов.

4.1.2. Модель приложения

Приложение рабочего процесса (w) моделируется как ориентированный ациклический граф (DAG):

$G = (N; E)$,

где N - множество n узлов, обозначающих задачи t_i ($1 \le i \le n$), а E - набор направленных ребер. Ребро $e(i,j) \in E$ соответствует зависимости между задачей t_i и задачей t_j, в которой t_i является непосредственной родительской задачей t_j, а t_j - непосредственной дочерней задачей t_i. Дочерняя задача не может быть выполнена до тех пор, пока не будут выполнены все ее родительские задачи. Матрица данных размерности n*n представляет объем данных, которыми обмениваются задачи.

4.1.3. Модель облачных ресурсов

Представим ресурсную модель, которая учитывает время передачи данных между несколькими центрами обработки данных, чтобы компьютерные ресурсы могли быть развернуты на разных региональных центрах обработки данных. Кроме того, рассматривается надежность как компьютерных ресурсов, так и сетевых связей.

Модель времени

В облаке IaaS компьютерные ресурсы создаются в виде виртуальных машин (VM). Как правило, облачные провайдеры предлагают несколько типов виртуальных машин. Каждый тип vm_i определяется с точки зрения его вычислительной мощности CC_{vmi}. Предполагаем, что для каждого типа виртуальной машины можно оценить ее вычислительную мощность в опе-

рациях с плавающей запятой в секунду (FLOPS). Таким образом, можно оценить время выполнения задачи на данном типе виртуальной машины. Кроме того, определим время вычислений $CT(t_i,vm_p)$ как расчетное время выполнения задачи t_i в виртуальной машине типа vm_p в соответствии с ее мощностью. I_{ti} аналогично определяется в операциях с плавающей запятой (FLOP) и вычисляется из уравнения (4.1):

$$CT(t_i, vm_p) = \frac{I_{t_i}}{CC_{vm_p}} \tag{4.1}$$

В нескольких облачных центрах обработки данных виртуальные машины развертываются в территориально разных регионах. Поэтому время передачи данных становится значительным и напрямую влияет на время отклика приложений рабочего процесса. Следовательно, необходимо учитывать эти времена при расчете времени выполнения рабочего процесса. Рассмотрим U центров обработки данных в облачной инфраструктуре.

Определим время передачи $TT_{(i;j)}$ между двумя задачами, соответствующее весу ребра $(i,j)\in E$ в графе приложения DAG, которое соответствует времени, затраченному на передачу данных из задачи t_i (выполняемой на vm_p, размещенной в центре обработки данных U_a) в задачу t_j (выполняется на vm_k, поданном в дата-центр U_b), как в следующем уравнении:

$$TT_{(i,j)} = \frac{data_{ij}}{Transfer_{rate_{(p,k)}}} \tag{4.2}$$

где $data_{ij}$ - размер данных, полученных задачей t_i и переданных в t_j .

Поскольку виртуальные машины расположены в разных центрах обработки данных, скорости передачи $Transfer_{rate_{(p,k)}}$ между ними неоднородны. При выполнении двух взаимодействующих задач на одной виртуальной машине время передачи равно нулю, а при запуске на разных виртуальных машинах, размещенных в одном центре обработки данных, время передачи равно нулю. Таким образом, предлагается новая модель, описы-

вающая несколько платформ облачных центров обработки данных. В модели общее время вычислений $TCT(t_i,vm_p)$ задачи в виртуальной машине вычисляется так, как показано в (4.3):

$$TCT(t_i, vm_p) = CT(t_i, vm_p) + \sum_{i=1}^{m} s_m r_m TT_{(i,j)} \quad (4.3)$$

В этом уравнении m - число ребер, в которых t_i является родительской задачей. Логическое значение s_m равно 0 всякий раз, когда t_i и t_j работают на одной и той же виртуальной машине (p=k) или 1 в противном случае. Кроме того, r_m равно 0 всякий раз, когда vm_p и vm_k размещаются в одном и том же центре обработки данных (U_a=U_b) или 1 в противном случае.

Общее время вычислений TCT(w) всех задач в рабочем процессе определяется из (4.4):

$$TCT(w) = \max\{TCT(t_i; vm_p)\} \quad (4.4)$$

Модель надежности

В данном исследовании под надежностью будем понимать частоту отказов. Предположим, что отказы виртуальных машин и каналов связи в системе имеют пуассоновское распределение [4.34]. Свяжем частоту отказов для каждой виртуальной машины и канала связи. Пусть λ_p обозначает интенсивность отказов vm_p в единицу времени, а $\lambda_{L_{(k;p)}}$ - интенсивность отказов линии связи $L_{(k;p)}$ между vm_k и vm_p. Эти показатели отказов могут быть получены из службы профилирования ресурса, журнала файлов и методов статистического прогнозирования [4.26].

В этой модели надежность задачи t_i равна вероятности ее успешного выполнения на виртуальной машине, которой она назначена, и вероятности успешной передачи данных этой задаче от ее непосредственных предшественников. Используем аппроксимацию надежности сети для передачи данных, предложенную в [4.22, 4.26]. Надежность $R(t_i; vm_p)$ t_i, выполненного или отображенного на vm_p, вычисляется следующим образом:

$$R(t_i, vm_p) = e^{-\lambda_p CT(t_i, vm_p)} \quad (4.5)$$

Поскольку время связи между задачами, назначенными на одной и той же виртуальной машине, равно нулю, то передача данных считается безотказной. Таким образом, элемент $R_{(j;i)}$, соответствующий надежности передачи между t_j, выполняемой на vm_k, и t_i, выполняемой на vm_p, может быть вычислен следующим образом:

$$R_{(i,j)} = e^{-\lambda_{L_{(k,p)}} TT_{(j,i)}} \quad (4.6)$$

С учетом (4.5) и (4.6) надежность $R(t_i)$ задачи t_i, может быть определена из (4.7), где $predt_i$ - множество задач-предшественников задачи t_i:

$$R(t_i) = \prod_{t_j \in pred_{t_i}} R(t_i, vm_p) R_{(i,j)} \quad (4.7)$$

Надежность рабочего процесса ω:

$$R(\omega) = \prod_{t_i \in N} R(t_i) \quad (4.8)$$

Критерий модели

Цель подхода к планированию состоит в том, чтобы повысить надежность и минимизировать время выполнения и передачи данных приложения рабочего процесса. Чтобы максимизировать качество планирования рабочего процесса, необходимо минимизировать следующий критерий:

$$R_{index}(\omega) = \sum_{vm_p \in VM} \sum_{t_i \in N} (\lambda_p CT(t_i, vm_p) + \lambda_{L_{(k,p)}} TT_{(j,i)}) \quad (4.9)$$

Задача может быть формально определена следующим образом: найти расписание S для рабочего процесса w с минимальным TCT(w) и максимальным R(w) (таким образом, с минимальным $R_{index}(w)$).

Чтобы решить эту проблему, предлагается стратегия, основанную на популяционной метаэвристике PSO. PSO имеет меньше параметров, чем другие метаэвристики, такие как GA или ACO, и быстро сходится.

4.2. Планирование с учетом надежности на основе оптимизации роя частиц

Дадим краткое описание эвристики PSO. Затем выделим дискретный PSO, ориентированный на надежность, RDPSO.

PSO - это популяционная метаэвристика, вдохновленная стадными такими моделями социального поведения. Она была представлена Кеннеди и Эберхартом в 1995 г. [4.15]. Алгоритм стохастической оптимизации основан на концепции, называемой частицей. Частица соответствует особи, которая обладает способностью перемещаться или летать в заданном проблемном пространстве. Каждая частица характеризуется положением X, скоростью V и значением пригодности.

Положения частиц рассматриваются как потенциальные решения задачи и оцениваются с помощью фитнес-функции. Задача - оптимизировать фитнес-функцию и дать ей возможность отслеживать лучшую позицию pbest и глобально лучшую позицию gbest. Скорость частицы представляет собой направление и величину следующего движения. Она вычисляется с учетом фактической скорости, pbest и gbest на каждом шаге алгоритма. Движение частиц к этим значениям обусловлено их фактической скоростью (взвешенной коэффициентом инерции), а также промежутками между их положением и значением pbest и gbest (взвешенными случайными числами для когнитивного и социального коэффициентов соответственно).

На каждой итерации алгоритма PSO генерируется значение целевой функции. Затем обновляются pbest и gbest. После этого положение и скорость частицы обновляются на основе уравнений (4.10) и (4.11) соответственно.

$$v_i^{k+1} = \omega v_i^k + c_1 r_1 [pbest_i^k - x_i^k] + c_2 r_2 [qbest_i^k - x_i^k] \quad (4.10)$$

$$x_i^{k+1} = x_i^k + v_i^{k+1} \quad (4.11)$$

где v_i^k - скорость частицы i на итерации k;

v_i^{k+1} - скорость частицы i на итерации k+1;

ω - инерционный вес;

x_i^k - текущее положение частицы i на итерации k;

x_i^{k+1} - положение частицы i на итерации k+1;

$pbest_i$ - лучшее положение частицы i;

gbest - положение лучшей частицы в популяции;

c_1; c_2 - коэффициенты ускорения;

r_1; r_2 - случайное число между 0 и 1.

Алгоритм PSO повторяется до тех пор, пока не будет удовлетворен критерий остановки, который обычно представляет собой заранее определенное значение целевой функции, оцениваемое как приемлемое, или заданное максимальное число итераций.

В работе используется дискретный вариант PSO под названием discrete particle swarm Optimization (DPSO), который обрабатывает зависимости задач, как это было предложено в [4.2, 4.20].

Каждое положение частицы - это возможное решение. В нашей модели частица имеет n измерений, соответствующих n задачам рабочего процесса. N задач должны быть назначены m виртуальным машинам, где m<n. Схема расположения частиц, показанная на рис. 4.1, представляет собой возможное графическое решение.

t_1	t_2	t_3	…	t_n
vm_2	vm_1	vm_m	…	vm_1

Рис. 4.1. Сопоставление частиц

Подход, основанный на DPSO, стремится оптимизировать время выполнения и надежность одновременно. Подход назван RDPSO для обеспечения надежности DPSO. Для оценки решения с RDPSO, фитнес-функция выражается через обе метрики - время выполнения и надежность.

Использовали весовые коэффициенты взвешенной суммы для включения времени выполнения и индекса надежности в единую целевую функцию, как указано в (4.12):

$$\text{Fitness} = \alpha_1 \frac{TCT}{T_{max}} + \alpha_2 \frac{R_{index}}{R_{index_{max}}} \tag{4.12}$$

Время выполнения и коэффициент надежности нормализуются в этом уравнении. Нормализация позволяет выровнять шкалу сравниваемых величин. На самом деле нормализация необходима для того, чтобы функция пригодности не была искажена одним из показателей, имеющих высокое значение.

В алгоритме значения T_{max} и $R_{indexmax}$ представляют собой максимальное значение времени выполнения и надежности соответственно, найденное на предыдущей итерации.

Параметры α_1, α_2 - это относительные веса, присвоенные времени выполнения и индексу надежности в соответствии с требованиями пользователя к компромиссу, например $\alpha_1+\alpha_2=1$.

Структурная схема алгоритма RDPSO представлена на рис. 4.2 с учетом особенностей фитнес-функции.

В алгоритме PSO для оценки качества решения целевая функция оценивается после вычисления TCT(w) и R_{index}(w) с использованием уравнений (4.4) и (4.9) соответственно.

4.3. Результаты экспериментов

В разделе представлены результаты экспериментов, реализованные для оценки эффективности предложенного подхода. Разработана имитационная модель, основанная на инструмент CloudSim [4.3] для моделирования облака с несколькими платформами центров обработки данных. Использованы различные рабочие процессы из разных областей с разной структурой и реализован стандартный подход для сравнения.

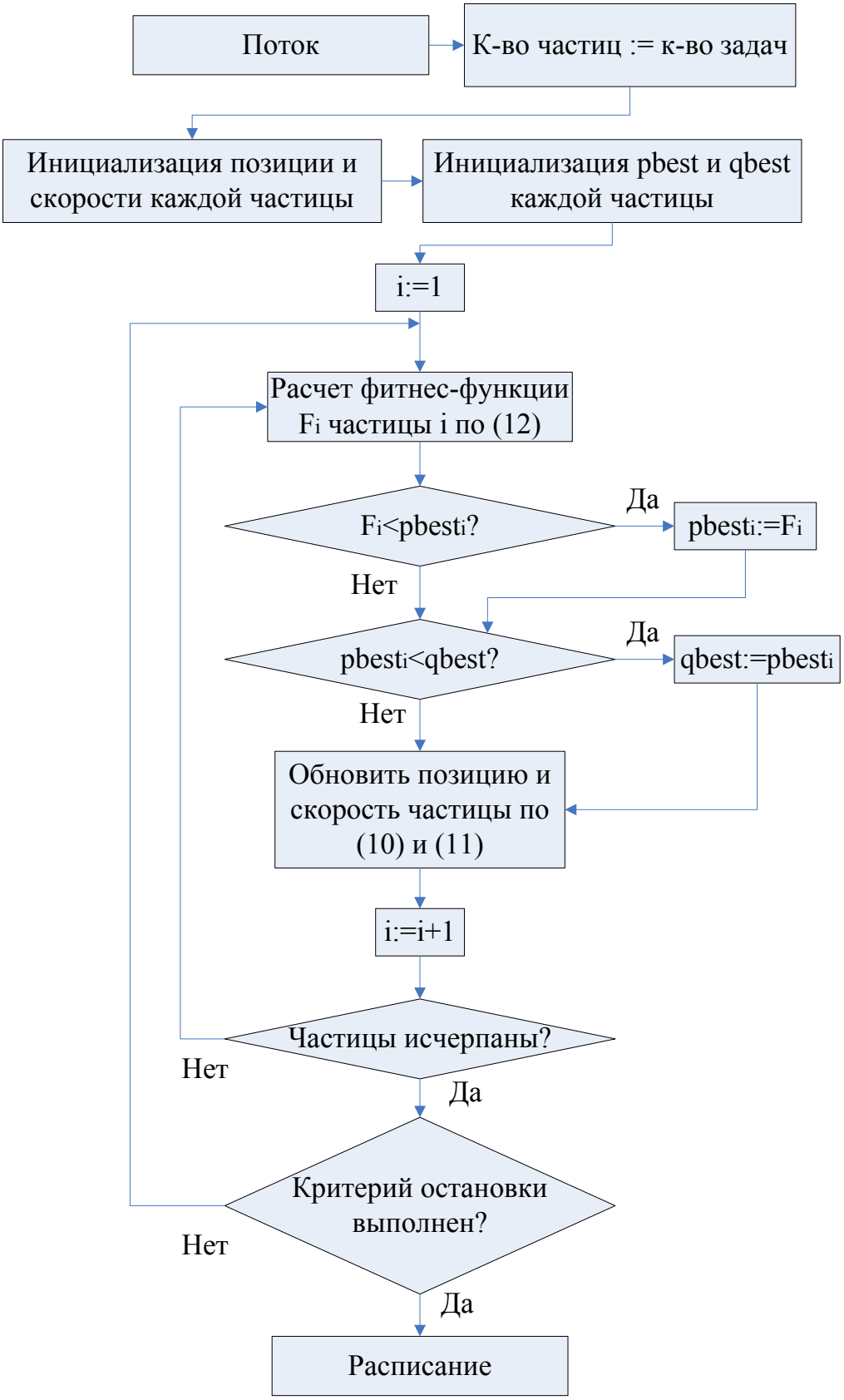


Рис. 4.2. Структурная схема алгоритма RDPSO

Протестирован предложенный подход с помощью моделирования, но чтобы большей адекватности использованы конфигурации виртуальных машин, предлагаемые поставщиком общедоступных облаков, и приложения на основе реальных рабочих процессов.

4.3.1. Настройки инфраструктуры IaaS

Смоделировано облако IaaS, предлагающее несколько гетерогенных центров обработки данных (3 или 5), каждый из которых имеет три или четыре различных типа виртуальных машин. Виртуальные машины основаны на стандарте Amazon Elastic Computing Cloud EC2 instance (http://aws.amazon.com/ec2), их параметры приведены в табл. 4.1.

Таблица 4.1

Тип виртуальной машины, используемых в экспериментах

Тип ВМ	Имя	EC2 vCPU	Clock speed (GHz)
1	m3.medium	1	2.5
2	m3.large	2	2.5
3	м3.xlarge	4	2.5
4	м3.2xlarge	8	2.5

В каждом центре обработки данных предусмотрено три или пять виртуальных машин. Используем обозначение (i)dcx(j)vm для указания конфигурации платформы, используемой для каждого теста, где (i) указывает количество центров обработки данных и (j) указывает максимальное количество виртуальных машин в каждом тесте. Например, обозначение 3dcx5vm указывает, что платформа имеет три центра обработки данных, каждый из которых имеет пять виртуальных машин, и таким образом общее количество виртуальных машин составляет 15. Итак, есть четыре платформы. Неоднородность центров обработки данных учитывается при использовании различных номеров и типов виртуальных машин. На платформах с пятью центрами обработки данных использованы все типы виртуальных машин,

включая тип 4, который является наиболее эффективным.

На платформах с тремя центрами обработки данных используем только первые три типа, перечисленные в табл. 4.2.

Для конфигурации сети использованы гетерогенные средства передачи данных между виртуальными машинами со скоростью от 500 Мбит/с до 1 Гбит/с, а между центрами обработки данных - со скоростью от 50 Мбит/с до 100 Мбит/с.

Таблица 4.2

Центры обработки данных, общее количество и типы виртуальных машин для каждой платформы

Платформа	Центры обработки данных	К-во ВМ	Тип ВМ
3dcx3vm	dc1, dc2, dc3	9	1, 2, 3
3dcx5vm	dc1, dc2, dc3	15	1, 2, 3
5dcx3vm	dc1, dc2, dc3, dc4, dc5	15	1, 2, 3, 4
5dcx5vm	dc1, dc2, dc3, dc4, dc5	25	1, 2, 3, 4

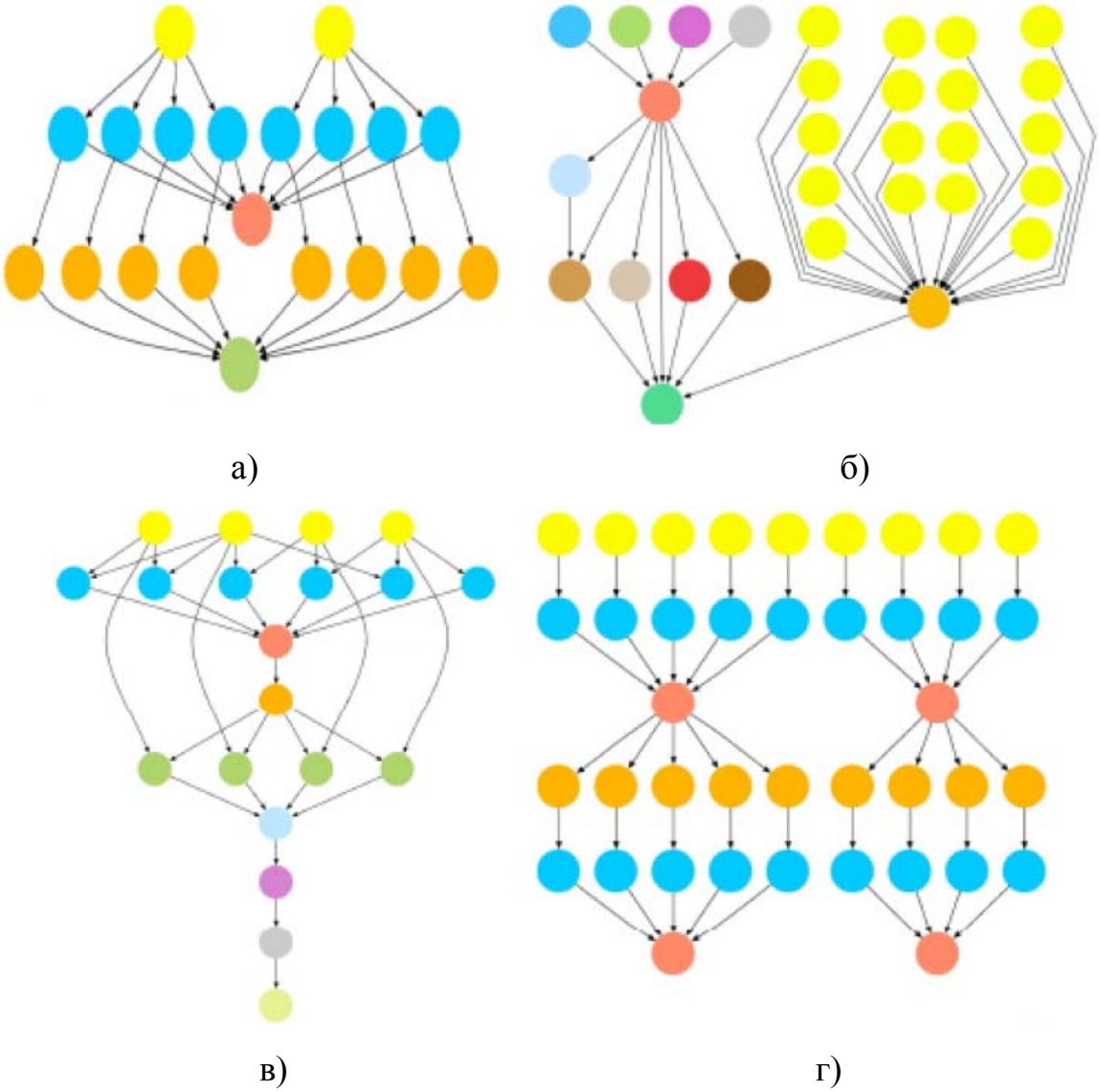
Параметры вероятностей отказов для виртуальных машин и для всех сетевых соединений задаются в пределах 10^{-3}-10^{-4}/ч [4.8, 4.29].

4.3.2. Приложения рабочего процесса

Применены четыре различных реальных процесса. Выбраны два процесса с интенсивной обработкой данных (Montage и CyberShake) и два интенсивных вычислительных процесса (Inspiral и SIPHT). Эти приложения опубликованы проектом Pegasus и описаны в [4.14].

Рабочий процесс CyberShake в основном является интенсивным приложением для обработки данных. Это приложение сейсмологии, используемое для описания землетрясений путем создания синтетических сейсмограмм. Montage - это еще один трудоемкий рабочий процесс, используемый в области астрономии, генерирующий пользовательские мозаики неба с использованием набора входных изображений. SIPHT, характери-

зующийся интенсивным использованием процессора, используется в биоинформатике для управления процессом поиска генов, кодирующих РНК. Наконец, Inspiral - это приложение в области физики для обнаружения гравитационных волн и считается ресурсоемким. Подробные описания рабочих процессов доступны в формате DAX (https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator) и включают в себя DAG, объемы передачи данных и эталонное время выполнения на основе восьмиядерных процессоров Xeon 2.33 ГГц. На рис. 4.3 показаны основные структуры четырех рабочих процессов.



а) б)

в) г)

Рис. 4.3. Процессы: а) CyberShake; б) SIPHT; в) Montage; г) Inspiral

Для оценки производительности RDPSO использованы четыре размера экземпляров рабочих процессов: малый (25 или 30 задач), средний (50 или 60 задач), большой (100 задач) и дополнительный большой (1000 задач).

4.3.3. Настройки алгоритмов

Чтобы настроить алгоритм RDPSO, определим его параметры следующим образом: число испытаний 10, инерционный вес ω=0.9, коэффициенты ускорения c_1=c_2=2 и параметры фитнес-функции α_1=α_2=0.5. Во время выполнения использовано 50 частиц в 100-500 итерациях. Эти значения приняты в аналогичных работах [4.2, 4.13, 4.20].

Для оценки алгоритма использована версия алгоритма HEFT [4.27] в качестве базовой после его улучшения с учетом задач надежности в реализации [4.9]. Затем алгоритм адаптирован к модели (4.3) для поддержки выполнения на нескольких платформах облачных центров обработки данных. Алгоритм назван RHEFT как устойчивый HEFT.

Алгоритм HEFT работает в два этапа:

1. Этап определения приоритетов, на котором рассчитываются приоритеты для всех задач. Затем список задач генерируется путем сортировки задач по их нисходящим приоритетам.

2. Этап выбора ресурсов, на котором задачи назначаются ресурсам, которые минимизируют их EFT (самое раннее время выполнения выполнения). EFT задачи вычисляется как время, в которое заканчивается ее выполнение, то есть время ее начала, добавленное к времени ее выполнения на компьютерном ресурсе.

Реализованный алгоритм RHEFT улучшает результат за счет выбора ресурсов виртуальной машины, что сводит к минимуму время выполнения и повышает надежность. Время связи вычисляется с использованием уравнения (4.3) для EFT.

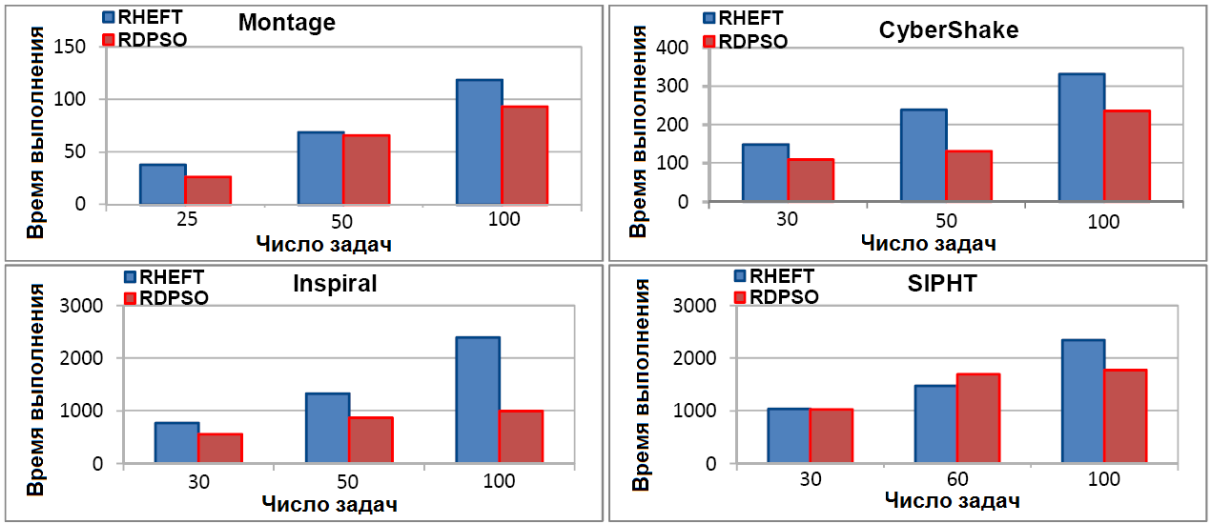
4.3.4. Результаты и обсуждения

Решение оценено в различных конфигурациях платформ (3dcx3vm, 3dcx5vm, 5dcx3vm и 5dcx5vm) с использованием четырех хорошо известных приложений. Время выполнения, переданные данные и общая надежность были сопоставлены с результатами, полученными алгоритмом RHEFT. Поскольку время передачи данных зависит от объема переданных данных, результаты представлены в соответствии с объемом.

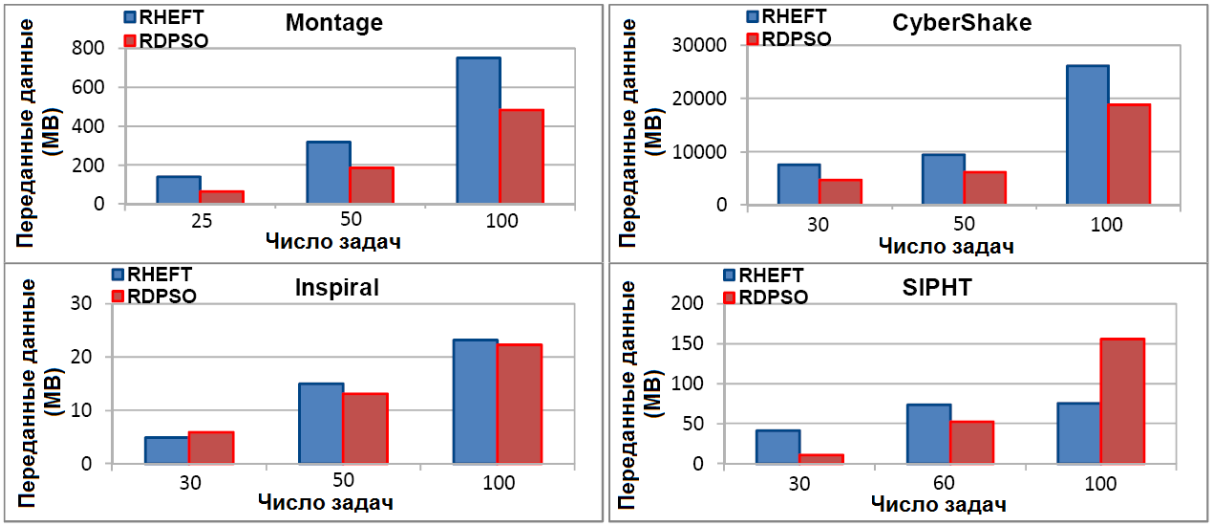
Влияние размера рабочего процесса

Во-первых, эксперименты направлены на изучение эффективности предложенного подхода при планировании рабочих процессов с возрастающей сложностью. Для этого варьировался размер рабочих процессов в ряде задач. Представлены в основном результаты, полученные для платформы 3dcx5vm, но они качественно близки на всех используемых платформах. Экспериментальные результаты, представленные на рис. 4.4 а-в, представляют собой время выполнения, количество переданных данных и общую надежность соответственно для разного количества задач. Использовано четыре приложения рабочего процесса с 30 (25 для Montage), 50 (60 для SIPHT) и 100 задачами.

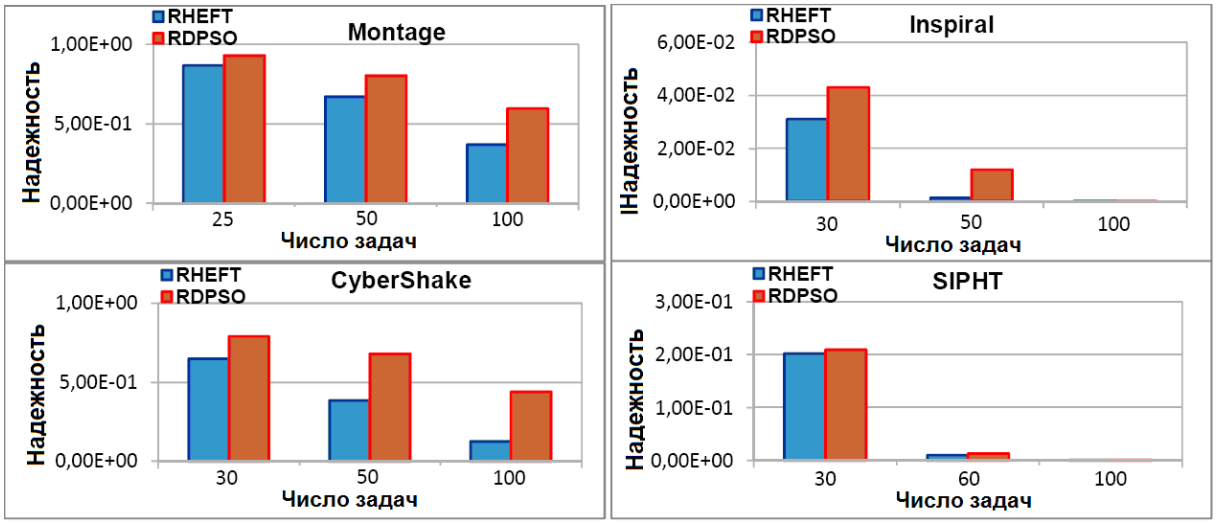
Как для алгоритмов RHEFT, так и для алгоритмов RDPSO, по мере увеличения числа задач каждого рабочего процесса, время выполнения увеличивается, но явно улучшается RDPSO (рис. 4.4а). Время выполнения, порожденное RDPSO снизилось примерно на 28% для Montage (25 задач), и это на 30% лучше, чем для CyberShake, который имеет 30 задач по сравнению с теми, которые были получены с помощью RHEFT. Улучшения во времени выполнения для RDPSO выражены для всех рабочих процессов, за исключением SIPHT (60 задач), для которых RHEFT превосходит RDPSO. Параллельная структура этих рабочих процессов определяет, что несколько задач имеют одинаковое значение ранга, но упорядочены в фиксированном списке с весом.



а) Время выполнения



б) Переданные данные



в) Надежность

Рис. 4.4. а) время выполнения, б) объем переданных данных, в) надежность платформы 3dcx5mv для Montage, CyberShake, Inspiral и SIPHT, с 30, 50 и 100 задачами

Значения, полученные для передаваемых данных (рис. 4б) с помощью рабочих процессов Montage и CyberShake, увеличиваются вместе с числом задач, поскольку эти два рабочих процесса являются интенсивными по данным. Эти значения снижаются при RDPSO по сравнению с теми, которые регистрируются при RHEFT.

Для рабочего процесса Inspiral результаты, полученные с помощью RDPSO, близки к тем, которые представлены RHEFT. Объемы передаваемых данных с использованием предложенного подхода уменьшились по сравнению с RHEFT для приложения SIPHT, имеющего 30 или 60 задач, но он не уменьшились для рабочего процесса SIPHT, имеющего 100 задач.

Что касается надежности (рис. 4в), то значения уменьшаются по мере увеличения числа задач для двух алгоритмов. Точно так же RDPSO зафиксировал лучшие значения по сравнению с RHEFT для рабочих процессов Montage, CyberShake и Inspiral.

В табл. 4.3 приведены сводные результаты, полученные для рабочих процессов, в которых задействовано 100 задач.

При использовании сверхбольших рабочих процессов полученные результаты RDPSO не улучшает во всех случаях из-за расширения пространства метаэвристического поиска из-за большой размерности частиц (табл. 4.4). Что касается времени выполнения, то значения, для приложений Montage и CyberShake с RDPSO лучше, но остаются близкими к значениям, найденным RHEFT. Время выполнения улучшается только для рабочего процесса Inspiral. Передаваемые данные в большинстве случаев уменьшаются, но общая надежность не улучшается.

Таблица 4.3

Сравнение результатов для рабочих процессов, имеющих 100 задач на платформе 3dcx5vm

рабочий процесс	время выполнения		переданные данные		надежность	
	RHEFT	RDPSO	RHEFT	RDPSO	RHEFT	RDPSO
Montage	117.22463	78.72292	750.45070	483.54444	$3.71 \cdot 10^{-1}$	$5.97 \cdot 10^{-1}$
CyberShake	331.8782	235.79208	2,857	2,393	$1.27 \cdot 10^{-1}$	$4.39 \cdot 10^{-1}$
Inspiral	2,384	999.06190	19.38876	18.49763	$8.37 \cdot 10^{-8}$	$2.73 \cdot 10^{-4}$
SIPHT	2,336	1,857	75.57317	156.14683	$1.23 \cdot 10^{-3}$	$1.41 \cdot 10^{-3}$

Таблица 4.4

Сравнение результатов для рабочих процессов с 1000 задачами на платформе 5dcx3vm

рабочий процесс	время выполнения		переданные данные		надежность	
	RHEFT	RDPSO	RHEFT	RDPSO	RHEFT	RDPSO
Montage	515.4528	667.5122	11.066	11.205	$4.12 \cdot 10^{-3}$	$1.51 \cdot 10^{-4}$
CyberShake	1,113	1,164	167.035	159.361	$1.58 \cdot 10^{-10}$	$4.09 \cdot 10^{-12}$
Inspiral	13.378	8.134	309.1684	303.66625	$3.77 \cdot 10^{-23}$	$1.76 \cdot 10^{-64}$
SIPHT	8.475	16.174	1.771	1.533	$7.57 \cdot 10^{-21}$	$8.77 \cdot 10^{-51}$

Влияние характеристик платформы

В данном подразделе показано влияние ресурсных характеристик и их количества на исследуемые критерии (время выполнения, передаваемые данные и надежность). Рабочие процессы со 100 задачами (Montage-100, CyberShake-100, Inspiral-100 и SIPHT-100) выбраны для изучения влияния количества виртуальных машин и центров обработки данных на время выполнения, объем передаваемых данных и надежность при обоих подходах. На рис. 4.5а-в представлены результаты, полученные на различных платформах. Для платформ с тремя центрами обработки данных (3dcx3mv и 3dcx5mv) и с пятью центрами обработки данных (5dcx3mv и 5dcx5mv), время выполнения значительно уменьшается с увеличением количества виртуальных машин (рис. 4.5а). Использование большего количества виртуальных машин позволяет использовать преимущества параллелизма, поскольку все используемые рабочие процессы имеют частично параллель-

ную структуру. Результаты измерений на платформах с тремя центрами обработки данных были меньше, чем на платформах с пятью центрами обработки данных около 40% для CyberShake, более чем на 45% для Inspiral и около 50% на Montage и SIPHT. Результаты, полученные с помощью четырех процессов, подтвердили, что RDPSO имеет наименьшее время выполнения, чем RHEFT в большинстве случаев. Например, улучшение, полученное RDPSO по сравнению с RHEFT, составляет около 25% для Montage, 28% для CyberShake и от 25% до 50% для Inspiral на платформах, состоящих из трех центров обработки данных.

Результаты остаются лучше с RDPSO для приложений Montage и CyberShake. Улучшение достигает 35% для Montage на 3dc5vm и 33% для CyberShake на 5dcx5vm.

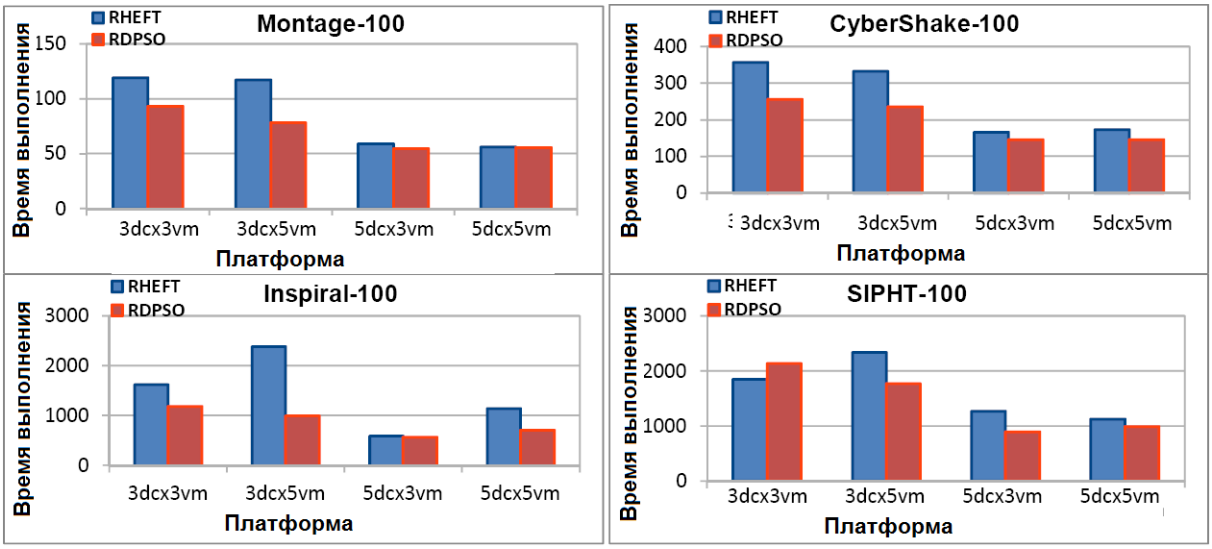
Общая надежность рабочих процессов со 100 задачами (рис. 4.5в) существенно не колеблется в зависимости от различных платформ для рабочих процессов Montage и CyberShake, в отличие от рабочих процессов Inspiral и SIPHT.

Это связано с тем, что эти последние имеют значительную вычислительную интенсивность и величина надежности пропорциональна вычислительному времени. Для всех рабочих процессов надежность RDPSO повысилась по сравнению с RHEFT в основном для приложений Montage и CyberShake.

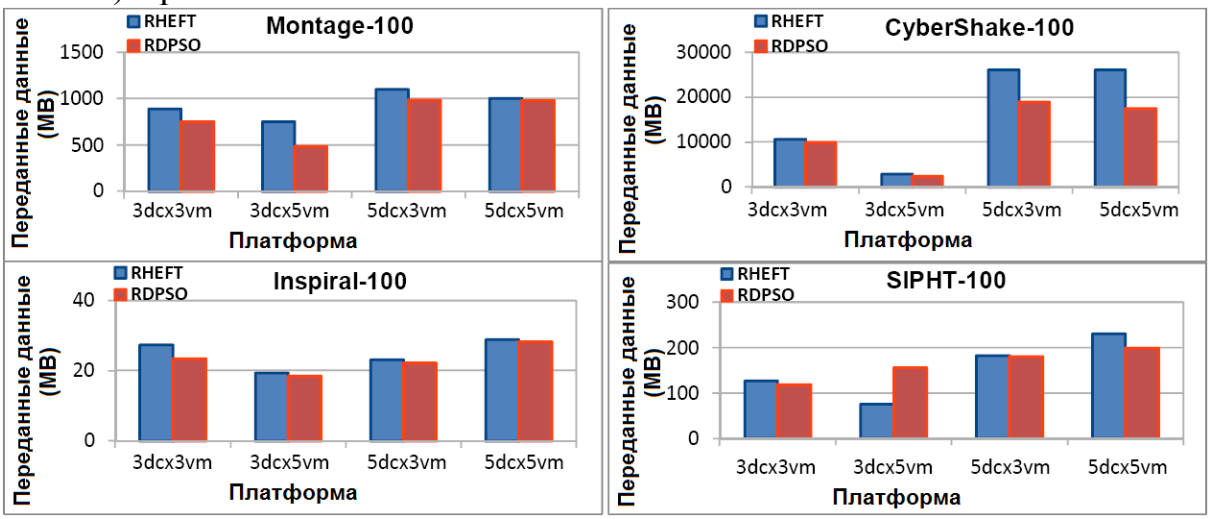
Влияние передачи данных

В этом разделе проанализирована передача данных между различными центрами обработки данных в рамках одной платформы, чтобы показать, как RDPSO оптимизирует передаваемые данные по сравнению с RHEFT.

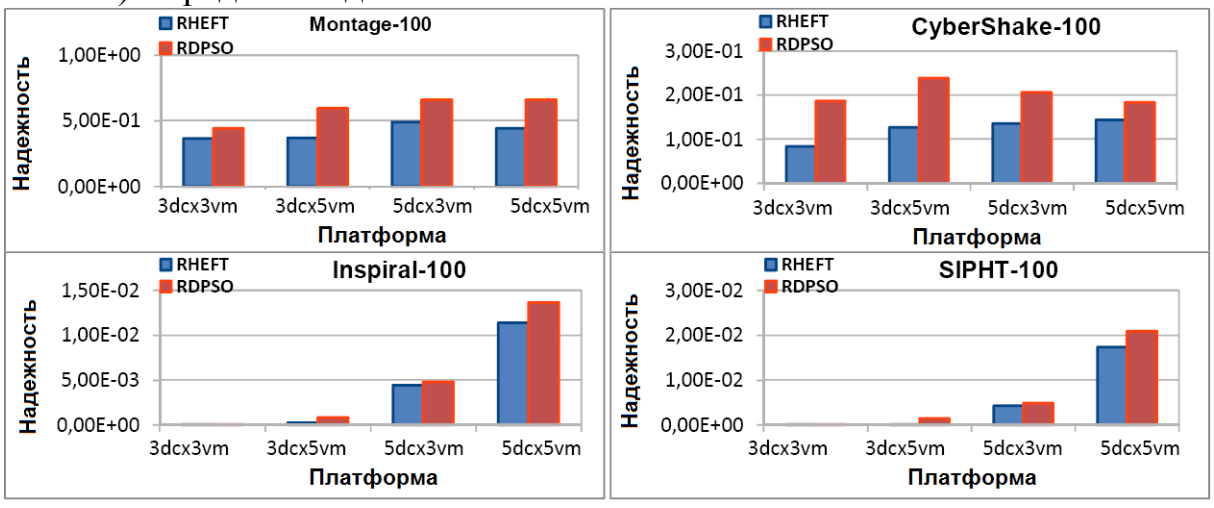
Дано подробное описание моделей передачи данных, выполняемых во время планирования рабочих процессов, имеющих 100 задач и 1000 задач на платформе 5dcx3vm.



а) Время выполнения



б) Переданные данные



в) Надежность

Рис. 4.5. а) время выполнения, б) передача данных, в) надежность на различных платформах для рабочих процессов Montage-100, CyberShake-100, Inspiral-100 и SIPHT-100

На рис. 4.6 показаны графики передачи данных для рабочих процессов, имеющих 100 задач, между каждой парой центров обработки данных на платформе 5dcx3vm. Обозначение (dci↔dcj) означает передачу данных между центром обработки данных i и центром обработки данных j. Установлено, RDPSO избегает больших пиков передачи данных, которые есть у RHEFT, выполняемой между dc1 и dc4 (dc1↔dc4), а также между dc2 и dc4 (dc2↔dc4) для рабочих процессов Montage и CyberShake. В случае рабочих процессов Inspiral и SIPHT передача данных, достигаемая RDPSO, ближе к передаче данных, выполняемой RHEFT.

Наконец, результаты подтверждаются в отношении передачи данных для сверхбольших рабочих процессов с использованием приложений Montage-1000, CyberShake-1000, Inspiral-1000 и SIPHT-1000 на платформе 5dcx3vm (рис. 4.7).

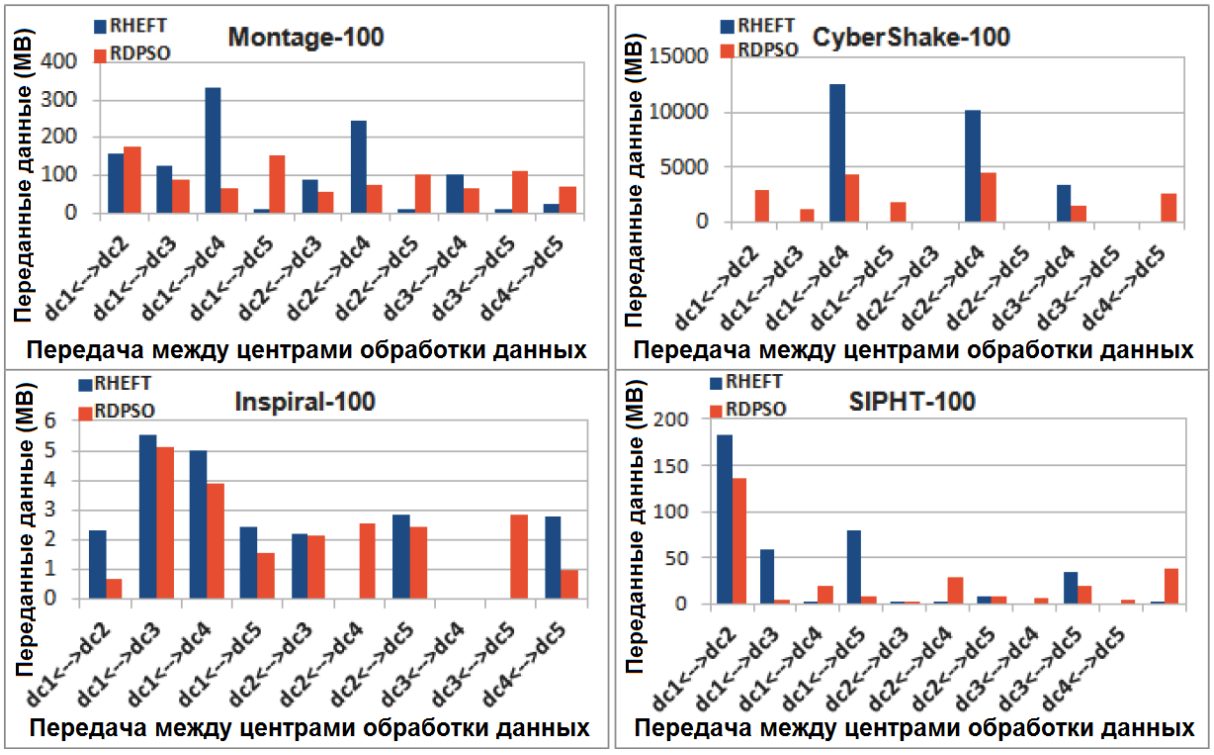


Рис. 4.6. Передача данных в 5dcx3vm для рабочих процессов (100 задач)

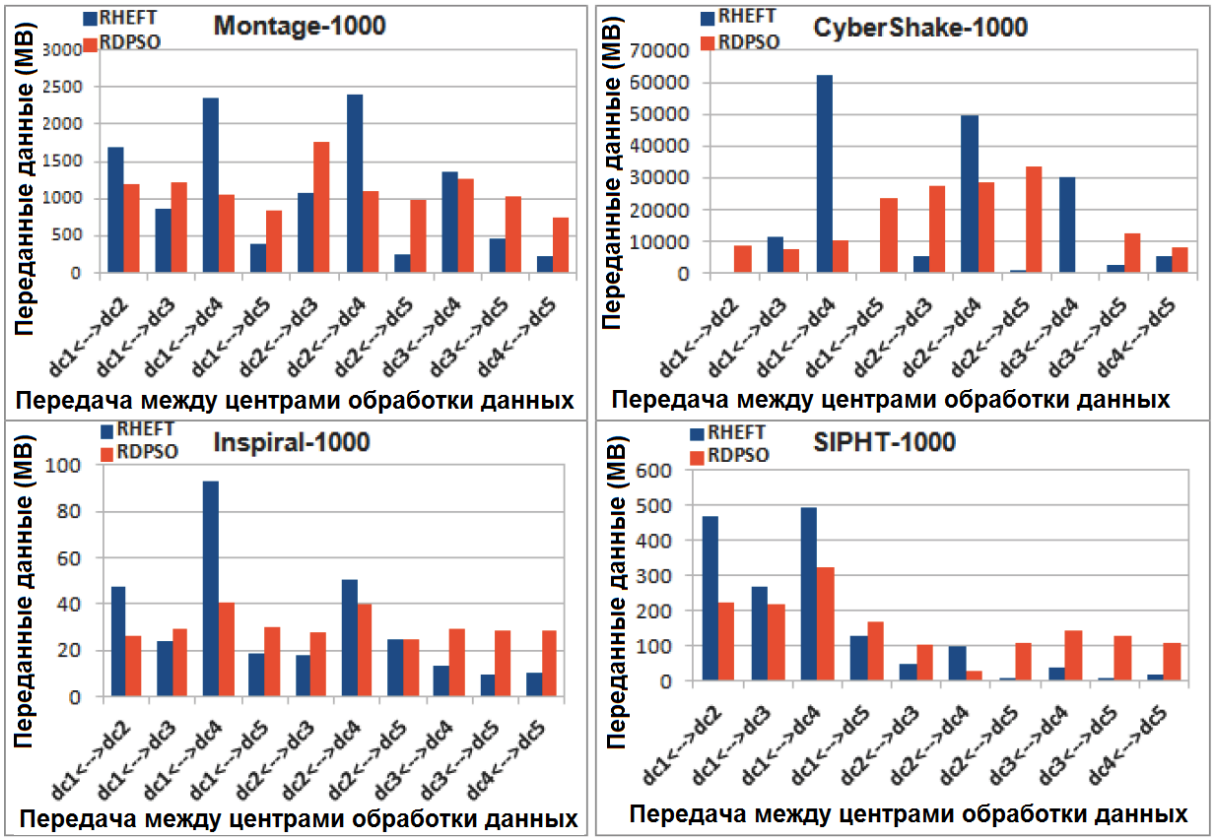


Рис. 4.7. Передача данных в 5dcx3vm для рабочих процессов (1000 задач)

Результаты для приложений Montage и CyberShake снова показывают, что RDPSO избегает больших пиков передач, которые есть у RHEFT, выполняемых между dc1 и dc4 (dc1↔dc4), и между с dc2 и dc4 (dc2↔dc4). Предложенный подход позволяет избежать значительных перемещений между dc1 и dc4 для Inspiral, а также между dc1 и dc2 и между dc1 и dc4 для SIPHT.

4.4. Архитектура прототипа системы управления распределением заданий в облачных средах

Архитектура прототипа системы управления распределением заданий в облачных средах изображена на рис. 4.8.

Рис. 4.8. Архитектура системы управления распределением заданий в облачных средах

4.4.1. Функциональная схема

Структурная схема программной системы, как правило, показывает наличие подсистем или других структурных компонентов. В отличие от программного комплекса отдельные части (подсистемы) программной системы интенсивно обмениваются данными между собой и. возможно, с основной программой. Структурная же схема программной системы этого обычно не показывает. Более полное представление о проектируемом программном обеспечении с точки зрения взаимодействия его компонентов между собой и с внешней средой дает функциональная схема.

Функциональная схема или схема данных– документ, разъясняющий процессы, протекающие в отдельных функциональных цепях изделия (ус-

тановки) или изделия в целом. Относительно программного обеспечения функциональная схема – это схема взаимодействия компонентов программного обеспечения с описанием информационных потоков, состава данных в потоках и указанием используемых файлов и устройств. Для изображения функциональных схем используют специальные обозначения, установленные ГОСТ 19.701-90. Функциональная схема изображена на рис. 4.9.

4.4.2. Описание работы

При запуске необходимо произвести вход в облачную систему, используя связку логин-пароль. Базовым является окно мониторинга, на котором отображается общая статистика: загруженность системы с графиком за последний период (определяется пользователем), предупреждения и ошибки. Меню содержит несколько пунктов: «Конфигурация системы», «Диагностика», «Ручное управление», «Помощь».

Пункт «Конфигурация системы» содержит подпрограммы «Управление компонентами» и «Настройка правил». При выборе первой подпрограммы выводится список компонентов с кратким описанием их состояния (ОК, обратить внимание, критические ошибки, остановлено). Пользователь может добавить компонент при расширении инфраструктуры либо управлять уже имеющимися путём остановки, запуска либо удаления. При добавлении пользователь вводит информацию о компоненте, например, вместимость дискового хранилища, количество ядер процессоров, количество ОЗУ, затем определяет условия добавления: дату и время, географический кластер, кластер в ТОД.

При выборе второй подпрограммы пользователь может настроить правила диспетчеризации задач, учитывая местные характеристики, например, скорость Интернет-подключения либо конкретную группу задач.

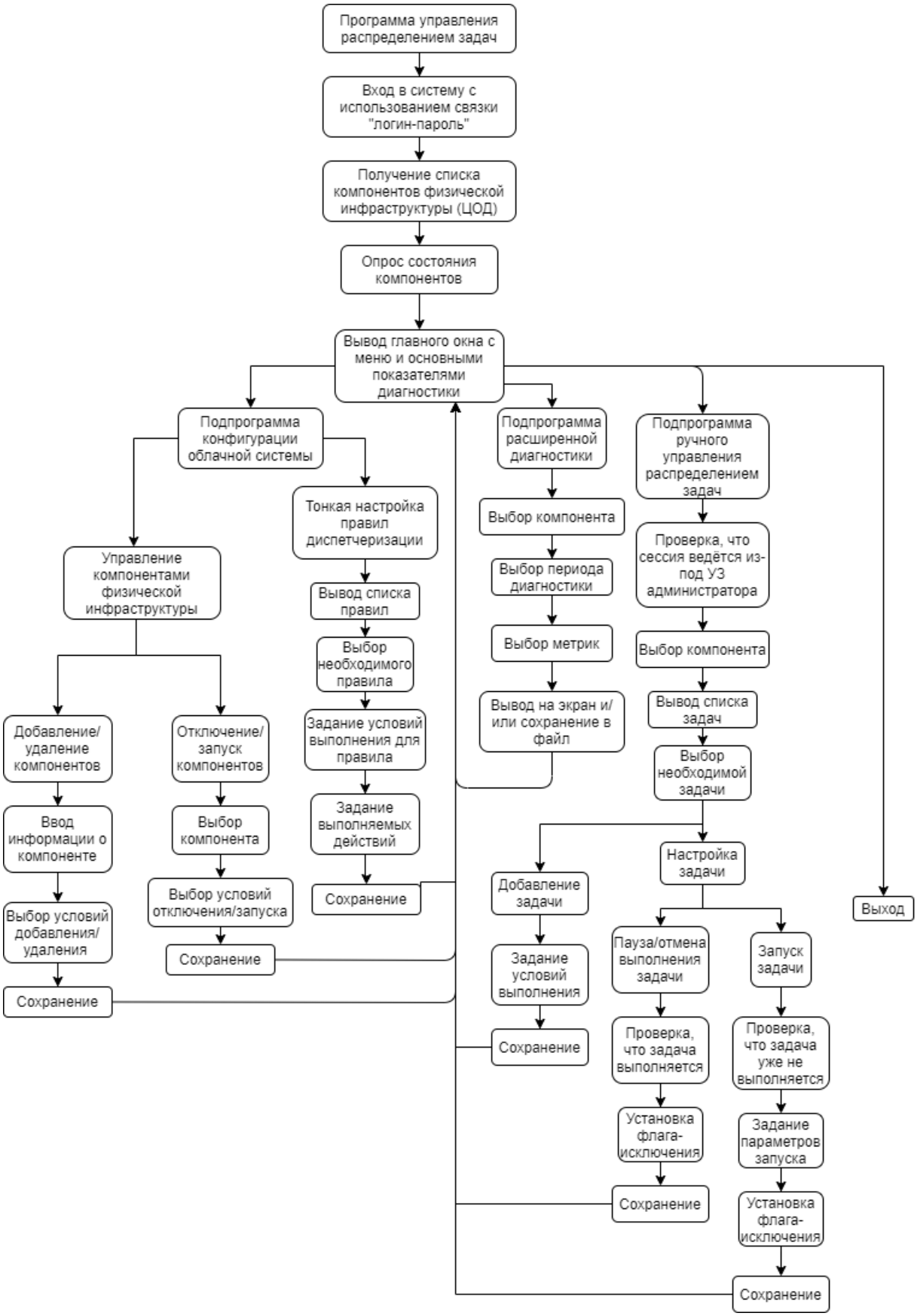


Рис. 4.9. Функциональная схема

Изменения вносятся в список уже существующих правил, сформированных системой. Пользователь выбирает необходимое и настраивает его, добавляя условия выполнения и действия, выполняемые по достижении этих условий. В конце каждой подпрограммы пользователь должен сохранить внесённые изменения, после чего происходит возвращение в главное окно программы.

Пункт «Диагностика» содержит в себе подпрограмму расширенной диагностики состояния системы. Пользователь выбирает компонент, определяет период, за который хотел бы просмотреть изменение метрик, после чего выбирает необходимые метрики, получая в итоге график, отражающий зависимость метрики от времени.

Подпрограмма ручного управления диспетчеризацией задач требует выполнения исключительно из-под учётной записи администратора. Он выбирает компонент, на котором выполняется или будет выполняться задача, программа выводит список всех задач, а администратор выбирает необходимую либо создаёт новую. Стоит отметить, что даже администратор может управлять не всеми задачами, а их обновление происходит не в режиме реального времени, так как многие из них выполняются чрезвычайно быстро. В основном ручное управление применяется для запланированных задач либо для внепланового добавления, например, для тестирования. Администратор может приостановить либо отменить выполнение задачи, либо запустить её. В любом случае происходит проверка на выполнение задачи: в зависимости от нужного действия система позволяет либо не позволяет внести изменения. В случае успешного прохождения проверки на данную задачу устанавливается флаг-исключение, и система впредь не будет оптимизировать данную конкретную задачу автоматически. Администратору остаётся лишь сохранить изменения. В случае ошибки при прохождении проверки система выводит сообщение с возможными причинами этой ошибки.

4.4.3. Примеры экранных форм

На рис. 4.10-4.14 представлены примеры экранных форм прототипа системы управления.

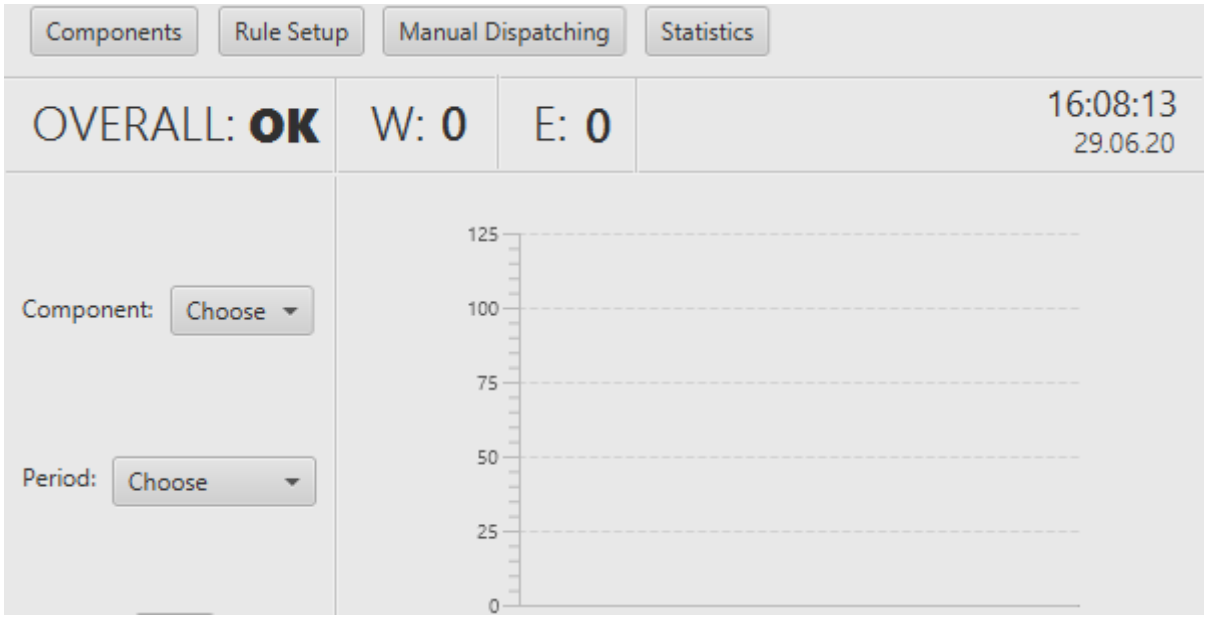


Рис. 4.10. Главное окно

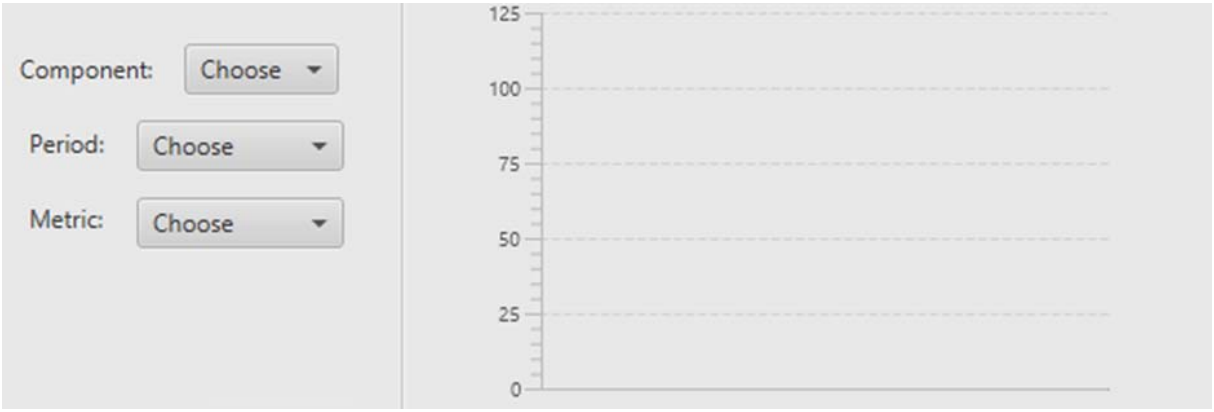


Рис. 4.11. Расширенная диагностика

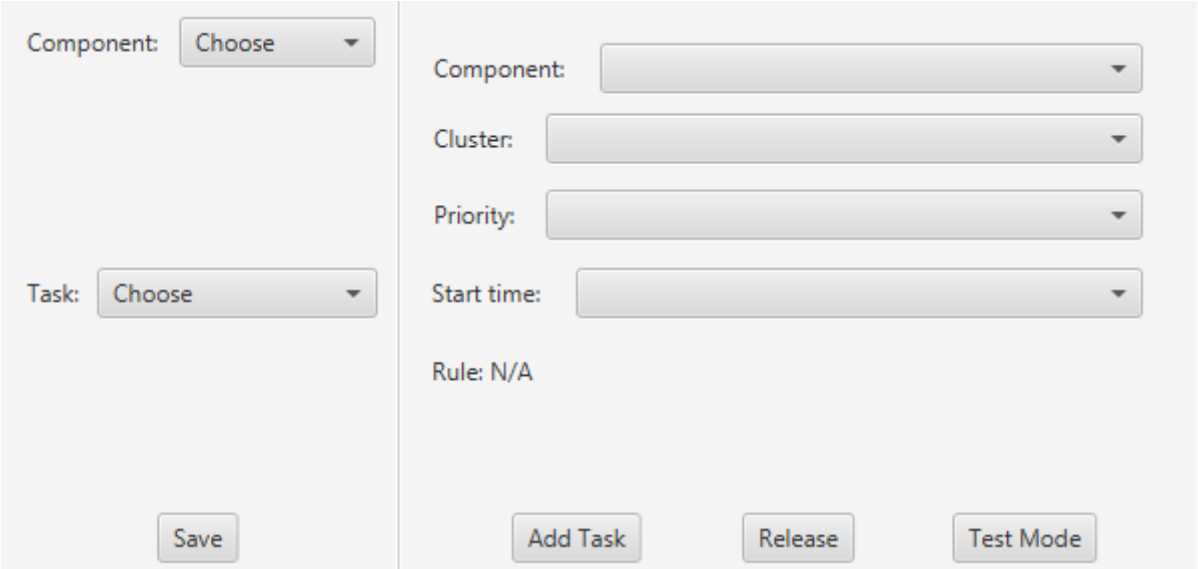


Рис. 4.12. Ручное управление задачами

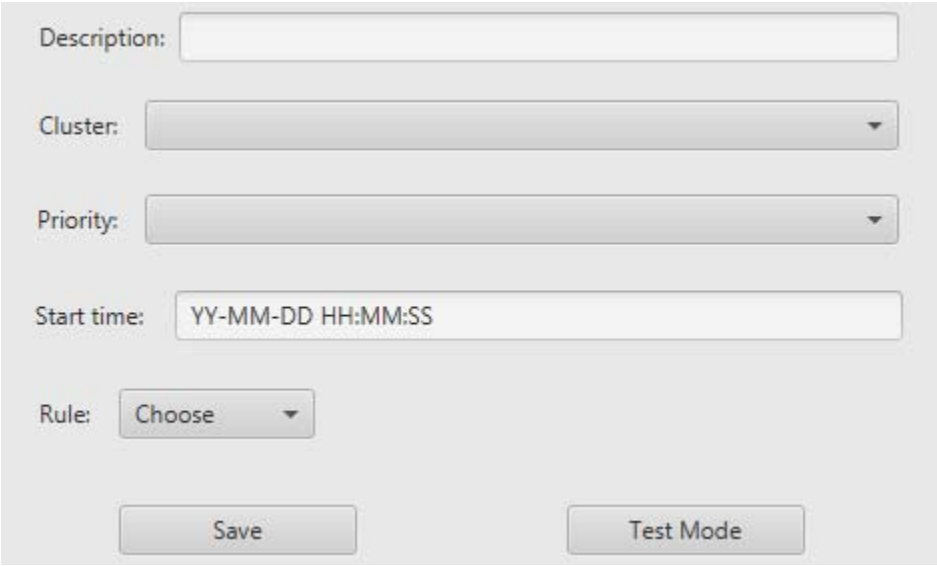


Рис. 4.13. Добавление компонента

4.4.4. Примеры использования

Примером использования может выступить оптимизация нагрузки на облачную систему при прогнозируемом её росте (например, во время выходных, праздничных дней). Также стоит отметить возможность оптимизации при добавлении новых вычислительных мощностей: администратор заносит в таком случае в имеющуюся базу данных параметры нового оборудования, а система автоматически рассчитывает количество создаваемых виртуальных машин и степень их использования).

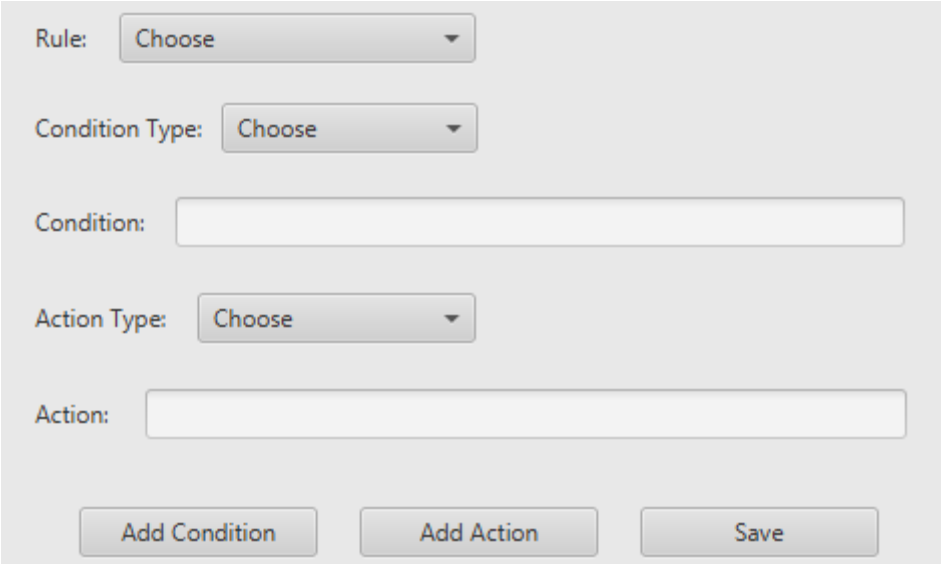


Рис. 4.14. Настройка правил диспетчеризации

Немаловажной является и возможность временного отключения части оборудования для её обновления как по части программного обеспечения, так и по части компонентов. Компонент «Диагностика» позволяет внепланово проверить систему на наличие проблем. Ручное управление позволяет протестировать выполнение конкретных задач в необходимых условиях.

4.5. Выводы

Время выполнения по-прежнему остается наиболее важной метрикой для оптимизации планирования приложений. В частности, на нескольких платформах облачных центров обработки данных этот показатель ухудшается из-за времени передачи данных. Кроме того, большинство исследований предполагают, что каналы связи однородны с равномерной полосой пропускания во время выполнения задачи или то, что сбои сетевых связей игнорируются. Представлен подход к планированию RDPSO, основанный на PSO. Ставилась задача минимизировать время выполнения и время передачи данных, одновременно повышая надежность рабочего процесса в нескольких облачных центрах обработки данных. Учитывалась неодно-

родность ресурсов, сбои виртуальных машин и сбои связей.

Разработан модифицированный алгоритм роя частиц настройки параметров расписания облачных вычислений, отличающийся использованием весовой фитнес-функции и обеспечивающий для минимизацию времени передачи данных, времени выполнения и повышение надежности выполнения рабочих процессов.

Проведена оценка алгоритма с использованием реальных рабочих процессов и представлены результаты, полученные разработанной эвристикой в противовес RHEFT. Результаты показывают, что RDPSO превосходит алгоритм PSO. Время выполнения улучшается у RDPSO в большинстве случаев и общая надежность повышается. Предложенный подход реализовал передачу с меньшим объемом данных, чем алгоритм RHEFT, даже с использованием очень больших рабочих процессов.

Список источников к главе 4

4.1. Babu L.D., Krishna P.V. Versatile time-cost algorithm (VTCA) for scheduling non-preemptive tasks of time critical workflows in cloud computing systems// International Journal of Communication Networks and Distributed Systems, 2013. Vol. 11, No. 4, pp.390–411.

4.2. Bouali L., Oukfif K., Bouzefrane S., Oulebsir-Boumghar F. A hybrid algorithm for DAG application scheduling on computational grids// International Conference on Mobile, Secure and Programmable Networking, 2015, pp.63–77, Springer.

4.3. Calheiros R.N., Ranjan R., Beloglazov A., De Rose C.A., Buyya R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms// Software: Practice and Experience, 2011. Vol. 41, No. 1, pp.23–50.

4.4. Chen J., Zhang J., Song A. Efficient data and task co-scheduling for scientific workflow in geo-distributed datacenters// 2017 Fifth International Conference on Advanced Cloud and Big Data (CBD), pp.63–68, IEEE.

4.5. Chen W-N., Zhang J. An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements// IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 2009. Vol. 39, No. 1, pp.29–43.

4.6. Chen W-N., Zhang J. A set-based discrete PSO for cloud workflow scheduling with user-defined QoS constraints// 2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC). 2012, pp.773–778.

4.7. Choudhary A., Gupta I., Singh V., Jana P.K. A GSA based hybrid algorithm for bi-objective workflow scheduling in cloud computing// Future Generation Computer Systems, 2018. Vol. 83, No. C, pp.14–26.

4.8. Dogan A., Ozguner F. Biobjective scheduling algorithms for execution time-reliability trade-off in heterogeneous computing systems// The Computer Journal, Vol. 48, No. 3, pp.300–314.

4.9. Dongarra J.J., Jeannot E., Saule E., Shi Z. Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems// Proceedings of the Nineteenth Annual ACM Symposium on Parallel Algorithms and Architectures, 2007, pp.280–288, ACM.

4.10. Fard H.M., Prodan R., Barrionuevo J.J.D., Fahringer T. A multi-objective approach for workflow scheduling in heterogeneous environments// Proc. of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2012), pp.300–309, IEEE Computer Society.

4.11. Guo T., Liu J., Hu W., Wei M. Energy-aware fault-tolerant scheduling under reliability and time constraints in heterogeneous systems// International Conference on Intelligent Computing, 2018, pp.36–46, Springer.

4.12. Hakem M., Butelle F. Reliability and scheduling on systems subject to failures// International Conference on Parallel Processing, 2007. ICPP 2007, pp.38–48.

4.13. Jian C., Tao M., Wang Y. A particle swarm optimisation algorithm for cloud-oriented workflow scheduling based on reliability// International Journal of Computer Applications in Technology, 2014. Vol. 50, Nos. 3–4, pp.220–225.

4.14. Juve G., Chervenak A., Deelman E., Bharathi S., Mehta G., Vahi K. Characterizing and profiling scientific workflows// Future Generation Computer Systems, 2013. Vol. 29, No. 3, pp.682–692.

4.15. Kennedy J. Particle swarm optimization// Encyclopedia of Machine Learning. - Springer, New York, NY, USA, 2011, pp.760–766.

4.16. Kianpisheh S., Charkari N.M., Kargahi M. Reliability-driven scheduling of time/cost-constrained grid workflows// Future Generation Computer Systems, 2016. Vol. 55, No. C, pp.1–16.

4.17. Kumar H., Chauhan N.K., Yadav P.K. A task allocation model for minimizing system cost and maximizing reliability of distributed computing system// International Journal of Communication Networks and Distributed Systems, 2018. Vol. 20, No. 2, pp.226–243.

4.18. Lee Y.C., Zomaya A.Y., Yousif M. Reliable workflow execution in distributed systems for cost efficiency// 2010 11th IEEE/ACM International Conference on Grid Computing (GRID), pp.89–96, IEEE.

4.19. Mell P., Grance T. The NIST Definition of Cloud Computing, National Institute of Standards and Technology. http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf. 2011.

4.20. Oukfif K., Bouali L., Bouzefrane S., Oulebsir-Boumghar F. (Energy-aware dpso algorithm for workflow scheduling on computational grids// 2015 3rd International Conference on Future Internet of Things and Cloud (FiCloud), pp.651–656, IEEE.

4.21. Poola D., Ramamohanarao K., Buyya R. Enhancing reliability of workflow execution using task replication and spot instances// ACM Transactions on Autonomous and Adaptive Systems (TAAS), 2016. Vol. 10, No. 4, p.30.

4.22. Qin X., Jiang H. A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems// Parallel Computing, 2006. Vol. 32, No. 5, pp.331–356.

4.23. Rehani N., Garg R. Meta-heuristic based reliable and green workflow scheduling in cloud computing// International Journal of System Assurance Engineering and Management, 2017. Vol. 9, No. 4, pp.1–10.

4.24. Sih G.C., Lee E.A. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures// IEEE Transactions on Parallel and Distributed systems, 1993, Vol. 4, No. 2, pp.175–187.

4.25. Sinha A., Malo P., Deb K. A review on bilevel optimization: from classical to evolutionary approaches and applications// IEEE Transactions on Evolutionary Computation, 2018. Vol. 22, No. 2, pp.276–295.

4.26. Tang X., Li K., Qiu M., Sha E.H-M. A hierarchical reliability-driven scheduling algorithm in grid systems// Journal of Parallel and Distributed Computing, 2012. Vol. 72, No. 4, pp.525–535.

4.27. Topcuoglu H., Hariri S., Wu M-Y. Performance-effective and low-complexity task scheduling for heterogeneous computing// IEEE Transactions on Parallel and Distributed Systems, 2002. Vol. 13, No. 3, pp.260–274.

4.28. Ullman J.D. NP-complete scheduling problems// Journal of Computer and System Sciences, 1975. Vol. 10, No. 3, pp.384–393.

4.29. Wang X., Yeo C.S., Buyya R., Su J. Optimizing the makespan and reliability for workflow applications with reputation and a look-ahead genetic algorithm// Future Generation Computer Systems, 2011. Vol. 27, No. 8, pp.1124–1134.

4.30. Wen Z., Cala J., Watson P., Romanovsky A. Cost effective, reliable and secure workflow deployment over federated clouds// IEEE Transactions on Services Computing, 2016. Vol. 10, No. 6, pp.929–941.

4.31. Wu F., Wu Q., Tan Y. Workflow scheduling in cloud: a survey// The Journal of Supercomputing, 2015. Vol. 71, No. 9, pp.3373–3418.

4.32. Xiao X., Xie G., Xu C., Fan C., Li R., Li K. Maximizing reliability of energy constrained parallel applications on heterogeneous distributed systems// Journal of Computational Science, 2018. Vol. 26, No. C, pp.344–353.

4.33. Zhang J., Wang M., Luo J., Dong F., Zhang J. Towards optimized scheduling for data-intensive scientific workflow in multiple datacenter environment// Concurrency and Computation: Practice and Experience, 2015. Vol. 27, No. 18, pp.5606–5622.

4.34. Zhang L., Li K., Li C., Li K. Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems// Information Sciences, 2017. Vol. 379, pp.241–256.

Заключение

Целью работы являлась разработка математического и программного обеспечения управления миграцией виртуальных машин в облачных средах на основе иерархической стратегии балансировки нагрузки.

В процессе выполнения исследования получены следующие основные результаты:

1. Создана модифицированная многоуровневая архитектура системы управления облачными средами с дополнительным миграционным слоем, обеспечивающая сокращение времени задержки, вызванной осуществлением миграции.

2. Разработан эвристический алгоритм планирования задач в облаке с полиномиальной по времени сложностью, обеспечивающий соблюдение регламентных сроков исполнения задач.

3. Создана иерархическая стратегия балансировки нагрузки для облачных многокластерных центров обработки данных, обеспечивающая уменьшение среднего времени отклика и служебных издержек межкластерной коммуникации.

4. Разработан оптимизационный алгоритм решения многокритериальной задачи управления распределением задач в облачных средах на основе алгоритма роя частиц, обеспечивающий оптимальное время выполнения и надежность как компьютерных ресурсов, так и сетевых связей.

5. Создан прототип системы диспетчеризации заданий в облачных средах. Элементы программного обеспечения зарегистрированы в ФИПС.

Список использованных источников

1. Вентцель Е.С., Овчаров Л.А. Теория случайных процессов и ее инженерные приложения. - М.: Наука. Гл. ред. физ.-мат. лит. - 1991.

2. Волков Д. Как оценить рабочую станцию // Открытые системы, №2, 1994, С. 44-48.

3. Волков Д., Французов Д., Новое поколение тестов SPEC. / Открытые системы, №4, 1996, С. 73-74.

4. Гамильтон С. Управление цепочками поставок с Microsoft Axapta. – М.: Альпина Бизнес Букс, 2005, 352 с.

5. Гешвинде Э., Шенинг Г.Ю. Разработка WEB-приложений на PHP и PostgreSQL. М.: ДиаСофт, 2003.- 608 С.

6. Ги К. Введение в локально-вычислительные сети. – М.: Радио и связь, 2000. – 190 с.

7. Гнеденко Б.В., Коваленко И.Н. Введение в теорию массового обслуживания, 2-е изд., М.:Наука, 1987. - 336 С.

8. Говорский А.Э., Кравец О.Я. Особенности взаимодействия подсистем при решении задач интегрального обслуживания неоднородного трафика // Системы управления и информационные технологии. 2008. Т. 31. № 1.1. С. 141-146.

9. ГОСТ 28195-89 Оценка качества программных средств. Общие положения. Введен 01.07.90. - 39 С.

10. ГОСТ 28806-90 Качество программных средств. Термины и определения. Введен 01.01.92. - 12 С.

11. ГОСТ Р ИСО/МЭК 9126-93 Информационная технология. Оценка программной продукции. Характеристики качества и руководства по их применению. Введен 01.07.94. - 19 С.

12. Гриневич А.И., Кулямин В.В., Марковцев Д.А., Петренко А.К., Рубанов В.В., Хорошилов А.В. Использование формальных методов для обеспечения соблюдения программных стандартов. – Тр. института системного программирования РАН. - 2006. - Т.10, с. 51-68.

13. Грязнов Н.Г., Димитриев Ю.К., Мелентьев В.А. Оптимизация отказоустойчивого вложения диагностического графа в тороидальные структуры живучих вычислительных систем// Автоматика и телемеханика. 2003. № 4. С. 133-152.

14. Дастин Э., Рэшка Д., Джон П. Автоматизированное тестирование программного обеспечения. Внедрение, управление и эксплуатация. - М.: Лори, 2003.

15. Дейт К. Введение в системы баз данных - 6-е изд. - Киев: Диалектика, 1998. - 784 С.

16. Дейтел Х., Дейтел П., Нието Т. Как программировать для Internet & WWW М.: Бином, 2002, - 1184 С.

17. Денисов А.А., Колесников Д.Н. Теория больших систем управления. Л.: Энергоиздат, 1982. - 288 С.

18. Диго С.М. Проектирование и использование баз данных. М.: Финансы и статистика, 1995. -208 С.

19. Дрожжинов В.И. От теста не уйдешь// Мир ПК, N 2, 1993.

20. Жожикашвили В.А., Вишневский В.М. Сети массового обслуживания. Теория и применение к сетям ЭВМ. М.: Радио и связь, 1988. – 192 С.

21. Журков А.П., Аминев Д.А., Гусева П.А., Мирошниченко С.С., Петросян П.А. Анализ возможностей применения подходов самодиагностирования к распределенной радиотехнической системе наблюдения// Системы управления, связи и безопасности. 2015. № 4. С. 114-122.

22. Задорожный В.Н. Модели и системы. Анализ научного мышления. - Омск, ОМГТУ, 1999. - 100 с.

23. Задорожный В.Н. Статистическое моделирование. - Омск: Изд-во ОмГТУ, 1996. - 92с.

24. Иванов П. Управление информационными системами: базовые концепции и тенденции развития // Открытые системы, №4, 1999, С. 37-43.

25. Камер Д. Компьютерные сети и Internet. М.: Вильямс, 2002.- 640 с.

26. Кастаньетто Д. и др. Профессиональное PHP программирование. М.: Символ – Плюс, 2001. - 912 С.

27. Кениг Д., Штоян Д. Методы теории массового обслуживания.- М.: Радио и связь, 1981. – 127 с.

28. Киллелиа П. Тюнинг WEB-сервера. СПб.: Питер, 2003. - 528 С.

29. Клейнен Дж. Статистические методы в имитационном моделировании. - М.: Статистика, 1978. - Вып. 1. - 221 с.; Вып. 2 - 335 с.

30. Клейнрок Л. Вычислительные системы с очередями: Пер. с англ. М.: Мир, 1979 - 600 С.

31. Клейнрок Л. Теория массового обслуживания: Пер. с англ. М.: Машиностроение, 1979 – 432 С.

32. Когаловский М.Р. Энциклопедия технологий баз данных. М.: Финстат, 2002. - 800 С.

33. Кокс Д. Р., Смит У. Л. Теория очередей: Пер. с англ. М.: Мир, 1966 - 218 С.

34. Колесов Ю.Б., Сениченков Ю.Б., Визуальное моделирование, СПб.: Мир и Семья, 2000. - 256 С.

35. Конвей Р., Максвелл В., Миллер Л. Теория расписаний: Пер с англ. / Под ред. Г.П. Башарина. – М.: Наука, 1975 – 159 с.

36. Ланг К., Чоу Д. Публикация баз данных в Интернете. – СПб.: Символ-Плюс, 2004.

37. Ларионов А.М., Майоров С.А., Новиков Г.И. Вычислительные комплексы, системы и сети. Л.: Энергоатомиздат, 1987. - 256 С.

38. Лебедев А.Н., Чернявский Е.А. Вероятностные методы в вычислительной технике. М.: Высшая школа, 1986. - 312 С.

39. Левенчук А. Интранет предлагает решения для корпорации// Рынок ценных бумаг, №16, 1996.

40. Липский Н. Комбинаторика для программистов. М.: Мир, 1985. – 374 С.

41. Мазуркевич А., Еловой Д. PHP: настольная книга программиста. М.: Новое знание 2003. -480 С.

42. Майерс Г. Надежность программного обеспечения. - М.: Мир, 1980.

43. Мальцева С.В. Информационное моделирование WEB-ресурсов Интернет. М.: Глобус, 2003. - 216 С.

44. Матвеев В.Ф., Ушаков В.Г. Системы массового обслуживания. М.: Изд-во МГУ, 1984. – 240 С.

45. Мельтцер К., Михальски Б. Разработка CGI-приложений на Perl. М.: Вильямс, 2001. - 400 С.

46. Мещеряков Е.В., Хомоненко А.Д. Публикация баз данных в Интернете Спб.: BHV-Санкт-Петербург, 2001. - 560 С.

47. Олифер В.Г., Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы. 2-изд. СПб: Питер-пресс, 2002. - 864 С.

48. Павловский Ю. Н. Имитационные модели и системы. М.: ФАЗИС, 2000. - 144 С.

49. Перегудов Ф.П., Тарасенко Ф.П. Введение в системный анализ. М: Высшая школа, 1989. - 367 С.

50. Петренко А., Бритвина Е., Грошев С., Монахов А., Петренко О. Тестирование сетевой инфраструктуры// Открытые системы, 09/2003.

51. Профессиональные стандарты в области информационных технологий. - М.: АП КИТ, 2008. - 616 с.

52. Ройс У. Управление проектами по созданию программного обеспечения. - М.: Лори, 2002.

53. Свами М., Тхуласираман К. Графы, сети и алгоритмы: Пер. с англ. М.: Мир, 1984. - 455 С.

54. Седова Н.А. Формирование лингвистических переменных для задач судовождения// Эксплуатация морского транспорта. – 2013. – №2(72). – С. 19-23.

55. Седова Н.А., Перечёсов В.С., Седов В.А. Удержание судна на курсе на базе нечеткой логики с учетом скорости судна// Автоматизация процессов управления. – 2013. – № 2. – С.74-79.

56. Советов Б.Я., Яковлев С.А. Моделирование систем //3-е изд., М:Высшая школа, 2001. - 344 С.

57. Создание Intranet. Официальное руководство Microsoft. - СПб.: BHV, 1998.

58. Спицын А.А. Алгоритм настройки параметров расписания облачных вычислений на основе оптимизации роя частиц// Вестник Рязанского государственного радиотехнического университета, № 75, 2021. С. 44-52.

59. Спицын А.А. Исследование расширенной архитектурной модели облачных вычислений // Сб. ст. Междунар. НПК «Материалы и методы инновационных научно-практических исследований и разработок» (Пенза, 17.03.2021 г.). – Уфа: OMEGA SCIENCE, 2021. – с. 36-44.

60. Спицын А.А. Оптимизация планирования в облачных вычислениях// IV Всероссийская научная конференция с международным участием

«Информационные технологии в моделировании и управлении: подходы, методы, решения». – Тольятти, 2021. С. 315-323.

61. Спицын А.А. Оптимизированный по стоимости эвристический алгоритм для планирования рабочих процессов в облачной среде сервисов IaaS// Системы управления и информационные технологии, №1(83), 2021. – С. 30-37.

62. Спицын А.А. Результаты вычислительного эксперимента по планированию облачных вычислений на основе оптимизации роя частиц// Информационные технологии моделирования и управления, №1(123), 2021. – С. 68-79.

63. Спицын А.А., Львович И.Я., Зеленина А.Н. Диспетчерское управление распределением процессов в облачных вычислениях. – Свидетельство о регистрации программы для ЭВМ № 2021611994 от 10.02.2021. М.: ФИПС, 2021.

64. Спицын А.А., Мутин Д.И. Оптимизация миграции виртуальных машин в облачных вычислениях с помощью эффективного алгоритма размещения// Матер. 16-й Международной научно-технической конференции «Современные инструментальные системы, информационные технологии и инновации» (МТО-56). – Курск, 2021. С. 251-256.

65. Спицын А.А., Мутин Д.И. Распределение ресурсов и планирование заданий в облачной среде на основе алгоритма оптимизации роя частиц и R-фактора// Моделирование, оптимизация и информационные технологии. 2020; 8(4). https://moitvivt.ru/ru/journal/pdf?id=869. DOI: 10.26102/2310-6018/2020.31.4.023.

66. Стрелкова Е. Интеграция данных предприятия / Открытые системы, №4, 2003, С. 58-60.

67. Технология системного моделирования/ Е.Ф. Аврамчук, А.А. Вавилов, С.В. Емельянов и др. / Под ред. С.В. Емельянова. – М.: Машиностроение, 1988. – 320 с.

68. Томсетт Р. Радикальное управление ИТ проектами. М.: Лори, 2005.

69. Увайсов С.У., Иванов И.А., Кошелев Н.А. Методика обеспечения диагностируемости электронных средств космических аппаратов по ранговому критерию на ранних этапах проектирования// Качество. Инновации. Образование. 2012. № 1 (80). С. 60-63.

70. Уемов А.И. Системный подход и общая теория систем. М.: Мысль, 1978.- 272 С.

71. Управление качеством продукции. Введение в системы менеджмента качества / С. Пономарев, С. Мищенко, В.Я. Белобрагин. М.: Стандарты и качество, 2005.

72. Феллер В. Введение в теорию вероятностей и её приложения: в 2-х т. - М.: Мир,1967. - т.1, 498 с.

73. Феррари Д. Оценка производительности вычислительных систем. М.: Мир, 1981. - 576 С.

74. Французов Д. Оценка производительности вычислительных систем. Открытые системы, №2, 1996, С. 58-66.

75. Фролов А.В. Фролов Г.В. Локальные сети персональных компьютеров. М.: ДИАЛОГ-МИФИ, 1993. - 176 С.

76. Харари Ф. Теория графов. М.: Мир, 1973. - 300 С.

77. Харрингтон Дж. Проектирование реляционных баз данных. М: Лори-Пресс, 2000. - 230 С.

78. Харт Д.М. Системное программирование в среде Win32 - 2-е издание. М.: Вильямс, 2001. 464 С.

79. Хилайер С., Мизик Д. Программирование Active Server Pages. М.: Русская редакция. 2000. - 320 С.

80. Храмцов П.Б., Брик С.А. и др. Основы web-технологий. М.: ИНТУИТ.ру, 2003. – 512 С.

81. Цаленко М.Ш. Моделирование семантики в базах данных. М.: Наука, 1989. - 287 С.

82. Черняк Л. Intranet - объективная реальность, данная нам// Рынок ценных бумаг, №3, 1997.

83. Чжо З.Е., Чжо З.Л., Наинг Л.А. Разработка методики децентрализованной диагностики и диагностируемости с использованием модели тестирования// Вести высших учебных заведений Черноземья. 2015. № 3. С. 60-70.

84. Чистяков В.П. Курс теории вероятностей - 5-е изд. М.: Агар, 2000. - 255 С.

85. Шагурина Н. Web-службы: новая парадигма интеграции? / Сетевой журнал №2, М., 2003 – С. 14-17.

86. Шеннон Р. Имитационное моделирование систем. М.: Мир, 1978. - 418 С.

87. Спицын А.А. Управление расписанием облачных вычислений на основе оптимизации роя частиц// Решение: матер. 11-й Всеросс. НПК. – Пермь: Изд-во Перм. нац. исслед. политехн. ун-та, 2022. – С. 255-259.

88. Abdalov A.V., Grishakov V.G., Loginov I.V., Spitsyn A.A. Resource planning algorithm for reconfigurable information system development in the case of several sources of resources// AIP Conference Proceedings, 2402, 050012, https://doi.org/10.1063/5.0071507. **(Scopus)**

89. Spitsyn A.A., Mutin D.I. Approach to creating a hierarchical load balancing strategy in cloud structures// Modern informatization problems in the technological and telecommunication systems analysis and synthesis (MIP-2021'AS): Proceedings of the XXVI-th International Open Science Conference (Yelm, WA, USA, January 2021). - Yelm, WA, USA: Science Book Publishing House, 2021. – p. 167-175.

90. Spitsyn A.A., Mutin D.I., Bikkulov I.M., Frantsisko O.Yu., Atlasov I.V. Multi-objective selection of structure variants for a corporate heterogeneous integrated system of information management// International Journal on Information Technologies & Security, № 1 (vol. 13), 2021, p. 27-38. **(WoS)**

91. Spitsyn A.A. About optimization approaches to cloud load balancing// Modern informatization problems in simulation and social technologies (MIP-2022'SCT): Proceedings of the XXVII-th International Open Science Conference (Yelm, WA, USA, January 2022). - Yelm, WA, USA: Science Book Publishing

House, 2022. – p. 174-183.

92. Spitsyn A.A., Mutin D.I. Architecture of the prototype of the task distribution management system in cloud environments// Modern informatization problems in the technological and telecommunication systems analysis and synthesis (MIP-2023'AS): Proceedings of the XXIIX-th International Open Science Conference (Yelm, WA, USA, January 2023). - Yelm, WA, USA: Science Book Publishing House, 2023.

93. Mateusz G., Alicja G., Pascal B. Cloud brokering: Current practices and upcoming challenges// IEEE Cloud Comput., 2015, Vol. 2, no.2, p. 40–47.

94. Milani A.S., Navimipour N.J. Load balancing mechanisms and techniques in the cloud environments: Systematic literature review and future trends// J. Netw. Comput. Appl., 2016. Vol. 71, no. 1, p. 86–98.

95. Jiang Y.C. A survey of task allocation and load balancing in distributed systems// IEEE Trans. Parallel Distrib. Syst., 2016, Vol. 27, no. 2, p. 585–599.

96. Ning J., Cao Z., Dong X., Liang K., Ma H., Wei L. Auditable-Time Outsourced Attribute-Based Encryption for Access Control in Cloud Computing// IEEE Transactions on Information Forensics And Security, 2017.

97. Jansen W., Grance T. Guidelines on security and privacy in public cloud computing// NIST, SP.800-144, 2011, p. 800-144.

98. Barcelo M., Correa A., Llorca J., Tulino A.M., Vicario J.L., Morell A. IoT-cloud service optimization in next generation smart environments// IEEE J. on Selected Areas in Communications, 2016, vol. 34, no. 12, p. 4077-4090.

99. Barcelo M., Llorca J., Tulino A., Raman N. The cloud service distribution problem in distributed cloud networks// IEEE ICC 2015 SAC - Data Storage and Cloud Computing, London, United Kingdom, Jun. 2015.

100. Stojmenovic I., Wen S. The fog computing paradigm: Scenarios and security// Federated Conference Computer Science and Information Systems (FedCSIS), on, pp. 1–8, September, 2014. https://doi.org/10.15439/2014F503.

101. Jamshidi P., Ahmad A., Pahl C., Cloud migration research: a systematic review// IEEE Transactions on Cloud Computing, 2013, vol.1, no.2, p.142-157.

102. Liu X., Wang C., Zhou B.B., Chen J., Yang T., Zomaya A.Y. Priority-based consolidation of parallel workloads in the cloud// IEEE Trans Parallel DistribSyst, 2012, vol.24, no.9.

103. Yao F., Yao Y., Chen H., Li T., Lin M., Zhang X. An intelligent scheduling algorithm for complex manufacturing system simulation with frequent synchronizations in a cloud environment// Memetic Computing, 2019, vol. 11, p. 357–370.

104. Wiseman Y., Feitelson D.G. Paired gang scheduling// IEEE Trans Parallel DistribSyst, 2003, vol. 14, no .6, p.581–592.

105. Hussein M.K., Mousa M.H., Alqarni M.A. A placement architecture for a container as a service (CaaS) in a cloud environment// Journal of Cloud Computing, 2019, vol.8, no.1, p.7.

106. Zhang R., Zhong A.M., Dong B., Tian F., Li R. Container-VM-PM architecture: A novel architecture for docker container placement// International Conference on Cloud Computing, Springer, 2018, p. 128-140.

107. Li C., Wang Y., Tang H., Luo Y. Dynamic multiobjective optimized replica placement and migration strategies for SaaS applications in edge cloud// Future Generation Computer Systems, 2019, vol.100, p.921-937.

108. Zaheer S., Malik A.W., Rahman A.U., Khan S.A. Locality-aware process placement for parallel and distributed simulation in cloud data centers// The Journal of Supercomputing, 2019, vol.1, p.23.

109. Zhou, R., Li, Z. and Wu, C., "An Efficient Online Placement Scheme for Cloud Container Clusters," IEEE Journal on Selected Areas in Communications, vol.37, no.5, pp.1046-1058, 2019.

110. Liu, B., Li, P., Lin, W., Shu, N., Li, Y. and Chang, V., "A new container scheduling algorithm based on multiobjective optimization," Soft Computing, vol.22, no.23, pp.7741-7752, 2018.

111. Nazir, B., "QoS-aware VM placement and migration for hybrid cloud infrastructure," The Journal of Supercomputing, vol.74, no.9, pp.4623-4646, 2018. https://doi.org/10.1007/s11227-017-2071-1

112. Satpathy, A., Addya, S.K., Turuk, A.K., Majhi, B. and Sahoo, G., "Crow search based virtual machine placement strategy in cloud data centers with live migration," Computers & Electrical Engineering, vol.69, pp.334-350, 2018.

113. JKRSastry, M TrinathBasu, Securing Multi-tenancy systems through user spaces defined within the database level, Jour of Adv Research in Dynamical & Control Systems, Volume 10, issue 7, Page 405-412, 2018

114. J. K. R. Sastry, K. Sai Abhigna, R. Samuel and D. B. K. Kamesh, Architectural models for fault tolerance within clouds at the infrastructure level, ARPN Journal of Engineering and Applied Sciences, VOL. 12, NO. 11, 2017, Pages 3463-3469

115. Jain, M., Singh, V. and Rani, A., "A novel natureinspired algorithm for optimization: Squirrel search algorithm," Swarm and evolutionary computation, vol.44, pp.148-175, 2019.

116. Mirjalili, S. and Lewis, A., "The whale optimization algorithm," Advances in engineering software, vol.95, pp.51-67, 2016.

117. Buyya R., Yeo C.S., Venugopal S., Broberg J., Brandic I. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems. 2009. 25(6), pp.599–616.

118. Hayes B. Cloud computing. Communications of the ACM, 2008, 51(7), pp.9–11.

119. Kennedy J., Eberhart R. Particle swarms optimization. IEEE International Conference on Neural Networks, 1995. 4, pp.1942–1948.

120. Pandey S., Wu L., Guru S.M., Buyya R. A Particle Swarm Optimization-Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments. 2010 24th IEEE international Conference on Advanced Information Networking and Applications (AINA), IEEE. 2010. pp.400–407.

121. Liu P. Cloud computing definition and characteristics. China cloud computing. 2009. http://www.chinacloud.cn/2009-2-25.

122. Kumar P., Verma A. Independent task scheduling in cloud computing

by improved genetic algorithm. International Journal of Advanced Research in Computer Science and Software Engineering, 2012. 2(5).

123. Roy P., Mejbah M., Das N. Heuristic based task scheduling in multiprocessor systems with genetic algorithm by choosing the eligible processor. International Journal of Distributed and Parallel Systems (IJDPS). 2012. 3(4).

124. Chalack S.A., Razavi S.N., Harounabadi A. Job scheduling on the grid environment using max-min firefly algorithm. International Journal of Computer Applications Technology and Research. 2014. 3(1), pp.63–67.

125. Vinothina V., Sridaran R., Ganapathi P. A survey on resource allocation strategies in cloud computing. International Journal of Advanced Computer Science and Applications, 2012. 3(6), pp.97–104.

126. Alkayal E.S., Jennings N.R., Abulkhair M.F. Efficient Task Scheduling Multi-Objective Particle Swarm Optimization in Cloud Computing. 2016 IEEE 41st Conference on Local Computer Networks Workshops (LCN Workshops). 2016. November; IEEE. pp.17–24.

127. Buyya A.R., Nath B. Nature's Heuristics for Scheduling Jobs on Computational Grids. 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000), 2000. India.

128. Zhang L., Chen Y., Yang B. Task Scheduling Based on PSO Algorithm in Computational Grid. 2006 Proceedings of the 6th International Conference on Intelligent Systems Design and Applications, Jinan, China. 2006. p.2.

129. Kalra M., Singh S. A review of metaheuristic scheduling techniques in cloud computing. Egyptian Informatics Journal, 2015. 16(3), pp.275–295.

130. Verma A., Kaushal S. Bi-criteria priority based particle swarm optimization workflow scheduling algorithm for cloud. 2014 Recent Advances in Engineering and Computational Sciences (RAECS). IEEE. 2014. pp.1–6.

131. Beegom A.A., Rajasree M.S. A Particle Swarm Optimization Based Pareto Optimal Task Scheduling in Cloud Computing. International Conference in Swarm Intelligence, Cham, Springer. 2014, pp.79–86.

132. Jun Xue S., Wu W. Scheduling workflow in cloud computing based on hybrid particle swarm algorithm. Indonesian Journal of Electrical Engineering, 2012. 10(7), pp.1560–1566.

133. Guo L., Zhao S., Shen S., Jiang C. Task scheduling optimization in cloud computing based on heuristic algorithm. Journal of Networks, 2012. (7), pp.547–553.

134. Varalakshmi P., Ramaswamy A., Balasubramanian A., Vijaykumar P. An optimal workflow based scheduling and resource allocation in cloud. Advances in Computing and Communications, First International Conference, ACC, 2011. pp.411–420.

135. Zhong H., Tao K., Zhang X. An Approach to Optimized Resource Scheduling Algorithm for Open-Source Cloud Systems. Fifth Annual China Grid Conference. 2010.

136. Selvarani S., Sadhasivam G. Improved Cost-Based Algorithm for Task Scheduling in Cloud Computing. IEEE International Conference on Computational Intelligence and Computing Research (ICCIC). 2010.

137. Liu Z., Wang X. A pso-based algorithm for load balancing in virtual machines of cloud computing environment. Advances in Swarm Intelligence, ser. Lecture Notes in Computer Science, Berlin, Heidelberg, Springer. 2012. Vol. 7331, pp.142–147.

138. Zhan S., Huo H. Improved PSO-based task scheduling algorithm in cloud computing. Journal of Information and Computational Science, 2012. 9(13), pp.3821–3829.

139. Liu J., Guo Luo X., Zhang X.M.F. Job scheduling algorithm for cloud computing based on particle swarm optimization. Advanced Materials Research, 2013. 662, pp.957–960.

140. Jeyarani R., Nagaveni N., Vasanth Ram R. Design and implementation of adaptive power-aware virtual machine provisioner (APA-VMP) using swarm intelligence. Future Generation Computer Systems, 2012. 28(5), pp.811–821.

141. Liu Y., Zhu H. A survey of the research on power management techniques for high-performance systems. Software Practice and Experience, 2010. 40(11), pp.943–964.

142. Feller E., Rilling L., Morin C. Energy-Aware Ant Colony Based Workload Placement in Clouds. 12th International Conference on Grid Computing, ser. Grid, IEEE Computer Society. 2011. Vol. 11, pp.26–33.

143. Patel, G., Mehta, R., and Bhoi, U., "Enhanced load balanced min-min algorithm for static meta task scheduling in cloud computing," Procedia Computer Science, vol. 57, pp. 545–553, 2015.

144. Bitam, S., "Bees life algorithm for job scheduling in cloud computing," in Proceedings of The Third International Conference on Communications and Information Technology, 2012, pp. 186–191.

145. Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., and Stoica, I., "Above the clouds: A Berkeley view of cloud computing," Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS, vol. 28, no. 13, p. 2009, 2009.

146. InfoTech, "What is cloud computing," IBM Journal of Research and Development, vol. 60, no. 4, pp. 41–44, 2012.

147. Savu, L., "Cloud computing: Deployment models, delivery models, risks and research challenges," in 2011 International Conference on Computer and Management (CAMAN). IEEE, 2011, pp. 1–4.

148. Masdari, M., ValiKardan, S., Shahi, Z., and Azar, S. I., "Towards workflow scheduling in cloud computing: a comprehensive analysis," Journal of Network and Computer Applications, vol. 66, pp. 64–82, 2016.

149. Zhou, X., Zhang, G., Sun, J., Zhou, J., Wei, T., and Hu, S., "Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based heft," Future Generation Computer Systems, vol. 93, pp. 278–289, 2019.

150. Rodriguez, M. A. and Buyya, R., "Budget-driven scheduling of scientific workflows in IAAS clouds with fine-grained billing periods," ACM Transactions on Autonomous and Adaptive Systems (TAAS), vol. 12, no. 2, pp. 1–22, 2017.

151. Sahni, J. and Vidyarthi, D. P., "A cost-effective deadline-constrained

dynamic scheduling algorithm for scientific workflows in a cloud environment," IEEE Transactions on Cloud Computing, vol. 6, no. 1, pp. 2–18, 2015.

152. Anwar, N. and Deng, H., "Elastic scheduling of scientific workflows under deadline constraints in cloud computing environments," Future Internet, vol. 10, no. 1, p. 5, 2018.

153. Nasr, A. A., El-Bahnasawy, N. A., Attiya, G., and El-Sayed, A., "Costeffective algorithm for workflow scheduling in cloud computing under deadline constraint," Arabian Journal for Science and Engineering, vol. 44, no. 4, pp. 3765–3780, 2019.

154. Manasrah, A. M. and Ba Ali, H., "Workflow scheduling using hybrid gapso algorithm in cloud computing," Wireless Communications and Mobile Computing, vol. 2018, 2018.

155. Liu, J., Pacitti, E., Valduriez, P., and Mattoso, M., "Parallelization of scientific workflows in the cloud," 2014.

156. Juve, G., Chervenak, A., Deelman, E., Bharathi, S., Mehta, G., and Vahi, K., "Characterizing and profiling scientific workflows," Future Generation Computer Systems, vol. 29, no. 3, pp. 682–692, 2013.

157. Sossa, M. A. R., "Resource provisioning and scheduling algorithms for scientific workflows in cloud computing environments," Ph.D. dissertation, University of Melbourne, Department of Computing and Information Systems, 2016.

158. Li, Z., Ge, J., Hu, H., Song, W., Hu, H., and Luo, B., "Cost and energy aware scheduling algorithm for scientific workflows with deadline constraint in clouds," IEEE Transactions on Services Computing, vol. 11, no. 4, pp. 713–726, 2015.

159. Chawla, Y. and Bhonsle, M., "A study on scheduling methods in cloud computing," International Journal of Emerging Trends & Technology in Computer Science (IJETTCS), vol. 1, no. 3, pp. 12–17, 2012.

160. Garey, M. R., "Computers and intractability: A guide to the theory of np-completeness," Revista Da Escola De Enfermagem Da USP, vol. 44, no. 2, p. 340, 1979.

161. Awad, A., El-Hefnawy, N., and Abdel kader, H., "Enhanced particle swarm optimization for task scheduling in cloud computing environments," Procedia Computer Science, vol. 65, pp. 920–929, 2015.

162. Rimal, B. P. and Maier, M., "Workflow scheduling in multi-tenant cloud computing environments," IEEE Transactions on parallel and distributed systems, vol. 28, no. 1, pp. 290–304, 2016.

163. Haidri, R. A., Katti, C. P., and Saxena, P. C., "Cost effective deadline aware scheduling strategy for workflow applications on virtual machines in cloud computing," Journal of King Saud University-Computer and Information Sciences, 2017.

164. Elsherbiny, S., Eldaydamony, E., Alrahmawy, M., and Reyad, A. E., "An extended intelligent water drops algorithm for workflow scheduling in cloud computing environment," Egyptian informatics journal, vol. 19, no. 1, pp. 33–55, 2018.

165. Kalra, M. and Singh, S., "A review of metaheuristic scheduling tech-

niques in cloud computing," Egyptian informatics journal, vol. 16, no. 3, pp. 275–295, 2015.

166. Jiang, Y., Huang, Z., and Tsang, D. H., "Towards max-min fair resource allocation for stream big data analytics in shared clouds," IEEE Transactions on Big Data, vol. 4, no. 1, pp. 130–137, 2016.

167. Fard, H. M., Prodan, R., Barrionuevo, J. J. D., and Fahringer, T., "A multi-objective approach for workflow scheduling in heterogeneous environments," in 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012). IEEE, 2012, pp. 300–309.

168. Zheng, W., Qin, Y., Bugingo, E., Zhang, D., and Chen, J., "Cost optimization for deadline-aware scheduling of big-data processing jobs on clouds," Future Generation Computer Systems, vol. 82, pp. 244–255, 2018.

169. Alworafi, M. A., Dhari, A., El-Booz, S. A., Nasr, A. A., Arpitha, A., and Mallappa, S., "An enhanced task scheduling in cloud computing based on hybrid approach," in Data Analytics and Learning. Springer, 2019, pp. 11–25.

170. Deelman, E., Vahi, K., Juve, G., Rynge, M., Callaghan, S., Maechling, P. J., Mayani, R., Chen, W., Da Silva, R. F., Livny, M. et al., "Pegasus, a workflow management system for science automation," Future Generation Computer Systems, vol. 46, pp. 17–35, 2015.

171. Cao, J., Wen, L., and Liu, X., Process-Aware Systems: First International Workshop, PAS 2014, Shanghai, China, October 17, 2014. Proceedings. Springer, 2015, vol. 495.

172. Chen, W. and Deelman, E., "Workflowsim: A toolkit for simulating scientific workflows in distributed environments," in 2012 IEEE 8th International Conference on E-Science. IEEE, 2012, pp. 1–8.

173. Voevodin, V. and Sobolev, S., Supercomputing: Third Russian Supercomputing Days, RuSCDays 2017, Moscow, Russia, September 25–26, 2017, Revised Selected Papers. Springer, 2017, vol. 793.

174. Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., and Buyya, R., "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," Software: Practice and experience, vol. 41, no. 1, pp. 23–50, 2011.

175. Rodriguez, M. A. and Buyya, R., "Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms," Future Generation Computer Systems, vol. 79, pp. 739–750, 2018.

176. Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M.-H., and Vahi, K., "Characterization of scientific workflows," in 2008 third workshop on workflows in support of large-scale science. IEEE, 2008, pp. 1–10.

177. Mehta, G., Juve, G., and Chen, W., "Workflow generator," URL: https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator, 2009.

178. Amar, M., Anurag, K., Rakesh, K., Rupesh, K., Prashant, Y.: SLA driven load balancing for web applications in cloud computing environment. Inf. Knowl. Manage. 1(1), 5–13 (2011)

179. Mann, Z.Б.: Allocation of virtual machines in cloud data centers—a survey of problem models and optimization algorithms. ACM Comput. Surv.

(CSUR), 48(1), 11 (2015)

180. Jung, G., Hiltunen, M.A., Joshi, K.R., Schlichting, R.D., Mistral, C.P.: Dynamically managing power, performance, and adaptation cost in cloud infrastructures. In: IEEE 30th International Conference on Distributed Computing Systems (ICDCS), pp. 62–73 (2010)

181. Katyal, M., Mishra, A.: A comparative study of load balancing algorithms in cloud computing environment. arXiv preprint arXiv:1403.6918 (2014)

182. Tian, W., Zhao, Y., Zhong, Y., Xu, M., Jing, C.: A dynamic and integrated load-balancing scheduling algorithm for cloud datacenters. In: 2011 IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS), pp. 311–315. IEEE (2011)

183. Malhotra, M., Singh, A.: Adaptive framework for load balancing to improve the performance of cloud environment. In: 2015 IEEE International Conference on Computational Intelligence & Communication Technology (CICT), pp. 224–228. IEEE (2015)

184. Hao, Y., Liu, G., Lu, J.: Three levels load balancing on cloudsim. Int. J. Grid Distrib. Comput. 7(3), 71–88 (2014)

185. Wang, S.C., Yan, K.Q., Liao, W.P., Wang, S.S.: Towards a load balancing in a three-level cloud computing network In: 2010 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT), vol. 1, pp. 108–113. IEEE (2010)

186. Dhurandher, S.K., Obaidat, M.S., Woungang, I., Agarwal, P., Gupta, A., Gupta, P.: A clusterbased load balancing algorithm in cloud computing. In: 2014 IEEE International Conference on Communications (ICC), pp. 2921–2925. IEEE (2014)

187. Culler, D.E., Karp, R.M., Patterson, D.A., Sahay, A., Santos, E.E., Schauser, K.E., Subramonian, R., von Eicken, T.: LogP: a practical model of parallel computation (1993)

188. Culler, D.E., Karp, R.M., Patterson, D.A., Sahay, A., Santos, E.E., Schauser, K.E., Subramonian, R., von Eicken, T.: LogP: a practical model of parallel computation. Commun. ACM 39(11), 78–85 (1996)

189. Xu, G., Pang, J., Fu, X.: A load balancing model based on cloud partitioning for the public cloud. Tsinghua Sci. Technol. 18(1), 34–39 (2013)

190. Smith, J., Nair, R.: The architecture of virtual machines. Computer 38(50), 32–38 (2005)

191. Chaoqun, X., Yi, Z., Wei, Z.: A load balancing algorithm with key resource relevance for virtual cluster. Int. J. Grid Distrib. Comput. 6(5), 1–16 (2013)

192. Xu, M., Tian, W., Buyya, R.: A survey on load balancing algorithms for VM placement in cloud computing. arXiv preprint arXiv:1607.06269 (2016)

193. Eager, D.L., Lazowska, E.D., Zahorjan, J.: Adaptive load sharing inhomogeneous distributed systems. IEEE Trans. Softw. Eng. 12(5), 662–675 (1986)

194. Badidi, E.: Architecture and services for load balancing in object distributed systems. Ph.D. thesis, Faculty of High Studies, University of Montreal, Mai (2000)

195. Babu L.D., Krishna P.V. Versatile time-cost algorithm (VTCA) for scheduling non-preemptive tasks of time critical workflows in cloud computing systems// International Journal of Communication Networks and Distributed Systems, 2013. Vol. 11, No. 4, pp.390–411.

196. Bouali L., Oukfif K., Bouzefrane S., Oulebsir-Boumghar F. A hybrid algorithm for DAG application scheduling on computational grids// International Conference on Mobile, Secure and Programmable Networking, 2015, pp.63–77, Springer.

197. Calheiros R.N., Ranjan R., Beloglazov A., De Rose C.A., Buyya R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms// Software: Practice and Experience, 2011. Vol. 41, No. 1, pp.23–50.

198. Chen J., Zhang J., Song A. Efficient data and task co-scheduling for scientific workflow in geo-distributed datacenters// 2017 Fifth International Conference on Advanced Cloud and Big Data (CBD), pp.63–68, IEEE.

199. Chen W-N., Zhang J. An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements// IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 2009. Vol. 39, No. 1, pp.29–43.

200. Chen W-N., Zhang J. A set-based discrete PSO for cloud workflow scheduling with user-defined QoS constraints// 2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC). 2012, pp.773–778.

201. Choudhary A., Gupta I., Singh V., Jana P.K. A GSA based hybrid algorithm for bi-objective workflow scheduling in cloud computing// Future Generation Computer Systems, 2018. Vol. 83, No. C, pp.14–26.

202. Dogan A., Ozguner F. Biobjective scheduling algorithms for execution time-reliability trade-off in heterogeneous computing systems// The Computer Journal, Vol. 48, No. 3, pp.300–314.

203. Dongarra J.J., Jeannot E., Saule E., Shi Z. Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems// Proceedings of the Nineteenth Annual ACM Symposium on Parallel Algorithms and Architectures, 2007, pp.280–288, ACM.

204. Fard H.M., Prodan R., Barrionuevo J.J.D., Fahringer T. A multi-objective approach for workflow scheduling in heterogeneous environments// Proc. of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2012), pp.300–309, IEEE Computer Society.

205. Guo T., Liu J., Hu W., Wei M. Energy-aware fault-tolerant scheduling under reliability and time constraints in heterogeneous systems// International Conference on Intelligent Computing, 2018, pp.36–46, Springer.

206. Hakem M., Butelle F. Reliability and scheduling on systems subject to failures// International Conference on Parallel Processing, 2007. ICPP 2007, pp.38–48.

207. Jian C., Tao M., Wang Y. A particle swarm optimisation algorithm for cloud-oriented workflow scheduling based on reliability// International Journal of Computer Applications in Technology, 2014. Vol. 50, Nos. 3–4, pp.220–225.

208. Juve G., Chervenak A., Deelman E., Bharathi S., Mehta G., Vahi K. Characterizing and profiling scientific workflows// Future Generation Computer Systems, 2013. Vol. 29, No. 3, pp.682–692.

209. Kennedy J. Particle swarm optimization// Encyclopedia of Machine Learning. - Springer, New York, NY, USA, 2011, pp.760–766.

210. Kianpisheh S., Charkari N.M., Kargahi M. Reliability-driven scheduling of time/cost-constrained grid workflows// Future Generation Computer Systems, 2016. Vol. 55, No. C, pp.1–16.

211. Kumar H., Chauhan N.K., Yadav P.K. A task allocation model for minimizing system cost and maximizing reliability of distributed computing system// International Journal of Communication Networks and Distributed Systems, 2018. Vol. 20, No. 2, pp.226–243.

212. Lee Y.C., Zomaya A.Y., Yousif M. Reliable workflow execution in distributed systems for cost efficiency// 2010 11th IEEE/ACM International Conference on Grid Computing (GRID), pp.89–96, IEEE.

213. Mell P., Grance T. The NIST Definition of Cloud Computing, National Institute of Standards and Technology. http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf. 2011.

214. Oukfif K., Bouali L., Bouzefrane S., Oulebsir-Boumghar F. (Energy-aware dpso algorithm for workflow scheduling on computational grids// 2015 3rd International Conference on Future Internet of Things and Cloud (FiCloud), pp.651–656, IEEE.

215. Poola D., Ramamohanarao K., Buyya R. Enhancing reliability of workflow execution using task replication and spot instances// ACM Transactions on Autonomous and Adaptive Systems (TAAS), 2016. Vol. 10, No. 4, p.30.

216. Qin X., Jiang H. A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems// Parallel Computing, 2006. Vol. 32, No. 5, pp.331–356.

217. Rehani N., Garg R. Meta-heuristic based reliable and green workflow scheduling in cloud computing// International Journal of System Assurance Engineering and Management, 2017. Vol. 9, No. 4, pp.1–10.

218. Sih G.C., Lee E.A. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures// IEEE Transactions on Parallel and Distributed systems, 1993, Vol. 4, No. 2, pp.175–187.

219. Sinha A., Malo P., Deb K. A review on bilevel optimization: from classical to evolutionary approaches and applications// IEEE Transactions on Evolutionary Computation, 2018. Vol. 22, No. 2, pp.276–295.

220. Tang X., Li K., Qiu M., Sha E.H-M. A hierarchical reliability-driven scheduling algorithm in grid systems// Journal of Parallel and Distributed Computing, 2012. Vol. 72, No. 4, pp.525–535.

221. Topcuoglu H., Hariri S., Wu M-Y. Performance-effective and low-complexity task scheduling for heterogeneous computing// IEEE Transactions on Parallel and Distributed Systems, 2002. Vol. 13, No. 3, pp.260–274.

222. Ullman J.D. NP-complete scheduling problems// Journal of Computer and System Sciences, 1975. Vol. 10, No. 3, pp.384–393.

223. Wang X., Yeo C.S., Buyya R., Su J. Optimizing the makespan and reliability for workflow applications with reputation and a look-ahead genetic algorithm// Future Generation Computer Systems, 2011. Vol. 27, No. 8, pp.1124–1134.

224. Wen Z., Cala J., Watson P., Romanovsky A. Cost effective, reliable and secure workflow deployment over federated clouds// IEEE Transactions on Services Computing, 2016. Vol. 10, No. 6, pp.929–941.

225. Wu F., Wu Q., Tan Y. Workflow scheduling in cloud: a survey// The Journal of Supercomputing, 2015. Vol. 71, No. 9, pp.3373–3418.

226. Xiao X., Xie G., Xu C., Fan C., Li R., Li K. Maximizing reliability of energy constrained parallel applications on heterogeneous distributed systems// Journal of Computational Science, 2018. Vol. 26, No. C, pp.344–353.

227. Zhang J., Wang M., Luo J., Dong F., Zhang J. Towards optimized scheduling for data-intensive scientific workflow in multiple datacenter environment// Concurrency and Computation: Practice and Experience, 2015. Vol. 27, No. 18, pp.5606–5622.

228. Zhang L., Li K., Li C., Li K. Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems// Information Sciences, 2017. Vol. 379, pp.241–256.

Научное издание

Андрей Александрович СПИЦЫН
Денис Игоревич МУТИН
Олег Яковлевич КРАВЕЦ

ПРОЦЕССЫ МИГРАЦИИ ВИРТУАЛЬНЫХ МАШИН В ОБЛАЧНЫХ СРЕДАХ

Монография

Издание публикуется в авторской редакции

Дизайн обложки С.А. Кравец

Подписано в печать 12.02.2024. Формат 60x84 1/16
Усл. печ. л. 9,25. Заказ 000. Тираж 500 экз.

ООО Издательство «Научная книга»
394077, Россия, г. Воронеж, ул. 60-й Армии, 25-120
http://www.sbook.ru/

Отпечатано с готового оригинал-макета
в ООО «Цифровая полиграфия»
394036, Россия, г. Воронеж, ул. Куколкина, 6
Тел. (473) 261-03-61