УДК 004.75

Анализ методов разгрузки задач в системах граничных вычислений со множественным доступом

Чипсанова Е. В.✉, **Елагин В. С.**
Санкт-Петербургский государственный университет телекоммуникаций им. проф. М. А. Бонч-Бруевича
Санкт-Петербург, 193232, Российская Федерация

***Постановка задачи**: Граничные вычисления с множественным доступом являются перспективным решением, которое может решить проблемы с емкостью и производительностью в устаревших системах, таких как мобильные облачные вычисления. Проблемы перегрузки базовой сети, большой задержки и плохого качества обслуживания возникают из-за ограниченных ресурсов мобильных устройств при соединении несколькими переходами между конечными пользователями и облаком, высокой нагрузки со стороны ресурсоемких и критичных к задержкам приложений. **Используемые методы**: сравнительный метод оценки. **Новизна:** Приведен анализ методов разгрузки задач в системах граничных вычислений со множественным доступом. Предложены три метода разгрузки задач для сравнительного анализа. **Результат**: ознакомление с существующими методами разгрузки задач, а также выявление из представленных наиболее актуального. **Практическая значимость:** в предложении метода наиболее актуального в вопросе разгрузки задач в сетях граничных вычислений со множественным доступом.*

***Ключевые слова:** разгрузка задач, граничные вычисления с множественным доступом, сеть пятого поколения, адаптивная структура, COSTA, линейная регрессия*

***Источник финансирования:** Научная статья подготовлена в рамках прикладных научный исследований СПбГУТ, регистрационный номер 123060900012-6 в ЕГИСУ НИОКТР.*

Введение

Мобильные устройства имеют ограничения, такие как: скорость, память, объем памяти [1]. Приложения требуют высокоскоростного подключения к Интернету и высокой вычислительной мощности, что невозможно в мобильном устройстве [2]. Однако ресурсоемкие задачи можно переносить на внешние платформы, такие как облако, сеть, сервер MEC – так называемая разгрузка задач [3], реализовать которую можно двумя способами [4]:

1) полная разгрузка (*от англ.* Fully Offload) – весь процесс или задача переносится на облачный сервер или сервер MEC; полная разгрузка также известна как разгрузка Corse;

2) частичная разгрузка (*от англ.* Partial Offload) – задачи или процессы сначала делятся на разные части, а некоторые из них переносятся на сервер; выбранные фрагменты являются небольшой частью кода частичной разгрузки.

Библиографическая ссылка на статью:
Чипсанова Е. В., Елагин В. С. Анализ методов разгрузки задач в системах граничных вычислений со множественным доступом // Информационные технологии и телекоммуникации. 2023. Т. 11. № 1. С. 1–15. DOI: 10.31854/2307-1303-2023-11-1-1-15

Reference for citation:
Chipsanova E., Elagin V. Task Offloading methods in Multi-Access Edge Computing Systems. *Telecom IT*. 2023; 11(1):1–15. DOI: 10.31854/2307-1303-2023-11-1-1-15

В настоящее время существует ряд сложившихся проблем: плохое качество обслуживания, большая задержка, перегрузка базовой сети. Данные проблемы возникают из-за ограниченных ресурсов мобильных устройств, при соединении несколькими переходами между конечными пользователями и облаком, высокой нагрузки со стороны ресурсоемких и критичных к задержкам приложений. Принимая во внимание ограниченность ресурсов, повышение эффективности разгрузки задач и задержки в приложениях граничных вычислений с множественным доступом является актуальной проблемой, требующей решения. Целью данной работы является анализ существующих методов разгрузки задач в граничных вычислениях с множественным доступом и выявление наиболее актуального из рассмотренных методов.

Изучая релевантные работы, посвященные заявленной проблеме, были выбраны три публикации в целях проведения сравнительного анализа использованных методов исследования. С. Чен предложил адаптивную структуру [3], которая принимает различные сетевые ситуации и соответственно принимает решение о разгрузке. Т. Х. Тран представил итерационный алгоритм, основанный на методах локального поиска – COSTA [5], отвечает за кэширование услуги с учетом затрат и назначение разгрузки задач. К. Ким использовал линейную регрессию [6] для разгрузки на основе прогнозирования в MEC.

Адаптивная структура

Разработанная в [3] адаптивная структура поддерживает мобильные приложения с возможностью разгрузки в MEC.

Во-первых, предлагается шаблон проектирования, позволяющий приложению оставаться доступным при изменении контекста устройства и динамически распределяться между устройством, сервером MEC и облаком. Рассматривается механизм объектного прокси для поддержки адаптивной разгрузки в контексте MEC. Выгруженные вычисления предназначены для удаленного создания, миграции и вызова объектов, выполняющих вычисления.

Во-вторых, модель оценки предназначена для автоматического определения схемы разгрузки в соответствии с контекстом устройства, при котором различные части приложения могут выполняться на девайсах, серверах MEC и в облаке. Генерируется граф потока управления основными функциями приложения и извлекаются перемещаемые классы и вызовы их методов. Затем вводятся информационные модели для сбора исторической информации о среднем трафике данных для каждого вызова метода и времени их выполнения на устройстве, в облаке и на серверах MEC. Исходя из этого, предлагается алгоритм определения оптимального места развертывания для каждого класса.

Платформа реализована для поддержки шаблона проектирования и модели оценки, которые развернуты на каждом вычислительном узле в мобильной пограничной среде и позволяет приложениям оставаться доступными и разгружаться между устройством, серверами MEC и облаком, когда мобильное устройство перемещается из одного места в другое. Более того, фреймворк может автоматически определять схему разгрузки для приложения.

Системная модель

Приложения для Android состоят из функциональных блоков, при этом главный модуль служит точкой входа для них. При каждом выполнении приложения вызываются методы класса, которые используют данные и методы, внутренние для того же класса, или вызывает определенные методы других классов. Такой процесс можно охарактеризовать графом потока управления.

Чтобы повысить производительность разгрузки, необходимо сгруппировать часто взаимодействующие объекты и разгрузить их как единое целое. В результате мы рассматриваем объекты, созданные в главном модуле, как основные объекты. Кроме того, другие объекты, созданные в телах функций основных объектов, должны быть разгружены вместе с соответствующими базовыми объектами как единое целое. Таким образом, задачу определения схемы разгрузки вычислений можно было бы свести к решению задачи плана размещения объектов для каждого класса в основной деятельности приложения, ядро адаптивной разгрузки показано на рис. 1.

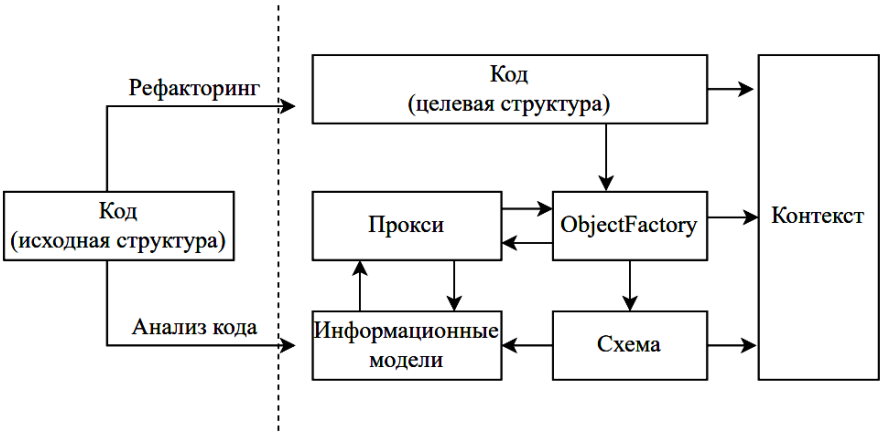

Рис. 1. Ядро адаптивной разгрузки

Существует множество факторов, которые могут повлиять на план развертывания объекта: например, вызовы методов, удаленные серверы и сетевые подключения (таблица 1).

Таблица 1 – Список факторов, которые могут повлиять на план размещения объекта

Символ	Описание
C	Набор вызовов методов для подвижного класса c_i
$Task_i$	Вызов k-го метода для основного подвижного класса c_i
t_i^k	Метод для основного подвижного класса c_i
N	Набор устройств, облачных и мобильных границ
n_i	Сервер
V	Скорость передачи данных между устройством и удаленными серверами
v_i	Скорость передачи данных между устройством и удаленным сервером n_i
RTT	Время прохождения туда и обратно между устройством и удаленными серверами
rtt_i	Время прохождения туда и обратно между устройством и удаленным сервером n_i

DOI: 10.31854/2307-1303-2023-11-1-1-15
URL: https://www.sut.ru/doci/nauka/1AEA/ITT/2023_1/01-15.pdf

При данном подходе используется ASM4 (универсальная платформа для обработки и анализа байт-кода Java) в целях создания графа потока управления главного модуля и извлечения основных перемещаемых классов и вызовов их методов.

Время отклика для каждого вызова метода состоит из времени выполнения и сетевой задержки, как показано ниже:

$$T_{response} = T_e + T_d.$$

Сетевую задержку вызова метода t_i^k на сервере n_t можно рассчитать по формуле:

$$T_d\left(t_i^k, n_t\right) = \frac{D(t_i^k)}{V_{n_t}} + rtt_{n_t}.$$

Время отклика на вызов метода t_i^k на сервере n_t можно рассчитать как:

$$T_{response}\left(t_i^k, n_t\right) = T_e\left(t_i^k, n_t\right) + T_d(t_i^k, n_t).$$

На основе моделей строится функция приспособленности для оценки схемы разгрузки, т. е. плана развертывания объекта для каждого основного класса, как указано ниже:

$$T_{response}(c_i, n_t) = \sum_{k=1}^{m} (T_{response}(t_i^k, n_t))$$

.

Кроме того, более эффективная схема получает меньшее значение функции приспособленности. Таким образом, можно определить оптимальное место развертывания для каждого класса в основной активности приложения.

Результаты моделирования

Оценивается повышение производительности на основе сравнения приложений традиционной разгрузки с приложениями адаптивной разгрузки. Исходное приложение полностью работает на устройстве. Традиционное разгруженное приложение разгружает подвижные объекты с интенсивными вычислениями на удаленный сервер с самым передовым сетевым подключением. Адаптивное разгруженное приложение может использовать любой удаленный сервер для разгрузки по требованию. Также рассматривается сценарий, когда устройства перемещаются между местоположениями. Модель изменения положения мобильных устройств представляет собой «переход из сада в детскую площадку, затем в учебный корпус и, наконец, в лабораторный корпус». Время отклика используется в качестве показателя производительности.

На рис. 2 показано сравнение производительности при работе License Plate Recognition System (LPRS) – системы распознавания номерных знаков на Honor MYA-AL10, когда происходит переключение между четырьмя местоположениями, оставаясь в каждом из них в течение 1 минуты. Традиционное выгруженное

приложение, скорее всего, выйдет из строя, когда устройство перейдет в новое место. Например, приложение не запускается в течение 20 с, когда устройство входит на детскую площадку. Причина сбоя приложения заключается в том, что при изменении контекста устройства объекты, которые были выгружены на исходный мобильный край, больше не могут быть доступны с устройства.

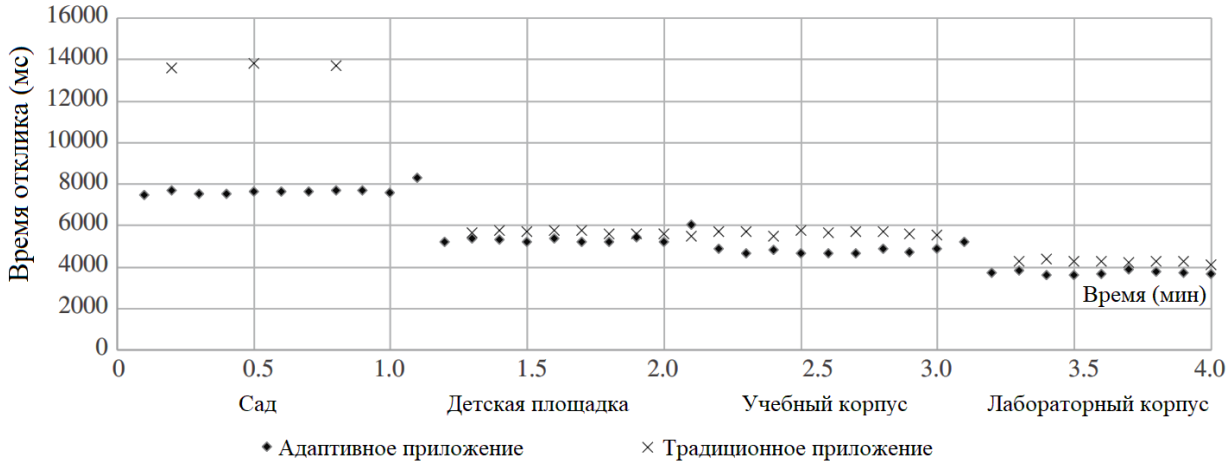

Рис. 2. Сравнение производительности при работе системы распознавания номерных знаков на устройстве Honor MYA-AL10

Рассматриваемое адаптивное разгруженное приложение остается доступным при изменении контекста устройства, хотя время отклика немного увеличивается, когда устройство переходит в новое место. При изменении контекста устройства рассматриваемый фреймворк может автоматически определить схему разгрузки, а затем за короткое время разгрузить вычисления. Таким образом, в этом разделе была рассмотрена адаптивная структура разгрузки для приложений Android в MEC. Для указанного Android-приложения платформа сначала рефакторит приложение для реализации специального шаблона проектирования, а затем анализирует статический код приложения для получения информации о перемещаемых классах. Он может определять схему разгрузки во время выполнения в соответствии с контекстом устройства и обеспечивать динамическую разгрузку приложений между устройством, серверами MEC и облаком. Результаты показывают, что данный подход может значительно повысить производительность и снизить потребление энергии.

COSTA

Предлагается итерационный алгоритм, основанный на методах локального поиска, который имеет название COSTA [5] (постоянно снижает общую стоимость по сравнению с базовыми алгоритмами за счет повторного выполнения одного из предопределенных локальных шагов). COSTA завершается за полиномиальное время и достигает локального оптимального значения, не превышающего постоянное отношение аппроксимации с точки зрения глобального оптимума.

Системная модель

На рис. 3 изображена беспроводная сеть с поддержкой MEC, в которой MEC-enabled Base Stations (MBS) – базовые станции с поддержкой MEC подключены к облаку, в котором хранятся все доступные услуги. Каждая MBS хранит подмножество услуг и может выполнять соответствующие задачи от ближайших пользователей.

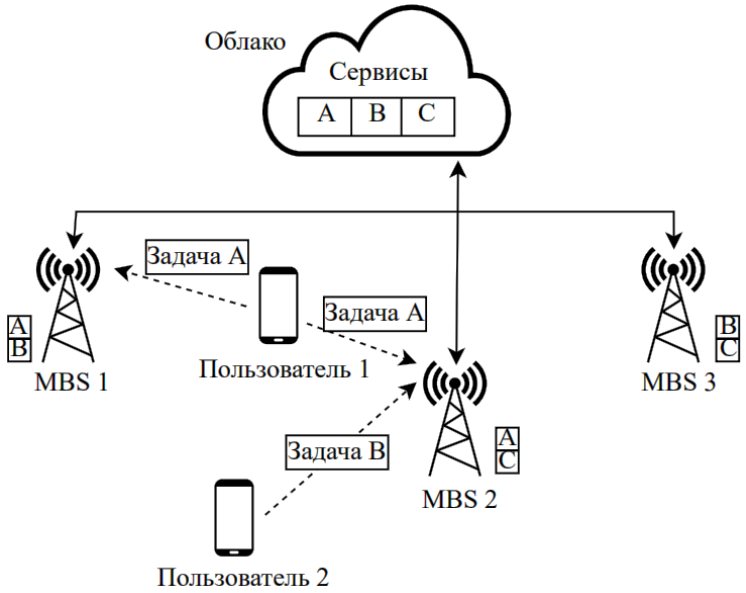

Рис. 3. Иллюстрация беспроводной сети с поддержкой MEC для метода COSTA

Рассматривается многосотовая сеть беспроводного доступа с поддержкой MEC, где существует набор S MBS, обозначенный как $S = \{1, 2, \dots, S\}$. Каждая MBS $s \in S$ оснащена небольшим пограничным сервером для предоставления услуг по разгрузке вычислений мобильным пользователям с ограниченными ресурсами, таким как смартфоны, планшеты и носимые устройства. Как правило, каждый сервер MEC может быть либо физическим сервером, либо виртуальной машиной с умеренными вычислительными возможностями, предоставляемыми сетевым оператором. MBS могут обмениваться данными с облачными и мобильными пользователями через транзитные каналы и каналы беспроводного доступа, соответственно. Система работает в дискретных временных интервалах, каждый из которых имеет продолжительность, соответствующую временной шкале, в которой могут обновляться кэширование услуги и решения о назначении задач.

Кэширование услуги [7], [8]

Рассматривается набор из M услуг, которые могут быть запрошены пользователями, обозначается как $M = \{1, 2, \dots, M\}$, и предположим, что в удаленном облаке могут размещаться все услуги в M.

Определяем переменные кэширования бинарных услуг как $x_{m,s} \in \{0,1\}$, где $x_{m,s} = 1$, если сервис m кэшируется в MBS s, и $x_{m,s} = 0$ в противном случае. Обозначаем $X \triangleq \{x_{m,s} | m \in M, s \in S, x_{m,s} = 1\}$ как общее решение о кэшировании услуги. Когда запрос на выгрузку, соответствующий услуге m, поступает в

DOI: 10.31854/2307-1303-2023-11-1-1-15
URL: https://www.sut.ru/doci/nauka/1AEA/ITT/2023_1/01-15.pdf

MBS s, если s кэширует услугу m и имеет достаточную вычислительную мощность, она может выполнить выгруженную задачу и вернуть результат пользователю. Определяем стоимость кэширования услуги m в MBS s как $c_{m,s} \geq 0$, которая статически определяется поставщиком сети/услуги.

Общая стоимость кэширования в системе:

$$C_{cache}(X) = \sum_{s \epsilon S} \sum_{m \epsilon M} x_{m,s} c_{m,s}.$$

Разгрузка задач

Набор мобильных пользователей, прибывающих в текущий временной интервал, обозначается как $N = \{1, 2, \dots, N\}$, в котором у каждого пользователя $i \in N$ есть задача с интенсивными вычислениями с тем же индексом, который он хочет разгрузить в MBS. Размер входных данных и потребность в вычислениях пользователя i равны b_i (бит) и f_i (цикл ЦП/с), соответственно, где цикл ЦП – цикл центрального процессора.

Пусть $y_{i,s} \in [0, 1]$ – непрерывная переменная решения, обозначающая долю задач от пользователя i, которая выгружается в MBS s, тогда считаем, что:

$$\sum_{s \in S} y_{i,s} = 1, \forall_i \in \mathrm{N}.$$

Количество входных данных и потребность в вычислениях, выгружаемых в MBS s от пользователя i, составляют $y_{i,s} b_i$ и $y_{i,s} f_i$, соответственно. Чтобы учесть ограниченные вычислительные ресурсы в MBS, считаем, что каждая MBS имеет максимальную вычислительную мощность F_s. Следовательно, ограничение вычислительной мощности может быть выражено как:

$$\sum_{i \epsilon N} y_{i,s} f_i \leq F_s, \forall_s \epsilon S$$

Результаты моделирования

Оценим преимущества алгоритма кэширования услуг и назначения задач (COSTA) путем сравнения его экономической эффективности с другими конкурирующими схемами в различных системных конфигурациях.

Данные трассировки. Был использован набор данных из кластера Google для извлечения информации о поступлении запросов на обслуживание и их расчетный запрос. Этот набор данных регистрирует более 3 миллионов поступлений запросов, каждое из которых записывается как запись с информацией о времени поступления, идентификаторе задачи, идентификаторе задания и потреблении центрального процессора (ЦП), в нормированных дробных единицах.

1. Влияние количества запросов на разгрузку

Оценивается производительность пяти схем при изменении количества запросов на разгрузку на временной интервал. Для каждого временного интервала из исходной трассировки случайным образом выбираются N запросов. Из рис. 4а, где рассматривается общая стоимость в зависимости от количества запросов, при F_s = 0,1, видно, что общая стоимость COSTA всегда очень близка к стоимости схемы OPT-CP (базовый уровень).

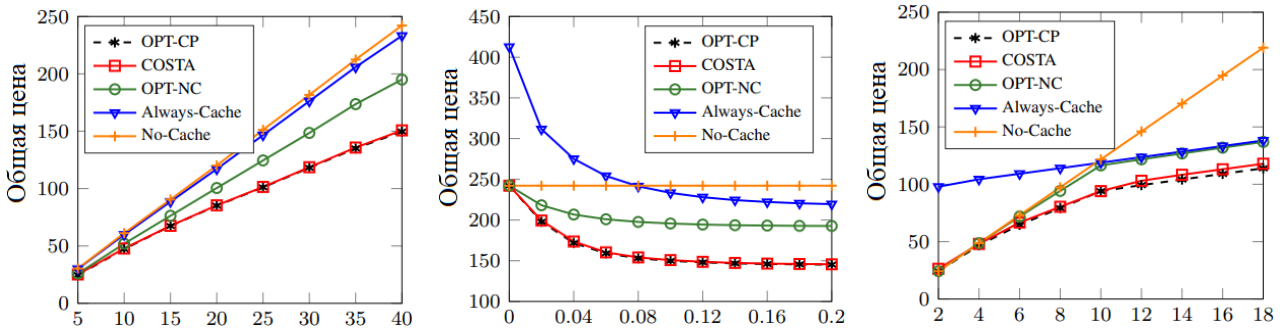

Рис. 4. Влияние различных параметров на общую стоимость пяти рассмотренных схем: а) количество запросов (N); б) вычислительная мощность (F_s); в) размер задачи в MB (b)

С другой стороны, общая стоимость COSTA значительно ниже, чем у трех других схем, и пробелы растут с увеличением количества запросов.

2. Влияние вычислительной мощности

Результаты на рис. 4б, где рассматривается общая стоимость в зависимости от вычислительной мощности при $N = 40$, только для однородной системы, где вычислительные мощности на разных MBS одинаковы. Изменяя вычислительную мощность F_s от 0 до 0,2, видно, что общая стоимость схемы No-Cache не изменяется, так как она не использует обработку в MBS. Затраты остальных четырех схем уменьшаются по мере увеличения F_s и в конечном итоге приближаются к своим соответствующим постоянным значениям, когда F_s очень высока. Предложенная схема COSTA всегда работает очень близко к схеме OPT-CP, но значительно превосходит схемы OPT-NC, Always-Cache и No-Cache.

3. Влияние размера входных данных задачи.

На рис. 4в, где рассматривается общая цена от размера задачи при $F_s = 0{,}1$ и $N = 20$, все задачи имеют одинаковый размер $b_i = b, \forall i \in \mathrm{N}$, и это значение варьируется от 2 до 18 МБ. Очевидно, что общие затраты всех схем увеличиваются с увеличением b. При больших размерах задач общая стоимость COSTA немного выше, чем у OPT-CP, но значительно ниже, чем у трех других схем.

Был рассмотрен итерационный алгоритм низкой сложности COSTA [5], основанный на методах локального поиска. COSTA завершается за полиномиальное время и дает локально оптимальное решение со стоимостью не более постоянного коэффициента аппроксимации по сравнению с оптимумом. По результатам моделирования, с трассировкой рабочей нагрузки из кластера Google, видно, что предложенный алгоритм COSTA обеспечивает небольшой разрыв в оптимальности при значительном снижении стоимости разгрузки по сравнению с конкурирующими схемами.

Линейная регрессия

В данном разделе для разгрузки задач предлагается метод линейной регрессии (с использованием прогнозирования по принципу машинного обучения на основе ожидаемой продолжительности обработки каждой задачи на серверах MEC). Кроме того, на каждом облачном сервере создается система мониторинга

DOI: 10.31854/2307-1303-2023-11-1-1-15
URL: https://www.sut.ru/doci/nauka/1AEA/ITT/2023_1/01-15.pdf

ресурсов. Разгрузка нескольких подзадач на несколько граничных узлов может снизить нагрузку на каждый узел, а одно большое приложение или задача могут быть обработаны быстрее, чем когда они обрабатываются на одном узле.

Системная модель

На рис. 5 показана системная модель данного подхода, состоящая из одного ядра облака, четырех узлов MEC и одного мобильного устройства. Пользователь может перенести свою задачу на несколько облачных узлов, включая серверы MEC и основное облако. Одно приложение или задание можно разделить на несколько подзадач, и в соответствии с их приоритетом или уровнем QoS каждая подзадача может быть выгружена на соответствующий узел MEC. Например, если на мобильном устройстве есть приложение, которое необходимо обработать в T_{limit}, и это приложение имеет четыре доступных для разгрузки подзадачи $\{task_1, task_2, task_3, task_4\}$. Среди этих подзадач может быть корреляция в отношении приоритета. Например, когда $task_2$ принимает результат $task_1$ в качестве входных данных, приоритет $task_1$ выше, чем приоритет $task_2$. Напротив, если $task_4$ не имеет отношения к другим подзадачам, ее можно обрабатывать во время обработки других задач. Т. е. $task_4$ и $task_1$ могут выполняться параллельно. В этом случае приоритет $task_4$ низкий.

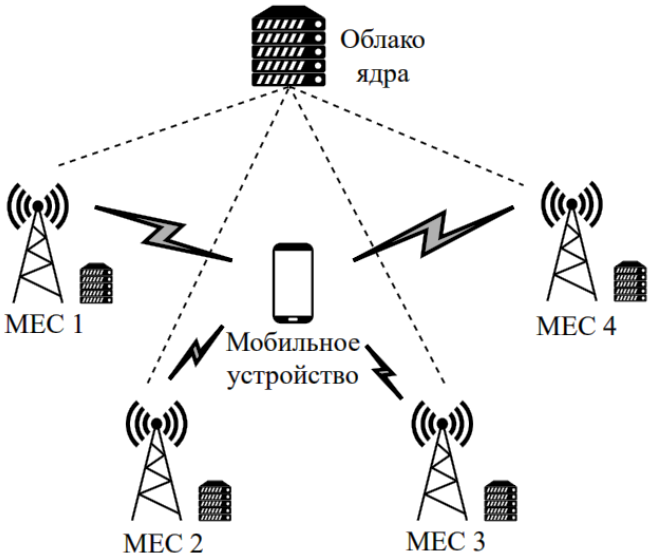

Рис. 5. Системная модель

После определения приоритета между каждой подзадачей на основе прогнозирования времени обработки каждой задачи на каждом сервере MEC можно перенести эти задачи на соответствующий сервер MEC. После выгрузки в соответствующий сервер для выполнения всей задачи каждый сервер должен общаться друг с другом.

Линейная регрессия на основе глубокого обучения

Чтобы предсказать продолжительность обработки задачи, используется линейная регрессия на основе глубокого обучения. Модель обучается линейной

регрессии при помощи трассировки Google-cluster data-2011-2, которая представляет 29-дневный период, содержащий информацию с мая 2011 г., в кластере из примерно 12,5 тыс. облачных узлов. Для применения модели данные (представлены в таблице 2) должны быть предварительно обработаны.

Таблица 2 – Пример используемых данных для обучения модели

T_CPU	T_Mem	T_Disk	M_CPU	M_Mem	M_Disk	Время
0,125	0,07446	0,00042	0,5	0,4995	12 %	7,4
0,03125	0,08691	0,000455	0,25	0,2493	52 %	10,4
0,1875	0,02141	0,000325	0,5	0,749	44 %	3,1
0,03748	0,01605	0,0051	0,46	1	11 %	4,3
0,2428	0,02811	0,00058	1	0,749	12 %	2,3
0,263	0,05215	0,000214	0,45	0,2498	84 %	15,1

Условные единицы: CPU – количество ядер, Memory – байты, Disk Space – байты. Максимальная емкость ресурса – 1,0.

M_CPU, M_Mem, M_Disk представляют текущее состояние ресурсов сервера MEC для задач. Время – это ожидаемая продолжительность выполнения в текущем состоянии узла. С помощью этих данных нейронная сеть была обучена и построена модель, которая принимает шесть входных данных (T_CPU, T_Mem, T_Disk, M_CPU, M_Mem, M_Disk).

Применяется функция потерь MSE (*аббр. от англ.* Mean Squared Error) для измерения и минимизации ошибки модели:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\widehat{Y_l} - Y_i)^2,$$

где $\widehat{Y_l}, Y_i$ – обозначают прогнозируемое и реальное значения.

Разгрузка подзадач на основе прогноза

Основываясь на ожидаемом времени выполнения на каждом граничном узле, можно перенести задачу на сервер MEC. Для прогнозирования сначала необходима информация о каждой задаче, включая требуемые ресурсы. Можно легко получить это, используя профилирование кода. Также необходимо текущее состояние ресурсов каждого узла MEC. Для этого разрабатывается модуль мониторинга ресурсов в основном облаке с использованием Grafana [9] и Influx DB [10]. Архитектура системы мониторинга изображена на рис. 6.

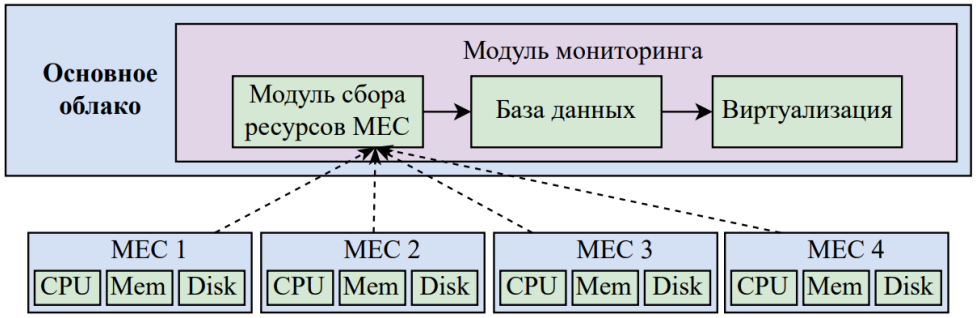

Рис. 6. Архитектура системы мониторинга

Каждые 3 секунды узлы MEC отправляют статус своего ресурса в основное облако, где полученная информация хранится в базе данных. Модуль визуализации отображает обновляющиеся в режиме реального времени данные, извлеченные из базы. Для прогнозирования времени выполнения задачи уже имеется описание задачи и статус ресурса MEC. Наконец, прогнозирование выполняется в основном облаке. Запрос на разгрузку с мобильного устройства также поступает в основное облако. Сообщение запроса содержит описание каждой подзадачи.

Результаты моделирования

При моделировании используется следующая топология: одно ядро облака, четыре сервера MEC и одно мобильное устройство. Базовое облако и серверы MEC подключены через проводное соединение AP. На сервере MEC установлены hostapd, благодаря которому Raspberry Pi работает как точка доступа Wi-Fi. Для тестирования используется алгоритм распознавания лиц [11], написанный на java. Разработанные авторами компоненты, включая модуль прогнозирования, модуль мониторинга и предлагаемый алгоритм, написаны на Python 3.5.

Результаты показывают, что приложение обрабатывается быстрее, когда раздел приложения разгружается на несколько серверов MEC, чем на один сервер. Также на рис. 7 показано использование ресурсов на каждом узле MEC. Синяя линия обозначает использование ресурсов, когда вся задача выгружается на один сервер MEC, а оранжевая линия – состояние ресурсов при использовании предложенной схемы.

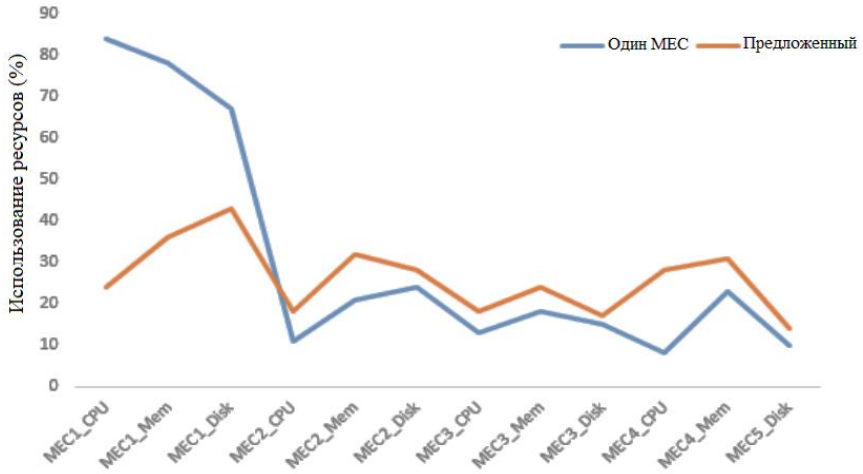

Рис. 7. Использование ресурсов

Распределенные ресурсы используются гораздо более равномерно. По результатам испытаний на модуле мониторинга в основном облаке наблюдается потребность в ресурсах для каждого сервера MEC в режиме реального времени. Вся информация о ресурсах отправляется с каждого сервера MEC каждые 36 с в основное облако. Основное облако хранит и информацию в базе данных, а затем по запросу извлекает ее. Это позволяет получать самые последние данные из каждой таблицы. Графики обновляются каждые 5 с.

Таким образом, в данном разделе была рассмотрена разгрузка подзадач на основе прогнозирования на несколько серверов MEC. Прогнозируется ожидаемое время обработки с помощью линейной регрессии. Затем с этим результатом сравнивается ожидаемое время выполнения, разгружая подзадачи на несколько серверов MEC по приоритету задачи. Если подзадачи могут обрабатываться независимо, то подзадачи этой задачи могут обрабатываться параллельно. Предлагаемая схема позволяет сократить среднее время выполнения приложения на каждом сервере MEC. Кроме того, с данной схемой повышается эффективность использования ресурсов. Однако в этой работе не учитывалась реальная ситуация в глобальной сети и задержка в очереди на серверах MEC.

Сравнение методов

В таблице 3 представлено сравнение вышеперечисленных методов [3, 5, 6].

Таблица 3 – Сравнение методов разгрузки задач

Характеристики	Метод		
	Адаптивная структура	COSTA	Линейная регрессия
Сценарий	Несколько устройств; многозадачный; многоузловой	Многозадачный; несколько базовых станций	1 пользователь; Многозадачный; многоузловой
Необходимая информация	Прокси; текущее местоположение устройства	Входные данные задачи	Входные данные задачи
Цель	Для поддержки мобильного приложения с возможностями разгрузки	Генерировать решение о разгрузке с учетом кэширования услуги	Принять решение о разгрузке на основе прогноза
Ограничения	Используется только для объектно-ориентированных приложений	Приведено единственное локально оптимальное решение	Используется только для сети LAN
Практическое применение	+	–	+

Адаптивная структура принимает различные сетевые ситуации и, соответственно, принимает решение о разгрузке. Алгоритм COSTA решает совместную проблему кэширования и разгрузки. Линейная регрессия используется для разгрузки на основе прогнозирования в MEC. Исходя из данных, приведенных в таблице 3, можно сделать вывод о том, что сценарии всех методов схожи, однако цели указывают на то, что каждый метод имеет различные пути разгрузки задач и, соответственно, разные алгоритмы расчета – от вычислений с применением сложных математических формул до глубокого обучения.

Таким образом, для метода адаптивной структуры производится разгрузка мобильных приложений, для метода COSTA – учет кэширования для генерации решения о разгрузки, а для метод линейной регрессии – принятие решения о разгрузке происходит на основе прогноза. То есть первый метод работает в связке с

конкретными приложениями, второй метод направлен экспериментально на учет кэширования, третий опирается на глубокое обучение.

Сравнение данных методов является трудной задачей, так как их объединяет общая цель, но пути и подцели разнятся. В связи с этим, для того, чтобы выбрать наиболее адекватный метод, можно ориентироваться на актуальные запросы в области MEC и телекоммуникаций, а именно: направленность на клиента, вопрос цены, практическое применение, дальнейшее развитие метода. Среди всех методов под это описание подходит COSTA, так как он не ограничен в применении для структуры сети, а также имеет хорошую математическую основу и не является направленным на определенное приложение или ПО.

Заключение

В данной работе были рассмотрены три метода для решения задачи разгрузки: адаптивная структура, COSTA, линейная регрессия.

Основная цель адаптивной структуры – поддержка мобильного приложения с возможностями разгрузки. Этот метод используется только для объектно-ориентированных приложений.

Цель схемы COSTA состоит в том, чтобы сгенерировать решение о разгрузке с учетом кэширования услуги.

Линейная регрессия – используется только для сети LAN.

В целях выбора наиболее адекватного метода для решения заявленной проблемы был выполнен анализ научных работ на тему разгрузки задач в граничных вычислениях со множественным доступом. Произведено сравнение представленных методов по пяти критериям: год выпуска, сценарий, необходимая информация, цель, ограничения, практическое применение.

В ходе сравнительного анализа публикаций были выявлены достоинства и недостатки рассмотренных методов. Что дало основание сделать следующее предположение: адекватное решение может быть найдено при использовании метода COSTA.

Литература

1. Cisco Networking Team. Global Mobile Data Traffic Forecast Update 2016-2021. White Paper // Cisco Visual Networking Index. 2017.

2. Liyanage M., Porambage P., Ding A. Y., Kalla A. Driving forces for Multi-Access Edge Computing (MEC) IoT integration in 5G // ICT Express. 2021. Vol. 7. Iss. 2. PP. 127–137. DOI: 10.1016/j.icte.2021.05.007

3. Chen X., Chen S., Ma Y., Liu B., Zhang Y., et al. An adaptive offloading framework for Android applications in mobile edge computing // Science China Information Sciences. 2019. Vol. 2. Iss. 8. DOI: 10.1007/s11432-018-9749-8

4. Vhora F., Gandhi J. A Comprehensive Survey on Mobile Edge Computing: Challenges, Tools, Applications // 2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC). 2020. DOI: 10.1109/ICCMC48092.2020.ICCMC-0009

5. Tran T. X., Chan K., Pompili D. COSTA: Cost-aware Service Caching and Task Offloading Assignment in Mobile-Edge Computing // 2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON). 2019. DOI: 10.1109/SAHCN.2019.8824854

6. Kim K., Lynskey J., Kang S., Hong C. S. Prediction Based Sub-Task Offloading in Mobile Edge Computing // 2019 International Conference on Information Networking (ICOIN). 2019. DOI: 10.1109/ICOIN.2019.8718183

7. Xu J., Chen L., Zhou P. Joint Service Caching and Task Offloading for Mobile Edge Computing in Dense Networks // IEEE INFOCOM 2018. IEEE Conference on Computer Communications. 2018. PP. 207–215. DOI: 10.1109/INFOCOM.2018.8485977

8. Chen L., Shen C., Zhou P., Xu J. Collaborative Service Placement for Edge Computing in Dense Small Cell Networks // IEEE Transactions on Mobile Computing. 2019. Vol. 20. Iss. 2. PP. 377–390. DOI: 10.1109/TMC.2019.2945956

9. Grafana: the open observability platform // Grafana Labs. URL: https://grafana.com (дата обращения 14.04.2023)

10. Influx DB Times Series Data Platform // Influx Data. URL: https://www.influxdata.com (дата обращения 14.04.2023)

11. Eigenface face recognition algorithm // darnok.org. URL: https://darnok.org/projects/face-recognition (дата обращения 14.04.2023)

Статья поступила 10 мая 2023 г.
Одобрена после рецензирования 19.06.2023 г.
Принята к публикации 05 июля 2023 г.

Информация об авторах

Чипсанова Елена Валерьевна – магистр кафедры инфокоммуникационных систем Санкт-Петербургского государственного университета телекоммуникаций им. проф. М. А. Бонч-Бруевича. E-mail: lenchip@mail.ru

Елагин Василий Сергеевич – кандидат технических наук, доцент кафедры инфокоммуникационных систем Санкт-Петербургского государственного университета телекоммуникаций им. проф. М. А. Бонч-Бруевича. E-mail: v.elagin@sut.ru

Task Offloading Methods in Multi-Access Edge Computing Systems

E. Chipsanova✉, V. Elagin
The Bonch-Bruevich Saint-Petersburg State University of Telecommunications,
St. Petersburg, 193232, Russian Federation

Problem definition. *Multi-access edge computing is a promising solution that can solve capacity and performance issues in legacy systems such as mobile cloud computing. Problems include core network congestion, high latency, poor quality of service, high utility cost of resources such as bandwidth and energy. The above problems arise due to the limited resources of mobile devices, when connecting with multiple hops between end users and the cloud, high load from resource-intensive and delay-critical applications.* ***Methods used:*** *comparative evaluation method.* ***Novelty:*** *Methods of task offloading in boundary computing systems with multiple access are analyzed. Three methods of task offloading for comparative analysis are proposed.* ***Result****: familiarization with the existing methods of task offloading, as well as identification of the most relevant from the presented ones.* ***Practical significance****: in the proposal of the most relevant method in the issue of task offloading in networks of boundary computation with multiple access.*

Keywords: *task offloading, multi-access edge computing, fifth generation network, adaptive structure, COSTA, linear regression*

Funding: *The scientific article was prepared in the framework of applied scientific research of St. Petersburg State University of Telecommunications, registration number 123060900012-6 in USIS R&D*

Information about Authors

Elena Chipsanova – Postgraduate Students of the Department of Infocommunication Systems (The Bonch-Bruevich Saint-Petersburg State University of Telecommunications). E-mail: lenchip@mail.ru

Vasiliy Elagin – Ph.D. of Engineering Sciences, Associate Professor of the Department of Infocommunication Systems (The Bonch-Bruevich Saint-Petersburg State University of Telecommunications). E-mail: v.elagin@sut.ru