УДК 519.68
DOI: 10.15827/0236-235X.141.107-114
Дата подачи статьи: 17.08.22
2023. Т. 36. № 1. С. 107–114

# *Разработка алгоритма применения распределенных вычислительных ресурсов на основе принципов Edge-вычислений*

*А.М. Воробьев* [1], *старший преподаватель, a.m.vorobev@utmn.ru*
*М.С. Воробьева* [1], *к.т.н., доцент, m.s.vorobeva@utmn.ru*
*Ю.В. Боганюк* [1], *аспирант, y.v.boganyuk@utmn.ru*

[1] *Тюменский государственный университет, г. Тюмень, 625003, Россия*

В статье рассматриваются вопросы реализации алгоритма распределения вычислительных задач по множеству распределенных вычислительных ресурсов с последующей агрегацией результатов. Данный алгоритм является ключевым в рамках проекта реализации центра обработки данных на принципах экономики совместного потребления. Прототип механизма реализован на языке Python 3.8 с применением СУБД PostgreSQL 14, система передачи сообщений – на базе RabbitMQ 3.9. В качестве платформы вычислительных узлов выступает ОС CentOS 8 Stream.

Цель работы заключается в реализации масштабируемого механизма выполнения распределенных вычислений для применения в качестве основного средства распределения задач и агрегации результатов в рамках исследуемого облика центра обработки данных на принципах экономики общественного потребления.

Предметом исследования являются методы резервирования и применения вычислительных мощностей, а также агрегации результатов работы программных алгоритмов.

Предложенный механизм решает задачу распределения вычислений с последующей агрегацией результатов среди вычислительных узлов с различными техническими характеристиками. Реализуется интерфейс, пригодный для интеграции в клиентские информационные системы как средство выгрузки вычислений с доступом в формате REST API-шлюза.

Теоретическая значимость работы заключается в комбинировании существующих принципов и идей Edge-вычислений для решения иного класса задач, где проблемой является недостаток вычислительного ресурса для задач информационной системы, а не характеристик имеющейся модели.

Практическая значимость состоит в разработке прикладного инструмента применения внешних вычислительных мощностей для решения широкого класса клиентских задач. Это открывает возможность организации коммерческого взаимодействия владельцев неиспользуемых вычислительных ресурсов и владельцев информационных систем, испытывающих недостаток вычислительных мощностей.

***Ключевые слова:*** *распределенные вычисления, шина обмена сообщениями, AMQP, асинхронные вызовы, виртуальная машина, REST API, выгрузка вычислений*.

Отечественная отрасль информационных технологий в 2022 году столкнулась с резким недостатком вычислительных мощностей, что поставило под угрозу экстенсивные компоненты планов развития профильных компаний и государственных учреждений. Основная причина возникновения перебоев с поставками – комплекс прямых и непрямых санкционных ограничений, введенных различными государствами, и, как следствие, политика самоограничения поставщиков. Также следует отметить миграционные процессы на отечественном ИТ-рынке и определенные риски при использовании услуг зарубежных хостинг-провайдеров. Многие отечественные информационные системы были переведены на территорию РФ, создав дополнительную нагрузку на местных хостинг-провайдеров.

Тем не менее немало владельцев информационных систем с собственными вычислительными мощностями не почувствовали разницы в характере загрузки своего оборудования, их бизнес-процессы не претерпели больших изменений. Такие служебные информационные системы часто обладают крупным резервированием вплоть до простаивающих вычислительных ресурсов, и применение их для удовлетворения нужд бизнеса, испытывающего ресурсный голод, может стать эффективным коммерциализируемым решением возникающих проблем.

Примечательны параллели, которые можно провести при анализе данной проблемы, с историей возникновения и развития Edge-вычислений, решающих проблему недоступности использования вычислительного ресурса с ми-

нимальной задержкой на передачу информации (при условии, что сам ресурс практически неисчерпаем) [1]. Аналогично прослеживается проблема недостатка ресурса с применением уже наработанных технологий и решений Edge-вычислений.

Концепция поиска и применения дополнительных ресурсов в сфере информационных технологий далеко не нова, вопросы эффективного управления неким изначально непрофильным ресурсом для выполнения различных задач с целью разгрузки главных исполнительных узлов решаются достаточно давно.

Например, применение процессорных мощностей видеокарты компьютера для проведения расчетов, не связанных с графикой (кодирование мультимедиа, майнинг криптовалюты или вычисление в рамках концепции Интернета вещей), описано авторами в статье [2]. Аналогично после повсеместного распространения облачных вычислений возник дефицит быстродоступной среды вычисления для задач, чувствительных к задержке. Несмотря на практически безграничный вычислительный ресурс облака, затраты на перемещение информации от места применения до места вычисления существенно снижают эффективность и работоспособность информационной системы. Само становление пограничных вычислений связано с дефицитом такого ресурса [3], что привело к реализации различных подходов для разрешения возникающих проблем: применение мобильных сотовых станций как узлов вычисления [4], применение вычислительных мощностей сетевого оборудования как среды для выполнения задач [5] и т.п.

В данной работе для решения проблемы дефицита вычислительных мощностей применены основные принципы, подходы и технические паттерны Edge-вычислений.

## Постановка задачи

Ключевые аспекты исследования изложены в [6] и сконцентрированы вокруг применения принципов экономики совместного потребления в данной предметной области с поиском потенциала коммерциализации. Был разработан технический прототип, реализующий одну из моделей применения ресурса на основе классических облачных концепций, комбинации решений подходов «инфраструктура как услуга» и «платформа как услуга».

В данной работе внимание фокусируется на разработке алгоритма применения распределенных вычислительных ресурсов на основе принципов Edge-вычислений, в том числе и для устранения потенциальных недостатков, выявленных в классическом подходе.

Алгоритм имеет фундаментальное значение для модели с точки зрения разработанной механики отправки задач и получения результатов: задается облик вычислительной нагрузки, пригодной для исполнения предложенным способом, а также определяется набор пользовательских задач, которые могут быть эффективно решены в рамках моделей центров обработки данных, построенных на принципах экономики совместного потребления.

В рамках исследования вместо традиционной идентификации задачи, поиска ресурса для решения задачи и разработки метода его применения будет проведена идентификация всех возможных свободных ресурсов в условиях потенциально нарастающего дефицита с последующей разработкой метода их эффективного использования. В итоге определится спектр решаемых разработанным способом задач с учетом всех выявленных ограничений, вызванных как природой ресурса (характер использования, надежность, владение, риски информационной безопасности и т.п.), так и предложенным механизмом его применения.

В определенном смысле повторяется ситуация с применением GPU для решения задач общего назначения. Идея применить специфичный вычислительный ресурс, зачастую простаивающий, породила целый стек технологий программирования вычислений (OpenCL, NVIDIA CUDA и др.) [7]. Отдельным полезным навыком применения подобных нестандартных ресурсов является идентификация задач, которые потенциально могут быть эффективно решены с применением GPU-ресурса. Его характеристики и особенности вычислений диктуют ограничения, обязательно учитываемые при планировании выполнения задач.

## Резервирование ресурса

Выделим необходимые вычислительные мощности привлекаемого ресурса:

– аппаратная часть представлена актуальным процессором архитектуры x86-64 с количеством vCPU не менее 6;

– резерв оперативной памяти не менее 2–4 Гб;

– хранение данных реализуется носителями с невысокой скоростью операций (HDD, недорогие SSD);

– современная *операционная система* (ОС) является представителем семейства Windows или Linux.

Дополнительные сведения о потенциальном ресурсе можно получить, только заведомо ограничив выборку, директивно требуя наличия определенных характеристик. Таким образом, необходимо решить задачу резервирования вычислительного ресурса и управления им в условиях крайней вариативности платформ.

Естественной является концепция виртуальных машин, которая позволяет реализовать эффективную изоляцию и единообразие рабочих окружений для запуска пользовательских полезных нагрузок, решая проблему менеджмента разнородного множества источников ресурсов.

Более современным вариантом решения проблемы является контейнер как средство изоляции и менеджмента ресурсов. Принципиально и контейнеры, и виртуальные машины предназначены для решения схожих задач, однако в данном контексте классические виртуальные машины предпочтительнее:

– традиционно контейнеры реализуются на технологических решениях ядра Linux (изоляция ресурсов, формирование пространств имен и др.), поддержка прочими ОС может не быть гарантированной, вносить дополнительные сложности в такие вопросы, как доступ к ресурсам основной ОС и т.п.;

– виртуальные машины не имеют никаких ограничений на применение внутренних технологий уровня ядра, контейнеры же обладают общим разделяемым с системой хоста ядерным пространством, поэтому применение технологий, требующих загрузки модулей ядра, ограничено.

С учетом принятой концепции можно вывести следующие требования к привлекаемому ресурсу:

– поддержка технологии виртуализации на уровне процессора и ОС;

– наличие резерва ресурса на обслуживание виртуальной машины – не менее 1 vCPU, не менее 512 Мбайт оперативной памяти, не менее 20 Гб хранилища (данные указаны для ОС CentOS 8 в минимальном варианте инсталляции).

Следует уточнить, что в качестве вычислительных узлов выступают унифицированные предварительно подготовленные виртуальные машины. Это дает возможность проектировать код управления подобными нагрузками непосредственно на местах вычисления – наличие виртуальной машины позволяет использовать широкий спектр технологий для реализации приема задач на исполнение.

## Архитектура разрабатываемого алгоритма

Составные части разрабатываемого алгоритма представлены на схеме (рис. 1).

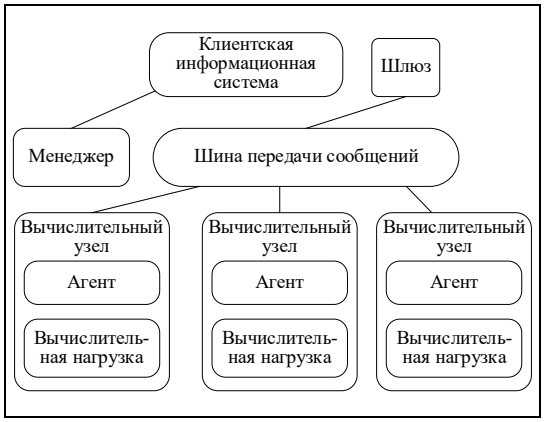


*Рис. 1. Архитектура механизма распределенных вычислений*

*Fig. 1. A distributed calculation mechanism architecture*

Основным компонентом, определяющим облик остальных частей механизма, является шина передачи сообщений. Имеет смысл воспользоваться готовым унифицированным отработанным механизмом передачи сообщений произвольного типа, что позволит осуществлять управление и передачу пользовательских данных через унифицированный канал.

В качестве шины данных был выбран протокол AMQP в реализации RabbitMQ. Это обусловлено следующими факторами:

– протоколы на базе AMQP (в том числе проприетарные) успешно применяются в решениях IoT и Edge-вычислений, имеется наработанная база реализации архитектур информационных систем, схожих с предлагаемой в рамках данного исследования [8];

– производительность аналогичных решений на базе AMQP признана удовлетворительной в сравнении с аналогами, например MQTT (авторы [9] пришли к выводу, что потенциальные проблемы производительности вызваны конкретной реализацией решения и ориентацией на конкретный формат взаимодействия);

– протокол AMQP широко применяется при построении микросервисных систем, по-

тенциально открывая путь к более глубокой интеграции разрабатываемой системы и пользовательских информационных систем;

– в рамках реализации протокола в проекте RabbitMQ наработан существенный опыт построения высокодоступных отказоустойчивых систем обмена сообщениями [10];

– реализованные механизмы изоляции и гибкие механики передачи сообщений позволяют совместно использовать единую шину данных для различных изолированных потребителей для передачи как команд управления, так и пользовательских данных [11].

В рамках реализации алгоритма предлагается организация шины обмена данными на основе нескольких публичных серверов с развернутыми на них экземплярами системы обмена сообщениями RabbitMQ в режиме отказоустойчивой распределенной федерации. Вопросы подобного развертывания неоднократно рассматривались другими авторами (например [12]) и являются технической задачей.

Исходя из принятого решения о применении протокола AMQP можно проанализировать требования к формату сообщений и реализации всех сторон обмена. Формат сообщений AMQP позволяет передавать произвольные данные без каких-либо существенных накладных расходов по типу base64-кодирования.

Таким образом, шина одинаково подходит для передачи управляющих JSON-сообщений и пользовательских сообщений различной формы с произвольным содержанием. Формат JSON-сообщений позволяет выполнять автоматическую конвертацию структур данных многих языков программирования в текстовый формат, пригодный для унифицированной передачи по каналам связи, что расширяет спектр применяемых на всех этапах разработки языков программирования и фреймворков.

Функционал шлюза интерфейса между информационной системой потребителя и системой распределенных вычислений заключается в организации трансляции запросов от компонентов информационной системы потребителя к нагрузкам, развернутым в рамках вычислительной группы. Учитывая выбранный протокол шины передачи данных, задачу можно свести к переупаковке полезного содержимого сообщений клиента в формат AMQP для последующей передачи. В рамках данной работы в качестве протокола взаимодействия был выбран стандартный REST-интерфейс на основе протоколов HTTP/HTTPS. Реализация шлюза как транслятора сводится к следующей программной конструкции:

– docker-образ, содержащий минимально необходимый для работы набор библиотек и используемый для дистрибуции; конфигурация выполняется с помощью переменных окружения, а каждый контейнер порождается под одну вычислительную группу;

– веб-сервер NGINX, осуществляющий прием HTTP-запросов широкого профиля и сохранение максимального числа HTTP-заголовков для передачи на обработку программному транслятору;

– программный транслятор, выполняющий подключение к указанным через переменные окружения виртуальным хостам системы обмена сообщениями RabbitMQ и программную упаковку содержимого полученного REST-запроса, всех HTTP-заголовков и необходимых внутренних метаданных.

Программный агент, развернутый на вычислительном узле, решает две основные задачи:

– получение команд управления (развертывание вычислительной нагрузки, включение в изолированное пространство передачи сообщений, начало и остановка работы и т.п.);

– получение сообщений для передачи полезной нагрузки и агрегации ответов.

Программный агент реализуется в формате Linux-сервиса, развернутого внутри вычислительного узла, с программным кодом на языке Python. В функции агента входит подключение к шине данных, ответственной за управление агентами, с применением предварительно известного публичного адреса точки входа в систему передачи сообщений. Предполагается целостное обновление агентской виртуальной машины для простоты администрирования вычислительных узлов.

Вычислительная нагрузка реализуется в формате Docker-контейнера с пользовательским программным кодом внутри. Нагрузка запускается в рамках окружения виртуальной машины, предварительная подготовка осуществляется средствами программного агента. В данной реализации получение агентами нагрузки для развертывания выполняется с помощью инфраструктуры DockerHub.

Специфичный характер среды исполнения обусловил внесение ограничений на работу программного кода нагрузки. Программный код должен реализовать следующий REST API:

– POST – передача в нагрузку набора задач для исполнения; данные должны быть представлены в виде, пригодном для передачи по HTTP-протоколу;

– GET – запрос результатов обработки данных; данные должны быть предоставлены в виде, пригодном для передачи протоколом

HTTP, допускающим агрегацию ответов методом неупорядоченной (в случайном порядке) конкатенации.

### Реализация операций алгоритма

На основе перечисленных решений каждой отдельной подзадачи можно сформулировать механизм создания вычислительных групп. Итоговая схема инфраструктуры вычислительной группы из двух узлов показана на рисунке 2.

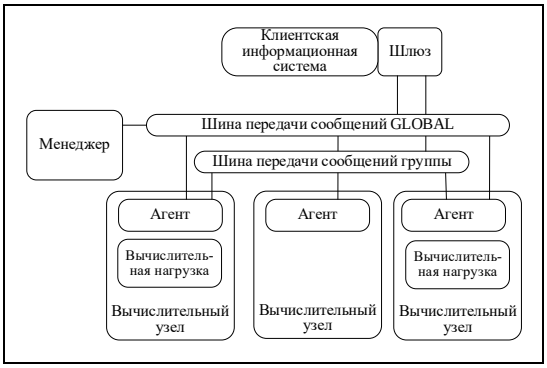


*Рис. 2. Инфраструктура вычислительной группы*

*Fig. 2. A computational group infrastructure*

Основные шаги алгоритма создания вычислительной группы:

– поступает запрос от клиента (пользователя системы) на формирование вычислительной группы, содержащий требования к количеству и характеристикам вычислительных узлов;

– менеджер осуществляет формирование вычислительной группы путем выборки из общего пула доступных вычислительных узлов на основе перечисленных в запросе условий;

– группе присваивается уникальный идентификатор, вычислительные узлы помечаются как занятые;

– менеджер начинает формирование инфраструктуры RabbitMQ для поддержки вычислительной группы: формирование виртуального хоста и соответствующих объектов RabbitMQ;

– в рамках созданного виртуального хоста регистрируются RabbitMQ-пользователи данного шлюза и отобранных агентов; пользователи получают право использовать компоненты виртуального хоста в соответствии со своими ролями;

– вычислительные узлы данной вычислительной группы получают сообщение с информацией для подключения к созданному виртуальному хосту обмена сообщениями, которое содержит наименование и конфигурацию вычислительной нагрузки;

– шлюз данной вычислительной группы получает сообщение с информацией для подключения к созданной инфраструктуре;

– вычислительная группа готова к работе, шлюз готов осуществлять прием HTTP-запросов для трансляции к нагрузкам.

Механизм распределения задач реализуется на основе созданной инфраструктуры очередей. Основные шаги алгоритма распределения:

– программные компоненты клиентской ИС обращаются к развернутому экземпляру контейнера шлюза с применением протокола HTTP/HTTPS;

– веб-сервер, развернутый внутри шлюза, осуществляет обработку HTTP-запроса и передачу содержимого в транслятор, который выполняет перепаковку содержимого тела запроса и всех заголовков в формат AMQP с последующей отправкой сообщения в RabbitMQ;

– шлюз отправляет сообщения в Exchange типа Direct, очередь связи с агентами принимает сообщения;

– в соответствии с выбранным алгоритмом распределения сообщений (по умолчанию Round Robin) агенты получают сообщение из очереди;

– агент реконструирует HTTP-запрос на основе данных, полученных из сообщения, выполняет HTTP-запрос к контейнеру с развернутой нагрузкой и ожидает ответа.

Механизм агрегации результатов реализуется с помощью аналогичных инструментов. Алгоритм агрегации включает следующие шаги:

– программные компоненты клиентской ИС обращаются к развернутому экземпляру контейнера шлюза с применением протокола HTTP/HTTPS;

– веб-сервер шлюза обрабатывает HTTP-запрос;

– шлюз отправляет сообщение, сигнализирующее о необходимости вернуть обработанные данные;

– агенты, получив сообщение, формируют HTTP-вызов к развернутой вычислительной нагрузке и получают обработанные в соответствии с принятыми ограничениями данные;

– агенты осуществляют упаковку полученной информации в пакеты AMQP; сообщения рассылаются через очередь обратной связи, единственным потребителем которой является шлюз;

– шлюз получает сообщения в неопределенном порядке.

## Анализ решения

Система реализует асинхронный механизм постановки задач и агрегации результатов, таким образом, сфера ее применения лежит в сфере задач пакетной обработки данных независимо от их объема. Вычислительные узлы последовательно получают на вход задачи (возможно, порциями входных данных) для обработки и так же заполняют очередь ответов без возможности пользователя диктовать порядок выполнения. С точки зрения пользователя вся вычислительная группа действует как единый механизм, который приоритезирует поставленные задачи по своей внутренней логике.

Проводя параллели с существующими механизмами, можно выделить следующие классы задач, применяемые для выполнения в указанных условиях: кодирование и обработка информации вне потока (преобразование мультимедиа в различные форматы для хостингов и т.п.); распознавание образов в изображениях, поиск паттернов, транскрипция аудио; обработка больших данных; работа систем анализа информации (боты аналитики социальных сетей, анализ генерируемого контента, модерация).

Следует отметить, что реализация некоторых принципов и идей Edge-вычислений направлена на решение иного класса задач и преодоление других ограничений существующих систем. Например, в [13] описывается посвоему аналогичное решение с выгрузкой вычислений существующих программных комплексов на Edge-платформы (в том числе и на базе публичных облачных провайдеров) для экономии ограниченного ресурса мобильной платформы.

Вопросы информационной безопасности заслуживают отдельного рассмотрения, но ключевые факторы достаточно очевидны: вычислительные узлы могут располагаться вне зоны административного контроля пользователя, без возможности обеспечения гарантии выполнения с точки зрения целостности обрабатываемых данных и программного кода от намеренного вмешательства или случайных неисправностей.

Передача больших данных не является проблемой для выбранной реализации шины данных RabbitMQ, что подробно разобрано в [14], однако буферизация больших сообщений (порядка сотни мегабайт) вызовет определенные сложности для хоста обмена сообщениями. Имеет смысл ограничить размер передаваемых данных и предусмотреть сборку результата на уровне решаемой пользовательской задачи.

В целом можно предположить наибольшую эффективность данного решения при реализации систем анализа информации, распложенной на внешнем ресурсе, например, группа ботов-аналитиков содержимого контента социальных сетей. Общий алгоритм работы подобного решения выглядит следующим образом:

– развертывание экземпляров ботов-аналитиков на каждом вычислительном узле;

– постановка задач (например, список страниц для анализа или паттерн) ботам с применением разработанного механизма;

– работа ботов-аналитиков в рамках распределенных между экземплярами задач;

– агрегация результатов с применением созданного механизма.

## Заключение

Поставленные перед исследователями задачи были успешно решены: разработан и апробирован метод резервирования, реализован алгоритм распределения задач и агрегации результатов, проведен анализ возможностей применения алгоритма для разного уровня задач (в том числе коммерческих).

Работа подразумевает широкое дальнейшее развитие вопросов стабилизации алгоритма и кода реализации, проработки нештатных и пограничных ситуаций, повышения надежности системы.

Также запланировано развитие подсистемы трансляторов: отход от асинхронного REST API, исследование различных путей организации обмена данными между пользовательской информационной системой и нагрузками, развернутыми на вычислительных узлах.

*Работа выполнена при финансовой поддержке РФФИ, грант № 20-47-720006.*

### *Литература*

1. Shi W., Cao J., Zhang Q., Li Y., Xu L. Edge computing: Vision and challenges. IEEE Internet of Things J., 2016, vol. 3, no. 5, pp. 637–646. DOI: 10.1109/JIOT.2016.2579198.

2. Yamato Y., Demizu T., Noguchi H., Kataoka M. Automatic GPU offloading technology for open IoT environment. IEEE Internet of Things J., 2019, vol. 6, no. 2, pp. 2669–2678. DOI: 10.1109/JIOT.2018.2872545.

3. Khan W., Ahmed E., Hakak S., Yaqoob I., Ahmed A. Edge computing: A survey. Future Generation Computer Systems, 2019, vol. 97, pp. 219–235. DOI: 10.1016/j.future.2019.02.050.

4. Abbas N., Zhang Y., Taherkordi A., Skeie T. Mobile edge computing: A survey. IEEE Internet of Things J., 2018, vol. 5, no. 1, pp. 450–465. DOI: 10.1109/JIOT.2017.2750180.

5. Sabireen H., Neelanarayanan V. A review on fog computing: Architecture, fog with IoT, algorithms and research challenges. ICT Express, 2021, vol. 7, no. 2, pp. 162–176. DOI: 10.1016/j.icte.2021.05.004.

6. Воробьев А.М., Боганюк Ю.В., Воробьева М.С., Козлова А.О. Экспериментальная реализация гибридной модели облачного сервиса на принципах экономики совместного потребления // Инженерный вестник Дона. 2021. № 9. С. 135–152.

7. Su C.-L., Chen P.-Y., Lan C.-C., Huang L.-S., Wu K.-H. Overview and comparison of OpenCL and CUDA technology for GPGPU. Proc. IEEE Asia Pacific Conf. on Circuits and Systems, 2012, pp. 448–451. DOI: 10.1109/APCCAS.2012.6419068.

8. Zanzi L., Cirillo F., Sciancalepore V., Giust F. et al. Evolving multi-access edge computing to support enhanced IoT deployments. IEEE Communications Standards Magazine, 2019, vol. 3, no. 2, pp. 26–34. DOI: 10.1109/MCOMSTD.2019.1800009.

9. Das A., Patterson S., Wittie M. EdgeBench: Benchmarking edge computing platforms. Proc. IEEE/ACM Int. Conf. UCC Companion, 2018, pp. 175–180. DOI: 10.1109/UCC-Companion.2018.00053.

10. Rostanski M., Grochla K., Seman A. Evaluation of highly available and fault-tolerant middleware clustered architectures using RabbitMQ. Proc. Annals of Computer Science and Information Systems, 2014, pp. 879–884. DOI: 10.15439/2014F48.

11. Dixit S., Madhu M. Distributing messages using Rabbitmq with advanced message exchanges. Int. J. Res. Stud. Comput. Sci. Eng., 2019, vol. 6, no. 2, pp. 24–28. DOI: 10.20431/2349-4859.0602004.

12. Ayanoglu E., Aytas Y., Nahum D. Mastering Rabbitmq. Birmingham, UK, Packt Publ. Ltd, 2016, 286 p.

13. Lin L., Liao X., Jin H., Li P. Computation offloading toward edge computing. Proc. IEEE, 2019, vol. 107, no. 8, pp. 1584–1607. DOI: 10.1109/JPROC.2019.2922285.

14. Sharvari T., Sowmya N.K. A study on modern messaging systems-kafka, rabbitmq and nats streaming. ArXiv, 2019. URL: https://arxiv.org/abs/1912.03715 (дата обращения: 09.08.2022).

Software & Systems
DOI: 10.15827/0236-235X.141.107-114

Received 17.08.22
2023, vol. 36, no. 1, pp. 107–114

## An algorithm for using distributed computing resources based on the Edge computing principles

***A.M. Vorobev*** [1], *Senior Lecturer, a.m.vorobev@utmn.ru*
***M.S. Vorobeva*** [1], *Ph.D. (Engineering), Associate Professor, m.s.vorobeva@utmn.ru*
***Yu.V. Boganyuk*** [1], *Postgraduate Student, y.v.boganyuk@utmn.ru*

[1] *University of Tyumen, Tyumen, 625003, Russian Federation*

**Abstract.** The article describes the issues of implementing the algorithm for distributing computational tasks over a set of distributed computing resources with subsequent aggregation of the results. This algorithm is the key one in the framework of the data center implementation project based on the sharing economy principles. The mechanism prototype is implemented in Python 3.8 using the PostgreSQL 14 DBMS, the message passing system is implemented on the basis of RabbitMQ 3.9, the computing node platform is CentOS 8 Stream OS.

The purpose of the work is to implement a scalable distributed computing mechanism for using as the main instrument of task distribution and aggregation of results within the framework of the data center concept based on the sharing economy principles.

The subject of the study is the methods of computing power redundancy and use, aggregation of the working results of software algorithms.

The proposed mechanism solves the computing distribution problem with subsequent aggregation of results among computing nodes with different technical characteristics. There is an implemented interface suitable for

integrating into client information systems as a means of uploading calculations with access in the REST API gateway format.

The theoretical significance of the work is in combining the existing principles and ideas of Edge computing to solve a different class of problems, where the problem is the lack of a computing resource for information system tasks, and not insufficient characteristics of the existing model.

The practical significance is in the development of an application tool for using external computing power to solve a wide class of client tasks, which gives the possibility of organizing commercial interaction between owners of unused computing resources and owners of information systems that lack computing power.

**Keywords:** distributed computing, messaging bus, AMQP, asynchronous calls, virtual machine, REST API, calculation offload.

***Acknowledgements.*** *The work was financially supported by the Russian Foundation for Basic Research, grant no. 20-47-720006.*

### *References*

1. Shi W., Cao J., Zhang Q., Li Y., Xu L. Edge computing: Vision and challenges. *IEEE Internet of Things J*., 2016, vol. 3, no. 5, pp. 637–646. DOI: 10.1109/JIOT.2016.2579198.
2. Yamato Y., Demizu T., Noguchi H., Kataoka M. Automatic GPU offloading technology for open IoT environment. *IEEE Internet of Things J*., 2019, vol. 6, no. 2, pp. 2669–2678. DOI: 10.1109/JIOT.2018. 2872545.
3. Khan W., Ahmed E., Hakak S., Yaqoob I., Ahmed A. Edge computing: A survey. *Future Generation Computer Systems*, 2019, vol. 97, pp. 219–235. DOI: 10.1016/j.future.2019.02.050.
4. Abbas N., Zhang Y., Taherkordi A., Skeie T. Mobile edge computing: A survey. *IEEE Internet of Things J*., 2018, vol. 5, no. 1, pp. 450–465. DOI: 10.1109/JIOT.2017.2750180.
5. Sabireen H., Neelanarayanan V. A review on fog computing: Architecture, fog with IoT, algorithms and research challenges. *ICT Express*, 2021, vol. 7, no. 2, pp. 162–176. DOI: 10.1016/j.icte.2021.05.004.
6. Vorobev A.M., Boganyuk Yu.V., Vorobeva M.S., Kozlova A.O. Experimental implementation of hybrid cloud service based on the sharing economy principles. *Engineering J. of Don*, 2021, no. 9, pp. 135–152 (in Russ.).
7. Su C.-L., Chen P.-Y., Lan C.-C., Huang L.-S., Wu K.-H. Overview and comparison of OpenCL and CUDA technology for GPGPU. *Proc. IEEE Asia Pacific Conf. on Circuits and Systems*, 2012, pp. 448–451. DOI: 10.1109/APCCAS.2012.6419068.
8. Zanzi L., Cirillo F., Sciancalepore V., Giust F. et al. Evolving multi-access edge computing to support enhanced IoT deployments. *IEEE Communications Standards Magazine*, 2019, vol. 3, no. 2, pp. 26–34. DOI: 10.1109/MCOMSTD.2019.1800009.
9. Das A., Patterson S., Wittie M. EdgeBench: Benchmarking edge computing platforms. *Proc. IEEE/ACM Int. Conf. UCC Companion*, 2018, pp. 175–180. DOI: 10.1109/UCC-Companion.2018.00053.
10. Rostanski M., Grochla K., Seman A. Evaluation of highly available and fault-tolerant middleware clustered architectures using RabbitMQ. *Proc. Annals of Computer Science and Information Systems*, 2014, pp. 879–884. DOI: 10.15439/2014F48.
11. Dixit S., Madhu M. Distributing messages using Rabbitmq with advanced message exchanges. *Int. J. Res. Stud. Comput. Sci. Eng.*, 2019, vol. 6, no. 2, pp. 24–28. DOI: 10.20431/2349-4859.0602004.
12. Ayanoglu E., Aytas Y., Nahum D. *Mastering Rabbitmq*. Birmingham, UK, Packt Publ. Ltd, 2016, 286 p.
13. Lin L., Liao X., Jin H., Li P. Computation offloading toward edge computing. *Proc. IEEE*, 2019, vol. 107, no. 8, pp. 1584–1607. DOI: 10.1109/JPROC.2019.2922285.
14. Sharvari T., Sowmya N.K. A study on modern messaging systems-kafka, rabbitmq and nats streaming. *ArXiv*, 2019. Available at: https://arxiv.org/abs/1912.03715 (accessed August 09, 2022).

**Для цитирования**

Воробьев А.М., Воробьева М.С., Боганюк Ю.В. Разработка алгоритма применения распределенных вычислительных ресурсов на основе принципов Edge-вычислений // Программные продукты и системы. 2023. Т. 36. № 1. С. 107–114. DOI: 10.15827/0236-235X.141.107-114.

**For citation**

Vorobev A.M., Vorobeva M.S., Boganyuk Yu.V. An algorithm for using distributed computing resources based on the Edge computing principles. *Software & Systems*, 2023, vol. 36, no. 1, pp. 107–114 (in Russ.). DOI: 10.15827/0236-235X.141.107-114.