Българска Академия на Науките
Институт по Математика и Информатика

Цветан Красимиров Цоков

IoT платформи и протоколи

Автореферат

на дисертация

за получаване на образователна и научна степен "Доктор"
в професионално направление 4.6. "Информатика и Компютърни Науки"
научна специалност "Информатика"

Научен ръководител:
Доц. Христо Костадинов

София, 2024

Съдържание

Въведение

Основната цел на дисертацията е да предостави решение за оптимално разпределение (планиране) и управление на изчислителни (процесор, памет, диск) и мрежови (латентност, капацитет на трафик) ресурси в Облачни/Крайни/Мъглявинни (Cloud/Edge/Fog) платформи от сферата на разпределените системи, които включват динамични и подвижни инфраструктурни възли и крайни потребители. Тя е чувствителна към инфраструктурната топология, което означава, че наблюдава и непрекъснато се адаптира към състоянието на инфраструктурата. Съобразена е със съвременните приложения изградени от микроуслуги с комплексни зависимости и инфраструктурни клъстери с ограничен ARM64 хардуер, използвани в Интернет на Нещата (Internet of Things - IoT). В такава динамична среда съвременните Облачни/Крайни/Мъглявинни платформи от научната и практическата области не успяват да управляват Качеството на Услугата (Quality of Service - QoS) и Качеството на Потребителското Изживяване (Quality of Experience - QoE) на латентно-чувствителните бизнес приложения, което води до големи общи закъснения и влошено потребителско изживяване. Предлагайки решение на този проблем, дисертацията позволява поддържането на IoT приложения работещи в реално време с мобилни възли като автономни превозни средства, добавена/виртуална реалност (AR/VR), космически изчисления, умни градове и други.

Дисертацията извършва цялостен анализ на свързаните публикации в областта и идентифицира няколко ключови параметъра, които трябва да бъдат поддържани от всяка Облачна/Крайна/Мъглявинна платформа, за да се справи ефективно с динамично движещи се инфраструктурни възли и с качеството на услугата/потребителското изживяване на приложението. Всички изследвания, включително предложеното решение в дисертацията, се сравняват спрямо тези параметри и накрая ползите от предложеното решение са валидирани и подчертани.

Методологията на дисертацията включва формулиране на представения проблем като математическа оптимизационна задача и предоставяне на модел на Смесено Целочислено Линейно Програмиране (Mixed-Integer Linear Programming - MILP) за намиране на най-оптималните решения. Архитектурата на модела е проектирана и внедрена в най-популярната Облачна/Крайна/Мъглявинна платформа, използвана в практиката, известна като Kubernetes (K8s). Имплементацията е осъществена на езика за програмиране Golang. Изграден е тестов клъстер, съдържащ устройства с ограничени ресурси (Raspberry Pi) и имплементираната

платформа е инсталирана върху него. Тестовият клъстер емулира динамична среда, съставена от географски разпределени мобилни Крайни/Мъглявинни възли, които се движат в пространството. Основната му цел е да оцени как предложеният модел намалява общата мрежова латентност на бизнес приложение в тази среда. Бизнес приложението, използвано за оценката, е част от дисертацията и се нарича Ecologic. То представлява практическа облачна IoT апликация и неговата функционалност е да измерва и контролира емисиите на въглероден диоксид от превозни средства, което го прави подходящ за работа в сценарии на умни градове. Получените резултати показват, че моделът ефективно планира, управлява и премества ресурси върху близки инфраструктурни възли, като същевременно се адаптира към движенията на тези възли по динамичен начин. Така мрежовата латентност на бизнес приложението непрекъснато се свежда до минимум по време на изпълнение. Един от най-важните резултати на дисертацията е, че решението е подходящо за практически инфраструктури с ограничени ARM64 устройства от реалния свят.

Глава 1

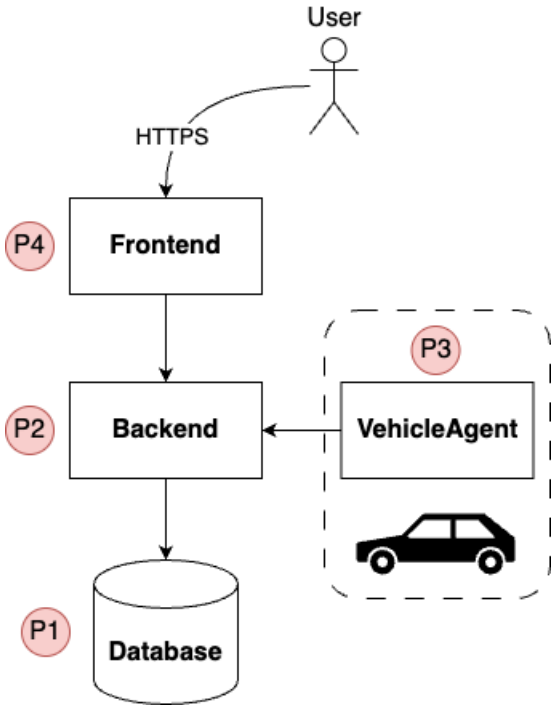
Обзор на Крайни/Мъглявинни разпределени системи

Ниската мрежова латентност и изискванията за мобилност са от решаващо значение за широк спектър от приложения за да може да се осигури тяхното ниво на качество на услугата и потребителско изживяване. Такива видове приложения, са например от сферата на добавена и виртуална реалност, автономни превозни средства и контрол на емисиите и трафика в умни градове [1]. Тези изисквания се адресират от новите Крайни/Мъглявинни платформи, които разширяват изчислителните ресурси на Облачните платформи с допълнителни инфраструктурни възли, разположени в периферията на мрежата и по-близо до IoT устройствата и крайните потребители [2]—[4]. Те осигуряват по-малка латентност и по-голям капацитет на трафик.

Облачните платформи разчитат на техники за виртуализация, за да утилизират използваните ресурси и да поддържат изолация на потребители. Контейнерно базираната виртуализация се превърна в стандартен и най-използван тип в Облачните платформи в сравнение с базирания на хипервизор тип, тъй като е

по-лек и има по-добра поддръжка за архитектури на микроуслуги [5]. На ниво хардуер, ARM процесорната архитектура става все по-популярна за сървъри, но нейната поддръжка за виртуализация е по-ограничена спрямо преобладаващата x86 процесорна архитектура [6]. Известно е, че Крайният (Edge) мрежови слой може да бъде съставен от възли с ниска мощност и ARM процесорна архитектура [7]. От изложеното по-горе следва, че ограничените устройства с ARM процесорна архитектура и контейнерния тип виртуализация са широко използвани в Крайните/Мъглявинни платформи и са актуална и важна изследователска тема.

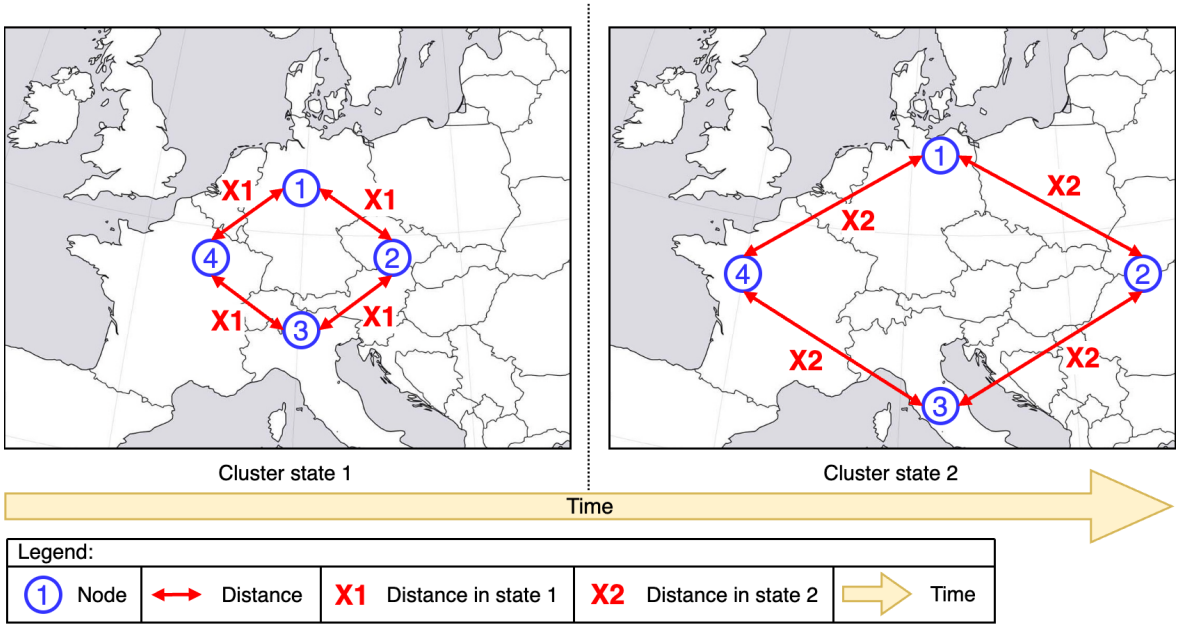
От друга страна, архитектурата с микроуслуги се превърна в широко приета и установи принципи за изграждане на софтуерни приложения, подходящи за изпълнение в Облачни платформи - Cloud-native [8] принципи и методологията Twelve-Factor App. В момента нов набор от принципи се появява в практиката, наречен Edge-native [9], [10]. Принципите Edge-native са разширение и еволюция на Cloud-native. Те описват как да се изгради набор от микроуслуги по най-изолирания, мащабируем и устойчив начин, така че те да могат да работят надеждно в среда на Крайна/Мъглявинна платформа. Известно е, че микроуслугите могат да образуват граф от свързани зависимости, които комуникират помежду си, наречени Вериги от Функции за Услуги (Service Function Chains - SFCs). Например Ecologic [1] е типично Edge-native приложение за наблюдение и контрол на емисиите на превозни средства в градове. Той се състои от четири взаимозависими микроуслуги, показани във Фиг. 1.1. Микроуслугата Database (P1) съдържа данни, свързани с параметрите на превозните средства и емисиите, докато услугата Backend (P2) предоставя интерфейс и анализ на тези данни. Микроуслугите VehicleAgent (P3) работят като инстанции върху хардуерните възли, разположени във всички регистрирани превозни средства. Те събират параметри на ниско ниво от превозните средства и ги изпращат до Backend (P2) микроуслугата. Приложението Frontend (P4) предоставя уеб-базиран потребителски интерфейс. Услугите Database (P1), Backend (P2) и Frontend (P4) могат да работят на неподвижни или подвижни възли. Всички микроуслуги поддържат множество реплики, които могат да работят върху различни хардуерни устройства. В такива динамични среди инфраструктурните възли образуват гео-разпределен мрежови клъстер и могат да се движат във времето и пространството. Примерен клъстер от възли, които променят местоположението си в две различни точки от времето е показана във Фиг. 1.2. Инсталирането (поставянето) на зависими микроуслуги на отдалечени възли без контролиране на латентността може да повлияе върху времето за реакция на приложението и да наруши цялостното качество на услугата и потребителското изживяване. Ако възлите се движат, първоначалното поставяне на зависимите микроуслуги може да е върху близки възли, но когато те се отдалечат един от друг, крайната латентност на приложението ще бъде увеличена и цялостното QoS/QoE - намалено. Необходимо е зависимите микроуслуги да бъдат преразположени (пре-инсталирани) върху по-близки възли, ако такива съществуват, и платформата динамично да се адаптира към мобилността на възлите във времето и пространството. Тя трябва да следи инфраструктурата от хетероген-



Фигура 1.1: Архитектура на микроуслугите на приложението EcoLogic [1].

ни възли, които имат различни ресурси и връзки с вариращи мрежово време на забавяне и капацитет на трафик.

Предишните изследвания по темата за латентно-чувствително планиране на изчислителни ресурси се фокусират предимно върху теоретични дефиниции (напр. [11]—[14]) или евристично-базирани алгоритми (напр. [15], [16]), които обикновено са валидирани на симулатори като CloudSim [17], EdgeCloudSim [18] и iFogSim [19]. Често ако валидацията е базирана на реални устройства, те имат голям капацитет от изчислителни ресурси и използват x86 или x64 процесорна архитектура. Реалните ограничени устройства с ARM процесорна архитектура от практиката не са разгледани. Освен това устройствата са неподвижни в пространството, без да притежават мобилност. В сферата на латентно-чувствителните приложения е важно Крайната/Мъглявинна платформа да поддържа разпределение на свързани микроуслуги върху мобилни възли, които променят своето местоположение във времето и пространството. Според горната експозиция няколко основни характеристики на Крайните/Мъглявинни платформи за изчисления са идентифицирани: **осведоменост (чувствителност) относно топологията на инфраструктурата**, **тип виртуализация**, **поддръжка на приложения с взаимозависими микроуслуги**, **динамичност**, **мобилност**, **осведоменост (чувствителност) относно мрежова латентност**, **поддръжка на ARM64 процесорна архитектура** и **начин на валидиране на решението**.



Фигура 1.2: Движение на възли, част от гео-разпределен (мулти-регионен) Краен/Мъглявинен клъстер.

Глава 2

IoT приложение EcoLogic

Един от най-засегнатите сектори от IoT е автомобилната индустрия. Наскоро се съобщи изследване, че десетки милиони автомобили са свързани с интернет и това число се очаква да нарасне до стотици милиони в близко бъдеще.

Дисертацията предлага решение за наблюдение и откриване на нарастващи нива от въглероден диоксид от превозни средства, наречено Ecologic. То също така се използва за оценка на представената платформа за планиране на ресурси в динамични Облачни/Крайни/Мъглявинни платформи с мобилни инфраструктурни възли, описана в следващите глави. Предложеното приложение Ecologic включва хардуерен модул, който измерва сензорни данни, свързани с емисиите на въглероден диоксид от превозно средство. Данните се изпращат към Облач-

но приложение, където се съхраняват и анализират. Резултатите от анализа се използват за контролиране на емисиите на въглероден диоксид чрез известия до водача и ограничения на мощността на превозното средство. Получените резултати, ресурси и изходен код на основните софтуерни компоненти на решението са публично достъпни. Хранилищата съдържат информация как да се настроят проектите и да се инсталират на хардуерно устройство, сървър или в Облак.

2.1 Системна архитектура

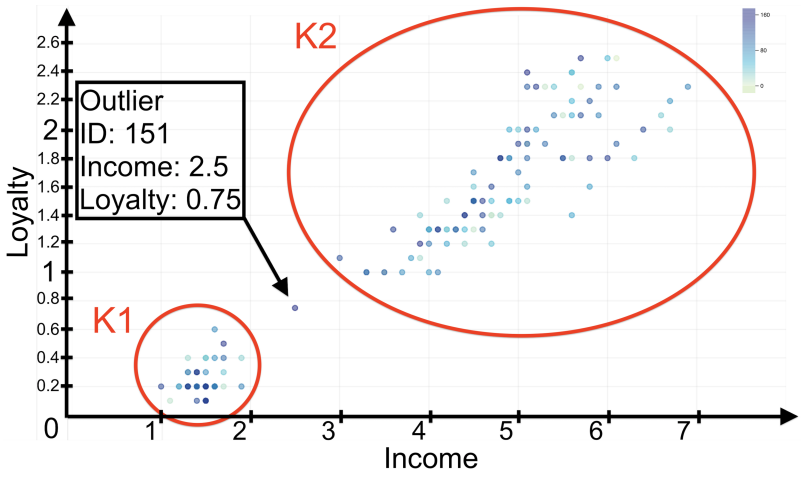
Системата Ecologic е съставена от две основни части: Edge-native хардуерни модули и Облачни микроуслуги. Хардуерните модули са разположени в превозни средства, а микроуслугите са инсталирани в Облачна платформа. Опростена архитектура на системата е показана във Фиг. 1.1. В последните години се установи стандарт в IoT системите да интегрират Облачни и Мъглявинни платформи в техните архитектури като основни компоненти, които предлагат много предимства като огромни изчислителни възможности, висока наличност, бързо възстановяване при бедствия, съхранение, мащабируемост и скорост на реакция в почти-реално време.

Хардуерният модул на системата има сензори, които измерват няколко физически параметъра на двигателя като въздушно налягане, температура на въздуха и горивна смес. Той може също да извлича параметри от бордовата диагностична система на превозното средство. Данните се изпращат към приложенията в Облачната платформа.

Облачните приложения са имплементирани като микроуслуги, които са проектирани по начин, който не зависи от платформата, за да се даде възможност за инсталиране на различни Облачни платформи. Облачните приложения съхраняват данни в релационна база от данни, която е предоставена от Облака като услуга. Те обработват входящите данни, съхраняват ги в базата данни и ги анализират. Хардуерните модули комуникират с Облака посредством безжична мрежа.

Базата данни съдържа всички измерени и изчислени физически параметри. Облачното приложение, наречено Analytics, анализира и открива аномалии върху потока от данни и извежда превозни средства, които имат неоптимални нива на въглероден диоксид. Алгоритъмът за откриване на аномалии е чрез групиране на данните в клъстери (K-Means). Хардуерният модул се уведомява, когато превозно средство бъде открито от системата като аномалия, с неоптимално количество емисии. След това хардуерният изпълнител се стартира автоматично, за да намали количеството на емисиите. Това създава контролна верига с обратна връзка (feedback control loop). По този начин системата наблюдава и контролира количеството въглероден диоксид в атмосферата в реално време.

Облачните приложения предоставят уеб потребителски интерфейс, базиран на HTML5, JavaScript и CSS. Той е публично достъпен за клиенти.



Фигура 2.1: Идентифицирани клъстери в множество от данни с известни аномалии.

2.2 Резултати от алгоритъма за анализ на данни

Тази глава доказва релевантността на алгоритъма използван в Ecologic чрез представяне на казус от практиката с два отделни набора от данни: тестов набор от данни и реален набор от данни. Целта на казуса е да провери способността на алгоритъма за идентифициране на аномалии в набора от данни, които представляват превозни средства с неоптимално количество емисии въглероден диоксид в контекста на регион с голям брой превозни средства. Облачната платформа и услугите, които са конфигурирани за казуса, са както следва:

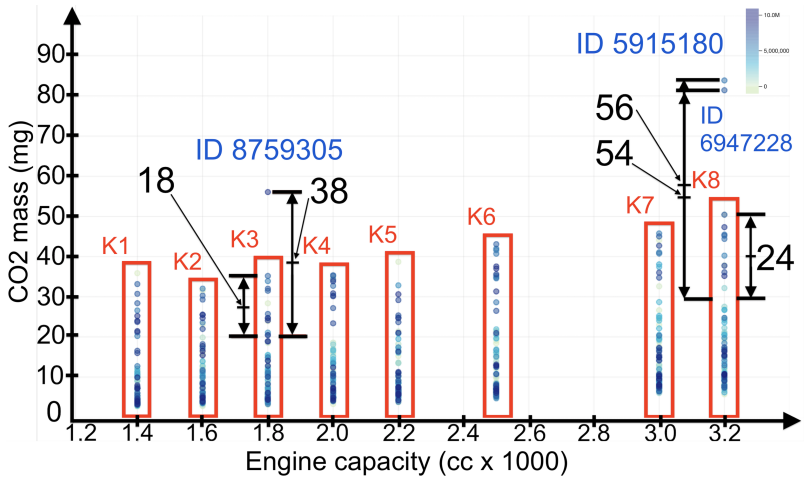
• Всички описани облачни приложения са инсталирани в Облачната платформа на SAP Cloud.

• Данните се съхраняват в релационна база данни HANA, предоставена като услуга за съхранение от Облачната платформа.

• Приложението за анализ на данни (Analytics) е осигурено като услуга от SAP Облачната платформа и предоставя алгоритъм за групиране на данни и откриване на аномалии.

Първо за валидиране на правилността на алгоритъма за откриване на аномалии се използва тестов набор от данни, който съдържа известни аномалии. Алгоритъмът се изпълнява върху тестовия набор от данни и резултатът, който е получен се сравнява с известния резултат. За тази цел е използван официален тестов набор от данни [20]. Той съдържа данни за клиенти със следните параметри: идентификационен номер (ID), име, продължителност на живота, нови разходи, доход и лоялност. Резултатите, получени при изпълнение на алгоритъма за групиране с откриване на аномалии върху тестовия набор от данни, са показани във Фиг. 2.1.

Абсцисната ос съдържа дохода на клиентите. Ординатната ос съдържа лоял-



Фигура 2.2: Идентифицирани клъстери в множество от данни получено от реално превозно средство.

ността на клиентите. Алгоритъмът успешно създава два клъстера (K1 и K2) и открива една аномалия. Клъстерите представляват два вида клиенти: клиенти с нисък доход и ниска лоялност и клиенти с висок доход и висока лоялност. Отклонението, маркирано във Фиг. 2.1 като аномалия (Outlier), представлява точка от данни, която е твърде далеч от центровете на клъстерите K1 и K2 спрямо другите точки от данни. Клиентът, който е идентифициран като аномалия, има идентификационен номер 151, доход 2.5 и лоялност 0.75. Резултатът, получен от алгоритъма, е същият като известния резултат от тестовия набор от данни, което потвърждава правилността на алгоритъма за този набор от данни.

Хардуерният модул на Ecologic е инсталиран в реално превозно средство с вътрешно горене с работен обем от 1800 кубически сантиметра и работещ с бензин. За да се придобие по-голям набор от данни, събраните реални данни се разширяват пропорционално със симулирани данни. Крайният оперативен набор от данни съдържа данни за превозни средства с различни работни обеми и количества емисии на въглероден диоксид. Абсцисната ос съдържа обема на двигателя, измерен в кубични сантиметри (cc). Ординатната ос съдържа масата на въглеродния диоксид, изхвърлен от превозното средство, измерена в милиграми (mg). Резултатите от групирането, получени след изпълнение на алгоритъма за групиране, са показани във Фиг. 2.2. Точките с данни със специфични идентификационни номера съответстват на отделните превозни средства.

Алгоритъмът за групиране би трябвало да поставя превозни средства с еднакъв двигателен обем и различни количества емисии в един и същи клъстер, когато е възможно. Това гарантира, че превозните средства с неоптимални емисии няма да бъдат поставени в никой клъстер и ще бъдат идентифицирани като аномалии. Алгоритъмът връща 8 групи (K1-K8) за текущия набор от данни, които представляват превозни средства, групирани в 8 различни двигателни обеми - 1400,

1600, 1800, 2000, 2200, 2500, 3000 и 3200 кубически сантиметра, съответно. Тези превозни средства имат различни стойности на въглероден диоксид, вариращи между 2.70 и 50.44 милиграма. Превозните средства с идентификационни номера 8759305, 6947228 и 5915180 са идентифицирани като аномалии, защото разстоянието от тях до техните най-близки групи е по-голямо от вътрешната дистанция на групата. Тези превозни средства нямат оптимални нива на въглероден диоксид емисии в сравнение с останалите превозни средства, разположени в клъстери K1-K8. Важно е да се отбележи, че аномалията с идентификационен номер 8759305 представлява реално превозно средство с измерени параметри по време на празен ход на студен двигател. Резултатът, получен от алгоритъма, е възможен на практика и потвърждава неговата приложимост към реалния набор от данни.

Глава 3

Сравнителен анализ на разпределение на ресурси в Крайни/Мъглявинни платформи

Тази глава прави сравнителен анализ на някои важни резултати за разпределение на ресурси в Крайни/Мъглявинни платформи свързани с основните характеристики идентифицирани в Глава 1. Тези характеристики са осведоменост (чувствителност) относно топологията на инфраструктурата, тип виртуализация, поддръжка на приложения с взаимозависими микроуслуги, динамичност, мобилност, осведоменост (чувствителност) относно мрежова латентност, поддръжка на ARM64 процесорна архитектура и начин на валидиране на решението.

В Крайните/Мъглявинни платформи ресурсите (процесор, памет, съхранение и мрежа) са ограничени по размер и количество, но те са от съществено значение за чувствителни към латентност приложения. Това направи разпределението на ресурсите в такива платформи важен изследователски предмет.

Проблемът с разпределението на ресурсите в областта на разпределените системи (Крайни/Мъглявинни платформи) се изучава и класифицира в [21]. Този обзор идентифицира няколко показателя, включително използване на ресурси, цена, енергия и мрежова латентност. Латентността е два вида: възел към възел и краен потребител до възел. Освен това този преглед класифицира разпределе-

нието на ресурсите чрез няколко техники: Целочислено Линейно Програмиране (ILP), Нелинейно Програмиране (NLP), евристични алгоритми, подходи основани на приспособяване, множество критерии за вземане на решения, подходи базирани на игри и машинно обучение. Много свързани изследвания обикновено използват ILP модели за намиране на оптималното решение въз основа на целева метрика. Основното ограничение на тези модели е невъзможността да се намери решение в приемлив период от време, което ограничава тяхната приложимост в реални практически среди. Въпреки това те могат да служат като индикатор за оптималността на евристични алгоритми.

Наличността и изолацията на ресурсите в Мъглявинните платформи обикновено се постигат с помощта на модел за виртуализация. Той е класифициран в [21] въз основа на два параметъра: виртуални машини (VM) и контейнеризация. Много статии разглеждат виртуални машини. Има и изследвания, които се фокусират върху оптимизирането на позиционирането на контейнерите, като вземат предвид различни показатели, включително справедлив график за планиране от много наематели, време за изчакване [22] и намаляване на общата латентност [23]—[25].

Друга важна тема във Мъглявинните изчисления е осведомеността (чувствителността) за инфраструктурната топология. Тя се разглежда в [26], където базирана на микроуслуги система за топологично чувствително планиране (MOTAS) е представена. Друг топологично чувствителен планировчик, наречен Gavel, се предлага в [27], който подобрява времето за изпълнение на задачите за машинно обучение чрез фокусиране върху хетерогените ресурси в клъстера като видео карта (GPU), процесор за изчисляване на тензори (TPU), конфигурируем хардуер (FPGA) и приложно-специфичен хардуер (ASIC). Схема за планиране, която е чувствителна към топологията, която разпределя контейнери с микроуслуги към ядра на машини в клъстер за да максимизира производителността е представена в [28]. Освен това там се демонстрира механизъм за динамично сливане на горещи услуги в един контейнер.

Проблемът с разпределението на ресурсите в среда на Облачна/Крайна/Мъглявинна платформа с контейнеризирани микроуслуги се решава в [29], където MILP формулировка е представена. Формулировката разглежда концепциите на взаимосвързани микроуслуги, няколко технологии за безжична комуникация (LPWAN) и целеви функции, сред които е и оптимизация на мрежовата латентност.

Динамични алгоритми за поставяне на реплики на микроуслуги за предоставяне на данни са представени в [11], [12], [30]. Те постигат подобряване на наличността на данните. Тези три работи използват средната латентност като показател и не разглеждат проблемите, свързани с разпределението на натоварването. Проблемът за оптимално поставяне на последователност от услуги в мобилни микрооблаци за минимизиране на средната цена в рамките на фиксиран времеви период е изследван в [31].

Алгоритми за динамично позициониране на реплики и поддържане на обща

мрежова латентност в рамките на предварително дефиниран праг се предлагат в две работи, наречени Hona [24] и Voila [25]. Те поддържат както стационарни, така и мобилни възли.

Разпределена система за оркестрация в Крайна/Мъглявинна платформа, наречена ORCH, е представена в [32]. Тя се занимава с проблема за поставяне на изчислителни задачи в разпределена динамична среда в Крайна/Мъглявинна платформа. Системата успява да постави задачи на потребители върху конкретни устройства, които са част от клъстер от устройства в географска област, така че да бъдат изпълнени.

Разширение на двете свързани работи [25], [32], наречена Violin, е показана в [33]. Демонстриран е оптимизационен модел за поставяне на задачи, разполагане на услуги и предоставяне на устройства в Крайна/Мъглявинна платформа.

Глава 4

Модел на Смесено Целочислено Линейно Програмиране (MILP)

Проблемът за динамично мрежово-чувствително разпределение на контейнери с микроуслуги в Крайни/Мъглявинни инфраструктури, съставени от набор от подвижни възли, които променят своите изчислителни и мрежови ресурси, може да се счита за оптимизационен проблем с NP-трудна сложност [34]. Много ILP модели предоставят оптимални решения на този проблем, но техните променливи и целеви функции не съответстват на модела на реалните Крайни/Мъглявинни платформи и не могат да бъдат приложени на практика. Те не са подходящи за ограничени IoT устройства с ARM процесорна архитектура и са валидирани чрез симулации, вместо в реална среда.

В дисертацията се предлага нов MILP оптимизационен модел за мрежово-чувствително разполагане на контейнери с микроуслуги в динамични Облачни/Крайни/Мъглявинни инфраструктури, съставени от подвижни и ограничени възли с ARM процесорна архитектура. Неговите целеви функции максимизират общия брой инсталирани бизнес приложения, минимизират общото движение на реплики между възлите и минимизират мрежовата латентност на бизнес приложенията, за да осигурят възможно най-доброто решение (планиране) за поставяне. Променливите, ограниченията и целевите функции на модела са обяс-

нени в следващите раздели.

4.1 Описание на модела и променливите

MILP моделът е адаптиран към приложния модел на K8s и може да се използва в широк диапазон от Облачни/Крайни/Мъглявинни инфраструктури. Той третира инфраструктурата като набор от възли с ограничени изчислителни ресурси, например процесор и памет. Възлите са свързани с мрежови връзки, които имат специфично забавяне и капацитет на трафика, които могат да варират между отделните изпълнения на модела. Микроуслугите, работещи върху клъстера, се представят като набор от бизнес приложения със специфични зависимости и изисквания по отношение на изчислителни ресурси, мрежова латентност и капацитет на трафика. Разположението на контейнерите с микроуслуги върху възлите пряко засяга стойностите на крайната латентност и капацитет на трафик на приложението. За да произведе план на разположение с минимална латентност в динамична среда, моделът цели да максимизира поставените бизнес приложения и минимизира движението на репликите и мрежовата латентност. Тези целеви функции се изпълняват една след друга.

Моделът третира набора от възли в Облачен/Мъглявинен клъстер, базиран на Kubernetes, като набор $N = \{n_1, n_2, \ldots\}$. Всеки възел n има следните изчислителни и мрежови ресурси:

- C_n: Общо количество на процесорни ядра в процесорни единици, където $C_n \in \mathbb{Q}^+, C_n > 0$.
- E_n: Общо количество на памет в мегабайт (Mbyte), където $E_n \in \mathbb{N}$.
- B_n: Общо количество на капацитет на трафик в мегабит/секунда (Mbit/sec), където $B_n \in \mathbb{N}$.

Параметрите на капацитет на трафик и латентност между всеки два възела се определят със следните матрици:

- B_{n_u,n_v}: Матрица за пропускателна способност, представляваща мрежовата пропускателна способност от изходен възел n_u до целеви възел n_v в мегабит/секунда (Mbit/sec).
- T_{n_u,n_v}: Матрица за латентност, представляваща мрежовата латентност от изходен възел n_u до целеви възел n_v в милисекунди (ms).

Възлите са разделени в инфраструктурни региони и зони, формиращи специфични групи. Всяка област се състои от зони. Всяка зона z е разположена само в една област u. Възел $n \in N$ може да бъде разположен само в една област и зона. Наборът от инфраструктурни региони и зони е определен като:

- $U = \{u_1, u_2, \ldots\}$: набор от инфраструктурни региони, където всеки регион u е наименован низ.

- $Z = \{z_1, z_2, \ldots\}$: набор от инфраструктурни зони, където всяка зона z е наименован низ.

Матрицата на членството на възел $M_{n,u,z}$ (бинарна) представя дали възелът n е член на регион u и зона z в този регион. Ще казваме $M_{n,u,z} = 1$, ако възелът n принадлежи към региона u и зоната z.

Всяко бизнес приложение се дефинира като w. Всички бизнес приложения са част от набор от бизнес приложения $W = \{w_1, w_2, \ldots\}$.

Микроуслугите се наричат Pod в модела на K8s. Ще използваме думите Pod и микроуслуга взаимозаменяемо. Всяко бизнес приложение w е съставено от набор $P = \{p_1, p_2, \ldots\}$. Матрицата на членството M_p^w (бинарно), представя дали микроуслуга p е член на бизнес приложението w. Ще отбелязваме с $M_p^w = 1$, ако микроуслуга p е член на бизнес приложението w.

Всяка микроуслуга има една или повече инстанции, наречени реплики в модела на K8s. Всяка микроуслуга е съставена от набор от реплики $R = \{r_1, r_2, \ldots\}$. Броят на исканите реплики за всяка микроуслуга p се определя като R_p^{req}. Максималният лимит на реплики за всяка микроуслуга p е определен като R_p^{max}.

Всяка микроуслуга има следните изисквания за ресурси, необходими за правилната му работа:

- c_p: изискване за процесор в процесорни единици (cpu), където $c_p \in \mathbb{Q}^+, c_p > 0$.

- e_p: изискване за памет в мегабайт (Mbyte), където $e_p \in \mathbb{N}$.

- b_p: изискване за мрежови капацитет на трафик в мегабит/сек (Mbit/sec), където $b_p \in \mathbb{N}$.

Като пример, изискване за процесор от 0.1 процесорни единици (т.е. 100 millicpu) означава, че тази микроуслуга има нужда от 10 процента процесорно ядро за да работи правилно.

Изискванията за мрежови капацитет на трафик и латентност между две зависими микроуслуги са определени чрез следните матрици:

- β_{p_i,p_j}: Матрица на капацитет на трафик, която дефинира изискването за мрежови капацитет на трафик във връзката от изходна микроуслуга p_i до целева микроуслуга p_j.

- τ_{p_i,p_j}: Матрица на закъснението, която дефинира изискването за мрежово забавяне във връзката от изходна микроуслуга p_i до целева микроуслуга p_j.

Освен това предикатните матрици на изискванията дефинират дали съответстващите изисквания за забавяне и капацитет на трафик между зависими микроуслуги са гарантирани или не:

- $\Omega^{\beta}_{p_i,p_j}$: Предикатна матрица на бизнес приложение за изискването за капацитет на трафик (бинарна). Ще отбелязваме с $\Omega^{\beta}_{p_i,p_j} = 1$, ако изискванията за капацитет на трафик от изходна микроуслуга p_i до целева микроуслуга p_j трябва да бъдат гарантирани.

- $\Omega^{\tau}_{p_i,p_j}$: Предикатна матрица на бизнес приложение за изискването за латентност (бинарна). Ще казваме $\Omega^{\tau}_{p_i,p_j} = 1$, ако изискванията за латентност от изходна микроуслуга p_i до целева микроуслуга p_j трябва да бъдат гарантирани.

Също така факторът за преместване на реплика δ се използва за строго ограничаване на общия брой възможни премествания на реплики на микроуслуга в рамките на клъстера, което засяга процеса на планиране и де-планиране (премахване).

Матрицата за поставени (планирани) бизнес приложения D_w (бинарна) представя състоянието на поставяне за всички бизнес приложения, дали всички негови микроуслуги са поставени на възли и работят напълно оперативно. Ще отбелязваме с $D_w = 1$, ако цялото бизнес приложение w е разположено и оперативно.

Матрицата за поставени микроуслуги D^w_p (бинарна) разглежда плана на разполагане за всяка микроуслуга от конкретно бизнес приложение. Ще казваме $D^w_p = 1$ ако цялата микроуслуга p от бизнес приложението w е поставена, което означава, че всички негови реплики са разположени.

Матрицата за поставени реплики $D^{w,p}_{r,n}$ (бинарна) представя състоянието на поставяне на реплики от микроуслуги върху конкретен възел. Ще отбелязваме с $D^{w,p}_{r,n} = 1$ ако реплика r от микроуслуга p, част от бизнес приложение w, е разположена на възел n. Матрицата на мрежовия поток $S^{w,p_i,r_k}_{p_j,r_m}(n_u, n_v)$ (бинарна) представя мрежовите потоци. Ще казваме $S^{w,p_i,r_k}_{p_j,r_m}(n_u, n_v) = 1$, ако реплика r_k на микроуслуга p_i е разположена на възел n_u и реплика r_m на микроуслуга p_j е разположена на възел n_v за бизнес приложение w. Тя показва, че мрежовият поток се постига между специфични реплики, разположени на конкретни възли.

Матрицата на закъснение T_w дава общото мрежово закъснение на бизнес приложение w в милисекунди.

Матрицата на движението на реплика за дадена итерация $\Delta^{w,p}_{r,n}$ (бинарна) представя дали реплика r_i е преместена от възел n върху друг. Ще казваме $\Delta^{w,p}_{r,n} = 1$ ако реплика r е преместена от възел n в друг според предходната итерация.

Матрицата на движението на реплика $\Delta^{w,p}_{r}$ (бинарна) показва дали реплика r_i е преместена от един възел върху друг. Ще отбелязваме с $\Delta^{w,p}_{r} = 1$, ако реплика r е преместена от възел според предходната итерация.

4.2 Целеви функции и ограничения

MILP моделът има следните целеви функции:

- Максимизиране на поставянето на бизнес приложения (MAX_WD).
- Минимизиране на движенията на репликите на микроуслугите през възлите между итерациите (MIN_RM).
- Минимизиране на мрежовото закъснение на бизнес приложенията (MIN_WL).

Тези целеви функции се изпълняват последователно една след друга.

Първо целевата функция за максимизиране на поставянето на бизнес приложения (MAX_WD) се изпълнява. Резултатът от нея преминава към втората целева функция, която обработва количеството на движения на репликите през възлите между итерациите (MIN_RM). Ако е необходимо, тя минимизира движенията според дефинираните ограничения и модифицира резултата. Резултатът от втората целева функция се предава на третата, която допълнително подобрява резултата с цел минимизиране на общото мрежово закъснение на бизнес приложението (MIN_WL).

Всички целеви функции са описани в следващите подсекции.

4.2.1 Максимизиране на поставяне на бизнес приложения (MAX_WD)

Целевата функция MAX_WD е отговорна за максимизиране на поставянето на бизнес приложения. Тя е обект на следните ограничения. Ограничение на микроуслуга към бизнес приложение

$$\sum_{p \in P} D_p^w = \sum_{p \in P} M_p^w,$$

където $D_w = 1$, за $\forall w \in W$, осигурява, че бизнес приложение w е напълно поставено (инсталирано) ако всички негови микроуслуги са поставени.

Тъй като микроуслугите съдържат реплики, огрничението

$$\sum_{r \in R_p^{max}} \sum_{n \in N} D_{r,n}^{w,p} = M_p^w \times R_p^{req},$$

където $D_p^w = 1$, за $\forall w \in W, \forall p \in P$, твърди, че микроуслуга p е напълно поставена ако изискания брой на нейните реплики (R_p^{req}) са поставени.

Поради това, че моделът решава върху кои възли да постави реплики, следните ограничения осигуряват, че репликите са планирани (поставени) само върху възли, които имат достатъчно налично количество на процесорни ядра, памет и

мрежови капацитет спрямо дефинираните изисквания на микроуслугите:

$$\sum_{w \in W} \sum_{p \in P} \sum_{r \in R_p^{max}} D_{r,n}^{w,p} \times c_p \leq C_n,$$

$$\sum_{w \in W} \sum_{p \in P} \sum_{r \in R_p^{max}} D_{r,n}^{w,p} \times e_p \leq E_n,$$

$$\sum_{w \in W} \sum_{p \in P} \sum_{r \in R_p^{max}} D_{r,n}^{w,p} \times b_p \leq B_n,$$

$$b_{p_i} = \sum_{\substack{j=1 \\ j \neq i}}^{|P|} \beta_{p_i,p_j},$$

за $\forall n \in N, \forall p_i \in P$, където мрежовия капацитет на трафик на микроуслуга е дефиниран с b_{p_i}.

Тогава моделът е способен да постави повече от една реплика от микроуслуга върху един и същ възел, която е нежелателно, следователно ограничението за разпространение на реплика през възли

$$\sum_{r \in R_p^{max}} D_{r,n}^{w,p} = 0 \text{ or } 1,$$

за $\forall w \in W, \forall p \in P, \forall n \in N$, осигурява, че всяка реплика от една микроуслуга е поставена на различен възел.

Също така, моделът може да постави всички реплики в един регион и зона, което отново е нежелателно поради невъзможност от висока наличност на бизнес приложението. Така, че ограничението за разпространение на реплика през региони и зони

$$\sum_{n \in N} \sum_{r \in R_p^{max}} D_{r,n}^{w,p} = \begin{cases} 1 & \text{ако } M_{n,u,z} = 1 \\ 0 & \text{ако } M_{n,u,z} = 0, \end{cases}$$

за $\forall w \in W, \forall p \in P, \forall u \in U, \forall z \in Z$, прави така, че репликите на дадена микроуслуга да са поставени в различни региони и зони.

Накрая целевата функция за максимизиране на поставянето на бизнес приложенията е дефинирана като:

$$\text{MAX\_WD} = max \sum_{w \in W} D_w.$$

Тя има за цел да постави колкото е възможно по-голям брой бизнес приложения върху множеството от възли, удовлетворявайки всички дефинирани ограничения.

4.2.2 Минимизиране на движения на реплики (MIN_RM)

Резултатът от първата целева функция (MAX_WD) съдържа конкретната схема на поставяне на репликите на микроуслугите върху множество от възли. Тя включва резултатната променлива за поставени бизнес приложения D_w. Този резултат се подава към втората целева функция (MIN_RM), която минимизира броят на движенията на репликите през възлите, спрямо входната променлива δ, която се използва за лимитиране на движенията на репликите. За да запази броят на вече поставените бизнес приложения от първата целева функция (MAX_WD), има допълнително ограничение дефинирано като:

$$\sum_{w \in W} D_w = \text{result\_from\_(MAX\_WD)}.$$

По този начин втората целева функция (MIN_RM) надгражда първият резултат запазвайки броят на поставените реплики от първата целева функция (MAX_WD).

Целевата функция за минимизиране на движенията на репликите на микроуслугите през възлите между итерациите подлежи на няколко ограничения. Нека да дефинираме резултатната променлива за движението на репликите между итерациите като:

$$\Delta_{r,n}^{w,p} = \begin{cases} 0 & \text{ако } D_{r,n}^{w,p} = 1 \wedge (D_{r,n}^{w,p})^{-1} = 1 \\ & \text{ако } D_{r,n}^{w,p} = 0 \wedge (D_{r,n}^{w,p})^{-1} = 0 \\ 1 & \text{В противен случай,} \end{cases}$$

за $\forall w \in W, \forall p \in P, \forall r \in R_p^{max}, \forall n \in N$, където $(D_{r,n}^{w,p})^{-1}$ е матрицата на поставянето от предната итерация. Тук итерация означава предишното изпълнение на MILP моделът.

Тогава матрицата на движенията на репликите, която съдържа движенията на всички реплики, е дефинирана като:

$$\Delta_r^{w,p} = \begin{cases} 0 & \text{ако } \sum\limits_{n \in N} \Delta_{r,n}^{w,p} = 0 \\ 1 & \text{ако } \sum\limits_{n \in N} \Delta_{r,n}^{w,p} \geq 1, \end{cases}$$

за $\forall w \in W, \forall p \in P, \forall r \in R_p^{max}$.

Допълнително, ограничението за общият брой на възможните движения на реплики от един възел към друг възел, за всяка микроуслуга p в W е дефинирано чрез:

$$\sum_{p \in P} \sum_{r \in R_p^{max}} \Delta_r^{w,p} \times M_p^w \leq \delta \times \sum_{p \in P} R_p^{req} \times M_p^w,$$

за $\forall w \in W$. От това ограничение следва:

- $\delta \geq R_p^{max}$, ако искаме да не ограничим движенията на репликите между възли.
- $\delta < R_p^{max}$, ако искаме да ограничим движенията на репликите между възли.

Накрая, целевата функция за минимизиране на движенията на репликите на микроуслугите през възлите между итерациите се дефинира с:

$$\text{MIN\_RM} = min \sum_{w \in W} \sum_{p \in P} \sum_{r \in R} \Delta_r^{w,p}.$$

Минимизирането на движенията на репликите е полезно ако искаме да намалим движенията на репликите между итерациите, защото в някои случаи миграциите на реплики причиняват допълнително натоварване на инфраструктурата, което води до мрежово забавяне и то е нежелано при много динамична среда, в която възлите на клъстера се движат много често.

4.2.3 Минимизиране на мрежово закъснение на бизнес приложения (MIN_WL)

Резултатът от втората целева функция (MIN_RM) съдържа модифицираната схема на поставяне на репликите на микроуслугите върху множеството от възли. Отново тя включва резултатната променлива за поставени бизнес приложения D_w. Този резултат се подава към третата целева функция (MIN_WL), която цели да минимизира мрежовото закъснение на бизнес приложенията. За да запази броят на вече поставените бизнес приложения от първата и втората целеви функции (MAX_WD, MIN_RM), има допълнително ограничение дефинирано като:

$$\sum_{w \in W} D_w = \text{result\_from\_(MIN\_RM)}.$$

По този начин третата целева функция (MIN_WL) допълнително надгражда резултатът, но запазвайки броят на поставените реплики от предишните целеви функции.

Минимизиране на мрежовото закъснение на бизнес приложенията е подложена на няколко ограничения относно инфраструктурната мрежа. Първо, матрицата на потока между репликите описва специфичните мрежови потоци между всички реплики поставени на определени възли. Тя е дефинирана по следния начин:

$$S_{p_j,r_m}^{w,p_i,r_k}(n_u, n_v) = \begin{cases} 1 & \text{ако } D_{r_k,n_u}^{w,p_i} \wedge D_{r_m,n_v}^{w,p_j} = 1 \\ 0 & \text{В противен случай,} \end{cases}$$

за $\forall w \in W, \forall p_i, p_j \in P, i \neq j, \forall r_k, r_m \in R_p^{max}, \forall n_u, n_v \in N, u \neq v$.

Ограничението за осигуряване на минимален капацитет на потока между реплики е дефинирано като:

$$\sum_{w \in W} \sum_{p_i, p_j \in P} \sum_{r_k, r_m \in R_p^{max}} \sum_{n_u, n_v \in N} S_{p_j, r_m}^{w, p_i, r_k}(n_u, n_v) = \sum_{p_i, p_j \in P} \Omega_{p_i, p_j}^{\beta} \times R_{p_i}^{req} \times R_{p_j}^{req},$$

за $\forall w \in W, \forall p_i, p_j \in P, i \neq j, \forall r_k, r_m \in R_p^{max}, \forall n_u, n_v \in N, u \neq v$. То осигурява, че всички мрежови потоци между микроуслугите, които изискват гарантиран капацитет на трафика, са удовлетворени и няма загуби на трафик в клъстера.

Ограничението за осигуряване на максимално допустимо забавяне на потока между реплики е дефинирано като:

$$\sum_{w \in W} \sum_{p_i, p_j \in P} \sum_{r_k, r_m \in R_p^{max}} \sum_{n_u, n_v \in N} S_{p_j, r_m}^{w, p_i, r_k}(n_u, n_v) = \sum_{p_i, p_j \in P} \Omega_{p_i, p_j}^{\tau} \times R_{p_i}^{req} \times R_{p_j}^{req},$$

за $\forall w \in W, \forall p_i, p_j \in P, i \neq j, \forall r_k, r_m \in R_p^{max}, \forall n_u, n_v \in N, u \neq v$. То осигурява, че всички потоци между микроуслугите, които изискват гарантирано максимално допустимо мрежово закъснение, са удовлетворени и няма загуби на трафик в клъстера.

Нека дефинираме факторът за капацитет на трафик чрез:

$$s_{p_i, p_j}^{\beta} = M_{p_i}^{w} \times M_{p_j}^{w} \times \Omega_{p_i, p_j}^{\beta},$$

за $\forall p_i, p_j \in P, i \neq j$. Ще казваме $s_{p_i, p_j}^{\beta} = 1$ ако изходна микроуслуга p_i зависи на целева микроуслуга p_j и изискванията за минимален мрежов капацитет между тях трябва да бъдат гарантирано изпълнени.

Тогава ограничението за капацитет на потока е дефинирано като:

$$\sum_{w \in W} \sum_{p_i, p_j \in P} \sum_{r_k, r_m \in R_p^{max}} s_{p_i, p_j}^{\beta} \times \beta_{p_i, p_j} \times S_{p_j, r_m}^{w, p_i, r_k}(n_u, n_v) \leq B_{n_u, n_v},$$

за $\forall n_u, n_v \in N, u \neq v$. То валидира, че общият наличен мрежов капацитет в клъстера е съобразен.

Нека дефинираме факторът за забавяне в поток чрез:

$$s_{p_i, p_j}^{\tau} = M_{p_i}^{w} \times M_{p_j}^{w} \times \Omega_{p_i, p_j}^{\tau},$$

за $\forall p_i, p_j \in P, i \neq j$. Ще отбелязваме с $s_{p_i, p_j}^{\tau} = 1$ ако изходна микроуслуга p_i зависи на целева микроуслуга p_j и изискванията за максимално мрежово забавяне между тях трябва да бъдат гарантирано изпълнени.

Тогава ограничението за забавяне в потока е дефинирано като:

$$\sum_{w \in W} \sum_{p_i, p_j \in P} \sum_{r_k, r_m \in R_p^{max}} s^{\tau}_{p_i,p_j} \times \tau_{p_i,p_j} \times S^{w,p_i,r_k}_{p_j,r_m}(n_u, n_v) \leq T_{n_u,n_v},$$

за $\forall n_u, n_v \in N, u \neq v$. То валидира, че общото допустимо мрежово забавяне между възлите е удовлетворено.

Също така, матрицата на мрежовото забавяне на бизнес приложенията е дефинирана като:

$$T_w = \sum_{p_i, p_j \in P} \sum_{r_k, r_m \in R_p^{max}} \sum_{n_u, n_v \in N} T_{n_u,n_v} \times S^{w,p_i,r_k}_{p_j,r_m}(n_u, n_v),$$

за $\forall w \in W$. Тя показва общото закъснение на бизнес приложение w в милисекунди.

Накрая, целевата функция за минимизиране на мрежовото закъснение на бизнес приложенията е дефинирана чрез:

$$\text{MIN\_WL} = min \sum_{w \in W} T_w.$$

За да осигури минимално общо закъснение на приложенията, тази целева функция поставя взаимозависими микроуслуги върху възли с по-малко мрежово забавяне, обикновено разположени близо един до друг в пространството.

Резултатите, които са показани в тази глава, са публикувани в [35].

Глава 5

Резултати и дискусия

В тази глава два примера и резултат от времето за изпълнение (оперативно) ще бъдат показани за да демонстрират как описаният модел се справя с основните характеристики на Крайните/Мъглявинни платформи, идентифицирани в Глава 1, в реална мобилна среда. Примерите се провеждат в тестова постановка, която използва Kubernetes (версия 1.24.3) Облачна/Мъглявинна платформа и контейнерна виртуализация. Те имат за цел да валидират реалистично бизнес приложение, в лицето на приложението Ecologic [1]. Неговите взаимосвързани микроуслуги са показани във Фиг. 1.1. Примерите използват микроуслугите Database ($P1$), Backend ($P2$), VehicleAgent ($P3$) и Frontend ($P4$), без да се използва микроуслугата

Микроуслуга	**Време изпълнение резултат 1 реплики #**	**Време изпълнение резултат 2 реплики #**	**Време изпълнение резултат 3 реплики #**	**Пример E1 и E2 реплики #**
Database (P1)	1	1	1	1
Backend (P2)	1	2	2	2
VehicleAgent (P3)	1	3	6	6
Frontend (P4)	1	2	2	2

Таблица 5.1: Брой на микроуслуги (Pod) и реплики (Replica) в приложението EcoLogic.

Analytics. Общият брой на използваните реплики (инстанции) на микроуслугите е единадесет. Всяка реплика е разположена върху различен възел. Тестовата постановка емулира динамична среда, съставена от географски разпределени мобилни Крайни/Мъглявинни възли, движещи се в пространството. Конфигурацията е подобна на тази, показана във Фиг. 1.2, но са използвани единадесет възела вместо четири. Неговата цел е да оцени как предложеният модел намалява общата мрежова латентност на бизнес приложението в тази среда.

Пример 1 (E1). Разглеждаме единадесет географски града, в които възлите N са разположени, образувайки реалистична гео-разпределена клъстерна инфраструктура, подобна на тази показана във Фиг. 1.2. Всички възли се намират в различни региони U и зони Z. Броят им е единадесет, което е равно на общия брой използвани реплики на микроуслуги R от приложението Ecologic W. Броят на репликите R_p^{req} за конкретните микроуслуги P, използвани в примерите, е показан в Таблица 5.1. Заявеният брой реплики е равен на максималния брой реплики, разрешени от модела ($R_p^{req} = R_p^{max}$). На всеки възел има само една поставена реплика на микроуслуга. Действителните мрежови закъснения между всички двойки избрани градове се използват за емулиране на географски разпределена инфраструктура върху тестовата постановка. Закъснението между възлите T_{n_u,n_v} варира от 10 до 60 милисекунди. Средното първоначално разстояние между възли x1 е 800 километра. На всяка стъпка всички възли се отдалечават един от друг с фиксирано разстояние Δx (Δx = x2 − x1). В **E1** Δx е равно на 10% от средната първоначална дистанция (80 километра). Използваме 20 стъпки за движение, което е достатъчно дълъг период за анализ на преместването на възлите и промените в характеристиките на мрежата. Всички микроуслуги изискват процесорни ресурси c_p и памет e_p от 100 millicpu и 100 Mb, съответно. Използвани са специфични закъснения τ_{p_i,p_j} и капацитет на трафика β_{p_i,p_j} между всяка двойка зависими микроуслуги със стойности от 30 милисекунди и 100 Mbps, съответно. Движенията на репликите през възлите не са ограничени ($\delta = R_p^{max} = 6$). Моделът за Смесено Целочислено Линейно Програмиране (MILP) се изпълнява на всяка стъпка на движение (итерация). На всяка стъпка, една и съща стойност на входните променливи се прилага, с изключение на забавянето между възлите (T_{n_u,n_v}), което се променя, за да съответства на фиксираното разстояние на преместване Δx. В края на всяка итерация MILP моделът връща схема на поставяне ($D_{r,n}^{w,p}$) и репликите

на микроуслугите се разполагат върху конкретни възли.

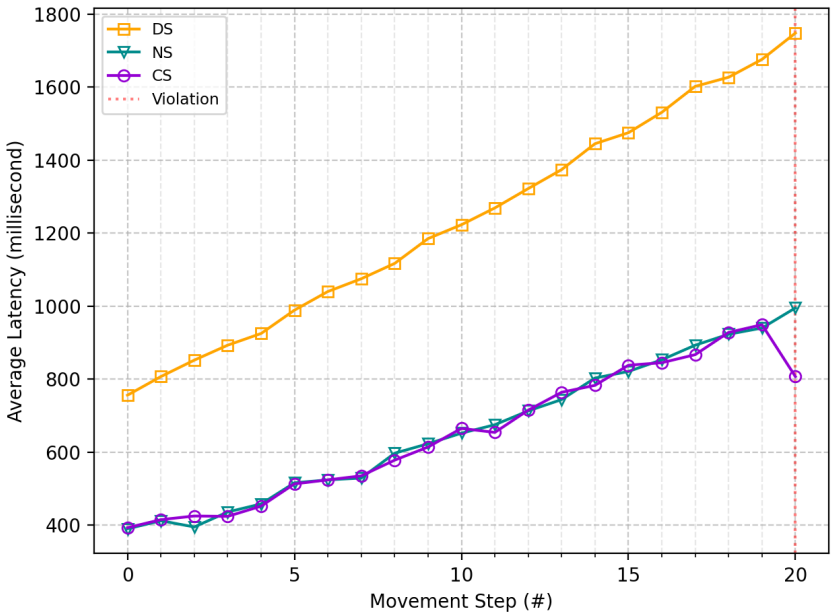
В зависимост от фиксираното разстояние на движение, изискването за максимално допустимо забавяне между микроуслугите τ_{p_i,p_j} трябва да бъде достатъчно ограничаващо, за да причини нарушения на изискванията за мрежата и да доведе до последващи премествания на микроуслуги и оптимизиране. Въпреки това, ако неговата стойност е твърде ограничаваща, може да възникне ситуация, при която не съществуват възможни възли за преместване. В този случай моделът ще остави текущата схема на поставяне на микроуслуги ($D^{w,p}_{r,n}$) в нейното последно състояние. Настоящо разположените бизнес приложения ще бъдат достъпни, но с намалено качество на услугата — общото им закъснение ще бъде увеличено и ще надхвърля конфигурирания лимит. Тази ситуация ще бъде разрешена ако възлите се придвижат до по-благоприятни местоположения или нови допълнителни възли се присъединят към клъстерната инфраструктура. Когато изискването за забавяне в мрежата е избрано правилно в тази гранична зона, валидацията може да бъде използвана за анализ на нивото на оптимизация на MILP модела в сравнение с други решения. Това се взема предвид в следващия пример.

Пример 2 (E2). Архитектурата и параметрите са същите както в **E1**, показани в Таблица 5.1. Единствената разлика е, че възлите се отдалечават един от друг с два пъти по-голямо фиксирано разстояние Δx, отколкото в **E1**, което е равно на 20% от средната първоначална дистанция (160 километра).

Резултатите от примерите **E1** и **E2** са визуализирани във Фиг. 5.1 и Фиг. 5.2, съответно. Фигурите съдържат стъпките на движение по абсцисната ос и средното общо закъснение на бизнес приложението в милисекунди по ординатната ос. Има три графики, представящи вградения по подразбиране планировчик на Kubernetes (DS), мрежово-чувствителен планировчик (NS) и планировчик с MILP модел (CS). Нарушенията на изискванията за мрежовите забавяния са показани като вертикални пунктирани линии на съответната графика на движение. Схемите на поставяне на NS и CS планировчиците се различават когато има нарушение, което води до различие в резултатите им за средно закъснение.

В резултатите на пример **E1** (Фиг. 5.1), NS и CS имат по-ниско общо мрежово закъснение в сравнение с DS. NS и CS имат почти едно и също закъснение до последната стъпка поради еднакви начални схеми на поставяне в първата стъпка. Има нарушение на изискването за мрежово забавяне, случващо се на стъпка 20, което води до преместване на микроуслуги и по-ниско закъснение за CS в сравнение с NS, намалено с 20%. Късното оптимизиране в последната стъпка означава, че изискването за забавяне в мрежата τ_{p_i,p_j} е твърде неограничаващо.

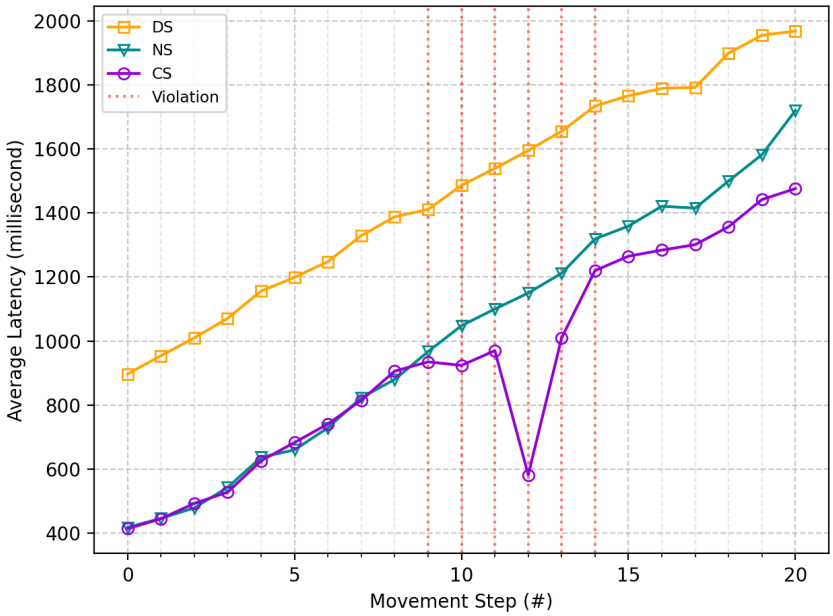
Пример **E2** премества избраните възли с по-голямо разстояние. Резултатът (Фиг. 5.2) показва, че NS и CS имат по-малко общо мрежово закъснение в сравнение с DS. Нарушенията започват рано в стъпка 9 и продължават до стъпка 14. CS има по-малко закъснение от NS, достигайки най-голямата разлика от повече от 500 милисекунди (приблизително 48% подобрение) в стъпка 12. Този пример има по-ограничаващо изискване за мрежово забавяне τ_{p_i,p_j} спрямо по-голямото разстояние на преместване, в сравнение с пример **E1**. Този резултат показва много



Фигура 5.1: Пример **E1** с малко разстояние на преместване Δx = 10%(80km) на стъпка. CS оптимизира общото забавяне с 20% в последната стъпка 20, сравнен спрямо съществуващи K8s планировчици.

по-голямо подобрение на общото закъснение.

Като допълнение, резултатите от средното време за изпълнение на модела в сравнение с другите обсъдени K8s планировчици, са представени във Фиг. 5.3. Фигурата съдържа общия брой използвани реплики на микроуслуги на бизнес приложението Ecologic по абсцисната ос и времето за изпълнение по ординатната ос. Има три графики, представящи вградения по подразбиране планировчик на Kubernetes (DS), мрежово-чувствителен планировчик (NS) и планировчик с MILP модел (CS). Също има и 30-минутен лимит, изобразен като хоризонтална прекъсната линия. Броят на използваните реплики на микроуслуги на приложението Ecologic са показани в Таблица 5.1. Има три измервания с общ брой реплики от 4, 8 и 11, съответно. DS има време за изпълнение от 0.06 до 0.12 секунди и NS - 0.07 до 0.13 секунди. За разлика от тях, CS има много по-дълго време за изпълнение от 27 до 117 минути. Това се дължи на MILP моделът, който отнема много време за намиране на най-оптималното решение за поставяне, както е доказано в горните резултати Фиг. 5.1, Фиг. 5.2. Обикновено, много приложения в Облачните/Мъглявинни платформи, които са критични и работят в реално време, трябва да отговарят на максимално допустимо време за изпълнение от 30 минути.

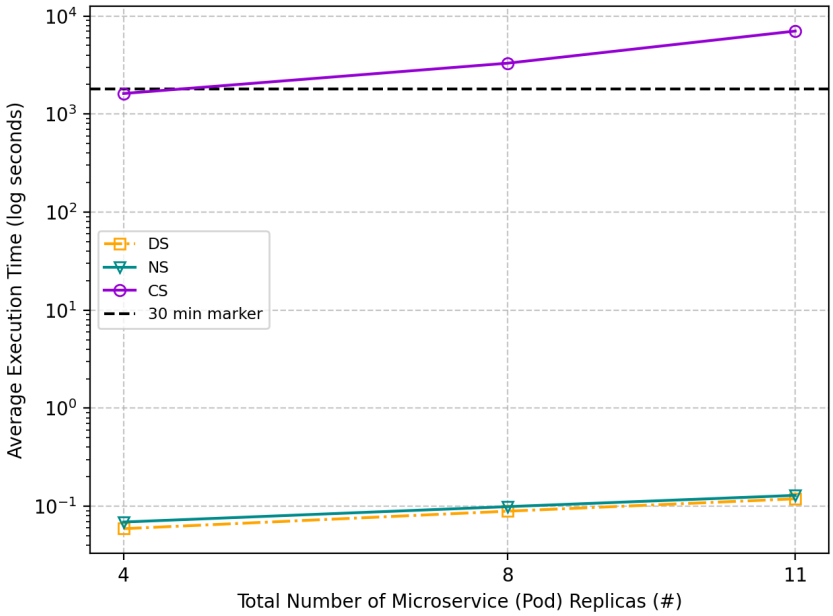


Фигура 5.2: Пример **E2** с голямо разстояние на преместване $\Delta x = 20\%(160km)$ на стъпка. CS оптимизира общото забавяне с до 48% започващо рано от стъпка 9, сравнен спрямо съществуващи K8s планировчици.

5.1 Дискусия

Получените резултати във Фиг. 5.1 и Фиг. 5.2 потвърждават, че в динамична среда с мобилни възли моделът поставя и премества микроуслуги на възли, които са близко един до друг като се приспособява към движенията им в пространството. По този начин общото мрежово закъснение на бизнес приложението се поддържа минимално, дългосрочно по време на изпълнението. В практиката преместването на реплики между възли може да причини прекъсване на работата на преместените реплики, докато тези, които не са преместени, ще продължат да работят без прекъсване. В този случай, факторът за преместване δ може да бъде допълнително конфигуриран за да се намали общият брой премествания на реплики и свързаните с тях нежелани прекъсвания. Освен това, периодът между изпълняването на отделните итерации на моделът може да бъде конфигуриран според честотата (скоростта), с която се движат възлите. Периодът между итерациите трябва да е обратно пропорционален на скоростта на движение. Изискванията за мрежово закъснение на микроуслугите (τ_{p_i,p_j}) трябва да бъдат достатъчно ограничаващи за да предизвикат оптимизации на поставянето на микроуслуги, в противен случай микроуслугите няма да бъдат преместени.

Резултатите на времето за изпълнение във Фиг. 5.3 показват, че MILP моделът отнема много време за да предостави най-оптималното решение, което е повече от



Фигура 5.3: Времето за изпълнение на CS с MILP е много по-голямо (27 до 117 мин), сравнено спрямо съществуващи K8s планировчици (0.06 до 0.13 сек).

30-минутния праг, използван от някои критични приложения в практиката, които трябва да работят в реално време. Това означава, че може да се прилага на практика в производствени среди като се изпълнява по-рядко, например веднъж или два пъти дневно. Въпреки това MILP моделът може да се използва като еталон за сравнение колко оптимални са други алгоритми, базирани на евристика, които могат да бъдат по-бързи. Освен това е универсален и може да се прилага на практика в широк спектър от Облачни/Мъглявинни сценарии на употреба. Всички входни променливи могат да бъдат конфигурирани от платформените инженери в съответствие с капацитета на ресурсите на инфраструктурата, моделите на движение и изискванията на бизнес приложенията.

Резултатите, които са показани в тази глава, са публикувани в [35].

Глава 6

Заключение

В последните години масовото внедряване на мобилни хетерогенни устройства с изчислителни възможности доведе до необходимостта от нови техники за динамично разпределение на ресурсите. Тази дисертация представи MILP алгоритъм за разполагане на контейнери в Облачни/Мъглявинни инфраструктури, съставени от възли, които се движат във времето и пространството. Той е подходящ за динамични и недетерминистични IoT сценарии, които съдържат мобилни обекти като превозни средства, сателити и въздушни транспортни средства. Използва няколко целеви функции, опитвайки се да увеличи броя на изпълняваните бизнес приложения, да намали миграциите им между възли и да подобри общото мрежово закъснение. Два примера с практическото приложение базирано на микроуслуги Ecologic [1] са показани. Тестовата постановка емулира реално движение на изчислителни възли между географски региони. Получените резултати показват, че решението има обещаваща перспектива да доведе до големи подобрения в общото мрежово закъснение на приложенията в практически Облачни/Крайни/Мъглявинни среди с мобилни възли. Въпреки големите времена за изпълнение, MILP моделът е универсален и може да служи като еталон за изследване и оценка на алгоритми за разпределение на ресурси в широк спектър от динамични и мобилни среди. Може да се адаптира към различни сценарии на употреба в зависимост от типовете ресурси на инфраструктурата, зависимостите на бизнес приложенията и моделите на движение.

Научни приноси

Целта на дисертацията е да предостави решение за оптимално управление на ресурси в съвременни Облачни/Крайни/Мъглявинни платформи, позволявайки из-

пълнението на сложни IoT приложения работещи в реално време върху клъстер от устройства с ограничени изчислителни ресурси, които се движат в пространството. То помага за по-добрата поддръжка на сценарии като свързани превозни средства, космически изчисления, добавена/виртуална реалност, излъчване на аудио/видео в реално време и др. Решението е внедрено в най-популярната Облачна/Крайна/Мъглявинна платформа в практиката, наречена Kubernetes, и е валидирано с комплексно IoT приложение в реална практическа среда.

Дисертацията съдържа следните приноси:

- MILP моделът, предложен в [36], е разширен с нова матрица на закъснението, която дефинира изискването за мрежово забавяне във връзките между микроуслуги (τ_{p_i,p_j}), нова променлива за наличност на възел в даден регион (U) и целева функция за минимизиране на движението на реплики между мобилни възли (MIN_RM).

- Векторите за капацитет и изисквания са заменени с директни променливи (B_n, C_n, E_n, b_p, c_p, e_p) използвани в ограниченията и целевите функции на описания MILP модел.

- Стъпките на изпълнение на MILP моделът са модифицирани за да се поддържа динамична непрекъсната оптимизация на реплики с подвижни възли. Стъпките съдържат фази и итерации.

- MILP оптимизационният модел е имплементиран в реална Облачна/Крайна/Мъглявинна платформа (Kubernetes).

- Алгоритъмът и платформата са валидирани в тестова постановка.

- Проектиране и имплементиране на реално практическо IoT приложение, наречено Ecologic [1], за следене и контрол на въглеродните емисии от превозни средства, подходящо за работа в сценарии на интелигентни градове.

- Валидиране на алгоритъма на Ecologic за анализ на данни, който открива превозни средства, които замърсяват прекомерно въздуха.

- Създаване на публично достъпни хранилища с отворен код, съдържащи програмен код и информация на основните микроуслуги на Ecologic.

- Извършване на валидация в тестова постановка, използвайки Ecologic като тестово приложение и правейки емулация на движение на възли в клъстер.

- Получените резултати от извършената валидация с бизнес приложението Ecologic показват намаление в общото му мрежово закъснение до 48% спрямо най-съвременните разработки в областта.

Апробация на дисертационната работа

1. T. Tsokov and H. Kostadinov, “System for monitoring and control of vehicle’s carbon emissions using embedded hardwares and cloud applications”, in Service-Oriented Computing - ICSOC 2020 Workshops, H. Hacid, F. Outay, H.-y. Paik, et al., Eds., Cham: Springer International Publishing, 2021, pp. 564-577, isbn: 978-3-030-76352-7. doi: https://doi.org/10.1007/978-3-030-76352-7_50. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-76352-7_50; SJR (Q2).

2. T. Tsokov and H. Kostadinov, “Dynamic network-aware container allocation in Cloud/Fog computing with mobile nodes”, Internet of Things, p. 101 211, 2024, issn: 2542-6605. doi: https://doi.org/10.1016/j.iot.2024.101211. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2542660524001525; Impact factor (Q1).

Доклади на научни форуми

1. T. Tsokov and H. Kostadinov, "Iot System for Monitoring and Control of Vehicle’s Carbon Emissions Using Embedded Hardwares and Cloud Applications 32nd National Seminar on Coding Theory "Acad. Stefan Dodunekov Chiflik, Troyan District, 10 October 2020.

2. T. Tsokov and H. Kostadinov, "System for Monitoring and Control of Vehicle’s Carbon Emissions Using Embedded Hardwares and Cloud Applications International Workshop on Artificial Intelligence in the IoT Security Services (Service-Oriented Computing - ICSOC 2020 Workshops), Dubai, 14-17 December 2020.

3. T. Tsokov, "IoT fog platform Problems and Methods Related to Coding Theory, Sofia, 08 February 2022.

Благодарности и посвещение

- Бих искал да изразя своята благодарност към моят ръководител, доц. Христо Костадинов, за неговата ценна помощ и принос към дисертацията и публикациите. Винаги ще бъда благодарен за уменията, които ми е предал в областта на математиката, науката, инженерството и за непрекъснатото му техническо и лично подпомагане.
- Бих искал да изразя благодарността си към д-р Константин Делчев, който ми предостави няколко Raspberry Pi устройства, използвани в проекта.
- Бих искал да изразя благодарността си към Института по Математика и Информатика към Българска Академия на Науките за възможността да работя в тяхната изследователска среда.
- Тази работа е частично подкрепена от Фонд "Научни Изследвания"на България, договор КР-06-N32/2-2019.
- Тази работа е подкрепена от Министерство на Образованието и Науката на България („Национален Център за Високопроизводителни и Разпределени Изчисления“, договор No. DO1-325/01.12.2023) и Оперативна Програма "Наука и Образование за Интелигентен Растеж“ в България (договор No. BG05M2OP001-1.001-0003).
- Изразявам благодарността си към моите родители, брат ми и баба ми, за тяхната подкрепа и съвети, когато имах най-голяма нужда от тях.

Библиография

[1] T. Tsokov и H. Kostadinov, “System for Monitoring and Control of Vehicle’s Carbon Emissions Using Embedded Hardwares and Cloud Applications”, в

Service-Oriented Computing – ICSOC 2020 Workshops, H. Hacid, F. Outay, H.-y. Paik и др., ред., Cham: Springer International Publishing, 2021, с. 564—577, ISBN: 978-3-030-76352-7. DOI: 10.1007/978-3-030-76352-7_50. url: https://doi.org/10.1007/978-3-030-76352-7_50.

[2] A. Laghari, A. Jumani и R. Laghari, "Review and State of Art of Fog Computing", *Archives of Computational Methods in Engineering*, т. 28, февр. 2021. DOI: 10.1007/s11831-020-09517-y.

[3] R. Das и M. Muhammad Inuwa, "A review on Fog Computing: Issues, Characteristics, Challenges, and Potential Applications", *Telematics and Informatics Reports*, т. 10, с. 100 049, февр. 2023. DOI: 10.1016/j.teler.2023.100049.

[4] R. Mahmud, R. Kotagiri и R. Buyya, "Fog Computing: A Taxonomy, Survey and Future Directions", в *Internet of Everything: Algorithms, Methodologies, Technologies and Perspectives*, B. Di Martino, K.-C. Li, L. T. Yang и A. Esposito, ред. Singapore: Springer Singapore, 2018, с. 103—130, ISBN: 978-981-10-5861-5. DOI: 10.1007/978-981-10-5861-5_5. url: https://doi.org/10.1007/978-981-10-5861-5_5.

[5] A. Bhardwaj и C. R. Krishna, "Virtualization in Cloud Computing: Moving from Hypervisor to Containerization—A Survey", *Arabian Journal for Science and Engineering*, т. 46, № 9, с. 8585—8601, апр. 2021. DOI: 10.1007/s13369-021-05553-3. url: https://doi.org/10.1007/s13369-021-05553-3.

[6] C. Dall, S.-W. Li, J. T. Lim, J. Nieh и G. Koloventzos, "ARM Virtualization: Performance and Architectural Implications", *SIGARCH Comput. Archit. News*, т. 44, № 3, с. 304—316, юни 2016, ISSN: 0163-5964. DOI: 10.1145/3007787.3001169. url: https://doi.org/10.1145/3007787.3001169.

[7] K. Sivaprakasam, P. Sriramalakshmi, P. Singh и M. Bhaskar, "Chapter 4 - An overview of low power hardware architecture for edge computing devices", в *5G IoT and Edge Computing for Smart Healthcare*, пор. Intelligent Data-Centric Systems, A. K. Bhoi, V. H. C. de Albuquerque, S. N. Sur и P. Barsocchi, ред., Academic Press, 2022, с. 89—109, ISBN: 978-0-323-90548-0. DOI: https://doi.org/10.1016/B978-0-323-90548-0.00004-8. url: https://www.sciencedirect.com/science/article/pii/B9780323905480000048.

[8] J. Alonso, L. Orue-Echevarria, V. Casola и др., "Understanding the challenges and novel architectural models of multi-cloud native applications – a systematic literature review", *J. Cloud Comput.*, т. 12, № 1, ян. 2023, ISSN: 2192-113X. DOI: 10.1186/s13677-022-00367-6. url: https://doi.org/10.1186/s13677-022-00367-6.

[9] "Edge-Native application principles". (2024-06-21), url: https://www.cncf.io/reports/edge-native-application-design-behaviors-whitepaper/ (дата на посещ. 21.06.2024).

[10] M. Jansen, A. Al-Dulaimy, A. V. Papadopoulos, A. K. Trivedi и A. Iosup, "The SPEC-RG Reference Architecture for The Compute Continuum", *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, с. 469—484, 2022. url: https://api.semanticscholar.org/CorpusID:257280275.

[11] A. Aral и T. Ovatman, "A Decentralized Replica Placement Algorithm for Edge Computing", *IEEE Transactions on Network and Service Management*, т. 15, № 2, с. 516—529, юни 2018, ISSN: 1932-4537. DOI: 10.1109/TNSM.2017.2788945.

[12] Y. Shao, C. Li и H. Tang, "A data replica placement strategy for IoT workflows in collaborative edge and cloud environments", *Computer Networks*, т. 148, с. 46—59, 2019, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2018.10.017. url: https://www.sciencedirect.com/science/article/pii/S1389128618311460.

[13] V. B. C. Souza, W. Ramírez, X. Masip-Bruin, E. Marín-Tordera, G.-J. Ren и G. Tashakor, "Handling service allocation in combined Fog-cloud scenarios", *2016 IEEE International Conference on Communications (ICC)*, с. 1—5, 2016.

[14] K. Zhang, M. Peng и Y. Sun, "Delay-Optimized Resource Allocation in Fog-Based Vehicular Networks", *IEEE Internet of Things Journal*, т. 8, № 3, с. 1347—1357, февр. 2021, ISSN: 2327-4662. DOI: 10.1109/JIOT.2020.3010861.

[15] S. Hassan, I. Ahmad, A. Rehman, S. Hussen и H. Hamam, "Design of Resource-Aware Load Allocation for Heterogeneous Fog Computing Environments", *Wireless Communications and Mobile Computing*, т. 2022, юни 2022. DOI: 10.1155/2022/3543640.

[16] L. Li, Q. Guan, L. Jin и M. Guo, "Resource Allocation and Task Offloading for Heterogeneous Real-Time Tasks With Uncertain Duration Time in a Fog Queueing System", *IEEE Access*, т. 7, с. 9912—9925, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2891130.

[17] R. N. Calheiros, R. Ranjan, C. A. F. D. Rose и R. Buyya, *CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services*, 2009. arXiv: 0903.2525 [cs.DC].

[18] C. Sonmez, A. Ozgovde и C. Ersoy, "EdgeCloudSim: An environment for performance evaluation of Edge Computing systems", *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, с. 39—44, 2017.

[19] H. Gupta, A. V. Dastjerdi, S. K. Ghosh и R. Buyya, *iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in Internet of Things, Edge and Fog Computing Environments*, 2016. arXiv: 1606.02007 [cs.DC].

[20] P. Mugglestone. "SAP HANA Academy". (2014-05-13), url: https://github.com/saphanaacademy/PAL/tree/master/Source%20Data/PAL (дата на посещ. 07.06.2024).

[21] J. B. Jr., B. Costa, L. R. Carvalho, M. J. F. Rosa и A. Araujo, "Computational Resource Allocation in Fog Computing: A Comprehensive Survey", *ACM Comput. Surv.*, март 2023, Just Accepted, ISSN: 0360-0300. DOI: 10.1145/3586181. url: https://doi.org/10.1145/3586181.

[22] A. Beltre, P. Saha и M. Govindaraju, "KubeSphere: An Approach to Multi-Tenant Fair Scheduling for Kubernetes Clusters", в *2019 IEEE Cloud Summit*, авг. 2019, с. 14—20. DOI: 10.1109/CloudSummit47114.2019.00009.

[23] D. Crankshaw, G.-E. Sela, X. Mo и др., "InferLine: Latency-Aware Provisioning and Scaling for Prediction Serving Pipelines", в *Proceedings of the 11th ACM Symposium on Cloud Computing*, пор. SoCC '20, Virtual Event, USA: Association for Computing Machinery, 2020, с. 477—491, ISBN: 9781450381376. DOI: 10.1145/3419111.3421285. url: https://doi.org/10.1145/3419111.3421285.

[24] A. J. Fahs и G. Pierre, "Tail-Latency-Aware Fog Application Replica Placement", в *Service-Oriented Computing: 18th International Conference, ICSOC 2020, Dubai, United Arab Emirates, December 14–17, 2020, Proceedings*, Dubai, United Arab Emirates: Springer-Verlag, 2020, с. 508—524, ISBN: 978-3-030-65309-5. DOI: 10.1007/978-3-030-65310-1_37. url: https://doi.org/10.1007/978-3-030-65310-1_37.

[25] A. J. Fahs, G. Pierre и E. Elmroth, "Voilà: Tail-Latency-Aware Fog Application Replicas Autoscaler", в *2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2020, с. 1—8. DOI: 10.1109/MASCOTS50786.2020.9285953.

[26] X. Li, J. Zhou, X. Wei и др., "Topology-Aware Scheduling Framework for Microservice Applications in Cloud", *IEEE Transactions on Parallel and Distributed Systems*, т. PP, с. 1—17, май 2023. DOI: 10.1109/TPDS.2023.3238751.

[27] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee и M. Zaharia, "Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads", в *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*, пор. OSDI'20, USA: USENIX Association, 2020, ISBN: 978-1-939133-19-9.

[28] V. Rao, V. Singh, K. S. Goutham и др., “Scheduling Microservice Containers on Large Core Machines Through Placement and Coalescing”, в *Job Scheduling Strategies for Parallel Processing: 24th International Workshop, JSSPP 2021, Virtual Event, May 21, 2021, Revised Selected Papers*, Berlin, Heidelberg: Springer-Verlag, 2021, с. 80—100, ISBN: 978-3-030-88223-5. DOI: 10.1007/978-3-030-88224-2_5. url: https://doi.org/10.1007/978-3-030-88224-2_5.

[29] J. Santos, T. Wauters, B. Volckaert и F. De Turck, “Towards end-to-end resource provisioning in Fog Computing over Low Power Wide Area Networks”, *Journal of Network and Computer Applications*, т. 175, с. 102 915, 2021, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2020.102915. url: https://www.sciencedirect.com/science/article/pii/S108480452030374X.

[30] C. Li, Y. Wang, H. Tang, Y. Zhang, Y. Xin и Y. Luo, “Flexible replica placement for enhancing the availability in edge computing environment”, *Computer Communications*, т. 146, с. 1—14, 2019, ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2019.07.013. url: https://www.sciencedirect.com/science/article/pii/S0140366418308296.

[31] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer и K. K. Leung, “Dynamic Service Placement for Mobile Micro-Clouds with Predicted Future Costs”, *IEEE Transactions on Parallel and Distributed Systems*, т. 28, № 4, с. 1002—1016, апр. 2017, ISSN: 1558-2183. DOI: 10.1109/TPDS.2016.2604814.

[32] K. Toczé и S. Nadjm-Tehrani, “ORCH: Distributed Orchestration Framework using Mobile Edge Devices”, в *2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC)*, 2019, с. 1—10. DOI: 10.1109/CFEC.2019.8733152.

[33] K. Toczé, A. J. Fahs, G. Pierre и S. Nadjm-Tehrani, “VioLinn: Proximity-Aware Edge Placement with Dynamic and Elastic Resource Provisioning”, *ACM Trans. Internet Things*, т. 4, № 1, февр. 2023, ISSN: 2691-1914. DOI: 10.1145/3573125. url: https://doi.org/10.1145/3573125.

[34] A. Haider, R. Potter и A. Nakao, “Challenges in resource allocation in network virtualization”, в *20th ITC specialist seminar*, ITC, т. 18, ян. 2009. url: https://api.semanticscholar.org/CorpusID:10681804.

[35] T. Tsokov и H. Kostadinov, “Dynamic network-aware container allocation in Cloud/Fog computing with mobile nodes”, *Internet of Things*, с. 101 211, 2024, ISSN: 2542-6605. DOI: https://doi.org/10.1016/j.iot.2024.101211. url: https://www.sciencedirect.com/science/article/pii/S2542660524001525.

[36] J. Santos, C. Wang, T. Wauters и F. D. Turck, “Diktyo: Network-Aware Scheduling in Container-based Clouds”, *IEEE Transactions on Network and Service Management*, с. 1—1, 2023. DOI: 10.1109/TNSM.2023.3271415.