Raspoređivanje kontekstno kategoriziranih usluga interneta stvari u okolini računarstva u magli

Krivić, Petar

Sveučilište u Zagrebu

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Petar Krivić

RASPOREĐIVANJE KONTEKSTNO KATEGORIZIRANIH USLUGA INTERNETA STVARI U OKOLINI RAČUNARSTVA U MAGLI

DOKTORSKI RAD

Zagreb, 2022.

Sveučilište u Zagrebu

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Petar Krivić

RASPOREĐIVANJE KONTEKSTNO KATEGORIZIRANIH USLUGA INTERNETA STVARI U OKOLINI RAČUNARSTVA U MAGLI

DOKTORSKI RAD

Mentor: Prof. dr. sc. Mario Kušek

Zagreb, 2022.

University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Petar Krivić

SCHEDULING OF CONTEXTUALLY CATEGORISED INTERNET OF THINGS SERVICES IN FOG COMPUTING ENVIRONMENT

DOCTORAL THESIS

Supervisor: Professor Mario Kušek, PhD

Zagreb, 2022

Doktorski rad izrađen je na Sveučilištu u Zagrebu Fakultetu elektrotehnike i računarstva, na Zavodu za telekomunikacije.

Mentor: prof. dr. sc. Mario Kušek

Doktorski rad ima: 112 stranica

Doktorski rad br.: _____

O mentoru

Mario Kušek rođen je 1972. godine u Zagrebu. Diplomirao je, magistrirao i doktorirao u polju elektrotehnike na Sveučilištu u Zagrebu Fakultetu elektrotehnike i računarstva (FER), 1997., 2001. odnosno 2005. godine.

Od 1997. godine radi na Zavodu za telekomunikacije FER-a. 2017. godine izabran je u zvanje redoviti profesor. Sudjelovao je ili sudjeluje na 2 znanstvena projekta Ministarstva znanosti obrazovanja i sporta Republike Hrvatske i 2 EU COST akcije, 2 projekta Hrvatske zaklade za znanost, 1 iz Obzora 2020, 2 IRI projekta, jednom bilateralnom projektu s institutom Telecommunications Research Center Vienna (FTW) te mnogim projektima s industrijom. Vodio je istraživačke projekte s tvrtkama Ericsson Nikola Tesla i Kate-Kom, a trenutno vodi jedan IRI2 projekt. Objavio je više od 90 radova u časopisima, zbornicima i knjigama u području raspodijeljenih sustava, višeagentskih sustava, samoorganizirajućih sustava u komunikaciji stroja sa strojem i Interneta stvari koji se rješavaju u okolini oblaka i na rubu pomoću mikrousluga.

Prof. Mario Kušek član je stručne udruge IEEE u kojoj je trenutno na poziciji koordinatora konferencija u Hrvatskoj sekciji IEEE-a, organizacije KES International i standardizacijske organizacije ETSI. Sudjeluje u 4 međunarodna programska odbora znanstvenih konferencija, kao programski kopredsjedatelj na međunarodnoj konferenciji KES AMSTA i kao glavni kopredsjedatelj na međunarodnoj konferenciji ConTEL, te je recenzent u većem broju međunarodnih konferencija i časopisa.

About the Supervisor

Mario Kušek was born in Zagreb in 1972. He received B.Sc., M.Sc. and Ph.D. degrees in electrical engineering from the University of Zagreb, Faculty of Electrical Engineering and Computing (FER), Zagreb, Croatia, in 1997, 2001 and 2005, respectively.

Since 1997 he has been working at the Department of telecommunications at FER. In 2017. he was promoted to Full Professor. He participated (or is participating) in 2 scientific projects financed by the Ministry of Science, Education and Sports of the Republic of Croatia and 2 EU COST actions, 2 projects by Croatian Science Fundation, 2 IRI projects, one bilateral project with The Telecommunications Research Center Vienna (FTW). In addition, he led research projects funded by companies Ericsson Nikola Tesla, Kate-Kom and Agrokor, and is currently leading one IRI project. He published more than 90 papers in journals, conference proceedings and books in the area of distributed systems, multi-agent systems, self-organized systems, machine-to-machine (M2M) Communications, and Internet of Things in the cloud and edge environment by using microservices.

Prof. Kušek is a member of IEEE, currently serving as Conference Coordinator of the IEEE Croatia Chapter, the KES International and standardization organization ETSI. He is participating in 4 conference international program committees. He serves as a program co-chair on the international conference KES AMSTA and general co-chair on the international conference ConTEL. Also, he is a technical reviewer for various international journals and conferences.

Zahvaljujem najprije mentoru prof. dr. sc. Mariju Kušeku na pomoći i savjetima tijekom istraživačkog rada i pisanja ove disertacije, a zatim i svim mojim kolegama na FER-u koji su mi pritom pružili pomoć, podršku, i ugodnu radnu atmosferu. Također, zahvaljujem se i svim mojim prijateljima na motivaciji i potpori tijekom prethodnih godina, a posebno hvala mojoj obitelji i djevojci Ani na njihovom strpljenju, ljubavi i bezuvjetnoj podršci.

Sažetak

Računarstvo u magli predstavlja novi koncept arhitekture za isporuku usluga u kojem se osim računalnog oblaka koriste i resursi računalnih čvorova između računalnog oblaka i krajnjih uređaja. Primjenom ovog koncepta omogućuje se optimizacija rješenja zbog blizine izvora podataka i čvorova za njihovu obradu, što je zbog karakteristične okoline posebno naglašeno u okviru Interneta stvari. Virtualizacija zasnovana na kontejnerima i druge moderne tehnologije omogućili su implementaciju raspodijeljenih IoT usluga koje se mogu efikasno izvoditi u promjenjivim heterogenim okolinama kakve podrazumijeva računarstvo u magli. Ipak, raspored izvođenja komponenti IoT usluga u raspodijeljenim okolinama ovoga koncepta ovisit će o optimizaciji koja se u konkretnom slučaju želi postići njegovom primjenom pa je upravo određivanje optimalnog rasporeda izvođenja usluga na dostupnim računalnim resursima računarstva u magli fokus ove disertacije. Stoga su u ovom radu najprije kategorizirane usluge Interneta stvari prema kontekstima relevantnim za primjenu računarstva u magli: kontekstu uređaja, korisnika, i usluge. Također, definirani su i parametri kvalitete usluge kojima se može ocijeniti učinkovitost primjene računarstva u magli te je određen njihov prioritet u prethodno definiranim kategorijama IoT usluga. Zatim je opisan formalni model usluga i specificiranih konteksta, kako bi se faktori koji utječu na izvedbu raspoređivanja usluga mogli uzeti u obzir u konkretnoj implementaciji algoritma. Tako se u razvijenom dinamičkom algoritmu raspoređivanja na temelju kontekstnih informacija o usluzi, korisniku, i dostupnim uređajima određuje raspored izvođenja komponenti usluge kojim se postiže veća kvaliteta njenog izvođenja u okviru definiranih parametara kvalitete usluge. Na kraju je izvedena verifikacija razvijenog dinamičkog algoritma kako bi se utvrdila njegova učinkovitost na odabranim slučajevima primjene. Dobiveni rezultati su potvrdili bolje performanse izvođenja usluga u okolini računarstva u magli primjenom razvijenog algoritma za dva odabrana scenarija (strujanje podataka i automatizaciju upravljanja). Također, rezultati su pokazali da je glavna prednost ovog algoritma u odnosu na postojeće pristupe, njegova mogućnost dinamičkog pokretanja raspoređivanja na temelju promjena u izvedbenoj okolini koje uzrokuju pad kvalitete usluge.

Ključne riječi: računarstvo u magli, Internet stvari, kontekstno-svjesni algoritam raspoređivanja, kontejnerska virtualizacija, dinamičko raspoređivanje, QoS

Extended abstract

Scheduling of contextually categorised Internet of Things services in fog computing environment

The concept of fog computing emerged as a novel architecture that could improve the efficiency of service delivery depending on the characteristics of a specific service scenario, particularly in the context of latency, reliability, security, and network traffic reduction. The extensive development of broadly available computing devices and the perpetual need to optimize the general efficiency in service delivery resulted in the idea to utilize local processing capacities in addition to the commonly used centralized cloud processing model. Offloading a certain amount of processing to the local environments reduces the amount of traffic in the public network and eliminates the propagation latency by default. Still, the realization of distributed processing presents a complex challenge that is, in this case, further emphasized by the volatility within the fog environment.

Internet of Things (IoT) also implies a distributed architecture, where processing and storage are usually performed within the cloud environment. However, the interaction with the physical world, which is the basis of IoT scenarios, is executed in most cases across the targeted local domains where fog processing could be applied to enhance the overall efficiency of service performance. Still, to efficiently achieve this goal, it was necessary to overcome the heterogeneity of devices within the local environments, which was enabled by the recent popularization of the microservice architecture combined with container-based virtualization. This design approach has made the components of IoT services more isolated and portable, which was the essential factor that enabled their execution across heterogeneous devices, and ultimately, the implementation of the efficient fog processing layer.

The technologies mentioned above have initiated the transition within the fog computing research from finding a way of its efficient inclusion to the cloud processing plane, towards investigating the problem of scheduling service components across a distributed execution environment. As fog computing widens the processing plane to different execution environments, the challenge now is to detect the most appropriate place where a particular service component should be deployed to deliver the best quality of service. Across the scientific literature, different algorithms that tackle this problem can be found, and the majority of them utilize Kubernetes as the operational base of their approaches. Kubernetes is an open-source container-orchestration tool intended primarily for the utilization within the cluster environment, and therefore, it shows certain limitations when applied in the volatile fog environment. Thus, the majority of existing algorithms cannot reach the full potential of fog computing since the responsiveness to fluctuating environmental conditions affecting the service delivery cannot be implemented.

This dissertation aims to overcome the stated shortcomings of existing approaches and pro-

poses an algorithm that reaches the full potential of fog computing within the IoT environment. The main contribution is the definition of a scheduling algorithm that supports a dynamic execution dependent on the fluctuating network parameters and the pre-defined application QoS requirements. However, to reach this goal, we had to conduct extensive research described in seven separate chapters of this dissertation.

The first chapter presents the theoretical background of the fog computing concept. Scientific papers that first mentioned the idea of applying principles of fog computing are described to explain the motivation for its emergence. Afterward, the OpenFog standard that defines this concept as a horizontal, system-level architecture that distributes computing, storage, control and networking functions closer to the users along a cloud-to-thing continuum, and describes its significant aspects is analyzed to introduce the idea of applying fog computing within the existing IoT architecture.

After a theoretical analysis of fog computing, the second chapter places more emphasis on its practical implementation and service scheduling. First, microservice architecture combined with container-based virtualization is described as the primary enabler for the implementation of distributed services and their scheduling across the fog computing environment. Then, the existing practical approaches described in the scientific literature are examined. The main conclusion is that service scheduling can enhance the quality of service delivery, but the scheduling procedure has to take into account all context information affecting the specific IoT service scenario to reach a better result.

Thus, the first step towards the definition of an efficient scheduling algorithm was to describe all relevant contexts affecting the feasibility and the efficiency of service scheduling across fog-to-cloud environments. These include available execution environment (device context), specific service scenario (service context), and specific service user (user context). The information about the device context is necessary to enable efficient resource management and determine the most suitable fog node to run a specific service component. Service context includes the information about the specific service scenario that defines its deployment limitations and operational goals, which have to be considered to efficiently schedule its components across the fog-to-cloud continuum. Finally, the user context describes a specific service user that enables improved location awareness and customized service delivery. The stated contexts were defined with the following properties:

- device context: CPU (frequency, number of processing units), RAM (frequency, capacity), data storage (frequency, capacity), location (network address, GPS location), power supply (available battery capacity, AC power supply), and communication (capacity of IP connection, other supported protocols);
- service context: data persistence (persistent/ non-persistent), user reach (general/ specific, user mobility), communication (IP connection capacity, other necessary communication

protocols), latency (maximum allowed latency), security (low/ high risk), statefulness (stateful/ stateless), and explicit limitations of QoS parameters;
- user context: user location (network address, GPS location), and explicit limitations of QoS parameters.

The second step was to identify relevant factors that evaluate the delivered service efficiency. Therefore, five QoS parameters of IoT services affected by the utilization of fog computing are defined in chapter four. These include:

- latency - the overall service response time calculated as the difference between the moment when the response is received on the user's side and the moment when the request is sent towards the system;
- network traffic reduction - quantified by the amount of data traffic that is not generated towards the public network or by the number of locally processed requests;
- reliability - system's ability to process the incoming requests during a specific time duration, measured as the ratio between the number of successfully received responses and the number of sent requests;
- scalability - implies using the optimal number of processing resources necessary for efficient service delivery and is measured as the system's ability to remain a constant throughput within the system;
- security - the overall service security level that is improved by prioritizing local processing within the fog-to-cloud continuum.

Afterward, the categorization of IoT services that determines the priority of identified QoS parameters and the deployment restrictions of a particular IoT scenario is specified. Four categories of IoT services are defined: data collection services, user-managed actuation services, automation services, and user-controlled automation services. Data collection and user-managed actuation services include basic scenarios, enabling simple IoT functionalities without intermediate data processing (sensing and actuation). Automation services include completely autonomous scenarios, where actuation is executed autonomously based on the sensor data collected from the targeted environment. Finally, user-controlled automation services are a similar category where the user controls the actuation to a certain extent.

In chapter five, all previously defined factors affecting the execution of service scheduling within the fog environment are summarized and formally described. Thus, the formal definition of service, user, and device context is presented along with the objective function determining the behaviour of the later-described scheduling algorithm.

The dynamic scheduling algorithm was designed based on the previously described objective function, and it is presented in chapter six. Its main goal is to determine the most efficient deployment location for each service component along the fog-to-cloud continuum based on the context information that specifies the service scenario and execution environment. Its execu-

tion is defined in five separate procedures determining its behaviour: setting up the deployment restrictions, categorizing the available fog devices, the first iteration of scheduling prioritizing the local execution, the second iteration of scheduling based on the communication latency, and finally, the procedure that enables the dynamic scheduling execution. The dynamic procedure is based on observing the values of fluctuating QoS parameters (latency and reliability) at the user's side and triggering the appropriate procedure upon their degradation. Deterioration of latency could be a consequence of two different reasons: (1) increased service demand that causes processing congestion due to the growth of incoming requests and (2) changes in network throughput and latency between distributed components. If the increased latency is a consequence of reduced throughput, the appropriate components are replicated to regain the desired system performance. Otherwise, the re-scheduling procedure is triggered again to determine the most efficient service schedule within the current state of the network environment, which is also the case if the deterioration of reliability occurs. Also, the scheduling system that utilizes the presented algorithm was implemented and described in the continuation of chapter six. It includes three separate components: the Scheduler, Service agent, and User agent. Each device that joins our system as the processing fog node has to run the Device agent component that handles all necessary interaction with the Scheduler and executes the received commands. User agent component runs on top of the user application, and it also interacts with the Scheduler. Its purpose is to send the initial request containing user context information to the Scheduler, upon which it executes the scheduling procedure and returns a response containing the address of the targeted service component. Also, if targeted user-specific limitations on QoS parameters are not satisfied, the User agent notifies the Scheduler with the request to execute a re-scheduling procedure. The Scheduler is the central component that collects the context information from User and Device agents and executes the scheduling algorithm.

Finally, in the seventh chapter, the procedure for the verification of scheduling efficiency is defined, and the verification of the presented dynamic scheduling algorithm is performed on two IoT service scenarios. Since the functionality of the presented algorithm is highly dependent on network parameters, we had to utilize a tool that enables the manipulation of network parameters to simulate diverse network conditions and verify the expected behavior of our scheduling algorithm. Thus, we executed our verification scenarios in a simulated environment running in IMUNES, an *Integrated Multiprotocol Network Emulator/Simulator* of IP-based networks that enables the simulation of different network environments and dynamic configuration of link properties, including bandwidth and latency.

The first scenario considered a streaming service designed to verify the expected behavior of our scheduling algorithm and to confirm its ability to dynamically execute re-scheduling if the user-defined limits on the values of QoS parameters are not met. To simulate this scenario, we implemented a simple end-device application that generates a data stream at a pre-defined

transmission rate towards the reception service over the opened socket link and a reception service that receives the incoming data and reports the number of received bits per second to the client. The end-device application also defines the minimum throughput required for its efficient execution, and the goal of performed simulations was to verify that our system enables the appropriate deployment of the reception service to fulfill this condition at all times. In our first test, we confirmed that the re-scheduling procedure was triggered instantly when the throughput value was under the pre-defined limit and that the service was then migrated to the more suitable fog node based on the ping value at the given moment. Our second test confirmed that the local fog node is prioritized ahead of publicly available fog nodes in our scheduling procedure.

The second scenario considered a more complex automation service category. The simulated service included three separate components: receive (reception of user requests), processing (simulated data processing), and storage (simulated database interaction). The simulated user application was implemented to constantly generate requests that were received by the receive component, then passed along to the processing component, and finally stored by the storage component, to measure the latency between each request and response. The goal of this simulation was to verify that our algorithm would successfully schedule a multi-component application and that it would adequately handle its latency deterioration or outage of a certain fog node. The simulation results confirmed the expected algorithm behaviour and showcased that it achieves better performance than the existing considered approaches (default kube-scheduler policy that considers only CPU and RAM usage for node prioritization while scheduling each pod and network-aware scheduling algorithm that considers also RTT while scheduling each pod but does not support responsive re-scheduling). The main advantage of our scheduling algorithm compared to the other two considered approaches in both simulations was its ability to dynamically re-schedule service components if the targeted QoS level was not satisfied. Thus, based on the research conducted in the scope of this dissertation, we conclude that it is important to consider as much contextual information as possible to achieve the most efficient service execution schedule in the fog computing environment. However, it is also crucial to ensure adaptive re-scheduling dependent on the volatile execution environment attributes to maintain the efficiency of service delivery at all times.

Keywords: fog computing, Internet of Things, container-based virtualization, context-aware scheduling algorithm, dynamic scheduling, QoS

Sadržaj

Uvod

Računarstvo u magli je koncept koji se pojavio kao odgovor na potrebu modernih aplikacija za efikasnijom i bržom obradom, a koji podrazumijeva proširenje resursa računalnog oblaka resursima uređaja koji se nalaze bliže krajnjim korisnicima. Tako je izvedbena okolina usluga horizontalno raspodijeljena na više slojeva, zbog čega je primjenjivost ovog koncepta posebno naglašena u scenarijima Interneta stvari (IoT - Internet of Things) koji također podrazumijevaju slojevitu i raspodijeljenu arhitekturu. Budući da danas postoji velik broj različitih područja primjene IoT-a, integracija sloja računarstva u magli se razlikuje ovisno o ciljevima koji se u konkretnom slučaju žele postići. Također, za efikasnu implementaciju ovakvih arhitektura u okruženjima Interneta stvari bilo je potrebno riješiti problem heterogenosti uređaja koji se u ovakvoj arhitekturi koriste za obradu na nižim slojevima, što je omogućeno primjenom mikroservisne arhitekture te virtualizacije zasnovane na kontejnerima. Korištenjem navedenih tehnologija u implementaciji IoT usluga, one se mogu isporučiti tako da njihove komponente podržavaju veću izoliranost i prenosivost, dok je cjelokupna usluga prilagođena raspodijeljenom izvođenju u nestalnoj i heterogenoj okolini kakvu podrazumijeva računarstvo u magli.

Mogućnost lakše implementacije raspodijeljene i fleksibilne arhitekture izvođenja IoT usluga potaknula je novi istraživački smjer u okviru koncepta računarstva u magli u kojem se proučava raspoređivanje komponenti usluga u raspodijeljenoj izvedbenoj okolini. Cilj u ovom istraživačkom području je odrediti način na koji bi se u karakterističnom raspodijeljenom okruženju odredio raspored izvođenja komponenti usluge kojim bi se isporučila veća razina kvalitete određena parametrima relevantnim za konkretan slučaj primjene. Za postizanje efikasnijeg rasporeda potrebno je pri raspoređivanju komponenti usluge uzeti u obzir različite kontekstne informacije koje određuju uslugu, korisnika, i izvedbenu okolinu. Na temelju tih informacija može se odrediti raspored izvođenja komponenti konkretnog scenarija kojim bi se, ako je to moguće, omogućila veća kvaliteta isporuke u okviru relevantnih parametara koji je određuju. Upravo je razvoj kontekstno svjesnog algoritma raspoređivanja koji bi mogao prilagođavati raspored izvođenja usluga ovisno o stanju okoline fokus ove disertacije. Izvorni znanstveni doprinos koji je ostvaren u disertaciji je:

- kategorizacija usluga Interneta stvari s obzirom na kontekste uređaja, usluga i korisnika te određivanje pripadajućih parametara kvalitete usluge,

- formalni model kategoriziranih usluga Interneta stvari i raspodijeljene okoline koja uključuje računarstvo u oblaku i računarstvo u magli,
- algoritam za dinamičko raspoređivanje kategoriziranih usluga Interneta stvari na čvorove raspodijeljene okoline s ciljem osiguravanja kvalitete usluge, i
- postupak verifikacije izvedbene učinkovitosti algoritma prema parametrima kvalitete usluge na odabranom slučaju uporabe.

Kako bi se ostvario navedeni doprinos, provedeno je istraživanje opisano u disertaciji koja je podijeljena u sedam poglavlja. U prvom poglavlju je napravljena analiza teorijskih principa koncepta računarstva u magli, dok je u drugom poglavlju naglasak na njegovoj praktičnoj izvedbi i postojećim pristupima raspoređivanju usluga. U trećem poglavlju su definirana i parametrizirana tri relevantna konteksta primjene računarstva u magli: kontekst uređaja, kontekst korisnika, i kontekst usluge na temelju kojih je u četvrtom poglavlju definirana kategorizacija IoT usluga. Dodatno su u četvrtom poglavlju specificirani i parametri kvalitete usluge za evaluaciju efikasnosti isporuke IoT usluga na koje utječe primjena računarstva u magli, te njihov prioritet u svakoj od prethodno definiranih kategorija. U petom poglavlju je opisan formalni model prethodno definiranih konteksta i ciljna funkcija koju je potrebno zadovoljiti algoritmom raspoređivanja. Razvijeni dinamički algoritam definiran je u šestom poglavlju, u kojem je opisan i sustav koji ga primjenjuje za izvođenje raspoređivanja usluga. Implementacija takvog sustava je napravljena kako bi se učinkovitost razvijenog algoritma mogla potvrditi prema postupku verifikacije definiranom u posljednjem poglavlju, u kojem su opisani i rezultati provedene verifikacije algoritma na dva odabrana slučaja primjene.

Poglavlje 1

Računarstvo u magli

Internet stvari (engl. *Internet of Things*) je jedan od vodećih pojmova suvremene evolucije Interneta, čiju inicijalnu namjenu dobro opisuje definicija Rellermeyera i ostalih koja ovaj koncept predstavlja kao ideju o stvarima koje se mogu identificirati, međusobno razmjenjivati podatke, te ih u nekoj mjeri obrađivati [1]. Nakon prve faze razvoja ovog koncepta, zbog njegovog velikog potencijala dodatno je proširen opseg rješenja koja su njime obuhvaćena, pa je danas prikladnija definicija dana u pregledu područja Interneta stvari standardizacijskog tijela ITU (*International Telecommunications Union*) gdje se definira kao globalna infrastruktura za informacijsko društvo, koja omogućuje napredne usluge međusobnim povezivanjem fizičkih i virtualnih stvari, baziranom na postojećim i budućim interoperabilnim informacijskim i komunikacijskim tehnologijama [2].

Osnovni entiteti takve globalne infrastrukture Interneta stvari su senzori i aktuatori – fizički uređaji na kojima se baziraju sve usluge ovog koncepta [3]. Takve usluge generiraju veliku količinu podataka za čiju je obradu potrebna dostatna i elastična infrastruktura kakvu nudi računalni oblak [4, 5]. Zato je većina trenutno postojećih rješenja zasnovana na prijenosu svih podataka mrežom upravo do računalnog oblaka, gdje se izvršava obrada nakon koje se rezultati isporučuju, također mrežom, do relevantnih entiteta i korisničkih uređaja [6]. Takva centralizirana arhitektura nudi dostatnu kvalitetu usluge u većini postojećih slučajeva uporabe, no nadolazeći zahtjevi novih usluga Interneta stvari zasnovani na autonomnim i reaktivnim procesima trebaju veću propusnost i manje kašnjenje kako bi se stvorili uvjeti za reakciju u stvarnome vremenu, zbog čega je važno rasteretiti mrežnu infrastrukturu i obradu približiti korisnicima kada je to moguće [7, 8]. Kao odgovor na takve zahtjeve pojavio se koncept računarstva u magli (engl. *fog computing*) iz čijeg je imena jasno da se radi o proširenju računarstva u oblaku i pozicioniranju računalnih resursa bliže korisnicima kako bi se omogućilo svojevrsno rasterećenje računalnog oblaka na račun entiteta u lokalnim okolinama [9, 10].

Zbog takve pozicije u mrežnom složaju, odnosno zbog blizine relevantnim okolinama, ovaj se koncept u literaturi često predstavlja kao tehnologija koja može omogućiti usluge Interneta

stvari koje su osjetljive na kašnjenje [11]. U literaturi također postoji i više sličnih djelomičnih definicija ovog koncepta [12, 13], a najpotpunija među njima je objavljena u standardu konzorcija OpenFog kojeg je objavila organizacija IEEE (*Institute of Electrical and Electronics Engineers*) [14]. U njoj je računarstvo u magli definirano kao horizontalna arhitektura na razini sustava, koja raspoređuje računalnu obradu, pohranu, te upravljačke i mrežne funkcije bliže korisnicima u složaju između računalnog oblaka i krajnjih uređaja. Ovim je standardom napravljen i sveobuhvatni pregled referentne arhitekture svih subjekata računarstva u magli iz različitih perspektiva, među kojima je posebno relevantna za ovu disertaciju softverska perspektiva.

U ovom poglavlju napravljen je pregled teorijske podloge koncepta računarstva u magli. Najprije je u potpoglavlju 1.1 opisana inicijalna faza razvoja ideje ovog koncepta u kojoj je naglasak bio na definiranju njegove uloge i izazova koje je potrebno riješiti kako bi on pružio ciljane funkcionalnosti te optimizirao postojeću arhitekturu za isporuku IoT usluga. Zatim su u potpoglavlju 1.2 opisani dijelovi referentne arhitekture OpenFog koji su važni u kontekstu ove disertacije.

1.1 Idejna definicija koncepta

Jedan od prvih radova u kojem se još 2012. godine spominje koncept računarstva u magli i u kojem se definira njegova namjena u okviru Interneta stvari je [15]. Autori u njemu ističu kako je računalni oblak efikasno eliminirao potrebu za posjedovanjem vlastitih poslužitelja za isporuku usluga, te omogućio dostupnost fleksibilnih računalnih resursa koji bi se koristili i plaćali ovisno o potrebama konkretne usluge. Takav centralizirani model izvođenja usluga u kojem se obrada izvodi u računalnom oblaku je ponudio veću efikasnost u većini slučajeva, ali su se vremenom pojavili scenariji koji su strogo zahtijevali nisko kašnjenje što zbog udaljenosti računalnog oblaka ovaj model samostalno nije mogao ispuniti. To je autorima bila motivacija da predstave novu "platformu" koju su nazvali računarstvo u magli (engl. *fog computing*) po uzoru na računarstvo u oblaku (engl. *cloud computing*). Takva nomenklatura je posljedica činjenice da je magla zapravo oblak bliže zemlji, a upravo je takvo spuštanje funkcionalnosti računalnog oblaka na niže slojeve ideja ovog koncepta. Autori konkretno definiraju računarstvo u magli kao virtualiziranu platformu koja pruža računalne, memorijske, i mrežne usluge između krajnjih uređaja i računalnog oblaka, a koja se tipično nalazi bliže "rubu" mreže. Kao glavne karakteristike koje definiraju takvu platformu istaknute su:

- lokacijska svijest i malo kašnjenje,
- široka geografska pokrivenost,
- podrška za mobilnost,
- veliki broj raspodijeljenih čvorova,
- bežična komunikacija,

- heterogena okolina i
- velik udio aplikacija koje podrazumijevaju strujanje podataka i obradu u stvarnom vremenu.

Na kraju su predstavljena tri slučaja primjene u kojima su autori prepoznali veliki potencijal ovog koncepta: povezana vozila, pametna energetska mreža, te velike senzorske ili aktuatorske mreže.

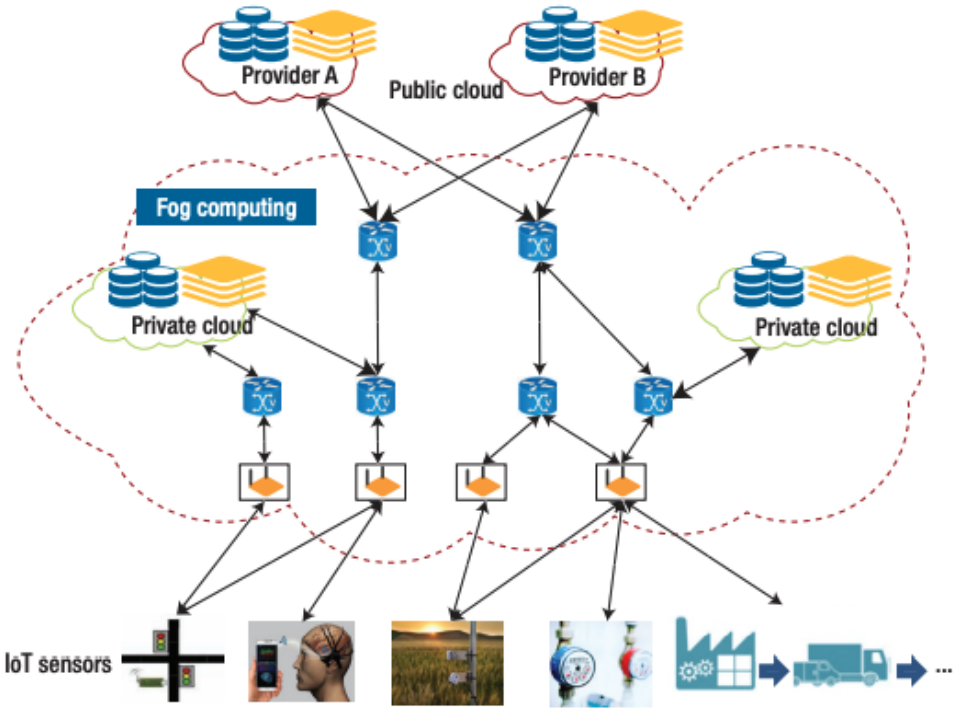
Ovaj inicijalni rad u kojem je predstavljena ideja koncepta računarstva u magli je u kratkim je crtama dobro opisao sve funkcionalnosti koje bi njegova primjena trebala omogućiti u isporuci usluga. Ipak, precizna definicija arhitekture i način implementacije računarstva u magli nisu opisani budući da je koncept još bio na razini ideje pa su i sami autori u zaključku naglasili da je u sljedećim koracima razvoja ovog koncepta potrebno definirati arhitekturu ovako robusne infrastrukture, istražiti orkestraciju i upravljanje resursima čvorova računarstva u magli, te analizirati scenarije u kojima bi primjena ovakvog arhitekturnog modela optimizirala izvedbene performance.

Nekoliko godina nakon predstavljanja ideje o konceptu računarstva u magli u prethodno opisanom radu, autori rada [10] su 2014. godine sumirali računarstvo u magli kao posljedicu nekoliko različitih trendova u primjeni tehnologije, te tehnološkog napretka rješenja koji su takve trendove omogućili. Tu kao glavne točke ističu: sveprisutnost računalnih uređaja, različite mogućnosti upravljanja uslugama i mrežom, povezivost na razini računarstva u magli i privatnost. Razvojem tehnologija i rješenja na području navedene četiri komponente zadovoljeni su preduvjeti za mogućnost implementacije i istraživanje primjene računarstva u magli, a daljnjim unapređenjem i rješavanjem izazova u ovim poljima dodatno bi se olakšala sama implementacija, te podigla efikasnost primjene ovog koncepta. U konkretnoj definiciji koncepta autori ističu da je računarstvo u magli scenarij u kojem velik broj heterogenih i sveprisutnih uređaja međusobno komunicira i surađuje kako bi se izvršili različiti zahtjevi za obradom i pohranom podataka. Takvi zahtjevi obično podrazumijevaju podršku uobičajenoj mrežnoj funkcionalnosti ili izvođenje logike modernih usluga i aplikacija, a korisnici koji nude dio resursa vlastitih uređaja za takve funkcije zauzvrat trebaju dobiti određeni poticaj. Kako bi takav scenarij zaista postao realnost i ušao u široku primjenu, autori ističu sljedeće ključne izazove koje je potrebno dodatno riješiti: mogućnost prepoznavanja ograničenja izvedbene okoline, ograničenja resursa za obradu i pohranu, upravljanje konfiguracijom čvorova, sigurnost, standardizacija, model monetizacije ovakvih rješenja, i programirljivost ovakvih rješenja.

Ovakva vizija računarstva u magli pokazuje da se ideje na kojima je utemeljen ovaj koncept i funkcionalnosti koje on treba ponuditi nisu mijenjale, ali da je za njegovu širu primjenu potrebno dodatno riješiti već poznate nedostatke sustava raspodijeljenih arhitektura. Slični se izazovi budućeg razvoja ovog koncepta navode i u radu [16], gdje se umrežavanje, kvaliteta usluge, jedinstvenost programskih sučelja, raspoređivanje radnog opterećenja, model naplate,

upravljanje dostupnim resursima, te sigurnost identificiraju kao potencijalne prepreke široj primjeni računarstva u magli. Tako se može zaključiti da je u prvoj fazi razvoja koncepta računarstva u magli istraživanje bilo većim dijelom usmjereno na teorijsku analizu načina primjene i funkcionalnosti koje bi koncept trebao ponuditi u isporuci usluga na koje se primjenjuje. Većina radova je tako bila usmjerena na iznošenje vlastitih varijanti definicije koncepta, detekciju izazova koje će biti potrebno riješiti kako bi se koncept efikasno primijenio u praksi, te identifikaciju potencijalnih područja primjene.

Ipak, već od 2015. godine mogu se pronaći radovi koji opisuju i prve konkretne eksperimente u kojima se primjenjuje računarstvo u magli i uspoređuje njegova efikasnost s arhitekturom u kojoj se koristi isključivo obrada u računalnom oblaku. Primjer takvog rada je [7] u kojem autori najprije odvajaju koncept računarstva u magli od drugih sličnih koncepata (lokalni oblak, "*cloudlets*", MCC (engl. *Mobile Cloud Computing*), te MEC (engl. *Mobile Edge Computing*)) koji podrazumijevaju slične principe zbog čega ih se često miješa u literaturi. Zatim je također napravljen pregled otvorenih izazova u razvoju računarstva u magli i ciljeva koje je potrebno ostvariti njegovom primjenom, te aplikacija u kojima bi ovakav pristup ostvario svoj potencijal. Na kraju su predstavljeni rezultati usporedbe performanci sustava u kojem je na istom scenariju primijenjena arhitektura koja uključuje računarstvo u magli i arhitektura koja je strogo usmjerena na računalni oblak. Rezultati su demonstrirali da je primjenom računarstva u magli moguće ostvariti značajne dobitke u kontekstu kašnjenja i kapaciteta brzine mrežnog prijenosa podataka.



Slika 1.1: Ideja primjene računarstva u magli u arhitekturi Interneta stvari [17].

U radu [17] se također spominju konkretna postojeća rješenja za implementaciju arhitekture računarstva u magli (Cisco IOx, Cisco Data in Motion, LocalGrid, i Cisco ParStream) te simulator IFogSim. Ovaj simulator omogućuje modeliranje i simulaciju okolina računarstva u magli pomoću kojih se može evaluirati različite načine upravljanja resursima i različite politike raspoređivanja u kontekstu njihova utjecaja na kašnjenje, potrošnju energije, zagušenost mreže, i operativnog troška. Koncept računarstva u magli autori definiraju kao raspodijeljenu paradigmu koja usluge računalnog oblaka spušta prema rubu mreže, te tako omogućuje optimizaciju rješenja zbog blizine izvora podataka i čvorova za njihovu obradu. Takva arhitektura u isporuci IoT rješenja prikazana je na slici 1.1, gdje se jasno mogu uočiti dvije važne karakteristike ovog koncepta: da funkcionalnosti računarstva u oblaku spušta u niže slojeve koji se nalaze bliže korisnicima i da sloj računarstva u magli ne isključuje računarstvo u oblaku već ga nadograđuje.

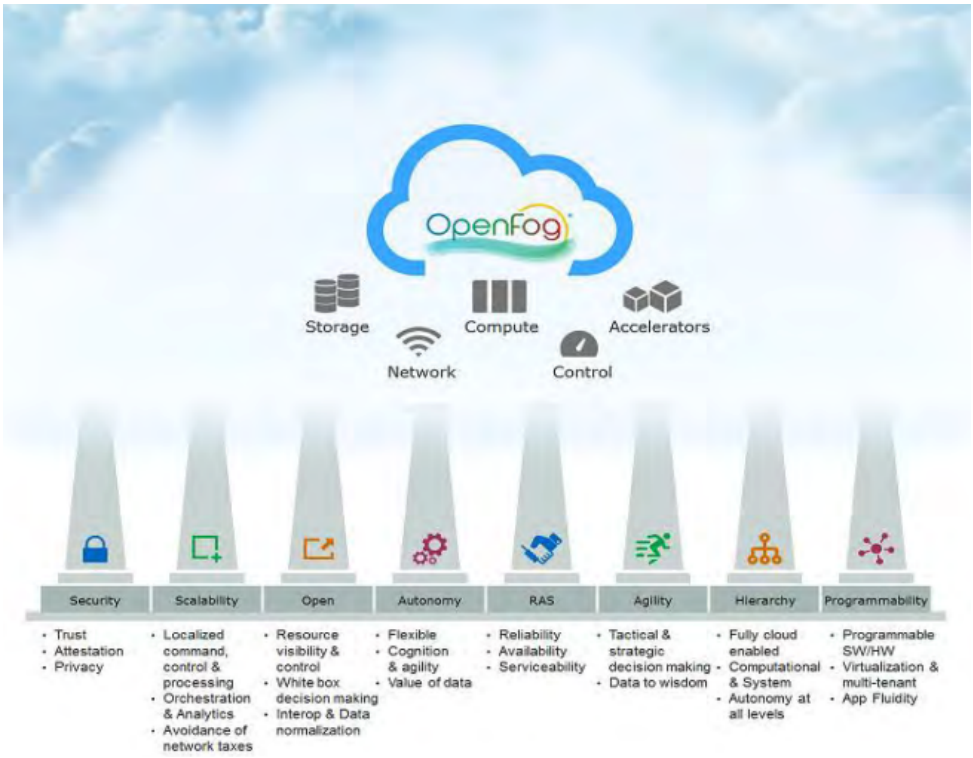
Može se tako zaključiti da je u većini radova u prvoj fazi razvoja ovog koncepta vizija uloge računarstva u magli bila slična iako nije postojao standard kojim je koncept bio specificiran. Također, vrlo brzo su se pojavili praktični primjeri iz kojih se moglo naslutiti da će primjena ovog koncepta primarno biti usmjerena na smanjivanje kašnjenja, povećanje sigurnosti, te rasterećenje mreže i računalnog oblaka. Ipak, kako bi se računarstvo u magli i njegova namjena precizirala važno je bilo specificirati standard koji bi odredio temeljne odrednice koncepta i njegove principe važne za evaluaciju rješenja koja ga primjenjuju. Zato je standard kojeg je napravila organizacija OpenFog, te kojeg je prihvatila i objavila organizacije IEEE bio važan korak za daljnji razvoj ovog koncepta.

1.2 Referentna arhitektura OpenFog

OpenFog konzorcij je standardizacijsko tijelo u čijem je fokusu koncept računarstva u magli. Osnovale su ga 2015. godine neke od vodećih svjetskih tehnoloških kompanija, kao što su ARM, Cisco, Dell i Microsoft, u suradnji sa Sveučilištem u Princetonu. Osnovan je s ciljem ubrzavanja usvajanja koncepta računarstva u magli kako bi se riješili izazovi propusnosti, kašnjenja i komunikacije koje prvenstveno predstavljaju slučajevi primjene u okviru Interneta stvari, 5G mreža, te umjetne inteligencije. Njegova primarna misija bila je definirati arhitekturni okvir za sigurnu i učinkovitu obradu informacija između oblaka, krajnjih točaka i usluga, odnosno usmjeriti industriju i akademsku zajednicu prema iskorištavanju potencijala kojeg nudi koncept računarstvo u magli. Od 2019. godine OpenFog konzorcij integriran je organizaciji Industrial Internet Consortium, koja je 2021. preimenovana u Industry IoT Consortium [18] i nastavlja djelovati s ciljem da omogući efikasnu transformaciju poslovanja optimalnom integracijom Interneta stvari.

1.2.1 Stupovi referentne arhitekture

OpenFog konzorcij je ostvario svoj inicijalni cilj objavom dokumenta *OpenFog Reference Architecture for Fog Computing* [19] u kojem je temeljito obrađen koncept računarstva u magli, te koji je kasnije organizacija IEEE prihvatila kao standard [14]. Njihova referentna arhitektura se zasniva na osam "stupova" (engl. *pillars*), odnosno ključnih svojstava koje sustav treba imati kako bi implementirao OpenFog arhitekturu računarstva u magli (slika 1.2). U nastavku ovog potpoglavlja napravljen je kratki pregled definiranih stupova.



Slika 1.2: Stupovi OpenFogove referentne arhitekture [19].

Sigurnost

Aplikacije Interneta stvari su naglašeno osjetljive na aspekt privatnosti budući da sigurnosni propusti u njima mogu imati posljedice u virtualnom, ali i u fizičkom svijetu. Stoga sigurnost predstavlja vrlo bitan pojam u okviru računarstva u magli, zbog čega OpenFogova referentna arhitektura opisuje mehanizme koje je potrebno primijeniti kako bi se osigurala cjelokupna vertikalna sigurnost čvora računarstva u magli.

Implementacija sigurnosti podrazumijeva osiguravanje privatnosti, anonimnosti, cjelovitosti, povjerenja, verifikacije i slično. Za postizanje navedenih svojstava u okolini računarstva u magli potrebno je izvršiti sigurnosnu verifikaciju čvora prije uspostavljanja povjerenja i njegova konačnog povezivanja raspodijeljenu okolinu. Također, uz sigurnost čvora potrebno je u obzir

uzeti i mrežnu sigurnost, te sigurnost upravljanja i orkestracije u sustavu kako bi se ostvarila sigurnost cjelokupne arhitekture računarstva u magli.

Stup sigurnosti referentne arhitekture OpenFog započinje jasnom definicijom osnovnih građevnih blokova. Svi čvorovi računarstva u magli moraju koristiti nepromjenjivi hardverski korijen povjerenja (engl. *hardware root of trust*). Hardverski korijen povjerenja je pouzdana hardverska komponenta koja prima kontrolu pri uključivanju te proširuje lanac povjerenja na druge hardverske i softverske komponente, te na komponente s ugrađenim softverom (engl. *firmware*). Zbog blizine ruba, čvorovi računarstva u magli često djeluju kao prvi čvor kontrole pristupa i šifriranja zbog čega moraju osigurati cjelovitost i izolaciju, te upravljati prikupljanjem osjetljivih podataka prije nego što napuste rub mreže.

Skalabilnost

Računarstvo u magli treba osigurati dinamičku skalabilnost za tehničke i poslovne zahtjeve aplikacija kojima je namijenjen ovaj koncept, a koji su također dinamički promjenjivi. Hijerarhijska svojstva računarstva u magli i njegov položaj na logičkim rubovima mreže osigurava ciljanu elastičnu skalabilnost, i to dodavanjem novih načina skaliranja:

- svaki čvor računarstva u magli može biti interno skalabilan pomoću hardverskih i softverskih proširenja;
- mreže računarstva u magli mogu se povećavati ili smanjivati dodavanjem novih čvorova kako bi se rasteretili oni koji su preopterećeni;
- mreža čvorova računarstva u magli može se skalirati u okruženju ovisno o potražnji;
- pohrana, mrežna povezanost i analitičke usluge mogu se skalirati u infrastrukturi računarstva u magli.

Takva elastična skalabilnost u implementaciji računarstva u magli potrebna je kako bi se sustav mogao prilagoditi radnom opterećenju, troškovima sustava, performancama i drugim promjenjivim poslovnim potrebama. Također, navedene su i različite dimenzije skalabilnosti koje je u okviru računarstva u magli potrebno osigurati u konkretnom slučaju primjene:

- skalabilne performance – rast kapaciteta računarstva u magli kao odgovor na zahtjeve performanci aplikacija;
- skalabilni kapacitet – promjena opsega mreža računarstva u magli ovisno o količini aplikacija, čvorova, te krajnjih uređaja ili korisnika koji se priključuju ili isključuju iz mreže;
- skalabilna pouzdanost – uvođenje redundantnih čvorova kako bi se povećala otpornost na ispade ili preopterećenja;
- skalabilna sigurnost – uvođenje dodatnih hardverskih i softverskih modula na čvor računarstva u magli ovisno o strožim sigurnosnim zahtjevima sustava;
- skalabilni hardver – izmjene konfiguracije internih elemenata čvorova računarstva u magli skaliranjem resursa procesora, prostora za pohranu podataka ili mrežnih kapaciteta;

- skalabilni softver – modularna arhitektura implementacije sustava i orkestracija njegova izvođenja u raspodijeljenoj okolini.

Otvorenost

Otvorenost je ključno svojstvo koje je potrebno osigurati kako bi sustavi računarstva u magli bili prilagođeni za povezivanje različitih platformi i aplikacija Interneta stvari. Rješenja ograničena na zatvoreni skup tehnologija mogu imati negativan utjecaj na trošak sustava, kvalitetu, i inovacije, pa je cilj ovog stupa osigurati interoperabilnost cjelokupne izvedbene okoline. Tako bi se jednaki softverski definirani čvorovi računarstva u magli mogli pokretati dinamički ovisno o potrebama sustava, neovisno o platformi na kojima se čvorovi nalaze. Neke od glavnih karakteristika otvorenosti sustava su:

- prilagođenost za međusobno povezivanje,
- interoperabilnost,
- otvorenost komunikacije i
- lokacijska transparentnost.

Autonomnost

Stup autonomnosti odnosi se na osiguravanje isporuke funkcionalnosti čvorova unatoč kvarovima na ostalim komponentama usluge. U ovakvoj arhitekturi podržana je autonomnost u svim slojevima hijerarhije, uključujući i uređaje na rubovima mreže, čime se arhitektura odmiče od centraliziranog donošenja odluka u računalnom oblaku. Referentna arhitektura OpenFog podrazumijeva autonomiju za širok raspon funkcija, a tipični primjeri koji se navode odnose se na autonomnost:

- otkrivanja čvorova,
- orkestracije i upravljanja uslugama,
- sigurnosti sustava i
- izvršavanja operacija.

Programirljivost

Stup programirljivosti odnosi se na stvaranje adaptivnih okruženja koja uključuju podršku za programiranje na softverskom i hardverskom sloju. Tako se omogućuje lakše izvršavanje automatske preraspodjelu poslova između čvorova računarstva u magli prema ciljanoj operativnoj dinamici rada sustava. Programirljivost čvora računarstva u magli uključuje sljedeće prednosti:

- adaptivna infrastruktura – osiguravanje podrške promjenama u poslovnim zahtjevima;
- učinkovita iskoristivost resursa izvedbene okoline – maksimiziranje iskoristivosti resursa korištenjem tehnologija koje to omogućuju, poput kontejnerizacije;

- paralelizam - mogućnost paralelnog izvođenja u izoliranim okolinama za izvođenje;
- ekonomične operacije – osiguravanje infrastrukture adaptivne prema promjenjivim zahtjeva;
- poboljšana sigurnost – automatska instalacija zakrpi (engl. *patch*) na brzo-razvijajuće sigurnosne prijetnje.

Pouzdanost, dostupnost i upotrebljivost

Pouzdanost arhitekture sustava odnosi se na njegovu mogućnost da isporuči očekivanu funkcionalnost pod normalnim i nepovoljnim radnim uvjetima. Pouzdanost u referentnoj arhitekturi OpenFog uključuje sljedeća svojstva:

- osiguravanje rada hardvera na kojem je pokrenut softver, te elastične i pouzdane mreže čvorova računarstva u magli;
- osiguravanje raspoloživosti i cjelovitosti podataka;
- autonomno izvođenje prediktivnih i prilagodljivih rutina za samostalni oporavak sustava, instalaciju nadogradnji, te općenito održavanje stabilnosti hardverske i softverske podloge sustava;
- validacija platforme sustava i arhitekture kroz testove za provjeru interoperabilnosti.

Raspoloživost osigurava kontinuirano upravljanje i orkestraciju, a mjeri se ukupnim vremenom tijekom kojeg sustav ispravno radi. Raspoloživost arhitekture podrazumijeva sljedeća svojstva:

- osiguran pristup za orkestraciju i upravljanje na svim razinama hijerarhije računarstva u magli;
- bolja izolacija i detekcija uzorka kvara, te primjena strojnog učenja kako bi se smanjilo srednje vrijeme oporavka (MTTR, engl. *Mean Time To Repair*);
- primjena funkcionalne podrške pomoću računalnog oblaka s izloženim sučeljima kroz cjelokupni sustav.

Upotrebljivost računarstva u magli osigurava ispravnu funkcionalnost u isporuci usluge. Upotrebljivost arhitekture sustava obuhvaća sljedeća svojstva:

- omogućena automatizirana instalacija, nadogradnja i popravak sloja računarstva u magli;
- omogućen samostalni oporavak hardvera i softvera ili servis od strane raznih proizvođača;
- jednostavnost upotrebe kako bi se olakšalo održavanje;
- redundantnost konfiguracije koja omogućuje trajnu produktivnost.

Agilnost

Fokus stupa agilnosti je transformacija velikih količina podataka kakve sam čovjek ne može analizirati u korisne informacije za donošenje operativnih odluka. Kako bi se omogućila takva funkcionalnost, agilan sustav treba omogućiti efikasan rad u dinamičnoj okolini računarstva u magli tako da odgovara brzo na promjene u okolini.

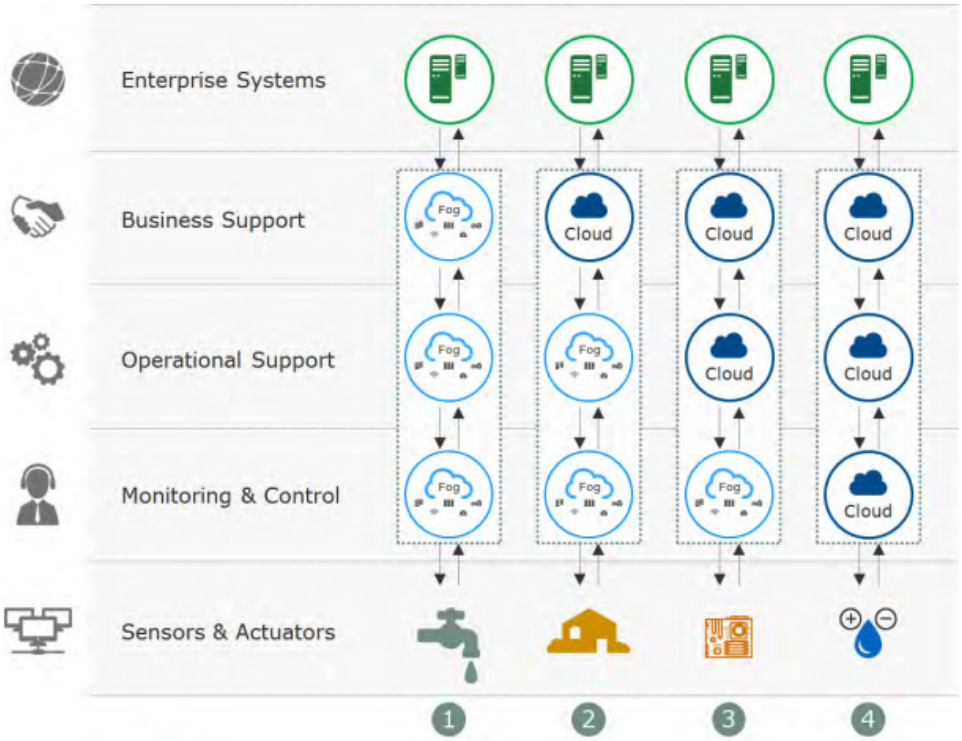
Razni senzori i sustavi obično generiraju velike podatkovne skupove koji u sustav mogu pristizati u različitim formatima i iz različitih kontekstnih okolina. Zato se njihov kontekst ponekad stvara tek kada se podaci usporede, agregiraju i analiziraju. Takva analiza podataka, a zatim i donošenje operativnih odluka, može se izvršavati u računalnom oblaku, ali to podrazumijeva porast kašnjenja zbog prijenosa svih podataka. Zato se preporučuje arhitekturni pristup u kojem se određuje kontekst podataka i donose operativne odluke na prvom sloju u kojem je to moguće, što je često sloj računarstva u magli.

Hijerarhijska struktura

Hijerarhijska organizacija arhitekture računarstva u magli ovisi o primarnoj namjeni i funkcionalnim zahtjevima sustava na koji se ona primjenjuje. Računarstvo u magli tako može ovisno o specifičnom scenariju primjene podrazumijevati mrežu međusobno povezanih sustava podijeljenih u različite fizičke ili logičke slojeve, ili jedinstveni sustav u kojem su čvorovi smješteni u zajedničkoj okolini. U nastavku su opisani hijerarhijski slojevi IoT sustava koji uključuje sloj računarstva u magli, namijenjeni različitim funkcionalnostima koje takvi sustavi trebaju ponuditi:

- sloj uređaja: senzorski i aktuatorski uređaji koji su ulaz ili izlaz djelovanja IoT sustava. Tako se u ovom sloju senzorskim uređajima generiraju telemetrijski podaci koji predstavljaju stanje ciljane okoline, dok se aktuatorskim uređajima može djelovati na stanje ako je to potrebno;
- sloj kontrolnog nadzora: sloj u kojem se izvodi kontrola stanja i generiranje naredbi kojima se djeluje na stanje ciljane okoline. Stanje okoline predstavljeno je parametrima koji se mogu opisati senzorskim podacima, a aktuatorima se može utjecati na vrijednosti tih istih parametara. Tako je glavni cilj u ovom sloju izvršavanje kontrolne logike nad stanjem okoline na temelju senzorskih podataka, te izvršavanje specifičnih aktivnosti kao rezultat toga;
- sloj operativne podrške: Sloj operativne podrške odgovoran je za analizu ulaznog telemetrijskog toka i pohranu analitičkih podataka. Cilj takvih analiza obuhvaća operativne aspekte ciljane fizičke okoline u specifičnom slučaju koji daju preglednu sliku njenog stanja. Izlazno stanje se obično predstavlja kroz grafička sučelja poput kontrolne ploče ili mobilne aplikacije.
- sloj surogatskih uređaja: izvođenje kompleksnih operacija može se delegirati između čvorova računarstva u magli kako bi se izjednačilo opterećenje te kako bi se omogućila obrada u slučajevima kada je jedan čvor ne može izvršiti samostalno. Ovaj sloj podrazumijeva implementaciju takve mogućnosti u sustavu pomoću dodatnih uređaja koji mogu izvršavati djelomičnu ili cjelokupnu obradu kada je to potrebno;
- sloj poslovne podrške: osnovna namjena ovog sloja je pohrana i analiza cjelokupne povi-

jesti IoT operacija u specifičnom scenariju. Na temelju rezultata tako provedenih analiza može se izvršavati poslovno planiranje, evaluacija operacijske učinkovitosti procesa, treniranju modela strojnog učenja, i slične aktivnosti koje mogu optimizirati sustav u poslovnom aspektu.



Slika 1.3: Hijerarhijski modeli primjene računarstva u magli [19].

Na slici 1.3 prikazani su različiti hijerarhijski modeli primjene računarstva u magli i računarstva u oblaku u vertikalnim IoT sustavima, odnosno različit opseg opisanih hijerarhijskih slojeva koji može obuhvaćati primjena računarstva u magli.

Prvi slučaj prikazuje hijerarhiju implementacije računarstva u magli koja je neovisna o računalnom oblaku. Ovakav model primjenjiv je u slučajevima kada obrada u računalnom oblaku nije efikasna zbog strogih zahtjeva na nisko kašnjenje, zahtjeva za visokom sigurnosti i privatnosti, ili nedostupnosti centralnog oblaka na određenim mjestima. Primjeri ovakvih scenarija osjetljivije primjene Interneta stvari kao npr. primjene u ratnim sustavima, upravljanje dronovima, primjene u određenim zdravstvenim sustavima, te primjene u bankovnim sustavima.

Drugi slučaj prikazuje hijerarhiju u kojoj se računalni oblak koristi za obradu informacija potrebnih za donošenje odluka kojima je vremenski okvir za dobivanje rezultata duži vremenski period iskazan u satima ili danima. Primjeri takvih sustava uključuju sustave za upravljanje pametnim zgradama, nadzor solarnih panela, optimizacije logističkih ruta i slično.

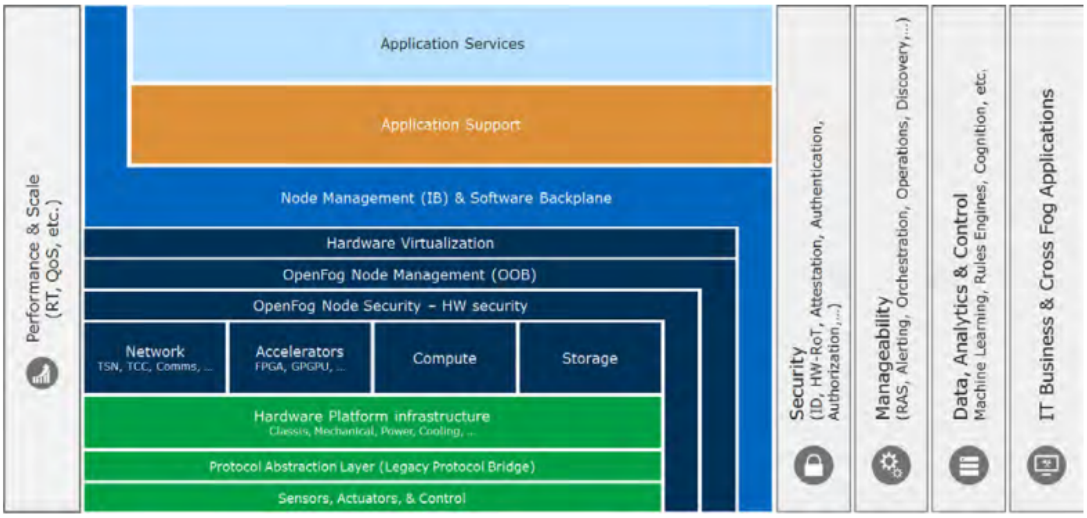
Treći slučaj prikazuje hijerarhijsku organizaciju gdje se računarstvo u magli koristi za vremenski osjetljive izračune, dok se računalni oblak koristi za obradu operativnih i poslovnih informacija. Ovakva arhitektura primjenjuje se najčešće u implementaciji sustava za nadzor

uređaja s neprekidnim izvorom napajanja (UPS, engl. *Uninterruptible Power Supply*), mreža za isporuku sadržaja (CDN, engl. *Content Delivery Networks*), te sustava za ubrzanje mobilnih mreža.

Četvrti slučaj prikazuje arhitekturu u kojoj se računalni oblak koristi u svim navedenim slojevima hijerarhije zbog ograničenih okruženja u kojima implementacija računarstva u magli ne bi bila izvediva ili ekonomična. Primjeri takvih sustava uključuju primjene u poljoprivredi, pametnu mobilnost (povezani automobili), te udaljene stanice za vremenske prognoze.

1.2.2 Strukturni opis referentne arhitekture

Osim prethodno opisanih stupova računarstva u magli, važan dio referentne arhitekture OpenFog je i opis funkcionalnosti svih sudionika u cjelokupnoj arhitekturi karakterističnoj za ovaj koncept. To uključuje proizvođače krajnjih uređaja, proizvođače sustava, integratore sustava, razvojne inženjere, te vlasnike aplikacijskih rješenja. U ovom potpoglavlju je zato napravljen kratki pregled perspektiva i važnih aspekata opisanih u referentnoj arhitekturi OpenFog koji se primjenjuju kako bi se u specifičnom scenariju ostvarila ciljana funkcionalnost računarstva u magli.



Slika 1.4: Strukturni pregled OpenFog referentne arhitekture [19].

Na slici 1.4 prikazan je složeni opis referentne arhitekture koji uključuje perspektive, prikazane sivim okomitim okvirima s tekstom na bočnim stranama opisa arhitekture. Perspektive u referentnoj arhitekturi OpenFog uključuju:

- performance: smanjivanje kašnjenja je jedan od glavnih pokretača razvoja računarstva u magli, pa se često primjenjuje za evaluaciju efikasnosti primjene ovo koncepta. Primjenom računarstva u magli može se djelovati na kašnjenje u obradi ili kašnjenje u komunikaciji pa je ova perspektiva važna za sam sustav i za raspored izvođenja njegovih komponenti;

- sigurnost: sigurnost cjelokupne arhitekture je jedan od kritičnih zahtjeva efikasne implementacije računarstva u magli. Za ostvarivanje potpune sigurnosti sustava potrebno je razmatrati sve komponente sigurnosti od krajnjih uređaja do viših slojeva kako bi se postigla kompletno sigurna i otporna arhitektura rješenja kojom se također osigurava i integritet podataka;
- upravljivost: mogućnost upravljanja svim aspektima implementacije računarstva u magli kroz cjelokupnu hijerarhiju slojeva je također jedan od kritičnih zahtjeva za postizanje ciljane efikasnosti primjenom ovog koncepta;
- analiza i kontrola podataka: za ostvarivanje autonomne obrade podataka na čvorovima računarstva u magli, potrebno je na njima ostvariti veći stupanj samostalne kontrole i upravljanja u obradi podataka. Na taj se način omogućuje izvođenje zahtijevane obrade na najprikladnijem mjestu u cjelokupnoj arhitekturi zbog čega se podiže i ukupna efikasnost sustava u specifičnom scenariju;
- IT aplikacije u okolinama računarstva u magli: heterogena izvedbena okolina zahtjeva da se aplikacijama omogući migracija i funkcionalnost neovisno o konkretnim platformama na kojima će se izvršavati. Također, kako bi se maksimalno iskoristile mogućnosti koje nudi koncept računarstva u magli potrebno je osigurati mogućnost njihova izvođenja kroz cjelokupnu arhitekturnu hijerarhiju, što se postiže njihovim modularnim razvojem.

Uz navedene perspektive posebno su naglašena tri važna aspekta referentne arhitekture OpenFog opisana u nastavku ovog poglavlja.

Aspekt čvora

Aspekt čvora uključuje entitete na najnižim razinama referentne arhitekture u kojima se nalaze senzori, aktuatori, kontrolni čvorovi, i čvorovi za apstrakciju protokola, pa je ovaj aspekt namijenjen prvenstveno proizvođačima uređaja, arhitektima *firmwarea*, i arhitektima sustava. Izdvojeni parametri koje je potrebno uzeti u obzir prije nego što se određenom čvoru dozvoli ulazak u mrežu računarstva u magli su:

- sigurnost čvora: kao što je već navedeno, sigurnost čvora računarstva u magli je jedan od ključnih elemenata za osiguravanja sigurnosti cjelokupnog sustava. U mnogim slučajevima čvorovi računarstva u magli djeluju kao mrežni prilaz (engl. *gateway*) kojeg senzori i aktuatori koriste za vezu s čvorovima viših slojeva, pa su zato upravo oni pogodni za implementaciju sigurnosnog prilaza krajnjim uređajima.
- upravljanje čvorom: čvor treba imati sučelja kojima omogućuje čvorovima na višim slojevima upravljanje i kontrolu njihova stanja. Takva interoperabilna sučelja omogućuju pristup čvorovima što je važan element za izvođenje orkestracije sustava u heterogenoj izvedbenoj okolini;
- mrežna komunikacija: svaki čvor računarstva u magli mora imati mogućnost mrežne

komunikacije. Također, zbog osjetljivosti na kašnjenje većine aplikacija računarstva u magli, u nekim je scenarijima potrebno omogućiti i vremenski osjetljivo umrežavanje (TSN, engl. *Time Sensitive Networking*).

- akceleratori: mnoge aplikacije računarstva u magli koriste akceleratore kako bi se zadovoljila ograničenja u kašnjenju i potrošnji energije. Akceleratori su dodatni moduli koji se integriraju s procesorom kako bi se osigurala veća računalna moć (npr. kriptografski čipovi i grafičke kartice);
- računalni resursi: čvor računarstva u magli treba imati dostatne resurse za izvođenje većine standardnih aplikacija. To je svojstvo važno kako bi se omogućila interoperabilnost, budući da se tako osigurava okolina u kojoj svi čvorovi mogu izvršavati ciljane aplikacije;
- resursi za pohranu: čvor treba imati sposobnost učenja, a preduvjet za takvu funkcionalnost je mogućnost pohrane podataka. Uređaji za pohranu podataka (priključeni ili ugrađeni u čvor računarstva u magli) moraju zadovoljavati zahtjeve za performance, pouzdanost, i cjelovitosti podataka u ciljanom scenariju;
- senzori i aktuatori: čvor mora imati podršku za povezivanje s raznim senzorskim ili aktuatorskim uređajima koji se smatraju elementima najnižeg sloja IoT sustava. Heterogenost krajnjih uređaja i različiti komunikacijski protokoli koje oni mogu koristiti zahtijevaju da čvor računarstva u magli na koji se oni povezuju podržava rad s različitim tipovima krajnjih IoT uređaja, a posebno onim tipom koji se očekuje u određenom scenariju;
- sloj apstrakcije protokola: zbog prethodno spomenute različitosti komunikacijskih protokola koje krajnji uređaji mogu koristiti, ponekad je njihova direktna povezivost s čvorom računarstva u magli nemoguća bez sloja apstrakcije protokola. Tim se slojem omogućuje logička poveznica između čvora računarstva u magli i krajnjih uređaja te se posredno ostvaruje njihova međusobna komunikacija.

Aspekt arhitekture sustava

Aspekt sustava u referentnoj arhitekturi OpenFog podrazumijeva jedan ili više čvorova računarstva u magli koji zajedno s drugim komponentama čine jedinstvenu platformu. Zato je ovaj aspekt računarstva u magli namijenjen u najvećoj mjeri arhitektima sustava, proizvođačima hardvera, i proizvođačima platformi. Ovaj aspekt je komplementaran na aspekt čvora, pa je potrebno uzeti u obzir oba aspekta za implementaciju optimalnog rješenja koje uključuje računarstvo u magli u specifičnom scenariju. Kao glavne stavke ovog aspekta izdvojene su:

- hardverska infrastruktura platforme - platforme računarstva u magli trebaju omogućiti mehaničku podršku i zaštitu svojim internim komponentama. U mnogim implementacijama platforme čvorova računarstva u magli trebaju biti otporne na različite vanjske uvjete okruženja u kojim se nalaze pa su zato neki od istaknutih zahtjeva: zaštita od vanjskog okoliša, otpornost na fizičke napade ili krađu, te prihvatljiva veličina, potrošnja energije

i težina. Također, platforma čvora računarstva u magli treba imati hardverske resurse prilagođene implementaciji modularne arhitekture kakvu podrazumijeva ovaj koncept;
- hardverska virtualizacija i kontejneri - mehanizmi hardverske virtualizacije dostupni su u gotovo svim platformama koje se koriste kao podloge čvorovima računarstva u magli. Ovakav način virtualizacije hardvera omogućuje da više entiteta koristi isti fizički sustav što omogućuje veću iskoristivost za implementaciju sustava računarstva u magli. Kontejnerska virtualizacija je noviji oblik virtualizacije koji u manjoj mjeri postiže izolaciju čvorova računarstva u magli, budući da se ona u ovom slučaju temelju na operacijskom sustavu, a ne na hardveru. Odabir oblika virtualizacije najčešće ovisi o razini sigurnosti koja se zahtjeva u specifičnom scenariju primjene.

Aspekt softverske arhitekture

Softverski aspekt referentne arhitekture OpenFog podrazumijeva softver koji je pokrenut na platformi sastavljenoj od jednog ili više čvorova računarstva u magli i drugih komponenti potrebnih za implementaciju sustava. Tako je ovaj aspekt primarno namijenjen integratorima sustava, softverskim arhitektima, dizajnerima rješenja, i razvojni inženjerima aplikacija za izvođenje u okolini računarstva u magli. Ovaj aspekt obuhvaća tri različita sloja:
- aplikacijske usluge - aplikacije koje su ovisne o infrastrukturi druga dva sloja, a izvode logiku rješenja za specifičan slučaj primjene;
- aplikacijska podrška - infrastrukturni softver koji samostalno nema svrhu, ali omogućuje podršku za izvođenje drugih aplikacija;
- upravljanje čvorovima i softverska podrška - softver za upravljanje čvorovima računarstva u magli i njihovom međusobnom komunikacijom u sustavu.

Sloj upravljanja čvorovima i softverske podrške treba omogućiti izvođenje različitih softverskih aplikacija i međusobnu komunikaciju čvorova. Kontejneri su jedan od glavnih mehanizama softverske podrške koji omogućuju izolaciju aplikacija, te pružaju podršku za izvođenje različitih softverskih rješenja na istom operacijskom sustavu. Kontejneri koriste predefiniranu količinu dostupnih računalnih resursa uređaja, ali se izvode nad istom jezgrom operacijskog sustava, pa se izoliranost temelji na razdvojenim adresnim prostorom koji su im dodijeljeni. Ovaj oblik virtualizacije tako omogućava lakši razvoj raspodijeljenih sustava, a kako bi se u takvoj okolini postigla i ciljana razina sigurnosti, ovaj sloj treba omogućiti i sigurno povezivanje komponenti sustava, odnosno kontrolirati pravilno proširivanje korijena povjerenja. Tako je sva kontrolna komunikacija između svih entiteta u ovakvoj arhitekturnoj hijerarhiji zapravo dio ovog sloja, pa se njime omogućuju i druge važne kontrolne funkcionalnosti (npr. otkrivanje usluga, otkrivanje čvorova, upravljanje stanjem, te upravljanje objavama i pretplatama). Upravljanje čvorovima je druga zadaća ovog sloja što podrazumijeva održavanje zadanog stanja hardvera i softvera na čvorovima računarstva u magli. Tako je svim uključenim čvorovima računarstva u magli

potrebno omogućiti upravljanje:

- konfiguracijom čvorova,
- izvođenjem operacija,
- sigurnošću,
- kapacitetom i
- dostupnošću.

Sloj aplikacijske podrške podrazumijeva dostupnost različitih alata i platformi koje su potrebne za izvođenje konkretnih aplikacija na višem sloju. Sustavi koje aplikacije trebaju za ostvarivanje vlastite funkcionalnosti (npr. baze podataka, sustavi za razmjenu poruka i sl.) mogu biti unaprijed instalirani na takvim čvorovima ili dostupni u obliku kontejnera kako bi ih aplikacije mogle brzo pokrenuti ako je to potrebno. Dostupnost takvih sustava u obliku kontejnera osigurava bolju izoliranost, dodatnu sigurnost i lakše skaliranje pa se preporučuje ovaj oblik isporuke aplikacijske podrške. Glavne funkcionalnosti koje ovaj sloj treba isporučiti aplikacijama na višem sloju su:

- upravljanje aplikacijama – verzioniranje, verifikacija, nadogradnja i generalno upravljanje aplikacijama i mikrouslugama;
- izvedbenu okolinu – virtualni strojevi, kontejneri, biblioteke programskih jezika i izvršne datoteke koje omogućuju okolinu za izvršavanje aplikacija i mikrousluga (npr. JVM, Node.js, NET framework, i sl.);
- aplikacijske poslužitelje – web-poslužitelji ili aplikacije koje služe za pokretanje mikrousluga (npr. JBoss, Tomcat, i sl.);
- sustave za razmjenu poruka i događaja – podrška za aplikacije koje primjenjuju komunikaciju porukama između mikrousluga ili zasnivaju funkcionalnost na događajima koji se najčešće ostvaruju sustavima objavi-pretplati (npr. ActiveMQ, ZeroMQ, RabbitMQ, i sl.);
- sigurnosne usluge – podrška za sigurnost aplikacija što podrazumijeva enkripcijske usluge, identifikacijske brokere, i sl. Također, za neke je aplikacije potrebno omogućiti i usluge dubinske provjere paketa, otkrivanje i sprječavanje neovlaštenog upada u sustav, nadzor sustava i mreže, filtriranje sadržaja, te roditeljsku kontrolu;
- upravljanje, pohranom i konzistentnosti podataka – podrška za transformaciju, pohranu, i skladištenje podataka. Potrebno je omogućiti sustave za konzistentnu pohranu podataka što uključuje SQL i NoSQL baze podataka (npr. SQLite, Cassandra, Mongo, i sl.), ali i druge načine pohrane kao što su *in-memory* baze podataka i priručne memorije (npr. Redis, Gemfire, i sl.);
- alate za analitičku obradu podataka – podrška za obradu velikih količina podataka (npr. Apache Spark, Apache Hadoop, i sl.).

Najviši sloj u aspektu softverske arhitekture obuhvaća konkretne aplikacije čije izvođenje

ovisi o specifičnom scenariju primjene i dostupnim resursima na kojim se mogu pokrenuti njihove komponente. One mogu biti isporučene u izvornom obliku ili u obliku kontejnera, a pravilnu funkcionalnost im omogućuje prethodno opisani sloj aplikacijske podrške. Postoje različiti tipovi komponenti koje zajedno čine kompletnu aplikacijsku uslugu, među kojima se najčešće izdvajaju:

- ulazne usluge računarstva u magli - služe za povezivanje s krajnjim IoT uređajima. Njihova osnovna namjena je pružiti apstrakciju komunikacijskih protokola kojim ostvaruju komunikaciju s različitim senzorskim i aktuatorskim uređajima, te uslugama na višim slojevima arhitekture računarstva u magli ponuditi sučelje za komunikaciju s krajnjim uređajima;
- temeljne usluge - usluge koje prikupljaju podatke s ruba mreže i omogućuju drugim sustavima i uslugama pristup takvim podacima. Njihova osnovna namjena je tako povezivanje viših slojeva s ciljanim krajnjim uređajima;
- usluge izvedbene podrške - uključuju mikrousluge koje izvode osnovne zadaće svakog softverskog sustava kao npr. vođenje zapisnika događaja (engl. *logging*), registracija usluga, raspoređivanje usluga, i brisanje podataka;
- analitičke usluge - uključuju usluge za izvođenje reaktivne i prediktivne analitike. Reaktivna analitika se obično odnosi na usluge u nižim slojevima računarstva u magli koje prate promjene direktno u podacima primljenim iz okoline. Prediktivna analitika se izvodi na čvorovima s jačim računalnim resursima budući da obično uključuje različite oblike strojnog učenja i izvođenje algoritama umjetne inteligencije;
- integracijske usluge - usluge koje omogućuju međusobno umrežavanje čvorova računarstva u magli i njihovu razmjenu podataka. Tako je ovim uslugama potrebno implementirati da se drugim čvorovima omogući pristup podacima čvora na kojem su pokrenute;
- usluge korisničkog sučelja - omogućuju korisnicima pristup podacima čvora računarstva u magli, status pokrenutih mikrousluga na njemu, rezultate izvedene analitike podataka, i upravljanje sustavom i operacijama konkretnog čvora.

Poglavlje 2

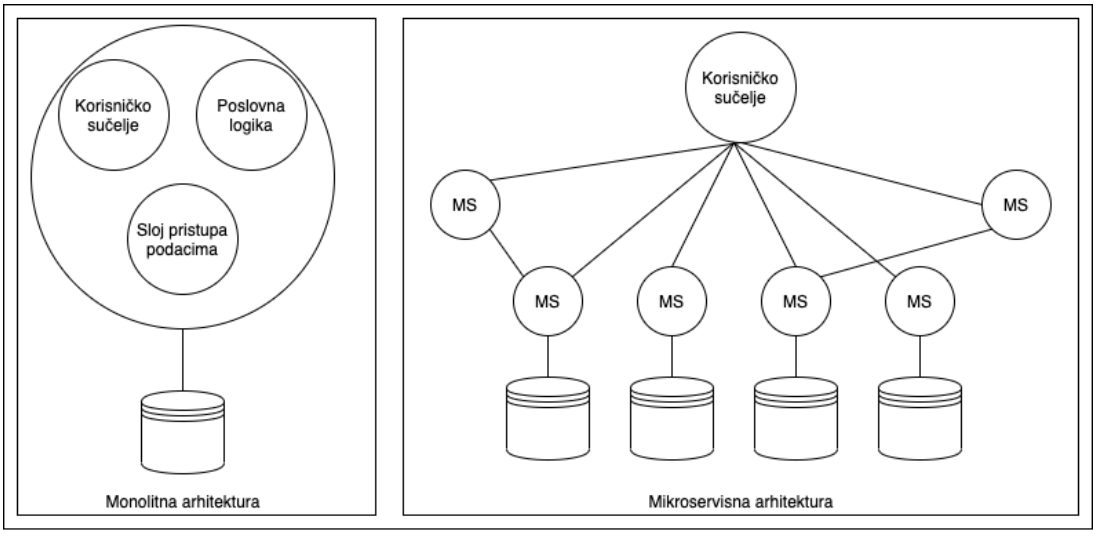
Raspoređivanje usluga u okolini računarstva u magli

Prethodno opisana teorijska podloga računarstva u magli definira ovaj koncept i daje pregled funkcionalnosti koje je potrebno implementacijski ostvariti kako bi se postigle dobiti njegove primjene. U ovom poglavlju naglasak je na praktičnoj izvedbi predstavljenog koncepta. Najprije je u potpoglavlju 2.1 opisana raspodijeljena arhitektura sustava zasnovana na mirkouslugama te druge konkretne tehnologije koje su omogućile praktičnu izvedbu ovog koncepta, odnosno koje su osigurale mogućnost efikasne implementacije sustava čiju infrastrukturu čine međusobno odvojeni uređaji u različitim okolinama. Zatim je u potpoglavlju 2.2 napravljen pregled praktičnih istraživačkih problema vezanih uz ovaj koncept te postojećih znanstvenih radova koji su ponudili vlastita rješenja i konkretne primjere sustava koji uključuju sloj računarstva u magli.

2.1 Implementacijske tehnologije

Obrada podataka se u većini postojećih implementacija usluga Interneta stvari izvršavala isključivo u računalnom oblaku, pa je tako u njenoj izvedbi zapravo korišten centralizirani arhitekturni model (slika 2.1, lijevo) koji je zasnovan na elastičnim resursima koje oblak po niskoj cijeni pruža različitim aplikacijama [7]. Upravo je neograničena količina dostupnih resursa u oblaku i glavna prednost ovakvog pristupa, ali takav centralizirani model ima i određene nedostatke među kojima je u slučaju usluga Interneta stvari najizraženije kašnjenje odgovora na poslani zahtjev zbog njegove fizičke, odnosno mrežne, udaljenosti. Dodatno, ovakav arhitekturni model podrazumijeva konstantnu dostupnost internetske mreže, slanje osjetljivih podataka javnom internetskom mrežom, i općenito generiranje prevelike količine podatka u mreži zbog čega se kod modernih aplikacija vezanih uz koncept Interneta stvari i pojavila potreba za raspodijeljenim arhitekturnim modelom [20]. Kako je koncept računarstva u magli ponudio upravo

raspodjelu centraliziranog modela zasnovanog na računalnom oblaku, njegova je efikasna implementacija brzo postala popularna istraživačka tema. Obradom ove teme u znanstvenoj je literaturi ubrzo zaključeno da bi se kombinacijom mikroservisne arhitekture i virtualizacije zasnovane na kontejnerima najprirodnije mogao i ostvariti ovaj cilj, budući da se ovim pristupom lakše mogu ostvariti modularna raspodijeljena okruženja karakteristična za usluge Interneta stvari [21, 22].



Slika 2.1: Usporedba monolitne i mikroservisne arhitekture [23].

Mikroservisna arhitektura (slika 2.1, desno) je trend, odnosno paradigma, u razvoju računalnih sustava koja se pojavila kao odgovor na ubrzanu dinamiku u razvoju softvera. Domenski-upravljan razvoj, kontinuirana isporuka, manji razvojni timovi, virtualizacija na zahtjev, automatizacija infrastrukture i imperativ skalabilnosti sustava, potaknuli su nastanak ovog trenda koji je ponudio novi modularni pristup u implementaciji računalnih sustava [24]. Monolitni sustavi imaju određenu prednost u brzini obrade zbog komunikacije kroz zajednički memorijski prostor, međutim brža mrežna komunikacija i moderne virtualizacijske tehnologije omogućile su da se i u raspodijeljenoj arhitekturi kakvu podrazumijevaju mikrousluge dodatni teret međusobne komunikacije svede na minimalnu razinu. Glavne prednosti koje ovakav arhitekturni model omogućuje su [24]:

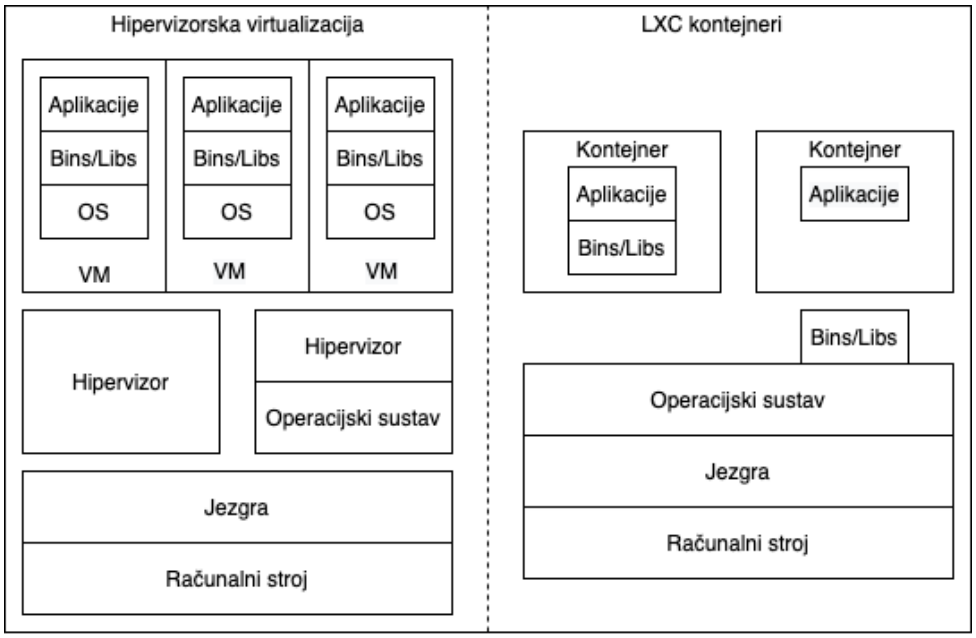
- heterogenost tehnologija - budući da je svaka mikrousluga odvojeni autonomni modul, omogućeno je korištenje različitih programskih tehnologija u razvoju cjelokupnog sustava;
- otpornost - sustav postaje otporniji zbog jednostavnije izolacije kvara, čime je smanjen utjecaj jednog problema na cjelokupan sustav;
- skalabilnost - za razliku od monolitnih sustava nije potrebno skalirati cjelokupan sustav ako dođe do povećanog volumena zahtjeva na neku od njegovih komponenata, već se odvojeno može skalirati sve njegove dijelove;

- jednostavnija implementacija - izmjene i nadogradnje sustava ne uzrokuju ponovno pokretanje cijelog sustava, već su izolirane u komponentama na kojima se izvode;
- organizacijsko poravnavanje - arhitektura sustava lakše se oblikuje prema strukturi same organizacije i njenih timova;
- komponiranost - svojstvo koje omogućuje korištenje dijelova sustava na različite načine, te njihovu lakšu prilagodbu aktualnim korisničkim trendovima;
- optimizirana zamjenjivost - dijelovi sustava se lakše nadograđuju ili po potrebi i u potpunosti zamjenjuju novim komponentama.

Suprotno navedenim prednostima postoje i nedostaci mikroservisne arhitekture, pa je tako posebno izražena kompleksnost u ostvarivanju integracije i efikasne komunikacije između mikrousluga, te u testiranju cjelokupnog sustava kojeg one tvore. Također, ovakva arhitektura zbog svoje raspodijeljenosti podrazumijeva i druge probleme karakteristične za sve raspodijeljene sustave, što se posebno odnosi na sigurnost i detekciju kvara u sustavu. Dizajn raspodijeljenog sustava kojeg tvore međusobno povezane mikrousluge zato je potrebno ostvariti s velikim stupnjem izoliranosti i otpornosti na pogreške kako bi se sustav mogao kontinuirano izvoditi, a potrebno je osigurati i pregledan nadzor procesa interakcije između povezanih mikrousluga kako bi se ubrzala detekcija mogućih neispravnosti.

Sama je mikrouslužna arhitektura u heterogenom okruženju kakvog podrazumijeva koncept računarstva u magli omogućila lakšu implementaciju modularnog sustava, ali je za postizanje dinamičnosti u takvoj okolini, upravo zbog različitosti uređaja na kojima se ovaj koncept izvodi, važan koncept virtualizacije zasnovane na kontejnerima. Iako kontejneri nisu inicijalno nastali zbog mikrousluga, nego kao odgovor na potrebu za alatom koji bi omogućio univerzalno i pouzdano pokretanje kompleksnih aplikacija u različitim okruženjima računalnih oblaka ili drugih poslužiteljskih okolina, zbog sličnih su idejnih principa brzo postali važan alat za efikasnu implementaciju mikroservisne arhitekture [25]. Jedno od temeljnih načela koje se naglašava u dokumentaciji popularnog pružatelja virtualizacije zasnovane na kontejnerima (Docker) je da svaki kontejner treba pokretati samo jedan proces kako bi se olakšala horizontalna skalabilnost i ponovna uporabljivost kontejnera, što je jasna potvrda sličnosti njihovih filozofija. Upravo zato autori u [25] zaključuju kako su mikrouslužna arhitektura i kontejnerska virtualizacija dva međusobno povezana koncepta koja vode prema zajedničkom cilju kontinuirane isporuke i operacijske efikasnosti.

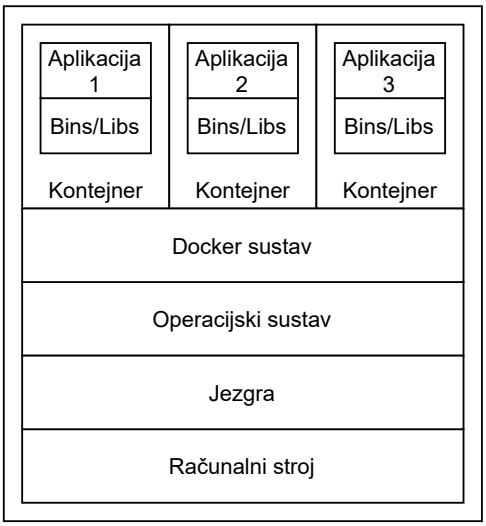
Tradicionalna hipervizorska virtualizacija omogućava podjelu resursa računalnog stroja na više odvojenih virtualnih strojeva što se može vidjeti na lijevoj strani slike 2.2. Zadaća hipervizora u ovakvoj izvedbi virtualizacije je da preslikava resurse virtualnog stroja kao što su CPU i RAM na konkretne resurse računalnog stroja na kojem se on i izvodi, te da korisniku omogući manipuliranje virtualnim strojevima. Za razliku od takvog pristupa kontejneri su ponudili jednostavniji način za virtualno razdvajanje aplikacija koji zauzima manje računalnih resursa



Slika 2.2: Usporedba hipervizorske i kontejnerske virtualizacije (LXC) [24].

za vlastito izvođenje. Postoje različite implementacije ovakvog načina virtualizacije, pa je tako na desnoj strani slike 2.2 prikazana izvedba Linux kontejnera (*LXC - Linux Containers*) u kojoj nema hipervizora i svaki je kontejner zapravo podstablo cjelokupnog stabla procesa u sustavu, kojem jezgra računalnog stroja dodjeljuje vlastite resurse sustava na kojem se izvodi. U takvom je pristupu važno naglasiti da svaki kontejner može biti različita distribucija operacijskog sustava na kojem se izvodi, ali je nužno da oni imaju istu računalnu jezgru (*Linux Kernel*) budući da se upravo u njoj nalazi stablo procesa računalnog stroja. Na slici 2.2 je također prikazano da u izvedbi kontejnerizacije pomoću LXC-a, kontejner može koristiti komponente operacijskog sustavu na kojem je pokrenut ako se u njemu koristi ista distribucija Linuxa, dok će se takve komponente nalaziti u samom kontejneru ako se koriste različite distribucije Linuxa. Zbog mogućnosti korištenja različitih distribucija istog OS-a u kontejneru i samog načina korištenja Linux kontejnera, u literaturi se često može pronaći kako je ova implementacija kontejnerizacije zapravo jednostavnija izvedba hipervizorske virtualizacije. Suprotno tome Docker se opisuje kao rješenje namijenjeno za ostvarivanje aplikacijske virtualizacije temeljene na kontejnerima, u kojem po dizajnu svaka aplikacija ima vlastiti kontejner i koje je zato namijenjeno implementaciji povezanih raspodijeljenih sustava [26]. Dodatno, Docker nudi korisnicima registar za pohranu privatnih i javno dostupnih slika (engl. *images*) u kojima su opisane konfiguracije na temelju kojih se kontejneri pokreću, te aplikaciju za njihovo jednostavno pokretanje na različitim sustavima. Zbog takvih mogućnosti i arhitekturne izvedbe prikazane na slici 2.3 Docker je postao popularan virtualizacijski alat zasnovan na kontejnerima koji omogućuje dostatan nivo elastičnosti i izoliranosti za primjenu u implementaciji usluga Interneta stvari koje se pokreću u dinamičnoj okolini između računalnog oblaka i nižih slojeva karakterističnih za računarstvo

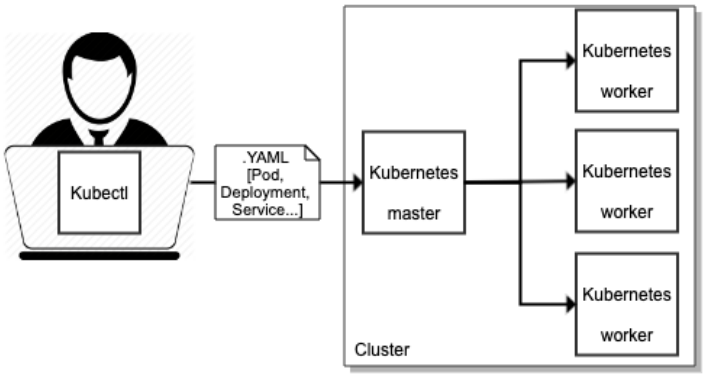
u magli. Ovakva izvedba virtualizacije zasnovane na kontejnerima u kombinaciji s mikrouslugama omogućila je postizanje lakše prenosivosti usluga u raspodijeljenoj okolini, ali je njihova orkestracija i upravljanje njihovom migracijom, te kontrola skalabilnosti sustava koji čini veći broj raspodijeljenih modula ostao jedan od izazova koji je bilo potrebno riješiti kako bi se ostvario efikasan sustav u okolini kakvu podrazumijeva koncept računarstva u magli.



Slika 2.3: Arhitekturni složaj Dockerove virtualizacije.

Danas je dostupan veći broj alata kojima se ostvaruje orkestracija kontejneriziranih usluga raspodijeljenog sustava, a koji u većini slučajeva rade na principu zadavanja željene konfiguracijske specifikacije sustava alatu koji onda samostalno upravlja rasporedom kontejnera na dostupnoj infrastrukturi (slika 2.4). Atribut koji ima važnu ulogu u ostvarivanju efikasne migracije usluga je odnos usluge i njenog stanja (engl. *statelessness*), odnosno čuva li određena usluga stanje ili ne. Migracija kontejnera usluga koje ne čuvaju stanje je jednostavna procedura koja se izvodi u dva koraka: pokretanje novog kontejnera na odredišnom čvoru, a zatim zaustavljanje starog kontejnera na izvornom čvoru [27]. Takva vrsta migracije se podrazumijeva u većini postojećih alata za orkestraciju kontejnera, pa je bez dodatne implementacije na aplikacijskom sloju njima nemoguće u okolini računarstva u magli ostvariti efikasno i automatsko raspoređivanje kontejneriziranih usluga koje čuvaju stanje. Taj je zaključak donesen i po evaluaciji najpoznatijih orkestracijskih alata: Marathon (Mesos), Kubernetes (Docker), i Docker Swarm, prema tri kriterija važna za njihovu primjenu u okruženju računarstva u magli: pridruživanje/napuštanje grozda, postavljanje usluge na točno definirani čvor, te preslikavanje resursa čvora računarstva u magli na kontejner koji se na njemu pokreće [22]. Kako se pokazalo da nijedan od promatranih alata ne ispunjava sva tri navedena kriterija, autori rada su također istaknuli kako će za mogućnost njihove primjene za ostvarivanje efikasne orkestracije usluga u okolini računarstva u magli biti potrebno napraviti značajne nadogradnje.

Osim orkestracijskih alata, danas postoji i niz softverskih radnih okvira (engl. *software*



Slika 2.4: Način rada alata za orkestraciju kontejnera Kubernetes.

frameworks) temeljenih na sličnim tehnologijama koje se također mogu iskoristiti u implementaciji sustava koji podrazumijevaju sudjelovanje entiteta na rubu mreže u cjelokupnoj arhitekturi sustava. Pružatelji usluga u računalnom oblaku prepoznali su trend djelomičnog pomicanja računalnih resursa za obradu na rub mreže, pa zato većina njih već danas nudi vlastita rješenja za integraciju rubnih uređaja s njihovim računalnim oblacima. Najpoznatiji među njima (AWS, Microsoft Azure, IBM Cloud) nude gotove programske proizvode koji omogućuju potrošačima lakšu integraciju vlastitih IoT rješenja s njihovim računalnim oblacima. IoT Greengrass [28] primjer je rješenja koje nudi AWS i prema njihovom opisu ono omogućuje proširenje usluga AWS-a na rubne uređaje kako bi oni lokalno mogli obrađivati podatke koje generiraju, a pritom bi računalni oblak koristili za izvođenje kompleksnije analitike, upravljanja i trajnog pohranjivanja podataka. Također, navedeno je i da se korištenjem ovog radnog okvira mogu ostvariti sljedeće dobiti: odgovor na lokalne događaje u gotovo stvarnome vremenu, odspojeni (engl. *offline*) način rada, sigurna komunikacija, pojednostavljeno programiranje uređaja uz podršku za kontejnere, smanjivanje troška izvođenja IoT usluga, te validacija uređaja na kojima se AWS Greengrass i izvodi. Slično rješenje nudi i Microsoft Azure svojom IoT Edge uslugom [29] koja prema njihovom opisu nudi mogućnost izvođenja usluga na rubnim uređajima korištenjem standardnih kontejnera. I u slučaju ovog radnog okvira ističu se slične dobiti, pa je tako naglašeno da se ovakvim pristupom smanjuje vrijeme potrebno za komunikaciju s poslužiteljem, brže se reagira na lokalne promjene i omogućuje se odspojeni način rada. Korištenje ovakvih radnih okvira podrazumijeva uglavnom integraciju njihovih unaprijed pripremljenih generičkih modula u vlastita rješenja korisnika, na temelju čega ona postaju dio određenog ekosustava u kojem mogu koristiti postojeće usluge i sučelja odabranog proizvođača. Zbog dobrog dizajna i široke primjene takvih sučelja, korisnicima je olakšano upravljanje vlastitim rješenjima te dodavanje nadogradnji pomoću različitih analitičkih usluga. Na taj se način omogućuje veći stupanj fleksibilnosti rješenja, ali zbog zatvorenosti koda, te arhitekture koja je ipak i dalje fokusirana na računalni oblak ovakvi su radni okviri podložni problemima kao što su pouzdanost i privatnost [20].

Softverski radni okviri otvorenog koda također se u većoj mjeri baziraju na ranije opisanim tehnologijama za orkestraciju kontejnera, ali za razliku od komercijalnih rješenja nude korisnicima veći stupanj kontrole nad vlastitim uređajima i podatkovnim tokovima njihovih aplikacija. Primjeri takvih rješenja su okviri IoFog, EdgeX, K3s, i sl., koji na službenim stranicama ističu prednosti rješenja otvorenog koda kao što su neovisnost i pouzdanost, ali zbog funkcioniranja na istom principu kojeg koriste i orkestracijski alati imaju nedostatak nemogućnosti izvedbe dinamičnog upravljanja uslugama. Prednost okvira otvorenoga koda u odnosu na komercijalna rješenja je što ne usmjeravaju arhitekturu prema računalnom oblaku i što se korisnicima omogućuje veća fleksibilnost u dizajnu arhitekture za njihova vlastita rješenja usmjerena na iskorištavanje potencijala koje nudi računarstvo u magli. Nedostatak ovakvih rješenja su učestale nadogradnje koje obično zahtijevaju i preinake u kodu rješenja koje ih koristi, što uzrokuje nestabilnost koda i smanjenu pouzdanost u nekim slučajevima.

Na temelju opisanog pregleda izvedbenih arhitektura usluga koje bi pri obradi podataka uključivale i uređaje na rubu mreže te tehnologija kojima se one mogu implementirati, može se zaključiti kako su mehanizmi za ostvarivanje povezane modularne arhitekture dostupni i dostatno razvijeni. Alati za orkestraciju i upravljanje izvođenjem modula takvih sustava u raspodijeljenoj okolini također su dostupni, ali su još uvijek u najvećoj mjeri bazirani na statičkim specifikacijama željenih konfiguracija u izvedbenoj okolini. Zato je za postizanje dinamičkog upravljanja na temelju stanja takvih okolina potrebno implementirati dio logike na aplikacijskom sloju samih usluga ili nadograditi postojeće algoritme raspoređivanja koje koriste spomenuti sustavi za orkestraciju usluga. Upravo su algoritmi raspoređivanja usluga u dinamičnoj okolini kakvu podrazumijeva računarstvo u magli popularan istraživački problem u okviru ovog koncepta koji se i obrađuje u nastavku disertacije.

2.2 Pregled istraživačkog područja

Uključivanje uređaja na nižim slojevima cjelokupne arhitekture Interneta stvari kao dio infrastrukture koji nadopunjuje računalni oblak za efikasniju isporuku usluga krajnjem korisniku predstavlja niz istraživačkih izazova. Heterogena i dinamična okolina koju čine uređaji različitih računalnih specifikacija i drugih svojstava, te različitost mogućih IoT scenarija na koje se ovaj koncept može primijeniti uzrok su kompleksnosti pronalaska optimalnog rasporeda za izvođenje usluge koji omogućuje njenu efikasnu isporuku u specifičnom slučaju. Zato su u ovom potpoglavlju izdvojeni neki od najznačajnijih postojećih znanstvenih radova koji obrađuju teme primjene računarstva u magli na okoline Interneta stvari, s naglaskom na problemima relevantnim za ovu disertaciju.

2.2.1 Prosljeđivanje zahtjeva

Jedna od inicijalnih tema u razvoju koncepta računarstva u magli je prosljeđivanje zahtjeva (engl. *task offloading*) prema čvoru raspodijeljene arhitekture koji može optimalno izvršiti njegovu obradu. Kako je koncept računarstva u magli inicijalno predstavljen kao nadogradnja računalnog oblaka uređajima na nižim slojevima arhitekture, bilo je potrebno istražiti načine odabira konkretnog čvora na kojem bi se određeni zahtjev u specifičnom slučaju i obradio. U takvoj okolini se prosljeđivanje zahtjeva može odvijati u različitim smjerovima ovisno o vrsti same aplikacije i trenutno raspoloživim resursima čvorova [30].

Prva opcija podrazumijeva prosljeđivanje obrade primljenih zahtjeva s uređaja na nižim slojevima prema uređajima na višim slojevima ili u krajnjem slučaju do računalnog oblaka. Ovakav smjer prosljeđivanja obrade obično se događa kada je aplikacija već raspoređena u okolinu blizu izvora podataka, ali zbog nedostatka resursa za obradu cjelokupnog ulaznog podatkovnog toka, uređaj prosljeđuje zahtjeve na čvorove u višim slojevima arhitekture. Senzorski uređaji krajnjeg korisnika, ili strojevi u industrijskim slučajevima primjene, prosljeđuju podatke za obradu i pohranu na pametne telefone ili čvorove računarstva u magli koji se nalaze u njihovoj blizini. Takva obrada obično podrazumijeva jednostavne funkcionalnosti izvještavanja i kontrole podataka, ali može uključivati i izvođenje zahtjevnih funkcija i analitike, što na čvorovima nižih slojeva često uzrokuje preopterećenje zbog njihovih ograničenih procesorskih i memorijskih resursa [31, 32]. U takvim slučajevima potrebno je proslijediti dio opterećenja na uređaje koji imaju više raspoloživih računalnih resursa kako bi aplikacija mogla ispravno funkcionirati.

Druga opcija podrazumijeva međusobno prosljeđivanje zahtjeva između čvorova računarstva u magli. Ovakav smjer prosljeđivanja prometa obično je karakterističan za scenarije kod kojih se ravnomjerno raspoređuje opterećenje u sustavu (engl. *load balancing*) ili je posljedica dostupnosti različitih podataka u različitim okolinama. Ravnomjerni raspored opterećenja u sustavu često je cilj raspodijeljenih arhitektura, a temeljni princip ovog pristupa u okolini računarstva u magli je da čvor koji se nalazi pod većim opterećenjem proslijedi dio ulaznih zahtjeva na druge čvorove koji su u tom trenutku pod manjim opterećenjem [33]. Time se sprječava nastanak "uskog grla" u raspodijeljenom sustavu i omogućuje se efikasna obrada svih zahtjeva u situacijama kada bi inače određeni čvor uzrokovao zagušenje. Drugi razlog prosljeđivanja zahtjeva je obično posljedica mobilnosti krajnjih uređaja. Čvorovi tako mogu mijenjati lokalnu okolinu u kojoj se nalaze i povezivati se na različite čvorove računarstva u magli. Kako bi čvorovi računarstva u magli omogućili ispravnu funkcionalnost aplikacija u takvim situacijama, potrebno je također omogućiti i migraciju njihovih zahtjeva ili korisničkih profila između različitih okolina [34].

Treća opcija odnosi se na prosljeđivanje obrade iz računalnog oblaka prema čvorovima na nižim slojevima arhitekture. Ideja koncepta računarstva u magli zasnovana je upravo na ovom principu koji podrazumijeva izvođenje aplikacijskih funkcija na nižim slojevima što je u većini

slučajeva posljedica naglašene potrebe za niskim kašnjenjem [35]. Često se za ovu primjenu navodi primjer kontrole cestovnog prometa u kojem se obrada zahtjeva prebacuje na čvorove uz same prometnice ako za njeno izvođenje nisu potrebne informacije cjelokupnog sustava pohranjene na računalnom oblaku [36].

Posljednji smjer prosljeđivanja zahtjeva podrazumijeva obostranu razmjenu između računalnog oblaka i uređaja računarstva u magli. Ovakvo prosljeđivanje obrade primjenjuje se najčešće u slučajevima kada u istom scenariju može postojati potreba za centralnom obradom, ali i za niskim kašnjenjem prijenosa podataka [37]. U takvim scenarijima dio obrade se izvodi na čvorovima računarstva u magli, dok se zahtjevi s različitih čvorova računarstva u magli za koje kašnjenje nije kritičan parametar izvode u računalnom oblaku. Drugi karakterističan primjer u kojem se primjenjuje obostrano prosljeđivanje zahtjeva su scenariji u kojima efikasnost obrade ovisi o promjenjivim parametrima mreže. U takvim se scenarijima ovisno o trenutnim vrijednostima parametara mreže obrada prosljeđuje na dio arhitekture koji može isporučiti najefikasnije rezultate.

Uz svaki od navedenih smjerova prosljeđivanja obrade u okolini računarstva u magli, važno je primijeniti i odgovarajuću strategiju odabira čvora koji u određenom trenutku može ponuditi najefikasniju kvalitetu usluge na dostupnoj infrastrukturi. U dostupnoj znanstvenoj literaturi primjenjivani su različiti algoritmi za ostvarivanje ovog cilja, a njihov pregled i kategorizaciju opisali su autori u [38]. Tako su kao mogući pristupi za donošenje odluke o najprikladnijem čvoru navedeni algoritmi zasnovani na Markovljevim lancima, algoritmi ostvareni pomoću težinskih grafova, optimizacijski algoritmi temeljeni na različitim parametrima, primjena dubokog učenja, te kvalitativna usporedba obrade na različitim dijelovima arhitekture. Ipak, u postojećim je radovima uglavnom promatran samo jedan smjer prosljeđivanja obrade od prethodno navedenih opcija, te se za odluku o najprikladnijem čvoru za izvođenje obrade u obzir uzimao samo određeni dio relevantnih parametara. Zato je u nastavku istraživanja potrebno u obzir uzeti što veći broj relevantnih kontekstnih parametara te obuhvatiti sve oblike preusmjeravanja obrade kako bi se u konačnici postiglo optimalno rješenje.

Opisani smjerovi preusmjeravanja obrade u raspodijeljenoj okolini računarstva u magli opisuju različite načine integracije uređaja na nižim slojevima, odnosno različite mogućnosti iskorištavanja potencijala ovog koncepta. Ipak, ovakav je pristup uglavnom primjenjivan u inicijalnoj fazi istraživanja ovog koncepta. Prosljeđivanje zahtjeva i preusmjeravanje podatkovnih tokova može dodatno opteretiti mrežu, dok se orkestracijom izvođenja raspodijeljenih usluga ovakav mrežni promet može smanjiti ili u potpunosti eliminirati. Zato je u novijim istraživanjima orkestracija izvođenja usluga popularniji istraživački izazov, koji je zapravo i tema ove disertacije.

2.2.2 Kontekstno-svjesno raspoređivanje usluga

Virtualizacija zasnovana na kontejnerima i alati za orkestraciju kontejneriziranih usluga stvorili su preduvjete za jednostavniju implementaciju migracije usluga što je potaknulo istraživanje problema njihova raspoređivanja u okolini računarstva u magli. Problem raspoređivanja može se definirati kao pronalazak optimalnog rješenja za raspored *n* komponenti usluge na dostupnim čvorovima računarstva u magli s različitim resursnim mogućnostima, uzimajući u obzir specifične parametre kvalitete usluge kako bi se u konkretnom slučaju optimizirala ciljna funkcija raspoređivanja [39]. Kako bi se moglo što efikasnije odrediti optimalan raspored u dinamičnoj i heterogenoj okolini kakvu podrazumijeva koncept računarstva u magli potrebno je u obzir uzeti informacije o različitim kontekstima koji mogu utjecati na kvalitetu isporuke aplikacijskih rješenja Interneta stvari. Kontekst podrazumijeva sve informacije kojima se može opisati stanje određenog entiteta relevantnog za interakciju korisnika i aplikacije, uključujući i njih same [40, 41]. Kontekstno-svjesni (engl. *context-aware*) sustavi ili aplikacije tako moraju biti osjetljive na promjene u relevantnoj okolini kako bi se povećala efikasnost sustava, te kako bi njegove performance bile optimizirane za trenutno stanje relevantnih parametara okoline u kojoj se on i izvodi [42]. Uzimanjem kontekstnih informacija u obzir pri određivanju rasporeda izvođenja određene usluge omogućuje se efikasnija isporuka usluge korisnicima prema relevantnim parametrima kvalitete usluge u konkretnom slučaju.

Sveobuhvatan pregled radova i postignutog napretka u istraživanju kontekstno-svjesnog raspoređivanja u okolini računarstva u magli napravljen je u radu [41]. Tako su na temelju dostupnih radova izdvojeni sljedeći konteksti relevantni za okolinu računarstva u magli i raspoređivanje usluga u njoj:

- *kontekst korisnika* - informacije koje određuju korisnički profil u okviru konkretnog scenarija (npr. mobilnost, povijest aktivnosti, vjerojatnost isključivanja, i dr.);
- *kontekst aplikacije* - operativni zahtjevi određene usluge na temelju kojih se mogu odrediti optimalni resursi potrebni za njenu kvalitetnu isporuku (npr. arhitektura, osjetljivost na kašnjenje, kompleksnost obrade, mrežni zahtjevi, itd.);
- *kontekst okoline* - svi parametri koji dodatno opisuju okolinu izvođenja između korisnika i usluge, a koji mogu biti relevantni za određivanje rasporeda u konkretnom scenariju (npr. lokacije relevantnih entiteta, vrijeme, dostupni resursi, itd.);
- *kontekst mreže* - mrežni atributi koji opisuju mrežne uvjete izvedbene okoline koji imaju važnu ulogu u isporuci usluga u računarstvu u magli (npr. kapacitet veze, jačina signala, kašnjenje u mreži, gubitci paketa, i dr.);
- *kontekst uređaja* - karakteristike uređaja koje mogu biti važne za efikasno raspoređivanje usluga (npr. stanje baterije, dostupni resursi, mobilnost, dostupni mrežni kapacitet, itd.).

Autori navode i da je raspoređivanje, odnosno komponenta koja izvodi raspoređivanje - raspoređivač (engl. *scheduler*), kontekstno-svjesna ako ima mogućnost razmatranja kontekstnih

informacija različitih dostupnih entiteta prema kojima može prilagoditi raspored kako bi se postiglo što efikasnije upravljanje dostupnim resursima. Također, predstavljena je i arhitektura sustava za kontekstno-svjesno raspoređivanje usluga koja se sastoji od dvije komponente: jezgrene komponente takvog sustava (engl. *context-aware engine*) i samog raspoređivača (engl. *context-aware scheduler*). *Jezgrena komponenta* obrađuje kontekstne informacije kroz četiri karakteristične faze:

- prikupljanje kontekstnih podataka od uključenih entiteta,
- spremanje prikupljenih podataka u repozitorij,
- razumijevanje prikupljenih kontekstnih podataka kako bi se dobile konkretne informacije o stanju okoline računarstva u magli i
- analiza kontekstnih informacija i davanje preporuka raspoređivaču na temelju dobivenih rezultata.

Raspoređivač pravovremeno upravlja rasporedom izvođenja komponenti usluge na dostupnim resursima optimizirajući njihovo opterećenje i prilagođavajući raspored trenutnom stanju okoline. Zatim je napravljena analiza postojećih radova koji opisuju vlastite implementacije ili teorijske pristupe kontekstno-svjesnom raspoređivanju u okolini računarstva u magli na temelju sljedećih parametara: primjenjivana metoda ili algoritam, konteksti koji su uzeti u obzir, konkretni parametri konteksta, promatrani slučaj primjene, metrika performanci, alat za evaluaciju, te prednosti i ograničenja predstavljenog pristupa. Na temelju provedene analize navedeni su sljedeći nedostaci postojećih pristupa, odnosno izazovi koje će u nastavku razvoja biti potrebno riješiti u okviru ovog istraživačkog problema:

- raspoređivanje u kojem se u obzir uzima što veći opseg parametara koji određuju navedene relevantne kontekste i metriku evaluacije,
- bolja podrška za mobilnost entiteta uključenih u karakterističnu okolinu računarstva u magli,
- razvoj modela za procjenu dostupnosti resursa u različitim situacijama,
- razvoj mehanizama za efikasnije raspoređivanje na temelju resursnih ograničenja čvorova i zahtjeva same usluge,
- optimizacija potrošnje energije u sustavu,
- međusobno dijeljenje konteksta između čvorova računarstva u magli kako bi se omogućila interoperabilnost,
- te daljnji razvoj primjene kontekstno-svjesnih politika raspoređivanja koje optimiziraju raspored komponenti usluge kako bi se ostvarila što veća kvaliteta u njihovoj isporuci.

Prethodno je u potpoglavlju 2.1 opisano kako je virtualizacija zasnovana na kontejnerima omogućila lakšu prenosivost komponenti usluge u heterogenoj okolini računarstva u magli i bolju podršku za integraciju izvođenja usluge u raspodijeljenom okruženju. Zato su posebno zanimljivi postojeći pristupi u kojima je implementacija algoritma raspoređivanja u obzir uzi-

mala kontejnerizirane komponente.

Jedan primjer takvog pristupa je [43] gdje je opisan algoritam raspoređivanja kontejneriziranih komponenti NAS (engl. *network-aware scheduling*) koji u obzir uzima parametre stanja mreže kako bi postigao efikasniju isporuku usluge korisnicima. Implementacija algoritma izvedena je kao nadogradnja osnovnog postupka raspoređivanja u Kubernetesu, pa je tako pristup namijenjen upravo kontejneriziranim uslugama dok je sama evaluacija napravljena na slučaju primjene pametnog grada.

Komponenta koja izvodi raspoređivanje u Kubernetesu (kube-scheduler) kroz dvije faze odabire čvor na kojem će pokrenuti kapsulu (*pod*) u kojoj je enkapsulirana jedna ili više kontejneriziranih komponenti: filtriranje i ocjenjivanje [44]. U prvoj fazi eliminiraju se dostupni čvorovi koji ne mogu pokrenuti kapsulu zbog nedostatka procesorskih ili memorijskih resursa, ili nekog drugog uvjeta navedenog u specifikaciji kapsule. U drugoj fazi se među preostalim čvorovima odabire konačni čvor na kojem će kapsula biti pokrenuta. Ta se odluka također donosi uzimajući u obzir samo dostupne procesorske i memorijske resurse preostalih čvorova. Ipak, obje navedene faze se mogu dodatno konfigurirati na različite načine: dodavanjem dodatnih predikata i/ili prioriteta koje je potrebno uzeti u obzir u obje faze, implementacijom specifične komponente za raspoređivanje koja bi se pokretala umjesto zadane kube-scheduler komponente, i proširenjem zadanog postupka raspoređivanja procesom koji bi se pozivao prije donošenja konačne odluke o čvoru na kojem će se Kubernetesova kapsula i pokrenuti.

Upravo su proširenjem zadanog postupka raspoređivanja autori u [43] implementirali vlastiti algoritam i to na sljedeći način:

- Najprije je svaki čvor koji je bio dio infrastrukture za izvođenje usluga, odnosno dio Kubernetesovog grozda dodatno opisan oznakama (engl. *label*) "*Min*", "*Med*", ili "*High*" za parametre označene ključnim riječima "*CPU*" i "*RAM*", oznakama "*Cloud*" ili "*Fog*" za parametar označen ključnom riječi "*Device Type*", te konačno vrijednostima očekivanog kašnjenja ovisno o lokaciji odakle je upućen zahtjev za parametar označen ključnom riječi "*RTT*" (engl. *Round Trip Time*);
- U samom postupku raspoređivanja dodana je logika proširenja prema kojoj bi se po filtriranju čvorova koji mogu pokrenuti kapsulu pozivao proces za konačni odabir čvora na kojem će kapsula biti i pokrenuta. U tom bi se procesu u obzir uzimale vrijednosti prethodno navedenih parametara, s posebnim naglaskom na parametar *RTT* kako bi se odabrao onaj čvor koji bi uslugu mogao isporučiti s najmanjim kašnjenjem.

Usporedbom opisanog pristupa sa zadanim raspoređivanjem koje se primjenjuje u Kubernetesu demonstrirane su uštede u kašnjenju pri isporuci odgovora na primljene zahtjeve, čime se potvrđuje podizanje efikasnosti u isporuci usluga u računarstvu u magli primjenom kontekstno-svjesnog raspoređivanja.

Sličan pristup predstavljen je u [45] gdje je također implementiran algoritam koji u obzir

uzima parametre stanja mreže kako bi postigao efikasniju isporuku usluge korisnicima. Cilj predstavljenog algoritma bio je odabrati čvor za izvođenje Kubernetesove kapsule koji na temelju podataka o trenutnom stanju mreže može u zadanom vremenskom roku izvršiti potrebnu obradu u konkretnom slučaju. Za implementaciju ovakvog pristupa autori su u opisu konfiguracije kapsule upisivali podatak o ciljanom roku izvršenja obrade, dok bi raspoređivač na temelju informacija o trenutnom stanju mreže onda pronalazio one dostupne čvorove na kojima bi usluga konkretne kapsule mogla zadovoljiti zadane uvjete. Informacije o stanju mreže kontinuirano su se ažurirale pomoću alata iperf. Tako je svaki čvor za obradu u Kubernetesovom grozdu (engl. *worker*) imao pokrenutu komponentu *iperf-agent*, koja je komponenti *iperf-server* pokrenutoj na upravljačkom čvoru grozda (engl. *master*) na zahtjev dostavljala ažurirane informacije o dostupnim performancama mreže na tom čvoru. Na temelju tako dobivenih informacija o performancama mreže koje koriste dostupni čvorovi za obradu, raspoređivač je mogao odabrati optimalni čvor koji bi mogao ispuniti obradu usluge koja se raspoređuje u zadanom roku.

Navedeni pristupi demonstrirali su iskoristivost sustava za orkestraciju kontejneriziranih usluga Kubernetes za raspoređivanje komponenti u okolini računarstva u magli, dok se algoritam raspoređivanja implementirao kao nadogradnja zadanom postupku kojeg primjenjuje osnovni raspoređivač - kube-scheduler. Evaluacija takvih rješenja izvedena je u simuliranoj okolini zbog čega je bilo moguće direktno izvesti pridruživanje čvorova računarstva u magli u Kubernetes grozd, budući da se svi čvorovi nalaze u istoj mreži i imaju statične IP adrese. U realnom scenariju, uređaji računarstva u magli se mogu nalaziti u različitim mrežama, uključujući i lokalne mreže u kojima su povezani uređaji iza usmjerivača (engl. *router*). Usmjerivači obično izvode dinamičko mapiranje adresa i vrata (engl. *Network Address Translation*) ili sadrže vatrozide (engl. *firewall*) pa uređaji u njihovoj lokalnoj mreži obično nemaju statične IP adrese kojima se može pristupiti iz javne mreže bez implementacije dodatnih procedura kao što je primjerice prosljeđivanje vrata (engl. *port forwarding*). Zato je za korištenje orkestracijskog alata Kubernetes u okolini računarstva u magli potrebno prethodno stvoriti preklapajuću mrežu (engl. *overlay network*) kako bi svaki čvor računarstva u magli mogao najprije postati dio simuliranog grozda, u kojem su svi čvorovi dostupni centralnom čvoru koji izvodi raspoređivanje [46]. To se može ostvariti implementacijom virtualne privatne mreže (VPN) koja bi sadržavala sve uključene entitete grozda - centralni upravljački čvor (*master*) i sve čvorove-radnike koji žele ponuditi vlastite resurse za obradu (*workers*) [47, 48, 49]. Time se omogućuje komunikacija upravljačkog čvora i čvorova radnika, odnosno izdavanje naredbi iz komponente pokrenute na upravljačkom čvoru *kube-apiserver* prema komponentama *kubelet* pokrenutim na čvorovima-radnicima, što je preduvjet za ispravnu funkcionalnost alata Kubernetes.

Ipak, ovakav pristup stvara dodatno opterećenje uz same komponente alata Kubernetes za čvorove računarstva u magli koji često imaju ograničene dostupne resurse, te potencijalno smanjuje pouzdanost i povećava kašnjenje zbog usmjeravanja cjelokupnog prometa komunikacije

kroz poslužitelj virtualne mreže. Zato je pri implementaciji ovakvog pristupa potrebno potvrditi njegovu efikasnost na realnom scenariju kako bi verificirala njegova stvarna učinkovitost, odnosno da su ostvarena poboljšanja kvalitete usluge veća od mogućih nedostataka uzrokovanih implementacijom virtualne privatne mreže.

Poglavlje 3

Konteksti u računarstvu u magli

Za efikasan razvoj i implementaciju algoritama raspoređivanja u arhitekturi koja uključuje računarstvo u magli potrebno je komponenti koja izvodi raspoređivanje omogućiti dostupnost relevantnih kontekstnih informacija. Prethodno je u potpoglavlju 2.2.2 opisana specifikacija konteksta relevantnih za računarstvo u magli definirana u radu [41]. Navedeno je pet različitih skupina kontekstnih informacija koje određuju karakterističnu izvedbenu okolinu računarstva u magli. Ipak, uzimajući u obzir parametre koji opisuju kontekst mreže i kontekst okoline, moguće ih je pridružiti preostalim navedenim kontekstnim skupinama u okviru scenarija Interneta stvari. Tako je u ovom potpoglavlju napravljena analiza tri osnovna konteksta relevantna za izvođenje usluga Interneta stvari i njihovo raspoređivanje u okolini računarstva u magli, te parametri koji opisuju njihovo stanje u različitim slučajevima primjene.

3.1 Kontekst uređaja

Kako karakteristična arhitektura Interneta stvari sadrži u pravilu obično tri sloja, uređaje koji sudjeluju u njenoj izvedbi treba promatrati najprije u kontekstu sloja kojem pripadaju (slika 3.1). U najnižem sloju postoje dvije osnovne skupine uređaja: senzori i aktuatori. Senzori su osjetila koja omogućuju praćenje različitih prirodnih fenomena na temelju kojih stvaraju odgovarajuće specifične podatke koje je moguće računalno interpretirati i obraditi. Aktuatori su uređaji specifične namjene koji djeluju kao produžena ruka korisnika, odnosno njima je omogućeno udaljeno djelovanje korisnika u stvarnim fizičkom procesima. S obzirom na današnju proširenost samog koncepta Interneta stvari te na činjenicu da su senzori i aktuatori bili prisutni i prije pojave ovog koncepta u različitim scenarijima uporabe, danas postoji velik broj takvih uređaja namijenjen raznim specifičnim primjenama.

Iako su upravo senzori i aktuatori stvorili uvjete za proširenje djelovanja Interneta iz isključivo virtualnih računalnih okolina u stvarne fizičke procese, te tako omogućili razvoj koncepta Interneta stvari, oni su samo ulazni i izlazni kanali za izvođenje specifične logike koja je im-

Slika 3.1: Osnovni arhitekturni model korišten u implementacijama primjena Interneta stvari.

plementirana na računalnim uređajima u višim slojevima. Takvi uređaji nalaze se u druga dva sloja karakteristične arhitekture Interneta stvari, te ih se može podijeliti prema različitim kriterijima ovisno o specifičnom slučaju uporabe. U početnim modelima izvedbe arhitektura Interneta stvari, mikrokontroleri na koje su bili priključeni senzori i aktuatori nisu imali dovoljno računalnih resursa za ikakvu obradu podataka ili za komunikaciju internetskim protokolom (IP), pa su se u drugom sloju nalazili mrežni prilazi (engl. *gateways*) čija je primarna namjena bila posrednička komunikacija između takvih mikrokontrolera na nižem sloju i poslužiteljskih strojeva u kojima se izvodila obrada i aplikacijska logika na višem sloju. Takva uloga ovih entiteta omogućila je da svaki računalni uređaj koji je povezan na internetsku mrežu, a koji istovremeno može komunicirati i nekim od jednostavnijih protokola koji se koriste za komunikaciju s mikrokontrolerima, bude iskorišten kao mrežni prilaz. U najvišem sloju ovakvih početnih arhitekturnih modela Interneta stvari nalazili su se računalni strojevi s dostatnom količinom računalnih resursa za obradu i pohranu svih podataka potrebnih za izvođenje specifične aplikacijske logike.

Razvojem računalne infrastrukture tijekom godina te pojavom sve specifičnijih slučajeva primjene koncepta Interneta stvari, ovakvi su se inicijalni modeli arhitektura korištenih u okviru ovog koncepta modernizirali. Uređaji korišteni u svim slojevima opisane arhitekture su vremenom modernizirani većom količinom računalnih resursa boljih performanci, pa se tako danas jednostavnija obrada može izvoditi i na samim krajnjim uređajima. Takav trend brzog napretka uređaja koji čine infrastrukturu u različitim primjenama Interneta stvari bio je ključni preduvjet koji je zapravo i omogućio izvedivost primjene modernijih koncepata računarstva na rubu, te računarstva u magli u okviru ovog koncepta. U tablicama 3.1 i 3.2 prikazani su primjeri razvoja jednog tipičnog mikrokontrolera koji se koristi kao krajnji uređaj (Waspmote), te uređaja koji je

Tablica 3.1: Usporedba hardverskih specifikacija različitih verzija mikrokontrolera Waspmote [50] [51]

Mikrokontroler	Waspmote v1.1	Waspmote v1.2	Waspmote v1.5
Godina izdavanja	2009	2012	2016
Frekvencija	8 Mhz	14 Mhz	14 Mhz
SRAM	8 kb	8 kb	8 kb
EEPROM	4 kb	4 kb	4 kb
FLASH	128 kb	128 kb	128 kb
Potrošnja energije (ON; Sleep; Deep sleep; Hibernate)	9mA; 62uA; 62uA; 0,7uA	15mA; 55uA; 55uA; 0,06uA	17mA; 30uA; 33uA; 7uA
Kriptografska autentifikacija	Ne	Ne	Da

Tablica 3.2: Usporedba hardverskih specifikacija uređaja Raspberry Pi (model B) [52]

Model	Raspberry Pi 1B	Raspberry Pi 2B	Raspberry Pi 3B	Raspberry Pi 4B
Godina izdavanja	2012	2015	2016	2019
CPU	1 x ARM1176JZF-S 700 MHz	4 x Cortex-A7 900 MHz	4 x Cortex-A53 1.2 GHz	4 x Cortex-A72 1.5 Ghz
SDRAM	512 MB	1 GB	1 GB	1-4 GB
Ethernet/WiFi	Da/Ne	Da/Ne	Da/Da	Da/Da
Potrošnja energije	700 mA (3.5 W)	220-820 mA (1.1-4.1 W)	300 mA-1.34 A (1.5-6.7 W)	600 mA-1.25 A (3-6.25 W)
Bluetooth	Ne	Ne	4.1 BLE	5.0

u velikom broju postojećih primjena Interneta stvari bio korišten kao mrežni prilaz (Raspberry Pi). U tablicama su odabrani najrelevantniji parametri koji pokazuju tendenciju proizvođača za unapređenjem specifičnih parametara njihovih uređaja prema njihovoj specifičnoj namjeni za primjenu u opisanoj arhitekturi (potrošnja energije kod krajnjih uređaja, te procesorska moć i komunikacijski moduli u slučaju posredničkih uređaja).

Paralelno sa spomenutom modernizacijom uređaja na nižim slojevima, računalni oblak je postupno zamijenio u većoj mjeri lokalna poslužiteljska računala [53]. Njegovom pojavom dodatno je ubrzan razvoj ICT rješenja budući da se početno ulaganje u infrastrukturu smanjilo, a dodatno je olakšana i elastičnost upravljanja potrebnim računalnim resursima ovisno o dinamici razvoja takvih projekata. Računalni oblak omogućio je tako pristupačnost računalnoj infrastrukturi sa znatno većim brojem resursa dostatnih za skladištenje velikih količina podataka te izvođenje kompleksnije aplikacijske logike u njihovoj obradi. Ipak, budući da su poslužitelji

računalnih oblaka često fizički dislocirani od okolina za koje su rezultati takve obrade relevantni, što je posebno naglašeno u scenarijima Interneta stvari, sam prijenos podataka u takvim slučajevima može predstavljati preveliku prepreku za postizanje potrebne efikasnosti.

Upravo je ovaj problem bio jedan od temeljnih pokretača ideje o konceptu računarstva u magli, u čijem je arhitekturnom modelu računalni oblak proširen resursima uređaja koji se nalaze u okolinama bliže korisniku. Time je stvorena mogućnost za minimiziranje kašnjenja odgovora zbog fizičke udaljenosti čvora na kojem se izvodi obrada, ali je i povećana kompleksnost upravljanja obradom zbog raspodijeljenosti takve arhitekture. Upravljanje se u takvoj arhitekturi može izvoditi na različite načine pomoću virtualizacijskih alata namijenjenih upravo orkestraciji i upravljanju izvođenja usluga u raspodijeljenim sustavima ili implementacijom međusobne komunikacije između sudjelujućih entiteta na aplikacijskom sloju. Jedan od važnih parametara koje treba uzeti u obzir pri odabiru načina implementacije upravljanja izvođenjem usluga u ovakvoj okolini su resursni kapaciteti uređaja koji sudjeluju u izvedbi sloja računarstva u magli. Budući da takvi uređaji nemaju veliku računalnu moć kao što je slučaj u računalnom oblaku, nego znatno manju i ograničenu količinu resursa, pri upravljanju izvođenjem usluga na njima važno je u obzir uzeti njihova ograničenja.

U postojećoj literaturi ne postoji jedinstvena specifikacija uređaja na kojima se može izvoditi sloj računarstva u magli već okvirna definicija potrebnih atributa koje oni trebaju sadržavati. Tako se u [54] navodi da, zbog još uvijek rane faze istraživanja ovog koncepta, nema puno praktičnih izvedbi prema kojima bi se mogao izdvojiti zajednički opis takvih uređaja, ali se zato kao realna pretpostavka navode sljedeća obilježja:

- sloj računarstva u magli čine inteligentni uređaji koji imaju mogućnost računanja, obrade i pohrane podataka te usmjeravanja i prosljeđivanja podatkovnih paketa prema višim slojevima;
- umreženi uređaji sloja računarstva u magli imaju mogućnost međusobnog dijeljenja mrežnog opterećenja, te tereta za obradu i pohranu podataka;
- uređaji ovog sloja omogućuju optimalnu podršku za mobilnost krajnjih uređaja.

U [55] je napravljena opširna analiza razvoja računarstva u magli i dotadašnjih znanstvenih radova vezanih uz ovaj koncept, a u opisu konfiguracije čvorova koji čine ovaj sloj navode se sljedeće vrste uređaja:

- poslužitelji sloja računarstva u magli (engl. *fog servers*) - geografski raspoređeni uređaji na mjesta veće potražnje za određenim uslugama (stajališta autobusa, trgovački centri, itd.);
- mrežni uređaji (engl. *networking devices*) - mrežni uređaji (usmjeritelji, komutatori, itd.) koji osim svojih primarnih namjena mogu ponuditi i infrastrukturu za izvođenje potrebnih zadaća obrade ili pohrane podataka;
- "cloudlets" - uređaji zamišljeni kao potpora za proširenje usluga izvođenih u računalnom

oblaku prema korisnicima mobilnih uređaja;
- bazne stanice - postojeće bazne stanice proširene resursima za obradu i pohranu podataka;
- vozila - računalni resursi dostupni u većini modernih automobila mogli bi se iskoristiti kao čvorovi za obradu na rubu mreže ako bi se privatnost podataka mogla garantirati.

Uz velik broj sličnih znanstvenih radova, također fokusiranih na opis koncepta računarstva u magli i njegovih primjena, uređaji na kojima bi se predstavljene ideje i ostvarile u praksi bili su rijetko opisivani. Kako za izvedbu implementacija arhitektura koje uključuju sloj računarstva u magli, definicija infrastrukture na kojima bi se njihova logika i izvodila ima važnu ulogu, autori rada *"What is a Fog Node? A Tutorial on Current Concepts towards a Common Definition"* [56] koncentrirali su se upravo na definicije i izazove koji se prema tadašnjoj literaturi vežu konkretno uz čvorove na kojima bi se u praksi i izvodio ovaj koncept. U tom se radu najprije ističe kako je za definiciju čvora računarstva u magli potrebno da on efektivno može kontrolirati skupinu krajnjih uređaja s jedne strane, a s druge, da može ostvariti pristup računalnom oblaku. Zatim se naglašava važnost krajnjih uređaja povezanih na sam čvor računarstva u magli u postupku definicije njegove uloge, pri čemu su oni podijeljeni u tri skupine: 1) jednostavni proizvođači, odnosno primatelji podataka, 2) pametni krajnji uređaji koji imaju mogućnost obrade vlastitih lokalnih podataka primljenih sa senzora koji se nalaze na njima, i 3) krajnji uređaji koji nude vlastite računalne resurse za izvođenje drugih usluga u raspodijeljenom sustavu. Konačno autori navode sljedeću definiciju:

Čvor računarstva u magli podrazumijeva raspodijeljene entitete računarstva u magli koji omogućuju izvođenje usluga ovog koncepta, a formiran je jednim ili s više fizičkih uređaja koji imaju mogućnost senzoriranja i obrade podataka. Svi fizički uređaji koji čine jedan takav čvor računarstva u magli povezani su različitim mrežnim tehnologijama (žičnim ili bežičnim), te su tako agregirani u jedinstvenu apstrakciju logičkog entiteta koji može izvoditi raspodijeljene usluge na isti način kao i da se izvode na jedinstvenom fizičkom uređaju.

Osim navedene definicije u radu se navode i otvoreni izazovi koje je dodatno potrebno istražiti, a potom i konkretnije definirati, kako bi se u konačnici čvor računarstva u magli mogao konceptualizirati u potpunosti. Tako se kao najveći izazov ističe definiranje načina za ostvarivanje efikasne virtualizacijske konfiguracije koja bi omogućila učinkovito objedinjenje više krajnjih uređaja i samog čvora računarstva u magli u jedinstvenu reprezentaciju. Preduvjet za ispunjenje toga cilja je još jedan istaknuti izazov, a to je postizanje jedinstvenog ili standardiziranog načina za agregaciju dostupnih računalnih resursa u raspodijeljenoj okolini, te njihova logička apstrakcija prema upravljačkom sloju u računalnom oblaku. Drugi izazovi koji se također navode ovom kontekstu su i potreba za implementacijom učinkovite podrške mobilnim uređajima koji bi sudjelovali kao infrastrukturni čvorovi računarstva u magli, način efikasne implementacije mrežne apstrakcije u očekivanim multi-protokolnim okruženjima, te postizanja općenito veće kvalitete usluge i više razine sigurnosti u sustavu.

Iako je u spomenutom radu specificirana glavnina relevantnih faktora koje je potrebno uzeti u obzir pri definiranju konteksta uređaja u okolini računarstva u magli, također je istaknuto i da će razina utjecaja svakog od opisanih parametara uvelike ovisiti o specifičnom scenariju uporabe na koji će se razvijano rješenje i primijeniti. Osim samog scenarija uporabe, veliki utjecaj ima i sam model cjelokupne okoline u kojoj se izvodi određeni algoritam specifičnog rješenja u konkretnom slučaju. Zato konkretne definicije i primjere načina modeliranja pojedinih parametara uređaja najbolje daju radovi u kojima su opisane praktične izvedbe eksperimenata primjene računarstva u magli. U njima su opisani korišteni modeli namijenjeni implementaciji različitih algoritama u kojima se čvorovi računarstva u magli specificiraju kvalitativnim i mjerljivim parametrima. Tako su u [57] čvorovi računarstva u magli opisani sljedećim parametrima:

- memorija za pohranu (oktet - engl. *byte*),
- kapacitet obrade ili maksimalna brzina obrade (MIPS - *Million Instructions Per Second*),
- radna memorija (oktet - engl. *byte*),
- broj procesorskih jedinica,
- brzina obrade procesorske jedinice (MIPS).

Kako su navedeni atributi tipični kvalitativni parametri računalnih uređaja, jasno je da će upravo oni, uz virtualizacijske alate kojima će se agregirati uređaji raspodijeljene okoline u jedinstvene apstraktne reprezentacije, biti glavne odrednice opisa konteksta sudjelujućih uređaja u okolini računarstva u magli. U nastavku su navedeni svi relevantni parametri kojima je određen kontekst uređaja u raspodijeljenoj okolini koja uključuje sloj računarstva u magli, uz dodatna pojašnjenja njihove primjene i uloge u konkretnim implementacijama mogućih izvedbenih modela.

3.1.1 Parametri konteksta uređaja

Procesor (CPU)

Procesor je osnovni entitet svakog računalnog uređaja, a osnovni parametri njegove tehničke specifikacije su frekvencija njegova rada i broj jezgri koje on sadrži. Osnovna funkcija procesora je izvođenje instrukcija, pa se u literaturi upravo brzina izvođenja instrukcija često koristi kao parametar njegovih performanci, a iskazuje se u milijunima instrukcija koje je moguće izvesti u sekundi (MIPS, engl. *million instructions per second*). MIPS zapravo predstavlja teorijsku vršnu brzinu koju procesor može ostvariti, a vrijednost objedinjuje sve njegove relevantne parametre (broj priključnica, broj jezgri, frekvenciju i broj instrukcija koje je moguće izvesti u jednom periodu). Ipak, zbog različitih arhitektura procesora, a često i pri usporedbi sličnih arhitektura, usporedbe procesora na temelju ovog parametra rijetko daju precizan rezultat, pa se za usporedbu njihovih performanci koriste i alati za njihovo vrednovanje (engl. *benchmarking tools*). Za pokretanje usluga na nižim slojevima arhitekture Interneta stvari nisu potrebne tako detaljne analize performanci procesora, ali je važno da se može utvrditi mogućnost pokretanja

određene usluge na nekom računalnom čvoru. Kako usluge obično imaju zadanu specifikaciju minimalnog broja jezgri koje procesor treba imati i minimalne frekvencije njegova rada da bi ih računalni čvor mogao pokrenuti, upravo se na temelju ta dva parametra takva informacija može i omogućiti. Dodatno se može razmatrati i postotak zauzetosti procesora ako se želi postići ravnomjeran raspored opterećenja među čvorovima. Tako bi se u slučajevima kada više čvorova može pokrenuti određenu uslugu na temelju zauzetosti njihova procesora odabrao onaj koji je trenutno najmanje opterećen, čime bi se u izvedbenoj okolini postigla ravnoteža opterećenja.

Radna memorija (RAM)

Radna memorija računalnih uređaja obično se u tehničkim specifikacijama opisuje s količinom okteta (engl. *byte*) koje nudi i frekvencijom (MHz ili GHz) kojom se podaci mogu čitati, odnosno zapisivati u nju. U postojećim modelima računarstva u magli, frekvencija radne memorije je najčešće zanemaren parametar budući da se u specifikacijama usluga obično navodi samo minimalna količina radne memorije koju čvor treba imati za njihovo pokretanje. Zato je za opis konteksta uređaja upravo količina radne memorije obvezan parametar na temelju kojeg se može utvrditi mogućnost određenog čvora za pokretanje konkretne usluge. Frekvencija rada radne memorije može se razmatrati kao dodatni parametar na kojem bi se temeljio konačan odabir u slučajevima kada više čvorova zadovolji sve ostale resursne preduvjete za pokretanje određene usluge.

Memorija za pohranu

Memorija za pohranu podrazumijeva količinu dostupne memorije u oktetima na tvrdom disku (engl. *hard drive*), SSD disku (engl. *solid state disk*), ili nekom drugom mediju za pohranu podataka ako se radi o uređajima s ograničenim resursima (npr. memorijska kartica). Brzina zapisivanja, odnosno čitanja podataka puno je veća kada uređaj sadrži SSD disk, ali je i u ovom slučaju količina dostupne memorije preduvjet za izvođenje određenih usluga, dok sama brzina rada utječe onda na razinu kvalitete usluge koju može pružiti određeni računalni čvor. Iako ovaj parametar ima manju važnost za performance izvođenja usluga od procesora i RAM memorije, pojedine usluge zahtijevaju određenu količinu dostupne memorije za izvođenje (npr. ako koriste vlastitu bazu podataka), pa je ovaj parametar također relevantan za opis računalnih uređaja koji sudjeluju u izvođenju sloja računarstva u magli.

Lokacija

Ovisno o implementacijskom modelu u specifičnom scenariju, lokacija uređaja može biti opisana na različite načine. Ipak, u okviru računarstva u magli najrelevantnija informacija o trenutnoj lokaciji određenog uređaja dobiva se iz podataka o njegovoj GPS lokaciji, te njegovoj mrežnoj lokaciji. Kako uređaji u dinamičkoj okolini računarstva u magli mogu biti mobilni i fiksni, tako se i lokacija takvih uređaja može mijenjati pa zato njihova adresa može biti dinamička

ili statična. Stoga je potrebno u svakom trenutku omogućiti sustavu informaciju o lokacijama svih dostupnih čvorova u njegovoj raspodijeljenoj okolini, neovisno o lokacijskim atributima koji se uzimaju u obzir, kako bi se u sustavu efikasno mogli raspoređivati zahtjevi i usluge.

Napajanje i mobilnost

Uređaji mogu imati stalan izvor napajanja (npr. ako imaju direktan priključak na električnu mrežu) ili mogu imati ograničenu količinu dostupne energije za rad (npr. ako su napajani baterijom). Baterijom napajani uređaji također se mogu podijeliti na dvije skupine: one napajane punjivim baterijama i one koji imaju jednokratne baterije ograničenog kapaciteta. Način napajanja uređaja u velikoj mjeri utječe na njegovu mobilnost, budući da su u najvećem broju slučajeva mobilni uređaji napajani baterijama, a fiksni uređaji imaju stalan izvor energije. Zato je pri raspoređivanju određenih usluga važno u obzir uzeti potrošnju energije pri njihovu izvođenju kako bi se u slučaju uređaja napajanih baterijom moglo izračunati vrijeme kroz koje bi one mogle biti dostupne. Također, zbog mobilnosti uređaja u sustavu je potrebno osigurati efikasnu podršku za ulaske i izlaske uređaja iz povezane raspodijeljene okoline, a dodatno je moguće i fiksnim uređajima dati prioritet budući da je u njihovom slučaju očekivana veća i stabilnija dostupnost.

Komunikacija

Komunikacija između uređaja u kontekstu Interneta stvari obuhvaća niz žičnih i bežičnih komunikacijskih protokola. Budući da je za ostvarivanje međusobne komunikacije uređaja potrebno da oni posjeduju komunikacijske module istog protokola, važno je parametarski omogućiti provjeru zadovoljavanja takvog uvjeta. S obzirom na velik broj postojećih komunikacijskih protokola, oblikovanje ovog parametra može biti izvedeno na općenitije načine. U literaturi se tako najčešće u obzir uzima samo provjera mogućnosti čvora da ostvari komunikaciju IP protokolom te kvaliteta i kapacitet veze kojom je čvor povezan na mrežu. Ovakvo bi modeliranje u osnovi moglo biti dostatno i u slučajevima primjene u kontekstu računarstva u magli, budući da svi uređaji koji ostvaruju ovaj dodatni sloj obrade moraju imati mogućnost komunikacije s računalnim oblakom koja se ostvaruje upravo internetskom mrežom. Tek naknadno bi se onda mogla provjeravati i mogućnost komunikacije drugim specifičnim komunikacijskim protokolima, kako bi se omogućilo izvođenje usluga koje zahtijevaju takvu opciju zbog ostvarivanja komunikacije sa senzorskim uređajima koji često nemaju mogućnost komunikacije IP protokolom.

Za većinu navedenih parametara potrebno je naglasiti da su njihove vrijednosti dinamične, odnosno da ovise i o ostalim procesima koji se izvode na uređaju. Zato je potrebno osigurati mogućnost dohvaćanja informacije o trenutnom stanju slobodnih resursa uređaja u svakom trenutku, kako bi se omogućilo efikasno upravljanje resursima za izvođenje usluga u raspodijelje-

noj i heterogenoj okolini kakvu podrazumijeva koncept računarstva u magli. Također, navedeni se parametri mogu modelirati na različite načine ovisno o specifičnom scenariju primjene, pa se može zaključiti da je njima sveobuhvatno određen kontekst uređaja, ali da će izbor korištenih parametara i način na koji će se modelirati njihov utjecaj u specifičnom slučaju uporabe ovisiti o namjeni razvijanog rješenja i ideji primjenjivanog algoritma.

3.2 Kontekst usluge

Kontekst usluga Interneta stvari i parametri koji ga određuju također ovise o namjeni cjelokupnog rješenja koje se želi implementirati. Zato je jednoznačno modeliranje takvog konteksta i njegova sveobuhvatna parametrizacija gotovo neizvediva na horizontalnom nivou koncepta Interneta stvari, pa je važno naglasiti da je u ovom radu postupak definicije konteksta usluga i parametara koji ga određuju proveden s fokusom na primjenu u konceptu računarstva u magli.

Za ostvarivanje takvog cilja u raspodijeljenoj okolini kakvu podrazumijeva ovaj koncept, potrebno je najprije definirati sam pojam IoT usluge, a zatim i izdvojiti te specificirati one parametre koji su važni faktori u njihovu izvođenju i raspoređivanju, odnosno kojima bi se kontekst takvih usluga mogao kvalitativno opisati. Pojam IoT usluge nema jedinstvenu definiciju, pa je u znanstvenoj literaturi takav termin poprimao različita značenja ovisno o kontekstu i promatranoj razini apstrakcije. Problem nejedinstvene definicije usluga Interneta stvari još 2012. godine ističu autori u [58], pa na temelju pregledane znanstvene literature navode sljedeću definiciju:

IoT usluga je transakcija između dvije stranke, pružatelja usluge i njenog potrošača. Ona podrazumijeva pružanje predodređene funkcije koja omogućuje interakciju s fizičkim svijetom tako da se pruža mjerenje stanja vanjskih entiteta ili zadavanje akcija koje uzrokuju promjene na vanjskim entitetima.

Osim navedene definicije u istom su radu napravljene i dvije klasifikacije IoT usluga na temelju različitih kriterija: klasifikacija temeljena na odnosu sa samim fizičkim entitetom i klasifikacija temeljena na životnom ciklusu usluge. Prva klasifikacija odnosi se na gradaciju apstrakcije od fizičkog entiteta do njegova modela na višem nivou koji se nudi korisnicima, i podrazumijeva sljedeće četiri vrste usluga:

- **primitivne usluge (engl. *low level service)*** - opcije koje neki uređaj može ponuditi (npr. vrste senzora ili aktuatora koje sadrži);
- **usluge resursa (engl. *resource service*)** - podaci koje uređaj može proizvesti na temelju vlastitih senzora ili akcija koje može izvršiti vlastitim aktuatorima;
- **usluge entiteta (engl. *entity service*)** - cjelovite usluge koje neki entitet može ponuditi na temelju kombiniranja vlastitih primitivnih usluga;
- **integrirane usluge (engl. *integrated service*)** - usluge koje međusobno integriraju usluge entiteta te ih povezuju s ostalim uslugama koje ne moraju biti direktno vezane za Internet

stvari.

Druga klasifikacija odnosi se na trenutni status usluge i u tom su kontekstu one podijeljene na sljedeće tri skupine:

- **rasporedive (engl. *deployable*)** - usluge koje su implementirane i nalaze se u repozitoriju usluga, ali nisu postavljene na računalni čvor koji bi ih mogao izvoditi;
- **raspoređene (engl. *deployed*)** - usluge koje se nalaze na računalnom čvoru koji bi ih mogao izvoditi, ali ne mogu biti pokrenute zbog različitih razloga (npr. plaćanje pretplate za korištenje);
- **izvedive (engl. *operational*)** - usluge koje su raspoređene i spremne za pokretanje.

Navedena definicija i opisane klasifikacije opisuju pojam IoT usluge na visokoj razini apstrakcije. Zato je njima obuhvaćen cjelokupan spektar konkretnih primjena ovog koncepta i stvorena je jasna ideja o tome što takve usluge trebaju ponuditi vlastitim potrošačima. Na istoj su razini apstrakcije usluge Interneta stvari promatrane i u [59] gdje je istraživan problem poslovnih modela za takve usluge pa ih se promatra u kontekstu cjelokupnog proizvoda. Autori u ovom radu ne navode direktnu definiciju IoT usluga, ali zaključuju da je najvažniji element kojeg one moraju ponuditi korisnicima, mogućnost izvođenja analitike nad podacima. Drugi istaknuti element, koji uvelike doprinosi mogućnosti eksploatacije ovakvih usluga, je otvoreni ekosustav kojim bi se omogućilo povezivanje i međusobno integriranje različitih usluga, što bi potrošačima stvorilo dodatnu vrijednost.

Ostali pregledani radovi koji istražuju problem definicije IoT usluga također ih definiraju s visokom razinom općenitosti, kako bi se obuhvatio što veći opseg njihovih različitih primjena [60]. Klasifikacije IoT usluga su također najvećim dijelom podrazumijevale općenite podjele u kojima ih se najčešće razdvajalo prema pripadajućem sloju prethodno opisane arhitekture Interneta stvari, uz nekoliko iznimki kada su napravljene specifične podjele temeljene na kriterijima koji bi se mogli iskoristiti kao smjernice za definiciju parametara konteksta usluge.

Tako je primjer specifične klasifikacije usluga Interneta stvari opisan u [61], gdje je cilj bio kategorizirati tipove IoT usluga kako bi se razvojni inženjeri mogli koncentrirati na same aplikacije, a ne na dizajn usluga i arhitektura na kojima bi se one temeljile. Navedeni su sljedeći tipovi usluga:

- **usluge povezane s utvrđivanjem identiteta (engl. *identity-related services*)** - obično sadrže dvije komponente: 1) sve stvari moraju imati jedinstveni identifikator (npr. RFID čip), 2) čitač koji prepoznaje spomenute identifikatore i može dohvatiti profil prepoznatog uređaja s poslužitelja;
- **usluge prikupljanja informacija (engl. *information aggregation services*)** - prikupljanje podataka s različitih senzora, te njihova obrada i prosljeđivanje internetom do aplikacija na poslužitelju;
- **usluge kolaborativne suradnje (engl. *collaborative-aware services*)** - usluge koje na te-

melju prikupljenih podataka sa senzora donose odluke i poduzimaju odgovarajuće akcije. Kako ove usluge trebaju prikupiti podatke i proizvesti odgovarajuće rezultate na temelju prikupljenih podataka, važno im je omogućiti pouzdanu i brzu komunikacijsku infrastrukturu, te terminale s dovoljno resursa za provođenje potrebne obrade ili dostupnost drugih čvorova koji bi mogli izvesti takvu obradu;

- **sveprisutne usluge (engl. *ubiquitous services*)** - slična kategorija prethodnoj skupini, ali se podrazumijeva konstantna prisutnost koja koristi svima i kojom bi se ostvarila ideja međusobnog povezivanja svih dostupnih povezivih resursa preko internetske mreže u jedinstvenu aplikaciju.

Usluge koje pripadaju prvoj skupini najvećim dijelom podrazumijevaju korištenje RFID tehnologije. Osim samih usluga koje se temelje na identifikaciji stvari, očekivano je da će i sve ostale skupine usluga morati imati ugrađenu neku vrstu ovakve usluge u sebi kako bi mogle identificirati povezane uređaje koji sudjeluju u konkretnim aplikacijama. Druga skupina odnosi se na prikupljanje podataka sa senzorskih mreža (engl. *Wireless Sensor Network*) i njihovo prosljeđivanje do aplikacija za obradu. Takav scenarij uglavnom podrazumijeva aplikacije koje omogućuju korisnicima praćenje određenih vanjskih fenomena i parametara izvođenja poslovnih procesa (engl. *monitoring*). Treća skupina obuhvaća usluge koje na temelju prikupljenih podataka donose određene odluke po zadanim algoritmima pa je u njihovom slučaju naglašena potreba za sigurnim i brzim prijenosom podataka do čvorova za njihovu obradu. Konačni cilj usluga iz sve tri navedene skupine je četvrta skupina koja podrazumijeva sveprisutnu povezanost između korisnika i stvari u njihovoj okolini. Sve četiri navedene skupine usluga zapravo predstavljaju stupnjevanje IoT usluga prema njihovoj kompleksnosti. Takav bi se kriterij djelomično mogao primijeniti i u parametrizaciji konteksta usluga, ali bi bez drugih parametara bio nedovoljno specifičan za određivanje optimalnog rasporeda izvođenja usluge u okolini računarstva u magli.

Jedan od razloga općenitosti prethodno opisanih klasifikacija je razdoblje u kojem su one nastale, u kojem se koncept Interneta stvari još uvijek ubrzano razvijao i nije postojao dovoljan broj konkretnih primjera kojima bi se potvrdila njihova ispravnost. U novijim znanstvenim radovima u obzir se uzimao širi spektar implementiranih IoT usluga, pa su se za takve klasifikacije lakše mogli prezentirati rezultati kojima bi se potvrdila efektivnost razvrstavanja postojećih usluga u pripadajuće grupe.

Takav primjer klasifikacije opisan je u [62], gdje je predstavljen sustav za klasifikaciju i grupiranje IoT usluga prema kriteriju sličnosti zasnovanom na jasno definiranim parametrima i unaprijed zadanim vrijednostima koje isti mogu i poprimiti. Za ostvarivanje takvog sustava autori su klasificirali sljedeće četiri razine izvođenja IoT usluge s pripadajućim kontekstnim parametrima kako bi ih se moglo najprije kontekstno opisati, a zatim i usporediti, te svrstati u grupe prema kriteriju sličnosti:

- **senzorska razina (engl. *sensing*)** - opis karakteristika krajnjih uređaja;
  - napajanje (*"power"*): *"self power", "AC/DC", "self recharge", "auto recharge"*;
  - IP: *"IP","non IP", "IP and non IP"*;
  - način rada (*"operation"*): *"event", "frequency", "event2frequency", "timer"*;
- **upravljanje podacima (engl. *data management*)** - opis jednostavnije obrade podataka na krajnjim uređajima ili poslužiteljima u svrhu njihove pripreme za izvođenje daljnje analitike nad njima;
  - priprema za obradu (*"preprocessing"*): *"preprocessing client", "preprocessing server", "IoT gateway"*;
  - pohrana podataka (*"data store"*): *"volatile", "non volatile"*;
  - prijenos podataka (*"transmission"*): *"distributed", "centralised"*;
  - stvaranje povjerljivosti (*"trust formation"*): *"single", "multiple"*;
- **obrada podataka (engl. *processing*)** - opis analitičkih modela i algoritama za donošenje odluka utemeljenih na obradi podataka;
  - računalni modeli za analizu podatka (*"computational models for data analysis"*): *"data parallel", "task parallel", "peak detection", "min", "max", "mean", "variance", "PSD analysis", "wavelet analysis", "correlation", "granger casuality"*;
  - modeli za obradu podataka i donošenje odluka (*"models for data manipulation or decision making"*): *"belief theory", "bayesian systems", "fuzzy logic", "weighted sum", "regression analysis", "superposition", "decision tree analysis", "threshold-based anomaly detection", "de-noising/artifact removal", "none"*;
  - QoS: *"real time", "capacity"*;
- **izvedbena razina (engl. *execution*)** - opis načina izvođenja cjelovite IoT usluge.
  - način izvođenja usluge: *"report", "alert", "action", "etc"*.

Ovakva klasifikacija daje konkretan primjer parametrizacije konteksta usluge Interneta stvari. Njihov kontekst se navedenim parametrima može precizno opisati, ali je konačan izbor relevantnih parametara u specifičnom scenariju ovisan o namjeni konkretne usluge. U okviru ove disertacije potrebno je napraviti specifikaciju konteksta IoT usluga za razvoj algoritma njihova raspoređivanja u okolini računarstva u magli. Zato je u obzir potrebno uzeti prvenstveno one parametre koji mogu postaviti određena ograničenja na lokaciju izvođenja komponenti usluge u raspodijeljenoj izvedbenoj okolini, te utjecati na kvalitetu njene isporuke. Stoga je u nastavku definirana parametrizacija konteksta usluge koja će se koristiti u nastavku ovog istraživanja za razvoj algoritma njihova raspoređivanja u računarstvu u magli.

3.2.1 Parametri konteksta usluge

Spremanje podataka (perzistentnost)

Podaci IoT usluga najčešće se spremaju u baze podataka ako ih je potrebno perzistentno

pohraniti, dok se za ne-perzistentnu pohranu obično koriste nestalne (engl. *volatile*) strukture interne memorije. Migracija usluga u kojima se podaci pohranjuju perzistentno podrazumijeva kompleksan i dugotrajan proces ako je s uslugom potrebno migrirati i same podatke (ako se ne nalazi u računalnom oblaku). Zato je u opisu konteksta usluge važno osigurati informaciju o perzistentnosti pohrane podataka kako bi se mogla procijeniti učinkovitost migracije njenih komponenti u konkretnom slučaju. Frekvencija migracija u scenarijima usluga koje ne zahtijevaju perzistentnu pohranu podataka ili usluga koje perzistentnu pohranu izvode na udaljenim bazama podataka, nije uvjetovana ovim parametrom budući da u takvim slučajevima nije potrebno migrirati pohranjene podatke. Suprotno tome postoje scenariji kada je najčešće zbog sigurnosnog rizika ili potrebe za otpornosti usluge na ispade internetske mreže potrebno perzistentnu pohranu izvoditi na čvoru gdje se izvodi i sama usluga. U takvim je scenarijima potrebno procijeniti efikasnost izvođenja migracija konkretne usluge, odnosno ograničiti njihovu frekvenciju na razinu kojom bi se osigurala isplativost njihova izvođenja.

Korisnički doseg usluge

IoT usluge su često usmjerene na različit opseg korisnika za koje ispunjavaju vlastitu svrhu pa je za implementaciju efikasnog raspoređivanja važno uzeti u obzir korisnički doseg njihove namjene. Doseg tako može biti individualiziran za svakog korisnika, usmjeren na manje skupine korisnika u zajedničkim okolinama (pametni ured, pametna zgrada, i sl.), ili može biti općenit ako se ista usluga nudi svim korisnicima bez prilagođenih profila. Stoga ovaj parametar određuje odnos usluge i njenih korisnika, zbog čega je važan najprije za određivanje prikladne arhitekture izvođenja same usluge, ali i za specifikaciju dinamike migracija njenih komponenti (ovisno o tome radi li se u konkretnom scenariju o mobilnim korisnicima ili ne).

Komunikacija

Način komunikacije između entiteta u raspodijeljenim okolinama Interneta stvari je uz podatke najvažnija komponenta koja je preduvjet za izvedbu svih usluga ovog koncepta. U smislu kontekstnog parametra kojim bi se specificirale IoT usluge u okolini računarstva u magli, svojstvo komunikacije podrazumijevalo bi najprije zahtijevanu kvalitetu internetske veze u određenom scenariju, a zatim i potrebu za drugim komunikacijskim protokolima potrebnim za njeno izvođenje. Iako se internetska povezivost danas gotovo podrazumijeva u izvedbi svih IoT usluga, postoji i manji broj usluga kojima za rad nije konstantno potrebna internetska veza. Zato su informacije o tome treba li određena IoT usluga konstantno dostupnu i stabilnu internetsku mrežu, koja je zahtijevana kvaliteta veze za njeno efikasno izvođenje, te može li određena usluga ostvariti vlastitu funkciju i bez internetske mreže, važne kako bi se ista mogla rasporediti na čvor koji bi potrebne parametre mogao i ispuniti. Osim komunikacije internetskom mrežom, u scenarijima Interneta stvari komunikacija se često ostvaruje i drugim komunikacijskim protokolima prilagođenim za resursno ograničene krajnje uređaje. Zato je u okviru parametra komu-

nikacije potrebno osigurati i informaciju o drugim komunikacijskim protokolima potrebnim za izvođenje usluge kako bi se takav uvjet također mogao uzeti u obzir pri njenom raspoređivanju.

Brzina odziva (kašnjenje)

Moderne primjene Interneta stvari često podrazumijevaju gotovo stvarnovremensku obradu podataka i jednako brze reakcije sustava na rezultate takvih obrada, pa je u takvim scenarijima važno u što većoj mjeri eliminirati moguće uzroke nepotrebnog kašnjenja. Ukupno kašnjenje između upućenog zahtjeva i dobivenog rezultata u raspodijeljenoj okolini, uz samo trajanje obrade, u velikoj mjeri ovisi i o brzini komunikacije. Zato se u scenarijima koji zahtijevaju vrlo visoke performance bolji rezultati i dalje mogu postići ako je usluga bliže entitetu koji šalje sam zahtjev, iako je brzina mrežne komunikacije danas na visokoj razini i omogućuje gotovo trenutan odziv. Upravo je to jedno od glavnih načela koje je i potaklo nastanak koncepta računarstva u magli, pa bi se jasan i mjerljiv rezultat efikasnog raspoređivanja usluga trebao vidjeti baš na usporedbi vrijednosti kašnjenja u sustavima gdje je usluga pokrenuta na udaljenom računalnom oblaku i u lokalnoj mrežnoj okolini. Kako bi se pri raspoređivanju mogli odrediti čvorovi računarstva u magli koji bi zadovoljili ciljane performance određenog scenarija, odnosno kako bi se pronašao onaj čvor koji bi ponudio najveću efikasnost u isporuci same usluge, potrebno je u njenoj specifikaciji omogućiti definiciju donje granicu brzine odziva koja se u konkretnom scenariju zahtjeva.

Sigurnost

Sigurnosni rizik od napada u scenarijima Interneta stvari je veći nego u slučajevima napada na klasična web-sjedišta, budući da opseg ugroze osim podatkovnog aspekta uključuje i fizički svijet. Upravo zato se sigurnost često ističe kao velika prepreka široj primjeni rješenja koje nudi koncept Interneta stvari što u velikoj mjeri usporava brzinu rasta cjelokupnog koncepta. Svaka IoT usluga podrazumijeva različitu razinu sigurnosnog rizika, pa je u opisu njena konteksta važno uzeti u obzir i tu činjenicu koja se može procijeniti na temelju sigurnosne osjetljivosti podataka u njenoj izvedbi, ili na temelju vrste krajnjih uređaja koji sudjeluju u konkretnom scenariju (senzori ili aktuatori). Sigurnost je parametar koji je u svakom slučaju potrebno uzeti u obzir od početka dizajna arhitekture svakog rješenja vezanog uz Internet stvari, a primjenom koncepta računarstva u magli dodatno bi se mogla podignuti razina zaštite od napada putem javne mreže ako bi se usluge u većoj mjeri izvodile u lokalnim korisničkim okolinama. Mehanizmi kojima se ostvaruje sigurnost u postojećim web-aplikacijama koje koriste model klijent-poslužitelj primjenjivi su u velikoj mjeri i na scenarije Interneta stvari. Njihova izvedba djelomično se razlikuje budući da model IoT usluga uključuje korisnike s jedne i uređaje s druge strane pri čemu i jedna i druga strana može inicirati komunikaciju. Ipak, iz perspektive čvora u kojem se izvodi obrada obje se strane mogu promatrati kao klijenti pa bi upravo takvi čvorovi bili logičan izbor za provođenje autentifikacije i autorizacije svih međusobno poveza-

nih entiteta. Dodatno, važno je da svi uređaji koji zajedno čine takav raspodijeljeni sustav pri samom ulazu u takvu okolinu izvrše potrebne procedure kojima bi ih se pravilno autentificiralo, i na temelju toga dodijelilo određene strukture pomoću kojih bi se onda provodila sigurna komunikacija, zaštićena od napada treće strane.

Ograničenja kvalitete usluge

Iako svi prethodno opisani parametri određuju kvalitetu isporuke određene IoT usluge, u ovoj se podjeli ova kategorija odnosi na parametre koji se upotrebljavaju u specifikacijama ili vrednovanjima kvalitete u isporuci klasičnih web-usluga. Tako je u nekim slučajevima važno u opisu IoT usluge naznačiti potrebne okvire za njenu propusnost, pouzdanost, preciznost, energetsku učinkovitost, vijek trajanja i slično, kako bi se onda prema tome u njenom raspoređivanju moglo pronaći što efikasnije mjesto za njeno izvođenje. Odabir relevantnih faktora utjecaja na određenu IoT uslugu između postojećih parametara za vrednovanje kvalitete web-usluga ovisit će o njenom specifičnom scenariju primjene i okoline koju takav scenarij podrazumijeva. Osim parametara za vrednovanje kvalitete u isporuci web-usluga, ovim se parametrom može također naznačiti i očekivana mjera u kojoj je potrebno zadovoljiti ostale parametre kojim se specificira kontekst IoT usluge. Neke usluge imaju stroge kriterije koji moraju biti zadovoljeni za njihovo izvođenje, dok druge dozvoljavaju određena odstupanja od zadanih okvira pa bi se ovaj dodatni parametar mogao koristiti kao mehanizam kojim bi se takav kriterij mogao i zadati. Na temelju zadane vrijednosti sustav bi mogao prepoznati kritičnost ispunjavanja zadanih specifikacija i odrediti raspored kojim bi zadani kriteriji bili u što većoj mjeri zadovoljeni, ili u slučaju nemogućnosti ispunjavanja kritičnih uvjeta obavijestiti administratora sustava o neizvedivosti zadane konfiguracije.

Čuvanje stanja (engl. statefulness)

Posljednji parametar koji ima važnu ulogu u dizajnu i implementaciji raspoređivanja IoT usluga u raspodijeljenoj okolini je njeno svojstvo čuvanja stanja. Usluge koje ne čuvaju stanje (engl. *stateless*) ne moraju imati nikakve zapise o prethodno provedenim operacijama, odnosno one vlastitu namjenu mogu uspješno izvršiti svaki put na isti način, bez potrebe za kontekstnim informacijama o prethodno izvedenim obradama. Usluge koje čuvaju stanje (engl. *statefull*) trebaju za vlastitu funkcionalnost kontekstne informacije o prethodno izvedenim operacijama, budući da o njima često ovisi krajnji rezultat takve obrade. Takve se informacije obično čuvaju u varijablama ili memorijskim strukturama pohranjenim u kontekstnom prostoru usluge na računalnom uređaju koji je pokreće, zbog čega je migracija takvih usluga kompleksan proces koji obično podrazumijeva i određeni period u kojem usluga nije dostupna (engl. *downtime*) [27]. Zato je u opisu konteksta usluge važno uvrstiti parametar čuvanja stanja kako bi se pri procjeni efikasnosti izvođenja migracije za scenarije u kojima se čuva stanje u obzir mogao uzeti i određeni period ispada usluge.

3.3 Kontekst korisnika

Posljednji izdvojeni kontekst koji je važno parametrizirati kako bi se omogućilo kontekstno-svjesno raspoređivanje IoT usluga odnosi se na njihove korisnike. Krajnji korisnik usluga omogućenih Internetom bio je čovjek ili određena grupa ljudi, dok u okviru Interneta stvari postoje scenariji kod kojih su korisnici autonomni računalni uređaji. Zato je u za određenu IoT uslugu potrebno najprije odrediti kakvim korisnicima je ona namijenjena, kako bi se utvrdili relevantni faktori koji bi utjecali na vrednovanje kvalitete u njenoj isporuci i raspored njena izvođenja.

Osnovni razlog zbog kojeg se inicijalno i pojavio koncept računarstva u magli je kašnjenje uzrokovano prevelikom udaljenošću između korisničkih uređaja i dislociranih računalnih oblaka koji izvode obradu i odgovaraju na zahtjeve korisnika. Zato je primarni cilj ovog koncepta približavanje izvođenja obrade samim korisnicima, odnosno izvođenje cjelovite ili djelomične obrade na računalnim uređajima u lokalnim korisničkim okolinama. Kako bi se takav cilj mogao ispuniti, potrebno je u svakom trenutku omogućiti sustavu informaciju o trenutnoj lokaciji korisnika te o stanju njihove lokalne okoline, što je zbog velikog udjela mobilnih korisnika velik izazov. U pregledanoj literaturi koja obrađuje kontekst korisnika IoT usluga, upravo se mobilnost ističe kao glavno obilježje korisnika kojima su namijenjene usluge izvođene djelomično ili u cijelosti u sloju računarstva u magli. Tako se u [63] navodi kako je računarstvo u magli zapravo usmjereno rješavanju problema nedovoljne svijesti usluga izvođenih u računalnom oblaku o stanju okoline na lokaciji njenih korisnika. Suprotno tome, usluge koje bi se izvodile na nižim slojevima računarstva u magli imale bi bolji uvid u stanje lokalne okoline, ali bi zato njihova dostupnost bila ograničena na manji opseg korisnika, te bi se dodatno podigla i kompleksnost podrške za mobilnost korisnika. Stoga je za iskorištavanje potencijala računarstva u magli potrebno osigurati mogućnost efikasne migracije usluga i korisničkih profila između različitih okolina kako bi se ponudila dostatna podrška za mobilnost korisnika i povećala dostupnost usluge.

Druga važna stavka korisničkog konteksta odnosi se na pružanje informacija sustavu o korisniku i njegovim aktivnostima koje mogu biti relevantne za određivanje efikasnog rasporeda izvođenja usluge. Korisnički kontekst je izrazito dinamičan zbog već naglašene mobilnosti korisnika [64], zbog čega je potrebno stalno pratiti parametre koji ga u specifičnom slučaju određuju i prilagođavati ponašanje aplikacije njegovom trenutnom stanju. Parametri koji određuju korisnički profil u specifičnom slučaju su različiti. U [64] se tako navodi da korisnički kontekst definiraju karakteristike specifičnog korisnika, povijest korištenja same usluge, i vjerojatnost prestanka korištenja iste. U [65] se ističe da se za opis korisničkog konteksta mogu iskoristiti podaci iz njihovih pametnih telefona, budući da su oni danas opremljeni velikim brojem senzora kojim se može doći do informacija o korisničkoj lokaciji, stanju korisničke okoline, a dodatno i o navikama te preferencijama korisnika. Tako se može zaključiti da se korisnički

kontekst može odnositi na širok spektar različitih informacija koje opisuju entitet koji ostvaruje interakciju s uslugom u specifičnom slučaju primjene [42]. Jedna od prednosti koje se mogu iskoristiti za prikupljanje relevantnih informacija o korisničkom kontekstu u okviru IoT usluga je što takvi scenariji obično podrazumijevaju prisutnost senzora koji takve informacije svakako prikupljaju u okviru aplikacije koje se primjenjuju. Zato se određeni podaci o korisničkom kontekstu koji su potrebni za raspoređivanje usluga mogu dobiti direktno iz podataka koji se koriste za implementaciju logike takvih aplikacija.

Sve navedene informacije o korisničkom kontekstu oblikuju korisnički profil o kojem ovisi način isporuke određene IoT usluge. Osim lokacije korisnika i parametara koji opisuju korisničku okolinu, važno je spomenuti i posljednji segment korisničkog konteksta - korisničke preferencije kojima sam korisnik može prilagoditi način isporuke usluge prema vlastitim željama [66]. Podaci o korisničkim preferencijama tako mogu podrazumijevati informacije o ciljanoj kvaliteti usluge ili parametre kojima se može utjecati na način isporuke IoT usluge u specifičnom slučaju primjene.

Navedene informacije o kontekstu korisnika potrebno je uzeti u obzir kako bi se mogao odrediti raspored izvođenja usluge koji će pružiti veću kvalitetu njene isporuke za ciljane korisnike. Kako bi se sustavu omogućile informacije o stanju opisanih segmenata konteksta korisnika, u nastavku su definirana tri parametra i način na koji će se oni opisivati kako bi se mogli uzeti u obzir pri raspoređivanju IoT usluga u okolini računarstva u magli. Dodatno treba naglasiti i da su informacije o kontekstu korisnika privatni podaci, pa je za njihovo dohvaćanje uvijek potrebno imati privolu korisnika. Zato je u implementaciji usluge parametre navedene u nastavku potrebno ili eksplicitno zatražiti od korisnika, ili ga obavijestiti o informacijama koje će sustav prikupljati u svrhu izvođenja raspoređivanja komponenata usluge kako je definirano u GDPR-u.

3.3.1 Parametri konteksta korisnika

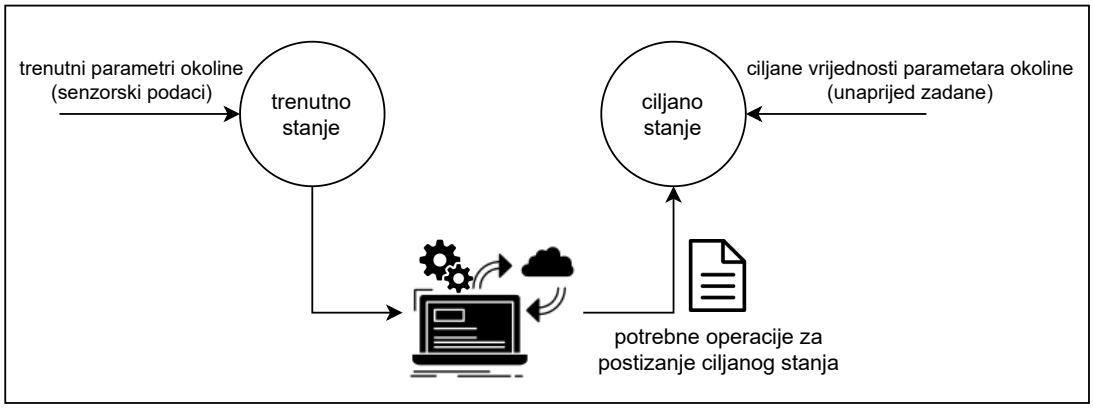
Lokacija korisnika

Osnovni parametar koji je preduvjet za ostvarivanje raspoređivanja usluga u okolini računarstva u magli je upravo lokacija korisnika. Kako je prethodno opisano, mobilnost je jedno od osnovnih obilježja najvećeg broja korisnika IoT usluga pa je za ostvarivanje efikasnog raspoređivanja usluga u lokalnim okolinama važno omogućiti sustavu informaciju o trenutnoj lokaciji korisnika u svakom trenutku. Na temelju te informacije sustav treba prepoznati mogućnost, a zatim i efektivnost raspoređivanja određene usluge u ciljanoj korisničkoj okolini, te na temelju toga izvršiti daljnje potrebne procedure. Lokacija korisnika se u sustavu može promatrati na različite načine, a najčešće se radi o mrežnoj lokaciji ili lokaciji određenoj GPS koordinatama. Ipak, kako se primjenom računarstva u magli želi primarno umanjiti utjecaj mrežnog kašnjenja, modeliranje lokacija uključenih entiteta korištenjem njihovih javnih i privatnih IP adresa omo-

gućuje dostatnu informaciju o njihovoj lokaciji za izvođenje dinamičkog raspoređivanja IoT usluga u raspodijeljenoj izvedbenoj okolini.

Stanje korisničke okoline

Stanje korisničke okoline podrazumijeva različite pripadajuće faktore ovisne o specifičnom slučaju primjene koji se u okviru Interneta stvari najčešće mjere senzorima postavljenim u ciljanoj korisničkoj okolini. Svi elementi koji određuju korisničku okolinu i koji su potrebni kako bi određena usluga mogla efektivno izvršiti vlastitu svrhu trebaju se kvalitativno izmjeriti, a zatim i opisati sustavu u specifičnom kontekstu usluge u okviru koje se primjenjuju. Namjena većine IoT usluga zasniva se upravo na takvim podacima proizvedenim u korisničkoj okolini te na procedurama kojim bi se utjecalo na vrijednosti parametara u njima, zbog čega je precizan način opisa korisničke okoline važan preduvjet za ostvarivanje visoke kvalitete u njihovoj isporuci. Osim parametara okoline koji se mogu mjeriti senzorskim uređajima, važan preduvjet za izvođenje efikasnog raspoređivanja usluga je i specifikacija konteksta uređaja u ciljanoj korisničkoj okolini. Preduvjet izvođenja IoT usluga u ciljanom korisničkom okruženju često su specifični računalni uređaji koji trebaju ispuniti zadane kriterije za njihovo izvođenje. Osim kriterija koji se odnose na računalnu moć samih uređaja, u velikom broju scenarija Interneta stvari preduvjet za njihovo izvođenje, odnosno za mogućnost specifikacije prethodno opisanih mjerljivih parametara korisničke okoline su specifični senzori koje određeno ciljano okruženje treba sadržavati. Zato je specifikacija takvih zahtjeva na kontekst uređaja u specifičnim scenarijima izvođenja preduvjet koji primarno mora biti zadovoljen kako bi se ostali parametri korisničke okoline uopće mogli i opisivati.



Slika 3.2: Osnovni scenarij IoT usluga koje djeluju na parametre okoline u korisničkom okruženju.

Poseban aspekt ovog parametra korisničkog konteksta odnosi se na dio IoT usluga kojima je konačni cilj djelovanje na elemente okoline u korisničkom okruženju (slika 3.2). Takve usluge dodatno zahtijevaju unaprijed zadanu specifikaciju željenih vrijednosti parametara korisničke okoline, kako bi sustav na temelju analize podataka o trenutnom stanju vrijednosti istih parametara i izvedivih akcija u promatranoj okolini mogao sugerirati korisniku ili automatski izvesti

potrebne zahvate kako bi se željeno stanje i postiglo.

Preferencije korisnika i dodatna ograničenja

Posljednji parametar koji određuje korisnički kontekst je profil korisnika u kojem su sadržane različite informacije na temelju kojih sustav prilagođava uslugu specifičnom korisniku ili grupi korisnika. U takvim se profilima nalaze različite informacije ovisno o vrsti same usluge, a najčešće su to preference korisnika za prilagođeni rad određene IoT usluge ili dodatna ograničenja koja je potrebno zadovoljiti u specifičnom slučaju primjene. Zadavanje ovog parametra važno je za raspoređivanje usluga budući da sustav dobiva određene smjernice od samog korisnika o specifičnom načinu rada konkretne usluge kojim bi se postigla željena razina kvalitete u njezinoj isporuci. Način oblikovanja korisničkog profila može biti izveden tako da sam korisnik kroz obrasce s unaprijed zadanim parametrima pridružuje istima neke od predefiniranih vrijednosti koje oni mogu poprimiti, ili sustav automatski prema navikama korisnika može zaključiti koje su preferencije korisnika. Sustav tako može pratiti povijest korištenja određene usluge, učestalost korištenja u određenim vremenskim intervalima, i slične ponašajne obrasce korisnika na temelju kojih se formira korisnički profil prema kojem mu se prilagođava isporuka konkretne usluge. Osim oblikovanja individualnih korisničkih profila, u određenim slučajevima može i administrator usluge zadavati određene preference i ograničenja u isporuci usluge, što je posebno primjenjivo na usluge namijenjene grupi korisnika. U takvim slučajevima administrator usluge može oblikovati korisnički profil na temelju raznih parametara o korisničkoj okolini prema kojem bi se u specifičnom slučaju usluga onda i isporučivala.

Poglavlje 4

Parametri kvalitete i kategorizacija IoT usluga

Primjena računarstva u magli može na različite načine povećati efikasnost isporuke IoT usluga, ali se evaluacija kvalitete u njihovoj isporuci obično provodi na temelju karakterističnih parametara kojim je određena kvaliteta usluga isporučenih preko mreže. Stoga je u potpoglavlju 4.1 predstavljena analiza parametara kvalitete usluge koji se koriste za evaluaciju IoT sustava, te su izdvojeni parametri koji će se koristiti za evaluaciju primjene računarstva u magli u okruženju Interneta stvari. U nastavku poglavlja napravljena je analiza IoT usluga koju je potrebno provesti da bi sustav mogao odrediti što efikasniji raspored komponenti usluge kojim bi se postigla što veća kvaliteta isporuke u specifičnom slučaju primjene. Najprije je u potpoglavlju 4.2 opisana kategorizacija IoT usluga, te su određeni pripadajući parametri kvalitete usluge koji u izdvojenim kategorijama u najvećoj mjeri određuju efikasnost njihove isporuke. Zatim je u potpoglavlju 4.3 napravljena analiza utjecaja relevantnih kontekstnih informacija definiranih u poglavlju 3 na raspoređivanje komponenti IoT usluga.

4.1 QoS parametri za evaluaciju primjene računarstva u magli

Isporučena kvaliteta usluge svih informacijsko-komunikacijskih aplikacija ovisi o njihovoj specifičnoj namjeni koja određuje mjerljive i iskustvene parametre prema kojima se može vrednovati njihova efikasnost. Iskustvena kvaliteta (QoE, engl. *Quality of Experience*) podrazumijeva subjektivni dojam krajnjeg korisnika kojeg ITU u kontekstu telekomunikacijskih usluga definira kao opći stupanj prihvatljivosti cjelokupne usluge iz perspektive krajnjeg korisnika [67]. Kvaliteta usluge (QoS, engl. *Quality of Service*) je sličan pojam koji se također odnosi na evaluaciju efikasnosti njene isporuke. ITU specificira QoS kao skup karakteristika usluge koje utječu na

njenu mogućnost da ispuni navedene i podrazumijevane potrebe vlastitih korisnika. Iako su prethodne definicije dane u kontekstu telekomunikacijskih usluga, primjenjive su i u okviru drugih aplikacija koje se korisnicima isporučuju kroz mrežu, što podrazumijeva i IoT usluge koje se izvode u kontinuumu od računalnog oblaka do lokalnih mrežnih okolina. U prethodno spomenutom dokumentu razlikuju se kvantitativni i kvalitativni QoS parametri:

- objektivni (kvantitativni) parametri - mogu se precizno izmjeriti, pa se prema njihovim vrijednostima može izvoditi objektivna evaluacija performanci u isporuci usluga,
- subjektivni (kvalitativni) parametri - korisnikova ocjena efikasnosti prema njegovom subjektivnom dojmu isporuke usluge.

Tako su u znanstveno-istraživačkom kontekstu precizniji pokazatelji razlike između usluga i efikasnosti u njihovoj isporuci kvantitativni parametri, dok se subjektivni parametri koriste kao druga opcija ili dodatna potvrda rezultata dobivenih ispitivanjem mjerljivih elemenata kvalitete usluge. Cilj ovog potpoglavlja je precizno definirati relevantne faktore kvalitete usluge na koje utječe primjena računarstva u magli, podijeliti ih na kvantitativne i kvalitativne, te prema tome odrediti opseg i granice njihovih vrijednosti.

Standardizirana specifikacija parametara kvalitete usluge u domeni Interneta stvari ne postoji, pa su se u postojećoj literaturi uzimali u obzir različiti faktori za procjenu kvalitete usluge ovisno o konkretnom slučaju primjene. Kako scenariji Interneta stvari obuhvaćaju različite komponente i višeslojnu arhitekturu, kvaliteta usluge može se promatrati iz različitih perspektiva ovisno o cilju usluge u konkretnom slučaju primjene. Tako se u postojećim radovima koji opisuju evaluacija kvalitete IoT usluga najčešće mogu pronaći primjeri parametara kojima su se evaluirale njihove performance u najnižem sloju na temelju rada krajnjih uređaja, ili u najvišem sloju na temelju njihove isporuke kroz računalni oblak. U tablici 4.1 je napravljen pregled literature u kojem se navode parametri za evaluaciju kvalitete usluge koji su uzeti u obzir pri vrednovanju IoT usluga u postojećim radovima. Uz svaki rad je stavljena i oznaka sloja cjelokupne arhitekture Interneta stvari kroz koji je u radu bila evaluirana kvaliteta usluge: V - viši sloj (računalni oblak); N - niži sloj (krajnji uređaji); S - cjelokupna arhitektura Interneta stvari.

Iz tablice je vidljivo da je najpotpuniji pregled parametara kvalitete usluge u Internetu stvari napravljen u [74], gdje su navedeni svi faktori koji utječu na jedan od tri elementa u njihovoj isporuci (komunikacija, obrada, i performance uređaja). Zato su upravo oni najprije uzeti u obzir kako bi se analizirala njihova upotreba za evaluaciju primjene računarstva u magli u izvođenju IoT usluga.

Primarni cilj primjene računarstva u magli je smanjivanje kašnjenja u komunikaciji s udaljenim računalnim oblakom i djelomično izvođenja obrade u lokalnim mrežnim okolinama. Zato su za evaluaciju primjene računarstva u magli važniji parametri koji se odnose na kvalitetu komunikacije i obrade u IoT uslugama (tablica 4.2) od navedenih parametara za evaluacija performanci uređaja. Utjecaj primjene računarstva u magli na uređaje Interneta stvari rijetko je bio

Tablica 4.1: QoS parametri IoT usluga u postojećoj literaturi

znanstveni rad	sloj IoT arhitekture	QoS parametri
[68]	N	vrijeme dostupnosti, prostor na kojem je usluga dostupna, vrijeme obrade, reputacija usluge.
[69]	N	potrošnja energije
[70]	V	kapacitet memorije za pohranu podataka, vrijeme operacijske dostupnosti, cijena usluge, razina pružene sigurnost.
[71]	S	metrika prethodnih korištenja (trajanje pokretanja usluge, lokacija korištenja, frekvencija korištenja usluge).
[72]	V	cijena, dostupnost, privatnost, brzina, kapacitet.
[73]	S	kašnjenje, propusnost, stopa uspješnosti isporuke, varijacija kašnjenja.
[74]	S	**komunikacija**: pouzdanost, dostupnost mreže, propusnost, varijacija kašnjenja, sigurnosti i privatnost, interoperabilnost, cijena, SLA (engl. *Service Level Agreement*); **uređaji**: interoperabilnost, potrošnja energije, dostupnost, dugoročna stabilnost, radna temperatura, osjetljivost, preciznost, bežično ažuriranje (OTA, engl. *Over-the-air programming*), memorija, razlučivost; **obrada**: skalabilnost, dinamička dostupnost, pouzdanost, cijena resursa, brzina odgovora, kapacitet, sigurnost i privatnost, preciznost, podrška za korisnike, reputacija.

motivacija postojećih istraživačkih radova. Tek se u nekim pristupima u obzir uzimala energetska učinkovitost pri raspoređivanju usluga, što ne utječe na kvalitetu isporuke usluge korisniku pa nema bitan utjecaj u okviru ove disertacije.

Tablica 4.2: Kvantitativni parametri kvalitete usluge za evaluaciju računarstva u magli [74]

komunikacija	obrada
• *pouzdanost* • *varijacija kašnjenja* • *interoperabilnost* • *cijena* • *dostupnost mreže*	• *skalabilnost* • *kapacitet* • *dostupnost* • *pouzdanost* • *cijena resursa* • *brzina odgovora*

Utjecaj računarstva u magli na navedene parametre koji se odnose na komunikaciju u [74] je značajan, što je posljedica prebacivanja izvođenja logike IoT usluge prema lokalnim okolinama. Tako čvorovi u međusobnoj komunikaciji postaju bliži, a često se nalaze i unutar istih mrežnih okolina, pa se smanjuje utjecaj nedostataka uzrokovanih komunikacijom kroz javnu internetsku mrežu. Komunikacija javnom internetskom mrežom predstavlja najveći rizik u slučajevima parametara kvalitete usluge kojima se evaluirala *pouzdanost*, *varijacija kašnjenja* (engl. *jitter*), te *sigurnost* i *privatnost* u komunikaciji, pa bi se primjenom računarstva u magli u najvećoj mjeri podigla efikasnost isporuke usluge u okviru ovih parametara. Potreba za komunikacijom javnom internetskom mrežom smanjila bi se ovisno o tome koliko bi obrada u sloju računarstva u magli ovisila o računalnom oblaku, što bi utjecalo na mjeru u kojoj bi se povećala efikasnost isporuke usluge u kontekstu navedenih parametara. U slučaju parametara *interoperabilnosti* i *cijene* komunikacije, utjecaj primjene računarstva u magli imao bi manji efekt. Ipak, u određenim scenarijima bi se i u smislu ovih parametara mogli ostvariti određeni dobici budući da čvorovi računarstva u magli često imaju mogućnost komunikacije LPWAN protokolima direktno s krajnjim uređajima. Manji utjecaj bi primjena računarstva u magli ostvarila u kontekstu parametara *dostupnosti javne mreže*, *propusnosti* i *SLA - Service Level Agreement*, budući da su to parametri izvedbene okoline na koje se ne može utjecati arhitekturom u isporuci određene IoT usluge.

Utjecaj računarstva u magli na navedene parametre za evaluaciju obrade podataka IoT usluga u [74] također je značajan, budući da je izvođenje obrade u lokalnim okolinama otpornije i omogućuje sustavu brže izvođenje vlastitih funkcija. Tako bi primjena računarstva u magli mogla podignuti kvalitetu usluge u kontekstu parametara: *skalabilnosti*, *dostupnosti*, *pouzdanosti*, *cijene resursa*, *brzine odgovora*, *kapaciteta*, te *sigurnosti i privatnosti*, dok u kontekstu *preciznosti*, *podrške za korisnike*, i *reputacije* ne bi imala značajan utjecaj. Dodavanjem re-

sursa za obradu u lokalne okoline podiže se razina *skalabilnosti* cjelokupnog sustava i njegov *kapacitet*, budući da sustav dobiva dodatne resurse za obradu, raspoređene tako da omoguće veću efikasnost u isporuci usluge. Mogućnost izvođenja lokalne obrade i filtriranja podataka u sloju računarstva u magli rasteretit će računalni oblak i omogućiti da se obrada izvede i u trenucima ispada mreže. Time će se povećati *dostupnost* obrade, podignuti *pouzdanost* sustava, te smanjiti *cijena resursa* potrebnih za obradu u računalnom oblaku. Ipak, najveći efekt primjena računarstva u magli može ostvariti u kontekstu parametra *brzine odgovora* koji se primjenjuje za evaluaciju svih sustava koji isporučuju uslugu kroz internetsku mrežu. Smanjivanje kašnjenja kojim se direktno utječe na brzinu odgovora bio je jedan od glavnih motiva za nastanak ovog koncepta, tako da je upravo ovaj parametar najčešće korišten za evaluaciju efikasnosti sustava koji integrira sloj računarstva u magli u vlastitoj obradi.

Iz navedene analize postojećih parametara za evaluacija kvalitete isporuke IoT usluga, može se zaključiti da ih je većina primjenjiva u svrhu analize efikasnosti primjene računarstva u magli. Kako bi se odredili oni parametri kojima bi se mogla usporediti kvaliteta isporuke IoT usluge postojećom arhitekturom i arhitekturom koja uključuje primjenu računarstva u magli, potrebno je izdvojiti one parametre kojima se kvantitativno mjeri efikasnost primjene ovog koncepta na ciljeve koji se njegovom primjenom žele postići.

U [75] je napravljen pregled taksonomija i zahtjeva koje računarstvo u magli treba ispuniti, te su analizirani razlozi primjene ovog koncepta u postojećim znanstvenim radovima. U najvećem broju radova motivi za primjenu ovog koncepta bili su smanjivanje kašnjenja i ušteda propusnosti u komunikacijskom kanalu. U manjem broju radova kao razlozi primjene navode se i raspodjela tereta računalne obrade, upravljanje izvođenjem usluga u lokalnim okolinama, privatnost i sigurnost podataka, praćenje rada krajnjih uređaja, energetska i novčana ušteda, te omogućavanje lokalnog priručnog spremanja podataka. U kontekstu navedenih ciljeva primjena računarstva u magli podigla bi efikasnost isporuke IoT usluga u odnosu na centralni model obrade kakvog podrazumijeva izvođenje kompletne obrade u računalnom oblaku. Tako se može zaključiti da je primjena računarstva u magli primarno usmjerena na rješavanje sljedećih nedostataka arhitekture usmjerene isključivo na računalni oblak: zagušenost propusnosti mrežnog komunikacijskog kanala, kašnjenje između zahtjeva i odgovora, povećani rizik za privatnost i sigurnost IoT usluga, te ovisnosti isporuke usluge o javnoj internetskoj mreži [76].

Zato najprije izdvajamo sljedeća četiri primarna cilja koja je potrebno uzeti u obzir pri evaluaciji efikasnosti primjene računarstva u magli na kvalitetu isporuke IoT usluga:

- smanjivanje kašnjenja odziva IoT usluge,
- smanjivanje generiranog mrežnog prometa u javnom komunikacijskom kanalu,
- smanjivanje ovisnosti cjelokupnog rješenja o računalnom oblaku,
- podizanje razine sigurnosti i privatnosti usluge.

Procjena kvalitete IoT usluge obuhvaća sve dijelove u njenoj isporuci zajedno. Zato su rele-

vantni parametri iz tablice 4.2 koji su u [74] analizirani razdvojeno prema funkcionalnom dijelu IoT scenarija kojem pripadaju integrirani u sljedeće parametre koje izdvajamo za evaluaciju primjene računarstva u magli: pouzdanost, smanjivanje podatkovnog prometa generiranog u javnu mrežu, kašnjenje, skalabilnost, te sigurnost i privatnost. U nastavku je opisan način na koji primjena računarstva u magli utječe na navedene parametre, te način na koji se pomoću njih može kvantificirati efikasnost IoT usluge u okviru prethodno navedenih ciljeva primjene računarstva u magli.

4.1.1 Kašnjenje

Kašnjenje je glavni parametar za evaluaciju efikasnosti raspodijeljenih računalnih sustava, a podrazumijeva vrijeme između trenutka u kojem je zahtjev poslan i trenutka u kojem je primljen odgovor koji potvrđuje uspješnu (ili neuspješnu) obradu istog. Ukupno kašnjenje u raspodijeljenom sustavu sastoji se od kašnjenja u slanju, propagacijskog (prijenosnog) kašnjenja u mreži, i operacijskog kašnjenja koje obuhvaća čekanje na obradu u odredišnom čvoru te samo vrijeme posluživanja. Računarstvo u magli omogućuje obradu zahtjeva na čvorovima bliže uređajima koji generiraju zahtjeve, ili u najboljem slučaju na čvorovima u njihovoj lokalnoj okolini. Tako se primjenom ovog koncepta može smanjiti prvenstveno propagacijsko i prijenosno kašnjenje u komunikaciji, a dodatno i kašnjenje u obradi zbog bolje skalabilnosti sustava. Rezultat primjene računarstva u magli na arhitekturu isporuke IoT usluga se u kontekstu ovog parametra može evaluirati mjerenjem ukupnog vremena (T_{UK}) između trenutka kada je zahtjev poslan ($t_{zahtjev}$) i trenutka kada je primljen rezultat njegove obrade ($t_{rezultat}$):

$$T_{UK} = t_{rezultat} - t_{zahtjev} \tag{4.1}$$

4.1.2 Pouzdanost

Pouzdanost usluge se definira kao vjerojatnost sustava da obradi ulazne zahtjeve u određenom vremenskom periodu. Primjena računarstva u magli može podići razinu pouzdanosti sustava budući da se omogućuje raspodijeljena obrada koja se često izvodi u lokalnim mrežnim okolinama. Tako se u određenim slučajevima dostupnost sustava nudi i u periodima kada dođe do ispada internetske mreže, te se lakše rješavaju situacije ispada čvorova na kojima se izvršava obrada, što podiže ukupnu razinu pouzdanosti usluge. Formula kojom se može evaluirati pouzdanost sustava (R) je omjer između broja uspješno obrađenih zahtjeva (n_S) i ukupnog broja poslanih zahtjeva u sustav (N_T):

$$R = \frac{n_S}{N_T} \tag{4.2}$$

4.1.3 Smanjivanje podatkovnog prometa prenesenog kroz javnu mrežu

Količina generiranog mrežnog prometa koju stvaraju međusobno povezani pametni uređaji Interneta stvari često se ističe kao nedostatak ovog koncepta. Iako je kapacitet javne internetske mreže još uvijek dovoljan za prijenos tako generiranih količina podatkovnog prometa, tempo rasta broja povezanih uređaja predstavlja rizik od nastanka mrežnog zagušenja u budućnosti. Primjena računarstva u magli može implicitno smanjiti količinu podatkovnog prometa koji se generira prema javnoj internetskoj mreži, budući da se usluge ili njihovi dijelovi mogu izvoditi u lokalnim mrežnim okolinama. Tako se čvorovi računarstva u magli mogu koristiti kao filtri redundancije u izlaznim podatkovnim tokovima ili za agregaciju većeg broja podataka u jedinstvene poruke kako bi se smanjila ukupna količina prometa koji se generira u javnu internetsku mrežu. Formula za precizno računanje ovog parametra ne postoji, ali se arhitekture isporuke određene IoT usluge mogu usporedno evaluirati prema količini podatkovnog prometa koji se u jedinici vremena ne proslijedi u javnu internetsku mrežu, odnosno broju zahtjeva obrađenih u lokalnoj mreži.

4.1.4 Skalabilnost

Skalabilna arhitektura podrazumijeva skaliranje računalnih resursa ovisno o broju ulaznih zahtjeva koji pristižu u sustav kako bi se zadržalo konstantno vrijeme obrade zahtjeva. Skaliranje se može izvoditi horizontalno i vertikalno, što najčešće ovisi o primjenjivanom arhitekturnom modelu rješenja. Tako se u računalnom oblaku obično primjenjuje i vertikalno i horizontalno skaliranje računalnih resursa u sustavu, dok računarstvo u magli podrazumijeva horizontalni pristup. Osim većeg broja resursa za obradu, horizontalnim skaliranjem se povećava i efikasnost u kontekstu komunikacijskog kašnjenja pa bi se ovakvim skaliranjem mogla postići veća ukupna efektivnost isporuke usluge u odnosu na jednodimenzionalno vertikalno skaliranje kakvo se primjenjuje u računalnom oblaku. Kako je cilj skalabilnog sustava prvenstveno osigurati konstantnu efikasnost u isporuci usluge, propusnost je potrebno povećati razmjerno povećanju ulaznih zahtjeva. To se može izračunati prema formuli za propusnost (P) u kojoj se ona definira kao broj posluženih zahtjeva (N) u jedinici vremena (t):

$$P = \frac{N}{t} \tag{4.3}$$

4.1.5 Sigurnost i privatnost

Sigurnosna ranjivost IoT uređaja i cjelokupnih arhitektura Interneta stvari često se spominje kao glavna prepreka za široku primjenu rješenja ovog koncepta. Razlog tome su brojna istraživanja koja su demonstrirala ranjivosti ovakvih scenarija uzrokovanih najvećim dijelom nedovoljnom sigurnosnom zaštitom krajnjih uređaja [77] [78], što je posljedica zanemarivanja sigurnosti i

privatnosti uređaja i aplikacija u inicijalnim fazama njihova dizajna i razvoja [79]. Kako bi se promijenila takva praksa, u okviru projekta OWASP (engl. *Open Web Application Security Project*) je pokrenuta posebna inicijativa usmjerena na IoT koja je namijenjena proizvođačima opreme, razvojnim inženjerima, i samim korisnicima IoT aplikacija, a cilj joj je upoznati ih sa sigurnosnim rizicima u Internetu stvari. U okviru tog projekta, objavljeno je deset rizika koje je potrebno izbjegavati u razvoju i korištenju IoT sustava kako bi se podigla razina njegove sigurnosti [80]. Pomoću navedenih točaka, te uzimajući u obzir postojeće sigurnosne rizike za web-aplikacije [81], mogu se identificirati ranjivosti IoT sustava. Kvantifikacija sigurnosti može se onda provesti primjenom metode STRIDE (engl. *Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege*) za klasifikaciju identificiranih prijetnji, te metode DREAD (engl. *Damage, Reproducibility, Exploitability, Affected Users, Discoverability*) za računanje ocjene koja opisuje razinu prijetnje od napada na promatranu arhitekturu sustava [82]. Ipak, primjenom računarstva u magli može se implicitno podignuti ukupna razina sigurnosti IoT sustava. Čvorovi računarstva u magli mogu se nalaziti u lokalnim okolinama krajnjih uređaja pa tako mogu izvoditi sigurnosne mehanizme za zaštitu ranjivih krajnjih uređaja te algoritme sigurnosne zaštite poruka prije njihova slanja u javnu internetsku mrežu. Stoga se može zaključiti da arhitektura koja uključuje sloj računarstva u magli osigurava preduvjete za podizanje razine sigurnosti IoT usluge, ali je za konačno postizanje takvog cilja potrebno dati prioritet lokalnom izvođenju komponenti usluge pri njihovom raspoređivanju. Evaluacija sigurnosti arhitekture IoT usluge zasniva se upravo na ovom načelu budući da se na sigurnost unutar aplikacijskih rješenja ne može utjecati primjenom različitih arhitektura u njihovoj isporuci.

4.2 Kategorizacija IoT usluga

Većina dostupnih IoT usluga podrazumijeva modularni dizajn u kojem su komponente raspoređene između ciljanih lokalnih okolina i računalnog oblaka. Iako danas postoji široki spektar primjena Interneta stvari, u svim se scenarijima mogu izdvojiti tri osnovne funkcionalne skupine u njihovoj isporuci koje zajedno ostvaruju cilj u specifičnom slučaju primjene. Zato najprije definiramo funkcionalne skupine u koje se može svrstati većina komponenti IoT usluga:

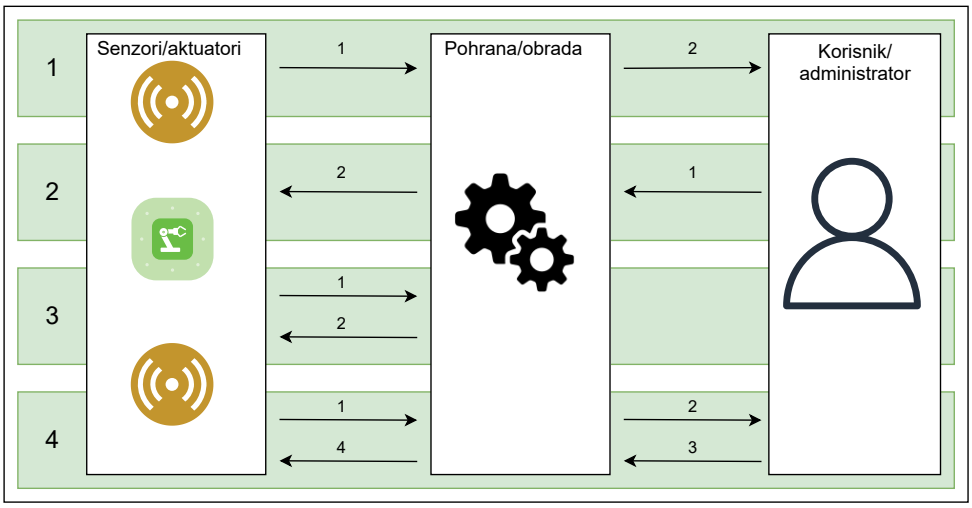
- interakcija s krajnjim uređajima (prikupljanje senzorskih podataka ili upravljanje aktuatorima),
- obrada podataka, i
- pohrana podataka.

Funkcionalne skupine koje će određeni scenarij sadržavati i njihov arhitekturalni raspored ovise o namjeni specifične usluge. Unutar svake funkcionalne skupine može se nalaziti više odvojenih komponenti, a za njihovo raspoređivanje važno je znati kojoj od navedenih funk-

cionalnih skupina pripadaju i u kojem redoslijedu se izvršava obrada zahtjeva u konkretnom scenariju. Na temelju tih informacija može se odrediti efikasniji raspored njihova izvođenja u raspodijeljenoj okolini Interneta stvari koja uključuje računarstvo u magli. Zato je prije dizajna algoritma za raspoređivanje IoT usluga napravljena njihova kategorizacija kojom su definirane četiri karakteristične kategorije koje opisuju osnovnu namjenu i redoslijed izvođenja obrade u specifičnom scenariju:

1. usluge prikupljanja podataka,
2. usluge neposredne aktuacije,
3. autonomne usluge i
4. usluge korisnički upravljane automatizacije.

Na slici 4.1 su zelenom bojom označene navedene kategorije IoT usluga, te je strelicama unutar svake kategorije opisan očekivani redoslijed interakcije između sudjelujućih entiteta u IoT scenarijima.



Slika 4.1: Redoslijed izvođenja funkcionalnih skupina i interakcije u četiri izdvojene kategorije IoT usluga.

Usluge prikupljanja podataka i usluge neposredne aktuacije podrazumijevaju jednostavnije scenarije IoT usluga u kojima se funkcionalnosti krajnjih uređaja isporučuju direktno korisnicima, bez dodatne obrade. Tako usluge prikupljanja podataka obuhvaćaju aplikacije u kojima se senzorski podaci prikupljaju i direktno isporučuju korisnicima ili se pohranjuju u baze podataka koje su najčešće smještene u računalnom oblaku (npr. *smart metering* usluge). Nedostaci ovakvih usluga koji se često ističu u njihovoj analizi, a na koje bi se moglo utjecati primjenom računarstva u magli su količina generiranog podatkovnog prometa kroz javnu internetsku mrežu i sigurnost podataka. Zato je kod ovakvih usluga raspoređivanje njihovih komponenti u sloju računarstva u magli potrebno usmjeriti prvenstveno na ova dva parametra kvalitete usluge.

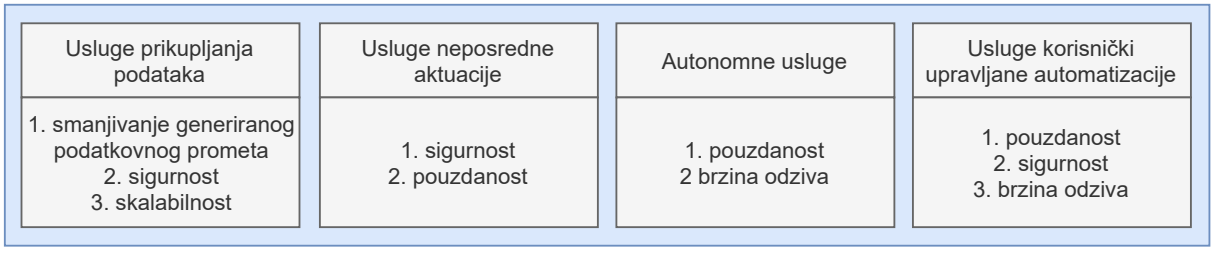
Usluge neposredne aktuacije podrazumijevaju aplikacije u kojima se korisnicima omogućuje udaljeno upravljanje aktuatorima smještenim u ciljanim okolinama (npr. usluge udaljenog

upravljanja pametnim bravama, udaljena kontrola sustava grijanja i hlađenja, itd.). U takvim je scenarijima posebno naglašena osjetljivost na sigurnost i pouzdanost rješenja budući da se aktuatorima utječe na fizički svijet. Zato je za raspoređivanje komponenti u ovakvim scenarijima potrebno najprije uzeti u obzir ova dva QoS parametra, a potom na ostale navedene faktore za evaluaciju isporuke IoT usluga.

Druge dvije kategorije podrazumijevaju kompleksnije scenarije koji obuhvaćaju sve prethodno navedene funkcionalne skupine IoT usluga. Autonomne usluge odnose se na scenarije u kojima se upravljanje aktuatorima izvodi u potpunosti autonomno na temelju senzorskih podataka prikupljenih iz ciljanih okolina (npr. autonomna vožnja, automatske parkirne rampe, itd.). U takvim scenarijima kritični parametri kvalitete usluge su pouzdanost i brzina odziva, budući da se reakcije na stanje u ciljanim okolinama često moraju izvršiti gotovo trenutno, u stvarnome vremenu.

Posljednja kategorija podrazumijeva usluge korisnički upravljane automatizacije, odnosno scenarije u kojima aktuacija, koja se također izvodi na temelju senzorskih podataka iz ciljanih okolina, nije u potpunosti autonomna, nego je u određenoj mjeri kontrolira korisnik (npr. pametni interfon, dron-kamere, itd.). Tako je u ovoj kategoriji pri raspoređivanju komponenti određenog scenarija, uz prethodno navedenu pouzdanost i brzinu odziva, poseban naglasak potrebno staviti na sigurnost. Razlog tome je što u ovakvim scenarijima također postoji mogućnost napada kojim bi kontrolu aktuacije mogle preuzeti neautorizirane treće strane, što zbog mogućnosti utjecaja na fizički svijet predstavlja dodatni rizik.

Tako se može zaključiti da je za raspoređivanje navedenih kategorija IoT usluga u okolini računarstva u magli potrebno u obzir uzeti sve navedene QoS parametre, ali ovisno o kategoriji njihov prioritet se može razlikovati. Na slici 4.2 prikazane su definirane četiri kategorije IoT usluga, te redoslijed prioriteta pripadajućih parametara kvalitete usluge koje je potrebno uzeti u obzir pri njihovu raspoređivanju.



Slika 4.2: Prioritet parametara kvalitete usluge u izdvojenim karakterističnim kategorijama IoT usluga.

4.3 Ograničenja definirana parametrima konteksta usluge

U poglavlju 3 su definirana tri konteksta koja utječu na izvođenje raspoređivanja IoT usluga u okolini računarstva u magli: kontekst uređaja, kontekst korisnika, i kontekst usluge. Kontekst

korisnika opisuje ciljanu kvalitetu korisnika i okolinu u kojoj se korisnik nalazi u određenom trenutku, dok kontekst uređaja definira dostupnu računalnu infrastrukturu u kojoj je potrebno izvesti raspoređivanje. Kontekst usluge definira konkretan scenarij IoT usluge, pa se tako parametrima ovog konteksta opisuju razlike u načinu izvođenja različitih IoT usluga koje utječu na izvedbu njihova raspoređivanja. Zato su upravo parametrima konteksta usluge određena pravila i ograničenja koja je potrebno uzeti u obzir kako bi se za konkretan scenarij prema trenutnom stanju korisničke i izvedbene okoline odredio i postigao efikasan raspored izvođenja komponenti usluge.

Ograničenja na temelju parametara konteksta usluge mogu se odnositi na ciljanu izvedbenu okolinu za njene komponente ili na učestalost izvođenja migracija njenih komponenti. Kako bi se ostvario potencijal primjene računarstva u magli, komponente IoT usluga se primarno pokreću u lokalnim okolinama ako postoji dostupna računalna infrastruktura koja to i omogućuje. Ipak, kod nekih scenarija je određene komponente potrebno pokrenuti u konkretnom sloju izvedbene okoline kako bi se postigla njihova funkcionalnost. Dodatno, kako je migracija komponenti usluge proces koji može uzrokovati određeno vrijeme ispada usluge, u nekim je situacijama potrebno ograničiti frekvenciju njena izvođenja. Zato je prije izvođenja raspoređivanja komponenti IoT usluge potrebno analizirati parametre njena konteksta koji određuju navedena ograničenja.

Perzistentnost pohrane podataka može utjecati na raspoređivanje komponenti određenog scenarija ako se u konkretnom slučaju radi o perzistentnoj pohrani podataka koja se izvodi unutar određene komponente. Izvođenje migracije takvih komponenti usluge uključivalo bi i kopiranje njihovog cjelokupnog podatkovnog skupa. Takav proces može znatno povećati vrijeme nedostupnosti usluge zbog duljeg trajanja migracije njenih komponenti, pa je u ovakvim scenarijima potrebno izbjegavati ili ograničiti frekvenciju izvođenja migracija.

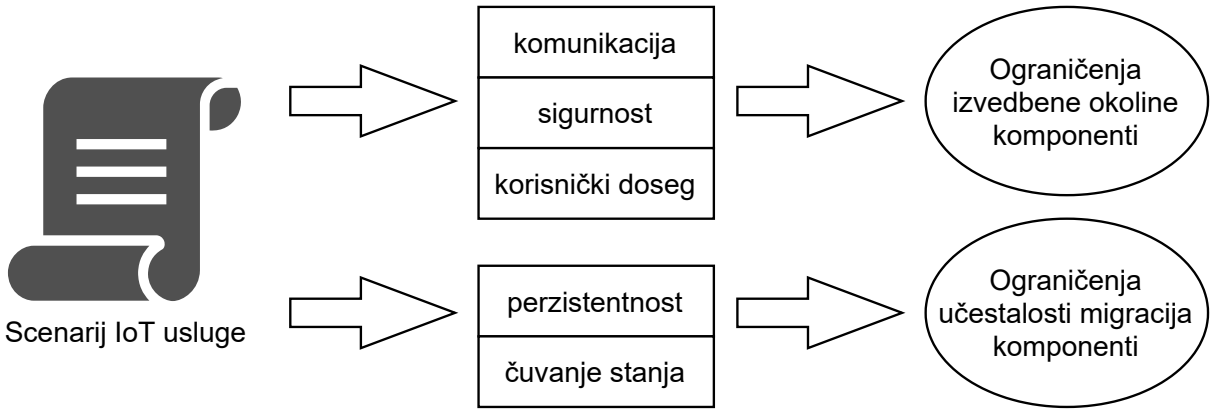
Jednak utjecaj na raspoređivanje komponenti kao i parametar perzistentnosti ima parametar čuvanja stanja. Kod migracije komponenti usluga koje čuvaju stanje potrebno je kopirati i njihovo cjelokupno stanje kako bi se one mogle nastaviti pravilno izvoditi u novoj okolini. Takav postupak podrazumijeva gotovo uvijek kratkotrajan period ispada komponenti koje se migriraju [27], zbog čega je za usluge koje čuvaju stanje obično potrebno ograničiti učestalost izvođenja migracija njihovih komponenti.

Parametar korisničkog dosega odnosi se na općenitost primjene određene IoT usluge te na mobilnost njenih korisnika. Za izvođenje raspoređivanja komponenti usluge koja je usmjerena na određenu korisničku skupinu (ili korisnika) posebno je važna informacija o njihovoj mobilnosti. Na temelju te informacije može se pretpostaviti dinamika kojom će biti potrebno izvoditi raspoređivanje komponenti usluge. Tako se u scenarijima koji podrazumijevaju mobilne korisnike očekuje veća učestalost migracija komponenti, pa je u nekim slučajevima potrebno postaviti ograničenja na frekvenciju njihova izvođenja.

Specifikacija komunikacije u okviru IoT scenarija može postaviti restrikcije na ciljanu izvedbenu okolinu za njene komponente. Posebno važan element ovog parametra konteksta usluge u tom smislu je specifikacija dodatnih komunikacijskih protokola potrebnih za njeno izvođenje. Tako će komponente usluge koje zahtijevaju specifične ograničene komunikacijske protokole za interakciju s krajnjim uređajima biti potrebno rasporediti u lokalne okoline na čvorove koji imaju module za izvođenje takve komunikacije.

Sigurnost je parametar kojim se definira kritičnost sigurnosne osjetljivosti IoT scenarija. Kako bi se omogućilo podizanje razine sigurnosti usluga u kojima je izražen sigurnosni rizik, potrebno je odgovarajućom arhitekturom umanjiti mogućnost napada. Zato ovaj parametar također ima ulogu u definiranju izvedbene okoline za određene komponente IoT usluga. Tako će u scenarijima gdje postoji visoki sigurnosni rizik biti potrebno rasporediti ulazne komponente u lokalne okoline kako bi se postigla dostatna razina sigurnosti.

Parametri konteksta usluge kojim je opisana brzina odziva i ciljana razina QoS parametara ne postavljaju nikakva ograničenja u raspoređivanju komponenti IoT usluge. Algoritam raspoređivanja treba u što većoj mjeri podići razinu svih navedenih QoS parametara za evaluaciju primjene računarstva u magli. Zato se ovi parametri uzimaju u obzir u implementaciji logike algoritma raspoređivanja zbog čega se na temelju njih ne zadaju nikakva dodatna ograničenja koja je potrebno uzeti u obzir.



Slika 4.3: Utjecaj parametara konteksta usluge na ograničenja njihova raspoređivanja.

Konačno se može definirati da komponente IoT scenarija koji podrazumijevaju interakciju s krajnjim uređajima korištenjem ograničenih komunikacijskih protokola, visoki sigurnosni rizik, ili su usmjereni na konkretnu korisničku skupinu, treba pokrenuti u lokalnoj korisničkoj okolini (gornji dio slike 4.3). To se posebno odnosi na komponente koje pripadaju funkcionalnoj skupini interakcije s krajnjim uređajima budući da upravo one najčešće trebaju biti blizu relevantne okoline kako bi se zadana ograničenja ispunila. Također, potrebno je ograničiti migraciju komponenti usluge koje čuvaju stanje ili podrazumijevaju perzistentno spremanje podataka (donji dio slike 4.3). Razlog tome je mala efektivnost izvođenja čestih migracija u takvim slučajevima zbog kompleksnosti i duljine trajanja migracija koje uzrokuju i ispad usluge. Zato je prije iz-

vođenja raspoređivanja komponenti IoT usluge potrebno analizirati navedene parametre njenog konteksta i postaviti navedena ograničenja za raspoređivanje njenih komponenti.

Poglavlje 5

Formalni model kategoriziranih usluga Interneta stvari

Kako bi se omogućio razvoj i implementacija sustava za raspoređivanje IoT usluga u okolini računarstva u magli potrebno je najprije formalno specificirati prethodno definirane kontekste. Ovo poglavlje definira formalni model konteksta uređaja, korisnika, i usluga kako bi se odredio način na koji bi se parametri koji određuju navedene kontekste primjenjivali u implementaciji sustava za raspoređivanje IoT usluga. U definiciji modela napravljena je specifikacija vrijednosti kojim se opisuju parametri navedenih konteksta i analiza njihove primjene u razvoju algoritma za raspoređivanje IoT usluga. Zatim je u poglavlju 5.4 predstavljena formalna specifikacija ciljne funkcije raspoređivanja IoT usluga u okolini računarstva u magli, kako bi se prema njoj mogao razviti algoritam raspoređivanja kojim bi se iskoristio potencijal primjene ovog koncepta i podigla efikasnost isporuke IoT usluga.

5.1 Model konteksta uređaja

Parametri konteksta uređaja definirani u potpoglavlju 3.1 opisuju dostupne resurse računalne infrastrukture u raspodijeljenoj okolini računarstva u magli. Kako bi se mogao razviti sustav koji bi izvodio raspoređivanje komponenti IoT usluga u takvoj okolini, potrebno je specificirati navedene parametre odgovarajućim atributima. Kontekst jednog uređaja modeliramo pomoću sljedećeg skupa parametara:

$$\boldsymbol{D} = \{\boldsymbol{P_S}, \boldsymbol{P_{RAM}}, \boldsymbol{P_{CPU}}, \boldsymbol{L}, \boldsymbol{E}, \boldsymbol{C}\},$$

gdje je P_S memorija za pohranu, P_{RAM} RAM memorija, P_{CPU} procesor, L lokacija, E način napajanja, a C način komunikacije konkretnog uređaja. Vrijednosti koje navedeni parametri mogu poprimiti opisane su u tablici 5.1, te se pomoću njih sustavu omogućava uvid u stanje dostupnih računalnih resursa i ograničenja čvorova na koje je moguće rasporediti izvedbene

komponente različitih IoT usluga.

Tablica 5.1: Model konteksta uređaja

Oznaka	Parametar konteksta uređaja	Vrijednost	Obavezno
m_{P_s}	količina memorije za pohranu	numerička (MB)	DA
f_{P_s}	frekvencija rada memorije za pohranu	numerička (MHz)	NE
$m_{P_{RAM}}$	količina RAM memorije	numerička (MB)	DA
$f_{P_{RAM}}$	frekvencija rada RAM memorije	numerička (MHz)	NE
$n_{P_{CPU}}$	broj jezgri procesora	numerička	DA
$f_{P_{CPU}}$	frekvencija rada procesora	numerička (MHz)	NE
s_E	način napajanja	BAT - baterijski AC - statični priključak	DA
c_E	dostupni kapacitet baterije	numerička (%)	DA (ako s_E=b)
b_C	maksimalna propusnost veze (kapacitet)	numerička (Mbps)	DA
p_C	drugi podržani komunikacijski protokoli	znakovna	NE
a_L	mrežna adresa	znakovna (IP adresa)	DA
o_L	drugi tip lokacije	znakovna	NE

Kod prva tri parametra u navedenom skupu koja opisuju memoriju za pohranu (P_S), *RAM* memoriju (P_{RAM}) i procesor (P_{CPU}) specificirane su po dvije vrijednosti koje ih određuju: broj jezgri ($n_{P_{CPU}}$) i frekvencija rada ($f_{P_{CPU}}$) u slučaju parametra koji opisuje procesor, te količina memorije (m_{P_s} i $m_{P_{RAM}}$) i frekvencija rada (f_{P_s} *i* $f_{P_{RAM}}$) u slučaju parametara koji opisuju *RAM* memoriju i memoriju za pohranu. Kako se u specifikaciji minimalnih resursa potrebnih za izvođenje komponenti usluge rijetko specificira frekvencija rada navedenih komponenti računalnog čvora, ona je opcionalna vrijednost u sva tri slučaja, te ju sustav može koristiti kao dodatni argument u odabiru ako više čvorova zadovoljava preduvjete za izvođenje usluge. Suprotno tome, količina memorije i broj jezgri čvora su obvezne informacije koje sustav treba imati kako bi mogao raspoređivati komponente usluge. Razlog tome je što se u specifikaciji minimalnih resursa za izvođenje IoT usluga preduvjeti za njihovo pokretanje najčešće zadaju upravo u okviru ovih vrijednosti.

Druga dva parametra koja se odnose na hardverske komponente računalnog čvora su napajanje (E) i komunikacija (C). Na temelju informacije o tome da li je određeni čvor baterijski napajan ili je direktno priključen na energetsku mrežu (s_E), sustav može pretpostaviti pouzdanost njegove dostupnosti. Tako je pretpostavka da će uređaji s direktnim napajanjem imati manju učestalost promjena stanja dostupnosti budući da takva izvedba u najvećem broju slu-

čajeva podrazumijeva njihovu stalnu lokaciju. Uređaji napajani baterijama uglavnom podrazumijevaju i svojstvo mobilnosti zbog čega češće mogu mijenjati okolinu u kojoj se nalaze pa se može pretpostaviti da je njihova dostupnost manje pouzdana. Zato je u slučajevima kada sustav radi odabir čvora za izvođenje usluge prednost potrebno dati uređajima napajanim direktno iz energetske mreže, budući da oni obično podrazumijevaju veću pouzdanost. Ako se uspoređuju isključivo uređaji napajani baterijom, potrebno je u obzir uzeti i parametar dostupnog kapaciteta baterije (c_E).

Parametar komunikacije (C) uređaja specificira se pomoću dvije vrijednosti koje opisuju dostupnu propusnost internetske veze (b_C) i druge protokole koje uređaj podržava (p_C). Prvi je parametar obavezan, budući da svaki uređaj treba imati mogućnost komunikacije sa sustavom za raspoređivanje usluga koji na temelju informacije o propusnosti dostupne mreže može uspoređivati dostupne čvorove. Drugi parametar kojim se opisuju dodatno podržani komunikacijski moduli na uređaju je opcionalan, i specificira se tek u slučajevima kada čvor ima mogućnost komunikacije specifičnim protokolima koji su potrebni za izvođenje određenih komponenti usluge koje sustav raspoređuje.

Jedini parametar koji nije vezan uz hardversku podlogu uređaja koji se opisuje, nego za informaciju o njegovu kontekstu u svrhu implementacije algoritma za raspoređivanja usluga je njegova lokacija (L). Ona se u okviru raspoređivanja u okolini računarstva u magli odnosi prvenstveno na mrežnu lokaciju (a_L), budući da upravo o mrežnoj komunikaciji najvećim dijelom ovisi nivo kvalitete usluge opisan prethodno specificiranim QoS parametrima. Opcionalno se parametar može opisati i dodatnim vrijednostima (o_L) koje se odnose na neki drugi oblik lokacije koji bi u konkretnom slučaju primjene omogućio sustavu za raspoređivanje mogućnost odabira prikladnijeg čvora za pokretanje određene komponente IoT usluge (npr. GPS lokacija, adresa reda asinkrone komunikacije, itd.).


```
{
  "deviceId": null,
  "storageUsableSpace": 187830,
  "storageSpeed": 0,
  "ramSize": 16,
  "ramSpeed": 0,
  "cpuCoreNumber": 2,
  "cpuSpeed": 2500,
  "powerSupply": "AC",
  "batteryLevel": null,
  "networkCapacity": 150,
  "supportedProtocols": null,
  "publicIpAddress": "93.138.102.239",
  "privateIpAddress": "192.168.1.4",
  "location": null,
  "publiclyAvailable": false
}
```


Slika 5.1: Primjer opisa uređaja (u formatu *JSON*) prema specificiranom modelu.

Na slici 5.1 prikazan je primjer specifikacije uređaja u formatu *JSON* kojim je prema opisanom formalnom modelu uređaja opisano osobno računalo. Ovim podacima određen je kontekst čvora računarstva u magli te se on može koristiti za pokretanje komponenti usluge.

5.2 Model konteksta korisnika

Korisnički kontekst je potrebno modelirati apstraktnijim parametrima kako bi se omogućila elastičnost u opisu korisnika čija svojstva uvelike ovise o konkretnom scenariju primjene Interneta stvari. Skup kojim je određen model konteksta korisnika i čije su vrijednosti navedene u tablici 5.2, sadrži tri parametra:

$$\boldsymbol{U} = \{\boldsymbol{L_U}, \boldsymbol{V}, \boldsymbol{Q}\},$$

gdje je L_U lokacija korisnika, V stanje korisničke okoline, a Q ciljana kvalitete usluge.

Parametar lokacije korisnika (L_U) jednako se opisuje kao i prethodno specificirani parametar lokacije uređaja, a upravo je odnos lokacija dostupnih uređaja za izvođenje komponenti usluge i korisničkih lokacija osnovna informacija na temelju koje sustav može raspoređivati komponente IoT usluga s ciljem postizanja njihove efikasnije isporuke. Tako je parametar lokacije korisnika također prvenstveno određen njegovom javnom i lokalnom IP adresom (a_L), dok se ovisno o specifičnom slučaju primjene može opisati i drugi tip lokacije (o_L).

Druga dva parametra omogućuju elastičnost primjene razvijanog algoritma prema korisničkim preferencijama, te prema namjeni konkretnog slučaja primjene. Prvi parametar kojim se opisuje stanje korisničke okoline (V) odnosi se na sve podatke koji u specifičnom scenariju određuju korisničku okolinu, odnosno koji sustavu opisuju trenutno stanje relevantnih faktora u ciljanoj korisničkoj okolini. Tako je primjerice u slučaju nekih IoT usluga potrebno postići zadano stanje okoline u kojoj se korisnik nalazi, za što je najprije potrebno omogućiti sustavu informaciju o trenutnom stanju iste kako bi se aktuatorima zadale odgovarajuće naredbe (npr. ako usluga nudi postavljanje zadane temperature u određenoj okolini, potrebno je poznavati trenutnu temperaturu kako bi se aktiviralo hlađenje ili zagrijavanje).

Posljednji parametar kojim se specificira korisnički kontekst (Q) opisuje korisničke preferencije na parametre koji određuju kvalitetu usluge definiranim ranije u poglavlju 4.1. Kako u različitim scenarijima specificirani parametri kvalitete usluge imaju drugačije prioritete i ograničenja koja je potrebno ispuniti, ovim se parametrom omogućuje da sam korisnik ili administrator neke usluge specificira sustavu koje je QoS parametre potrebno uzeti u obzir, odnosno koje su konkretne granice unutar kojih trebaju biti njihove vrijednosti. Ipak, evaluacija isporuke usluge određenim rasporedom se obično izvodi na strani samog korisnika pa sustav vrijednosti ovog parametra (q_r, q_l, q_c, i q_s) treba razmatrati samo ako je potrebno prilagoditi prioritet kojim se QoS parametri uzimaju u obzir pri raspoređivanju konkretnog scenarija.

Na slici 5.2 prikazan je primjer specifikacije korisnika u formatu *JSON* kojim je prema opisanom formalnom modelu korisnika opisan njegov kontekst. Osim navedenih parametara u tablici 5.2, može se primijetiti i dodatni parametar (*serviceId*) kojim se specificira zahtijevana usluga.

Tablica 5.2: Model konteksta korisnika

Oznaka	Parametar konteksta korisnika	Vrijednost	Obavezno
a_L	mrežna adresa	znakovna (IP adresa)	DA
o_L	drugi tip lokacije	znakovna	NE
V	stanje korisničke okoline	znakovna	NE
q_r	QoS preference (pouzdanost)	numerička (%)	NE
q_l	QoS preference (kašnjenje)	numerička (ms)	NE
q_c	QoS preference (podatkovni promet)	numerička (MB)	NE
q_s	QoS preference (sigurnost)	znakovna (da/ne)	NE


```
{
  "userId": "1",
  "userPublicIpAddress": "93.138.102.239",
  "userPrivateIpAddress": "192.168.1.4",
  "location": null,
  "env": null,
  "qosR": 99,
  "qosL": 5,
  "qosC": null,
  "qosS": true,
  "serviceId":"aut1"
}
```


Slika 5.2: Primjer opisa korisnika (u formatu JSON) prema specificiranom modelu.

5.3 Model konteksta usluge

Kontekst usluge određen je parametrima opisanim u poglavlju 3.2. Navedene parametre je potrebno modelirati tako da sustav za raspoređivanje komponenti može na temelju njihovih vrijednosti prepoznati kategoriju usluge i ograničenja koja je potrebno uzeti u obzir (definirana u poglavlju 4.3) pri određivanju raspreda za njenu efikasniju isporuku. Stoga kontekst usluge definiramo sljedećim skupom parametara:

$$\boldsymbol{S} = \{\boldsymbol{P}, \boldsymbol{R}, \boldsymbol{C_S}, \boldsymbol{K}, \boldsymbol{G}, \boldsymbol{I}, \boldsymbol{Q_S}\},$$

gdje je P perzistentnost čuvanja podataka, R korisnički doseg usluge, C_S način komunikacije unutar usluge, K maksimalno vrijeme odziva, G sigurnosna osjetljivost, I čuvanje stanja u specifičnoj usluzi, a Q_S označava aplikacijska ograničenja kvalitete usluge.

Elementi skupa se specificiraju vrijednostima koje su navedene u tablici 5.3. Parametar komunikacije (C_S) je određen istim opisnim atributima kao i u modelu konteksta uređaja, pa se tako b_C odnosi na kapacitet internetske veze, dok p_C specificira ograničene komunikacijske protokole potrebne za rad konkretne IoT usluge. Ipak, za razliku od modela konteksta uređaja, vrijednost kapaciteta internetske veze u modelu konteksta usluge (b_C) definira minimalni kapa-

citet veze koji je potreban za ispravnu funkcionalnost konkretnog scenarija izvođenja usluge.

Tablica 5.3: Model konteksta usluge

Oznaka	Parametar konteksta usluge	Vrijednost	Obavezno
P	perzistentnost čuvanja podataka	znakovna (da/ne)	DA
u_R	ciljana korisnička skupina	znakovna (da/ne)	DA
m_R	mobilnost korisnika	znakovna (da/ne)	DA
b_C	minimalna propusnost veze (kapacitet)	numerička (Mbps)	NE
p_C	drugi podržani komunikacijski protokoli	znakovna	NE
K	maksimalno vrijeme odziva	numerička (ms)	NE
G	sigurnosna osjetljivost	znakovna (da/ne)	NE
I	čuvanje stanja	znakovna (da/ne)	DA
Q_S	ograničenja aplikacijske kvalitete usluge	znakovna	NE

Drugi parametar konteksta usluge koji se opisuje s dvije vrijednosti je njen korisnički doseg (R). Prvi njegov atribut (u_R) označava ciljanu korisničku skupinu IoT usluge, odnosno specificira stupanj općenitosti u njenoj isporuci (općenite namjena ili namjena specifičnoj korisničkoj skupini). Drugi atribut (m_R) daje sustavu informaciju o mobilnosti korisnika, što je važno za definiranje prethodno opisanih ograničenja u izvođenju raspoređivanja IoT usluge.

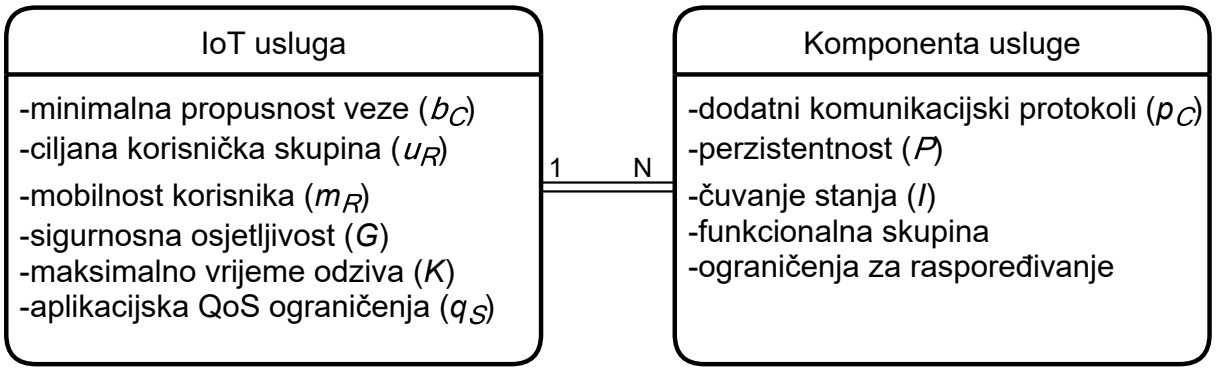
Parametri kojim se specificira perzistentnost (P) i sigurnosni rizik (G) u IoT scenariju opisuju se atributima koji mogu poprimiti samo dvije različite vrijednosti. Njima se sustavu pruža informacija o tome koristi li se u konkretnom slučaju primjene perzistentna pohrana i da li je u njegovom izvođenju naglašena sigurnosna osjetljivost. Na temelju tih informacija sustav prepoznaje treba li pri raspoređivanju IoT usluge uzimati u obzir ograničenja perzistentne pohrane i ograničenja određena visokom sigurnosnom osjetljivošću.

Slično se i parametar koji definira čuvanje stanja u određenom scenariju (I) definira prvenstveno s dvije vrijednosti koje pružaju sustavu informaciju o tome zahtjeva li se pri migraciji konkretne usluge i migracija njenog stanja. U slučajevima IoT usluga koje zahtijevaju čuvanje stanja sustav za raspoređivanje treba uzeti u obzir takvo ograničenje budući da njihova migracija podrazumijeva i određen period ispada usluge.

Posljednja dva parametra opisuju sustavu granice kvalitete usluge koje je potrebno zadovoljiti rasporedom njenih komponenti. Maksimalno vrijeme odziva (K) potrebno je definirati za usluge koje ne mogu isporučiti vlastitu funkcionalnost ako zadani uvjet nije zadovoljen. Slično se u okviru parametra kojim se specificiraju ograničenja aplikacijske kvalitete usluge (Q_S) mogu zadati i drugi uvjeti koje je u specifičnom scenariju primjene potrebno uzeti u obzir pri izvođenju raspoređivanja.

Dodatno je u okviru formalizacije usluga za njihovo raspoređivanje u okolini računarstva u magli potrebno razdvojiti pojmove IoT usluge i njenih komponenti, te odrediti koje bi od navedenih parametara trebalo pridružiti navedenim entitetima. Tako je IoT usluga zapravo apstrakcija cjelovitog scenarija koji pruža korisnicima određeno aplikacijsko rješenje, koje ovisno o namjeni može pripadati različitim kategorijama IoT usluga definiranim u poglavlju 4.2. IoT usluge obično sadržavaju više raspodijeljenih komponenti koje u međusobnoj interakciji ostvaruju namjenu usluge kojoj pripadaju. Takve komponente danas se obično isporučuju u kontejnerima koji omogućuju lakšu prenosivost i izvođenje na heterogenim platformama što je karakteristika okoline računarstva u magli. Sustav za raspoređivanje tako izvršava orkestraciju izvođenja (kontejneriziranih) komponenti usluge, a navedene parametre konteksta usluge uzima u obzir kako bi se pritom u obzir uzela razna ograničenja koje različiti scenariji mogu zahtijevati.

Na slici 5.3 je prikazan model odnosa IoT usluge i njenih komponenti, te su razdvojeni parametri modela usluge koji ih opisuju. Parametri koji opisuju IoT uslugu primjenjuju se najvećim dijelom kako bi se definirala ograničenja raspoređivanja za sve komponente koje ona sadrži. Zato je među parametre komponente usluge dodan parametar ograničenja u njihovom raspoređivanju kojim se definiraju restrikcije na okoline u kojim određena komponenta može biti pokrenuta. Također, definiran je i drugi dodatni parametar komponente IoT usluge koji može utjecati na okolinu u kojoj će ona biti pokrenuta, a to je funkcionalna skupina kojoj ona pripada. Tako se na kraju može zaključiti da sustav za raspoređivanje donosi odluku o čvoru na kojem će određena komponenta IoT usluge biti pokrenuta na temelju parametara kojim se ona opisuje na slici 5.3. Pritom se parametri IoT usluge također uzimaju u obzir budući da oni određuju ograničenja za raspoređivanje svake pripadajuće komponente. Primjer specifikacije usluge prema opisanom formalnom modelu prikazan je na slici 5.4, dok je na slici 5.5 prikazan opis jedne komponente usluge.



Slika 5.3: Odnos podatkovnog modela IoT usluge i njenih komponenti.

```
{
  "serviceId":"aut1",
  "minThroughput": 100,
  "specificUserReach":true,
  "hasMobileUsers":true,
  "highSecurityRisk":false,
  "maxLatency":null,
  "persistentDataStorage":true,
  "serviceComponentIds":{
       "component-process": null,
       "component-receive": null,
       "component-store": null
  }
}
```


Slika 5.4: Primjer opisa IoT usluge (u formatu JSON) prema specificiranom modelu.


```
{
  "scId":"component-receive",
  "imageName":"receive.jar",
  "otherProtocolsRequired":false,
  "persistentDataStorage":false,
  "statefull":true,
  "operationalGroup":"DEVICE_MANAGEMENT",
  "envRestrictions" : "local"
}
```


Slika 5.5: Primjer opisa komponente usluge (u formatu JSON) prema specificiranom modelu.

5.4 Ciljna funkcija raspoređivanja usluga u okolini računarstva u magli

Prije razvoja algoritma za raspoređivanje IoT usluga potrebno je definirati ciljnu funkciju koja obuhvaća sve prethodno opisane faktore koji utječu na efikasnost primjene računarstva u magli. Osnovna ideja primjene ovog koncepta u okruženjima Interneta stvari je podizanje kvalitete u isporuci usluga izvođenjem njihovih komponenti bliže korisnicima. Tako se za sve prethodno navedene kategorije IoT usluga primjenom računarstva u magli može postići veća razina kvalitete u njihovoj isporuci prema ranije definiranim QoS parametrima. Zato je primarni cilj algoritma za raspoređivanje dati prioritet izvođenju komponenti u lokalnim okolinama korisnika, budući da lokalna interakcija između korisnika (U) i usluge (S) podiže razinu kvalitete u okviru svih navedenih QoS parametara, pa definiramo sljedeću ciljnu funkciju:

$$\Phi(sc_i) = max(locality^{fn_k,U}) \tag{5.1}$$

Kako je prethodno opisano, IoT uslugu (S) definiramo kao skup jedne ili više komponenti (sc) koje izvode njenu logiku:

$$sc_i \in S, \quad i = 1,...,n \tag{5.2}$$

dok izvedbenu okolinu računarstva u magli (F) čini m dostupnih čvorova za pokretanje komponenti usluge. Dostupni čvorovi računarstva u magli mogu biti smješteni u računalnom oblaku (F_c), lokalnoj mreži korisnika (F_l), ili drugoj lokalnoj mreži s preduvjetom da im je omogućen

pristup iz javne internetske mreže (F_p), što se može opisati sljedećim izrazom:

$$fn_j \in F, \quad j = 1,...,m \quad \text{i} \quad F = F_c \cup F_l \cup F_p \tag{5.3}$$

Kako bi se dao prioritet izvođenju komponenata usluge u lokalnim okolinama, u prvom koraku raspoređivanja potrebno je razmatrati samo čvorove koji se nalaze u lokalnoj okolini korisnika:

$$fn_k \in F_l, \quad k = 1,...,z \quad i \quad z < m \tag{5.4}$$

Pritom je potrebno uzeti u obzir da određene komponente mogu imati ograničenja koja strogo definiraju okolinu njihova izvođenja. Tako je prema parametrima opisanim u poglavlju 4.3 koji određuju takva ograničenja moguće definirati dvije vrste restrikcija: (*i*) komponenta treba biti pokrenuta u lokalnoj okolini ili (*ii*) komponenta treba biti pokrenuta u okolini računalnog oblaka, što se može opisati sljedećim izrazom:

$$\Phi(sc_i) \rightarrow fn_j, \quad fn_j \in F_c \quad ili \quad fn_j \in F_l \tag{5.5}$$

Ako se u prvom koraku raspoređivanja ne uspiju pokrenuti sve komponente usluge u lokalnoj okolini (ili u računalnom oblaku ako postoje takva ograničenja), potrebno je preostale komponente pokrenuti na drugim dostupnim čvorovima izvedbene okoline. U tom slučaju definiramo kašnjenje u komunikaciji između korisnika i čvora računarstva u magli kao relevantan faktor za odabir čvora na kojem će biti pokrenuta određena komponenta, budući da je upravo kašnjenje jedan od glavnih motiva primjene ovog koncepta. Zato je u drugoj iteraciji raspoređivanja cilj odabrati one čvorove koji imaju najmanje kašnjenje u komunikaciji s korisnikom, pa definiramo sljedeću ciljnu funkciju:

$$\Psi(sc_i) = min(latency^{fn_h,U}) \tag{5.6}$$

Pritom je važno naglasiti da se u ovoj iteraciji uzimaju u obzir samo dostupni čvorovi računarstva u magli koji imaju javnu IP adresu kako bi korisnici iz vlastitih lokalnih okolina mogli s njima uspostaviti komunikaciju:

$$fn_h \in F_c \cup F_p, \quad h = 1,...,s \quad and \quad s \leq m \tag{5.7}$$

Navedenim izrazima (5.1 i 5.6) definirana je ciljna funkcija koja određuje ponašanje algoritma za raspoređivanje komponenti IoT usluge u okolini računarstva u magli. Funkcijom se daje prioritet izvođenju komponenti bliže korisnicima, odnosno u okolini računarstva u magli, kako bi se povećala efikasnost u isporuci usluge određena prethodno definiranim QoS parametrima. Također, funkcijom je određeno i da je u dizajnu algoritma potrebno omogućiti prilagođeni način izvođenja raspoređivanja kako bi se mogla zadovoljiti specifična ograničenja

određenih IoT scenarija. Tako se može zaključiti da je navedena pravila opisane ciljne funkcije potrebno uzeti u obzir kao smjernice pri razvoju algoritma raspoređivanja, kako bi se u konačnici ostvario potencijal primjene računarstva u magli.

Poglavlje 6

Dizajn i implementacija algoritma za raspoređivanje IoT usluga

6.1 Algoritam za dinamičko raspoređivanje IoT usluga

Cilj raspoređivanja komponenti IoT usluga je odrediti čvor za njihovo pokretanje u raspodijeljenoj izvedbenoj okolini između računalnog oblaka i sloja računarstva u magli kako bi se u što većoj mjeri povećala efikasnost isporuke kompletne usluge. Za postizanje takvog cilja u raspodijeljenom i dinamičkom okruženju IoT usluga, odnosno za osiguravanje zadane kvalitete usluge uz podršku mobilnosti svih sudjelujućih entiteta, potrebno je razviti algoritam prema ranije definiranoj ciljnoj funkciji. Takav algoritam sastoji se od različitih procedura koje su opisane u nastavku ovog poglavlja, a konačno je u potpoglavlju 6.1.5 opisana i procedura koja omogućuje njegovo dinamičko izvođenje. Time je osigurana njegova reaktivnost na promjene u izvedbenoj okolini što omogućuje održavanje zadane razine kvalitete usluge prema ranije definiranim QoS parametrima. Redoslijed izvođenja navedenih procedura definiran je u algoritmu 1.

Algorithm 1 Redoslijed izvođenja procedura algoritma za raspoređivanje

1: *serviceComponentsToSchedule* = *service.getServiceComponents*
2: Postavljanje ograničenja za raspoređivanja komponente
3: Kategorizacija dostupnih čvorova na lokalne i javno-dostupne
4: Prva iteracija raspoređivanja - lokalni čvorovi
5: **if** (*serviceComponentsToSchedule.notEmpty*) **then**
6: Druga iteracija raspoređivanja - javno-dostupni čvorovi
7: **end if**

6.1.1 Postavljanje ograničenja na temelju parametara konteksta usluge

Algorithm 2 Postavljanje ograničenja za raspoređivanja komponente

Input: service context, *serviceComponentsToSchedule*

Output: *serviceComponentsToSchedule* with deployment restrictions

```
1:  for each (SC in serviceComponentsToSchedule) do
2:      if (SC.otherProtocolsRequired OR (S.highSecurityRequired AND SC.getType == device_interaction)) then
3:          SC.setHardDeploymentRestriction(local)
4:      else if (S.hasMobileUsers AND (SC.isStateful OR SC.hasPersistentData)) then
5:          SC.setHardDeploymentRestriction(cloud)
6:      else if (S.hasStaticUsers AND (SC.isStateful OR SC.hasPersistentData)) then
7:          SC.setSoftDeploymentRestriction(device.powerSupply==AC)
8:      else
9:          no deployment restrictions
10:     end if
11: end for
```

Prva procedura (algoritam 1, linija 2) koju treba izvesti u okviru algoritma za raspoređivanje komponenti usluge određuje ograničenja pri njihovom raspoređivanju na temelju parametara konteksta usluge (definirano algoritmom 2). Uloga ovakvih ograničenja opisana je u poglavlju 4.3, dok se u izvedbi algoritma za raspoređivanje njima definira konkretna okolina u kojoj komponenta mora (ili treba) biti pokrenuta. Izvedba procedure se može pokrenuti kada sustav za raspoređivanje primi opisnu specifikaciju usluge i njenih komponenti. Sustav na temelju parametara konteksta usluge postavlja sljedeće tri restrikcije:

- obavezno lokalno izvođenje (linije 2-3) - postavlja se za komponente usluge koje trebaju ostvariti interakciju s krajnjim uređajima (*SC.getType == device_interaction*), a pripadaju uslugama u kojima je naglašen sigurnosni rizik (*S.highSecurityRequired*). Time se omogućuje primjena određenih sigurnosnih mehanizama za zaštitu komunikacije prije slanja podataka u javnu internetsku mrežu. Drugi slučaj kada se postavlja ova restrikcija su komponente koje trebaju ostvariti komunikaciju s krajnjim uređajima nekim od ograničenih komunikacijskih protokola (*SC.otherProtocolsRequired*), budući da je doseg takvih protokola obično ograničen;
- obavezno izvođenje u računalnom oblaku (linije 4-5) - postavlja se za komponente koje su dio scenarija namijenjenog mobilnim korisnicima (*S.hasMobileUsers*), a pritom čuvaju stanje (*SC.isStateful*) ili perzistentno pohranjuju podatke (*SC.hasPersistentData*). Razlog tome je što bi učestale migracije, koje su očekivane u scenarijima namijenjenim mobilnim korisnicima, zbog kompleksnosti njihove izvedbe i trajanja u slučaju ovakvih

komponenti uzrokovale pad kvalitete njihovih performanci;

- preferirano izvođenje na čvorovima sa stalnim izvorom energije (linije 6-7) - jedina restrikcija koja ne zadaje obaveznu okolinu izvođenja, nego preporučuje stabilnije čvorove sa stalnim napajanjem ($SC.setSoftDeploymentRestriction(device.powerSupply == AC)$) postavlja se za komponente koje čuvaju stanje ($SC.isStateful$) ili perzistentno pohranjuju podatke ($SC.hasPersistentData$), a pritom su dio scenarija namijenjenog statičnim korisničkim uređajima ($S.hasStaticUsers$).

6.1.2 Kategorizacija dostupnih čvorova

Jedna od glavnih informacija na kojoj se temelji određivanje rasporeda komponenti usluge prema prethodno definiranoj ciljnoj funkciji je odnos lokacije korisnika i lokacije dostupnih čvorova računarstva u magli. Kada sustav od korisnika primi zahtjev za određenom uslugom i njegovu trenutnu mrežnu lokaciju, pokreće se postupak kategorizacije dostupnih uređaja (algoritam 3) na lokalne (linije 4-5) i javno-dostupne (linije 6-7) čvorove. Tako se po završetku izvođenja ove procedure dobivaju dva skupa koja sadrže samo one čvorove koji su trenutno dostupni za pokretanje komponenti usluge, te s kojima korisnik može uspostaviti interakciju.

Algorithm 3 Kategorizacija dostupnih čvorova na lokalne i javno-dostupne

Input: user context, device context, *availableFogDevices*
Output: two groups of *availableFogDevices* (local and public)

```
1: for each (fogDevice in availableFogDevices) do
2:     if (fogDevice.unavailable) then
3:         availableFogDevices.remove(fogDevice)
4:     else if (fogDevice.publicIP == user.publicIP) then
5:         FogDevicesLocal ⇐ fogDevice
6:     else if (fogDevice.publiclyReachableFlag) then
7:         FogDevicesPublic ⇐ fogDevice
8:     end if
9: end for
```

6.1.3 Raspoređivanje komponenti - prva iteracija

U prvoj iteraciji raspoređivanja komponenti usluge (algoritam 4) daje se prioritet njihovom pokretanju u lokalnim okolinama korisnika kako je i definirano ciljnom funkcijom u poglavlju 5.4. Pritom je obvezno uzimati u obzir i prethodno definirana ograničenja raspoređivanja koja se postavljaju ranije izvedenom procedurom opisanom algoritmom 2. U slučaju da se neka od obveznih restrikcija ne može ispuniti za određenu komponentu, kompletan postupak raspoređivanja se prekida, te se korisnik obavještava o nemogućnosti isporuke tražene usluge. Razlog ovakvog ponašanja je pretpostavka da usluga ne može ostvariti svoju funkcionalnost ako njena stroga ograničenja nisu ispunjena, pa se zato i ne pokreće.

Komponente koje nemaju postavljena stroga ograničenja raspoređivanja u ovoj se iteraciji nastoje također pokrenuti u lokalnoj okolini korisnika. Pritom prioritet imaju čvorovi koji imaju stalan izvor napajanja, ako komponenta koja se raspoređuje ima definirano ovo opcionalno ograničenje. Po završetku izvođenja ove iteracije, komponente usluge su pokrenute u lokalnoj okolini korisnika ili u računalnom oblaku. Ako su sve komponente uspješno pokrenute u ovoj iteraciji raspoređivanja, algoritam raspoređivanja je završen i korisnik se obavještava o adresi na kojoj može pristupiti traženoj usluzi.

Algorithm 4 Prva iteracija raspoređivanja (lokalni čvorovi)

Input: *serviceComponentsToSchedule*, device context
Output: deployed *serviceComponentsToSchedule*

```
1: for each (serviceComponent in serviceComponentsToSchedule) do
2:     if (serviceComponent.hardExecutionEnvironment == local) then
3:         LocalFogDevices ⇐ serviceComponent
4:     else if (serviceComponent.hardExecutionEnvironment == cloud) then
5:         Cloud ⇐ serviceComponent
6:     else
7:         try: LocalFogDevices ⇐ serviceComponent
8:     end if
9: end for
```

6.1.4 Raspoređivanje komponenti - druga iteracija

Ako u prvoj iteraciji raspoređivanja nisu sve komponente usluge pokrenute, izvodi se druga iteracija njihova raspoređivanja (algoritam 5). Kako je definirano ciljnom funkcijom, u drugoj iteraciji raspoređivanja potrebno je odabrati javno-dostupni čvor za pokretanje komponente usluge koji ima najmanje kašnjenje u komunikaciji s korisnikom. Kako bi se odredio takav čvor korisnički uređaj treba izmjeriti brzinu odziva u komunikaciji sa svakim javno-dostupnim čvorom, te potom izmjerene vrijednosti isporučiti sustavu za raspoređivanje.

U prvom koraku ove procedure sustav za raspoređivanje šalje adrese svih javno-dostupnih čvorova računarstva u magli korisniku (linija 2). Zatim korisnički uređaj treba ispitati vrijeme kašnjenja u komunikaciji sa svakim javno-dostupnim čvorom (linije 3-9), te poslati novi zahtjev sustavu za raspoređivanje koji sadrži prethodno izmjerene vrijednosti kašnjenja (linija 10). Na temelju primljenih vrijednosti sustav za raspoređivanje sortira dostupne čvorove računarstva u magli (linija 12), te potom na njima pokreće preostale komponente usluge (linije 13-17).

Po izvođenju ove procedure sve komponente usluge bi trebale biti raspoređene, budući da javno-dostupni čvorovi podrazumijevaju uređaje računarstva u magli u drugim okolinama, ali i virtualne strojeve u računalnom oblaku. U tom slučaju korisniku se šalje adresa na kojoj može pristupiti traženoj usluzi. Ipak, ako se sve komponente usluge ne pokrenu nakon izvođenja ove iteracije, korisnik se obavještava o nemogućnosti isporuke tražene usluge.

Algorithm 5 Druga iteracija raspoređivanja (javno-dostupni čvorovi)

Input: *serviceComponentsToSchedule*, device context

Output: deployed *serviceComponentsToSchedule*

1: **if** (*serviceComponentsToSchedule.isNotEmpty*) **then**
2: $Scheduler \xrightarrow{send(FogDevicesPublic)} UserAgent$
3: **procedure** MEASURE PING DURATIONS(*FogDevicesPublic*) ▷ runs on *UserAgent*
4: **for each** ($fogDevice$ in *FogDevicesPublic*) **do**
5: $t \Leftarrow fogDevice.pingDuration$
6: **if** (*pingLatencyThreshold.notDefined* OR *t<pingLatencyThreshold*) **then**
7: $addEntryToPingMap([fogDevice : t])$
8: **end if**
9: **end for**
10: $UserAgent \xrightarrow{send(PingMap)} Scheduler$
11: **end procedure**
12: *PingMap.sortOnPingDurationAscending*
13: **for each** (*serviceComponent* in *serviceComponentsToSchedule*) **do**
14: **for each** ($fogDevice$ in *PingMap*) **do**
15: try: $fogDevice \Leftarrow serviceComponent$
16: **end for**
17: **end for**
18: **end if**

6.1.5 Dinamičko raspoređivanje na temelju promjena QoS parametara

Prethodno opisani algoritam raspoređivanja osigurava arhitekturu izvođenja usluge kojom se podiže kvaliteta njene isporuke u okviru QoS parametara opisanih u poglavlju 4.1. Davanjem prioriteta izvođenju komponenti usluge u lokalnim okolinama omogućuje se lakše smanjivanje količine generiranog podatkovnog prometa u javnoj mreži. Sigurnost usluge je ovakvim pristupom također podignuta na višu razinu budući da se svi istaknuti sigurnosni rizici IoT usluga definirani u [80] lakše mogu izbjeći pokretanjem komponenti za interakciju s korisnicima u njihovoj lokalnoj okolini. Tako se na takvim komponentama mogu izvoditi mehanizmi za zaštitu privatnosti korisnika, enkripciju komunikacije, i upravljanje krajnjim uređajima što omogućuje veću razinu općenite sigurnosti cjelokupne usluge. Zato se može zaključiti da je izvođenjem prethodno opisanog algoritma raspoređivanja svakako osigurana kvaliteta usluge u okviru parametara smanjivanja prometa generiranog u javnu mrežu i sigurnosti.

Isporuka usluge u okviru preostalih relevantnih QoS parametara (kašnjenje, pouzdanost, i skalabilnost) se također podiže primjenom prethodno opisanog algoritma, ali se vrijednosti ovih

parametara mijenjaju pod utjecajem prvenstveno mrežnih, ali i drugih vanjskih faktora zbog nestabilne izvedbene okoline i mobilnosti korisnika. Zato je potrebno omogućiti dinamičko izvođenje prethodno opisanog algoritma ovisno o stanju atributa koji određuju navedene parametre, odnosno pokrenuti novo izvođenje raspoređivanja kada atributi navedenih parametara padnu ispod zadanih granica. Atributi navedenih parametara opisuju kvalitetu isporuke usluge, pa je njihove vrijednosti potrebno pratiti s korisničke strane. Kada se detektira pad kvalitete potrebno je o tome obavijestiti sustav za raspoređivanje koji onda izvodi odgovarajuće procedure kako bi detektirao i otklonio uzrok takvog ponašanja.

Iz navedenih razloga razvijen je algoritam 6 kojim se omogućuje dinamičko izvođenje raspoređivanja komponenti usluge na temelju stanja opisnih vrijednosti prethodno definiranih QoS parametara. Korisnička aplikacija može mjeriti kašnjenje i pouzdanost u komunikaciji s uslugom, dok je propusnost moguće utvrditi samo na strani sustava koji uslugu i isporučuje. Zato korisnička aplikacija mjeri kašnjenje u komunikaciji s uslugom (linije 1-4) i šalje sustavu za raspoređivanje obavijest ako je trajanje kašnjenja veće od zadane granice. Kada dobije takvu obavijest, sustav za raspoređivanje treba utvrditi razlog povećanog kašnjenja što može biti da povećana potražnja za istom uslugom uzrokuje zagušenje sustava (propusnost), ili da promjene u atributima mreže uzrokuju kašnjenje u komunikaciji između komponenti usluge i korisnika (kašnjenje). Kako bi utvrdio pravi razlog povećanog kašnjenja sustav najprije provjerava opterećenje na dostupnim čvorovima, te ako je opterećenje na nekom od njih iznad 90% pretpostavlja da je uzrok pad propusnosti sustava i pokreće replikacije komponenti pokrenutih na preopterećenom čvoru (linija 7-12). Ako nema preopterećenih čvorova, sustav može zaključiti da su uzrok povećanog kašnjenja mrežne fluktuacije pa je potrebno zaustaviti sve pokrenute komponente usluge i ponovo pokrenuti izvođenje algoritma raspoređivanja (linije 14-15).

Pouzdanost se također može mjeriti u korisničkoj aplikaciji prema formuli iz poglavlja 4.1. Pad kvalitete u okviru ovog parametra se detektira ako aplikacija ne primi jedan odgovor, ili određeni postotak odgovora u jedinici vremena, od usluge s kojom je u interakciji (linija 18). Uzrok smanjene pouzdanosti tako može biti posljedica ispada neke od komponenti usluge ili migracije korisnika između različitih mrežnih okolina. Ipak, u slučaju pada pouzdanosti sustav ne treba utvrditi konkretan razlog između navedena dva uzroka, budući da je u oba slučaja potrebno ponovno izvršiti raspoređivanje komponenti usluge. Tako sustav u slučaju primljene obavijesti od aplikacije o padu pouzdanosti ispod zadane granice, pokreće ponovno raspoređivanje komponenti usluge te tako prilagođava njenu isporuku novim uvjetima u izvedbenoj okolini, odnosno novoj lokaciji korisnika (linije 19-20).

Ovakvim procedurama omogućena je reaktivnost izvođenja raspoređivanja komponenti usluge na temelju vrijednosti koje opisuju stanje relevantnih QoS parametara. Tako je navedenim algoritmom osigurano dinamičko raspoređivanje IoT usluga u okolini računarstva u magli s ciljem osiguravanja zadane kvalitete usluge.

Algorithm 6 Dinamičko raspoređivanje na temelju QoS parametara

Input: service context, user context, QoS constraints
Output: replication or re-scheduling

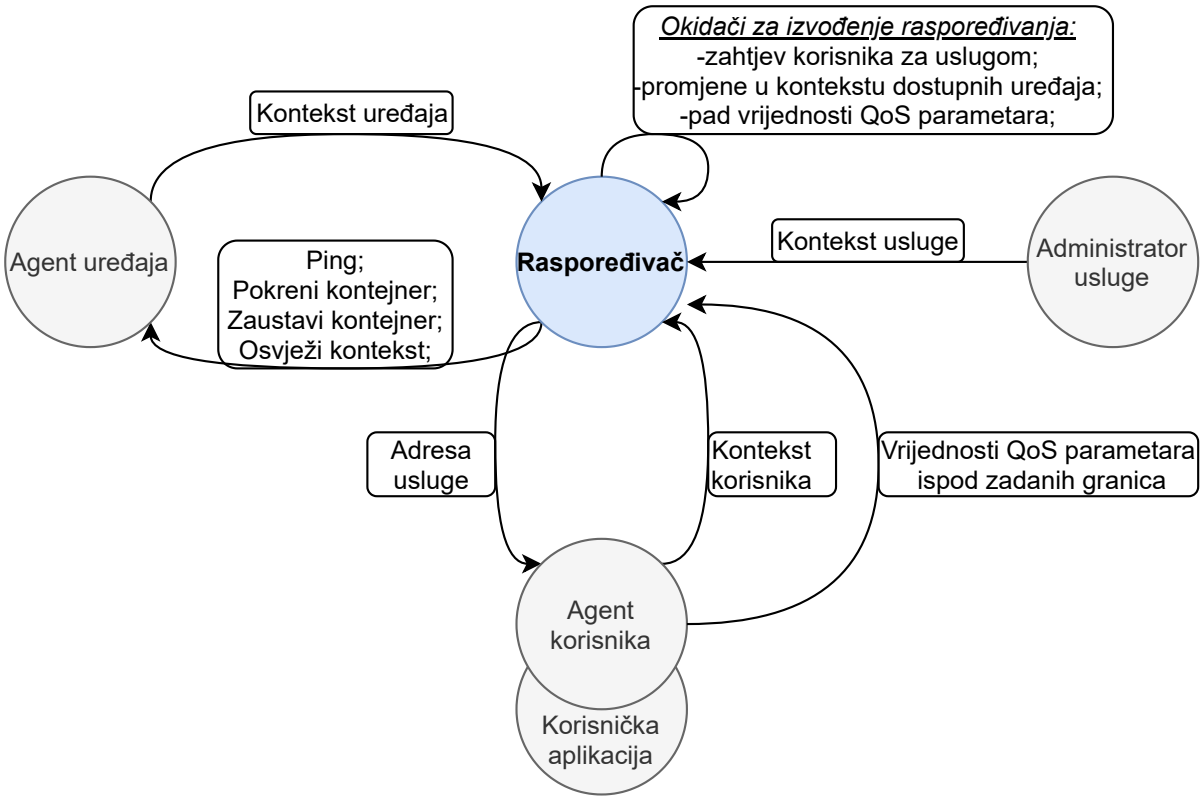
for each *sentRequest* **do** ▷ runs on $UserAgent$
  if ($response.isReceived$) **then**
    $latency = t_{response} - t_{request}$
    if ($latency >$ threshold) **then**
      $UserAgent \xrightarrow{send(latency-downgrade)} Scheduler$
      $throughput \Leftarrow false$ ▷ runs on the *Scheduler*
      for each $fogNode$ in $fogNodesExecutingServiceComponents$ **do**
        if $fogNode.getCurrentCpuLoad > 90\%$ **then**
          $fogNode.replicateServiceComponents$
          $throughput \Leftarrow true$
        end if
      end for
      if $throughput.isFalse$ **then**
        $previouslyScheduledServiceComponents.stop$
        $schedulingAlgorithm.execute$
      end if
    end if
  else if ($response.notReceived$ AND $reliability <$ threshold) **then**
    $UserAgent \xrightarrow{send(reliability-downgrade)} Scheduler$
    $schedulingAlgorithm.execute$
  end if
end for

6.2 Implementacija sustava za raspoređivanje

Zbog ograničenja primjene sustava za orkestraciju kontejneriziranih usluga Kubernetes opisanih u poglavlju 2.2.2 i kompleksnosti nadogradnje drugih, manje popularnih, rješenja za raspoređivanje usluga u računarstvu u magli, za testiranje predstavljenog algoritma dizajniran je i implementiran novi sustav za raspoređivanje. Glavni cilj u njegovom dizajnu i implementaciji je bio izgraditi sustav za raspoređivanje komponenti IoT usluga čiji se rad temelji na primjeni opisanog algoritma. Pritom je njegovu logiku trebalo ostvariti uzimajući u obzir navedene kontekstne informacije, te specificirane kategorije i ograničenja u raspoređivanju određenih IoT scenarija. Kako bi se implementirao takav sustav razvijene su tri različite komponente:

- Raspoređivač,
- Agent uređaja i
- Agent korisnika.

Apstraktni dijagram njihovog odnosa i interakcije prikazan je na slici 6.1. U nastavku ovog poglavlja opisana je uloga svakog od uključenih entiteta, te tehnička izvedba potrebnih funkcionalnosti u ovakvom raspodijeljenom sustavu.



Slika 6.1: Model entiteta uključenih u rad sustava za raspoređivanje i njihove interakcije.

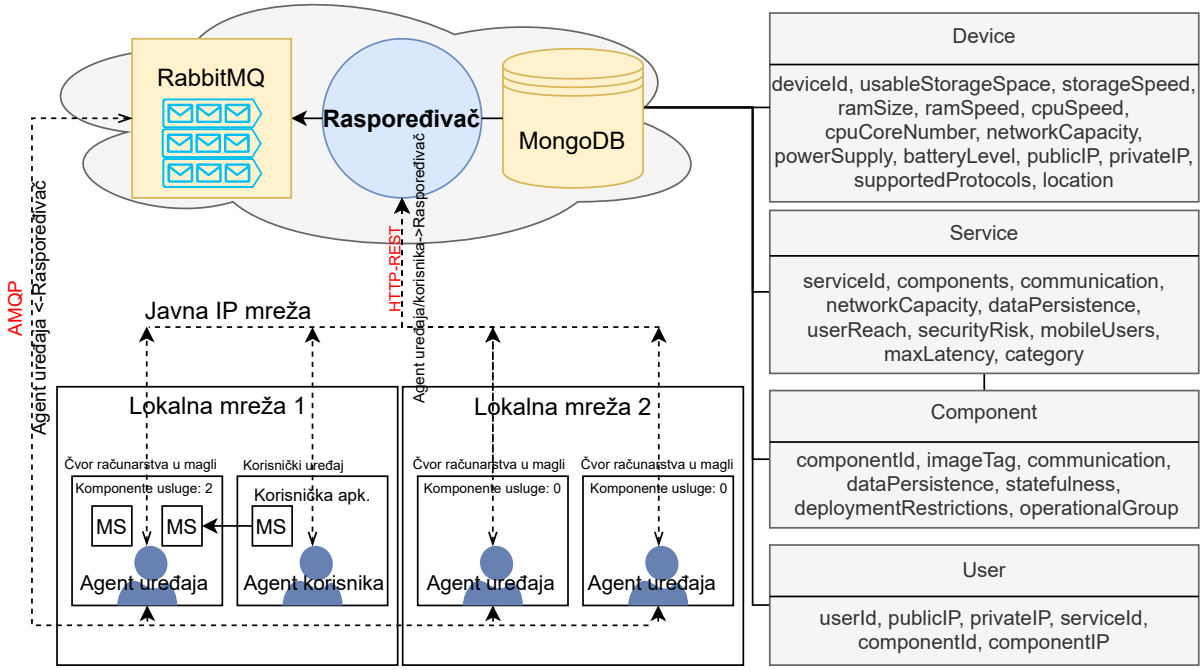
6.2.1 Raspoređivač

Raspoređivač je centralna komponenta razvijenog sustava koja na temelju dostupnih kontekstnih informacija o svim uključenim entitetima izvodi raspoređivanje komponenti IoT usluge prema ranije opisanom algoritmu. Tehnički model cjelokupne izvedbene okoline prikazana je na slici 6.2. Tako je vidljivo da je Raspoređivač komponenta koja ima javnu IP adresu na kojoj je izloženo REST sučelje kroz koje može primati ulazne HTTP-zahtjeve od ostalih entiteta u sustavu (Agenta uređaja i Agenta korisnika). Također, prikazano je i kako Raspoređivač za ostvarivanje vlastite funkcionalnosti koristi dva vanjska sustava: bazu podataka MongoDB i broker za komunikaciju porukama RabbitMQ. Zato je prije pokretanja ove komponente potrebno u konfiguracijskoj datoteci Raspoređivača unijeti parametre za uspostavljanje veze s navedenim sustavima (slika 6.3).

Baza podataka MongoDB koristi se za pohranu svih podataka o kontekstu registriranih uređaja, korisnika, te usluga, prema podatkovnom modelu na slici 6.2. Raspoređivač prima informacije o kontekstu uređaja i korisnika od njihovih agenata, dok kontekst usluge definira administrator usluge (slika 6.1). Dodatna funkcionalnost Raspoređivača koju također ostvaruje pomoću baze podataka MongoDB je otkrivanje adresa pokrenutih komponenti (engl. *service discovery*). Tako se u bazi pohranjuju informacije o lokaciji izvođenja svih prethodno raspoređenih komponenti, kako bi se na zahtjev njihove adrese mogle isporučiti drugim komponentama

kojima su one potrebne za ostvarivanje njihove funkcionalnosti.

Broker za razmjenu poruka RabbitMQ je uključen u arhitekturu sustava kako bi se omogućilo uključivanje uređaja iz lokalnih mreža, koji nemaju statičke javne IP adrese, u izvedbenu okolinu. Primjenom Kubernetesa ovakvi uređaji nisu mogli biti uključeni u izvedbenu okolinu bez dodatnih procedura opisanih u poglavlju 2.2.2, što je upravo i bio glavni razlog razvoja novog sustava za raspoređivanje. Raspoređivač tako po registraciji svakog uređaja stvara red u RabbitMQ brokeru na koji se Agent registriranog uređaja treba pretplatiti kako bi postao dostupan Raspoređivaču. Tako je posredstvom brokera RabbitMQ ostvareno slanje zahtjeva od Raspoređivača prema Agentima uređaja, čime je riješen problem njihove nedostupnosti zbog dinamičke adrese koja obično nije javno dostupna. Dodatno treba naglasiti da je i sva komunikacija od Raspoređivača prema Agentima uređaja sinkrona, kako bi sustav potvrdio da su zadane naredbe uspješno izvršene, odnosno prepoznao ispad čvora koji ne vrati odgovor.

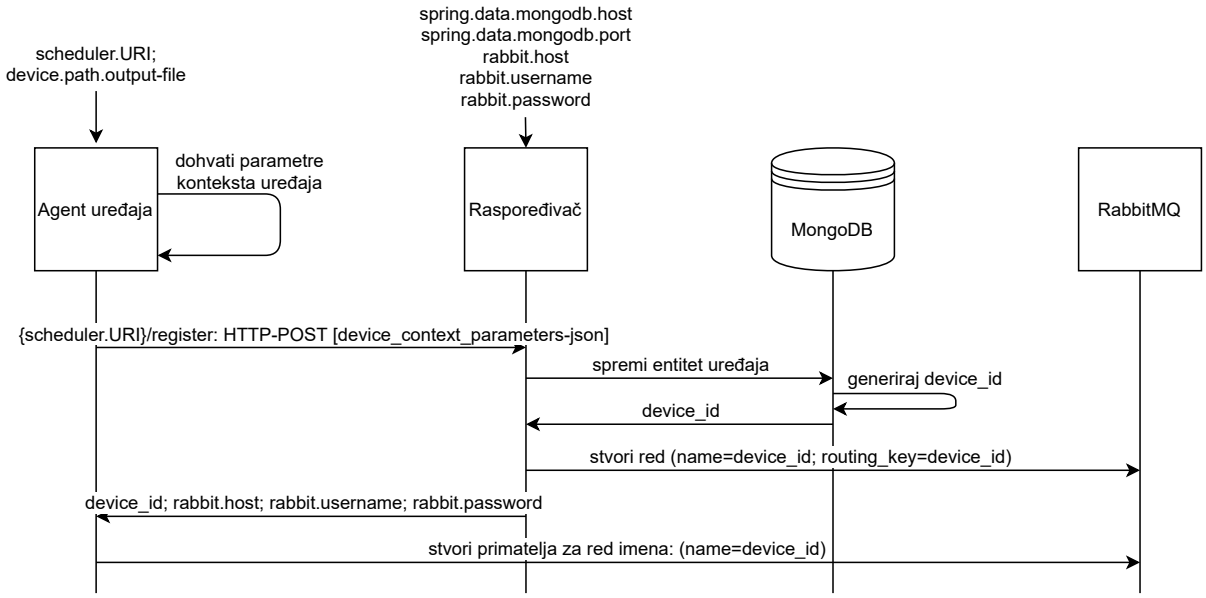


Slika 6.2: Arhitektura sustava i korišteni podatkovni model opisanih konteksta.

6.2.2 Agent uređaja

Svaki uređaj koji se želi pridružiti izvedbenoj okolini kao čvor računarstva u magli, najprije pokreće komponentu Agenta uređaja kojom se ostvaruje sva interakcija s Raspoređivačem i izvršavaju naredbe navedene na slici 6.1. Tako se pokretanjem Agenta uređaja šalje inicijalni zahtjev za registracijom čvora Raspoređivaču, koji sadrži sve informacije o kontekstu uređaja koji se registrira (slika 6.3). Kao odgovor na zahtjev za registracijom, Agent uređaja prima podatke potrebne za uspostavljanje veze s RabbitMQ brokerom i generirani identifikator ure-

đaja (*device_id*). Primljeni identifikator je ujedno i adresa reda u RabbitMQ brokeru na koji se Agent uređaja treba pretplatiti kako bi postao dostupan Raspoređivaču, odnosno kako bi mogao primati naredbe koje treba izvršiti (*ping* zahtjev, zahtjevi za pokretanjem ili zaustavljanjem kontejnera, te zahtjev za osvježavanjem parametara konteksta uređaja). Zahtjevom za pokretanjem ili zaustavljanjem kontejnera Raspoređivač obavještava Agenta uređaja o komponenti usluge koju treba pokrenuti, a koja zato treba biti dostupna u obliku OCI (Open Container Initiative) slike (engl. *image*) tj. Docker slike koja opisuje konfiguraciju kontejnera. Primjenom kontejnera omogućena je podrška različitih računalnih platformi što je važno zbog heterogenosti okoline kakvu podrazumijeva računarstva u magli. Također, na ovaj način je olakšana i prenosivost komponenti, budući da se OCI slika tražene komponente može dohvatiti iz javno dostupnih repozitorija ako je ciljani uređaj nema u lokalnoj memoriji.



Slika 6.3: Sekvencijski dijagram postupka registracije čvora računarstva u magli.

6.2.3 Agent korisnika

Agent korisnika je komponenta koja se pokreće na korisničkom uređaju, povrh korisničke aplikacije koja ostvaruje interakciju sa samom uslugom. Uloga ove komponente je da najprije Raspoređivaču pošalje inicijalni zahtjev koji sadrži informacije o kontekstu korisnika i specificira traženu uslugu. Raspoređivač po primanju ovakvog zahtjeva pokreće algoritam raspoređivanja te kao odgovor Agentu korisnika vraća adresu na kojoj korisnička aplikacija može pristupiti traženoj usluzi. Druga uloga Agenta korisnika je da u interakciji s korisničkom aplikacijom kontrolira kvalitetu isporuke usluge. Tako u slučaju pada kvalitete isporuke ispod definiranih granica, Agent korisnika o tome obavještava Raspoređivača slanjem odgovarajućeg HTTP-zahtjeva kako bi se detektirao i otklonio uzrok ako je to moguće. Dodatno je važno naglasiti

da se komunikacija između korisničke aplikacije i usluge po izvođenju raspoređivanja izvodi direktno, bez posrednika, što se može vidjeti u donjem lijevom dijelu slike 6.2.

Poglavlje 7

Verifikacija učinkovitosti algoritma raspoređivanja

7.1 Postupak verifikacije učinkovitosti raspoređivanja

Verifikacija izvedbene učinkovitosti algoritma za raspoređivanje IoT usluga u raspodijeljenoj izvedbenoj okolini računarstva u magli treba potvrditi ciljeve koji se žele postići njegovom primjenom. Parametri za evaluaciju kvalitete u isporuci IoT usluga, opisani u poglavlju 4.1, dobri su pokazatelji učinkovitosti različitih arhitektura za isporuku konkretnog IoT scenarija. Na osnovu njihovih vrijednosti može se usporediti efikasnost različitih arhitektura, odnosno rasporeda komponenti usluge kojima se isporučuje njena funkcionalnost ciljanim korisnicima. Ipak, važno je naglasiti kako će ovisno o kategoriji usluge njenu efikasnost određivati različiti podskup navedenih parametara, što je također prethodno definirano u poglavlju 4.2. Tako će se verifikacija izvedbene učinkovitosti algoritma provoditi na temelju parametara kvalitete usluge relevantnih za specifični slučaj primjene, koji trebaju pokazati veću efikasnost u isporuci usluge od rasporeda kojim bi se ona isporučivala bez primjene razvijenog algoritma.

Postupak verifikacije izvedbene učinkovitosti algoritma definiramo sljedećim koracima:

1. formalni opis konteksta usluge za odabrani slučaj primjene,
2. definicija kategorije i pripadajućih parametara koji određuju kvalitetu usluge u specifičnom slučaju primjene,
3. primjena algoritma koji se verificira na konkretan slučaj primjene,
4. usporedba ostvarene učinkovitosti isporuke usluge s arhitekturom koja uključuje samo računalni oblak,
5. usporedba ostvarene učinkovitosti isporuke usluge s postojećim algoritmima raspoređivanja usluga u okolini računarstva u magli.

U prvom koraku definiranog postupka potrebno je opisati specifični slučaj primjene prema specificiranom formalnom modelu usluge kako bi se u drugom koraku mogli odrediti parametri

kvalitete usluge koji služe za evaluaciju učinkovitosti njene isporuke u konkretnom IoT scenariju. U trećem koraku se na odabrani IoT scenarij primjenjuje razvijeni algoritam koji se verificira, kako bi se dobili rezultati njegove učinkovitosti u isporuci IoT usluge prema relevantnim QoS parametrima određenim u prethodnom koraku. Dobiveni rezultati u četvrtom i petom koraku trebaju potvrditi dva glavna cilja kako bi se verificirala učinkovitost razvijenog algoritma.

Prvi cilj odnosi se na potvrdu učinkovitosti primjene koncepta računarstva u magli na konkretan IoT scenarij. Tako je u okviru četvrtog koraka postupka verifikacije učinkovitosti algoritma potrebno dokazati da se njegovom primjenom postiže efikasnija isporuka usluge na temelju relevantnih QoS parametara, nego arhitekturom u kojoj se obrada izvodi isključivo u računalnom oblaku. Ovaj korak nije potrebno izvoditi ako se u zadnjem koraku postupka verifikacije efikasnost uspoređuje s postojećim algoritmima koji u obzir uzimaju i raspored u kojem su sve komponente raspoređene u računalnom oblaku.

Drugi cilj kojim se potvrđuje učinkovitost razvijenog algoritma raspoređivanja je potvrditi njegovu veću učinkovitost u odnosu na postojeće algoritme. Za upravljanje raspodijeljenom obradom kontejneriziranih usluga najčešće se koristi Kubernetes, koji u pretpostavljenoj varijanti primjenjuje jednostavan algoritam zasnovan na dostupnim resursima čvorova za pokretanje usluga kako je opisano u poglavlju 2.2.2. Zato je prvenstveno potrebno dokazati učinkovitost razvijenog algoritma u odnosu na Kubernetesov pretpostavljeni način raspoređivanja. Ipak, ako postoji algoritam raspoređivanja koji prema prethodno određenim parametrima kvalitete usluge može odrediti efikasniji raspored u specifičnom slučaju primjene, verifikaciju učinkovitost je potrebno dokazati u odnosu na takav algoritam. Ako se na odabranom slučaju primjene dokaže veća učinkovitost razvijenog algoritma od postojećih pristupa, može se zaključiti da se ovim postupkom verifikacije potvrdila relevantnost razvijenog algoritma.

7.2 Verifikacija učinkovitosti na odabranim uslugama

Većina parametara za evaluaciju kvalitete usluge je ovisna o mrežnoj komunikaciji, pa je za simulaciju izvedbene okoline i mrežnih uvjeta u njoj bilo potrebno odabrati alat koji nudi mogućnost upravljanja parametrima mreže. IMUNES [83] (*Integrated Multiprotocol Network Emulator/Simulator*) je simulator TCP/IP mreže koji omogućuje simulacije različitih mrežnih okruženja u kojima su virtualni čvorovi povezani s drugim virtualnim čvorovima, ili s fizičkim mrežnim sučeljima, pomoću virtualnih mrežnih veza [84, 85, 86]. Takve virtualne mrežne veze između čvorova u simulaciji imaju različite konfiguracijske parametre, uključujući kašnjenje i propusnost, kojima se dinamički može upravljati. Zato je za izvedbu simulacija kojima bi se proveo postupak verifikacije razvijenog algoritma odabran upravo ovaj simulator.

Kako bi se razvijeni sustav za raspoređivanje mogao izvoditi u IMUNES-u, trebalo je naj-

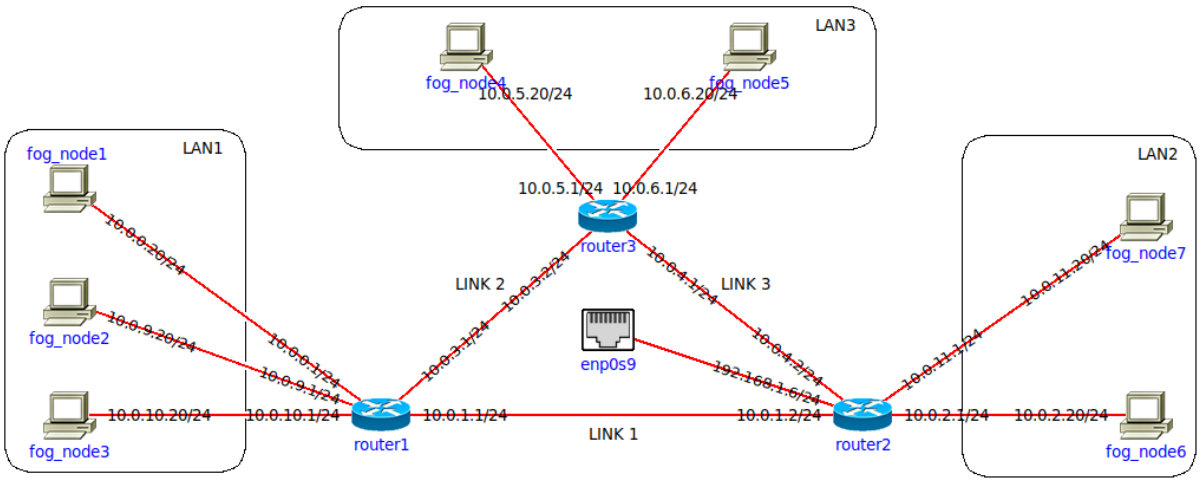
prije prilagoditi njegovu implementaciju tako da se njime raspoređuju komponente u obliku paketa tehnologije koja se koristi, umjesto komponenti koje su u OCI slikama tj. Docker slikama. U ovom slučaju su to *jar* datoteke jer su komponente razvijane u programskom jeziku Java. Razlog korištenja paketa tehnologije je što je svaki čvor u IMUNES-u zapravo Docker kontejner, a proces pokretanja drugog Docker kontejnera unutar postojećeg se ne preporučuje. Ipak, važno je naglasiti da je funkcionalnost originalnog sustava za raspoređivanje testirana u stvarnom okruženju kako bi se najprije potvrdila njegova funkcionalnost u radu s Docker kontejnerima. Prilagodba sustava za raspoređivanje u simulacijskoj okolini ostvarena je implementacijom prilagođenih dijelova programskog koda kojima se izvodio proces raspoređivanja i pokretanja komponenti usluge. Tako je omogućeno izvođenje iste logike raspoređivanja kao i u originalnoj verziji, ali se umjesto rada s OCI slikama i pokretanjem kontejnera funkcionalnost izvodila s *jar* datotekama i pokretanjem Javinog virtualnog stroja. U takvoj izvedbi Agent uređaja po primanju zahtjeva za pokretanjem određene komponente najprije dohvaća njenu *jar* datoteku iz repozitorija u privatnom računalnom oblaku (umjesto centralnog Dockerovog repozitorija), te je potom pokreće na virtualnom uređaju u simuliranom okruženju.

Kako bi se moglo verificirati ponašanje algoritma, te njegova učinkovitost dizajnirano je simulirano okruženje prikazano na slici 7.1. Tako se može vidjeti da ono sadrži tri različite lokalne mreže (LAN, engl. *Local Area Network*) koje su međusobno povezane kroz tri usmjeritelja, dok je cjelokupno simulirano okruženje povezano s javnom mrežom kroz vanjsko sučelje (*enp0s9*). Svaki virtualni čvor računarstva u magli (*fog_node*) je Docker kontejner koji sadrži Javu, što omogućuje pokretanje razvijenih komponenti sustava za raspoređivanje (Agenta uređaja i Agenta korisnika), ali i *jar* komponenti IoT usluga koje se raspoređuju u Javinom virtualnom stroju. Raspoređivač je zajedno s bazom podataka MongoDB i brokerom RabbitMQ pokrenut na poslužitelju u javnoj internetskoj mreži kako bi se simulirao računalni oblak, pa je čvorovima unutar simulacije dostupan kroz sučelje prema vanjskoj mreži.

Ideja dizajniranog testnog okruženja je simulirati tri različite lokalne mreže koje se mogu nalaziti bilo gdje u stvarnoj globalnoj mreži, zbog čega bi na kvalitetu komunikacije između njih mogli utjecati različiti faktori koji bi uzrokovali promjene u okviru vrijednosti promatranih parametara kvalitete usluge. Iako bi u realnom slučaju između prikazanih usmjeritelja postojala udaljenost od više skokova (engl. *multi-hop*), u simuliranom okruženju je dovoljna jedna virtualna veza kako bi se simulirali različiti mrežni uvjeti upravljanjem njenih konfiguracijskih parametara, prvenstveno kašnjenja i propusnosti.

Za izvođenje petog koraka definiranog postupka verifikacije implementirane su sljedeće verzije Raspoređivača:

- verzija koja primjenjuje dinamički algoritam iz ovog rada (Dynamic),
- verzija koja primjenjuje Kubernetesov osnovni algoritam (K8s) i
- verzija koja primjenjuje algoritam NAS iz [43].



Slika 7.1: Mreža u simuliranom okruženju u IMUNES-u.

Način rada navedenih algoritama opisan je u poglavlju 2.2.2. Tako se u verziji Raspoređivača koji primjenjuje Kubernetesov algoritam, pri raspoređivanju u obzir uzimaju vrijednosti parametara konteksta uređaja koje opisuju njegove dostupne procesorske i memorijske resurse. Zato se u ovoj verziji za pokretanje komponente odabire čvor koji u trenutku raspoređivanja ima najveću količinu navedenih resursa. Također, nakon inicijalnog raspoređivanja komponenti usluge, Kubernetes ne pokreće novi postupak raspoređivanja osim u slučaju ispada neke od komponenti. Zato se u ovoj verziji ne pokreće novo raspoređivanje komponenti usluge neovisno o parametrima kvalitete mreže koji mogu uzrokovati pad kvalitete u isporuci usluge.

NAS je implementiran povrh Kubernetesa, te osim navedenih resursa, u obzir pri odabiru čvora za pokretanje komponente uzima i kašnjenje u komunikaciji između korisnika koji šalje zahtjev i dostupnih čvorova za pokretanje komponenti usluge. Zato je verzija koja implementira ovaj algoritam ostvarena tako da se konačan odabir čvora za pokretanje komponenti donosi na temelju trajanja kašnjenja u komunikaciji korisnika i dostupnih čvorova, kao što je slučaj u drugoj iteraciji razvijenog algoritma (algoritam 5). Kako se NAS izvodi u Kubernetesu, ni ova verzija raspoređivača nema mogućnost izvođenja raspoređivanja prema padu kvalitete u mreži, već samo u slučaju ispada neke od komponenti.

Svaka od simulacija opisanih u nastavku ovog poglavlja je najprije izvedena primjenjujući dinamički algoritam predstavljen u poglavlju 6.1, a zatim prilagođene verzije Raspoređivača koje implementiraju postojeće algoritme raspoređivanja. Prema opisu prilagođenih verzija može se pretpostaviti da je osnovni nedostatak postojećih pristupa u odnosu na algoritam predstavljen u ovoj disertaciji, nemogućnost adaptiranja rasporeda promjenama u izvedbenoj okolini koje uzrokuju pad kvalitete u isporuci usluge. Zato je primarni cilj simulacija u nastavku bio potvrditi dinamičko izvođenje razvijenog algoritma kojim se adaptira raspored komponenti ovisno o stanju izvedbene okoline, te usporediti njegovu učinkovitost s navedenim algoritmima.

Sve simulacije izvedene su na virtualnom stroju pokrenutom na računalu MacBook Pro s

2.5 GHz Dual-Core Intel Core i5 procesorom i 16 GB RAM memorije (od kojih je 8 GB bilo na namijenjeno virtualnom stroju) na kojem je instaliran operacijski sustav UBUNTU 20.04. Simulirano okruženje računalnog oblaka u kojem je bio pokrenut Raspoređivač, RabbitMQ i MongoDB izvedeno je također na virtualnom stroju s operacijskim sustavom UBUNTU 18.04 sa 16 GB RAM memorije, a pokrenutom na poslužiteljskom računalu s 2.40 GHz Intel Xeon octa-core procesorom (model E5-2630 v3).

7.2.1 Scenarij usluge strujanja podataka

Prvi scenarij je dizajniran kako bi se potvrdilo očekivano ponašanje razvijenog algoritma u slučaju promjena stanja izvedbene okoline, odnosno njegova mogućnost adaptacije rasporeda u slučajevima kada kvaliteta isporuke usluge padne ispod zadanih granica. Strujanje podataka je tipična IoT usluga koju u nekom obliku sadrži većina kompleksnijih IoT scenarija ili se izvodi samostalno, a podrazumijeva slanje i primanje različitih podatkovnih tokova. U okviru slučajeva primjene u Internetu stvari, krajnji senzorski uređaji šalju podatkovne tokove vlastitih očitanja prema čvorovima s više dostupnih memorijskih resursa, budući da oni sami obično nemaju dovoljnu količinu memorije za lokalnu pohranu. Zato je za efikasnu isporuku usluga strujanja podataka važno osigurati dostatnu propusnost mrežne komunikacije kako bi se omogućio prijenos podatkovnog toka od krajnjih uređaja prema čvorovima na kojima se podaci pohranjuju.

Za izvođenje ovog scenarija razvijena je jednostavna korisnička aplikacija koja kroz otvorenu vezu (engl. *socket*) prema drugom čvoru generira podatkovni tok zadanom brzinom. Također, implementirana je i usluga koja u ovom slučaju sadrži samo jednu komponentu koja prima podatkovni tok te šalje pošiljatelju obavijest o broju bitova koji su primljeni u svakoj sekundi (bps). Tako je iznimno u ovom scenariju korisničkoj aplikaciji omogućena informacija o propusnosti usluge, na koju u simulaciji utječu samo mrežni uvjeti budući da usluga koja prima ulazni tok može prihvatiti neograničenu količinu podataka. Zato u ovom iznimnom slučaju, korisnička aplikacija ne treba samostalno izvoditi mjerenje kašnjenja kako je definirano u razvijenom algoritmu, nego tu informaciju prima od usluge, i prosljeđuje je Agentu korisnika. Agent korisnika treba onda na temelju unaprijed zadane razine kvalitete usluge, koju u ovom slučaju određuje pouzdanost, odrediti je li njena razina dostatna ili treba ponovo pokrenuti raspoređivanje komponenti usluge kako bi se probala postići isporuka u zadanim granicama kvalitete.

Provjera izvođenja dinamičkog raspoređivanja

Prvim testom trebalo je verificirati da se raspoređivanje komponenti usluge pokreće kada razina kvalitete padne ispod zadane granice, te da se za izvođenje njenih komponenti odabire čvor

koji u tom trenutku može isporučiti najveću kvalitetu usluge. U simulaciji su bili pokrenuti Agenti čvora na virtualnim čvorovima računarstva u magli 5 i 6 (*fog_node5* i *fog_node6* na slici 7.1), Raspoređivač na poslužiteljskom računalu u javnoj mreži, te Agent korisnika na čvoru *fog_node3*.

U trenutku *t1* Agent korisnika na čvoru (*fog_node3*) šalje Raspoređivaču zahtjev za registracijom kojim se pokreće raspoređivanje tražene usluge. Kako nema dostupnih čvorova za pokretanje usluge u lokalnoj mreži korisnika, Raspoređivač vraća Agentu korisnika odgovor u kojem mu daje popis adresa javno-dostupnih čvorova (*fog_node5* i *fog_node6*) za koje je potrebno utvrditi trajanje kašnjenja. Agent korisnika ispituje trajanje kašnjenja za navedene uređaje, te ponovno Raspoređivaču šalje zahtjev koji sadrži tražene vrijednosti na temelju kojih on može odlučiti koji je od dostupnih čvorova najprikladniji za pokretanje usluge. U tablici 7.1 su prikazane izmjerene vrijednosti kašnjenja koje je Agent korisnika poslao Raspoređivaču za vrijeme izvođenja algoritma raspoređivanja u ovom testu. Prema tim vrijednostima Raspoređivač bi za pokretanje usluge koja prima podatkovni tok odabrao čvor koji u konkretnom trenutku ima manju vrijednost kašnjenja i zadovoljava ostale minimalne QoS zahtjeve (*fog_node5* ili *fog_node6*). Korisnička aplikacija generirala je podatkovni tok konstantnom brzinom od 66 Mbps, a granica minimalne prihvatljive propusnosti bila je postavljena na 30 Mbps. Zato je za pokretanje ponovnog raspoređivanja trebalo smanjiti propusnost ispod zadane granice na virtualnoj vezi između čvora na kojem je pokrenuta korisnička aplikacija (*fog_node3*) i čvora računarstva u magli na kojem je u promatranom trenutku pokrenuta usluga primanja podatkovnog toka (*fog_node5* ili *fog_node6*), što je prikazano u tablici 7.2.

Na slici 7.2 su prikazani rezultati izvedenih simulacija koji pokazuju ostvarenu propusnost primjenom različitih algoritama raspoređivanja u opisanom testu. Podebljane linije prikazuju rezultate razvijenog dinamičkog algoritma raspoređivanja, a različite boje označavaju čvor na kojem je komponenta usluge bila pokrenuta u određenom trenutku. Tako je nakon inicijalnog izvođenja raspoređivanja po registraciji korisnika u trenutku *t1* usluga najprije pokrenuta na čvoru *fog_node5*, te je propusnost jednaka brzini generiranja toka (66 Mbps) do trenutka *t2*. Tada propusnost pada ispod zadane granice (30 Mbps), budući da je smanjena propusnost na vezi *link2* prema čvoru *fog_node5* (kako je prikazano u tablici 7.2), što uzrokuje ponovno pokretanje algoritma raspoređivanja kojim se odabire čvor *fog_node6* za pokretanje usluge zbog manjeg kašnjenja u trenutku *t2* (što je vidljivo u tablici 7.1). Isti se postupak ponavlja i u ostalim trenucima (*t3*, *t4*, *t5*), pa se može potvrditi da dinamički algoritam u trenucima pada propusnosti ispod zadane granice doista pokreće ponovno izvođenje raspoređivanja kojim se usluga migrira na prikladniji čvor, odnosno onaj čvor koji u konkretnom trenutku ima manje trajanje kašnjenja.

Rezultati primjene druga dva algoritma prikazani su linijom crvene boje. Može se primijetiti da je u oba slučaja rezultat isti, a razlog tome je što su oba pristupa zapravo implementacije komponente kube-scheduler koja u Kubernetesu izvodi raspoređivanje. Zato se nakon inicijalno

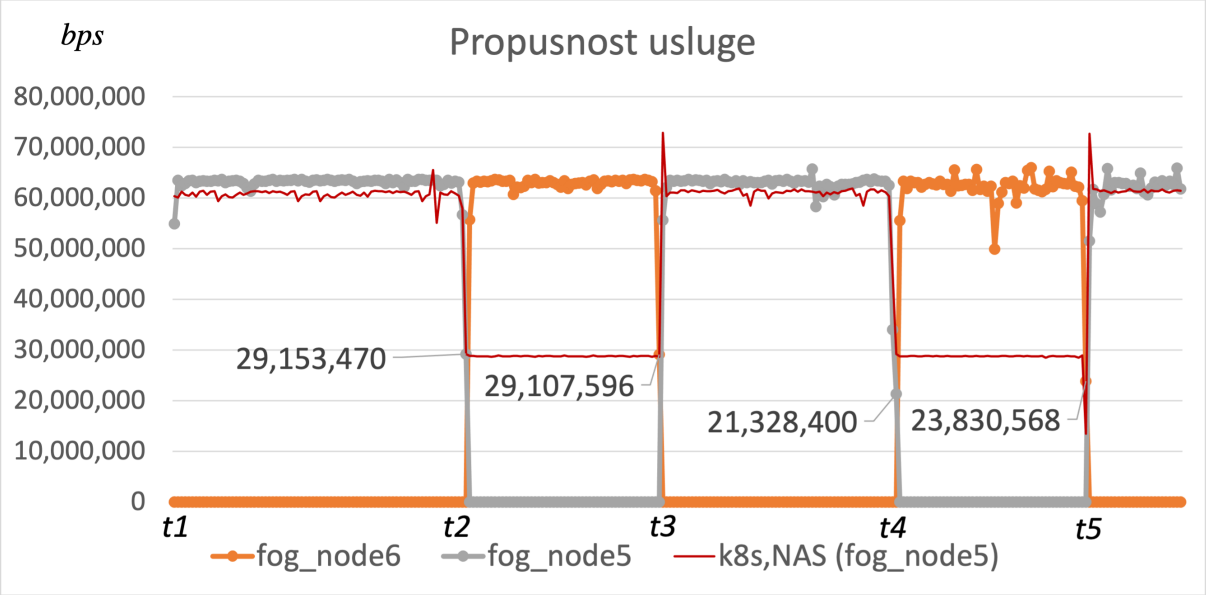
odabranog čvora za pokretanje tražene usluge, novo raspoređivanje ne pokreće neovisno o promjenama vrijednosti parametara kvalitete usluge. Tako se u trenucima kada bi se propusnost na vezi između korisničkog uređaja i čvora na kojem je pokrenuta usluga (*fog_node5*) smanjila, u ovom slučaju nije izvodilo novo raspoređivanje, pa je propusnost usluge konstantno odgovarala postavljenoj vrijednosti propusnosti na virtualnoj vezi *link2* prema tablici 7.2.

Tablica 7.1: Kašnjenje u komunikaciji između Agenta korisnika (*fog_node3*) i dostupnih čvorova (ms).

	t1	***t2***	***t3***	***t4***	***t5***
fog_node5	**544**	1393	**60**	1215	**71**
fog_node6	1568	**83**	1538	**58**	922

Tablica 7.2: Propusnost na virtualnim vezama link1 i link2 (Mbps).

	t1	***t2***	***t3***	***t4***	***t5***
LINK 1 (fog_node6)	100	100	30	100	30
LINK 2 (fog_node5)	100	30	100	30	100

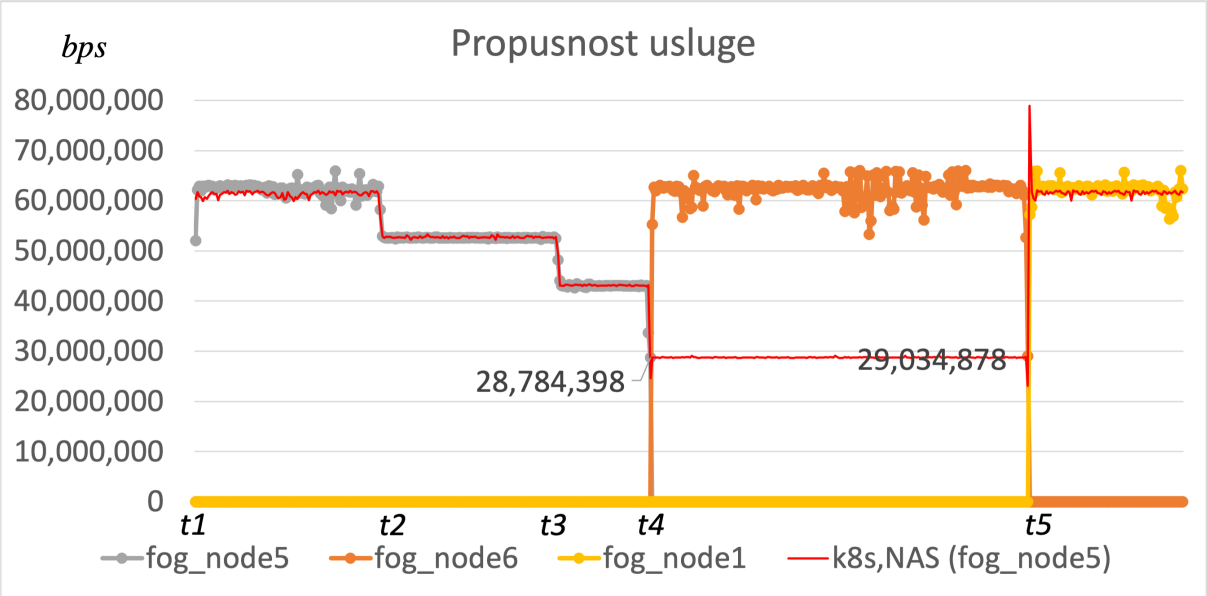


Slika 7.2: Propusnost usluge pri konstantnoj brzini generiranja podataka od 66 Mbps (test 1).

Provjera davanja prioriteta lokalnim čvorovima

U drugom testu je uz entitete iz prethodne simulacije dodatno bio pokrenut i Agent uređaja na čvoru *fog_node1*. Cilj u drugom testu je bio potvrditi još jedan važan korak u izvođenju pred-

stavljenog algoritma, a to je da se prioritet u odabiru čvora za pokretanje usluge daje uređajima u lokalnoj mreži korisnika.



Slika 7.3: Propusnost usluge pri konstantnoj brzini generiranja podataka od 66 Mbps (test 2).

Na slici 7.3 su prikazani rezultati izvođenja simulacije, a podebljana linija označava postignutu propusnost primjenom dinamičkog algoritma raspoređivanja. Najprije se vidi da se novo raspoređivanje ne pokreće kada se dogodi pad kvalitete usluge (*t2* i *t3*), sve dok promatrana vrijednost relevantnog parametra kvalitete ne padne ispod zadana minimalne granice (*t4* i *t5*). U trenutku *t4* propusnost pada ispod zadane granice pa se kao i u prethodnom testu usluga migrira na prikladniji čvor prema tablici 7.3, odnosno s čvora *fog_node5* na čvor *fog_node6*. Zatim se između trenutaka *t4* i *t5* pokreće Agent uređaja na čvoru *fog_node1* koji se nalazi u lokalnoj mreži korisnika. Zato se pri raspoređivanju u trenutku *t5* usluga pokreće na njemu, bez zahtijevanja ispitivanja kašnjenja u komunikaciji između Agenta korisnika i drugih dostupnih uređaja. Time je potvrđeno očekivano ponašanje predstavljenog dinamičkog algoritma raspoređivanja.

Crvena linija na slici 7.3 prikazuje rezultate druga dva algoritma raspoređivanja koja iz istog razloga kao i u prethodnom testu imaju jednako ponašanje. Tako se ni u ovoj simulaciji njihovom primjenom ne izvodi novo raspoređivanje komponenti, neovisno o promjenama u isporučenoj kvaliteti usluge koja se u ovom slučaju odnosi na postignutu propusnost. Dodatno treba naglasiti da je od trenutka *t5* postignuta jednaka učinkovitost kao i u slučaju dinamičkog algoritma za raspoređivanje, ali razlog tome nije migracija usluge na čvor u lokalnoj okolini korisnika, nego povećana propusnost na virtualnoj vezi prema čvoru *fog_node5* na kojem je primjenom ova dva algoritma usluga bila pokrenuta tijekom cijele testne simulacije.

Tablica 7.3: Propusnost na virtualnim vezama link1 i link2 (Mbps).

	***t*1**	***t*2**	***t*3**	***t*4**	***t*5**
LINK 1 (fog_node6)	100	100	100	100	30
LINK 2 (fog_node5)	100	55	45	30	100

7.2.2 Scenarij usluge automatizacije upravljanja

Nakon verifikacije ponašanja razvijenog dinamičkog algoritma raspoređivanja na jednostavnoj usluzi iz kategorije prikupljanja podataka, dizajniran je i implementiran drugi scenarij kako bi se provjerila učinkovitost algoritma na kompleksnijoj kategoriji autonomnih usluga. Simulirani scenarij usluge automatizacije upravljanja sastojao se od tri komponente:

- komponenta za prihvat podataka,
- komponenta za izvođenje simulirane obrade podataka i
- komponenta za pohranu podataka u bazu podataka.

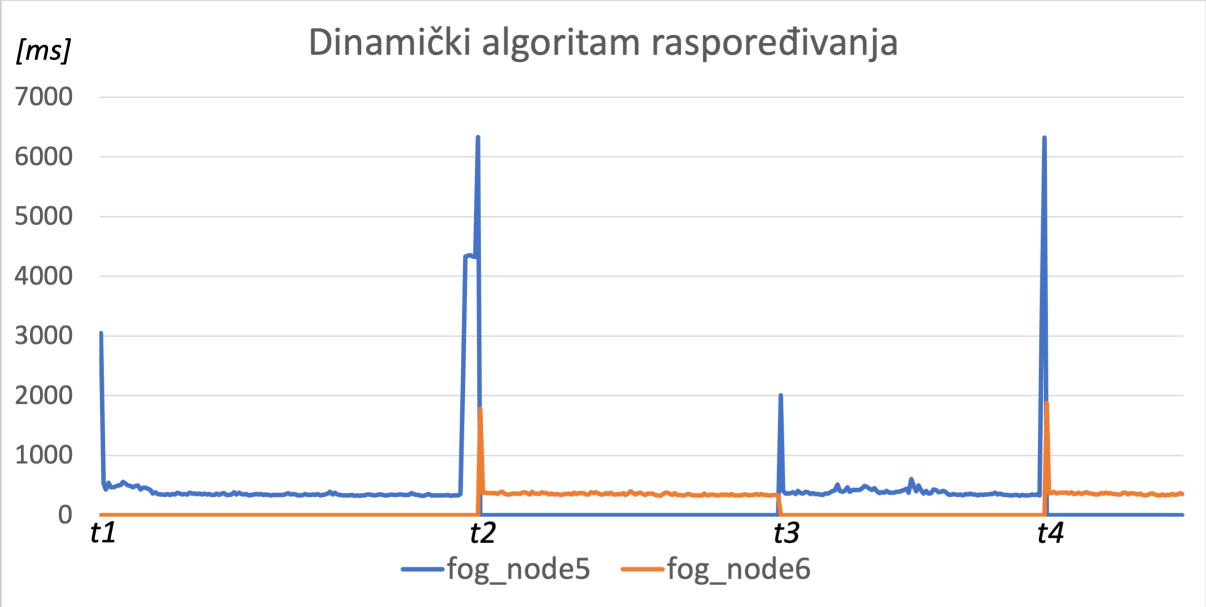
Korisnička aplikacija je implementirana tako da konstantno šalje zahtjeve prema komponenti za prihvat podataka, te pritom mjeri vrijeme kašnjenja do trenutka kad primi odgovor. Unutar usluge, komponenta za prihvat podataka prosljeđuje primljeni zahtjev komponenti za obradu, koja ga nakon obrade prosljeđuje komponenti za pohranu. Zatim se u suprotnom smjeru kroz navedene komponente propagira odgovor o uspješnosti obrade i šalje se odgovor prema korisničkoj aplikaciji koja je uputila zahtjev. Cilj ovog scenarija bio je potvrditi učinkovitost primjene razvijenog algoritma za raspoređivanje kompleksnije usluge koja sadrži više komponenti, i verificirati njegovu učinkovitost u odnosu na postojeće algoritme.

Simulacija ovog scenarija izvedena je također u simuliranoj okolini prikazanoj na slici 7.1. Agenti uređaja bili su pokrenuti na čvorovima (*fog_node5* i *fog_node6*), Raspoređivač na poslužiteljskom računalu u javnoj mreži, te Agent korisnika na čvoru *fog_node3*. Opis konteksta usluge bio je definiran tako da algoritam 2 ne postavlja nikakva ograničenja za okolinu pokretanja njenih komponenata. Prihvatljiva granica kašnjenja bila je postavljena na 5 milisekundi (ms), dok je vrijeme čekanja odgovora bilo 10 ms. Razlika te dvije vrijednosti je napravljena da bi se primili i oni odgovori čije je kašnjenje iznad zadane granice, kako bi bili vidljivi na grafu rezultata.

Primjena dinamičkog algoritma raspoređivanja

Slika 7.4 prikazuje graf rezultata simulacije primjene razvijenog dinamičkog algoritma za raspoređivanje na opisani scenarij usluge za automatizaciju upravljanja. Na čvoru *fog_node3* pokrenuta je korisnička aplikacija zajedno s Agentom korisnika, koji šalje Raspoređivaču zahtjev

za registracijom i raspoređivanjem tražene usluge. Tako se u trenutku *t1* izvodi algoritam raspoređivanja kojim se sve komponente usluge pokreću na čvoru *fog_node5*, budući da je provedenim ispitivanjem utvrđeno da je njegovo kašnjenje u komunikaciji s korisnikom manje od kašnjenja u komunikaciji korisnika s čvorom *fog_node6*. Kašnjenje je bilo ispod zadane granice od 5 ms sve do trenutka *t2*, u kojem je na virtualnoj vezi *link2* povećana vrijednost kašnjenja iznad zadane granice pa je odgovor primljen tek nakon 6 ms. Tako je u trenutku *t2* ponovno pokrenut algoritam raspoređivanja, te su sve komponente migrirane na čvor *fog_node6*. Na njemu su se izvodile komponente usluge sve do trenutka *t3* u kojem je čvor *fog_node6* isključen s mreže, čime je uzrokovan istek vremena čekanja na odgovor, te novo pokretanje algoritma za raspoređivanje. Komponente su tako u trenutku *t3* opet pokrenute na čvoru *fog_node5*, budući da je u međuvremenu kašnjenje na vezi *link2* postavljeno opet na prihvatljivu vrijednost. U trenutku *t4* ponovno je postavljeno kašnjenje na vezi *link2* iznad prihvatljive granice, pa je opet pokrenut algoritam raspoređivanja kojim su komponente prebačene na čvor *fog_node6* koji je opet pokrenut nakon isključenja u *t3*.



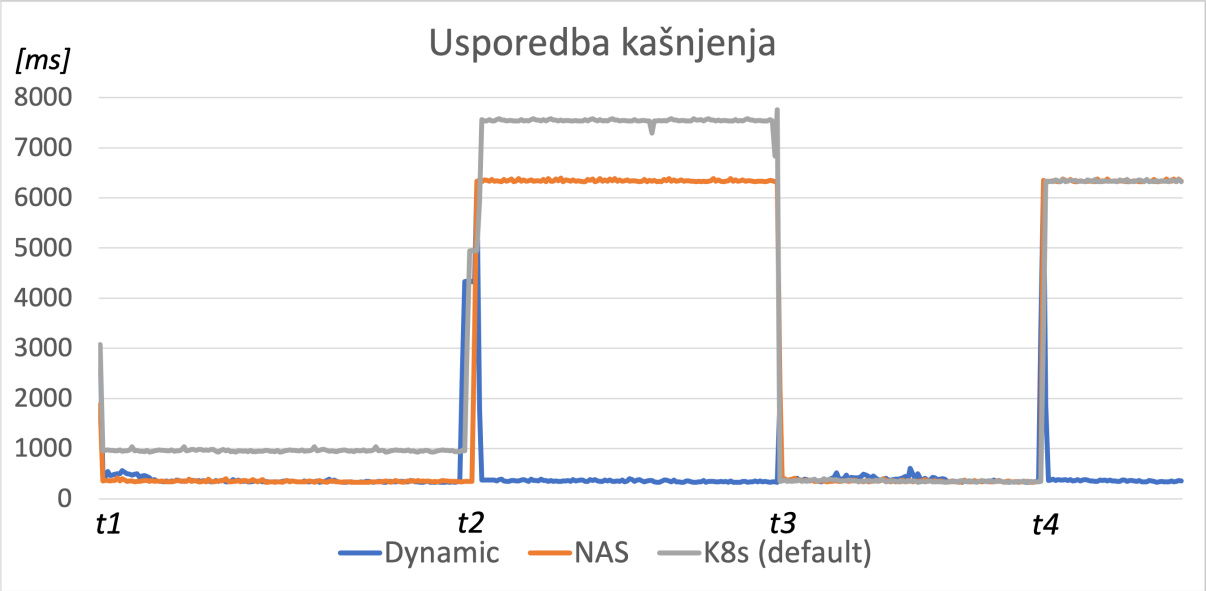
Slika 7.4: Kašnjenje u simulaciji usluge automatizacije upravljanja primjenom dinamičkog algoritma raspoređivanja.

Usporedba rezultata s postojećim algoritmima raspoređivanja

Kako bi se omogućilo raspodijeljeno izvođenje opisane usluge u Kubernetes okruženju u kojem bi se svaka komponenta mogla neovisno raspoređivati, sve tri komponente usluge izvodile bi se u vlastitoj kapsuli, budući da se sve komponente iste kapsule moraju izvoditi na istom uređaju [87]. Zadani algoritam raspoređivanja u Kubernetesu uzima u obzir samo dostupne procesorske i memorijske resurse pri odabiru čvora za pokretanje kapsule. Tako bi se ovakvim pristupom

prva kapsula usluge pokrenula na čvoru *fog_node5* ili *fog_node6*, budući da svi čvorovi imaju jednaku količinu dostupnih resursa. Zatim bi se druga kapsula pokrenula na drugom čvoru, budući da bi čvor na kojem je pokrenuta prva kapsula imao manju količinu dostupnih resursa. Na kraju bi se treća kapsula pokrenula opet na jednom od dva dostupna čvora, ovisno o tome koji bi od njih imao veću količinu dostupnih resursa, odnosno koja bi od prethodno pokrenutih kapsula više opteretila čvor na kojem je pokrenuta.

Primjenom NAS algoritma sve bi se kapsule rasporedile na čvor koji u trenutku raspoređivanja ima najmanje trajanje kašnjenja, budući da se u ovom pristupu povrh Kubernetesovih uvjeta uzima u obzir i RTT između korisničkog uređaja i dostupnih čvorova za pokretanje komponenti usluge. Ipak, ni primjenom ovog pristupa algoritam raspoređivanja ne bi bio ponovo pokrenut u slučaju povećanja kašnjenja iznad zadane granice, odnosno pada isporučene kvalitete usluge. Tako bi se komponente nastavile izvoditi na istom čvoru, iako u izvedbenoj okolini može postojati drugi čvor koji bi mogao zadovoljiti definirane granice kvalitete usluge. Tako bi i u ovom scenariju najveća razlika učinkovitosti između razvijenog algoritma i drugih relevantnih algoritama trebala biti posljedica njegove mogućnosti dinamičnog izvođenja ovisno o parametrima kvalitete usluge u specifičnom slučaju primjene.



Slika 7.5: Usporedba učinkovitosti primjene različitih algoritama raspoređivanja u simulaciji scenarija automatizacije upravljanja.

Rezultati simulacija primjene druga dva algoritma raspoređivanja na opisani scenarij usluge za automatizaciju upravljanja prikazani su na slici 7.5. Na istoj slici je dodana i plava linija koja označava rezultate dobivene primjenom dinamičkog algoritma u prethodno izvedenoj simulaciji kako bi se lakše demonstrirala razlika u ostvarenoj učinkovitosti. Između trenutaka *t1* i *t2* postignuta je jednaka efikasnost isporuke primjenom NAS algoritma kao i u slučaju primjene dinamičkog algoritma raspoređivanja, budući da on također raspoređuje sve komponente na

čvor *fog_node5* koji ima kraće trajanje kašnjenja u komunikaciji s korisnikom. Kubernetesov zadani algoritam raspoređivanja pokrenuo je jednu komponentu na čvoru *fog_node6*, a druge dvije komponente na čvoru *fog_node5*. Međusobna komunikacija između navedena dva čvora koja se odvija na vezi *link3* zbog interakcije komponenti usluge u obradi zahtjeva uzrokuje određeno kašnjenje, pa se može primijetiti nešto manja efikasnost u isporuci usluge.

Nakon povećanja kašnjenja prema čvoru *fog_node5* u trenutku *t2*, kašnjenje u isporuci usluge se razmjerno povećava primjenom oba algoritma (NAS i K8s) budući da se ne pokreće novo izvođenje raspoređivanja komponenti. Tako je na slici vidljivo da nakon trenutka *t2* razvijeni dinamički algoritam zadržava jednako kašnjenje, budući da je izvedeno novo raspoređivanje komponenti koje su onda prebačene na *fog_node6*, dok je kašnjenje primjenom druga dva relevantna algoritma povećano razmjerno povećanju kašnjenja na vezi *link2*.

U trenutku *t3* je ponovno smanjeno kašnjenje prema čvoru *fog_node5*, a čvor *fog_node6* je isključen. Ispad čvora na kojem su pokrenute komponente prepoznaje Kubernetes te pokreće novi postupak raspoređivanja, kojim se komponente pokreću na čvoru *fog_node5*. Zato je između trenutka *t3* i *t4* primjenom svih algoritama raspoređivanja ostvarena ista učinkovitost, budući da se i dinamičkim algoritmom prepoznaje ispad čvora kako je opisano u prethodnoj simulaciji.

Konačno je u trenutku *t4* ponovno povećano kašnjenje na vezi *link2* čime se u primjeni razvijenog dinamičkog algoritma pokrenulo ponovno izvođenje raspoređivanja komponenti usluge koje su tako prebačene na čvor *fog_node6*. Primjenom drugih algoritama nije pokrenuto novo raspoređivanje komponenti, pa je tako kašnjenje usluge u slučaju primjene oba algoritma odgovaralo povećanju kašnjenja na vezi *link2*. U ovom slučaju nema razlike između primjene NAS-a i zadanog Kubernetesovog algoritma raspoređivanja kao između trenutaka *t1* i *t3* jer su u trenutku *t3* u oba slučaja sve komponente pokrenute na čvoru *fog_node5* koji je bio jedini dostupan za izvođenje komponenti usluge.

Zaključak

Glavni cilj ove doktorske disertacije bio je analizirati primjenu računarstva u magli za izvođenje usluga u Internetu stvari, te povećati efikasnost izvođenja IoT usluga primjenom ovog koncepta. U teorijskoj analizi računarstva u magli istaknute su glavne prednosti koje se mogu postići njegovom primjenom u arhitekturi IoT sustava, ali je zbog različitosti usluga Interneta stvari način primjene uvijek potrebno prilagoditi konkretnom scenariju. Zato je najprije napravljena analiza cjelokupne okoline računarstva u magli koja uključuje tri ključna konteksta važna za njegovu primjenu u arhitekturi Interneta stvari: kontekst uređaja, kontekst korisnika i kontekst usluge. Za svaki navedeni kontekst definirani su parametri koji ga određuju, te se na temelju vrijednosti ovih parametara mogla odrediti mogućnost primjene računarstva u magli u okolini specifičnog IoT scenarija. Također, napravljena je i specifikacija parametara kvalitete usluge koji određuju efikasnost primjene računarstva u magli na scenarije Interneta stvari, a potom su određene i kategorije IoT usluga kojima je definiran prioritet navedenih QoS parametara u njihovoj isporuci.

Za efikasnu primjenu računarstva u magli u arhitekturi Interneta stvari, potrebno je odrediti raspored izvođenja komponenti usluge kojim se postiže najveća kvaliteta u njihovoj isporuci. Kako bi se postigao takav cilj bilo je potrebno implementirati algoritam raspoređivanja komponenti usluge koji bi na temelju prethodno definiranih konteksta i kategorije konkretne usluge raspoređivao njene komponente u dostupnoj izvedbenoj okolini. Sve popularnija mikroservisna arhitektura i virtualizacija zasnovana na kontejnerima pružili su tehnološku podlogu za lakšu implementaciju različitih rasporeda izvođenja IoT usluga u heterogenoj i nestabilnoj okolini kakvu podrazumijeva računarstvo u magli. Zato je na temelju ovih tehnologija razvijen sustav za raspoređivanje komponenti IoT usluga, koji je za izvođenje vlastite funkcionalnosti koristio formalno definirani model prethodno izdvojenih relevantnih konteksta u primjeni računarstva u magli. Logika raspoređivanja implementirana je prema razvijenom dinamičkom algoritmu, koji daje prioritet izvođenju komponenti bliže korisnika, ali uzima u obzir i ograničenja konkretnog slučaja primjene. Također, glavna njegova karakteristika kojom se izdvaja od postojećih algoritama za raspoređivanje je mogućnost dinamičke adaptacije rasporeda kako bi se osigurale zadane granice kvalitete usluge u njenoj isporuci.

Na kraju je definiran postupak verifikacije izvedbene učinkovitosti algoritma za raspore-

đivanje prema isporučenoj kvaliteti usluge koja se postiže njegovom primjenom na specifični IoT scenarij. Njime je definirano da se efikasnost razvijenog algoritma dokazuje ako je isporuka usluge rasporedom kojeg on određuje efikasnija, u okviru relevantnih QoS parametara za konkretnu IoT uslugu, od isporuke usluge u računalnom oblaku i od isporuke primjenom postojećih algoritama za raspoređivanje. Najčešće se za raspoređivanje koristi Kubernetesov zadani algoritam, pa je u postupku verifikacije potrebno uzeti njega u obzir, te druge algoritme koji bi u specifičnom slučaju primjene bili relevantni. Tako je u verifikaciji razvijenog algoritma, osim usporedbe s Kubernetesovim zadanim algoritmom, učinkovitost uspoređena i s algoritmom NAS koji je implementiran povrh Kubernetesovog algoritma te dodatno u obzir uzima i vrijeme odziva pri odabiru čvora za pokretanje usluge.

Izvedenim simulacijama demonstrirana je verifikacija učinkovitosti razvijenog algoritma na dva simulirana IoT scenarija. U oba slučaja je demonstrirano da se primjenom razvijenog algoritma postiže veća efikasnost isporučene usluge u okviru QoS parametra koji su u specifičnom slučaju bili relevantni. Glavna prednost u odnosu na postojeće pristupe bila je mogućnost dinamičkog izvođenja, odnosno adaptacije rasporeda na pad kvalitete uzrokovan vanjskim faktorima okoline. Tako se može zaključiti da je pri raspoređivanju komponenti IoT usluga u nestalnoj okolini računarstva u magli važno u obzir uzimati što veći broj kontekstnih informacija kako bi se postigao što efikasniji raspored izvođenja, ali i da je potrebno osigurati mogućnost adaptacije rasporeda na promjene kako bi se zadržala zadana razina kvalitete u isporuci usluge.

Literatura

[1] Rellermeyer, J. S., Duller, M., Gilmer, K., Maragkos, D., Papageorgiou, D., Alonso, G., "The software fabric for the Internet of Things", in The Internet of Things, Floerkemeier, C., Langheinrich, M., Fleisch, E., Mattern, F., Sarma, S. E., (ur.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, str. 87–104.

[2] ITU, "Overview of the Internet of Things", dostupno na: https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-Y.2060-201206-I!!PDF-E (15. lipnja 2012.).

[3] Madakam, S., Ramaswamy, R., Tripathi, S., "Internet of Things (IoT): A literature review", Journal of Computer and Communications, Vol. 3, 04 2015, str. 164-173.

[4] Truong, H. L., Dustdar, S., "Principles for engineering IoT cloud systems", IEEE Cloud Computing, Vol. 2, 2015, str. 68-76.

[5] Sotomayor, B., Montero, R., Llorente, I., Foster, I., "Virtual infrastructure management in private and hybrid clouds", Internet Computing, IEEE, Vol. 13, 11 2009, str. 14 - 22.

[6] Lu, H., Zhao, S., Xiong, X., Zheng, K., Chatzimisios, P., Hossain, M. S., Xiang, W., "Internet of Things cloud: Architecture and implementation", IEEE Communications Magazine, Vol. 54, 09 2016.

[7] Yi, S., Hao, Z., Qin, Z., Li, Q., "Fog computing: Platform and applications", in 2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb), 2015, str. 73-78.

[8] Morabito, R., Cozzolino, V., Ding, A. Y., Beijar, N., Ott, J., "Consolidate IoT edge computing with lightweight virtualization", IEEE Network, Vol. 32, No. 1, 2018, str. 102-111.

[9] Osanaiye, O., Chen, S., Yan, Z., Lu, R., Choo, K.-K. R., Dlodlo, M., "From cloud to fog computing: A review and a conceptual live vm migration framework", IEEE Access, Vol. 5, 2017, str. 8284-8300.

[10] Vaquero, L. M., Rodero-Merino, L., "Finding your way in the fog: Towards a comprehensive definition of fog computing", SIGCOMM Comput. Commun. Rev.,

Vol. 44, No. 5, oct 2014, str. 27–32, dostupno na: https://doi.org/10.1145/2677046.2677052

[11] Li, J., Zhang, T., Jin, J., Yang, Y., Yuan, D., Gao, L., "Latency estimation for fog-based Internet of Things", in 2017 27th International Telecommunication Networks and Applications Conference (ITNAC), 2017, str. 1-6.

[12] Naha, R. K., Garg, S., Georgakopoulos, D., Jayaraman, P. P., Gao, L., Xiang, Y., Ranjan, R., "Fog computing: Survey of trends, architectures, requirements, and research directions", IEEE Access, Vol. 6, 2018, str. 47 980-48 009.

[13] Dastjerdi, A. V., Gupta, H., Calheiros, R. N., Ghosh, S. K., Buyya, R., "Fog computing: Principles, architectures, and applications", 2016.

[14] OpenFog Consortium, "IEEE standard for adoption of OpenFog Reference Architecture for fog computing", IEEE Std 1934-2018, 2018, str. 1-176.

[15] Bonomi, F., Milito, R., Zhu, J., Addepalli, S., "Fog computing and its role in the Internet of Things", in Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, ser. MCC '12. New York, NY, USA: Association for Computing Machinery, 2012, str. 13–16, dostupno na: https://doi.org/10.1145/2342509.2342513

[16] Yi, S., Li, C., Li, Q., "A survey of fog computing: Concepts, applications and issues", in Proceedings of the 2015 Workshop on Mobile Big Data, ser. Mobidata '15. New York, NY, USA: Association for Computing Machinery, 2015, str. 37–42, dostupno na: https://doi.org/10.1145/2757384.2757397

[17] Dastjerdi, A. V., Buyya, R., "Fog computing: Helping the Internet of Things realize its potential", Computer, Vol. 49, No. 8, 2016, str. 112-116.

[18] Industry IoT Consortium, "About us", dostupno na: https://www.iiconsortium.org/about-us.htm (19. studeni 2021.).

[19] OpenFog Consortium, "OpenFog Reference Architecture for fog computing", dostupno na: https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf (9. veljace 2017.).

[20] Antonini, M., Vecchio, M., Antonelli, F., "Fog computing architectures: A reference for practitioners", IEEE Internet of Things Magazine, Vol. 2, No. 3, 2019, str. 19-25.

[21] Krivic, P., Skocir, P., Kusek, M., Jezic, G., "Microservices as agents in IoT systems", in KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications. Springer, 2017, str. 22–31.

[22] Hoque, S., De Brito, M. S., Willner, A., Keil, O., Magedanz, T., "Towards container orchestration in fog computing infrastructures", in 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), Vol. 2, 2017, str. 294-299.

[23] Wachal, M., "What are microservices and what are their benefits?", dostupno na: https://blog.softwaremill.com/what-are-microservices-and-what-are-their-benefits-d2aa64c52426 (29. rujna 2020.).

[24] Newman, S., Building Microservices, 1st ed. O'Reilly Media, Inc., 2015.

[25] Nadareishvili, I., Mitra, R., McLarty, M., Amundsen, M., "Microservice architecture: Aligning principles, practices, and culture", 2016.

[26] Banerjee, T., "Understanding the key differences between LXC and Docker", dostupno na: https://archives.flockport.com/lxc-vs-docker (5. listopad 2020.).

[27] Puliafito, C., Vallati, C., Mingozzi, E., Merlino, G., Longo, F., Puliafito, A., "Container migration in the fog: A performance evaluation", Sensors (Basel, Switzerland), Vol. 19, 2019.

[28] AWS, "AWS IoT Greengrass", dostupno na: https://aws.amazon.com/greengrass/ (29. listopada 2020.).

[29] Azure, M., "Azure IoT Edge", dostupno na: https://azure.microsoft.com/en-us/services/iot-edge/ (29. listopada 2020.).

[30] Aljanabi, S., Chalechale, A., "Improving iot services using a hybrid fog-cloud offloading", IEEE Access, Vol. 9, 2021, str. 13 775-13 788.

[31] Kramer, K., Hedin, D., Rolkosky, D., "Smartphone based face recognition tool for the blind", in 2010 Annual International Conference of the IEEE Engineering in Medicine and Biology, 2010, str. 4538-4541.

[32] Li, Q., Zhao, J., Gong, Y., Zhang, Q., "Energy-efficient computation offloading and resource allocation in fog computing for Internet of Everything", China Communications, Vol. 16, No. 3, 2019, str. 32-41.

[33] Yousefpour, A., Ishigaki, G., Jue, J. P., "Fog computing: Towards minimizing delay in the Internet of Things", in 2017 IEEE International Conference on Edge Computing (EDGE), 2017, str. 17-24.

[34] Vilalta, R., Lopez, V., Giorgetti, A., Peng, S., Orsini, V., Velasco, L., Serral-Gracia, R., Morris, D., De Fina, S., Cugini, F., Castoldi, P., Mayoral, A., Casellas, R., Martinez,

R., Verikoukis, C., Munoz, R., "Telcofog: A unified flexible fog and cloud computing architecture for 5g networks", IEEE Communications Magazine, Vol. 55, No. 8, 2017, str. 36-43.

[35] Li, L., Guo, M., Ma, L., Mao, H., Guan, Q., "Online workload allocation via fog-fog-cloud cooperation to reduce iot task service delay", Sensors, Vol. 19, No. 18, 2019, dostupno na: https://www.mdpi.com/1424-8220/19/18/3830

[36] Brennand, C. A. R. L., Filho, G. P. R., Maia, G., Cunha, F., Guidoni, D. L., Villas, L. A., "Towards a fog-enabled intelligent transportation system to reduce traffic jam", Sensors, Vol. 19, No. 18, 2019, dostupno na: https://www.mdpi.com/1424-8220/19/18/3916

[37] Wang, J., Li, D., "Adaptive computing optimization in software-defined network-based Industrial Internet of Things with fog computing", Sensors, Vol. 18, No. 8, 2018, dostupno na: https://www.mdpi.com/1424-8220/18/8/2509

[38] Wu, H., Zhang, Z., Guan, C., Wolter, K., Xu, M., "Collaborate edge and cloud computing with distributed deep learning for smart city Internet of Things", IEEE Internet of Things Journal, Vol. 7, No. 9, 2020, str. 8099-8110.

[39] Ghobaei-Arani, M., Souri, A., Rahmanian, A., "Resource management approaches in fog computing: a comprehensive review", Journal of Grid Computing, Vol. 18, 03 2020.

[40] Dey, A., "Understanding and using context", Personal and Ubiquitous Computing, Vol. 5, 02 2001, str. 4-7.

[41] Islam, M. S. U., Kumar, A., Hu, Y.-C., "Context-aware scheduling in fog computing: A survey, taxonomy, challenges and future directions", Journal of Network and Computer Applications, Vol. 180, 2021, str. 103008, dostupno na: https://www.sciencedirect.com/science/article/pii/S1084804521000357

[42] Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., Steggles, P., "Towards a better understanding of context and context-awareness", in Handheld and Ubiquitous Computing, Gellersen, H.-W., (ur.). Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, str. 304–307.

[43] Santos, J., Wauters, T., Volckaert, B., De Turck, F., "Resource provisioning in fog computing: From theory to practice", Sensors, Vol. 19, No. 10, 2019, dostupno na: https://www.mdpi.com/1424-8220/19/10/2238

[44] Kubernetes, "Kubernetes scheduler", dostupno na: https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/ (8. prosinca 2021.).

[45] Caminero, A. C., Muñoz-Mansilla, R., “Quality of service provision in fog computing: Network-aware scheduling of containers”, Sensors, Vol. 21, No. 12, 2021, dostupno na: https://www.mdpi.com/1424-8220/21/12/3978

[46] Kayal, P., “Kubernetes in fog computing: Feasibility demonstration, limitations and improvement scope : Invited paper”, in 2020 IEEE 6th World Forum on Internet of Things (WF-IoT), 2020, str. 1-6.

[47] Kubernetes, “Control plane-node communication”, dostupno na: https://kubernetes.io/docs/concepts/architecture/control-plane-node-communication/ (2.12.2021).

[48] Wang, Z., Buyya, R., “Integration of fogbus2 framework with container orchestration tools in cloud and edge computing environments”, 2021.

[49] Feiszli, A., “8 use cases for Kubernetes over VPN: Unlocking multicloud flexibility”, dostupno na: https://itnext.io/8-use-cases-for-kubernetes-over-vpn-unlocking-multicloud-flexibility-3958dab2219f

[50] Libelium, “Waspmote (v1.1) vs Waspmote pro (v1.2)”, dostupno na: https://www.libelium.com/v11-files/downloads/Waspmote-v11-VS-v12.pdf (1. srpnja 2020.).

[51] Libelium, “New generation of Libelium product lines. 2016.”, dostupno na: http://www.libelium.com/downloads/new_generation_libelium_product_lines.pdf (1. srpnja 2020.).

[52] SocialCompare, “Raspberry Pi models comparison”, dostupno na: https://socialcompare.com/en/comparison/raspberrypi-models-comparison (1. srpnja 2020.).

[53] Galov, N., “25 must-know cloud computing statistics in 2020”, dostupno na: https://hostingtribunal.com/blog/cloud-computing-statistics/#gref (1. srpnja 2020.).

[54] Sarkar, S., Misra, S., “Theoretical modelling of fog computing: a green computing paradigm to support IoT applications”, IET Networks, Vol. 5, No. 2, 2016, str. 23-29.

[55] Mahmud, R., Kotagiri, R., Buyya, R., “Fog computing: A taxonomy, survey and future directions”, Internet of Everything, Oct 2017, str. 103–130, dostupno na: http://dx.doi.org/10.1007/978-981-10-5861-5_5

[56] Marin-Tordera, E., Masip, X., Garcia Almiñana, J., Jukan, A., Ren, G.-J., Zhu, J., Farre, J., “What is a fog node a tutorial on current concepts towards a common definition”, 11 2016.

[57] Yousefpour, A., Patil, A., Ishigaki, G., Kim, I., Wang, X., Cankaya, H. C., Zhang, Q., Xie, W., Jue, J. P., "Fogplan: A lightweight QoS-aware dynamic fog service provisioning framework", IEEE Internet of Things Journal, Vol. 6, No. 3, 2019, str. 5080-5096.

[58] Thoma, M., Meyer, S., Sperner, K., Meissner, S., Braun, T., "On IoT-services: Survey, classification and enterprise integration", in 2012 IEEE International Conference on Green Computing and Communications, 2012, str. 257-260.

[59] Ju, J., Kim, M.-S., Ahn, J.-H., "Prototyping business models for IoT service", Procedia Computer Science, Vol. 91, 2016, str. 882 - 890, promoting Business Analytics and Quantitative Management of Technology: 4th International Conference on Information Technology and Quantitative Management (ITQM 2016), dostupno na: http://www.sciencedirect.com/science/article/pii/S1877050916312911

[60] Asir, R., Manohar, H., Anandraj, W., Sivaranjani, K., "IoT as a service", International Conference on Innovations in information, Embedded and Communication Systems (ICI-IECS), 03 2016.

[61] Gigli, M., Koo, S. G. *et al.*, "Internet of Things: Services and applications categorization.", Adv. Internet of Things, Vol. 1, No. 2, 2011, str. 27–31.

[62] Lee, D., Lee, H., "IoT service classification and clustering for integration of IoT service platforms", J. Supercomput., Vol. 74, No. 12, Dec. 2018, str. 6859–6875, dostupno na: https://doi.org/10.1007/s11227-018-2288-7

[63] Luan, T. H., Gao, L., Li, Z., Xiang, Y., Wei, G., Sun, L., "Fog computing: Focusing on mobile users at the edge", 2016.

[64] Barros, C., Rocio, V., Sousa, A., Paredes, H., "Context-aware mobile applications in fog infrastructure: A literature review", in Trends and Innovations in Information Systems and Technologies, Rocha, Á., Adeli, H., Reis, L. P., Costanzo, S., Orovic, I., Moreira, F., (ur.). Cham: Springer International Publishing, 2020, str. 318–328.

[65] Rahman, M. A., Hossain, M. S., Hassanain, E., Muhammad, G., "Semantic multimedia fog computing and IoT environment: Sustainability perspective", IEEE Communications Magazine, Vol. 56, No. 5, 2018, str. 80-87.

[66] Singh, M., Baranwal, G., "Quality of Service (QoS) in Internet of Things", in 2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU), 2018, str. 1-6.

[67] Union, I. T., Quality of Service Regulation Manual. ITU, 2017, dostupno na: https://www.itu-ilibrary.org/content/publication/pub-8108e11f-en

[68] Jin, X., Chun, S., Jung, J., Lee, K.-H., "A fast and scalable approach for IoT service selection based on a physical service model", Information Systems Frontiers, Vol. 19, No. 6, Dec. 2017, str. 1357–1372, dostupno na: https://doi.org/10.1007/s10796-016-9650-1

[69] Alsaryrah, O., Mashal, I., Chung, T., "Energy-aware services composition for Internet of Things", in 2018 IEEE 4th World Forum on Internet of Things (WF-IoT), 2018, str. 604-608.

[70] Giacobbe, M., Di Pietro, R., Zaia, A., Puliafito, A., "The Internet of Things in oil and gas industry: A multi criteria decision making brokerage strategy", 07 2018.

[71] Qi, L., Dai, P., Yu, J., Zhou, Z., Xu, Y., "Time–location–frequency aware Internet of Things service selection based on historical records", International Journal of Distributed Sensor Networks, Vol. 13, No. 1, 2017, str. 1550147716688696.

[72] Singla, C., Mahajan, N., Kaushal, S., Verma, A., Sangaiah, A. K., Modelling and Analysis of Multi-objective Service Selection Scheme in IoT-Cloud Environment. Cham: Springer International Publishing, 2018, str. 63–77, dostupno na: https://doi.org/10.1007/978-3-319-70688-7_3

[73] Badawy, M. M., Ali, Z. H., Ali, H. A., "QoS provisioning framework for service-oriented Internet of Things (IoT)", Cluster Computing, 2019, str. 1–17.

[74] Singh, M., Baranwal, G., Tripathi, A. K., "QoS-aware selection of IoT-based service", Arabian Journal for Science and Engineering, 2020, str. 10033–10050.

[75] Ahmed, A., Arkian, H., Battulga, D., Fahs, A. J., Farhadi, M., Giouroukis, D., Gougeon, A., Gutierrez, F. O., Pierre, G., Souza, P. R., Tamiru, M. A., Wu, L., "Fog computing applications: Taxonomy and requirements", dostupno na: https://arxiv.org/abs/1907.11621 2019.

[76] Agiwal, M., Saxena, N., Roy, A., "Towards connected living: 5g enabled Internet of Things (IoT)", IETE Technical Review, Vol. 36, No. 2, 2019, str. 190-202, dostupno na: https://doi.org/10.1080/02564602.2018.1444516

[77] Hewlett-Packard, "Hp study reveals 70 percent of internet of things devices vulnerable to attack", dostupno na: https://www8.hp.com/us/en/hp-news/press-release.html?id=1744676#.YGGra2QzZrh (29. srpnja 2014.).

[78] Exploitee.rs, "All your things are belong to us", dostupno na: https://blog.exploitee.rs/2017/08/ (18. kolovoza 2017.).

[79] Rauf, A., Shaikh, R. A., Shah, A., “Security and privacy for iot and fog computing paradigm”, in 2018 15th Learning and Technology Conference (L T), 2018, str. 96-101.

[80] OWASP, “Owasp internet of things project”, dostupno na: https://wiki.owasp.org/index.php/OWASP_Internet_of_Things_Project#tab=Main (1. studenog 2019.).

[81] OWASP, “Owasp top ten”, dostupno na: https://owasp.org/www-project-top-ten/ (1. listopada 2020.).

[82] Mahmood, H., “Application threat modeling using dread and stride”, dostupno na: https://haiderm.com/application-threat-modeling-using-dread-and-stride/ 31. listopada 2017.).

[83] IMUNES, “Integrated multiprotocol network emulator/simulator”, dostupno na: http://imunes.net/ (26. listopada 2021.).

[84] Salopek, D., Vasić, V., Zec, M., Mikuc, M., Vašarević, M., Končar, V., “A network testbed for commercial telecommunications product testing”, in 2014 22nd International Conference on Software, Telecommunications and Computer Networks (SoftCOM), 2014, str. 372-377.

[85] Hill, Z., Chen, S., Wall, D., Papa, M., Hale, J., Hawrylak, P., “Simulation and analysis framework for cyber-physical systems”, in Proceedings of the 12th Annual Conference on Cyber and Information Security Research, ser. CISRC ’17. New York, NY, USA: Association for Computing Machinery, 2017, dostupno na: https://doi.org/10.1145/3064814.3064827

[86] Puljiz, Z., Penco, R., Mikuc, M., “Performance analysis of a decentralized network simulator based on IMUNES”, in 2008 International Symposium on Performance Evaluation of Computer and Telecommunication Systems, 2008, str. 519-525.

[87] KubernetesBootcamp, “Pods and nodes”, dostupno na: https://kubernetesbootcamp.github.io/kubernetes-bootcamp/3-1.html

Skraćenice

ITU International Telecommunications Union
IEEE Institute of Electrical and Electronics Engineers
MTTR Mean Time To Repair
UPS Uninterruptible Power Supply
CDN Content Delivery Networks
TSN Time Sensitive Networking
JVM Java Virtual Machine
MCC Mobile Cloud Computing
MEC Mobile Edge Computing
RTT Round Trip Time
IP Internet Protocol
NAT Network Address Translation
VPN Virtual Private Network
QoS Quality of Service
OTA Over the Air Programming
LPWAN Low-power Wide Area Network
SLA Service Level Agreement
ICT Information and communications technology
RAM Random Access Memory
REST Representational State Transfer
IMUNES Integrated Multiprotocol Network Emulator/Simulator
LAN Local Area Network
bps bits per second
OCI Open Container Initiative
GDPR General Data Protection Regulation

Životopis

Petar Krivić rođen je 05. ožujka 1992. u Splitu gdje je završio osnovnu školu i gimnaziju. Diplomirao je na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu u srpnju 2016. godine studij Informacijske i komunikacijske tehnologije (profil Telekomunikacije i informatika). U rujnu iste godine započeo je Doktorski studij također na Fakultetu elektrotehnike i računarstva u Zagrebu.

Na Zavodu za telekomunikacije Fakulteta elektrotehnike i računarstva u Zagrebu se zaposlio u rujnu 2016. godine na radnom mjestu zavodskog suradnika, te sudjeluje u istraživanju vezanom uz Horizon 2020 projekt Europske Unije pod nazivom "symbIoTe – Symbiosis of smart objects across IoT environments". U rujnu 2018. godine zaposlen je također na Zavodu za telekomunikacije na radnom mjestu istraživača, te sudjeluje u istraživanju vezanom uz projekt "Helm Smart Grid". Od rujna 2016. godine sudjeluje i u izvođenju nastave na preddiplomskom i diplomskom studiju na kolegijima Objektno orijentirano programiranje, Konkurentno programiranje, Informacija, logika i jezici, te Laboratorij telekomunikacija i informatike 1.

U istraživačkom radu usmjeren je na područje modernih primjena Interneta stvari i računarstva u magli, te suvremenih tehnologija koje se koriste za njihovu implementaciju. U sklopu svog istraživanja objavio je više radova na međunarodnim konferencijama i jedan rad u časopisu s međunarodnom recenzijom. Od 2017. godine član je međunarodnog strukovnog društva IEEE.

Popis objavljenih djela

Izvorni znanstveni radovi u indeksiranim časopisima

1. Krivić, Petar; Kušek, Mario; Čavrak, Igor; Skočir, Pavle. Dynamic Scheduling of Contextually Categorised Internet of Things Services in Fog Computing Environment // Sensors, 22 (2022), 2; 465, doi:10.3390/s22020465

Znanstveni radovi u drugim časopisima

1. Krivić, Petar; Živkovic, Jakov; Kušek, Mario. Agent-Based Control of Service Scheduling Within the Fog Environment. // Smart Innovation, Systems and Technologies, 186 (2020), 83-92.
2. Krivić, Petar; Skočir, Pavle; Kušek, Mario. Agent-Based Approach for Energy-Efficient IoT Services Discovery and Management // Smart Innovation, Systems and Technologies, 96 (2018), 57-66.
3. Krivić, Petar; Skočir, Pavle; Kušek, Mario; Ježić, Gordan. Microservices as Agents in IoT Systems. // Smart Innovation, Systems and Technologies, 74 (2017), 22-31.
4. Skočir, Pavle; Krivić, Petar; Tomeljak, Matea; Kušek, Mario; Ježić, Gordan. Activity Detection in Smart Home Environment. // Procedia Computer Science, 96 (2016), 672-681.

Znanstveni radovi u zbornicima skupova s međunarodnom recenzijom

1. Krivić, Petar; Guberović, Emanuel; Podnar Žarko, Ivana; Čavrak, Igor. Evaluation of selected technologies for the implementation of meter data management system. // Proceedings of the 10th International Conference on the Internet of Things, Malmö, Švedska, 2020. str. 1-8.
2. Krivić, Petar; Kušek, Mario. Location-aware scheduling of IoT services in fog computing. // SoftCOM 2019 PhD Forum Book of Abstracts, Split, 2019. str. 10-11.
3. Guberović, Emanuel; Krišto, Fran; Krivić, Petar; Čavrak, Igor. Assessing compression algorithms on IoT sensor nodes. // 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, 2019. str. 913-918.
4. Krivić, Petar; Kušek, Mario. Microservices in the Fog layer of IoT. // SST 2018 PhD Forum Book of Abstracts, Osijek, 2018. str. 19-20.

Biography

Petar Krivić was born on 5th of March in 1992. in Split, where he finished the primary school and gymnasium. He graduated the profile of Telecommunications and Informatics in Information and Communication Technology study programme, on Faculty of Electrical Engineering and Computing in July 2016. In September of the same year he started the doctoral study also on Faculty of Electrical Engineering and Computing in Zagreb.

He was employed at the Department of Telecommunications on Faculty of Electrical Engineering and Computing at Zagreb in September 2016. as a research associate, where he participated on EU's Horizon 2020 project "symbIoTe – Symbiosis of smart objects across IoT environments". Afterwards, in September 2018. he was employed also at the Department of Telecommunications as a researcher, where he participates on project "Helm Smart Grid". He also participates in teaching duties of undergraduate and graduate study on courses Object Oriented Programming, Information, Logic and Languages, Concurrent Programming and Laboratory of Telecommunications and Informatics 1.

His research work is directed towards modern applications of the Internet of Things and fog computing, and modern technologies used in their implementation. In scope of his research work he published several scientific papers in proceedings of international conferences and one journal paper. He is also a member of professional society IEEE.