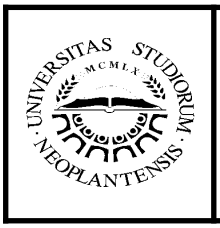


УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
НОВИ САД
Департман за рачунарство и аутоматику
Одсек за рачунарску технику и рачунарске комуникације

Дистрибуирани лансер за окружење за федеративно учење МПТ-ФЛА

Завршни (Bachelor) рад

Кандидат: Павле Васиљевић
Број индекса: РА 207/2020
Ментор рада: Проф.др Мирослав Поповић

Нови Сад, Јул 2024

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:		
Идентификациони број, **ИБР**:		
Тип документације, **ТД**:		Монографска документација
Тип записа, **ТЗ**:		Текстуални штампани материјал
Врста рада, **ВР**:		Завршни (Bachelor) рад
Аутор, **АУ**:		Павле Васиљевић
Ментор, **МН**:		проф. др Мирослав Поповић
Наслов рада, **НР**:		Дистрибуирани лансер за окружење за федеративно учење МПТ-ФЛА
Језик публикације, **ЈП**:		Српски / ћирилица
Језик извода, **ЈИ**:		Српски
Земља публиковања, **ЗП**:		Република Србија
Уже географско подручје, **УГП**:		Војводина
Година, **ГО**:		2024.
Издавач, **ИЗ**:		Ауторски репринт
Место и адреса, **МА**:		Нови Сад; трг Доситеја Обрадовића 6
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога)		7/36/0/4/19/0/0
Научна област, **НО**:		Електротехника и рачунарство
Научна дисциплина, **НД**:		Рачунарска техника и рачунарске комуникације
Предметна одредница/Кључне речи, **ПО**:		Системи на ивици интернета, Интернет ствари, Дистрибуирани Системи, Децентрализована интелигенција, Федеративно учење, Пајтон
УДК		
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, **ВН**:		
Извод, **ИЗ**:		У овом раду представљени су пројекат и решење дистрибуираног лансера за окружење за федеративно учење МПТ-ФЛА, дистрибуиране апликације која поједнустављује и аутоматизује покретање апликација писаних за окружење МПТ-ФЛА. Програмско решење евалуирано је валидацијом већ постојећих примера МПТ-ФЛА апликација притом мерећи и анализирајући њихове перформансе.
Датум прихватања теме, **ДП**:		
Датум одбране, **ДО**:		
Чланови комисије, **КО**:	Председник:	проф. др Илија Башичевић
	Члан:	проф. др Иван Каштелан
	Члан, ментор:	проф. др Мирослав Поповић

Потпис ментора

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:		
Identification number, **INO**:		
Document type, **DT**:		Monographic publication
Type of record, **TR**:		Textual printed material
Contents code, **CC**:		Bachelor Thesis
Author, **AU**:		Pavle Vasiljević
Mentor, **MN**:		Miroslav Popović, PhD
Title, **TI**:		Distributed launcher for the MPT-FLA federated learning framework
Language of text, **LT**:		Serbian
Language of abstract, **LA**:		Serbian
Country of publication, **CP**:		Republic of Serbia
Locality of publication, **LP**:		Vojvodina
Publication year, **PY**:		2024
Publisher, **PB**:		Author's reprint
Publication place, **PP**:		Novi Sad, Dositeja Obradovica sq. 6
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		7/36/0/4/19/0/0
Scientific field, **SF**:		Electrical and Computer Engineering
Scientific discipline, **SD**:		Computer Engineering and Computer Communications
Subject/Key words, **S**/**KW**:		Edge systems, Distributed systems, Internet of things, Decentralized intelligence, Federated learning, Python
UC		
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia
Note, **N**:		
Abstract, **AB**:		This paper presents the design and implementation of the distributed launcher for the MPT-FLA federated learning framework, a distributed application that simplifies and automates the execution of applications written utilising MPT-FLA framework. The aplication is evaluated by validating existing examples of MPT-FLA applications while measuring and analyzing their performance.
Accepted by the Scientific Board on, **ASB**:		
Defended on, **DE**:		
Defended Board, **DB**:	President:	Ilija Bašičević, PhD
	Member:	Ivan Kaštelan, PhD
	Member, Mentor:	Miroslav Popović, PhD

Menthor's sign

	УНИВЕРЗИТЕТ У НОВОМ САДУ • **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА** 21000 НОВИ САД, Трг Доситеја Обрадовића 6	Број:
	ЗАДАТАК ЗА ЗАВРШНИ РАД	Датум:

Студијски програм:	Рачунарство и аутоматика		
Студент:	Павле Васиљевић	Број индекса:	RA 207/2020
Степен и врста студија:	ОАС		
Област:	Електротехника и рачунарство		
Ментор:	проф. др Мирослав Поповић		

НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ЗАВРШНИ РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:
- проблем – тема рада;
- начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;

Наслов завршног рада:

Дистрибуирани лансер за окружење за федеративно учење МПТ-ФЛА

Текст задатка

У оквиру овог дипломског рада потребно је урадити следеће:
1. Упознати се са МПТ-ФЛА
2. Израдити пројекат дистрибуираног лансера за МПТ-ФЛА
3. Имплементирати дистрибуирані лансер према израђеном пројекту
4. Тестирати имплементацију
5. Урадити експерименталну валидацију дистрибуираног лансера кроз експерименталну евалуацију примера МПТ-ФЛА апликација, мерећи време њиховог извршења, као функцију броја инстанци, на жичаној или бежичној локалној мрежи рачунара.
6. Уочити предности и мане имплементираног дистрибуираног лансера.

Руководилац студијског програма:	Ментор рада:

Примерак за: ◯ - Студента; ◯ - Ментора

Захвалност

Захваљујем се ментору проф. др Мирославу Поповићу на сарадњи, саветима и стручној подршци приликом настанка овог рада.

Посебну захвалност бих изразио према мојој породици, пријатељима и колегама на безрезервној подршци током студија.

Садржај

Списак слика

Списак табела

Скраћенице

MPT-FLA - MicroPython окружење за развој алгоритама за федеративно учење

PTB-FLA - Python окружење за развој алгоритама за федеративно учење

ФЛ - федеративно учење

API - програмска спрега апликације

ДЛ - дистрибуирани лансер за окружење за федеративно учење MPT-FLA

GUI - графичка корисничка спрега

TCP - Transmission Control Protocol

HTTP - Hypertext Transfer Protocol

JSON - JavaScript Object Notation

IANA - Internet Assigned Numbers Authority

DOM - објектни модел документа

DV - DataVault

CH - CommunicationHandler

1. Увод

При раду са апликацијама писаних за окружење за федеративно учење MPT-FLA које се извршавају на више рачунара потребно је на сваком рачунару ручно путем командне линије покренути инстанцу апликације. Овај процес успорава тестирање, склон је грешкама и лоше се скалира на више од пар уређаја, те представља представља отежавајућу околност при употреби MPT-FLA окружења

Дистрбуирани лансер за окружење за федеративно учење MPT-FLA као своју основну идеју има решавање претходно наведених решења уз додатне могућности олакшања проналаска уређаја који желе да учествују у извршењу дистрибуираних апликација, валидације и мерења перформанси MPT-FLA апликација, као и анализе прикупљених података унутар једне апликације. Он такође представља и логичког наследника лансера који је доступан уз изворни код PTB-FLA і MPT-FLA али има ограничење рада на само једном рачунару.

У оквиру овог рада ће бити представљене теоријске основе које су предуслов за пројектовање једног оваквог система, концепт, то јест пројекат, решења дистрибуираног лансера, анализа реализованог програмског решења као и његова евалуација кроз валидацију већ постојећих примера апликација доступним уз изворни код окружења MPT-FLA. Последње поглавље имаће за циљ уочавање предности и мана дистрибуираног лансера и осврт на успешност испуњења постављених циљева.

2. Теоријске основе

У оквиру овог поглавља биће описан IP мултикаст који се користи као механизам за проналажење између чворова дистрибуираног лансера, концепт федеративног учења који је фокус окружења PTB-FLA и MPT-FLA као и ове платформе са нагласком на MPT-FLA као конкретна циљна платформа за коју је дистрибуирани лансер намењен.

2.1. IP Multikast

IP мултикаст [1] се дефинише као пренос IP датаграма групи крајњих тачака (*енг.* host group), скупу једног или више чланова одређених једном одредишном IP адресом. Мултикаст датаграм се доставља свим члановима одредишне групе са истом поузданошћу као и обични уникаст IP датаграми, тј. не постоји гаранција да ће сви чланови одредишне групе примити пакете, или да ће ти пакети стићи у жељеном редоследу.

Чланство у мултикаст групи је динамичко, тј. чланови могу напустити и прикључити се групи у било ком тренутку. Не постоје ограничења по питању локације чланова или броја чланова у мултикаст групи Један уређај може бити члан више група истовремено, али не мора бити члан групе да би слао пакете на адресу мултикаст групе.

Адресе које се додељују мултикаст групама припадају адресама класе Д (њихова прва 4 бита су 1110) и налазе се у опсегу од 224.0.0.0 до 239.255.255.255. Мултикаст група може бити трајна или пролазна. Трајне мултикаст групе имају у напред познате званично додељене IP адресе. Њихова трајност се дефинише постојаношћу њихових адреса, а не чланством у групи. IP адресе које нису резервисане за трајну употребу доступне су за динамичку доделу пролазним групама, које постоје све док имају бар једног члана.

2.1.1. Слање миултикаст IP датаграма

Мултикаст IP датаграми се шаљу истом "Send IP" операцијом која се користи у слању уникаст IP датаграма. На модулу протокола вишег нивоа је само да специфицира адресу групе, уместо адресе једне крајње тачке. Мултикаст уз ову основну функционалност нуди и одређени број проширења.

Модули који омогућавају слање мултикаст датаграма морају дати начин подешавања времена живота (time-to-live), у случају да корисник не наведе жељену вредност овог параметра она се поставља на 1, како би слање ван једне мреже било омогућено корисник мора намерно да постави овај параметар на жељену вредност. Ако је уређај који жели да шаље мултикаст саобраћај истовремено повезан на више мрежа, мора постојати механизам који ће одредити који мрежна спрега ће бити коришћен за слање порука. У случају да је уређај који шаље датаграме такође и члан мултикаст групе, уводи се оптимизација у виду прослеђивања пакета том члану групе преко спреге повратне петље (*енг.* loopback interface).

2.1.2. Пријем мултикаст IP датаграма

Надолазећи мултикаст IP датаграми се примају истом "Receive IP" операцијом као и уникаст датаграми. Пре пријема било ког пакета пријемник се мора придружити се мултикаст групи. За контролу чланства у мултицаст групи користи се IGMP (енг. Internet Group Management Protokol). Пре него што се пошаљу датаграми на адресу мултикаст групе неопходно је је да група има бар једног члана. Укључивање у мултикаст групу се обавља операцијама JoinHostGroup која тражи дозволу да уређај постане члан мултикаст групе на одређеној адреси. LeaveHostGroup операција захтева да уређај који је извршава напусти мултикаст групу.

Ако је дозвољено чланство у мултикаст групи користећи више од једне мрежне спреге, примљени датаграми су умножени. Такође је могуће да више апликација на истом уређају добије чланство у мултикаст групи.

2.2. Федеративно учење

Федеративно учење [2] (скраћено ФЛ) представља врсту машинског учења у којој више клијентских уређаја заједнички тренирају модел уз организацију процеса учења од стране једног централизованог сервера, док податци за обуку остају децентрализовани [3]. Децентрализованост података у федеративном учењу као непосредну последицу има побољшање приватности у апликацијама који користе приступ ФЛ у односу на скупљање, макар и анонимизованих, у централизовано складиште података. Оно што издваја ФЛ у односу на друге врсте дистрибуираног машинског учења јесте: хетерогена расподела података међу клијентима (локални податци нису нужно репрезентативни за укупну дистрибуцију свих података и они вррарирају од клијента до клијента по њиховој количини која се налази код сваког клијента), масовна дистрибуираност тј. број чворова је много већи од броја локалних података, неуравнотежена количина тренирања и коришћења апликација и ограничена количина комуникације међу учесницима у ФЛ. Такође клијенти који учествују у федеративном учењу врло често подлежу непоузданим везама и имају ограничену количину ресурса. ФЛ у својој првобитној појави је било дефинисано као централизовано али су у међувремену развијени и алгоритми децентрализованог ФЛ.

Централизовано ФЛ продразумева да један централни чвор (сервер) организује читав процес учења, одговоран је за одабир чворова који ће учествовати у учењу, и за скупљање и спајање приспелих модела у нови модел који ће бити коришћен у наредним итерацијама учења. Мана централизованог федаративног учења јесте свакако што сервер може да постане уско грло система, али и потенцијална јединствена тачка отказа целог система.

У децентрализованом ФЛ чворови међусобно сарађују како би дошли до глобалног модела. Ово елеминише јединствену тачку отказа система, али је понашање система зависно од топологије мреже.

Федеративно учење се често сматра добрим решењем за проблеме где појединачни корсници имају потребу да тренирају моделе на већим скуповима података него што поседују локално, али не желе да деле своје податке било то због бриге за приватност, правних или финансијских разлога.

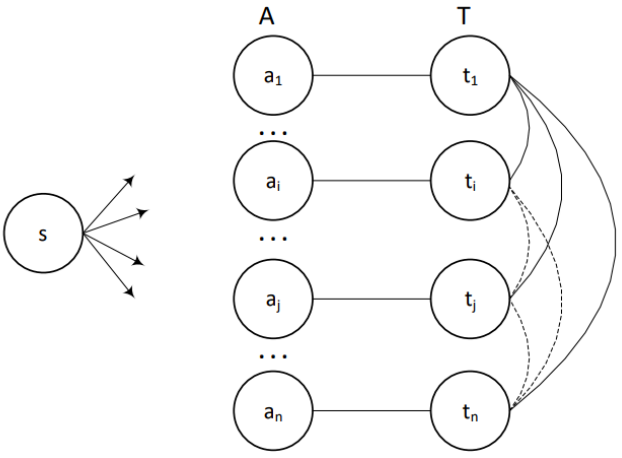
2.3. PTB-FLA

PTB-FLA [4] (*енг*. Python Testbed for Federated Learning Alghoritms)представља окружење за равој алгоритама федеративног учења, тј. извршно окружење за ФЛ алгоритме на једном рачунару. Писан је у чистом Python-у, што значи да за рад користи само стандардне Python пакете као што је multiprocessing. Оваква имплементација одабрана је како би само окружење било што компактније и како би инсталација окружења била што једноставнија за корисника.

PTB-FLA од корисника при писању алгоритама тражи две кључне ствари: корисник мора написати једну апликацију која се касније покреће од стране PTB-FLA лансера као скуп независних процеса који на основу своје идентификације (скраћено ID) испољавају одређено понашење; други захтев је да корисник обезбеди функције повратног позива (*енг*. callback functions) које представљају понашање сервера и клијента, које се у извршавању позивају од стране генеричког алгоритма федеративног учења унутар окружења PTB-FLA. Унутар окружења подржани су како централизовани тако и децентрализовани алгоритми федеративног учења.

2.3.1. Архитектура PTB-FLA

Архитектура PTB-FLA (predstavljena na slici 2.1), (представљена на слици 2.1), састоји се од процеса апликације лансера, S, дистрибуиране апликације $A = \{a_1, a_2, ..., a_n\}$ (која представља скуп инстаци апликације a_i) и дистрибуираног оружења $T = \{t_1, t_2, ..., t_n\}$ (скупа инстанци окружења), где је $i = 1, 2, ..., n$; а n представља број инстанци од A и T.



Слика 2.1 - Блок дијаграм архитектуре PTB-FLA

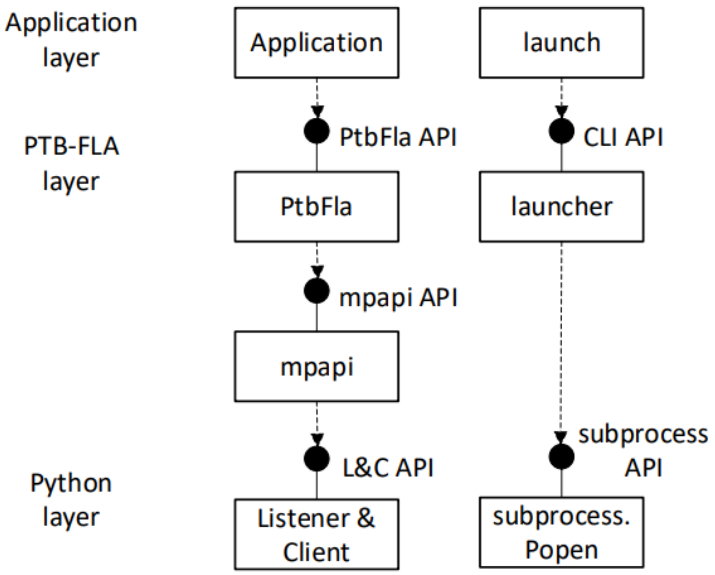
(уз дозволу аутора, преузето из референце [4])

Током извршавања дистрибуирана апликација *A* користи дистрибуирано окружење *T* како би извршавала дистрибуирани алгоритам, који је дефинисан у функцији повратног позива унутар апликације и прослеђен генеричком алгоритму.

Извршавање се одвија на следећи начин: Свака инстанца a_i припрема податке на основу аргумената командне линије, затим потива генеричке API функције (fl_centralized или fl_decentralized) над инстанцом окружења t_i.

Инстанца окружења на основу ID-ја *i* извршава своју улогу у генеричком алгоритму притом размењујући поруке са другим инстанцама окружења.

Архитектура PTB-FLA састоји се од 3 слоја (слика 2.2), дистрибуиране апликације на врху (дефинише је корисник), PTB-FLA слоја (апстрахује генеричке алгоритме и комуникацију међу чворовима тј. инстанцама) и Pythonслоја (који обезбеђује класе за рад са мултипроцессинг процесима као и друге примитиве као што су Python Queue, Listener, Client i Popen iz paketa subprocess). Преглед архитектуре PTB-FLA је преузет из референце [4].



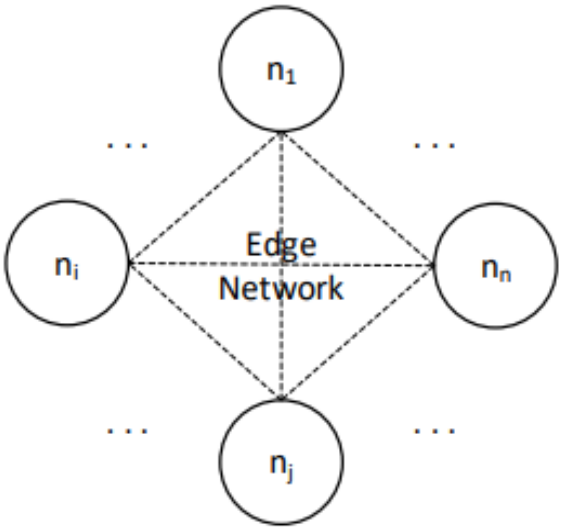
Слика 2.2 - *UML дијаграм архитектуре PTB-FLA*
(уз дозволу аутора, преузето из референце [4])

2.4. MPT-FLA

MPT-FLA [5] (*енг.* MicroPython Testbed for Federated Learning Algorithms) представља наследника PTB-FLA окружења, са фокусом на рад у локалној мрежи и на уређајима на ивици интернета (*енг.* edge devices), за разлику од његовог предходника који је радио само на једном рачунару. MPT-FLA је заснован на абстракцијама које нуди Python-ov asyncio пакет (asyncio корутине, токови и догађаји) и покреће се у MicroPython окружењу (непотпуна верзија Python-a 3 која се извршава директно на микроконтролерима и замењује оперативни систем).

2.4.1. Архитектура MPT-FLA

Систем заснован на MPT-FLA је систем на ивици интернета (*енг.* edge system) који се може представити као граф са *n* чворова (са идентификаторима од 1 до *n* (*у имплементацији обележени од 0 до n-1*)) повезаних луковима, где су чворови процеси који се извршавају на неком уређају (паметни уређаји, рачунари итд.), а лукови су TCP везе међу њима (слика 2.3). На слици 2.3 лукови су приказани испрекиданим линијама јер се везе између чворова динамички стварају на захтев за размену порука између учесника у комуникацији, а након завршетка комуникације се раскидају.



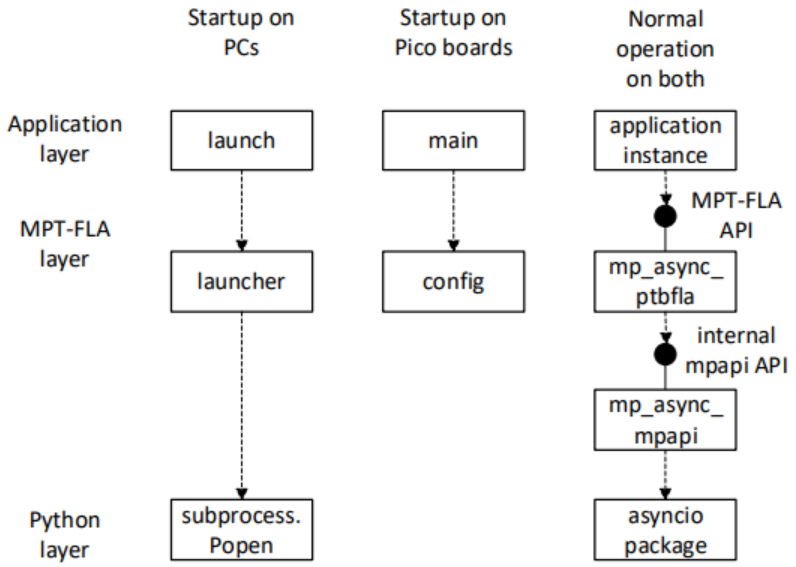
Слика 2.3 - *Архитектура MPT-FLA,*
(уз дозволу аутора, преузето из референце [5])

При покретању система, и већином времена, везе између чворова не постоје, и у току извршавања се по потреби динамички успостављају и раскидају након завршетка

њихове употребе. Из тога проилзилази да ако два чвора n_i и n_j никада не комуницирају директно између њих никада неће бити успостављена веза. Које везе ће бити успостављене зависи директно од режима у ком систем ради тако да: код генеричког централизованог ФЛ алгоритма граф приказан на слици 2.3 узима облик звезде, код децентрализованих ФЛ алгоритама облик клике, а код временски мултиплексиране размене података (*енг.* TDM peer data exchange) узима облик произвољног графа где су само парови чворова повезани.

Сваки чвор n_i извршава једну инстанцу апликације a_i који на почетку ствара инстанцу окружења t_i. Скуп инстанци $\{a_1, ..., a_n\}$ чини дистрибуирану апликацију *A,* док скуп инстанци MPT-FLA окружења $\{t_1, ..., t_n\}$ чини дистрибуирано окружење *T.* У току извршавања *A* користи окружење *T* како би извршавао жељени алготитам користећи генричке алгоритме дате од стране MPT-FLA. Генерички алгоритми које обезбеђује MPT-FLA су генеричко централизовано ФЛ, генеричко децентрализовано ФЛ и временски мултиплексирана размена података.

Архитектура MPT-FLA је организована у три слоја: Апликациони слој, MPT-FLA слој и Python слој. Између архитектуре овог окружења и његовог преткодника може се повући јасна паралела са разликом у средњем слоју који замењује стари PTB-FLA слој заснован на Python multiprocessing апстракцијама, новим MPT-FLA слојем заснованим на asyncio пакету. MPT-FLA слој се састоји од модула mp_async_ptbfla и mp_async_mpapi (где префикс mp_async означава респективно MicroPython и asyncio). Осим хоризонталних слојева на слици 2.4 можемо увидети и вертикалне колоне које предствљају начин покретања MPT-FL апликације како на рачунарима који користе потпуну верзију Pythona 3 (*лево*) тако и на уређајима који користе MicroPython (*средина*), уз начин рада заједнички за обе врсте уређаја (*десно*).



Slika 2.4:*UML* дијаграми архитектуре MPT-FLA,

(уз дозволу аутора, преузето из референце [5])

При покретању, инстанца апликације користи MPT-FLA API који јој пружа класа PtbFla, из модула mp_async_ptbfla, ради стварања и уништавања инстанце окружења, покретања чвора, и покретања једног од три генеричка алгоритма.

Класа PtbFla користи mpapi API (API за слање порука) који обезбеђује следеће asyncio корутине: server_fun, који се извршава од стране asyncio задатка, који ствара покретање PtbFla корутине; sendMsg шаље једну поруку; rcvMsg прима једну поруку; broadcastMsg шаље поруку свим чворовима; rcvMsgs прима одређени број порука (обично *n-1*).

MPT-FLAокружење има стриктно слојевиту структуру такву да сваки слој комуницира само са слојем директно изнад њега тиме пружајући добро дефинисан начин за коришћење од стране корисника, притом апстрахујући сву међусобну комуникацију између чворова, чиме омогућује корснику да се усредсреди на развој саме апликације федеративног учења. Преглед архитектуре MPT-FLA је преузет из референце [4].

3. Концепт решења

Предмет овог поглавља ће бити пројекат дистрибуираног лансера за MPT-FLA (скраћено ДЛ). Првобитно ћемо се посветити архитектури целокупног система и комуникацији између делова система, а након тога и архитектури појединачних компоненти дистрибуираног лансера.

Захтеви које ДЛ мора да испуни су следећи: проналажење рачунара који желе да учествују у извршавању MPT-FLA апликација; покретање више инстанци MPT-FLA апликација на *n* рачунара са једног рачунара на једноставан начин, притом бирајући одговарајућу апликацију са њеним параметрима која ће се извршавати на рачунарима у локалној мрежи изабраним од стране корисника; обезбедити механизам за мерење времена извршавања MPT-FLA апликација, њихово чување и понављање мерења м пута; обрада, анализа и приказ података прикупљених у току мерења.

3.1. Преглед архитектуре дистрибуираног лансера

Дистрибуирани лансер, дистрибуирана апликација за покретање апликација писаних за окружење MPT-FLA, састоји се од три јасно одвојене целине: графичке корсничке спреге (GUI), главног чвора и агентских чворова.

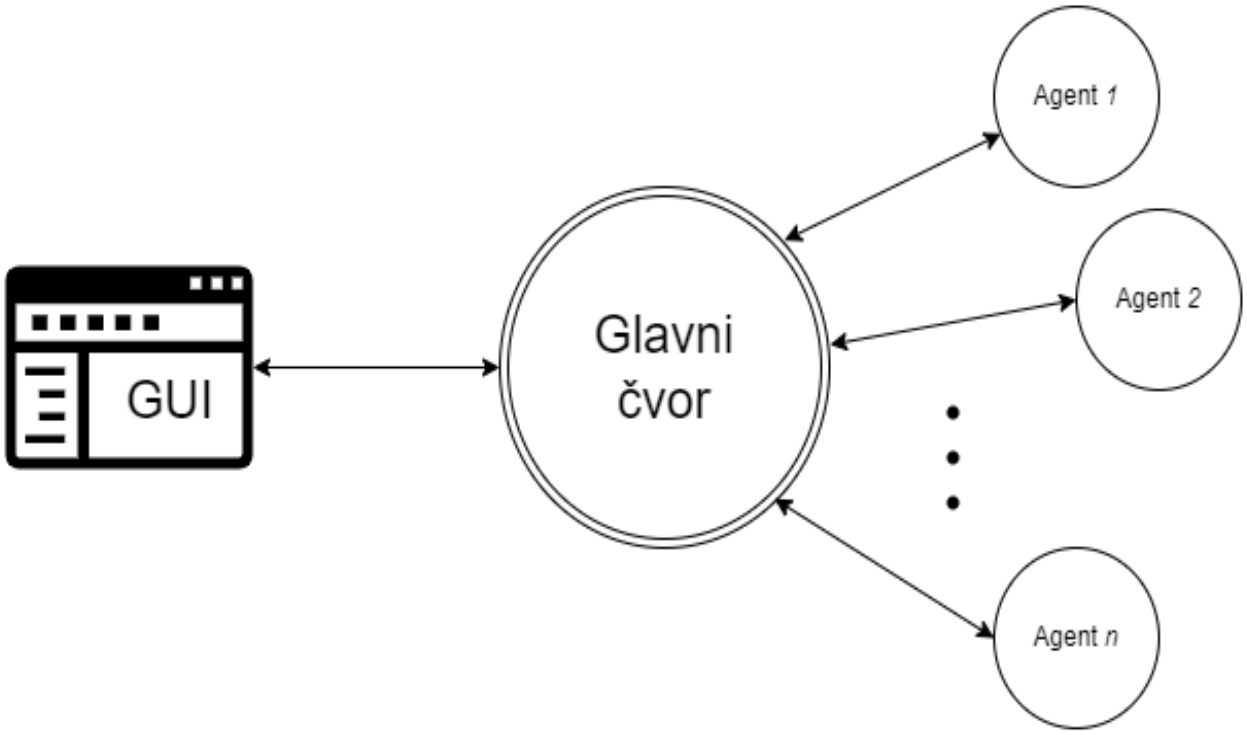
GUI има задатак да кориснику пружи лако коришћење свих функционалности које лансер пружа, на интуитиван начин, без потребе за поновним уношењем свих жељених параметара преко тастатуре при сваком коришћењу неке од функционалности, и у овом пројекту биће реализован као веб апликација клијентске стране.

Главни чвор обавља функцију веб сервера и самим тим представља везу између корисничке спреге и остатка дистрибуиране апликације. Задатак главног чвора је да координира остале чворове који учествују у покретању MPT-FLA апликације, да обезбеђује кориснику увид у тренутно доступне агенте, прикупља података добијених у

мерењима и чува их на начин погодан за даљу обраду и обрађује и генерише резултате. Уз ове функционалности главни чвор такође опслужује корисничку спрегу податцима неопходним за приказ ради унапређења корисничког искуства при употреби ДЛ.

Улога агентских чворова у дистрибуираном лансеру је да извршавају команде издате од стране главног чвора, након пријављивања да су у стању слушања наредби. По пријему команде за покретање

MPT-FLA апликације, на агентском чвору је да припреми окружење за њено извршавање и покрене је, притом исписујући излаз као и евентуалне грешке покренуте апликације у конзолу. У режиму мерења, агенти су дужни да доставе измерена времена извршавања главном чвору ради даљег чувања и обраде.



Слика 3.1 - *Архитектура дистрибуираног лансера*

3.1.1. Моделовање понашања

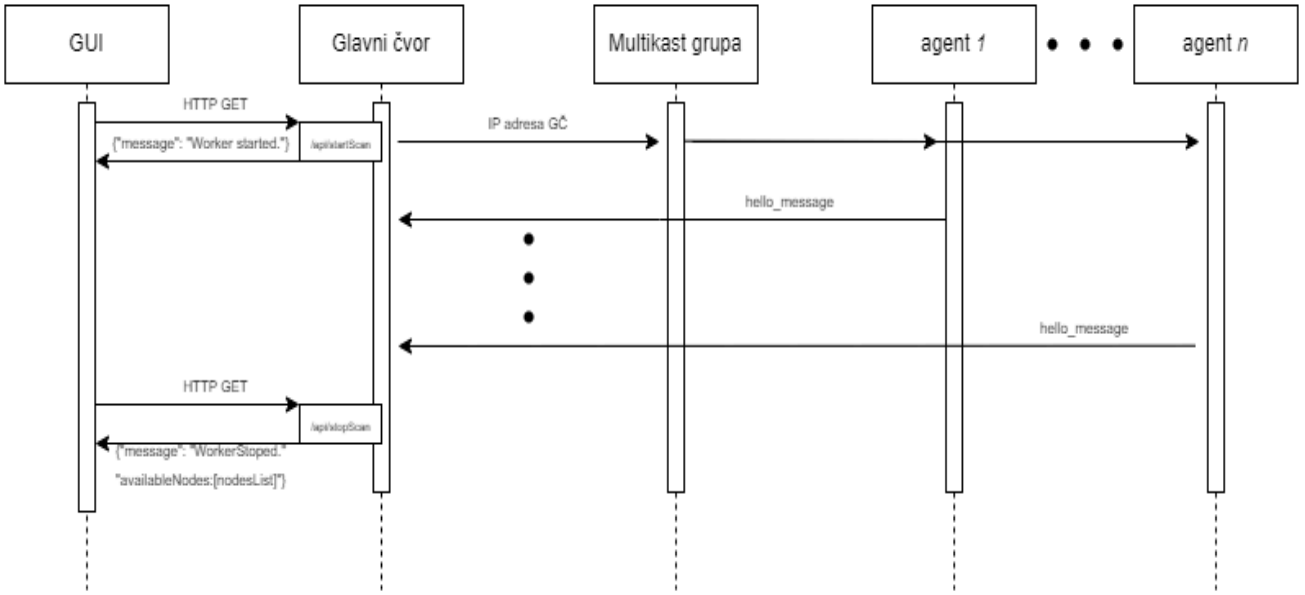
У апликацији дистрибуираног лансера, пријављене агенте који су доступни за извршавање називаћемо чворовима које идентификују пар IP адресе чвора и назива рачунара (*енг.* hostname).

Ради лакшег рада са подацима при операцијама мерења и анализе података увешћемо одређене апстракције. Прва од њих је сесија, која се везује за покретање једне MPT-FLA апликације више пута са варирајућим параметрима. Покретање исте апликације унутар једне сесије m пута са истим параметрима, називаћемо пакетом мерења. Сваки пакет мерења у себи садржи m покретања где свако покретање има произвољан број чворова условљен параметрима пакета мерења. Тих n чворова који су учествовали у извршавању мерења сваки имају и своје податке, који представљају резултате мерења. Након процеса анализе података свака сесија и пакет мерења добијају своје резултате.

3.1.2. Комуникација између компоненти

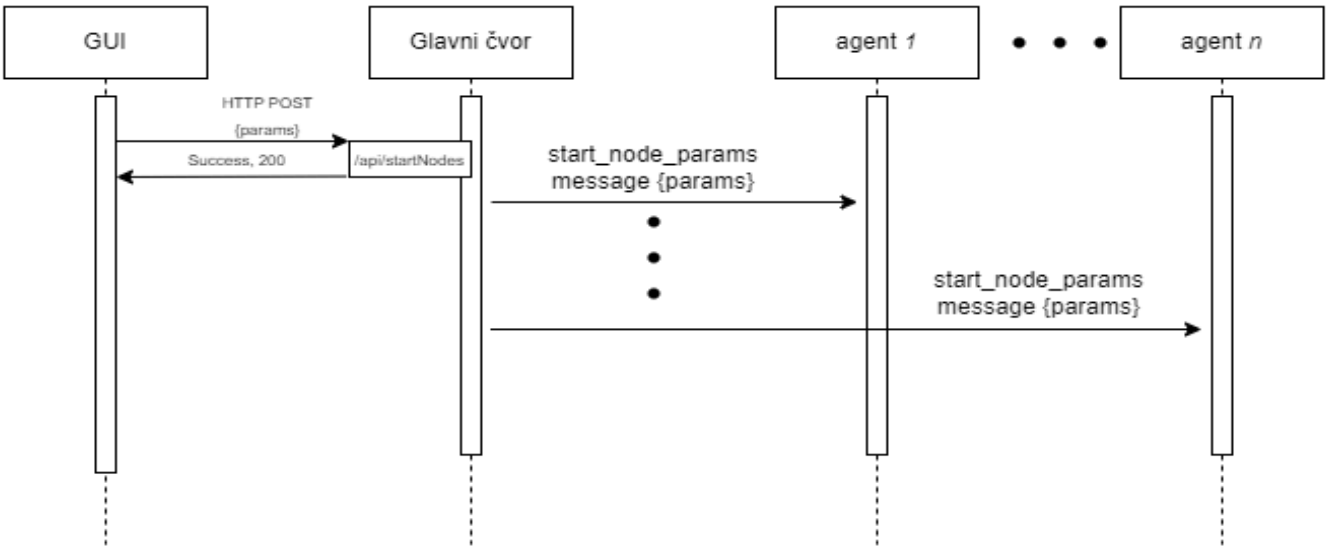
Унутар апликације дистрибуираног лансера постоје два одвојена сегмента комуникације спојена заједничком тачком, главним чвором. За комуникацију између графичке корисничке спреге и главног чвора користи се HTTP протокол, где GUI има улогу клијента, а главни чвор улогу сервера. Сервер тј. главни чвор користи RESTful API архитектуру за дефинисање крајњих тачака комуникације при опслуживању HTTP захтева, формат порука је JSON (*енг.* JavaScript Object Notation).

За оглашавање IP адресе главног чвора, агентима користи се IP мултикаст, где се на једну од адреса разервисаних за употребу унутар локалне мреже од стране IANA (*енг.* Internet Assigned Numbers Authority) шаље IP адреса главног чвора. Адреса мултикаст групе је унапред позната свим чворовима. При овоме уколико имамо више подмрежа у нашој локалној мрежи неопходно је специфицирати време живота датаграма који је послат на мултикаст адресу. Након добијања IP адресе главног чвора, агенти је јављају главном чвору путем TCP протокола. Оглашавање IP адресе, почиње слањем GET захтева на крајњу тачку /api/startScan, а завршава се слањем GET захтева на крајну тачку /api/stopScan када се у одговору прослеђују пријављени чворови (слика 3.2).

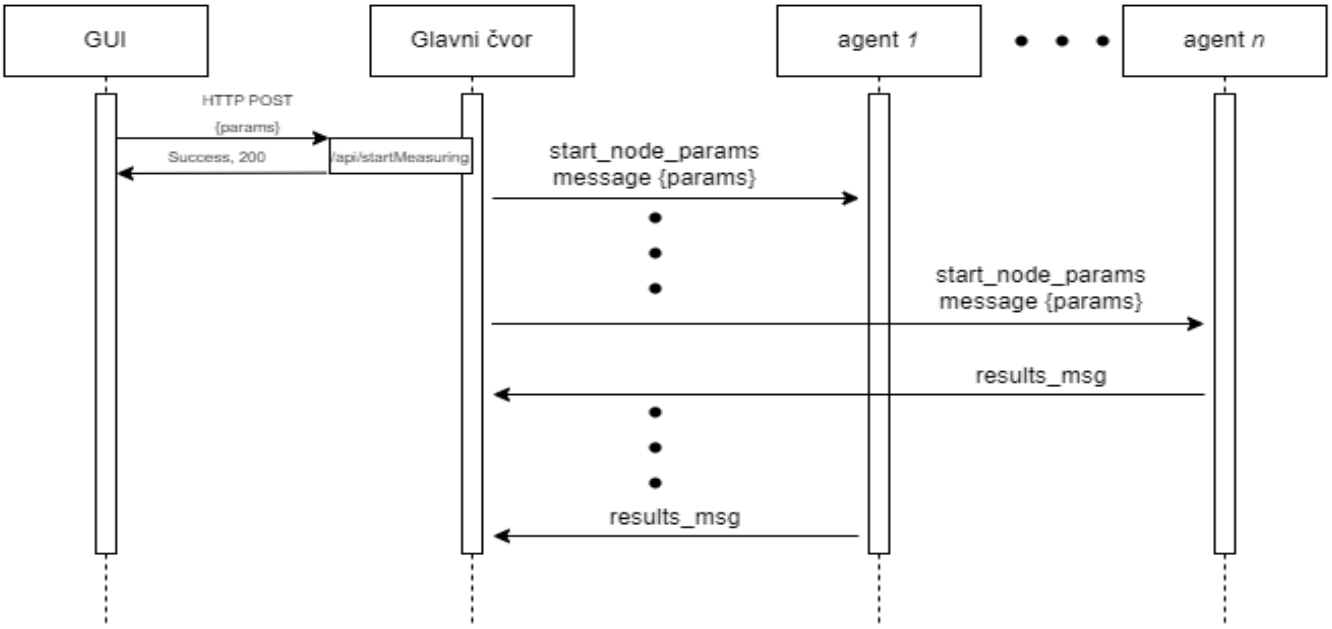


Слика 3.2 - *MSC дијаграм процеса пријављивања агената*

Комуникација између главног чвора и агентских чворова се одвија употребом TCP протокола, а формат порука је JSON. Захтев за једно понављање се шаље на крајњу тачку /api/startNodes, док се при мерењу шаље захтев на крајњу тачку /api/startMeasuring, након чега се TCP везом сваком агенту појединачно шаље TCP порука која садржи JSON са оговарајућом командом у пољу мессаге са осталим релевантним параметрима (слика 3.3). Захтев за поновљено покретање при мерењу је веома сличан претходно описаном захтеву изузев тога што се приликом слања захтева шаље и број понављања покретања, и након завршетка апликације агентски чвор шаље главном чвору резултате мерења (слика 3.4). Такође током мерења клијент (GUI) периодично прозива крајну тачку /api/measuringStatus како би добио податке о напретку процеса мерења.



Слика 3.3 - *MSC дијаграм процеса покретања апликација*



Слика 3.4 - *MSC дијаграм процеса покретања апликација при мерењу*

3.2. Графичка корисничка спрега

Апликација графичке корсничке спреге комуницира са главним чвором путем HTTP захтева и састоји се од три засебна одељка: панела за покретање апликација, панела за мерење и панела за анализу података.

Панел за покретање апликација, мора корснику да омогући: проналазак чворова, избор чворова на којима ће се извршити задате команде, избор апликације која ће бити покренута од стране лансера, додавање и уређење параметара за покретање апликације, механизам за покретање, као и механизам за удаљено искључивање чворова.

Панел за мерење је уско везан за модел понашања који смо описали у одељку 3.1.1, који моделује сесију, пакете мерења и сама мерења. кориснику се мора омогућити стварање као и избор сесије. Када је сесија изабрана, неопходно је омогућити избор чворова који ће учествовати у пакету мерења, параметре пакета мерења, број понављања покретања унутар једног пакета мерења као и начин за покретање пакета мерења. Уз ове захтеве дефинишемо и функционални захтев приказа напретка мерења које је у току како би смо побољшали корисничко искуство.

Панел за анализу је такође везан за појам сесије. Кориснику је неопходно омогућити избор сесије. Унутар панела за анализу неопходно је обезбедити механизам за извршење анализе унутар главног чвора, извоз података у формату погодном за даљу обраду као и механизам за приказ резултата генерисаних у процесу анализе.

3.2.1. Сервиси унутар корисничке апликације

Ради могућности поновне употребе програмског кода, лакшег одржавања и тестирања функционалности увешћемо две апстракције кроз класе CommunicationHandler и DataVault.

Сервис DataVault представља механизам за баферовање података у корисничкој апликацији предњег краја (*енг.* frontend application), притом имплементирајући пројектни образац синглтон. Он обезбеђује механизме за чување, освежавање података, праћење њихове застарелости. Употреба овог сервиса побољшава перфомансе система у виду смањења захтева ка серверу али и омогућава дељење садржаја између компоненти апликације.

Сервис CommunicationHandler у себи енкапсулира све позиве ка API-ју притом раздвајајући комуникацију од компоненти које се баве интеракцијом са корисником. Такође његов задатак је и да употребом сервиса DataVault након измене података апликације освежава бафероване податке.

3.3. Главни чвор

Главни чвор има двоструку улогу координатора агентских чворова као и улогу сервера за клијентску апликацију. Његове операције се покрећу слањем HTTP захтева на одговарајуће крајње тачке. Још једна подела коју можемо уочити јесте на улогу опслуживача подацима и улога извршиоца рутина.

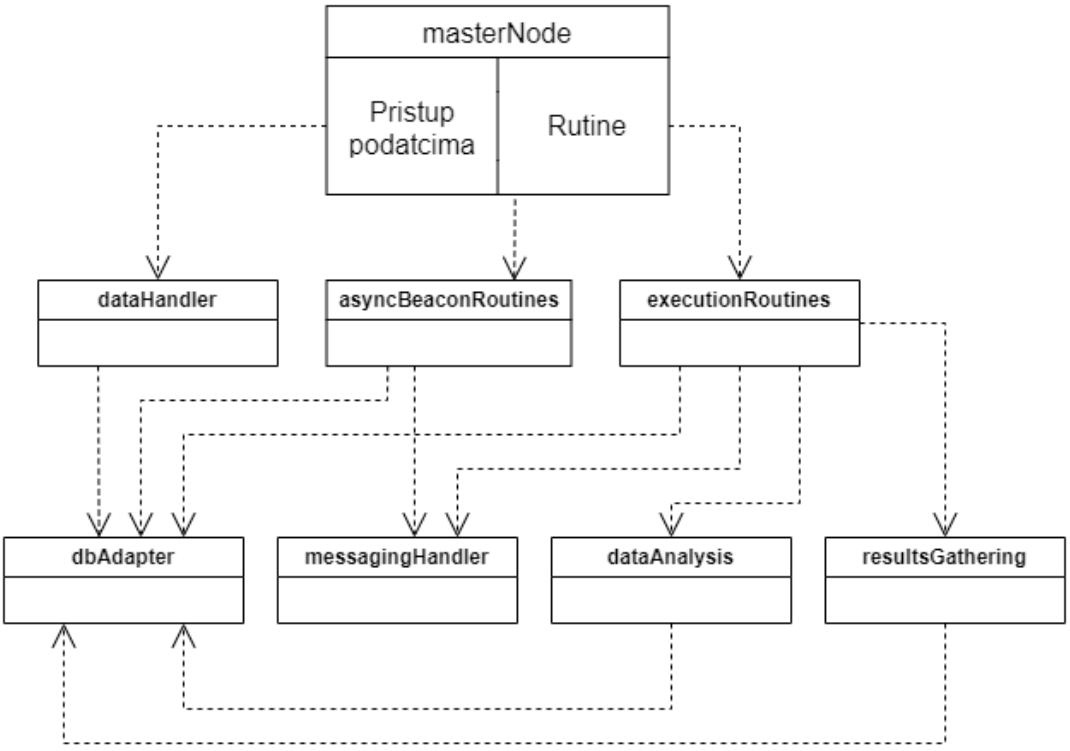
Подацима се приступа преко крајњих тачака:

- POST /api/cmdParams, ствара нови параметар за покретање апликација;
- GET /api/cmdParams, добавља листу свих параметара за покретање апликација;
- PUT /api/cmdParams/<int:item_id>, ажурира параметар за покретање апликација са задатим item_id;
- DELETE /api/cmdParams/<int:item_id>, брише параметар за покретање апликација са задатим item_id;
- POST /api/sessions, ствара нову сесију;
- GET /api/sessions , добавља листу свих сесија;
- GET /api/sessions/sessionResults, добавља резултате сесије;
- GET /api/sessions/batchResults,добавља резултате свих пакета мерења унутар сесије;
- GET /api/measuringStatus, добавља тренутни статус мерења;

Извршење рутина дефинисано је следећим крајњим тачкама:

- GET /api/stopScan, зауставља оглашавање адресе главног чвора и враћа листу пријављених чворова;
- GET /api/startScan, покреће оглашавање адресе главног чвора;
- POST /api/startNodes, покреће одабране чворове;
- POST /api/shutdownAgents, гаси агенте на одабраним чворовима;
- POST /api/startMeasuring, покреће пакет мерења;
- POST /api/doAnalasys, анализира сесије;
- POST /api/exportCSV, извози податаке у CSV формат.

Главни чвор за приступ податцима користи модул dataHandler, док за извршавање рутина користи два модула: asyncBeaconRoutines, чија је улога оглашавање IP адресе главног чвора и пријем пријава агентских чворова главном чвору и executionRoutines, чији је задатак да покреће извршавање, рутина покретања апликација, пакета мерења и анализе. Архитектура главног чвора дата је на слици 3.5.



Слика 3.5 - *Архитектура главног чвора*

Модул dbAdapter представља апстракцију над базом података која служи за трајно чување праметара коришћених за покретање апликација, резултата мерења и информација о чворовима који су били активни. База података такође има улогу механизма за дељење података између делова апликације главног чвора.

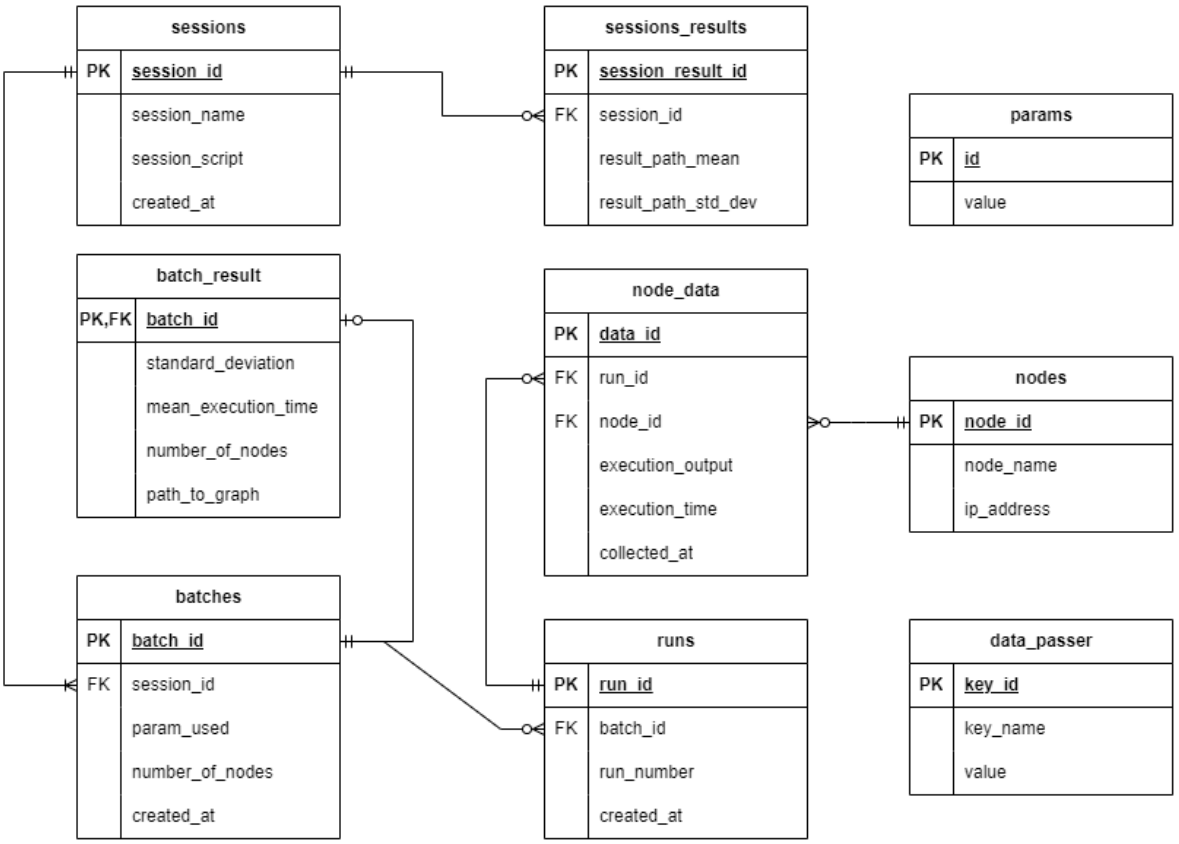
Задатак модула messagingHandler је да апстрахује сву комникацију између главног чвора и агентских чворова и пружи модулима који га корсте једноставну спрегу за мрежну комуникацију.

Модул resultsGatheringима за задатак да кроз конкурентно извршавање обезбеди прикупљање резултата мерења на ефикасан начин, чувајући притом резултате у бази података и обезбеђујући информацију о напретку процеса мерења.

Модул dataAnalysis обавља анализу и извоз података прикупљених при мерењима, генерише графике на основу резултата и чува их.

3.3.1. Пројекат базе података

Ради ефикасног складиштења и коришћења података чуваних у систему неопходно је ваљано испројектовати базу података. С обзиром на сложене односе између модела података изнетих у одељку 3.1.1 изабрана је релациона SQL база података. На слици 3.6 дат је ER дијаграм (енг. entity relationship diagram) базе података датог система.



Слика 3.6 - *ER дијаграм базе података главног чвора*

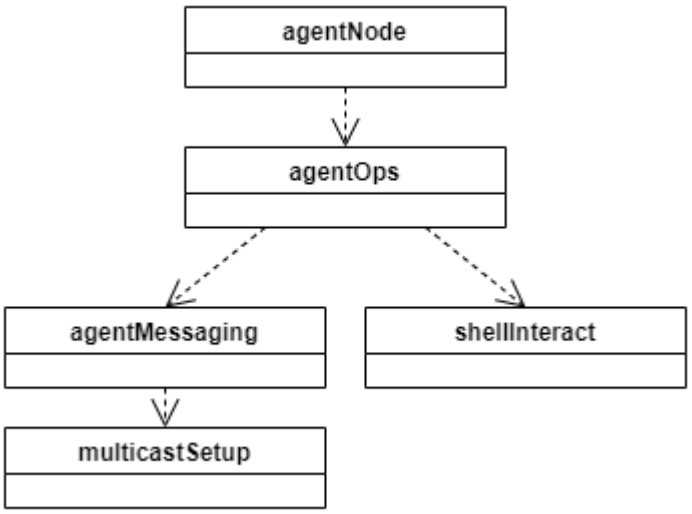
Шема базе података се придржава принципа нормализације база података, сви атрибути су атомски, атрибути зависе искључиво од примарних кључева који јединствено идентификују сваки запис, а везе између табела уређене су употребом страних кључева.

3.4 Агентски чвор

Примарна улога агентског чвора је да извршава команде послате од стране главног чвора. Битан захтев при развоју агенског чвора јесу коришћење што мање количине ресурса и преносивост. Његов животни циклус се одвија у две фазе, фаза проналаска, која се одвија на самом почетку извршавања, и фаза извршења у којој агент покреће MPT-FLA апликације на захтев главног чвора.

У фази проналаска агент се придружује у напред дефинисаној мултикаст групи и слуша поруку која садржи IP адресу главног чвора. Мултикаст комуникација и подешавање неопходних параметара обухваћено је модулу multicastSetup. Након успешно добијене адресе главног чвора путем TCP везе агент шаље главном чвору своју IP адресу и име рачунара на којем се агент извршава тиме објављујући главном чвору да је у стању слушања даљих инструкција и свој прелазак у фазу извршења.

У фази извршења агент слуша на TCP утичници инструкције главног чвора и након тога на основу параметара прослеђених у поруци извршава неку од предефинисаних операција. Ако је агент примио команду shutdown_agent он завршава свој животни циклус и гаси се. Ако је добио поруку старт_ноде_парамс агент припрема окружење за извршење MPT-FLA апликације и започиње извршење MPT-FLA апликације. Ова функционалност је обезбеђена модулом shellInteract. Након извршења покренуте апликације на основу параметара агент шаље информације о извршењу; времене извршавања и излаза извршења у виду статуса извршења. Након завршетка процеса извршења, агент поново улази у процес слушања даљих инструкција. Преглед архитектуре агентског чвора дат је на слици 3.7.



Slika 3.7: *Arhitektura agentskog čvora*

4. Програмско решење

Предмет овог поглавља биће представљена имплеметација програмског решења дистрибуираног лансера. Преглед програмског решења ће пратити сличну структуру као и претходно поглавље притом пратећи архитектуру ДЛ. Прва обрађена целина ће бити графичка корисничка спрега, затим главни чвор и коначно агентски чвор дистрибуираног лансера.

4.1. Програмско решење графичке корисничке спреге

У имплементацији графичке корисничке спреге изабрана је спрега базирана на веб технологијама због независности од платформе на којој се покреће и одсуства потребе за инсталацијом како библиотека за приказ апликације, тако и саме GUI апликације.

Технологије у којима је имплементиран GUI дистрибуираног лансера јесу JavaScript и његова React библиотека.Уз ове две технологије коришћено је и проширење JavaScript језика које омогућава статичку типизацију TypeScript ради конзистентнијег рада са подацима унутар неких модула. React је JavaScript библиотека развијена од стране Facebook-а намењена прављењу интерактивних и вишекратно употребљивих компоненти кориснике спреге. Посебно је погодан за развој једностраничних апликација (*енг.* single-page application). React код се пише у проширењу JavaScript-а под називом JSX које омогућује употребу HTML (*енг*. HyperText Markup Language) елемената унутар Javascript функција, притом примењујући декларативну парадигму. React апликације су веома ефикасне због примене концепта виртуалног објектног модела документа (*енг.* Virtual DOM), уместо да при променама у корисничкој спреги долази до поновног учитавања React мења виртуално стабло објектног модела документа (*енг*. DOM) унутар меморије. Након тога, React упоређује промене између

виртуалног DOM стабла и претходног стања, идентификујући само неопходне промене у стварном DOM стаблу за ефикасније ажурирање корисничке спреге.

4.1.1. Дизајн графичке корисничке спреге

Графичка корисничка спрега је реализована унутар три компоненте које енкапсулирају главне функционалности лансера између којих се навигација одвија употребом заглавља са три дугмета која мењају тренутно учитану компоненту. Те компоненте су: LauncherPanel, MeasurementPanel i AnalysisPanel.

LauncherPanel унутар себе има дугме за покретање проналаска, компоненте за избор чворова, апликације, параметара (уз могућност стварања измене и брисања истих) и дугмад за покретање извршавања апликације (launch дугме) и гашење одређених агентских чворова(shutdown nodes дугме).

MeasurementPanel до ког се долази кликом на натпис Measurement у заглављу омогућава кориснику избор сесије или стварање нове сесије које отвара прозор за стварање сесије. Након избора сесије, кориснику је омогућен избор већ пронађених чворова који ће учествовати у мерењу, избор параметара мерења, броја покретања унутар једног пакета мерења. Притиском на дугме за започињање мерења (Start measuring) испод њега се појављује показивач напретка (*енг.* progress bar) који кориснику даје повратну информацију о проценту покретања која су извршена.

AnalysisPanel такође даје кориснику избор сесије као и MeasurementPanel. Након избора сесије, ако анализа још увек није извршена кориснику се нуде дугмад за анализу података (Analyse data), извоз података у CSV формату (Export CSV) и освежавање резултата. Након клика на дугме за анализу података и завршетка анализе корисник добија потврду да је она завршена и кликом на дугме за освежавање резултата појављују се информације о пакетима мерења дате сесије као и прикази генерисаних графика.

4.1.2. Сервиси клијентке апликације предњег краја

У имплементацији графичке корисничке спреге ради побољшања перформанси и боље расподеле задатака између компоненти уведена су два сервиса: DataVault и CommunicationHandler.

4.1.2.1. Класа DataVault

Класа DataVault (DV) представља имплементацију механизма баферовања унутар апликације предње стране. Конзистентност чуваних података између свих објеката класе DV обезбеђена је имплементацијом пројектног образца синглтон, у овим

случају стварајући нову инстанцу класе само једном, чувајући референцу на њу унутар статичког члана инстанце, и враћајући ту референцу као резултат позива статичке методе getInstance. Члан storage, класе DataVault, је одговоран за чување података док је члан dirtyодговоран за праћење застарелости података унутар механизма баферовања. Оба атрибута имплементирана су употребом генеричког типа Map унутар TypeScript језика. API DataVault класе је дефинисан кроз методе:

- getInstance( ): DataVault;
- setItem(key: string, value: any);
- setDirty(key: string);
- getItem(key: string)/ any[] | undefined
- getDirty(key: string)/ boolean;
- removeItem(key: string);
- clear( );

Унутар имплементације апликације графичке корисничке спреге уз помоћ класе ДатаВаулт реализована је пиши-кроз (*енг*. write-through) стратегија баферовања.

4.1.2.2. Klasa CommunicationHandler

CommunicationHandler (CH) класа енкапсулира комуникацију клијентске апликације са сервером унутар једне целине. Ова класа користи Javascript библиотеку Axios за слање HTTP захтева ка северу тј. главном чвору. Она нуди интуитивну и униформну спрегу за слање асинхроних HTTP захтева ка серверу кроз JavaScript-ов механизам обећања(*енг.* promise). Унутар CH класе можемо увидети и поделу на класе које позивају операције главног чвора и функције које служе за рад са подацима. Приликом позива функција за добављање података ажурира се и садржај чуваних података унутар механизма за баферовање. Методе које дефиншу API CommunicationHandler класе су:

- startScan( )
- stopScan( )/Promise<any>
- startNodes(startParams: any)
- stopNodes(stopParams: any)
- postCmdParam(data: any)/Promise<any>
- getCmdParams( )/Promise<any>
- deleteCmdParam(id: number)/Promise<any>
- putCmdParam(id: number, data: any)/Promise<any>
- postSession(data: any)/Promise<any>

- getSessions( )/Promise<any>
- getBatchUsedParams( )/Promise<any>
- fetchDataMeasuringStatus( )/Promise<any>
- startMeasuringTask(data: any)
- exportCsv()
- doAnalysis(data: any)
- fetchImage(imagePath: string)/Promise<string>
- getSessionResults(sessionId: string)/Promise<any>
- getBatchResults(sessionId: string)/Promise<any>

И оне су аналогне крајњим тачкама дефинисаним у пројекту главног чвора.

4.2. Програмско решење главног чвора

Основу програмског решења главног чвора представља HTTP сервер написан у Flask радном оквиру (*енг.* framework).Преко доступних крањих тачака слањемHTTP захтева корисник приступа свим операцијама који се могу извршити употребом главног чвора али и подацима доступним кроз њега. Главни чвор такође опслужује апликацију графичке корисничке спреге клијенту (претраживачу) и координира акције агентских чворова. Састоји се од три концептуална модула: dataHandler, asyncBeaconRoutines и executionRoutines. Поред ових модула имамо и модуле који пружају услуге неопходне за функционисање претходних модула: messagingHandler, dbAdapter, dataAnalysis и resultsGathering. Поред ових модула који су урађени према пројекту главног чвора дистрибуираног лансера такође је имплементиран и утилс модул који енкапсулира честе радње које се понављају у модулима а идејно се не уклапају ни у један модул као што су парсирање и манипулација текста. Сви модули имплементирани су у програмском језику Python користећи његове уграђене модуле: asyncio, csv, datetime, json, math, matplotlib, multiprocessing, os, queue, re, socket, sqlite3, sys, threading, time и typing.

4.2.1. Модул dataHandler

Модул dataHandler представља абстракцију над CRUD (стварање, читање, ажурирање и брисање) операцијама обављаним од стране главног чвора на захтев апликације графичке корисничке спреге. Логички групише радње над подацима како базе података употребом модула dbAdapter тако и из система датотека рачунара на којем се извршава апликација главног чвора употребом програмског пакета os језика Python.

4.2.2. Модул asyncBeaconRoutines

Модул asyncBeaconRoutines кроз функцију beacon_process објављује IP адресу главног чвора на адресу унапред одређене мултикаст групе употребом функције send_ip_to_multicast модула messagingHandler, све док прослеђени догађај stop (инстанца Event класе модула multiprocessing) nije postavljen. За то време функција incomingListener_processкојој је прослеђен исти догађај stopкао и функцији beacon_process уз објекат класе PipeConnection који представља један крај цеви (*енг.* pipe). incomingListener обрађује доспеле захтеве за пријављивање и након што је догађај stop постављен употребом функције stop_beacon, уз помоћ цеви incomingListenerшаље листу пријављених агентских чворова финкцији која у том тренутку чека чворове на другом крају цеви. Функције beacon_process и incomingListener_process се позивају посредством multiprocessing модула по пристизању HTTP GET захтева на /api/startScan крајну тачку, док се stop_beacon функција позива након HTTP GET захтева на /api/stopScan крајњу тачку. Након заустављања процеса проналажења преко другог краја цеви се добавља листа агентских чворова која се прослеђује као одговор на HTTP GET захтев за завршетком.

4.2.3. Модул executionRoutines

Модул executionRoutines енкапсулира кључне операције главног чвора, координацију операције агентских чворова и обраду резултата. Обрада резултата се базира на функцијама doAnalysis и generateCSVкоје представљају фасаде над функционалностима модула dataAnalysis.

Функције shutdownAgentsGracefully и startScriptsPreset шаљу појединачне команде агентским чворовима употребом функција messagingHandler модула. Пре слања команде ка одабраним агентским чворовима startScriptsPreset обавља прилагођавање општих параметара за покретање апликације чвору заменом унапред познатих токена “id” и “mip” (представљају идентификатор чвора у MPT-FLA апликацији и адресу инстанце MPT-FLA апликације која ће бити прва покренута) одговарајућим вредностима.

Функција startBatch се користи при мерењу ради покретања једног пакета мерења. Она иницијализује нови пакет мерења у бази и покреће *m* понављања. Свако понављање се покреће позивом функције startRepetition. Функција startRepetition ствара ново понављање у бази података, иницијализује праћење напретка мерења и позива функцију startScriptsPreset. Након овога се покрећу нити listener_thread, која извршава функцију results_listener resultsGathering модула слушајући надолазеће резултате и

processor_thread, која извршава функцију process_queue_messages и обрађује резултате. Након овога функција чека да пристигне број резултата једнак броју порука након чега се пошаље догађај стоп ка нитима да би се оне након тога придружиле.

4.2.4. Модул resultsGathering

Модул resultsGathering у себи садржи функције за пријем надолазећих резултата мерења и њихову обраду. Ове две операције прате пројектни образац произвођач/потрошач а канал комуникације између њих је Python Queue објекат. Након покретања, пријем и обрада резултата трају све док се прослеђени објекат Eventкласе не постави вредност своје унутрашње променљивне стања, имплементирајући пројектни образац посматрача.

4.2.5. Модул messagingHandler

Модул messagingHandler реализује слање и пријем порука употребом TCP протокола коршћењем Python библиотеке socket у JSON формату при комуникацији главног чвора и једног од агентских чворова. У овом модулу се такође реализује слање IP адресе главног чвора на мултикаст адресу у процесу проналаска.

4.2.6. Модул dbAdapter

Модул dbAdapter кроз класу SQLiteDBAdapter енкапсулира рад са SQLite базом података.

SQLite је уграђен систем за управљање базама података садржан у C програмској библиотеци, у Python програмском језику доступна је кроз модул sqlite3. Изабрана је за имплементацију овог система због одсуства потребе за сервером базе података (*енг.* serverless) и њене доступности на сваком рачунару који има инсталиран Python интерпретер.

SQLiteDBAdapter унутар себе реализује функције за иницијализацију празне базе података на шему дату у пројекту базе главног чвора као и угњеждене класе које обухватају рад са појединачним табелама као и класе Analysis и DataPasser. Класа Analysis унутар класе SQLiteDBAdapter агрегира податке прикупљене при мерењу. Класа DataPasser представља механизам за поуздано дељење података између више контекста, нити, процеса или само модула и функција који су слабо повезани на једноставан начин.

4.2.7. Модул dataAnalysis

Модул dataAnalysis има две сврхе, прва је извоз података у формату погодном за даљу обраду, у овом случају CSV формату (функција generate_csv_from_combined_data), а друга је аутоматизована обрада резултата. Аутоматизована обрада постиже се функцијом do_analysis која секвенцијално извршава две фазе обраде: анализу резултата пакета мерења и анализу сесије. Функције calculate_mean_time, calculate_standard_deviation рачунају средње време извршења апликације унутар једног пакета мерења и стандардну девијацију, док функције plot_times_with_stats, plot_meantime_nodes_number, plot_std_dev_nodes_number исцртавају графике који приказују добијене резултате.

4.3. Програмско решење агентског чвора

Агентски чвор дистрибуираног лансера као и главни чвор имплементиран је у програмском језику Python. Његови задаци су проналажење главног чвора употребом мултикаста и извршавање команди које шаље главни чвор путем TCP протокола. Агентски чвор по свом покретању у оквиру функције agentProcess улази у рутину проналаска (функција discovery) све док не добије информацију о IP адреси главног чвора, када прелази у фазу извршења. Фаза извршења траје све док функција execution која енкапсулира фазу извршења не врати вредност False тиме излазећи из петље у којој се одвија извршење агентског процеса и завршавајући извршење апликације агента. Функција discovery позива функцију connect модула agentOps која као резултат враћа IP адресу добијену путем мултикаста, коју потом чува за даље коришћење. Функција execution позива функцију wait_for_instructions модула agentOps тиме започињући фазу извршења команди. Апликација агентског чвора се састоји још и од модула: agentOps, agentMessaging, multicastSetup i shellInteract.

4.3.1. Модул agentOps

Модул agentOps у себи садржи следеће функције: connect, wait_for_instructions, start_node_params, run_script. Функција connect добавља IP адресу главног чвора посредством модула agentMessaging и његове функције receive_ip_from_multicast, а затим шаље поруку за пријављивање главном чвору кроз функцију send_hello модула agentMessaging. Функција wait_for_instructions чека пријем команде путем TCP протокола посредством функције receive_command модула agentMessaging, и по пријему

поруке на основу поља message извршава једну од предефинисаних рутина.У случају да је вредност поља message “start_node_params” позива се истоимена функција у којој се генерише команда која ће покренути жељену MPT-FLA апликацију и позива се функција run_script која покреће извршавање апликације. U okviru run_script функције покреће се жељена MPT-FLA апликација посредством класе ShellHandler модула shellInteract и мери се време извршења исте које се у режиму мерења шаље назад ка главном чвору.

4.3.2. Модул agentMessaging

Унутар модула agentMessaging енкапсулирана је сва мрежна комуникација коју обавља агентски чвор. У случају мултикаст саобраћаја користи се модул multicastSetup radi pojednostavljenog pristupa sadržaju koji se šalje putem multikast grupe. ради поједностављеног приступа садржају који се шаље путем мултикаст групе. У оквиру овог модула за комуникацију користе се функције Python socket библиотеке како би се остварила комуникација путем TCP протокола у JSON формату.

4.3.3. Модул multicastSetup

Модул multicastSetup у себи садржи класу MulticastSocket која у оквиру своје иницијализације обавља процес придруживања мрежне утичнице мултикаст групи [6][7] на унапред дефинисаној адреси. MulticastSocketкласа такође нуди и спрегу за пријем мултикаст порука путем своје recvfrom функције.

4.3.4. Модул shellInteract

Модул shellInteract кроз класу ShellHandler нуди једноставну спрегу за покретање MPT-FLA апликација. У иницијализацији објекта класе ShellHandler, уједно се иницијализују и процес који ће извршавати апликацију, посредством пакета subprocess као и Python виртуално окружење (*енг*. Python virtual environment,venv) у ком ће се извршавати апликација. Након тога се позивом функције run_command покерће жељена апликација. На крају извршавања апликације њен стандардни излаз се прослеђује као повратна вредност функције и процес се уништава.

5. Евалуација дистрибуираног лансера

У оквиру овог поглавља кроз валидацију примера датим уз програмски код окружења MPT-FLA [8] биће обављена евалуација дистрибуираног лансера за окружење MPT-FLA. Резултати евалуације биће представљени кроз резултате извршења MPT-FLA апликација добијених мерењем и анлизом измериних података. У оквиру валидације коришћено је осамнаест рачунара сличних спецификација од којих репрезентативни примерак поседује процесор Intel Core i7 6700 и 16 GB ddr3 RAM меморије који су били спојени преко dva D-LINK DGS-1016D мрежна комутатора употребом UTP каблова.

5.1. Опис методологије

У оквиру евалуације дистрибуираног лансера урађена је валидација четири примера:

- mp_async_example1_fedd_mean.py,
- mp_async_example2_cent_avg.py,
- mp_async_example3_decent_avg.py,

За сваки од ових примера формирана је посебна сесија. У свакој сесији извршено је пет пакета мерења од којих је сваки имао тридесет понављања. Између пакета мерења вариран је број чворова n код свих примера, где је $n=2,6,10,14,18$; док је у оквиру mp_async_example6_odts.py између пакета мерења вариран и број временских исечака s алгоритма као зависна променљива у односу на n где је зависност дата једначином $s=n-1$.

5.2. Опис мера и приказа резултата

При мерењу, након завршетка извршавања MPT-FLA апликације сваки агентски чвор шаље дужину трајања извршења изражену у секундама главном чвору који их чува. У току анализе у оквиру једног покретања издваја се најдуже време извршења апликације као репрезентативна вредност за цео узорак. На основу ње се даље рачунају средње време унутар пакета мерења и стандардна девијација унутар пакета мерења. Такође се у оквиру анализе исцртавају графици зависности средњег времена извршења апликација и стандардне девијације времена извршења у одосу на број чворова *n*.

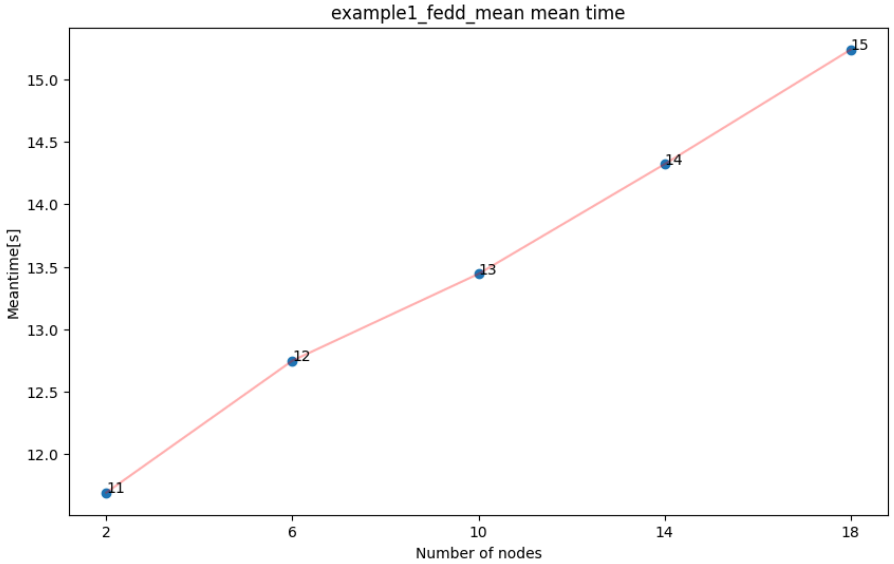
5.3. Преглед резултата

5.3.1. Сесија example1_fedd_mean

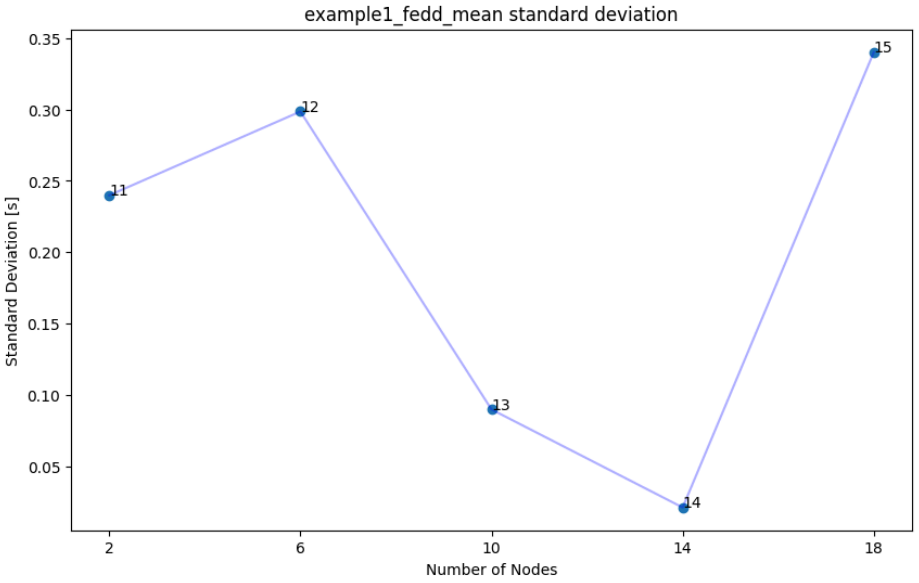
У оквиру сесије example1_fedd_mean валидирана је MPT-FLA апликација mp_async_example1_fedd_mean.py чији је циљ упросечавање вредности читања сензора изнад одређеног прага [5] . У табели 5.1 су дати резултати анализе датог примера, док су на сликама 5.1 и 5.2 дати графици зависности средњег времена извршења и стандардне девијације од броја чворова.

Табела 5.1: *Резултати анализе примера mp_async_example1_fedd_mean.py*

Параметри пакета мерења	Број чворова	Средње време извршења [s]	Стандардна девијација [s]
2 id 0 mip	2	11.689702397809254	0.23989394383172483
6 id 0 mip	6	12.746586724731353	0.2986550064467578
10 id 0 mip	10	13.442132173200546	0.08975034866481962
14 id 0 mip	14	14.321342797166654	0.021045074682002368
18 id 0 mip	18	15.239078516399974	0.3402784597477361



Слика 5.1 - *График зависности средњег времена извршења од броја чворова примера mp_async_example1_fedd_mean.py*



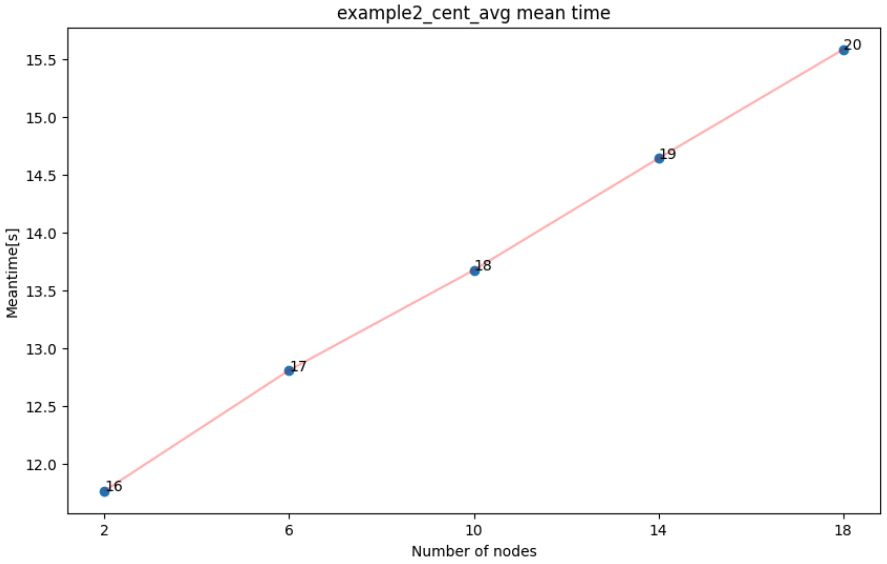
Слика 5.2 - *График зависности стандардне девијације од броја чворова примера mp_async_example1_fedd_mean.py*

5.3.2. Сесија example2_cent_avg

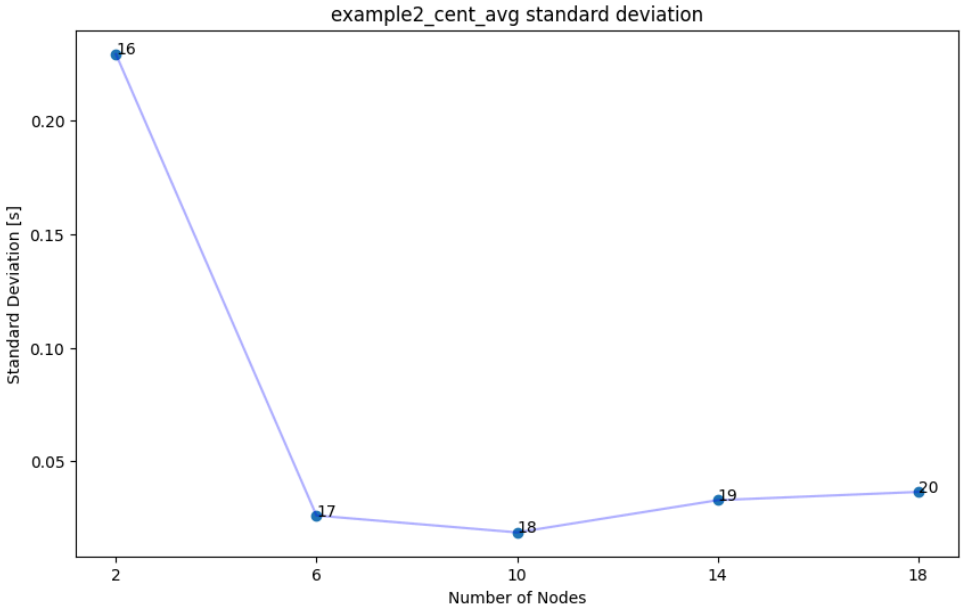
У оквиру сесије example2_cent_avg валидирана је MPT-FLA апликација mp_async_example2_cent_avg.py чији је циљ централизовано упросечавање клијентских модела модела [5] . У табели 5.2 су дати резултати анализе датог примера, док су на сликама 5.3 и 5.4 дати графици зависности средњег времена извршења и стандардне девијације од броја чворова.

Табела 5.2: *Резултати анализе примера mp_async_example2_cent_avg.py*

Параметри пакета мерења	Број чворова	Средње време извршења [s]	Стандардна девијација [s]
2 id 0 mip	2	11.765030595338116	0.22944452418222172
6 id 0 mip	6	12.809425546866265	0.02602417499488429
10 id 0 mip	10	13.676457096700448	0.01857781196134222
14 id 0 mip	14	14.321342797166654	0.0327790755765831
18 id 0 mip	18	15.58672114296714	0.036420799855348476



Слика 5.3 - *График зависности средњег времена извршења од броја чворова примера mp_async_example2_cent_avg.py*



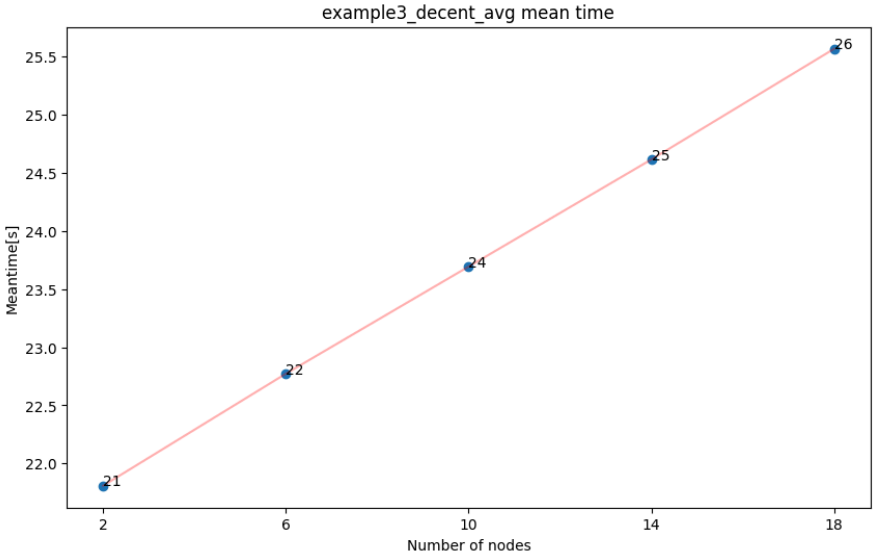
Слика 5.4 - *График зависности стандардне девијације од броја чворова примера mp_async_example2_cent_avg.py*

5.3.3. Сесија example3_decent_avg

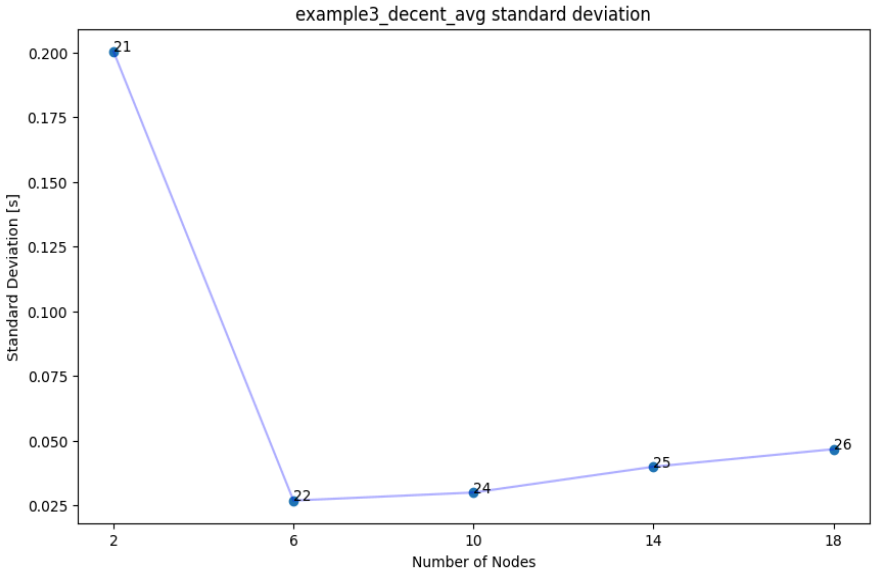
У оквиру сесије example3_decent_avg валидирана је MPT-FLA апликација mp_async_example3_decent_avg.py чији је циљ децентрализовано упросечавање клијентских модела модела [5] . У табели 5.3 су дати резултати анализе датог примера, док су на сликама 5.5 и 5.6 дати графици зависности средњег времена извршења и стандардне девијације од броја чворова.

Табела 5.3: *Резултати анализе примера mp_async_example3_decent_avg.py*

Параметри пакета мерења	Број чворова	Средње време извршења [s]	Стандардна девијација [s]
2 id mip	2	21.807611438591138	0.20041920498702515
6 id mip	6	22.771782142533507	0.026810578543281817
10 id mip	10	23.693824532766783	0.029919639340332326
14 id mip	14	24.61597974380032	0.039893112560694374
18 id mip	18	25.568527368666153	0.0466699464549452



Слика 5.5 - *График зависности средњег времена извршења од броја чворова примера mp_async_example3_decent_avg.py*



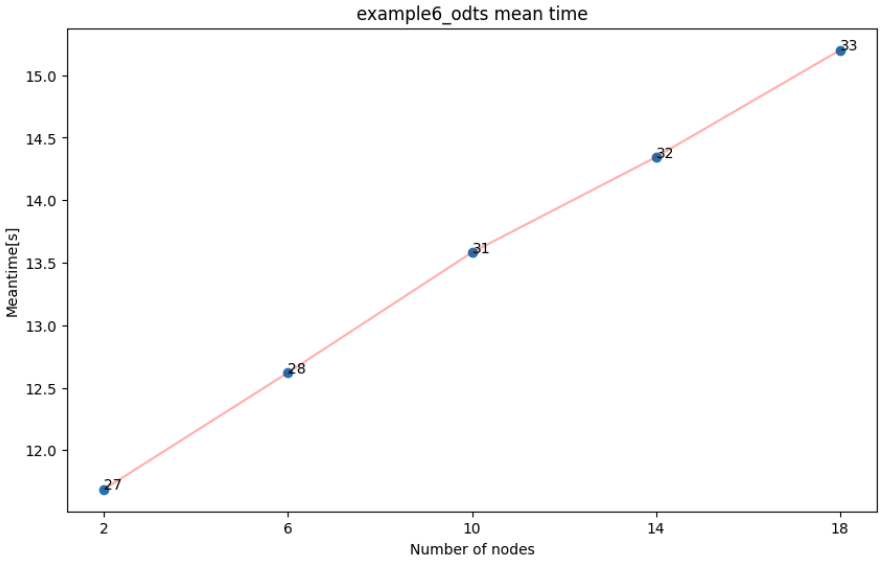
Слика 5.6 - *График зависности стандардне девијације од броја чворова примера mp_async_example3_decent_avg.py*

5.3.4. Сесија example6_odts

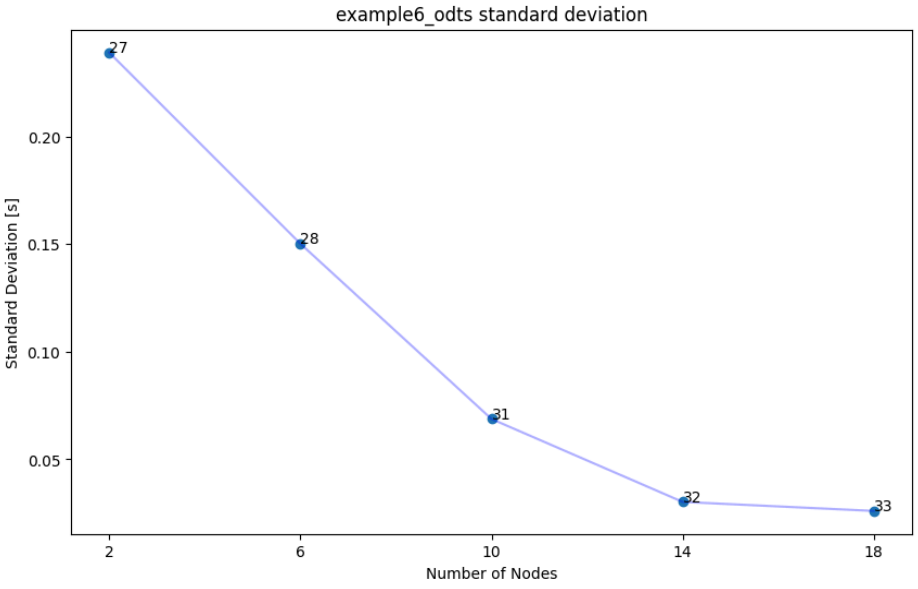
У оквиру сесије example3_decent_avg валидирана је MPT-FLA апликација mp_async_example6_odts.py чији је циљ симулација процеса одређивања орбите (позиције и брзине) и синхронизација унутрашњих сатова сателита [5] . У табели 5.4 су дати резултати анализе датог примера, док су на сликама 5.7 и 5.8 дати графици зависности средњег времена извршења и стандардне девијације од броја чворова.

Табела 5.4: *Резултати анализе примератр_async_example6_odts.py*

Параметри пакета мерења	Број чворова	Средње време извршења [s]	Стандардна девијација [s]
2 id 1 1 mip	2	11.687394253567138	0.23907608818105144
6 id 1 5 mip	6	12.62080397836762	0.15024239753704693
10 id 1 9 mip	10	13.58252334619999	0.06874976779276533
14 id 1 13 mip	14	14.345005305966817	0.030215501711063453
18 id 1 17 mip	18	15.200796974766854	0.02602536064172448



Слика 5.7 - *График зависности средњег времена извршења од броја чворова примера mp_async_example6_odts.py*



Слика 5.8 - *График зависности стандардне девијације од броја чворова примера mp_async_example6_odts.py*

6. Закључак

У предходним поглављима поставили смо теоријске основе неопходне за развој дистрибуираног лансера, представили његов пројекат и имплементацију и евалуирали његов рад. На основу тестирања и евалуације може се доћи до закључка да дистрибуирани лансер испуњава захтеве који су постављени при његовом пројектовању: проналажење агентских чворова, покретање апликација на *n* рачунара, мерење времена извршавања апликација, анализа и приказ података прикупљених у току мерења.

Предности дистрибуираног лансера леже у његовој релативно једноставној корисничкој спрези, поједностављењу покретања апликација на више рачунара и аутоматизацији процеса покретања, тестирања и валидације апликација писаних за окружење MPT-FLA.Такође ДЛ смањује могућност покретања погрешне апликације или прослеђивања погрешних параметара апликацији што се врло лако може десити при тестирању више апликација на великом броју рачунара. Још једна предност дистрибуираног лансера је његова преносивост и што је за његово покретање неопходан само Python интерпретер, како ДЛ користи само модуле који долазе са свим инсталацијама Python програмског језика и претраживач у случају покретања главног чвора, без потребе за додатном инсталацијом притом подржавајући како Windows тако и Linux базирана окружења. Главна мана дистрбуираног лансера је ограниченост мерења перформанси само на време извршења.

Нека од могућих унапређења система би свакако био додатак енкрипције и аутентификације при комуникацији између чворова, увођења механизама за отпорност на отказе, праћење отказа система као и аутоматизована дистрибуција изворног кода MPT-FLA апликација на жељене рачунаре које користе апликацију дистрибираног лансера.

7. Литература

[1] S. E. Deering, RFC 1112: Host extensions for IP multicasting, IETF Datatracker, Доступно на адреси: https://datatracker.ietf.org/doc/html/rfc1112#appendix-I (приступљено 10. јуна 2023)

[2] H.B. McMahan, E. Moore, D. Ramage, S. Hampson, B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in Proc. of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS), JMLR: W&CP volume 54, pp. 1-10, 2017.

[3] P. Kairouz, H. B. McMahan, B Avent, A. Bellet et al., "Advances and Open Problems in Federated Learning", 2019, doi: https://doi.org/10.48550/arxiv.1912.04977.

[4] M. Popovic, M. Popovic, I. Kastelan, M. Djukic, and S. Ghilezan, A Simple Python Testbed for Federated Learning Algorithms, 2023, in: Proceedings of the 2023 Zooming Innovation in Consumer Technologies Conference, 2023, pp. 148-153, https://doi.org/10.1109/ZINC58345.2023.10173859.

[5] M. Popovic, M. Popovic, I. Kastelan, M. Djukic, and I. Basicevic, "MicroPython Testbed for Federated Learning Algorithms" , 2024, doi: https://doi.org/10.48550/arxiv.2405.09423.

[6] IPPROTO_IP socket options. Доступно на адреси: https://docs.microsoft.com/en-us/windows/win32/winsock/ipproto-ip-socket-options, (приступљено 5. јуна 2023)

[7] ip - Linux IPv4 protocol implementation. Доступно на адреси:: https://linux.die.net/man/7/ip (приступљено 5. јуна 2023)

[8] M. Popovic, Github repozitorijum PTB-FLA. Доступно на адреси:i: https://github.com/miroslav-popovic/ptbfla, (приступљено 20. јуна 2023.)