BRAINE

BRAINE - Big data Processing and Artificial Intelligence at the Network Edge

Project Title:	**BRAINE - Big data Processing and Artificial Intelligence at the Network Edge**
Contract No:	876967 – BRAINE
Instrument:	ECSEL Research and Innovation Action
Call:	H2020-ECSEL-2019-2-RIA
Start of project:	1 May 2020
Duration:	36 months

Deliverable No: D4.3

Final project report on the status of WP4 Part1

Due date of deliverable:	30 August 2022
Actual submission date:	06 January 2023
Version:	1.0

ECSEL Joint Undertaking

Project funded by the European Community under the H2020 Programme for Research and Innovation.

Project ref. number	876967
Project title	BRAINE - Big data Processing and Artificial Intelligence at the Network Edge

Deliverable title	Final project report on the status of WP4-Part1
Deliverable number	D4.3
Deliverable version	Version 1.0
Previous version(s)	-
Contractual date of delivery	31 Sept 2022
Actual date of delivery	31 December 2022
Deliverable filename	Final project report on the status of WP4-Part1
Nature of deliverable	Report
Dissemination level	PU
Number of pages	45
Work package	WP4
Task(s)	T4.1, T4.2, T4.3, T4.4
Partner responsible	DELL
Author(s)	Javad Chamanara (LUH), Ahmed Khalid (DELL), Sean Ahearne (DELL), Adam Flizikowski (ISW), Munjure Mowla (ISW), Vojtěch Janů (CTU), Hemant Mehta (UCC), Martin Ron (FS), Roberto Bifulco (NEC)
Editor	Javad Chamanara (LUH)

Abstract	Report on the set of BRAINE platform components developed in WP4 for data management, 5G vRAN, factory motif discovery, and AI profiling.
Keywords	WP4, Data management, 5G, Industry 4.0, AI

Copyright

Deliverable history

Version	Date	Reason	Revised by
00	01.05.2021	Document Outline	Javad Chamanara
01	06.10.2022	List of Components	Javad Chamanara
02	20.12.2022	WP level review, executive summary, and conclusion	Javad Chamanara
1.0	30.12.2022	Final Revision	Javad Chamanara

List of abbreviations and Acronyms

Abbreviation	Meaning
5G	5th Generation
AI	Artificial Intelligence
API	Application Programming Interface
CPU	Central Processing Unit
CU	Centralized Unit
DU	Distributed Unit
EMDC	Edge Mobile Data Center
EPC	Evolved Packet Core
EU	European Union
GDPR	General Data Protection Regulation
GPU	Graphics Processing Unit
IoT	Internet of Things
IT	Information Technology
KPI	Key Performance Indicator
MES	Manufacturing Execution Systems
MOD	MOtif Discovery
RAN	Radio Access Network
UE	User Equipment
USRP	Universal Software Radio Peripheral
DoA	Description of Action
K8S	Kubernetes
DB	Database
DLCM	Data LifeCycle Manager
JSON	JavaScript Object Notation
GUI	Graphical User Interface
REST	Representational State Transfer
VNF	Virtual Network Function
MQTT	Message Queuing Telemetry Transport

Table of Contents

List of Figures

1. Executive summary

Work Package 4 delivers 15 software components that not only interact with each other to build part of the EMDC platform, but also, they are utilized by the use-cases. This deliverable is the technical report that communicates the final outcome the first part 1 of the WP4 components. In the first part, 7 components are described in details, their state-of-the-art as well as the advancements they contribute are presented, and for each component a list of performance indicators have been introduced, measured, and reported. The listed components include:

- Data management framework for AI environments that provides applications at the edge nodes with access to data while ensuring data availability, co-location, replication, provenance and other capabilities as defined in the task
- Data handling in edge nodes consisting of dataset generation to serve as inputs for AI-bound use cases, as well as software systems for their resource-efficient ingestion and processing at the edge.
- vRAN optimizations based on AI/ML use-cases analysis to identify optimization strategies for vRAN, based on the contextual information produced by the AI/ML models delivered in the work package.

This deliverable presents the technical details, the state-of-the-art, the advancements, and the performance evaluation of the listed WP4 components.

2. Introduction

WP4 has produced 14 components, which their specification, features, and performance will be reported in two deliverables, D4.3 (this document) and D4.4. Based on the DoA's requirements, D4.3 includes the following sections and results, which cover 8 of the components, belonging to the following categories:

- **A data management framework for AI environments:** This deliverable section will be a software system that provides applications at the edge nodes with access to data while ensuring data availability, co-location, replication, provenance and other capabilities as defined in the task.
- **Encapsulation of data sovereignty with data access APIs:** This deliverable section will be a software system that is able to encapsulate a given dataset so that its usages a redefined, agreed upon in advance, and controlled during the utilization time. The software guarantees that the data can only be used for the purposes defined and only by the target audience. The software provides access to the contained data only through its defined APIs.
- **Data handling in edge nodes:** This deliverable section will consist of dataset generation to serve as inputs for AI-bound use cases, as well as software systems for their resource-efficient ingestion and processing at the edge.
- **vRAN optimizations based on AI/ML use-cases analysis:** This deliverable section will identify optimization strategies for vRAN, based on the contextual information produced by the AI/ML models delivered in the work package.

The rest of the components will be reported in the same way in D4.4.

To clarify the components' distribution among the two deliverables, table below indicates partners responsible for each of the components and the deliverable (D4.3 or D4.4) each component will be reported in.

Partner	**Components**	**Deliverable**
ISW	vRAN with adjustments (C4.12)	D4.3
FS	Motif Discovery Tool (C4.11)	D4.3
UCC	Data Placement (C4.9)	D4.3
DELL	Data lifecycle manager (C4.1)	D4.3
	Policy manager (C4.2)	D4.3
	Global File System (C4.3)	D4.3
NEC	AI platform profiling engine (C4.13)	D4.3
LUH	Active Data Product (C4.4)	D4.4
	Monitoring Dashboard (C4.8)	D4.4
SIC	Catalyzr tool (C4.5)	D4.4
ECC	Authoring tool (C4.6)	D4.4
	Service Orchestrator (C4.7)	D4.4
IMC	Healthcare Assisted Living (C4.10)	D4.4
MLNX	Network Telemetry Framework (C4.14)	D4.4

An architecture diagram of where these components integrate as part of the overall BRAINE architecture can be seen in … below highlighted in green. Note that while C4.11 and C4.13 are developed by use case partners, the features provided are capable of acting as a service which could be consumed by any other use cases where desired, hence their inclusion in WP4. C4.12 integrates as part of the overall workload placement framework, influencing workload distribution of vRAN components based on the overall systems current state.

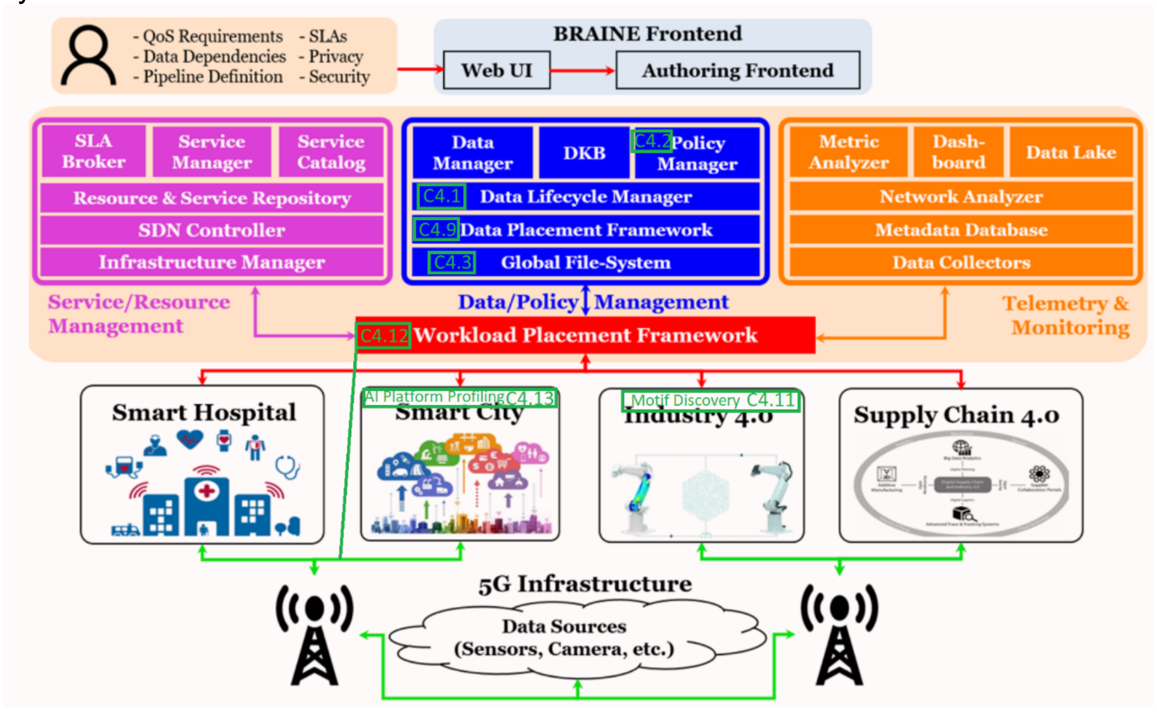


Figure 2.1 BRAINE Architecture Diagram showing components developed in D4.3

3. Status of the components

3.1. Data lifecycle manager (C4.1)

Component ID	Component Name	Development	Owner
C4.1	Data lifecycle manager	100%	DELL

GitLab Repository: https://gitlab.com/braine/wp4-datalifecyclemanager

Containerized: Yes

Registered on BRAINE platform image registry: Yes

Deployed as a pod and is functional on BRAINE platform: Yes

Integrated with other platform components: Yes – Apache Ranger/Ozone + K8S

Status: Completed

3.1.1. Technical description

The Data Lifecycle Manager (DLCM) is designed as a means to immutably track the use of a users' data on the BRAINE platform. This is relevant for both user data privacy, and government authority data regulations. For any platform that manages external customer data, it is important that a verifiable method for monitoring and auditing any and all events that occurred with a specific users' data exists. The DLCM enables this functionality in conjunction with the policy manager, global filesystem, and Kubernetes.

As part of a standard deployment of the BRAINE platform, 2 sets of data policies can be described. One set is defined according to EU GDPR best practices and guidelines for data management. The other set is user-configurable polices to allow/deny data sharing and access to other users and services as requested by the user. These polices are stored by Ranger and as the Apache Ozone global filesystem is accessed, any data event such as creation, access, modification, and deletion are captured by Ozone and Ranger and stored in its Audit DB, along with the defined policies in its Policy DB.

One potential issue with Rangers Audit DB is that all events are stored in an unencrypted database, which could easily be attacked and/or modified by a malicious user or software. This makes it an unreliable and unusable method of auditing how a users' data was managed on the platform on its own. As such a method which can guarantee the integrity of the audit DB needs to be designed. An immutable ledger is a transaction log where once the transaction is written in an entry, it cannot be deleted or modified in anyway way. A blockchain is the most common example of an application of immutable ledgers.

The IOTA blockchain was chosen as the immutable ledger to use to provide this feature for the platform, due to its low processing requirements for adding new transactions and data to the blockchain. It's low compute requirements give it good flexibility for use as an immutable ledger in edge environments. It can also be easily distributed across edge nodes, and functions as a fully private blockchain distributed across BRAINE EMDC(s) in this application.

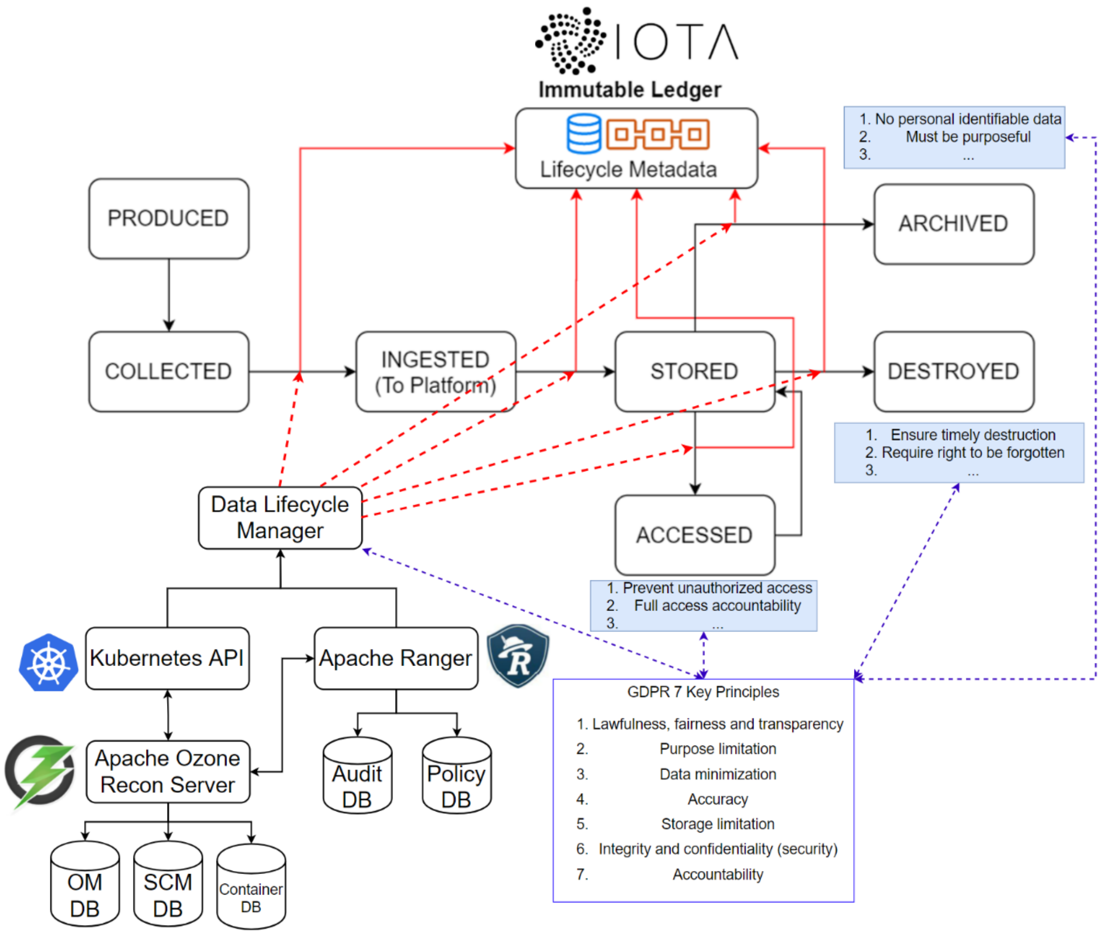
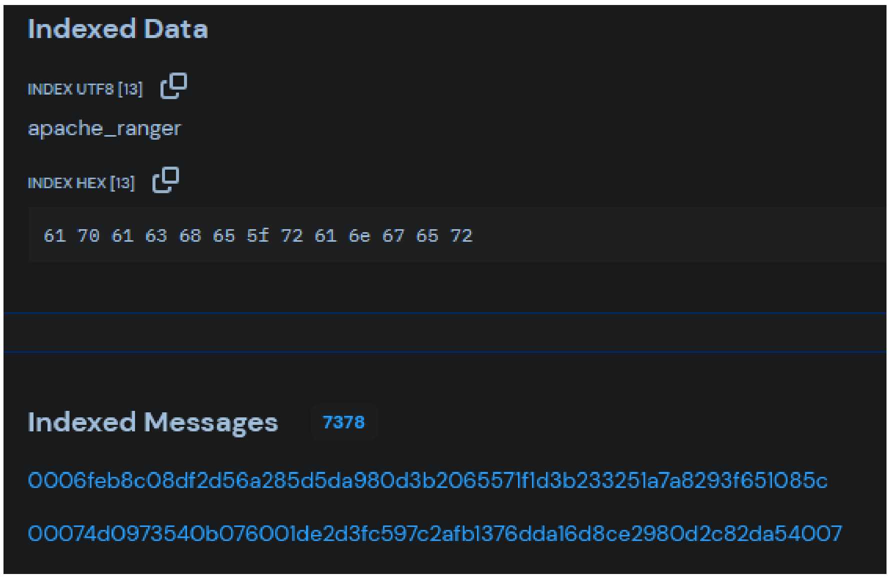


Figure 3.1 Data Lifecycle Manager Architecture

The DLCM thus takes in the auditory information from Apache ranger (plus the associated data users from Kubernetes), and formats and converts that data for ingestion as a transaction to the IOTA blockchain ledger. The DLCM maintains an active sync state between the IOTA blockchain and Apache Ranger audit DB and will actively monitor for intrusion detection and system attacks by periodic state comparisons between both systems. In order to maintain blockchain state at least one message per second is required to be written to the blockchain to verify its synchronization. In order to easily identify these “blank” messages from those which are useful, an index value can be attached to a message in the IOTA client when sending data to be written to the blockchain. In this case we can simply choose the index “apache_ranger” to signify the where the data originates from and enable a user to quickly filter for messages that only pertain to a particular application they are looking for. This method can also be applied for other BRAINE components and use cases that would benefit from an immutable ledger. An example of the searching and filtration of indexed messages can be seen in Figure 3.2 below, showing 7378 indexed data lifecycle events recorded.

Figure 3.3 shows the data contained within one of these indexed messages, which is formatted to be JSON readable. Note the similarities to how Ranger captures data lifecycle events in C4.2. The component health, sync status, and message rate can also be seen in the top right. Hornet is the term given to the IOTA blockchains main service component that maintains the front-end API’s and message/block writing. Note that to reduce overall storage consumption the hash values of Ranger entries could also be stored instead of all metadata. This comes with the disadvantage of only detecting a tampered entry, but what values were actually modified will not be known.


Indexed Data

INDEX UTF8 [13]

apache_ranger

INDEX HEX [13]

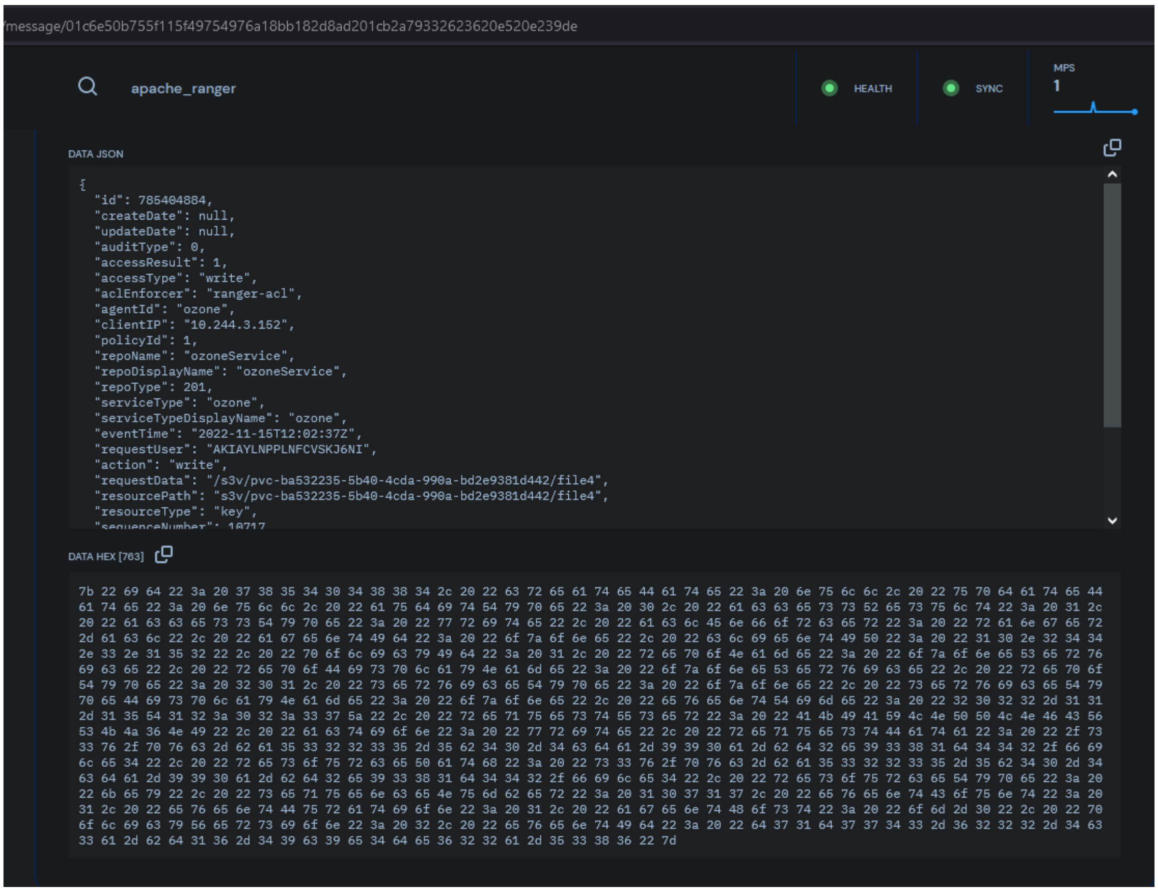
```
61 70 61 63 68 65 5f 72 61 6e 67 65 72
```

Indexed Messages 7378

0006feb8c08df2d56a285d5da980d3b2065571f1d3b233251a7a8293f651085c

00074d0973540b076001de2d3fc597c2afb1376dda16d8ce2980d2c82da54007

Figure 3.2 Indexed Ranger Entries in IOTA Blockchain


/message/01c6e50b755f115f49754976a18bb182d8ad201cb2a79332623620e520e239de

apache_ranger

DATA JSON

```
{
  "id": 785404884,
  "createDate": null,
  "updateDate": null,
  "auditType": 0,
  "accessResult": 1,
  "accessType": "write",
  "aclEnforcer": "ranger-acl",
  "agentId": "ozone",
  "clientIP": "10.244.3.152",
  "policyId": 1,
  "repoName": "ozoneService",
  "repoDisplayName": "ozoneService",
  "repoType": 201,
  "serviceType": "ozone",
  "serviceTypeDisplayName": "ozone",
  "eventTime": "2022-11-15T12:02:37Z",
  "requestUser": "AKIAYLNPPLNFCVSKJ6NI",
  "action": "write",
  "requestData": "/s3v/pvc-ba532235-5b40-4cda-990a-bd2e9381d442/file4",
  "resourcePath": "s3v/pvc-ba532235-5b40-4cda-990a-bd2e9381d442/file4",
  "resourceType": "key",
  "sequenceNumber": 10717
```

DATA HEX [763]

```
7b 22 69 64 22 3a 20 37 38 35 34 30 34 38 38 34 2c 20 22 63 72 65 61 74 65 44 61 74 65 22 3a 20 6e 75 6c 6c 2c 20 22 75 70 64 61 74 65 44
61 74 65 22 3a 20 6e 75 6c 6c 2c 20 22 61 75 64 69 74 54 79 70 65 22 3a 20 30 2c 20 22 61 63 63 65 73 73 52 65 73 75 6c 74 22 3a 20 31 2c
20 22 61 63 63 65 73 73 54 79 70 65 22 3a 20 22 77 72 69 74 65 22 2c 20 22 61 63 6c 45 6e 66 6f 72 63 65 72 22 3a 20 22 72 61 6e 67 65 72
2d 61 63 6c 22 2c 20 22 61 67 65 6e 74 49 64 22 3a 20 22 6f 7a 6f 6e 65 22 2c 20 22 63 6c 69 65 6e 74 49 50 22 3a 20 22 31 30 2e 32 34 34
2e 33 2e 31 35 32 22 2c 20 22 70 6f 6c 69 63 79 49 64 22 3a 20 31 2c 20 22 72 65 70 6f 4e 61 6d 65 22 3a 20 22 6f 7a 6f 6e 65 53 65 72 76
69 63 65 22 2c 20 22 72 65 70 6f 44 69 73 70 6c 61 79 4e 61 6d 65 22 3a 20 22 6f 7a 6f 6e 65 53 65 72 76 69 63 65 22 2c 20 22 72 65 70 6f
54 79 70 65 22 3a 20 32 30 31 2c 20 22 73 65 72 76 69 63 65 54 79 70 65 22 3a 20 22 6f 7a 6f 6e 65 22 2c 20 22 73 65 72 76 69 63 65 54 79
70 65 44 69 73 70 6c 61 79 4e 61 6d 65 22 3a 20 22 6f 7a 6f 6e 65 22 2c 20 22 65 76 65 6e 74 54 69 6d 65 22 3a 20 22 32 30 32 32 2d 31 31
2d 31 35 54 31 32 3a 30 32 3a 33 37 5a 22 2c 20 22 72 65 71 75 65 73 74 55 73 65 72 22 3a 20 22 41 4b 49 41 59 4c 4e 50 50 4c 4e 46 43 56
53 4b 4a 36 4e 49 22 2c 20 22 61 63 74 69 6f 6e 22 3a 20 22 77 72 69 74 65 22 2c 20 22 72 65 71 75 65 73 74 44 61 74 61 22 3a 20 22 2f 73
33 76 2f 70 76 63 2d 62 61 35 33 32 32 33 35 2d 35 62 34 30 2d 34 63 64 61 2d 39 39 30 61 2d 62 64 32 65 39 33 38 31 64 34 34 32 2f 66 69
6c 65 34 22 2c 20 22 72 65 73 6f 75 72 63 65 50 61 74 68 22 3a 20 22 73 33 76 2f 70 76 63 2d 62 61 35 33 32 32 33 35 2d 35 62 34 30 2d 34
63 64 61 2d 39 39 30 61 2d 62 64 32 65 39 33 38 31 64 34 34 32 2f 66 69 6c 65 34 22 2c 20 22 72 65 73 6f 75 72 63 65 54 79 70 65 22 3a 20
22 6b 65 79 22 2c 20 22 73 65 71 75 65 6e 63 65 4e 75 6d 62 65 72 22 3a 20 31 30 37 31 37 2c 20 22 65 76 65 6e 74 43 6f 75 6e 74 22 3a 20
31 2c 20 22 65 76 65 6e 74 44 75 72 61 74 69 6f 6e 22 3a 20 31 2c 20 22 61 67 65 6e 74 48 6f 73 74 22 3a 20 22 6f 6d 2d 30 22 2c 20 22 70
6f 6c 69 63 79 56 65 72 73 69 6f 6e 22 3a 20 32 2c 20 22 65 76 65 6e 74 49 64 22 3a 20 22 64 37 31 64 37 37 34 33 2d 36 32 32 32 2d 34 63
33 61 2d 62 64 31 36 2d 34 39 63 39 65 34 64 65 36 32 32 61 2d 35 33 38 36 22 7d
```

Figure 3.3 Data contained within an IOTA blockchain message

3.1.2. State of the Art (SOTA)

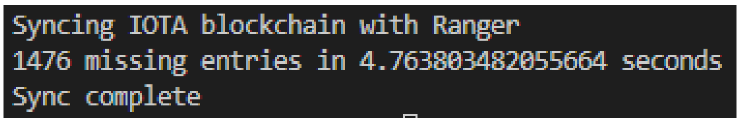
To the author’s knowledge, this is the first time that an immutable ledger has been integrated with Apache Ranger for the purposes of data lifecycle management and monitoring of GDPR compliance. Publications as recent as 2021 discuss the possibility of implementing an immutable ledger blockchain for this purpose but did not proceed feeling limitations of blockchain size would be too severe[27]. Our solution consumes ~250GB over 10 years plus the lifecycle metadata and number of replicas. We deem this to be acceptable considering current node storage capacities and future growth in storage capacity. Other publications mention that while Ranger forms a good base for monitoring data platform GDPR compliance, it requires supporting components in order to provide verifiability and auditability to an acceptable EU standard[i].

3.1.3. Advancements

As mentioned in section 3.1.2, the integration of Ranger, IOTA, Ozone, and Kubernetes to form this data lifecycle manager is unique and has not been previously performed according to current literature. The integration of a blockchain-based immutable ledger to form part of an overall data privacy and security monitoring platform has also been discussed and designed, but seemingly not implemented. A data management platform based entirely on the blockchain does exist however[28]. This represents new advancements in the field of data lifecycle management within the BRAINE project and with the open-source nature of the supporting components this implementation can be distributed to the public and continue to be further improved on in future. While the system is functional, improvements to performance, scalability, and additional features are possible.

3.1.4. Performance Evaluations and comparisons

Performance evaluations for this component come in the form of analysis of the IOTA blockchain and lifecycle manager to maintain synchronization with Apache Ranger. For this test we investigate the maximum number of entries and hence messages/blocks are written within a given time. In Figure 3.4 we start from a state of 1476 data lifecycle events in Apache Ranger and an uninitialized IOTA blockchain. We then start the synchronization service and measure the time taken to sync all entries between IOTA and Ranger. It can be seen here that it synchronized the 1476 entries in 4.7638 seconds. Giving a messaging and block writing rate of approx. 310 messages per second. As mentioned in the previous section there is no direct comparison for this evaluation to a similar standard, however [28] gives the closest approximation, with a write throughput of 167 transactions per second.



```
Syncing IOTA blockchain with Ranger
1476 missing entries in 4.763803482055664 seconds
Sync complete
```

Figure 3.4 Testing maximum message rate from Ranger to IOTA

The messaging rate can also be monitored in real-time via the IOTA Hornet service dashboard. This can be seen below in Figure 3.5 at the top right, showing a message rate of 306 messages per second under load. Note the total system data storage and RAM usage also.

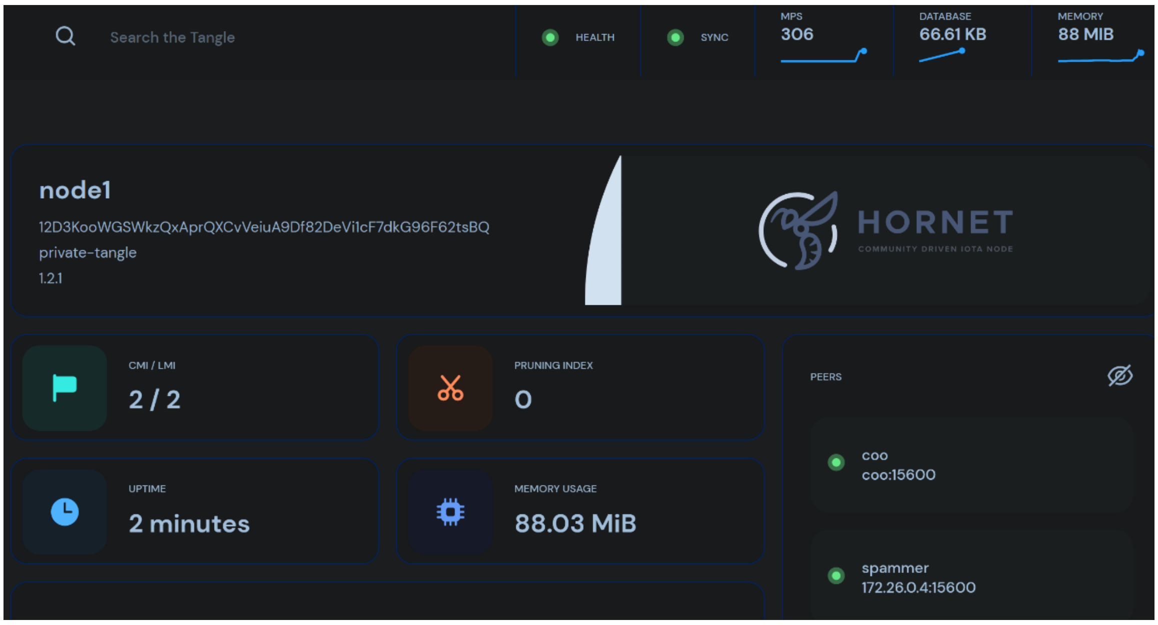
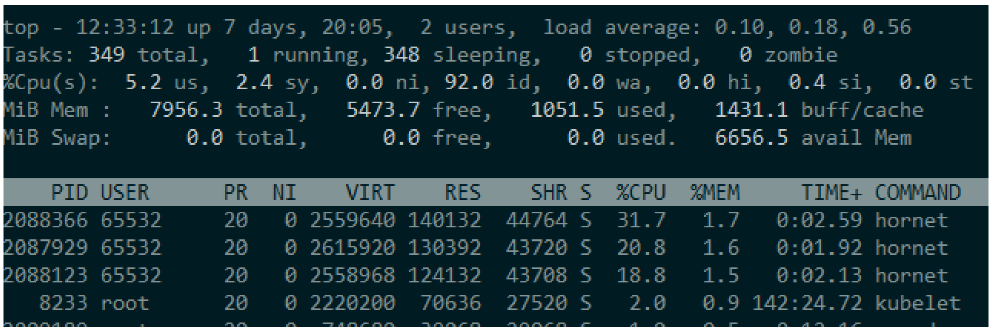

Search the Tangle
HEALTH
SYNC
MPS
306
DATABASE
66.61 KB
MEMORY
88 MIB
node1
12D3KooWGSWkzQxAprQXCvVeiuA9Df82DeVilcF7dkG96F62tsBQ
private-tangle
1.2.1
HORNET
COMMUNITY DRIVEN IOTA NODE
CMI / LMI
2 / 2
PRUNING INDEX
0
PEERS
coo
coo:15600
spammer
172.26.0.4:15600
UPTIME
2 minutes
MEMORY USAGE
88.03 MiB

Figure 3.5 Message Per Second (MPS) Rate as reported by IOTA under load

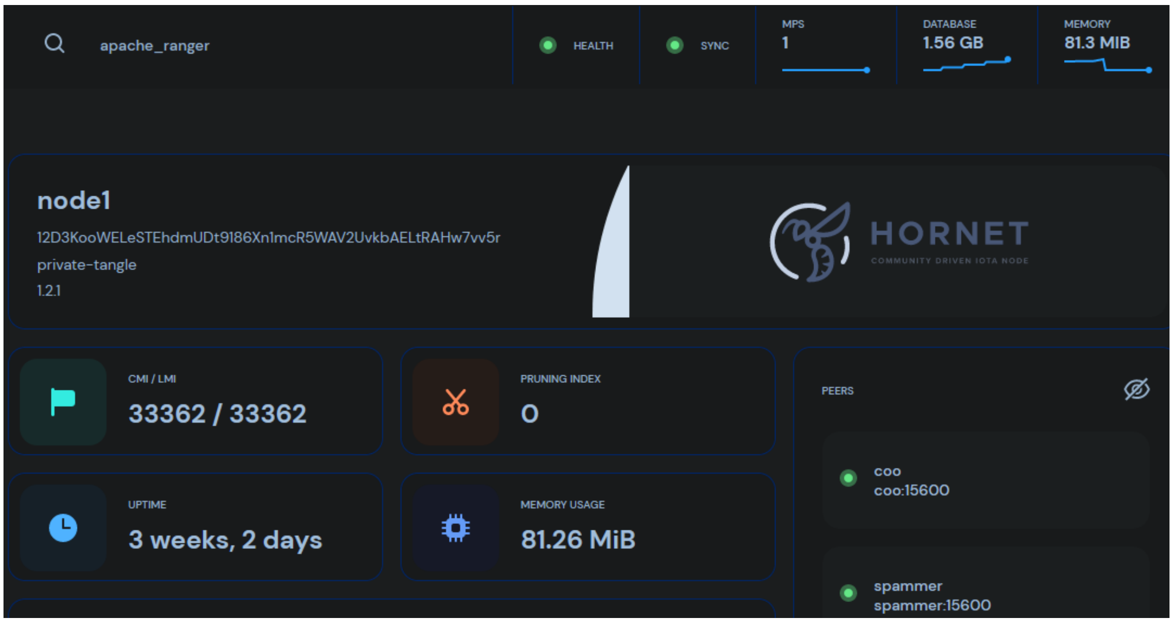
The CPU load rate of the ingestion node (i.e. the server) can be seen in Figure 3.6. The ingestion node is parallelized into 4 threads/processes, with only 3 of 4 threads being utilized and none to maximum load. This implies that the performance limit in I/O load is currently due to the client side (the data lifecycle manager service) which is single threaded. This service can be multi-threaded in future to improve performance above 1200 MPS, however 300 MPS was already deemed sufficient for demonstration and current BRAINE use case requirements. Note also the RAM usage of up to 140MB per ingestion process under load.


```
top - 12:33:12 up 7 days, 20:05,  2 users,  load average: 0.10, 0.18, 0.56
Tasks: 349 total,   1 running, 348 sleeping,   0 stopped,   0 zombie
%Cpu(s):  5.2 us,  2.4 sy,  0.0 ni, 92.0 id,  0.0 wa,  0.0 hi,  0.4 si,  0.0 st
MiB Mem :   7956.3 total,   5473.7 free,   1051.5 used,   1431.1 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.   6656.5 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
2088366 65532     20   0 2559640 140132  44764 S  31.7   1.7   0:02.59 hornet
2087929 65532     20   0 2615920 130392  43720 S  20.8   1.6   0:01.92 hornet
2088123 65532     20   0 2558968 124132  43708 S  18.8   1.5   0:02.13 hornet
   8233 root      20   0 2220200  70636  27520 S   2.0   0.9 142:24.72 kubelet
```

Figure 3.6 IOTA server-side load at 300 MPS

Figure 3.7 shows the long-term stability of the system and resource profile over time. The CPU usage of the DLCM components while idling at 1MPS is below 1% usage, with RAM usage below 512MB in total. Each ingestion process cleans its RAM pool once it reaches ~140MB. This can be seen in Figure 3.7 where it drops to 81.3MB. It can also be seen in Fig 3.7 that the storage usage of the blockchain has grown to 1.56GB over 3 weeks and 2 days, this includes the DLCM data (7378 entries). It can then be estimated that the total storage consumption of the blockchain itself (less data) to be ~250GB over 10 years. This is an acceptable range for an EMDC edge node, where multi-TB NvME storage can be expected on each node.


apache_ranger

HEALTH

SYNC

MPS 1

DATABASE 1.56 GB

MEMORY 81.3 MIB

node1
12D3KooWELeSTEhdmUDt9186Xn1mcR5WAV2UvkbAELtRAHw7vv5r
private-tangle
1.2.1

CMI / LMI 33362 / 33362

PRUNING INDEX 0

UPTIME 3 weeks, 2 days

MEMORY USAGE 81.26 MiB

PEERS

coo
coo:15600

spammer
spammer:15600

Figure 3.7 Long term stability and resource usage

3.2. Policy manager (C4.2)

Component ID	Component Name	Development	Owner
C4.2	Policy manager	100%	DELL

GitLab Repository: https://gitlab.com/braine/dmf

Containerized: Yes

Registered on BRAINE platform image registry: Yes

Deployed as a pod and is functional on BRAINE platform: Yes

Integrated with other platform components: Yes. Data Storage and Data Lifecycle Manager

Status: Development of the policy manager is complete with containerized application running as pods on the BRAINE platform along with its dependencies. The manager is based on Apache Ranger and is integrated with Apache Ozone which is the distributed data storage system of the BRAINE platform.

3.2.1. Technical description

The policy manager is a sub-component of the data lifecycle management component and performs three main tasks.

- It provides a way to administrators for creating and managing policies and store them in a machine-readable format.
- It serves as an authorization agent and in addition to managing policies it enforces them through a plugin. The policy manager ensures that any access to user data is in accordance with the policies set out by data controllers or administrators.
- The policy manager also creates an audit log of the access requests and provides them as an input to the data lifecycle manager (C4.1) This access log is comprised of a list of all access requests by all authorized as well as unauthorized clients, i.e., all allowed and rejected access requests are logged.

In BRAINE platform, Apache Ranger is selected as the policy manager and is used to managed and enforce access policies. A Web UI is configured to act as the Ranger admin and provide an interface for administrators to add or modify policies. Apache Ranger is also configured with the global file system (C4.3) through a Ranger plugin that replaces the default Ozone authorization class and connects with the Ranger admin.

In terms of granularity, the policies can be applied to all the hierarchical levels of Ozone file system i.e., volume, bucket, or a key. A key in Ozone represents an object or a file. Similarly, the audit logs can be enabled for each level of granularity, by applying them to the corresponding policies. The permission types include read, write, create, list, and delete. Both policies and audit logs can be configured to manage one or multiple of these permissions.

Additionally, the policy manager component also manages location-based policies or hardware and space requirements that are applied by the BRAINE data placement framework (C4.9) as a constraint when optimizing or selecting data nodes. These policies are managed using a key-value store which is accessed by the data placement framework when needed.

3.2.2. State of the Art (SOTA)

Section 3.4.2 mentions some of the existing works in literature that handle and apply policies when carrying out certain operations. The data management and placement framework in BRAINE goes beyond these SOTA approaches and handles more dynamic and versatile set of policies. The job of the policy manager is to ensure that these policies are stored in an efficient manner and made available to various management components when needed.

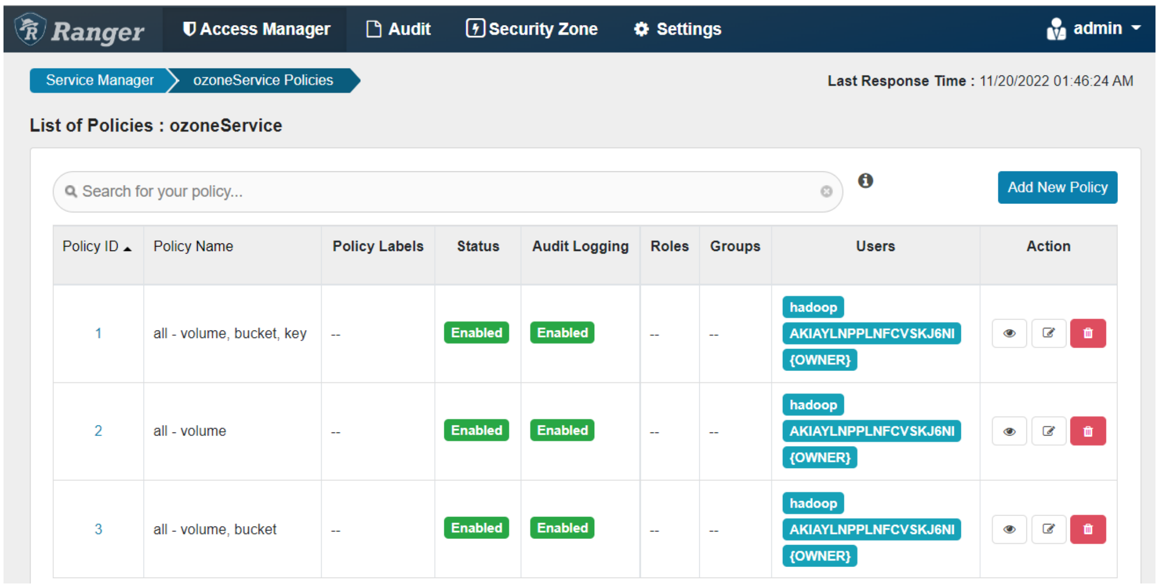
3.2.3. Advancements

One of the key features of the BRAINE policy manager is that it runs as a micro-service in BRAINE's Kubernetes-based platform and provides a fully containerized solution with well-defined endpoints and APIs for access by other system and management components. This is also true for all of its dependencies as well as integration with other BRAINE components, e.g., Ozone ranger plugin mentioned in Section 3.2.1.

Furthermore, the policy manager provides administrators with an ability to define and store, complex and granular policies, in a well-defined structure. E.g., for a particular application, an administrator may define policies and requirements in terms of the type of hardware needed, amount of storage space required, privacy or sharing constraints, list of other applications that are allowed or not allowed to share a node with this application. The ability to define such policies can extend the functionality of the data lifecycle management in a dynamic and heterogonous environment and allow applications to be managed and scheduled in a more efficient and secure way.

3.2.4. Performance Evaluations and comparisons

The figure below shows a sample list of policies created for Ozone storage.



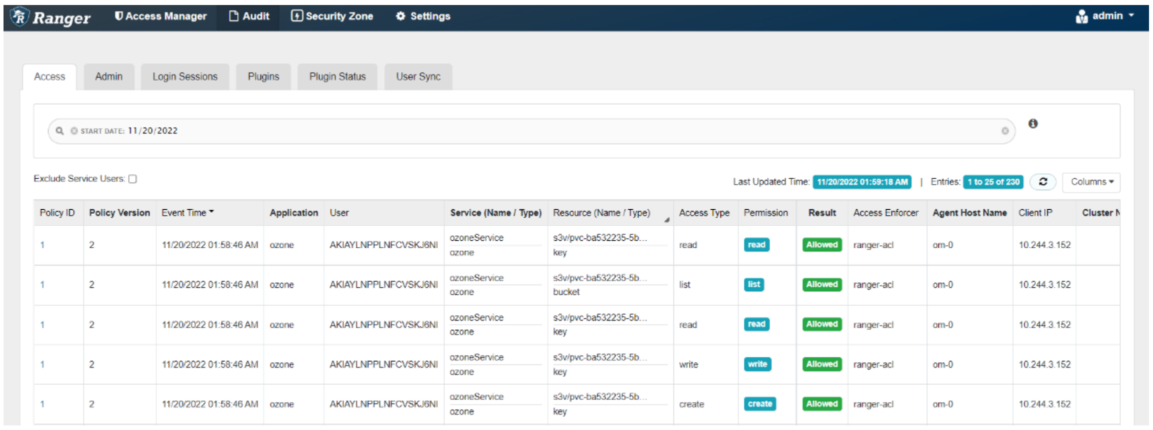
Last Response Time : 11/20/2022 01:46:24 AM

List of Policies : ozoneService

Policy ID	Policy Name	Policy Labels	Status	Audit Logging	Roles	Groups	Users	Action
1	all - volume, bucket, key	--	Enabled	Enabled	--	--	hadoop AKIAYLNPPLNFCVSKJ6NI {OWNER}	
2	all - volume	--	Enabled	Enabled	--	--	hadoop AKIAYLNPPLNFCVSKJ6NI {OWNER}	
3	all - volume, bucket	--	Enabled	Enabled	--	--	hadoop AKIAYLNPPLNFCVSKJ6NI {OWNER}	

Figure 3.8 An example of policy definition

As the figure shows, the provided interface allows adding policies on a highly granular level in Ozone in terms of the files or directories as well as users or groups of users. Similarly, the access logs are also generated with high granularity, as shown in the figure below.



START DATE: 11/20/2022

Exclude Service Users: ☐

Last Updated Time: 11/20/2022 01:59:18 AM | Entries: 1 to 25 of 230

Policy ID	Policy Version	Event Time	Application	User	Service (Name / Type)	Resource (Name / Type)	Access Type	Permission	Result	Access Enforcer	Agent Host Name	Client IP	Cluster N
1	2	11/20/2022 01:58:46 AM	ozone	AKIAYLNPPLNFCVSKJ6NI	ozoneService ozone	s3v/pvc-ba532235-5b... key	read	read	Allowed	ranger-acl	om-0	10.244.3.152	
1	2	11/20/2022 01:58:46 AM	ozone	AKIAYLNPPLNFCVSKJ6NI	ozoneService ozone	s3v/pvc-ba532235-5b... bucket	list	list	Allowed	ranger-acl	om-0	10.244.3.152	
1	2	11/20/2022 01:58:46 AM	ozone	AKIAYLNPPLNFCVSKJ6NI	ozoneService ozone	s3v/pvc-ba532235-5b... key	read	read	Allowed	ranger-acl	om-0	10.244.3.152	
1	2	11/20/2022 01:58:46 AM	ozone	AKIAYLNPPLNFCVSKJ6NI	ozoneService ozone	s3v/pvc-ba532235-5b... key	write	write	Allowed	ranger-acl	om-0	10.244.3.152	
1	2	11/20/2022 01:58:46 AM	ozone	AKIAYLNPPLNFCVSKJ6NI	ozoneService ozone	s3v/pvc-ba532235-5b... key	create	create	Allowed	ranger-acl	om-0	10.244.3.152	

Figure 3.9 A screenshot of the policy definition UI

One of the key purposes of the policy manager, as mentioned in Section 3.2.1, is to provide a list of policies or audit logs to other system components in the BRAINE platform, e.g., data lifecycle manager (C4.1), global file system (C4.3) and data placement framework (C4.9). This was achieved successfully as is shown in the results and evaluation of the corresponding components, where large-scale experiments were conducted with hundreds of policy requests or audit log requests. The policy manager was able to successfully handle these requests and serve BRAINE components in an efficient and satisfactory manner (see results of the aforementioned components for further evaluation).

3.3. Global File System (C4.3)

Component ID	Component Name	Development	Owner
C4.3	Global File System	100%	DELL

GitLab Repository: https://gitlab.com/braine/dmf

Containerized: Yes

Registered on BRAINE platform image registry: Yes

Deployed as a pod and is functional on BRAINE platform: Yes

Integrated with other platform components: Yes

Status: The development of data storage system based on Apache Ozone is complete and the sub-components have been deployed as pods and services in the BRAINE platform. C4.3 is a storage solution that provides a unified filesystem and object store for applications and workloads running on the platform. It has been integrated with other system components such as data placement framework and policy manager, as well as use-case applications that require persistent storage. Further integration activities may be carried out in WP5, based on use-case requirements.

3.3.1. Technical description

The goal of the distributed data storage solution is to configure all the available storage space on BRAINE EMDC as a single file system, creating a distributed pool of storage resources, such that they can be accessed by different workloads dynamically. As the BRAINE platform is based on Kubernetes (K8s), the storage solution must also provide stateful K8s applications with a persistent volume (PV) to store their stateful data. This is achieved through Container Storage Interface (CSI). However, additional, and multiple access interfaces must also be provided for external applications or applications that are built with certain requirements e.g., S3 interface or Hadoop HDFS. Based on these and other storage system requirements for BRAINE's EMDC and use-cases, we compared different file systems, and selected Apache Ozone to build the storage system on. Ozone is a top-level Apache project designed to scale to billions of objects and manage thousands of nodes. In addition to integrating Ozone with policy manager (as explained in Section 3.2), there were two key features added to Apache Ozone to meet the requirements missing from the default Ozone packages.

Modification of Ozone source code for interaction with data placement framework

Ozone makes a data placement decision when it receives a request to store a file or object. By default, Ozone randomly selects nodes for data placement. This is not a suitable approach for applications with various constraints and we modify this behavior. We modify this step to call an external API which implements placement algorithm. The API call returns the list of data nodes on which the application's data is to be placed to satisfy the specified constraints. We implemented the call to the placement algorithm as an external API as this allows re-use of this component when integrating with other object stores than Ozone.

The Ozone implementation details are depicted in the following Figure. The Ozone Manager (OM) component receives requests to store files. OM forwards the request to the Storage Container Manager (SCM) as a block allocation request. The SCM forwards this request to the Block Manager (BM) component. We modified the BM to invoke the external placement API which returns a list of nodes to be used for placement. After this the BM uses the list obtained to send a file-creation request to the Container Manager. The Container Manager creates the blocks on the selected data nodes to store the files.

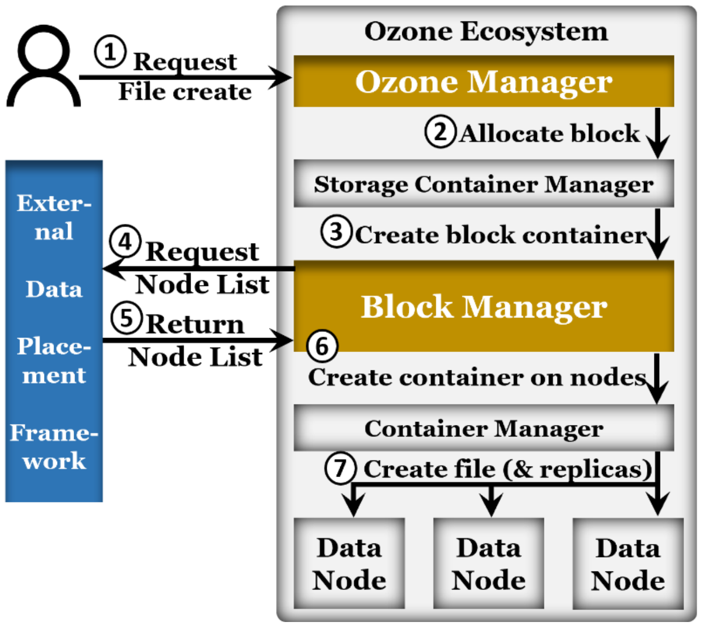


Figure 3.10 Ozone Internal Working and Call to Placement Framework

Multi-platform Docker images for Ozone and its dependencies

BRAINE EMDCs consist of nodes with different CPU architectures and a solution is required that can seamlessly integrate and manage all data nodes. Specifically, the EMDC consists of both ARM and x86 AMD nodes. Apache Ozone binaries available, as of the version 1.2.1, were built for AMD nodes, hence are insufficient for seamlessly integrating heterogeneous architectures, such that they can be accessed by different workloads dynamically and transparently, especially as the workloads move from one node to another.

This limitation comes from the container runtime interface, where a standard Docker image can only execute on the CPU architecture that it was built on. To concurrently deploy containers on different CPU architectures, a single image is required that contains variants for each architecture. Building such multi-platform images with Docker involves following steps:

- Build an image for each arch (arm64v8 & amd64 in BRAINE EMDCs), either using an emulator (which might be limited in functionality) or on native machines with the target architecture.
- Push each image to the image repository with correct architecture tags.
- Create a Docker manifest file and append each image. Push the multi-platform image to the repo.

Multi-platform images were prepared with these steps for Ozone and all of the Ozone dependencies (e.g., CSI drivers and provisioners). When deploying the image as a micro-service in K8s, the container runtime (Docker in this case) picks the image in manifest with the matching CPU architecture. The updated solution can seamlessly run on a multi-architecture Edge device such as the BRAINE EMDC.

Finally, to efficiently deploy all Apache Ozone components as containers, Kubernetes manifest files were used. The examples provided in Ozone distribution package are modified and deployed in the following way:

- Kubernetes has a volume called HostPath that can mount local file system of a node inside containers.
- HostPath is used to create Persistent Volumes for Ozone data node containers on each node.
- Note that this step is needed because Ozone, when running as a container on the local Edge nodes, can only utilize local storage if the storage is mounted as a volume in its container.
- Furthermore, node affinities are created for these Persistent Volumes to ensure correct remounting in case of rebooted (or crashed) Ozone components.
- Statefulsets, DaemonSets, and Services are created and applied for management, storage, S3 gateway and CSI components.

With the updated manifest files, Ozone is deployed as a micro-service and exposes multiple APIs to meet application requirements:

- K8s applications can directly mount Ozone storage as persistent volumes through CSI.
- Other applications can use S3 (AWS), Ofs (Hadoop), O3fs (Ozone) or shell for access operations, such as read, write, create, delete, or list.

An example use of Ozone storage class through Kubernetes is shown below where a persistent volume claim is created with Ozone storage class and an application pod can simply claim this Ozone persistent volume without any additional changes to the manifest file.

Table 3-1 An Example of Ozone storage class through Kubernetes

Volume Claim Manifest File	Pod Manifest File
apiVersion: v1	apiVersion: v1
kind: PersistentVolumeClaim	spec:
metadata:	template:
name: ozone-csi-test	**volumes:**
spec:	- **name: webroot**
storageClassName: ozone	**persistentVolumeClaim:**
accessModes:	**claimName: ozone-csi-test**
- ReadWriteOnce	containers:
resources:	- **volumeMounts:**
requests:	- **name: webroot**
storage: 1Gi	mountPath: /www
……	……
	……

3.3.2. State of the Art (SOTA)

Edge environments and BRAINE platform have certain requirements for the storage system. There are numerous solutions available in the literature and industry. Table 3-2 presents a qualitative comparison among some of the storage systems based on the key requirements.

Table 3-2 Qualitative comparison of storage systems based on the key requirements

Features/KPIs	HDFS	GlusterFS	CEPH	Ozone
Lightweight	*	**	***	****
Integration with BRAINE components	***	**	*	****
Multi-arch support (AMD, ARM)	***	**	****	*
Compatible access APIs	**	*	***	****
Location-based data policies	-	-	-	**
Scalable	**	**	***	***
Performance (latency)	*	***	*	***
Small files storage / Object & block storage	*	*	***	***
Fault tolerance/replication	***	***	***	***
Native Kubernetes integration	-	**	****	**
Decentralized management	**	**	*	***

Based on these requirements, Apache Ozone was chosen as the underlying file system and additional features were added, as described in the next section.

3.3.3. Advancements

They key advancements for an Edge storage system are based on the features shared in Section 3.3.1. An Edge storage system has certain requirements such as:

- A lightweight file system that can preserve scarce resources at Edge.
- A distributed system capable of:
  - Managing multiple nodes with heterogeneous architectures (x86/ARM)
  - Seamless data movement or migration across heterogeneous nodes, if/when required
- Support for persistent volumes in a micro-service environment (Kubernetes)
- Ability to configure location-based policies and enforce regulations e.g., GDPR
- Multiple access APIs. E.g., to support Hadoop-based access APIs for backward compatibility or S3 API access for S3-based applications

As mentioned in previous sections, some of these features were lacking in the state of the art solutions and were hence developed in this task. Specifically,

- Interaction with an external data placement framework to get a list of optimal or best data nodes for data placement.

- Seamless integration and management of heterogeneous data nodes under a single file system using multi-platform container images.
- Integration with policy manager and enforcer, capable of handling and managing advanced placement, access, privacy and security policies and requirements

Deployment of the storage system as a micro-service with potential scalability to billions of files or objects.

3.3.4. Performance Evaluations and comparisons

We analyze the performance of Apache Ozone and its main components (Amazon AWS CLI, Amazon S3 Gateway (S3G), Ozone Manager (OM) and Storage Container Manager (SCM). We also perform the analysis when data is written with encryption and without encryption. The results discuss the CPU usage and instructions executed by different Ozone components (AWS CLI, S3G, OM, SCM) while performing the write data.

We collected the data while writing files of different size (100KB, 500KB, 1MB, 10MB, 50MB, 100MB and 1GB).

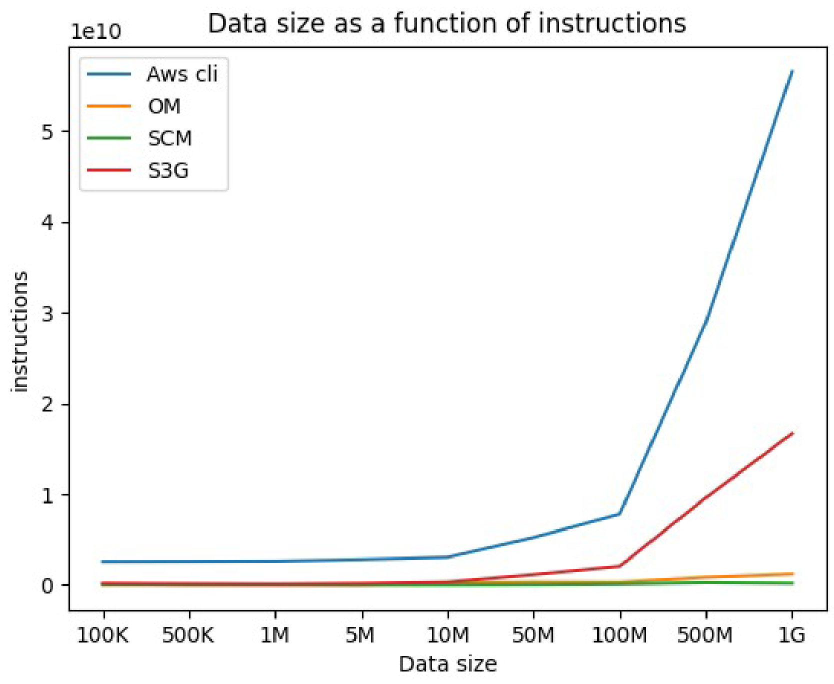


Figure 3.11 C4.3 performance evaluation, data size as a function of instructions

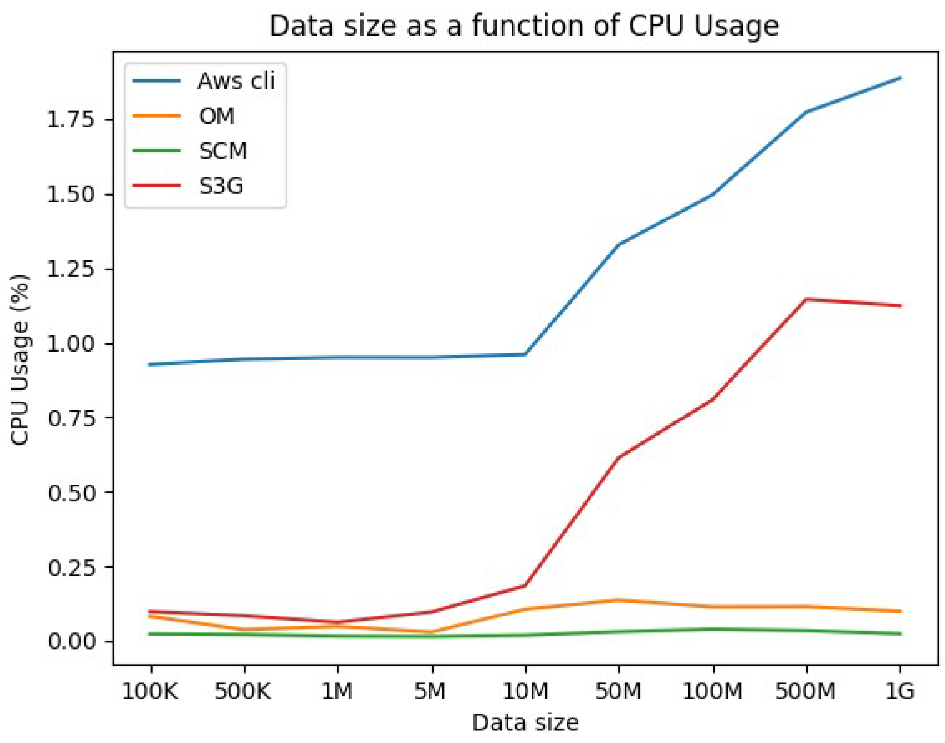


Figure 3.12 C4.3 performance evaluation, data size as a function of CPU usage

We observe that the while writing with encryption it takes 17% longer time and uses 30% more CPU as depicted in the following two figures.

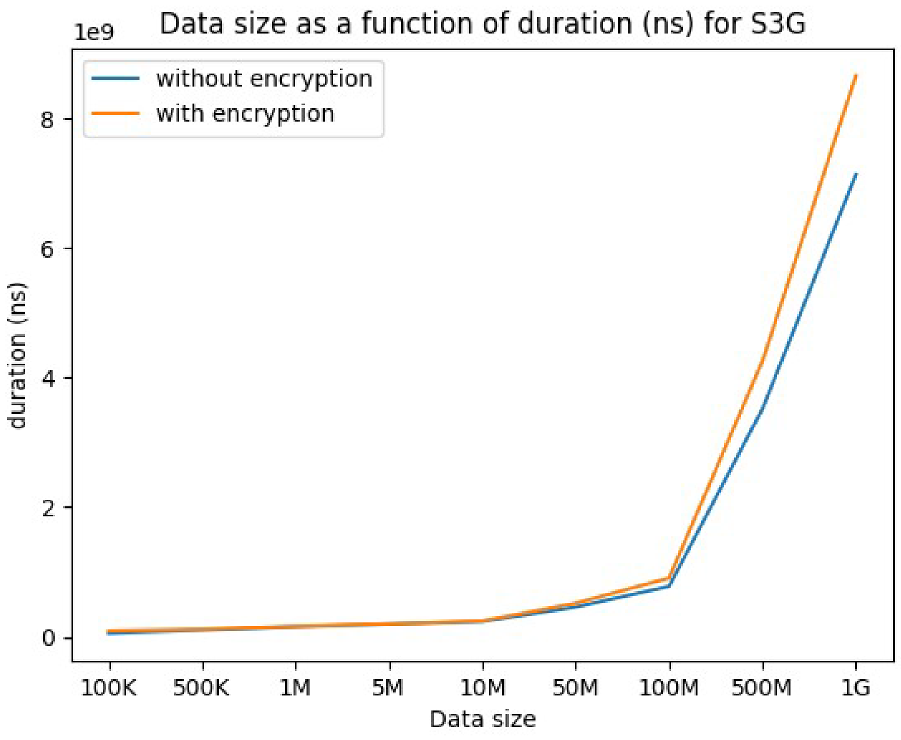


Figure 3.13 C4.3 performance evaluation, data size as a function of duration

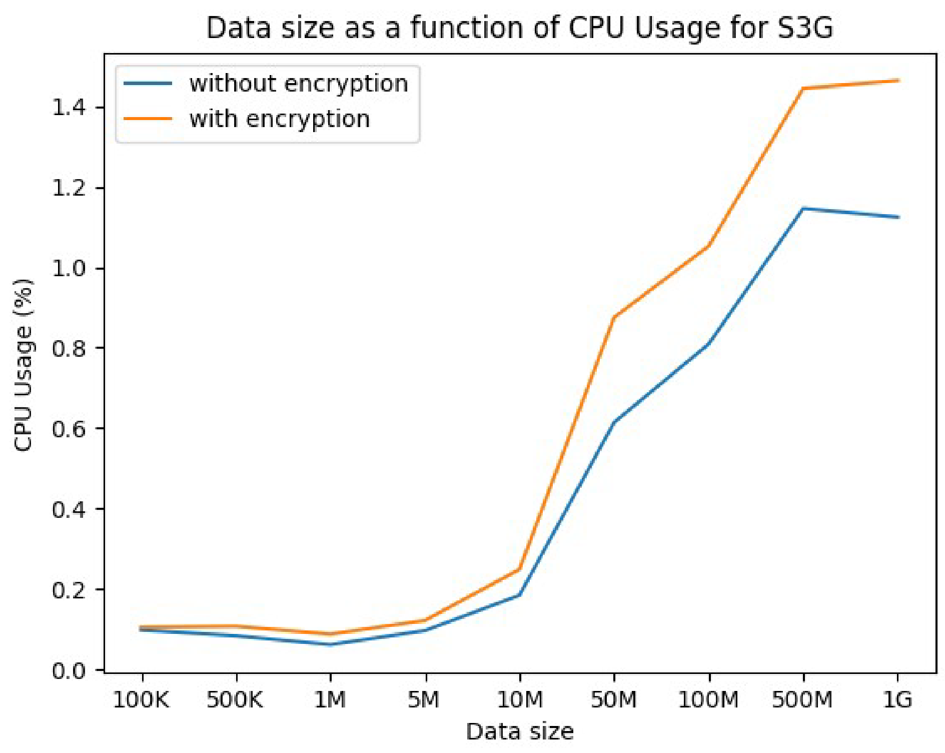


Figure 3.14 C4.3 performance evaluation, data size versus CPU usage for S3G

3.4. Data Placement (C4.9)

Component ID	Component Name	Development	Owner
C4.9	Data Placement	100%	UCC

GitLab Repository: https://gitlab.com/braine/c49dataplacementframework

Containerized: Y

Registered on BRAINE platform image registry: Y

Deployed as a pod and is functional on BRAINE platform: Y

Integrated with other platform components: Y – see diagram

Status: Developed an external data placement framework to optimize the number of nodes used while respecting all the constraints of various applications deployed on the cluster. Upon receiving a request to store the data, modified Apache Ozone invokes the placement API to suggest the list of suitable nodes to be used to store the data.

3.4.1. Technical description

This is a constraints-based data placement algorithm that is developed and implemented as an external component that is independent of the distributed object store (Apache Ozone). This enables the user to invoke the placement algorithm API for any potential workload on any available data store. This component is a sub-component of a larger system architecture designed to optimize and accelerate workloads in edge computing environments. Within this architecture this algorithm forms the basis of a software component known as the data placement framework. This component interacts with a number of other components in the system architecture using APIs to retrieve application data constraints and policy information, (forming the data management framework) and adapt the algorithm output to impact workload placement on edge nodes. A subset of these edge systems architectural components which the data placement algorithm interacts with, is shown in in the following figure.

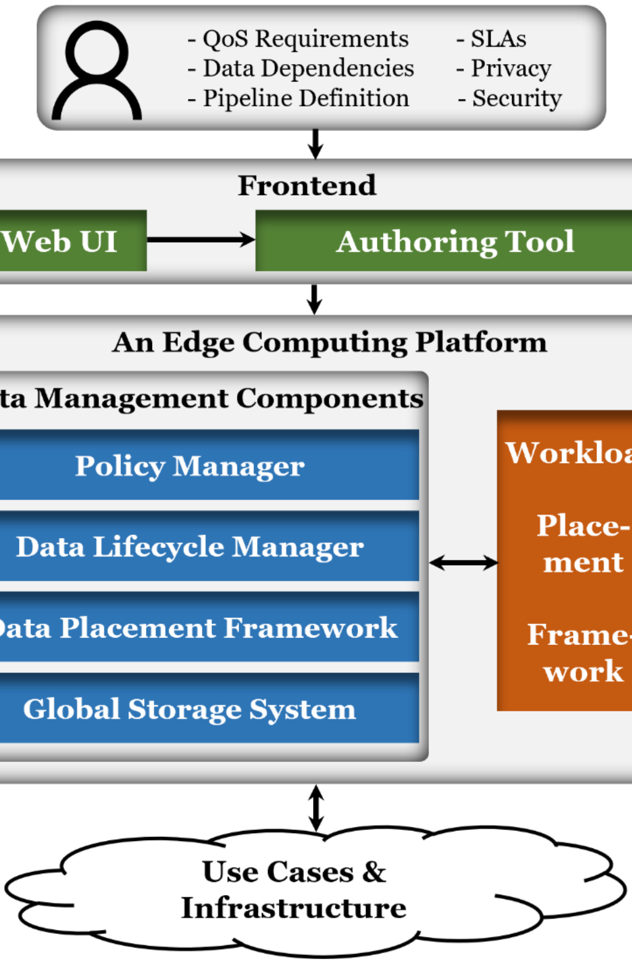


Figure 3.15 Data Placement Reference Architecture

The placement algorithm has two underlying components as:

- An optimization model that minimizes the number of data nodes while respecting all application constraints. The problem is formulated as an Integer Linear Program and solved using a Constrained Programming with Satisfiability methods (CP-SAT) solver. This solution enables the system to enforce the desired constraints using the minimum number of nodes. Lesser active nodes lead to lower energy consumption.
- A heuristic algorithm to cope with the real-time evolution of the data store. This works as an add-on to the optimization model, after the initial placement, to efficiently handle the modifications in the applications and constraints, and to minimize data movement operations.

This framework is implemented using an external REST API, independent of a specific data store. This independence allows the user to integrate the API with any data store.

API requests detail the required action and pass on information on current node state (see example API call in the following Listing 1). The API obtains information on application constraints via a data base (key-value store). Using the current state and the constraints, the framework creates the list of the nodes for the request.

Listing 1. Example Call to Placement API in Stateless Mode

```
endpoint = 'placement.api:80/perform'
placement_request = {
  'application': '2',
  'operation': 'Add_Application',
  'node_state': {
    'a': {'available_space': 200,
        'total_space': 400, 'hardware': 'ARM'
        'applications': ['1', '11']},
    'b': {'available_space': 200,
        'total_space': 200,'hardware': 'x86',
        'applications': []},
    'c': {'available_space': 200,
        'total_space': 200, 'hardware': 'ARM'
        'applications': []},
    }
}
response = requests.post(endpoint,
                    json=placement_request)
```

The framework supports both stateless and stateful mode for maintaining the current state i.e., application allocation of different nodes. In stateless mode the framework expects the storage system to maintain the current state of the node allocation and passes it with each request. On the other hand, in stateful mode the framework maintains the current allocation state itself and updates it after every allocation. Both of the modes come with their respective advantages and drawbacks as the stateless mode may be more up-to-date as it also includes the local decisions regarding data movement made by the storage system but while invoking API the storage system has to pass the current state with each of the requests and it consumes extra bandwidth. The stateful mode saves the bandwidth but may not be up-to-date.

We changed the part of the data store that selects the list of data nodes. The modified Ozone uses the API to obtain the data node list to fulfill the demands of the application. In the stateful mode, after creating the list, API updates its records for current allocation to have the details of this new request.

The placement algorithm is implemented in Python. The data placement API framework is written in Flask, which is a micro web framework written in Python. An example call to the *ADD_APPLICATION* function of heuristics looks like the pseudo-code fragment in Listing 1.

The API also supports the other three operations i.e. *Remove_Application*, *Add_Constraint* and *Remove_Constraint*. The API supports both GET and POST HTTP methods. The POST method handles the stateless mode and expects the current state of the nodes along with the application name and operation to be performed. The GET method handles the stateful mode and expects only application name and operation to be performed. The response to each API call is an updated node state, similar to the one shown in Listing 1.

3.4.2. **State of the Art (SOTA)**

This subsection covers the state of the art work from the literature about two topics (optimization in data placement and data placement using sensitivity) as:

Data Placement Optimization

There are numerous works on optimization in data stores. However, we observe that these works are not focused on the constraint-based data placement problem as considered in this work.

Long et al. presets a multi-objective optimized replication management policy for cloud storage clusters [30]. This is offline strategy on five objectives, considering the factors affecting replication including mean file unavailability, mean service time, load variance, energy consumption and mean access latency. However, this work does not consider constraints during optimization.

To reduce the computation delay and response time of the tasks, Li et al. presents a solution that jointly performs optimization at two levels [31]. First on placement of data blocks and secondly on the scheduling of tasks in edge computing. This work also does not consider the constraints during the placement of data blocks.

Chikhaoui et al. presents a multi-objective data placement strategy for Storage-as-a-Service in a federated cloud [32]. They consider the cost of the storage, migration and latency during the data placement. This work considers the constraints related to the capacity and performance local and external storage. This work does not consider the privacy, architectural and location related constraints.

Data Storage Frameworks Targeting Sensitivity

Padmanabhan and Mukesh have presented a Sensitivity-aware data placement strategy for Hadoop, named HadoopSec [33]. This framework uses a machine learning based approach for secure data placement and it uses the sensitivity information associated with the file to be stored to perform the file allocation in Hadoop. It was suggested that companies are concerned with building a single large cluster that contains the data of multiple projects. The possible reason of such concerns are the security vulnerability and privacy invasion by malicious attackers and internal users. This framework works on input data defined by the client including 1) affinity levels between different groups storing data inside this Hadoop cluster, and 2) the sensitivity levels of the file information to be stored. The framework also uses a rack awareness script that utilizes the available information to find the affinity levels of data nodes. It was concluded that the HadoopSec framework adds overhead to Hadoop but as a trade-off it protects sensitive information.

HadoopSec 2.0 is an extension of HadoopSec [34]. It uses the prescriptive analytic algorithm to compute the sensitivity levels of the input file based on metadata and the content of the file if it is not provided by the client. It uses a prescriptive, adaptive, and intelligent system to identify patterns in the input data and group those with similar security concerns. It uses the Doc2Vec algorithm to convert the representation of each file to be stored into a vector for further quantitative analysis. The HadoopSec and HadoopSec 2.0 uses data sensitivity-based constraints. In contrast, the proposed work deals with a variety of constraints from real life applications on the data sharing, data node location, disk space on node and disk space demand of applications and the node's architectural specification. The proposed data placement framework is based on integer programming.

3.4.3. Advancements

The proposed data placement framework deals with a variety of constraints from real life applications on the data sharing, data node location, disk space on node and disk space demand of applications and the node's architectural specification. The proposed data placement framework is based on integer programming.

3.4.4. Performance Evaluations and comparisons

A paper is under review in an international conference. As a general observation, we can see that heuristic algorithm (named as CATER in results) and optimization model (named as optim in results) both manage to respect all of the application constraints throughout the experiments, and that they do so while using fewer storage nodes than Ozone default. Dynamic heuristic algorithm outperforms optimization model in regards to algorithm execution time, but also more significantly, sees a substantial reduction in the number of data movements, where applications must be moved between data nodes in order to ensure that constraints remain satisfied.

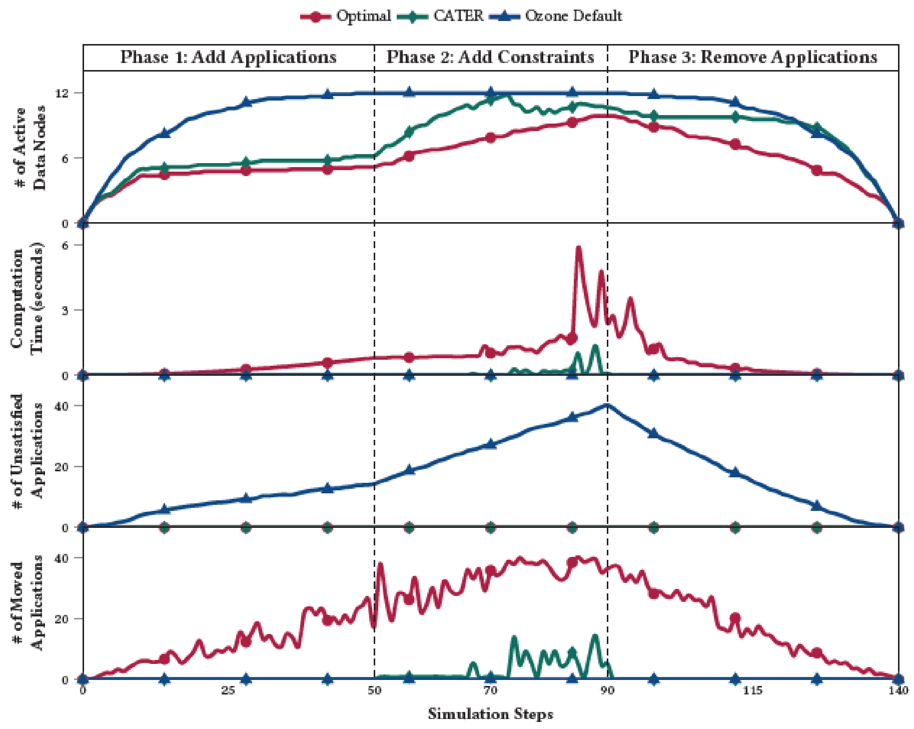
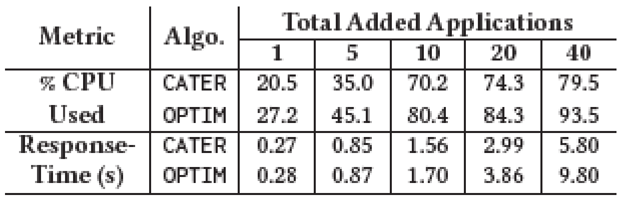


Figure 3.16 Performance Evaluation of the Data Placement Component

Simulation results averaged over multiple runs. Unlike Ozone default, CATER and Optimal satisfy all application requirements. In comparison to Optimal, CATER uses more nodes but improves computation time and application movement.

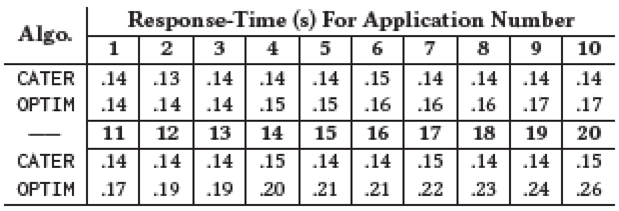
From the results we conclude that heuristic algorithm is an attractive choice when seeking to respect application constraints, even when the set of applications and constraints is highly dynamic. Using the optimal approach results in a substantially higher number of movements which is detrimental to the operation of an Edge storage system and if not handled carefully can lead to delays for applications in accessing their data. Over the experiment lifetime, optimization model occupies just 1.6 fewer storage nodes compared to heuristic algorithm, thus reinforcing the clear benefit of heuristic algorithm as an efficient solution for constraint-based edge storage.

Table 3-3 CATER vs. Optimal Performance in real deployment on Apache Ozone. In comparison to Optimal, CATER yields better response time and consumes less CPU.



Metric	Algo.	Total Added Applications				
		1	5	10	20	40
% CPU Used	CATER	20.5	35.0	70.2	74.3	79.5
	OPTIM	27.2	45.1	80.4	84.3	93.5
Response-Time (s)	CATER	0.27	0.85	1.56	2.99	5.80
	OPTIM	0.28	0.87	1.70	3.86	9.80

Table 3-4 CATER vs. Optimal API Response Time per application added. Optimal takes longer to respond as more applications are added into the system.



Algo.	Response-Time (s) For Application Number									
	1	2	3	4	5	6	7	8	9	10
CATER	.14	.13	.14	.14	.14	.15	.14	.14	.14	.14
OPTIM	.14	.14	.14	.15	.15	.16	.16	.16	.17	.17
—	11	12	13	14	15	16	17	18	19	20
CATER	.14	.14	.14	.15	.14	.14	.15	.14	.14	.15
OPTIM	.17	.19	.19	.20	.21	.21	.22	.23	.24	.26

3.5. Motif Discovery Tool (C4.11)

Component ID	Component Name	Development	Owner
C4.11	Motif Discovery Tool	100 %	FS

GitLab Repository: https://gitlab.com/braine/wp4-mod-discovery-module-fs

Containerized: Y

Registered on BRAINE platform image registry: Y

Deployed as a pod and is functional on BRAINE platform: Y

Integrated with other platform components: Y – with MOD Learning Module (WP4.2) and MOD core module with GUI and InfluxDB and MongoDB databases.

Status:

The Motif Discovery Module of Motif Discovery Tool (MOD) enables the discovery of all repetitive patterns of any size in any time-series data. The time-series data from the perspective of BRAINE are sensory data from machines and devices on the shop floor. The discovered patterns represent unique operations of the machine.

MOD is divided into several containerized modules, which can be deployed individually on different host machines. Within WP4, the Discovery module and the Detection module are being developed and implemented.

Components were tested on CNIT Braine Testbed. These components are integrated in the UC3 with the Learning Module of Motif Discovery Tool (WP3).

3.5.1. Technical description

Motif Discovery module

The Motif Discovery module enables the discovery of all repetitive patterns of any size in any time-series data. The time-series data from the perspective of BRAINE are sensory data from machines and devices on the shop floor. The discovered patterns represent the unique operations of the machine.

The Motif Discovery module is based on the PyTorch tensor engine for fast processing of batched data. It utilizes a proprietary API for front-end communication and batched-run specification. When the run is specified and started, the process pipeline described in Figure 3.17 is started.

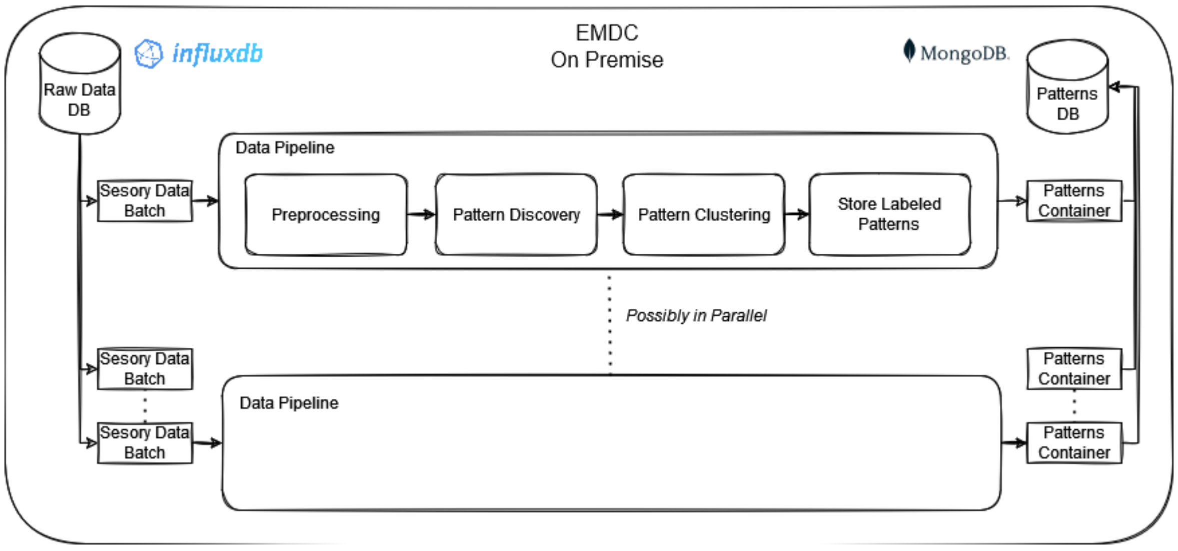


Figure 3.17 Discovery module data processing pipeline

The module connects to InfluxDB to download the batch of the raw time series data. The time series is processed, and the results are stored in the MongoDB as a priory structured result that can be then post-processed to connect automatic IDs of motifs with the human-readable operation tags. This post-processing is done using the MOD core GUI. The primary programming language used is Python.

Detection Module

The Detection module of the MOD application uses state-of-the-art machine learning models to detect the current operational state of the machine. This module uses online time-series data as an input and returns the current operation of a device. Additionally, the Detection module checks the incoming time-series data for deviations from its typical behaviour. When a deviation occurs, an alarm is set on, and subscribers are notified. The design of the data processing pipeline is depicted in Figure 3.18.

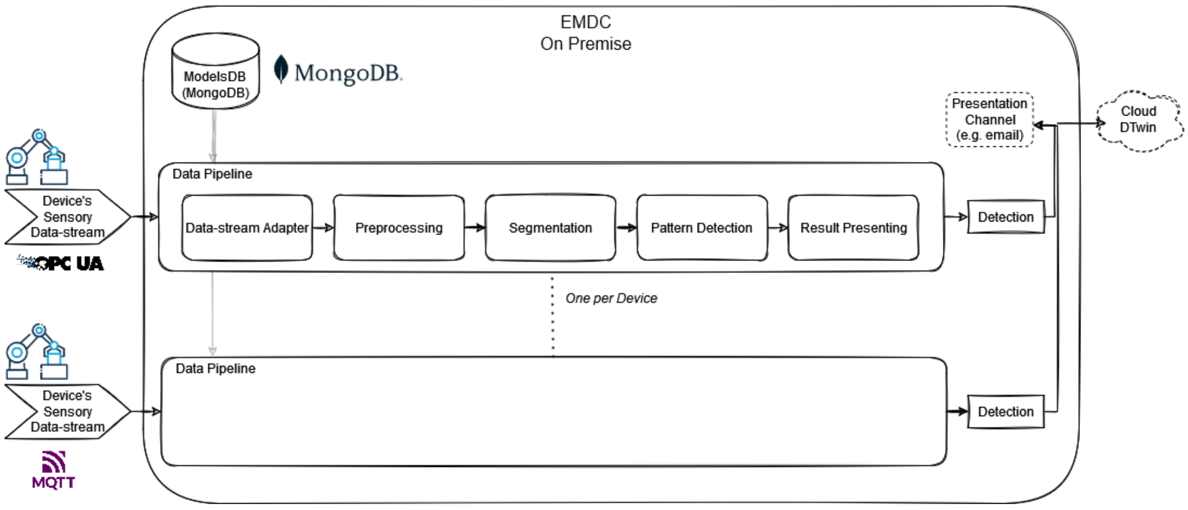


Figure 3.18 Motif Discovery- Detection module pipeline

This module builds on vectorized model likelihood evaluation of the streamed data. This approach allows for a better flexibility in the modeled patterns, which results in better detection performance. This comes at a price of higher computational complexity, but this is covered by increased processing power of the BRAINE HW platform developed in WP2. The detection module supports two types of stream adapters, that is the Open Platform Communications Unified Architecture (OPC UA) standard and Message Queuing Telemetry Transport (MQTT) communication. The programming language is Python, utilizing the PyTorch framework.

Digital Twin

The Digital Twin module runs in a cloud platform. It utilized an MQTT broker that is accessible from both the BRAINE edge and the cloud. It subscribes for the asynchronous events presented from the Detection Module and keeps one event-log for each monitored machine. The Digital Twin runs a probabilistic model of event sequences observed in the events stream and is capable to provide the probabilistic assessment of the current state of the monitored machines fleet. This assessment is presented to the user working on the cloud. The key is the compression ratio provided by a high coverage of data by discovered motifs. The events log can be utilized for data reconstruction on the side of the cloud.

3.5.2. State of the Art (SOTA)

The current market offers either toolsets for developers to bring this young topic of time series motif discovery for testing, or only a very basic manual system for hand-picking motif candidates from raw time series. An example is a commercial product Trendalyze. The Trendalyze application provides tools for an assisted selection of motif candidates which are then searched for in the raw data. Such an approach does not automate the discovery step and relies completely on the luck and knowledge of the user of the application. MOD on the other hand focuses on stepping one level higher and automates this motif-candidate generation step.

An example of the development frameworks that focuses on motif discovery is, e.g., STUMPY framework, which provides state of the art methods such as the Matrix Profile, which needs to be provided with considerable number of tuning parameters to successfully discover motifs in data. Our MOD Motif Discovery module offloads this responsibility from the user and leaves only a minimal number of parameters to tune, which are comprehensive for the user.

For pattern detection, state of the art solutions for analysis of data-streams are, e.g., BitSwan platform. BitSwan provides number of data connectors from variable sources, but builds primarily on pattern similarity measures. MOD Detection module utilized ML models with higher modeling capabilities and implicit model likelihood support.

References:

Time Series Intelligence and AI 3.0, whitepaper, https://trendalyze.com/wp-content/uploads/2019/12/Trendalyze-Introduction.pdf

Scientific Approach for Visual Motif Discovery, whitepaper, https://trendalyze.com/wp-content/uploads/2019/02/Scientific-Approach-of-Visual-Motif-Discovery.pdf

STUMPY framework, Github repository, https://github.com/TDAmeritrade/stumpy

BitSwan platform, product webpage, https://libertyaces.com/solutions/industry.html

[35]

3.5.3. Advancements

The MOD Motif Discovery module specializes in discovering of motifs in data. This is a fundamental step in pattern detection and currently, this step is offered in a form of a customer service. The MOD application offers detection-patterns preparation as a feature in the hands of the end user. This is a novelty for the current market, as current commercial solutions do not provide such a feature.

3.5.4. Performance Evaluations and comparisons

One of the main objectives of this work package within the BRAINE project is data privacy. The owners of the data need to provide the Discovery module access to their data. This is achieved by access authorization using a database. Another objective was an efficient processing of data to save bandwidth usage during production deployment of the detection module. The Motif Discovery is a fundamental step for this objective as it produces the compression dictionary used on the compression and de-compression side of the communication.

We conducted functional tests of the motif discovery phase. The tests were conducted on benchmark datasets containing variable ratios of motifs. Results are listed in Table 3-5. Our algorithm discovers motifs of the closest motifs ratio to the expectations when compared to the Matrix Profile algorithm of the STUMPY developer framework.

Table 3-5 Discovery coverage comparison of MOD and SotA

Dataset nametag	Ground-truth coverage	MOD coverage	Matrix Profile coverage
ECG	0.97	0.98	0.757
GAP	0.95	0.95	0.153
Robot	0.373	0.387	0.828

All tests were conducted on the CNIT Testbed BRAINE (described in deliverable D5.5) cluster with emulated cluster nodes as described in D5.5. Further testing and evaluation will be completed in WP5 in the UC3 deliverable.

3.6. AI platform profiling engine (C4.13)

Component ID	**Component Name**	**Development**	**Owner**
C4.13	AI platform profiling engine	85%	NEC

GitLab Repository: No Gitlab repo is provided, since it is a subcomponent integrated into an NEC proprietary solution.

Containerized: Y

Registered on BRAINE platform image registry: N

Deployed as a pod and is functional on BRAINE platform: N

Integrated with other platform components: Y – UC2 video analysis application

Status: The tool is integrated into applications by leveraging direct import in high-level domain-specific machine learning frameworks, such as PyTorch and TensorFlow. Therefore it can be shipped jointly with the application deployment units.

3.6.1. Technical description

The execution of deep neural networks requires performing a large number of mathematical operations in a sequence of neural network's layers. Each of the layers has different computation profiles and its operations are generally defined independently from those of other layers. Nonetheless, the combination of layers in a sequential structure affects the actual observed execution profile and the overall runtime performance. This is the case since each layer might imply a specific set of data movements, intermediate results and require specific hardware subsystems. As such, the actual runtime performance of a deep neural network, even in the cases of a simpler static execution of a fixed set of layers, is hard to predict without analyzing the details of the involved computations. We developed a profiler to assess the execution performance of deep neural networks taking into account the computation graph deriving from the static analysis of the neural network layers (or by tracing execution of the neural network algorithm when in presence of dynamic runtime structures).

Our tool builds an acyclic computation graph of the neural networks, tracking all the data transformations (including input and output shapes). The system then maps the computation graph to the corresponding mathematical operators, which are provided by backend libraries (for example, cuDNN for execution on NVIDIA GPUs). The backend mapping allows the system to assess a potential execution schedule on the target device, thereby enabling it to predict the sequence of computations that will be actually performed. In this process, the profiler is also capable to estimate potential backend optimizations, such as operator fusion.

With this information, the profiler performs a best guess about the execution performance, taking into account a best-effort cost model for each combination operation-backend. The cost model can be updated over time with data collected from actual runtime executions.

3.6.2. State of the Art (SOTA)

The current state-of-the-art in profilers for Deep Neural Networks are mostly based on black box approaches: an executor runs the neural network on the target hardware and for a given input, and measures the runtime [26]. The measurement is then scaled to larger/different inputs. This current approach works well for a number of simpler static feedforward networks, but shows limitations when the actual computation depends on the characteristics of the input being processed, and when the type of target hardware cannot be established a priori. In this context the construction of a computation graph can enable deeper analysis and accurate static evaluation of different behaviors, including taking into account learned hardware cost models.

3.6.3. Advancements

We integrated the profiling tool into a compiler toolchain for neural networks, in order to generate, at deployment time, an implementation for a deep neural network that best suites the target executor. That is, our profiler can guide a code synthesis process within the compiler. This combination of profiler and compiler is especially effective since the compilation toolchain needs to build similar data structures create execution schedules. The profiler adds the ability to attach performance forecasts to the different schedules, before there are even generated or tested on the target hardware.

3.6.4. Performance Evaluations and comparisons

The evaluation of our profiler in isolation is challenging since it is part of a larger compiler toolchain, however, we can extract a relevant performance metric: execution time for common neural network architectures (e.g., ResNet, MobileNet, etc). We perform the evaluation by considering the profiling on unseen hardware and compare that to APACHE TVM [3], currently employed in the Amazon SageMaker products for the generation of efficient neural network executors. In all the tested cases our profiler could provide a verdict in few seconds, and in any case under a minute. TVM required instead several hours for its evaluation, since it requires to perform several execution tests on the target hardware before building a consistent performance model.

3.7. vRAN with adjustments (C4.12)

Component ID	Component Name	Development	Owner
C4.12	vRAN with adjustments	80%	ISW

GitLab Repository: https://gitlab.com/braine/wp4-vranwithadjustment-isw

Containerized: Yes

Registered on BRAINE platform image registry: No

Deployed as a pod and is functional on BRAINE platform: Yes

Integrated with other platform components: Y – Telemetry system and cognitive framework (in progress)

Status: Under testing and integration with the BRAINE platform

3.7.1. Technical description

The components of C4.12 encompass the 5G vRAN (containerized 5G SW) equipped with the scaling mechanism. The mechanism is there to perform offloading of the current vRAN instance component to another EMDC. In here we assume that the offloading (handling user data at scale) serves for the purpose of handling more users than would be possible to serve with just single EMDC (and thus the vertical scaling in place). Figure 3.19 shows a conceptual diagram of the vRAN data handling framework.

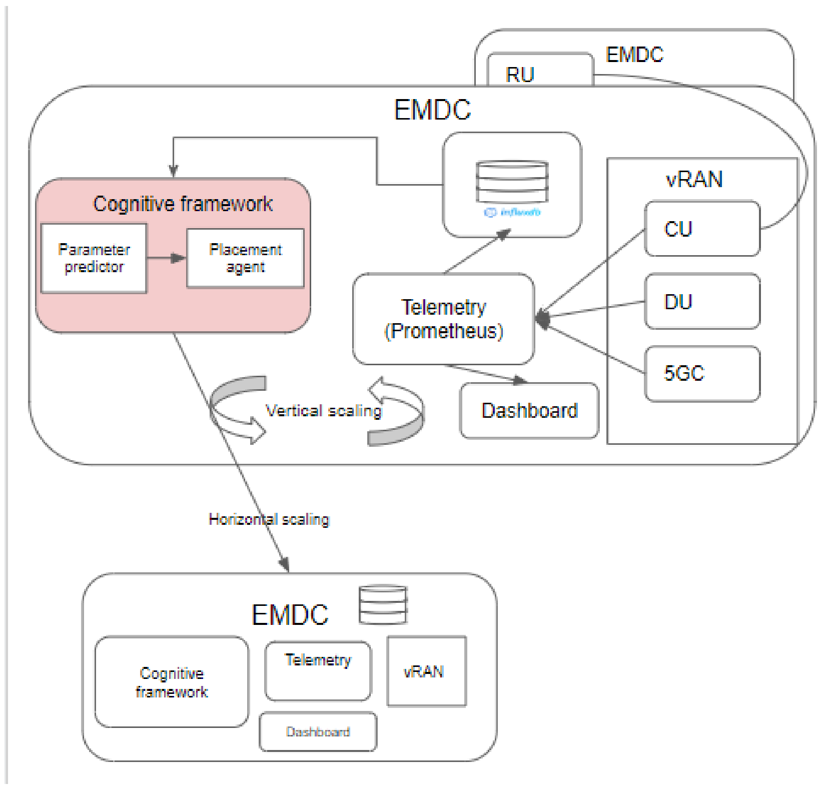


Figure 3.19 Conceptual diagram of the vRAN data handling framework

Figure 3.20 shows the proposed system model of workload prediction. At the first step, the metrics which will be used for the prediction algorithm are collected from the 5G Open RAN radio stack gNB network function and they are delivered with internal EMDC messaging to the Resource Manager (RM) entity. A predictive technique is defined as a statistical model that can be applied to known data of a given phenomenon to estimate future metric evolution.

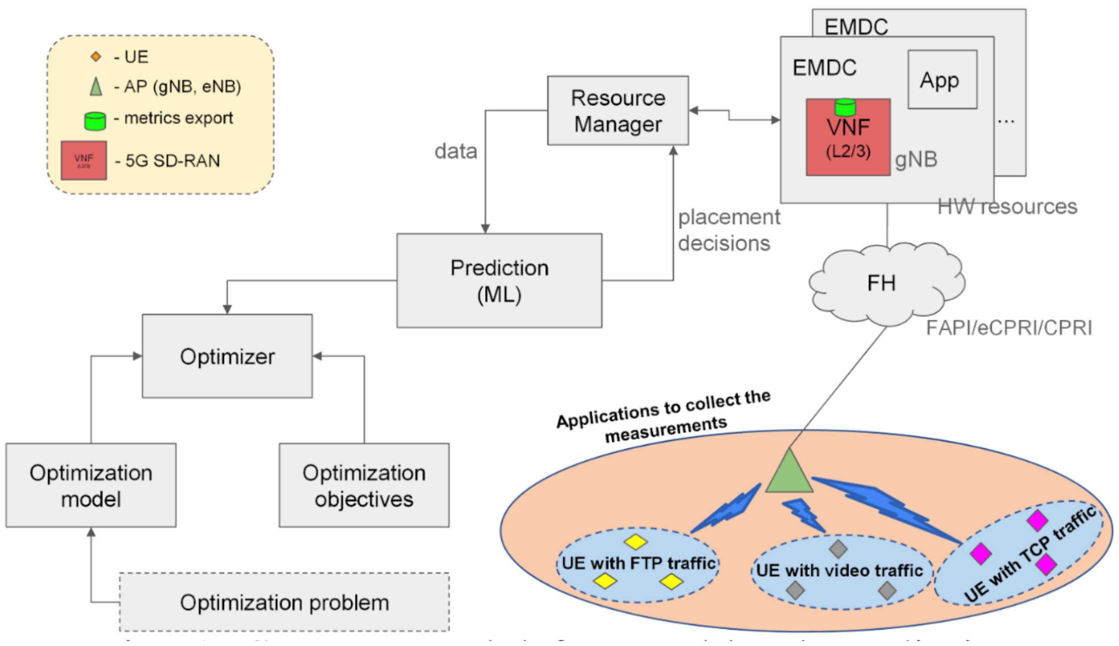


Figure 3.20 System model for workload prediction

In this work, the RM utilizes this data to feed arbitrary prediction techniques based on several ML algorithms such as ARIMA [20], LSTM [21], and N-BEATS [22], with proper inputs that allow characterization of the virtualized gNB operation regarding its demand for computing resources of the EMDC**.**

3.7.2. State of the Art (SOTA)

The number of edge devices increases every day, with their capabilities continuously evolving. In a typical edge computing paradigm, multiple edge servers are placed close to the end users to support quick computation and required bandwidth. However, the escalated devices will introduce several challenges of resource management and elasticity towards vRAN in the EMDC. The accurate prediction of the future workload such as central processing unit (CPU) consumption is critical to the efficiency of vRAN resource management. Due to resource constraints of the edge servers, precisely predicting the workload of various edge servers can be of great importance in proficiently utilizing the EMDCs. The role of combined compute and radio resource scaling in case of virtualized 5G networks deployments is essential as it allows to perform the paradigm shift from a network provisioning based on busy hours.

In [3], the Open RAN (O-RAN) case is presented where the telemetry is obligatorily collected to support AI/ML model training. The 3GPP network data analytics function (NWDAF) framework collection is considered, where the telemetry data collector includes a monitoring server (e.g. Prometheus) for collecting performance measurement data from the NWDAF, the virtualized infrastructure and the transport network elements. Here the 5G dataset is utilized in order to train AI/ML models to predict cell-load as well as energy efficiency. Requirements for network analytics in 5G are provided in addition to application and slice analytics, e.g., interference handling, channel quality prediction, dynamic radio topology control, multi-slice radio resource management [4]. Author in [5] presented a proof-of-concept design and implementation for blockchain-based slice orchestration at the Network Edge aligned with European Telecommunications Standards Institute (ETSI) Network Function Virtualization (NFV) standard. Such scaling and migration features are identified in the 3GPP specs defining slicing mechanism (3GPP SA5 with TS28.531, TS28.526; ETSI GS ZSM with ZSM003). Comprehensive overview of common online data analytics frameworks is presented in [6].

Several artificial intelligence/machine learning (AI/ML) based predictive methodologies and schemes are proposed to estimate the future resources demand. Author in [14] proposes a containerized edge computing framework for dynamic resource provisioning. This framework integrates workload prediction and resource pre-provisioning to provide high utilization of edge resources. Pramanik et al. characterize the computational and memory requirements of virtual RANs with regression models to predict better demands for resources [15]. They use 4G vRAN test-bed leveraging the non-disaggregated open-source mobile communication platform and general-purpose processor-based servers. The ML based virtualized network functions (VNFs) prediction and placement in the network edge is investigated in [16] where authors propose a neural-network model to assist in proactive auto-scaling by predicting the number of VNF instances required as a function of the network traffic. This research also investigates the placement of these VNF instances at the edge nodes with a primary objective of minimizing end-to-end latency from all users to their respective VNFs. Authors from Microsoft introduce a user space deadline scheduling framework Concordia [17] for the vRAN on Linux. It builds prediction models using quantile decision trees to predict the worst case execution times of vRAN signal processing tasks. These predictions are used to calculate and proactively reserve the least number of cores required to perform the vRAN pool operation in the next slot (e.g., 1ms), releasing the rest of the cores to the operating system (OS) for other tasks.

There have been several studies in the literature reporting data collection frameworks. Authors in [7] refer to the usage of Prometheus and Netdata for metrics collection of VNF to support slice monitoring. Moreover, multiple telemetry sensors are defined in various levels of the stack of 5G infrastructure. Prometheus provides an easily machine-readable format. In [8] authors consider that telemetry systems should be able to instrument and monitor the different devices composing the overall infrastructure in order to deal with highly distributed 5G small cells. They also point that such systems should be able to generate aggregated and derived metrics, as well as follow the distributed approach. They selected Prometheus as the most suitable platform for a two-tier virtualization architecture. It is highly scalable due to its hierarchical federation capabilities. A similar environment with Kubernetes and Prometheus is considered in work [9] which uses it to monitor 4G networks based on open air interface (OAI). On the other hand, the solution inspired by the Barometer framework uses InfluxDB and Grafana is considered in [10], for holistic slice management. 5G network monitoring with the usage of P4 switches is presented in the work of authors in [11]. Especially the user plane function offloading approach in EMDC switch is considered. Authors in [12] describe the architecture of a monitoring system which is able to be deployed on top of network function virtualization (NFV) Objects, such as Platform as a Service (PaaS). The need for a distributed big data repository to handle telemetry data is highlighted by authors in [13].

However, none of the above mentioned research indicates the workload prediction of vRAN in the EMDC type platform in addition to the data collection frameworks. Moreover, the AI/ML based approaches of data prediction with the real testbed is yet missing in the vRAN scenarios for the edge micro data centers.

3.7.3. Advancements

In the alignment of vRAN optimization, several research and development works have been performed from both industry and academia. It should be noted that scaling functionalities of vRAN includes the workload prediction and placement in most of the cases. Our main focus aims to handle data effectively at scale which maps to addressing performance requirements of any BRAINE workload, and especially the secure 5G wireless communication infrastructure. The BRAINE aims to boost the development of the edge networks with multi-processor accelerators on board. Network slicing which considers the virtualized logical networks on the same network platform is thus required to get the full benefits of huge data management in EMDC. At the same time, a complete data collection framework is needed to collect and measure the data for a target variable to evaluate the system performances. It is important to note that data handling and collection mentioned in this work is directly aligned with 3GPP [1] and ETSI [2] standards defining the slice management. However none of the standards address the particular mechanisms and algorithms (e.g. auto-scaling, self-healing or migration) to assure proper realization of slice management in micro-data center based networks with 5G disaggregated virtual networks at the edge.

From the perspective of vRAN optimization, we propose a novel system model of the workload prediction and CPU usage forecasting mechanism in an EMDC architecture. We present the data collection framework for such deployment in a local test bed considering AI/ML based decision models applied in realistic settings. The considered EMDC architecture can host the monolithic VNFs of 4G/5G radio protocol stack in a disaggregated manner. In addition, several ML algorithms for CPU usage prediction is proposed in EMDC architecture based on Long Short-Term Memory Neural Network (LSTM), Auto Regressive Integrated Moving Average (ARIMA), and interpretable time series forecasting (N-BEATS) in combination with collecting the data from the real testbed.

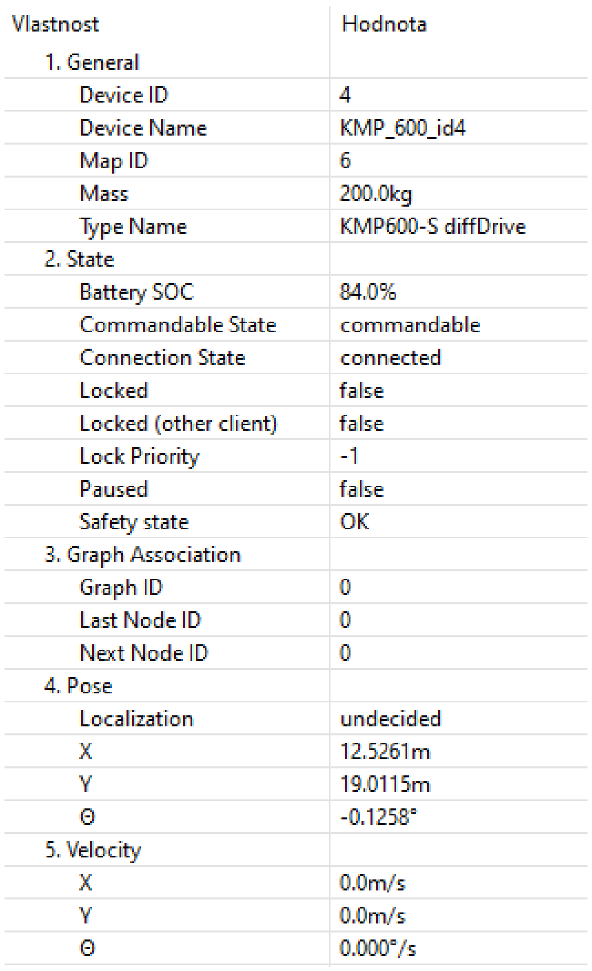
The approaches to enhance handling data at scale, with the focus on capabilities attributed to vRAN and its mechanisms, were described in the D4.2 deliverable (Section

3.3). There we have introduced the “Extended approaches”, which utilize the knowledge of metadata describing the use-case context of execution in order to provide indicative cues for the resource allocation in 5G workload SW. In BRAINE use-cases rely on various workloads deployed inside the EMDC HW to support the user-applications (e.g. eHealth, industrial plant, chip design fabric, surveillance of the city). All the use-cases are similar in the fact that they produce and consume information. What is different is among others, the amount of traffic exchanged (traffic pattern), different criticality of this traffic (class of service or QCI) and level of flexibility to have some traffic adjusted or complemented. An interesting example of providing more options is the UC-3 (industry 4.0).

3.7.3.1. The use of context for interplay between use-case and 5G infrastructure of EMDC

In case of the UC3 the deployment of 5G vRAN in EMDC would address mostly the indoor facility (i.e. factory, warehouse). Here the important parameters determining the 5G infrastructure potential for contextual adjustments are the: a) size of the facility, b) required number of antennas (radio units), c) number of end-terminals, their mobility patterns, role of the data flows, d) criticality of the information delivery and so on. In the case of the UC3, we are dealing with mobile robots that are moving some items between locations. The robots themselves perform a set of well identified routines, but the way they operate is executed according to some states in which they are operating. This way it is possible to i) recognize the location of the robot, ii) identify the current target/objective, iii) understand the next steps based on some contextual information.

Assuming the information about the current snapshot of parameters describing the robots, its actual target, an environment and logical state it is in – it is possible to encode such variables into “context” and regularly deliver it to the 5G infrastructure decision makers, using BRAINE telemetry to the centralized repository. This way 5G vRAN algorithms that manage infrastructure resources, can benefit from accessing the contextual data of a use-case (e.g. UC3) by carefully configuring the subscription of suitable metadata. The means to combining the different domains can be follow e.g. ORAN xApps or ETSI MEC Apps to blend the resource management with the contextual status. The following information is available considering the UC3 and it could be reached via telemetry mechanisms.



Vlastnost	Hodnota
1. General	
Device ID	4
Device Name	KMP_600_id4
Map ID	6
Mass	200.0kg
Type Name	KMP600-S diffDrive
2. State	
Battery SOC	84.0%
Commandable State	commandable
Connection State	connected
Locked	false
Locked (other client)	false
Lock Priority	-1
Paused	false
Safety state	OK
3. Graph Association	
Graph ID	0
Last Node ID	0
Next Node ID	0
4. Pose	
Localization	undecided
X	12.5261m
Y	19.0115m
Θ	-0.1258°
5. Velocity	
X	0.0m/s
Y	0.0m/s
Θ	0.000°/s

Figure 3.21 Example of robot data which could be passed to the vRAN component for optimization

In figure 3.21 it can be observed that location, speed, connectivity as well as states related to docking (related to conveying goods) and charging (loading batteries) are available. When we combine it with the situational awareness on the map of a factory/warehouse we get complete picture of activities currently performed by each robot separately as well as global picture. The latter is related to the overall production/warehousing process status and efficiency.

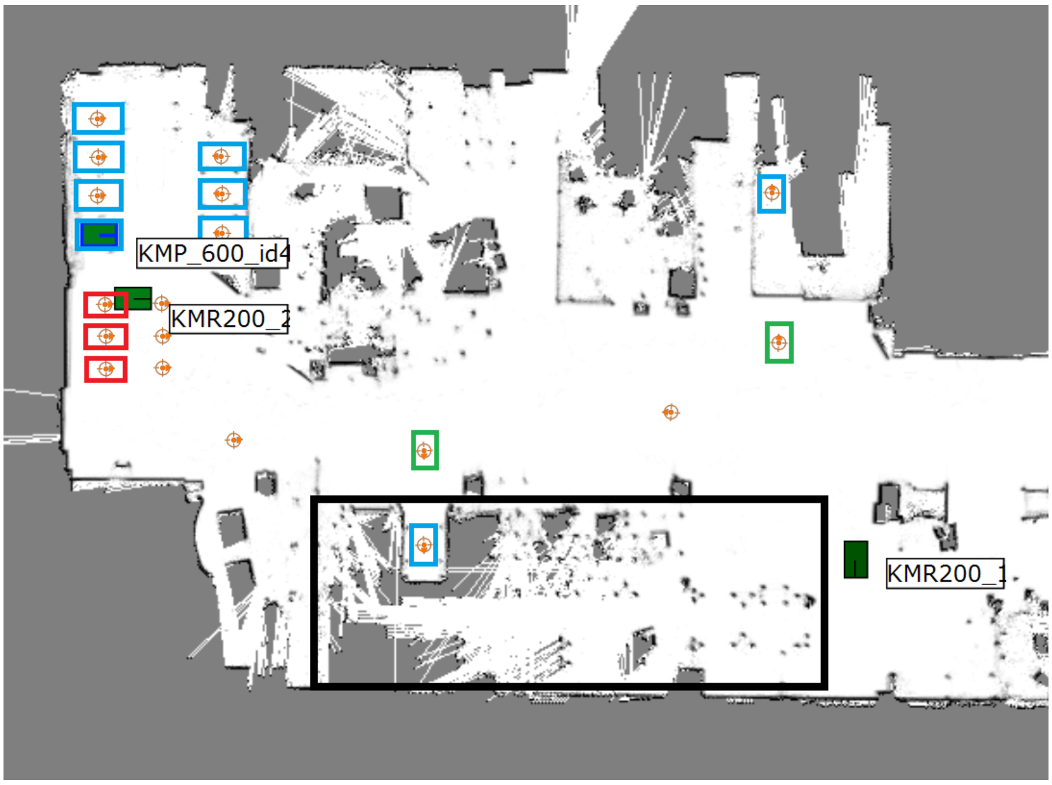


Figure 3.22 Map of factory floor

In figure 3.22 above it can be seen that the white area represents that part of factory that has already been successfully explored by single robot or multiple ones. The robot first scans an area multiple times to find out where are exactly the walls and obstacles (the grey area on the map). One can then send robots to an arbitrary position on the map (but only to the white area), and the robot will get there with certain accuracy. If position/target needs to be more accurate than that, the robot can be taught the exact position (orange points).

Knowledge of shared space outline allows robots to move more flexibly based on the (production, storage) targets specified for each of them. The production line is the black box in the schematic. The docking stations are marked in blue, charging places in red, and pre-docking positions, in which the robot waits for approval from safety to enter the area, in green. They have different purposes. The docking station is for loading and unloading purposes only. The charging station is equipped with chargers and is for charging only.

3.7.4. Performance Evaluations and comparisons

Figure 3.23 illustrates a data collection framework which demonstrates practical deployment of a 5G vRAN as the Kubernetes workload in the edge server represented by two legacy computers (desktop PCs). The mobile network is a full-featured 5G packaged into SW components according to the functional splits defined by 3GPP. Besides the general purpose computers the only specialized HW is the radio head device represented by the USRP node. All the yellow boxes in Figure 3.23, except for RU, can be subject to placement in various partitions of the access and metro network, assuming the required throughput/latency requirements are met.

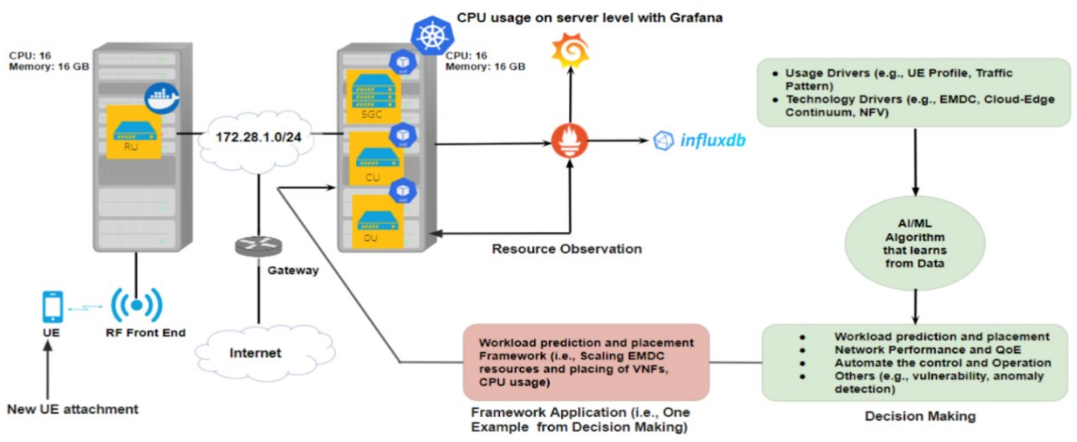

CPU usage on server level with Grafana
CPU: 16
Memory: 16 GB
CPU: 16
Memory: 16 GB
RU
5GC
CU
DU
172.28.1.0/24
influxdb
Usage Drivers (e.g., UE Profile, Traffic Pattern)
Technology Drivers (e.g., EMDC, Cloud-Edge Continuum, NFV)
AI/ML Algorithm that learns from Data
Resource Observation
Gateway
UE
RF Front End
Internet
New UE attachment
Workload prediction and placement Framework (i.e., Scaling EMDC resources and placing of VNFs, CPU usage)
Framework Application (i.e., One Example from Decision Making)
Workload prediction and placement
Network Performance and QoE
Automate the control and Operation
Others (e.g., vulnerability, anomaly detection)
Decision Making

Figure 3.23 Data Collection Framework for vRAN

In order to provide access to a comprehensive set of metrics of a 5G vRAN a framework based on Prometheus exporters, Grafana for visualization and InfluxDB are utilized. With such instrumentation it is possible to properly profile resource consumption of both (i) computing and (ii) radio metrics. Based on such profiling various AI/ML models can be trained in order to be able to predict resource demand (e.g. CPU consumption, memory, throughput, etc.) based on the observed user traffic evolution in time. The collected metrics are 5G vRAN performance measurements on uplink and downlink and resources consumption observation.

We use a practical deployment framework which demonstrates 5G vRAN components deployed as the Kubernetes pods in the EMDC. In order to provide access to a comprehensive set of metrics of 5G vRAN a framework based on Prometheus exporters, Grafana for visualization and InfluxDB are utilized. With such instrumentation it is possible to accurately profile resource consumption of both (i) computing and (ii) radio metrics. Based on such profiling various AI/ML models (e.g., ARIMA, LSTM, and N-BEATS) can be trained in order to be able to predict resource demand.

For this experiment, we used a 5G network deployed on the EMDC which consists of servers on different network configurations. In the EMDC, the centralized unit (CU) and distributed unit (DU) can be deployed as VNF on one server with Kubernetes cluster and radio unit (RU) connected as USRP (i.e. RF front end) on another server, where a physical layer is deployed as Docker Swarm. We considered a commercial UE to establish connectivity for data sending and receiving.

We evaluate the performance of the proposed ML-based workload prediction algorithms by collecting the data from the experimental setup. All analytical results were performed by using the PyTorch library in Python [23] where ML-based models are trained and tested based on the data collected from the real-time test scenarios running in the testbed. Performance of the LSTM model is quite lower than the ARIMA model with transfer learning but the LSTM model has advantage in application and deployment on the EMDC and it does not require additional computation resources in comparison with transfer learning with ARIMA. Workload prediction of CPU usage by N-BEATS model is the worst among the three models.

Mean absolute error (MAE) and mean absolute percentage error (MAPE) are referred to as a loss function for defining error by the model evaluation in order to measure the accuracy of the specified models. The following experimental values are recorded for MAE (5.09, 6.40, and 14.21) and MAPE (0.14, 0.20, and 0.38) in case of ARIMA, LSTM, and N-BEATS, respectively.

Two research works have been prepared from these contributions addressing the data handling mechanisms and data collection frameworks in the EMDC structure, which is accepted in [18] as well as presenting the justification of the workload prediction of virtualized RAN in the EMDC which is submitted in [19].

4. Conclusion

This document provides the status of 7 of the key software components developed under WP4 for development and integration as part of the overall BRAINE platform. Most of the development effort has been completed. Some components are still undergoing integration and testing with other WPs, more specifically with use-cases in WP5. Partners are planning the integration activities with corresponding use-cases.

5. References

[1] ETSI, “Multi-access edge computing (MEC); framework and reference architecture,” ETSI GS MEC 003, V3.1.1, pp. 1–29, 2022.

[2] 3GPP, “Technical specification group services and system aspects; management and orchestration; study on enhancements of edge computing management,” TR 28.814 V17.0.0, pp. 1–49, 2021.

[3] A. Giannopoulos et al., "Supporting Intelligence in Disaggregated Open Radio Access Networks: Architectural Principles, AI/ML Workflow, and Use Cases," in IEEE Access, vol. 10, pp. 39580-39595, 2022, doi: 10.1109/ACCESS.2022.3166160.

[4] E. Pateromichelakis, F. Moggio, C. Mannweiler, P. Arnold, M. Shariat, M. Einhaus, Q. Wei, . Bulakci, and A. De Domenico, “End-to-end data analytics framework for 5G architecture,”

IEEE Access, vol. 7, pp. 40 295–40 312, 2019.

[5] K. Papadakis-Vlachopapadopoulos, I. Dimolitsas, D. Dechouniotis, E. E. Tsiropoulou, I. Roussaki, and S. Papavassiliou, “Blockchain-based slice orchestration for enabling cross-slice communication at the network edge,” in 2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C), 2020, pp. 140–147.

[6] B. Ma, W. Guo, and J. Zhang, “A survey of online data-driven proactive 5G network optimisation using machine learning,” IEEE Access, vol. 8, pp. 35 606–35 637, 2020.

[7] D. Giannopoulos, P. Papaioannou, C. Tranoris, and S. Denazis, “Monitoring as a Service over a 5G Network Slice,” in 2021 Joint European Conference on Networks and Communications and 6G Summit (EuCNC/6G Summit), 2021, pp. 329–334.

[8] J. Prez-Romero, V. Riccobene, F. Schmidt, O. Sallent, E. Jimeno, J. Fernndez, A. Flizikowski, I. Giannoulakis, and E. Kafetzakis, “Monitoring and analytics for the optimisation of cloud enabled small cells,” in 2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2018, pp. 1–6.

[9] A. Mudvari, N. Makris, and L. Tassiulas, “ML-driven scaling of 5G Cloud-Native RANs,” in 2021 IEEE Global Communications Conference (GLOBECOM), 2021, pp. 1–6.

[10] P. Veitch, J. Browne, and J. Krogell, “An integrated instrumentation and insights framework for holistic 5G slice assurance,” in 2020 6th IEEE Conference on Network Softwarization (NetSoft), 2020, pp. 247–251.

[11] F. Cugini, D. Scano, A. Giorgetti, A. Sgambelluri, P. Castoldi, and F. Paolucci, “P4 programmability at the network edge: the BRAINE approach [invited],” in 2021 International Conference on Computer Communications and Networks (ICCCN), 2021, pp. 1–9.

[12] L. Sanabria-Russo and C. Verikoukis, “A cloud-native monitoring system enabling scalable and distributed management of 5G network slices,” in 2021 IEEE International Mediterranean Conference on Communications and Networking (MeditCom), 2021, pp. 42–46.

[13] R. Casellas, R. Martnez, L. Velasco, R. Vilalta, P. Pavn, D. King, and R. Muoz, “Enabling data analytics and machine learning for 5G services within disaggregated multi-layer transport networks,” in 2018 20th International Conference on Transparent Optical Networks (ICTON), 2018, pp. 1–4.

[14] S. Hu, W. Shi, and G. Li, “CEC: A Containerized Edge Computing Framework for Dynamic Resource Provisioning,” IEEE Transactions on Mobile Computing, pp. 1–1, 2022.

[15] S. Pramanik, A. Ksentini, and C. F. Chiasserini, “Characterizing the computational and memory requirements of virtual rans,” in 2022 17th Wireless On-Demand Network Systems and Services Conference

(WONS) IEEE, 2022, pp. 1–8.

[16] T. Subramanya and R. Riggio, “Machine learning-driven scaling and placement of virtual network functions at the network edges,” in 2019 IEEE Conference on Network Softwarization (NetSoft), IEEE, 2019, pp. 414–422.

[17] X. Foukas and B. Radunovic, “Concordia: Teaching the 5G vRAN to share compute”, in Proceedings of the 2021 ACM SIGCOMM 2021 Conference, 2021, pp. 580–596.

[18] A. Flizikowski, E. Alkhovik, M. M. Mowla, and M. A. Rahman, “Data Handling Mechanisms and Collection Framework for 5G vRAN in Edge Networks,” 2022, IEEE Conference on Standards for Communications and Networking (CSCN), ACCEPTED.

[19] Flizikowski, Adam; Alkhovik, Evgeniy; Mowla, Md Munjure; Rahman, Md Arifur (2022): Importance of Workload Prediction of Virtualized RAN in the Edge Micro Data Center. TechRxiv. Preprint. https://doi.org/10.36227/techrxiv.21644708.v1

[20] R. Agrawal and R. Adhikari, “An introductory study on time series modeling and forecasting,” Nova York: CoRR, 2013.

[21] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural computation, vol. 9, no. 8, pp. 1735–1780, 1997.

[22] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio, “N-BEATS: Neural basis expansion analysis for interpretable time series forecasting,” arXiv preprint arXiv:1905.10437, 2019.

[23] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga et al., “Pytorch: An imperative style, high-performance deep learning library,” Advances in neural information processing systems, vol. 32, 2019.

[24] Y.-W. Cheung and K. S. Lai, “Lag order and critical values of the augmented dickey–fuller test,” Journal of Business & Economic Statistics, vol. 13, no. 3, pp. 277–280, 1995.

[25] L. N. Smith, “Cyclical learning rates for training neural networks,” in 2017 IEEE winter conference on applications of computer vision (WACV). IEEE, 2017, pp. 464–472

[26] Chen, Tianqi, et al. "{TVM}: An automated {End-to-End} optimizing compiler for deep learning." 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18). 2018.

[27] Alsulbi, Khalil, et al. "Big data security and privacy: A taxonomy with some HPC and blockchain perspectives." International Journal of Computer Science & Network Security 21.7 (2021): 43-55.

[28] Rhahla, Mouna, Sahar Allegue, and Takoua Abdellatif. "Guidelines for GDPR compliance in Big Data systems." Journal of Information Security and Applications 61 (2021): 102896.

[29] Truong, Nguyen Binh, et al. "Gdpr-compliant personal data management: A blockchain-based solution." IEEE Transactions on Information Forensics and Security 15 (2019): 1746-1761.

[30] Chikhaoui, et al. Multi-objective optimization of data placement in a storage-as- a-service federated cloud. ACM Transactions on Storage (TOS) 17, 3 (2021), 1–32.

[31] Li, C., Bai, J., et al. “Joint optimization of data placement and scheduling for improving user experience in edge computing”. Journal of Parallel and Distributed Computing 125 (2019), 93–105.
[32] Long, S.-Q et al. “A multi-objective optimized replication management strategy for cloud storage cluster.” Journal of Systems Architecture 60, 2 (2014), 234–244.

[33] Padmanaban, R., et al. “HadoopSec: Sensitivity-aware Secure Data Placement Strategy for Big Data/Hadoop Platform using Prescriptive Analytics.” GSTF Journal on Computing (JoC) 5, 3 (2020).
[34] Revathy, P., et al. “Hadoopsec 2.0: Prescriptive analytics-based multi-model sensitivity-aware constraints centric block placement strategy for hadoop.” Journal of Intelligent & Fuzzy Systems 39, 6 (2020), 8477–8486.

[35] C. -C. M. Yeh et al., "Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View That Includes Motifs, Discords and Shapelets," 2016 IEEE 16th International Conference on Data Mining (ICDM), 2016, pp. 1317-1322, doi: 10.1109/ICDM.2016.0179.