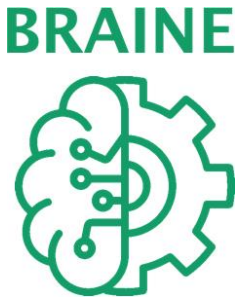


BRAINE - Big data Processing and Artificial Intelligence at the Network Edge

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: D3.3

Third report on the status of WP3

Due date of deliverable:	31 August 2022
Actual submission date:	01 February 2022
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 WP3 – Part 1
Deliverable number	D3.3
Deliverable version	Version 1.0
Previous version(s)	-
Contractual date of delivery	31 August 2022
Actual date of delivery	01 February 2023
Deliverable filename	Final project report on the status of WP3 – Part 1
Nature of deliverable	Report
Dissemination level	PU
Number of pages	44
Work package	WP3
Task(s)	T3.1, T3.2, T3.3, T3.4
Partner responsible	DELL
Author(s)	Mustafa Al-Bado (Dell), Javad Chamanara (LUH), Edgard Marx (ECC), Adam Flizikowski (ISW), Vojtěch Janů (CTU), Radoslav Gerganov (VMware), Antonino Albanese (ITL), Luca Valcarenghi (SSSA), Alessandro Pacini (SSSA), Andrea Sgambelluri (SSSA), Emilio Paolini (SSSA).
Editor	Mustafa Al-Bado (Dell)

Abstract	
Keywords	

Copyright

Deliverable history

Version	Date	Reason	Revised by
00	01.08.2022	Table of Contents - version 00	Mustafa Al-Bado
01	18.11.2022	Proof reading	Mustafa Al-Bado
02	23.11.2022	First review	Mustafa Al-Bado
02	28.02.2023	Final review	Mustafa Al-Bado

List of abbreviations and Acronyms

Abbreviation	Meaning
5G	5th Generation
AI	Artificial Intelligence
API	Application Programming Interface
CPU	Central Processing Unit
CU	Centralized Unit
DSP	Digital Signal Processors
DU	Distributed Unit
ECG	ElectroCardioGram
EEG	ElectroEncephaloGram
EMDC	Edge Mobile Data Center
EPC	Evolved Packet Core
ERP	Enterprise Resource Planning
EU	European Union
FPGA	Field Programmable Gate Arrays
GDPR	General Data Protection Regulation
GPU	Graphics Processing Unit
HRC	Human-Robot Collaboration
iDT	intelligent Digital Twin
ICT	Information and Communication Technologies
IP	Internet Protocol
IoMT	Internet of Medical Things
IoT	Internet of Things
IT	Information Technology
KPI	Key Performance Indicator
MES	Manufacturing Execution Systems
MOD	MOtif Discovery
PoC	Proof of Concept
QSD	Qualified Synthetic Data
RAN	Radio Access Network
TBC	To Be Confirmed
TBD	To Be Defined
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TFLOPS	Tera Floating Point Operations Per Second

TSN	Time-Sensitive Networking
UE	User Equipment
URI	Uniform Resource Identifier
URLCC	Ultra-Reliable Low-Latency Communication
USRP	Universal Software Radio Peripheral

Table of Contents

List of Figures

1. Executive summary

Information technologies have evolved enormously in many domains. In addition, the constraints of many applications elevate the need for edge computing. The BRAINE Work Package 3 (WP3) is dedicated to delivering components to The BRAINE solution that enable intelligent allocating, placement, and monitoring workloads in edge environments. In this deliverable, we report the achievements that have been produced regarding data-processing workloads, resource collaboration, and the Repository of the edge nodes. Specifically, the current deliverable highlights the following achievements:

1) Mapping of AI data-processing workloads in Edge Nodes: This section provides in-depth details for AI pipeline applications that run at edge nodes (5G vRAN, MOD learning module, multiagent communication, and smart city video transcoding). The delivered details include components readiness, performance, and the comparison with state-of-the-art.

2) Blockchain algorithms for resource collaboration: Here, we present a profound description of the used Byzantine agreement protocols, including readiness, performance, and the comparison with state-of-the-art.

3) Repository of the edge node capabilities: This section presents the final version of the BRAINE repository system that runs on edge nodes. The main focus is on the data model, workflow definition, and registry interface functionalities derived from instrumentation, monitoring, and classification sources. In addition, the section covers components' readiness, performance, and the comparison with state-of-the-art.

4) L2 scheduler (slice scaling) for vRAN: This section focuses on detailing the accomplishments of the L2 scheduler for virtual RAN.

2. Introduction

The edge computing environment is inherently heterogeneous and dynamic, with a vast number of nodes that possess varying capabilities, characteristics, purposes, and ownership. These properties are subject to change over time, making it a challenge to effectively manage, control, and monitor the infrastructure and workloads running on it. The solution is to dynamically allocate resources to applications as needed. The goal of this deliverable is to report about the following:

- **Mapping of AI data-processing workloads in Edge Nodes:** This deliverable section presents the functionalities of AI related components in WP3 including performance, development and deployment details.
- **Blockchain algorithms for resource collaboration:** an in-depth description of the used Byzantine agreement protocols, including readiness, performance, and the comparison with state-of-the-art are presented.
- **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.
- **Repository of the edge node capabilities**: This section highlights the final version of the BRAINE repository system that operates on edge nodes. Emphasis is placed on the data model, workflow definition, and registry interface features obtained from instrumentation, monitoring, and classification sources. Furthermore, the section delves into the readiness, performance, and comparison of the system's components with the-state-of-the-art.
- **L2 scheduler (slice scaling) for vRAN:** This section is dedicated to report about the achievements of L2 scheduler for vRAN. The fundamental assumption is that the existing EMDC has limited resources, such as CPU, memory, and storage, allocated per 5G vRAN cluster (or group of pods). However, other EMDCs in the federation can be utilized to increase the current vRAN capacity. This can be achieved by disaggregating the vRAN and offloading specific parts of its stack to the secondary EMDC.

2.1. Reported components

Table 2.1.1 lists the components reported in D3.3 including partners, components' names and Figure 2.1.1 shows an architecture diagram of where these components integrate as part of the overall BRAINE.

Partner	Components	Deliverable
FS	MOD – Learning Module (C3.7)	D3.3
CTU	Multiagent Communication Tool (C3.14)	D3.3
SMA	Data Processing Pipeline (C3.15)	D3.3
IMC	Low bitrate blockchain protocols (C2.19)	D3.3
ECC	Registry Interfaces (C3.8.1)	D3.3
ISW	Placement Agent (C3.19)	D3.3
ISW	Parameter Predictor (C3.20)	D3.3
ISW	Baseline Scheduler 2 (C3.22)	D3.3

Table 2.1.1: Reported components

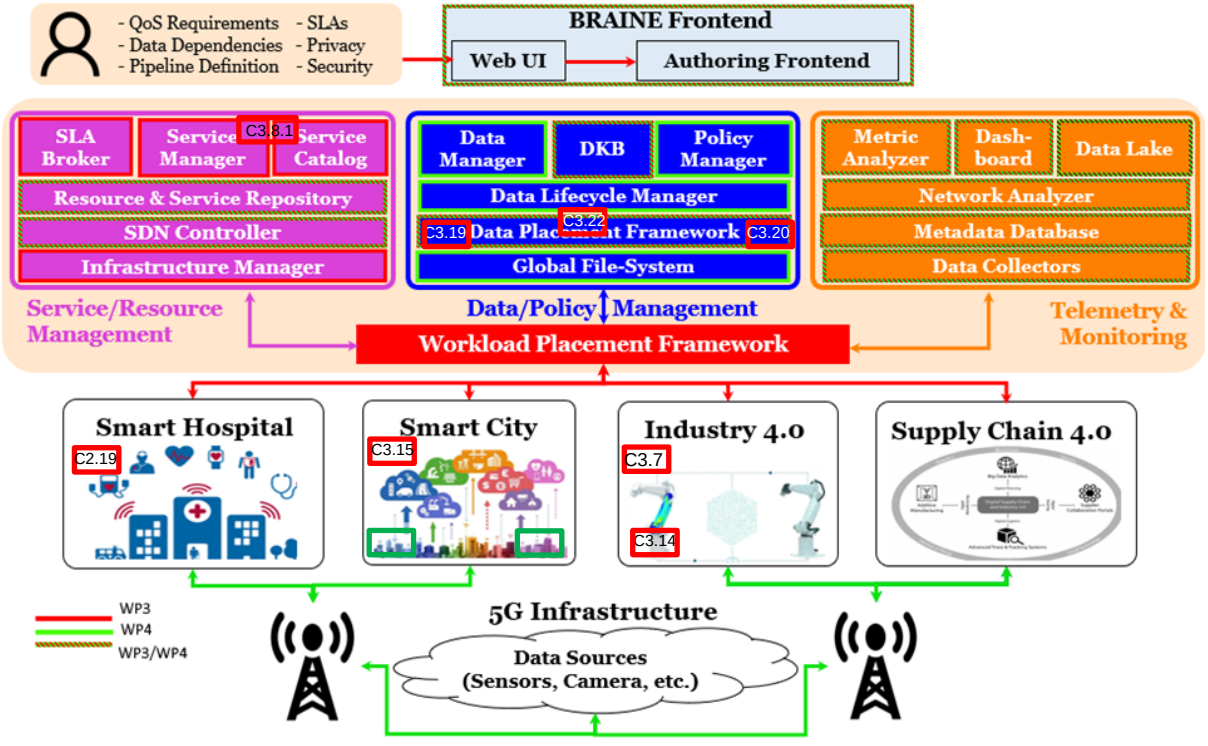


Figure 2.1.1: An architecture diagram of where components integrate as part of the overall BRAINE

3. Mapping of AI Data-processing Workloads in Edge Nodes

This section contains API specifications, implementation details, and endpoints for both vertical and horizontal communication between AI workloads and other UC components in the BRAINE edge environment..

3.1. Parameter predictor and placement agent for 5G vRAN

As regards the implementation of the L2 scheduler based on the workload placement and prediction the current progress has been indicated below.

1. Metrics are collected with telemetry system
   1.1. HW metrics (EMDC resources consumption) with Prometheus
   1.2. Radio statistics (vRAN) collected in csv file
2. Telemetry sends data to dashboard (Grafana)
3. Telemetry system sends data to database located locally
   3.1. EMDC metrics are sent in InfluxDB with HTTP requests
4. Parameter predictor makes predictions of resource consumption based on historical data
   4.1. Trained LSTM model stored locally
   4.2. Parameter predictor receives data in csv file
5. Predicted values are sent to Placement agent
   5.1. Placement agent makes a decision to perform vertical or horizontal scaling

The architecture that is followed for that (prior to Braine 2.0 release) is the one presented in Figure 3.1.1. It is deployed locally as the vRAN deployment in the CNIT testbed is still in progress.

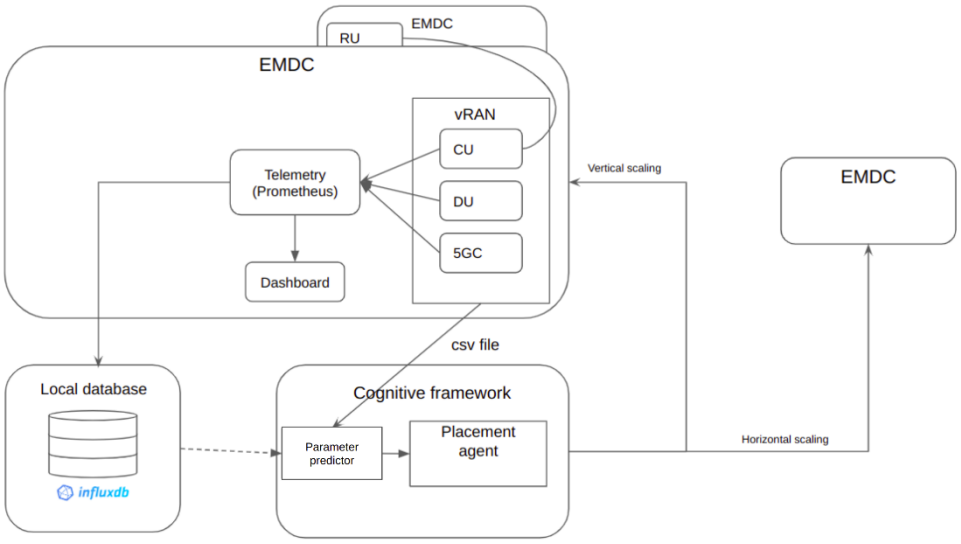


Figure 3.1.1: 5G vRAN workload prediction and placement – BRAINE Rel1.0

In the next stage (Braine 2.0) we will be implementing the “horizontal scaling” referring to the network/slice modification approaches in 3GPP and ETSI, and the architecture foreseen will be targeted like Figure 3.1.2.

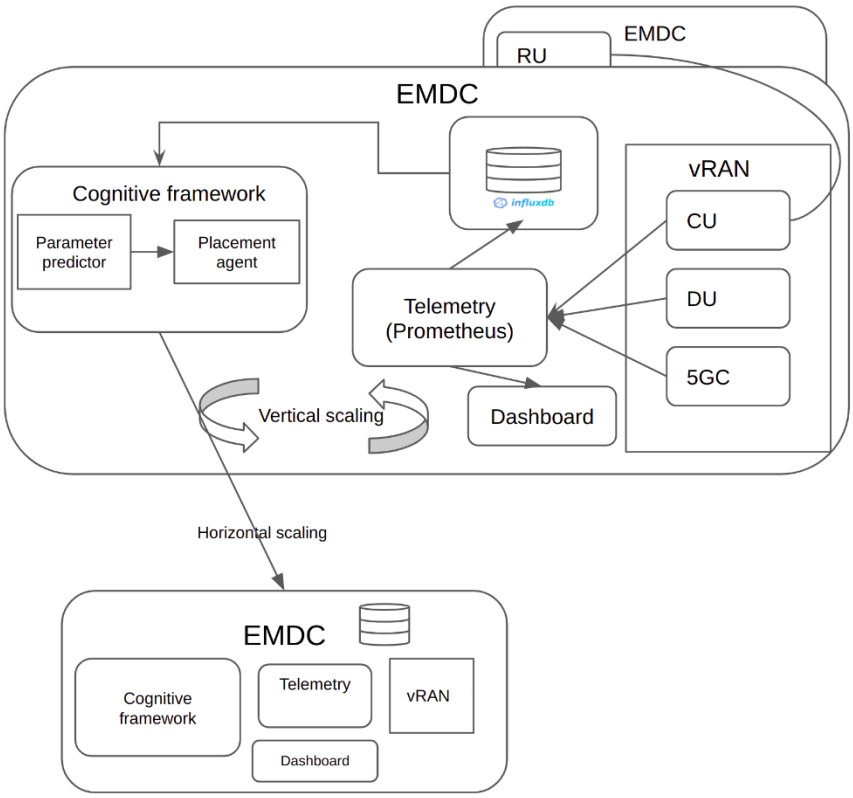


Figure 3.1.2: 5G vRAN workload prediction and placement – BRAINE Rel2.0

The main components engaged are described below.

- 'Cognitive framework' is responsible for horizontal and vertical communication between EMDCs with deployed vRAN.
- Telemetry system collects metrics from EMDC hardware and from vRAN radio logs.
    - Data is collected with Prometheus which sends collected data to InfluxDB v1.8 via HTTP requests with port 3000.
    - Currently, the database is located locally on the ISW machine, further it will be deployed on the EMDC cluster as VolumeClaim. Also, Prometheus sends data to the dashboard system and it can be accessed with port 3200.
- On the current stage radio statistics from vRAN are provided to the cognitive framework, especially to the ML algorithm with a csv file which is updated each second. Predictions are fed to the Placement Agent, which makes a decision either to proceed with vertical or horizontal scaling for vRAN CU-UP.
    - In the next step focusing will be on providing radio statistics exporting from vRAN with the Prometheus instead of receiving data in csv files. Another thing is containerizing the cognitive framework and deploying it on a Kubernetes cluster.
    - Also, deploying the InfluxDB database on EMDC will decrease latencies and improve framework performance.

Component ID	Component Name	Development	Owner
C3.20	**Parameter Predictor**	**100%**	**ISW**
GitLab Repository: Containerized: Y Registered on BRAINE platform image registry: Y Deployed as a pod and functional on BRAINE platform: Y Integrated with other platform components: Y – Status Report: The predictor component is done and validated (release 1.0), results are collected for the prediction accuracy (and delivered to journal). But at the moment we are waiting for the developers to finalize implementation of the “Placement agent” in order to be able to integrate together: Parameter Predictor, Placement Agent with the BRAINE and admission control of the 5G vRAN.			

Table 3.1.1: Component status

3.2. MOD Learning Module

The Learning module as an AI module for UC3 is incorporated into the BRAINE platform on the deployment registry level and on the application runtime level (Table 3.2.1). During the deployment of the application to the BRAINE Service Catalog, the Authoring Frontend is used. The image is registered on the BRAINE platform image registry, from where, BRAINE pods are built and started by end-users of the MOD application. For that, BRAINE pod manifest is used – the SLA files.

Component ID	Component Name	Development	Owner
C3.8	**MOD – Learning module**	**100%**	**FS**
GitLab Repository: https://gitlab.com/braine/wp3-mod_learning_module-fs Containerized: Y Registered on BRAINE platform image registry: Y Deployed as a pod and functional on BRAINE platform: Y Integrated with other platform components: Y – Discovery Module and Detection Module of MOD Application Status Report: Learning module is done, and it was tested on CNIT Braine Testbed. Currently this component is being integrated within the UC3 and other modules of Motif Discovery Tool (WP4).			

Table 3.2.1: Component status

During application runtime deployment, the module connects to the rest of the MOD modules using MQTT broker (exposed on port 1883, or 8883). Connecting namely to:

- Discovery module (WP4.2),
- Detection module (WP4.2),
- MOD core module with GUI
- DTwin on cloud (WP4.2).

Additionally, connection to two databases is required:

- InfluxDB (usually port 8086, using InfluxDB v2 API – HTTP-based API) for raw timeseries data
- MongoDB (usually port 27017, MongoDB API – HTTPS) for storing the resulting learned models.

When required, new instances of each database or MQTT broker are started within the app local network. All MOD services share the same virtual network using the BRAINE Service Mesh as a transparent layer. For communication with the cloud-located DTwin module, MQTT protocol is used and an initial direct upload of the referenced models to the cloud-side MongoDB. All the databases use persistent volumes. The Learning pipeline is depicted in Figure 3.2.1.

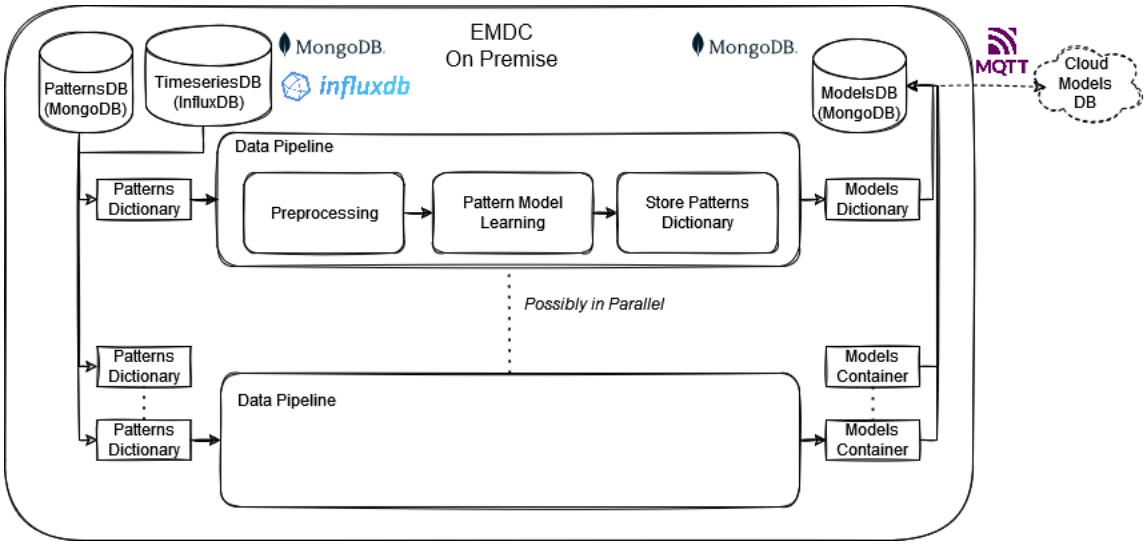


Figure 3.2.1: The Learning pipeline of UC3

3.2.1. Readiness and Performance

Tests on preliminarily collected datasets depicted in Figure 3.2.2 and Figure 3.2.3 lead to a classification pipeline composition of 2 models with the performance metric of Predictive Marginal Log-Likelihood (MLL) listed in Table 3.2.2.

The MLL measures the goodness of fit of the model on the given data. It is a logarithmic measure and ranges in real numbers. The usual baseline is data-dependent, thus we report also the baseline value for completely random model parameters (the untrained model). The advantage of the MLL measure is that it can be applied on variable-length examples and still gives comparable results. On the other hand, it can be applied only on models that calculate probability of samples of the examples.

The models were trained on an internal benchmark dataset of 22 examples of model ID 1. and 28 examples of model ID 2. This benchmark dataset was obtained from the Motif Discovery step, as described in D4.3.

Model ID	Predictive MLL (higher is better)	Untrained Model MLL (baseline for comparison)	Speed (iterations/s)
1.	1.43 ± 0.03	-1.73 ± 0.06	40.78 ± 0.67
2.	2.93 ± 0.05	-1.38 ± 0.16	40.13 ± 2.78

Table 3.2.2: The classification accuracy was tested under the WP4 and reported in D4.3.

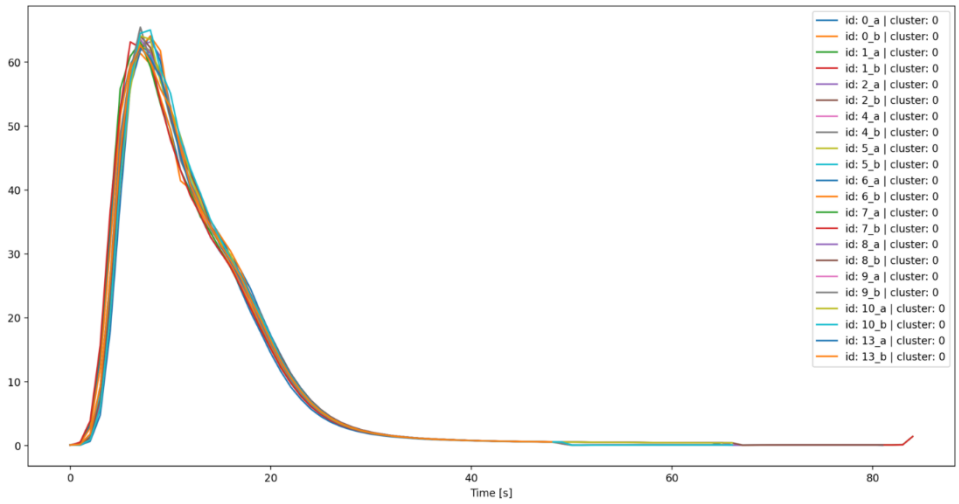


Figure 3.2.2: Data for Model ID 1 from Table 3.2.2

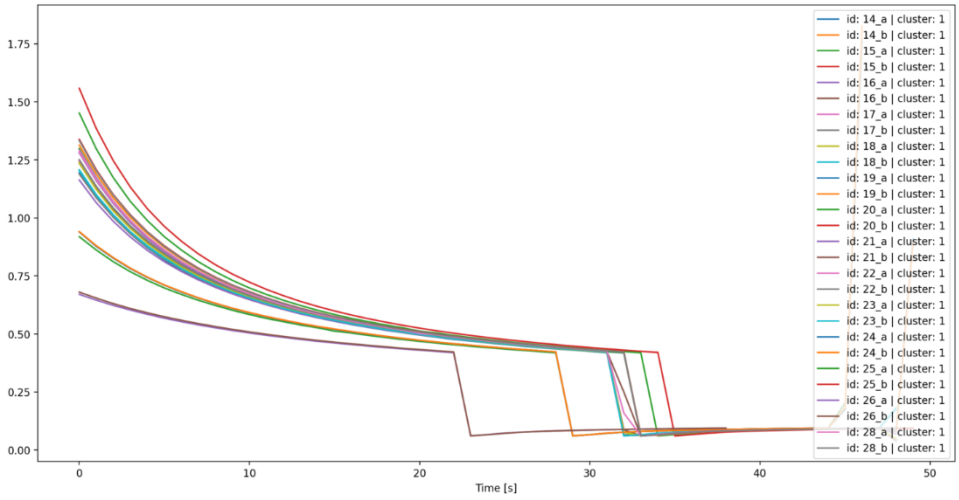


Figure 3.2.3: Data for Model ID 2 from Table 3.2.2

The training settings were picked as a tradeoff between model likelihood and its complexity. This is a feature of this component that the models are tunable to fit the needs of the particular user.

3.2.2. Comparison with State of the Art

Current commercially available methods provide approaches based on a typical representant template that provides baseline to compare other examples to. In the classification pipeline, the model with the best similarity is chosen as the class. For example, the Trendalyze application provides tools for assisted selection of examples that serve as a template which is searched for in the rest of the data [Trendalyze1,

Trendalyze2]. Such an approach lacks the context of the clustered examples which are combined into our model. The price for our model-based approach is a higher computational complexity, but this is addressed by the powerful HW developed in WP2.

3.3. Multiagent communication

The P2P communication implements a message structure based on FIPA ACL using AMQP protocol. Each agent is equipped in the northbridge with a communication mechanism that connects as a client to the RabbitMQ server and allows sending messages to particular agents. The messages for each agent are stored in the agent's queue from which the agent can retrieve the incoming message (See Figure 3.3.1).

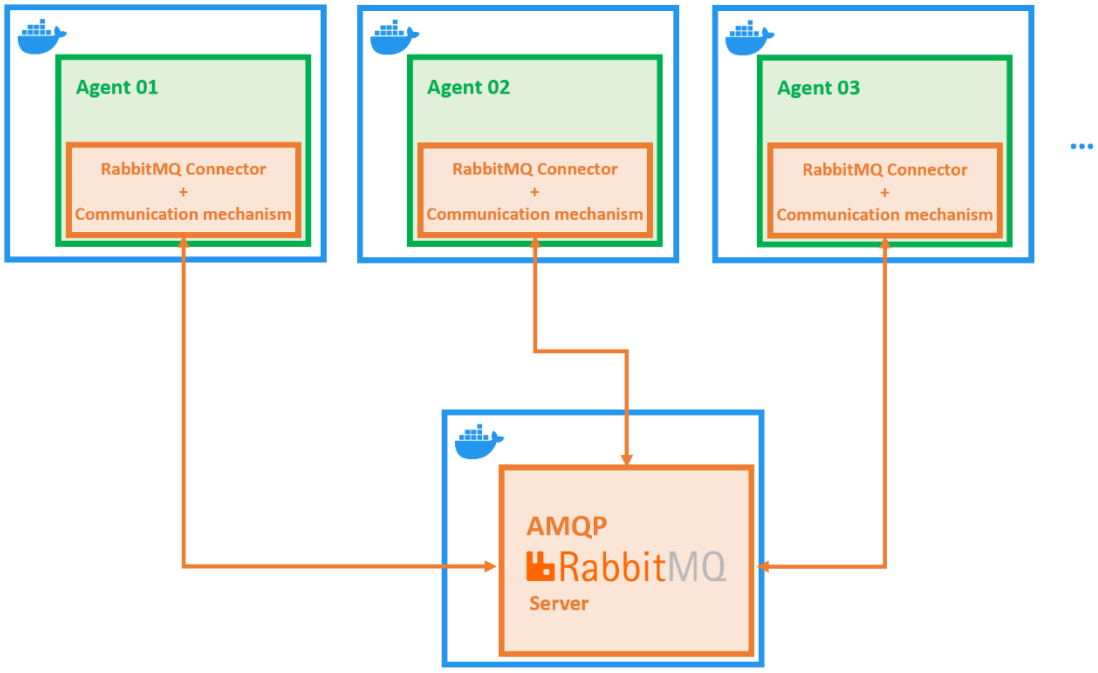


Figure 3.3.1: Dockerized multiagent architecture from the point of view of the communication

3.3.1. Message Structure

The message structure (in Table 3.3.1) is based on FIPA ACL Message Structure Specification and is inspired by HTTP headers (RFC 6648, RFC 4229). The structure was designed to give precise information about the current state and intentions of agents participating in the conversation.

Name	Based on	Mandatory	Description
Sender	FIPA	Yes	Denotes the identity of the sender of the message
Receiver	FIPA	Yes	Denotes the identity of the intended recipients of the message

Ontology	FIPA	No	Denotes the ontology(s) or other knowledge structure used to give meaning to the symbols in the content expression
Message ID	Custom	Yes	Used to identify the particular message in a conversation
In reply to Message ID	FIPA	No	Denotes an expression that references an earlier action to which this message is a reply
Reply to	FIPA	No	This parameter indicates that subsequent messages in this conversation thread are to be directed to the agent named in the `reply-to` parameter instead of to the agent named in the `sender` parameter
Conversation ID	FIPA	No	Introduces an expression (a conversation identifier) that is used to identify the ongoing sequence of communicative acts
Performative	FIPA	Yes	Denotes the type of the communicative act of the message
Communication protocol	FIPA	No	Denotes the interaction protocol that the sending agent is employing
Content format	HTTP	Yes	Content type expressed as MIME type
Content encoding	FIPA	No	Denotes the specific encoding of the content
Timestamp	HTTP	No	In HTTP called `date`. Contains the date and time at which the message was originated.
Authorization		No	It is used to provide credentials that authenticate an agent.
Accept content format	HTTP	No	Denotes content types that the agent understands. An example is `text/json`
Authenticate	HTTP	No	Authentication methods ("challenges") might be used to authenticate an agent. In HTTP `www-authenticate`
Expires	FIPA	No	Denotes a time and/or date expression which indicates the latest time by which the sending agent would like to receive a reply. In FIPA called `reply-by`
Accept encoding	HTTP	No	Denotes the content encoding that the agent can understand
Cors	HTTP	No	Denotes desire to block cross-origin\cross-tenant communication
Content	FIPA	Yes	Denotes the content of the message; equivalently denotes the object of the action. The meaning of the content of any

			ACL message is intended to be interpreted by the receiver of the message.

Table 3.3.1: Message fields of multi-agent communication

3.3.2. Readiness and Performance

The communication was tested on the same configuration of robots present in Testbed for Industry 4.0. The configuration consists of four agents representing manipulators KUKA Agilus, one agent representing Motrac monorail conveyor, and a register.

The goal of the setup was to negotiate ten steps in process of building of simple car model. From Figure 3.3.2, we can see that the messaging system was responsive and kept a maximum of two messages in a queue.

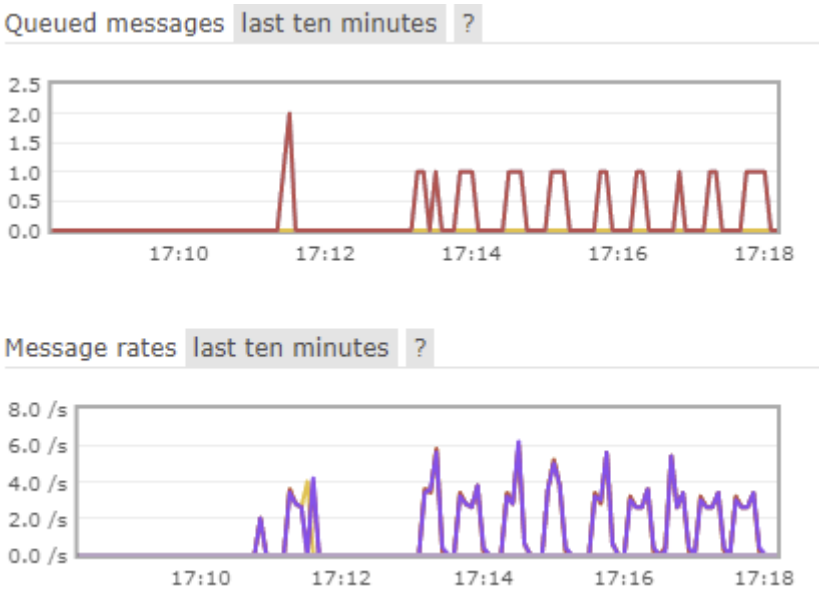


Figure 3.3.2: Performance test

3.3.3. Comparison with State of the Art

There are many archaic multiagent systems such as FIPA OS, JADE, or Cougaar, that were trying to solve the problem of multiagent messaging. Problem is that those systems were focused on different goals and none of them fulfills the requirements of a system built on top of open-source and production-ready software that is still in maintenance and suitable for industrial environment. The comparison with those systems is therefore irrelevant. At the same time, there is no other system known to the authors, that has the same focus.

3.4. Data processing pipeline for UC2

Use case 2 implements a live stream analytics pipeline composed of a few high-level blocks, as depicted in Figure 3.4.1.

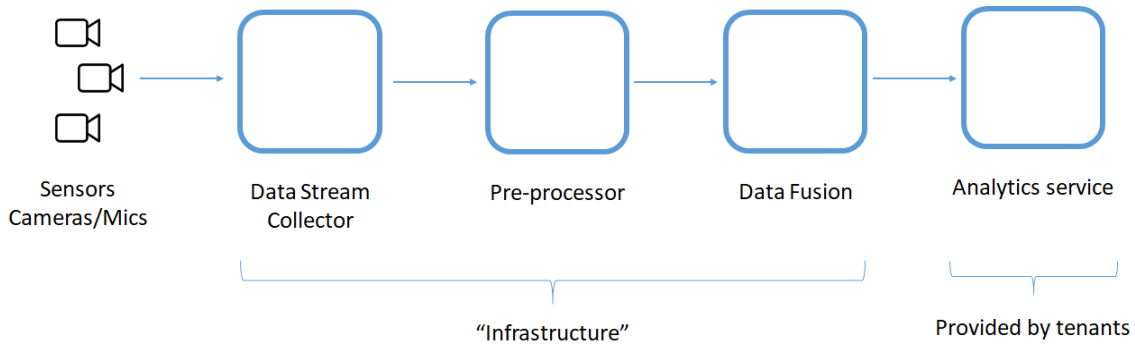


Figure 3.4.1: High-level computation pipeline for UC2

The pipeline is design to work on live captured traffic, although it might eventually include storage components at different levels. For instance, one of the analytics services could provide saving of results in a database.

The different processing blocks have heterogeneous requirements in terms of hardware, with the Pre-processor and Analytics Service being the most demanding components. In particular, the pre-processor is generally in charge of video/audio transcoding and transformation. This is a generally expensive computation, which can benefit from widely parallel processor architectures, such as GPUs. Likewise, the Analytics components can employ Deep Learning algorithms, which in turn benefit from GPUs and other accelerators.

The overall pipeline works therefore as follows: data stream collectors are connected to the data stream sources, e.g., cameras, and work as an adaptation layer. This mostly I/O workload has to do with protocol conversion and with the handling of connectivity issues. The standard streams generated by the stream collector are then provided to the pre-processor. As mentioned, this component performs transcoding, but also adaptation of video rate and quality. In fact, the pre-processor offers an external interface that can be leveraged by the platform orchestrator to adapt the video quality according to the contextual resources' constraints. The Data Fusion element that follows is a sort of "data-hub and policy enforcement point". It provides routing of the raw data streams to the analytics service. This requires potential duplication of the streams, but also chaining of the stream through multiple functions, before finally delivering it to the specific analytics component. For instance, we show a case in which we perform face blurring on a video stream before providing the video to an analytics service, in order to provide privacy protection. The last component, the analytics services, finally receive the modified stream and implement the custom analysis required by the specific application use case.

3.4.1. Video Transcoding component - VTU

The VTU acts as data stream collector and pre-processor in the data processing pipeline of UC2. It can convert audio and video streams from one format to another. The source stream can originate from a file within the local storage system, or maybe a packetized

network stream. The requested transcoding service can be monodirectional, as in video streaming, or bi-directional, like in videoconferencing. The transcoding capabilities of the VTU are provided by Libav. Libav is an open-source library, which can handle a wide variety of audio and video coding standards. For the most computationally intensive video encoding tasks, the VTU relies on Graphical Processing Unit (GPU) resources. In Figure 3.4.2 is depicted the block diagram of the SW component:

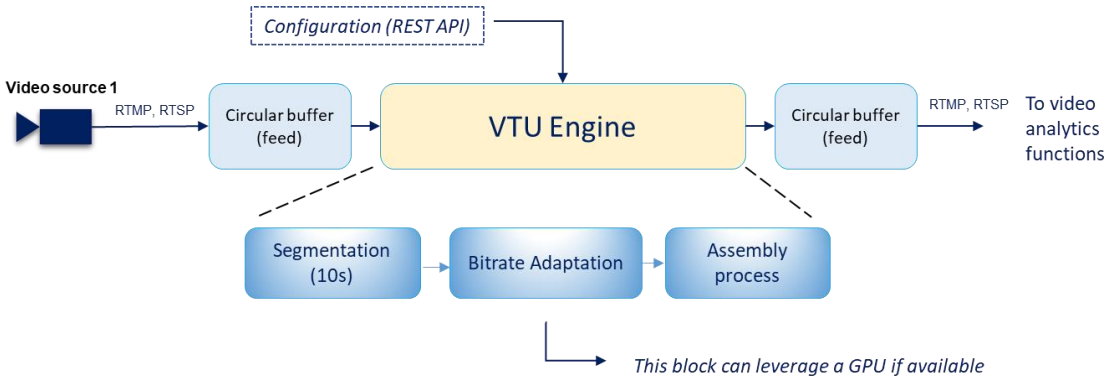


Figure 3.4.2: VTU block diagram

The main features of VTU are the following:

- Real-time video streaming management from different sources
- Video processing: transcoding, transrating, transizing
- Capability to share video among sources/users/applications
- Protocol support: rtmp, rtsp, http, hls, rtp, websocket.

3.4.2. Performance evaluation scenario

A performance evaluation was carried out on the testbed hosted at CNIT premises using the BRAINE platform. Figure 3.4.3 shows the test scenario: a recorded video, locally stored, is transcoded using first the x86 CPU and then a NVIDIA GPU hosted on the PCIe bus. In particular, the output parameter used for the transcoding task are:

Codec: H.264:

- SW, using CPU - x264
- HW, using GPU – h264 nvenc

Bitrate: 2Mb/s

Video Size: full HD

Frame size: 25

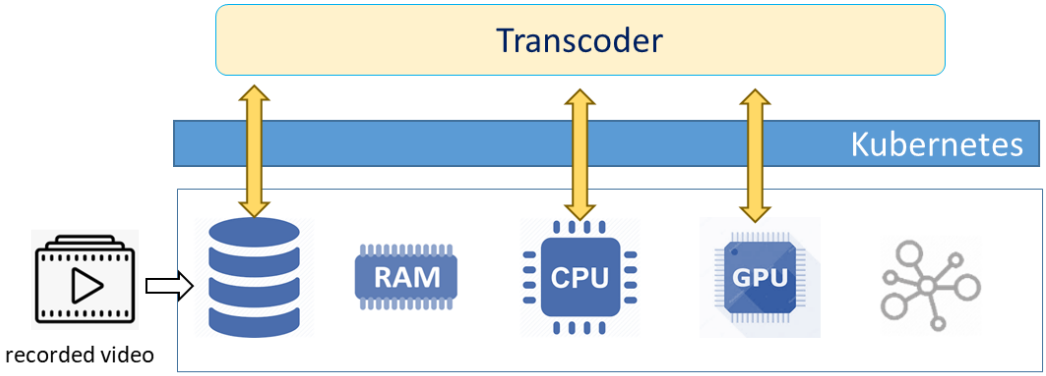


Figure 3.4.3: Performance evaluation scenario.

3.4.3. Test Results

HW used	Frame per second (max)	Speed
CPU	90	2,98x
GPU	342	11,4x

Table 3.4.1: CPU vs. GPU transcoding performance

As reported in Table 3.4.1, using a GPU allows for a 3.8x increase in transcoding performance. Further tests will be carried out on the EMDC HW as it becomes available.

4. Blockchain algorithms for resource collaboration

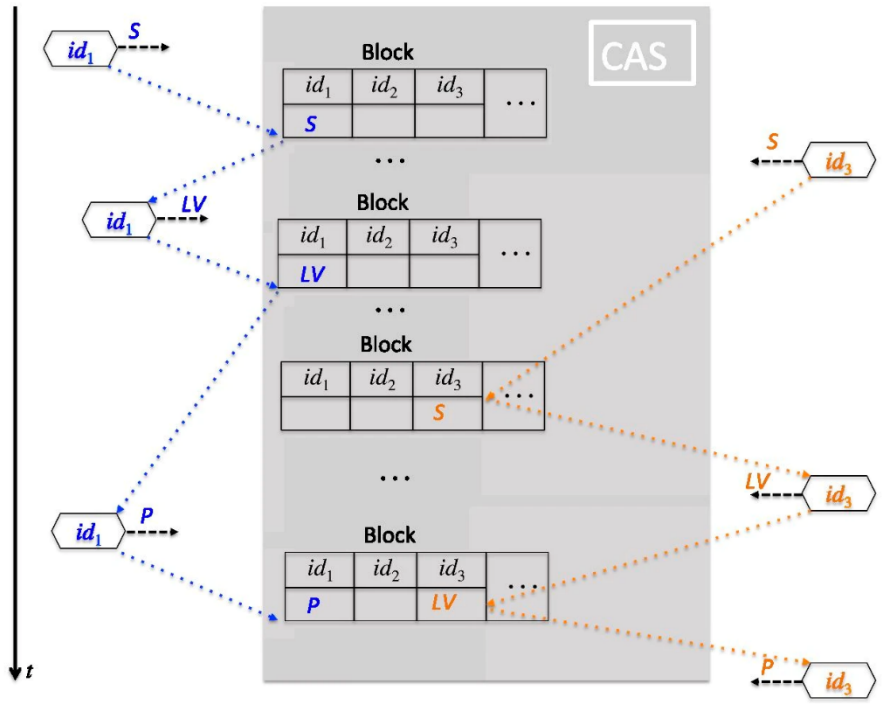
4.1. System description

This section consists of adaptations and optimizations to Byzantine agreement protocols used in KB from D3.1.2 based on findings of physical broadcast effects on edge resources collaboration developed in WP2 (T2.3).

It was established earlier that the use of LoRa communications ensures stronger resistance to interference. Due to the “capture effect”, the attacker signal has no effect on the legitimate broadcast of the sequencer until the power of the interfering signal exceeds that of the legitimate one by approximately 3db. When this happens, the legitimate signal is completely suppressed by the attacker, rather than distorted beyond the capabilities of the FEC. From the physical point of view, the attack has a hard edge, all or nothing, so the usual AI based intercept methods are unlikely to provide sufficient early warning at the physical level.

The blockchain communication on which our proposed resource collaboration is based, must use robust protocols to safeguard against such attacks. We developed and analysed a protocol, SLVP, and have found that it has the necessary security properties that give us adequate defence against an “all or nothing” attacker.

The principle of the defense is that the protocol produces a thread of S-,LV- and P-messages, each of which can be suppressed, but the IoT device monitors the Sequencer’s broadcasts and will repeat blockchain submission until the correct message is published intact on the blockchain (see Figure 4.1.1). Only then will the protocol progress to the next message. This would be hazardous, since the attacker is able to insert their own messages spoofing the user IoT device. However, due to the V-part of the LV-message, and a special search algorithm sketched in the Verify column of the last row, the Blockchain server is able to reject attack messages based solely on the preservation of a short-term secret. The cryptography used in this is Post Quantum, hash–based. The SLVP protocol makes it possible to remove intrusion detection from the physical layer (where it is expensive, and resource limited) and rely entirely on the blockchain client-server algorithms.



Block	Transmit	Verify	BC Action
b_0	$P_1 = H(N_1)$	Out of Band (Enrolment)	Post
b_1	$S_1 = \mathbf{E}^*_{N_1}(H(N_2), M_1)$	—	Post
b_2	$LV_1 = H(N_2) \oplus N_1 \,\|\|\, H(H(N_2)\|\|N_1)$	—	Post
...	...	...	...
b_n	$P_k = H(N_k)$	*as for* b_{n+3}, *assume success*	Post
b_{n+1}	$S_k = \mathbf{E}^*_{N_k}(H(N_{k+1}), M_k)$	—	Post
b_{n+2}	$LV_k = H(N_{k+1}) \oplus N_k \,\|\|\, H(H(N_{k+1})\|\|N_k)$	—	Post
b_{n+3}	$P_{k+1} = H(N_{k+1})$	for $b_n < b < b_{n+3}$: find **first** LV such that $H(P_{k+1}\|\|L \oplus P_{k+1}) = V$ **if found in block** $\hat{b}$: **unless** $\exists b \in (b_n, \hat{b}), L' \in B_b$: $H(L' \oplus P_{k+1}) = P_k$ **else**	 Post Reject

Figure 4.1.1: SLVP protocol messages

Component C2.19 consists of the following units:

1. PLS blockchain manager
2. PLS sequencer
3. Web client
4. IoT client

Unit 1 is developed using Flask in Python. The solution includes Content-Addressable Store within the server's security perimeter.

Unit 2 is prototyped as part of Unit 1 but will be separated out for uploading into a sealed sequencer, prototyped as an IoT platform using Bluetooth and LoRa communications.

Unit 3 is based on the following API (see Figure 4.1.2 - Figure 4.1.7):

Message

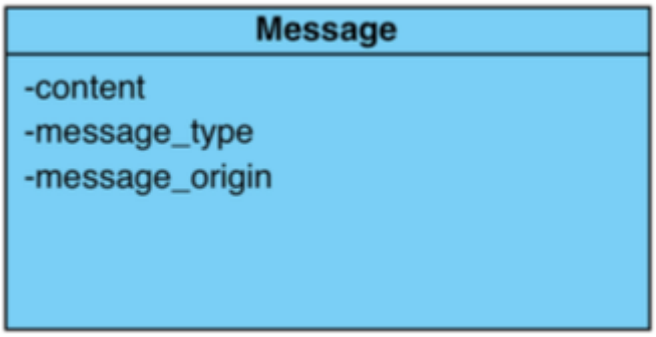


Figure 4.1.2: The Message attributes

Message types

- PROOF_PLS
- LINK_PLS
- SIGNATURE_PLS
- ENROLMENT
- ACK
- FAIL
- SINGATURE_SLVP
- LINKVERIFY_SLVP
- PROOF_SLVP

Fog Server

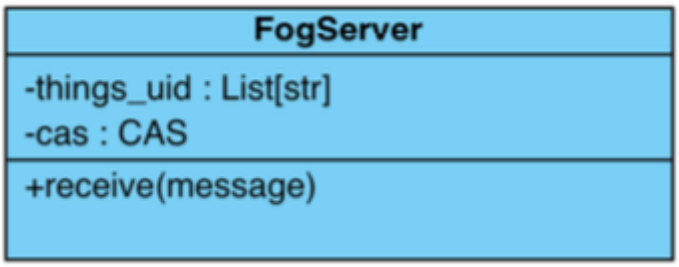


Figure 4.1.3: The Fog server attributes

The main method is ‘receive’, which receives a message and processes it according to its type.

Thing

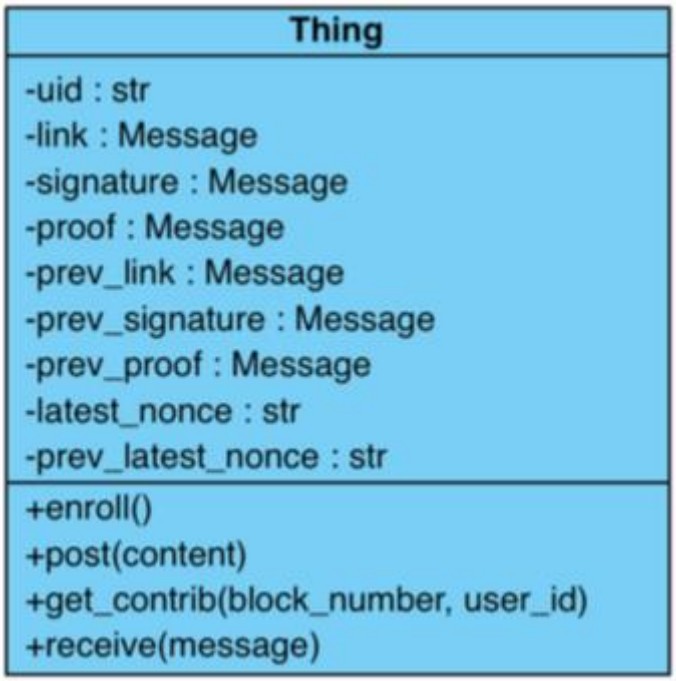


Figure 4.1.4: Thing attributes

This is an emulation of the IoT platform.

Methods:

- enroll: register the client in the blockchain
- post: post user content on the blockchain
- get_contrib: obtain a list of contributions and adjunct hashes by user ID and block number
- receive: receive a message and perform action depending on message type

Sequencer

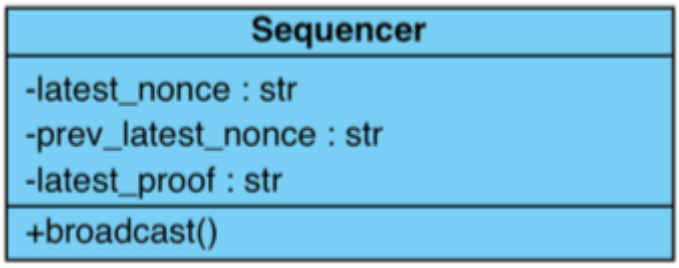


Figure 4.1.5: The sequencer attributes

The sequencer class, emulating the PLS blockchain sequencer.

The main method is ‘broadcast’, intended for sending PLS messages to all blockchain clients.

CAS

Content-addressable storage

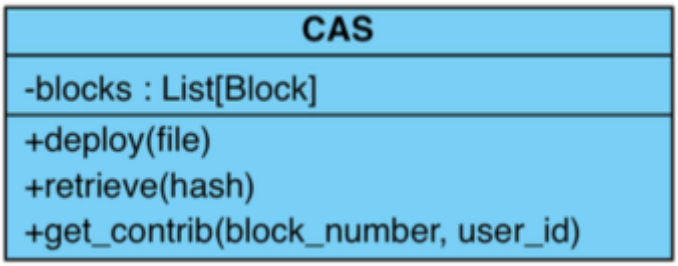


Figure 4.1.6: The CAS attributes

Methods:

- deploy: deposit a record in CAS
- Retrieve: retrieve a record based on its hash

Block

The class of the block of the PLS blockchain

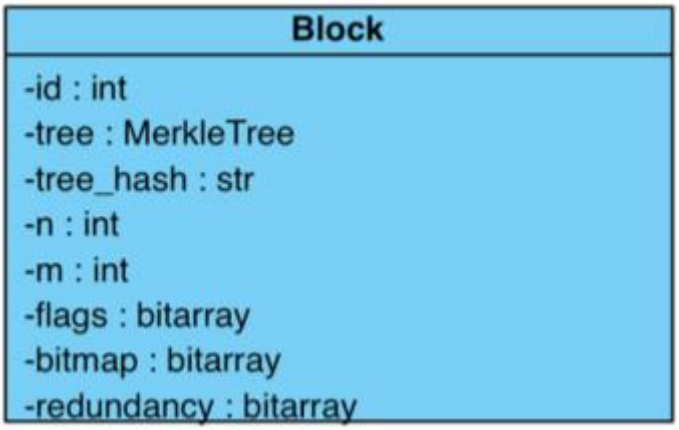


Figure 4.1.7: The Block attributes

All classes have been implemented. The Server and the Sequencer use the Python Flask package to support the Web implementation of the API.

Unit 4 has not been developed yet.

Component ID	Component Name	Development	Owner
C2.19	Low bitrate blockchain protocols	70%	IMC
GitLab Repository: N/A Containerized: N Registered on BRAINE platform image registry: N Deployed as a pod and functional on BRAINE platform: N			

Integrated with other platform components: In progress This component is being integrated with Use Case 1 applications

Table 4.1.1: BRAINE service mesh component for SLVP protocol

4.1.1. Level of readiness

The protocol stack and services have been implemented in Python to demonstrate the feasibility of the blockchain solution based on the proposed protocols (See Table 4.1.1). This comprises subsystems:

- Fog_server
- Sequencer
- IoT blockchain comms library
- CAS

The server-side code can remain in Python form due to the low rate of transactions in an IoT swarm, which makes it possible to use interpreted code without risk of a performance bottleneck. However, the IoT blockchain library requires an implementation in C++ since that is the language low-power platforms are coded for to achieve the best performance and resource utilisation.

IoT blockchain comms library in C++ is the main outstanding item.

4.1.2. Performance vs state of the art

The main problem for an underpowered IoT platform in using a block chain is access to the blocks. Since the platform is communication-challenged (duty cycle of 1% or down to 0.1% on LoRa with a maximum of 50Kbits/sec, effective indexing is the key issue (see [Shafarenko1]). We researched available indexing structures of the blocks and developed an original one, the Merkle-Tunstall (MT) Tree, using which it has become possible to reduce the communication load to suit the available bandwidth.

Figure 4.1.8 shows some evaluation data:

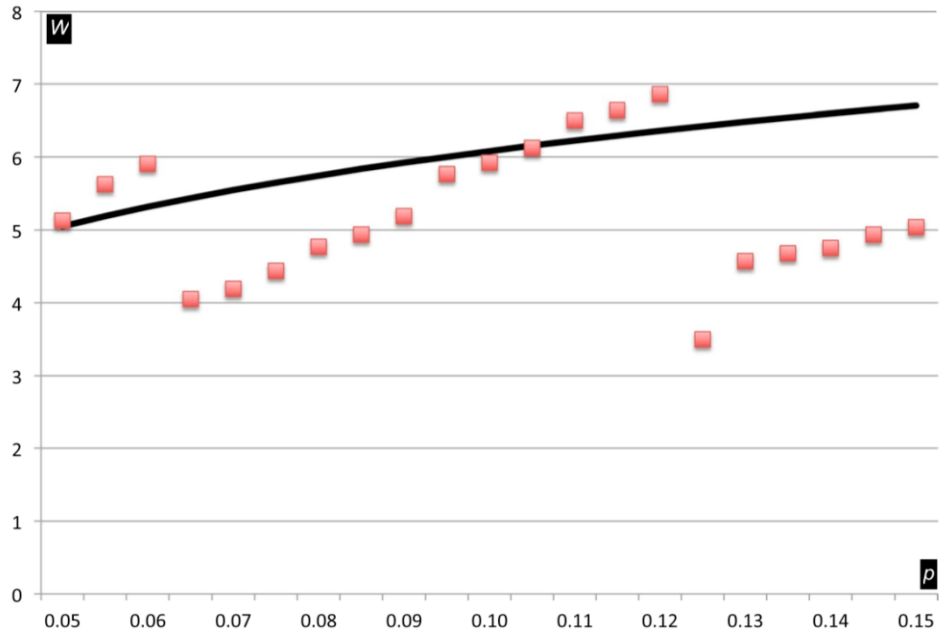


Figure 4.1.8: The amount of communication (scattered dots) for the MT is typically better than the standard Merkle-Patricia Trie (MPT, solid line)

The vertical access is the average path length W from the root (which is the Root of Trust) to a leaf for a swarm of 1024 IoT devices. The horizontal axis is for the probability *p* for a given IoT device to contribute to any given block. We can see that the amount of communication (scattered dots) for the MT is typically better than the standard Merkle-Patricia Trie (MPT, solid line). At the same time, MT is a zero-cost solution for proofs of absence. At *p*~0.1, the search for records is zero cost in 90% of the blocks, and in the 10% it is on a par or better than the standard MPT. This demonstrates that our solution is up to an order of magnitude better than MPT (see [Shafarenko2]).

5. Repository of the edge node capabilities

This section presents the latest updates regarding the final version of the BRAINE repository system that runs on edge nodes. The updates focus on the data model, workflow definition, and registry interface functionalities derived from instrumentation, monitoring, and classification sources.

5.1. Data Model

Over the last interaction, we have extended the vocabulary to support workflow placement and description. The schema was extended to support Images descriptions through Docker deployments descriptor format (Listing 5.1.2). Services, using Kubernetes deployment descriptors (Listing 5.1.1) and Workflows using Argo description language which is an extension of Kubernetes deployment description language itself. We call those descriptions manifest. By relying on an attribute called manifest we allow the use of the same model later to other existing description languages. Figure 5.1.1 gives an overview of the vocabulary developed for resource management and service deployment.

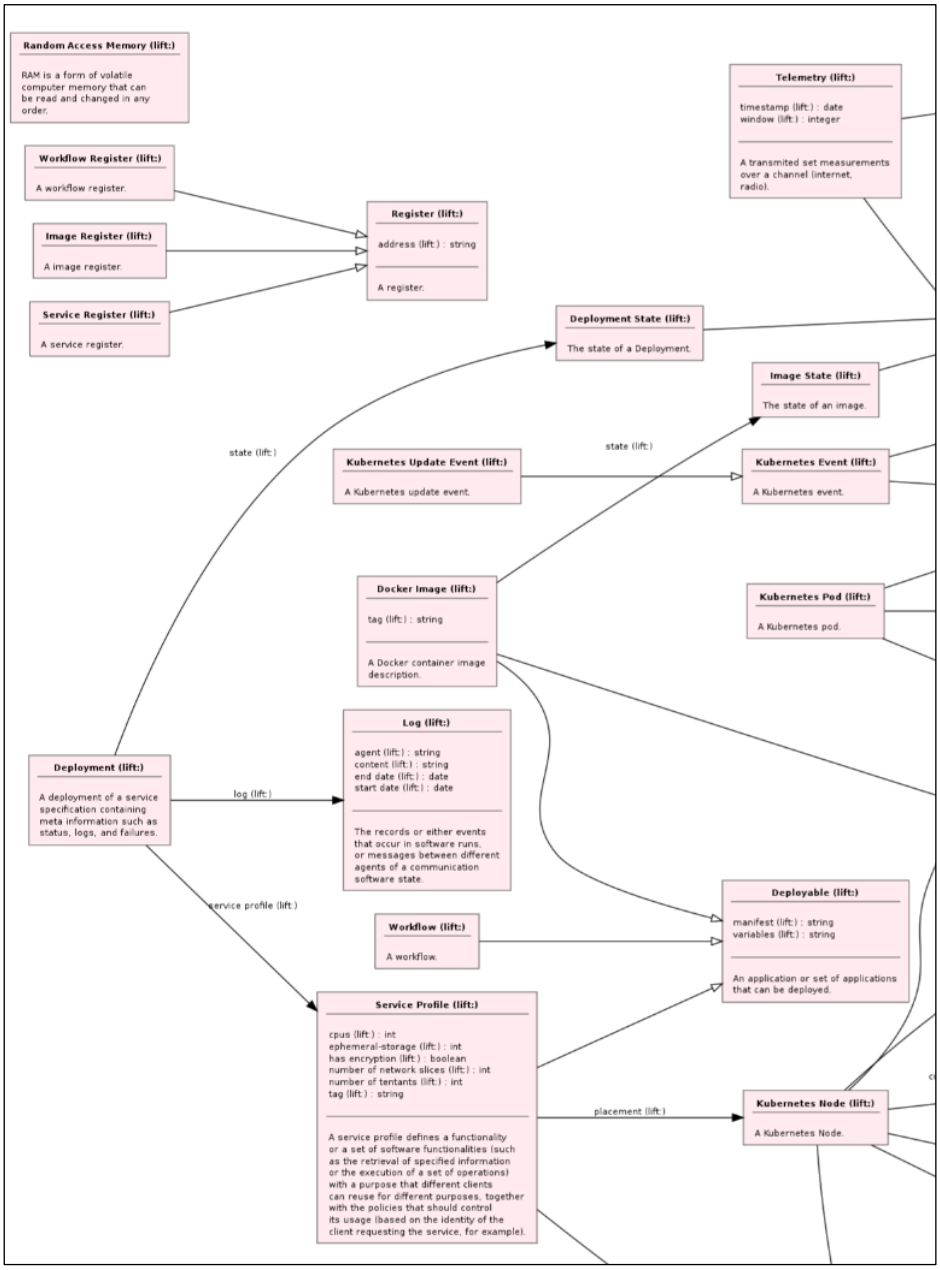


Figure 5.1.1: Excerpt of BRAINE schema (lift) highlighting Service, Workflow and Image Descriptors.

```
kind: Pod
metadata:
       labels:
       run: helloworld
       name: helloworld
spec:
       runtimeClassName: rune
       containers:
       - command:
       - /bin/hello_world
```

```
      env:
      - name: RUNE_CARRIER
      value: occlum
      image: helloworld
      imagePullPolicy: IfNotPresent
      name: helloworld
      workingDir: /run/rune
EOF
```

Listing 5.1.1: Kubernetes manifest example.

```
FROM alpine
CMD ["echo", "Hello BRAINE!"]
```

Listing 5.1.2: Image manifest example.

In addition, we have also extended the vocabulary (Figure 5.1.2) allowing users to add services, images and workflow register endpoints to the BRAINE knowledge graph. This extension allows users to directly interact with the endpoints without the necessity of using the Service or Image Orchestrator. This approach significantly simplifies the deployment and management.

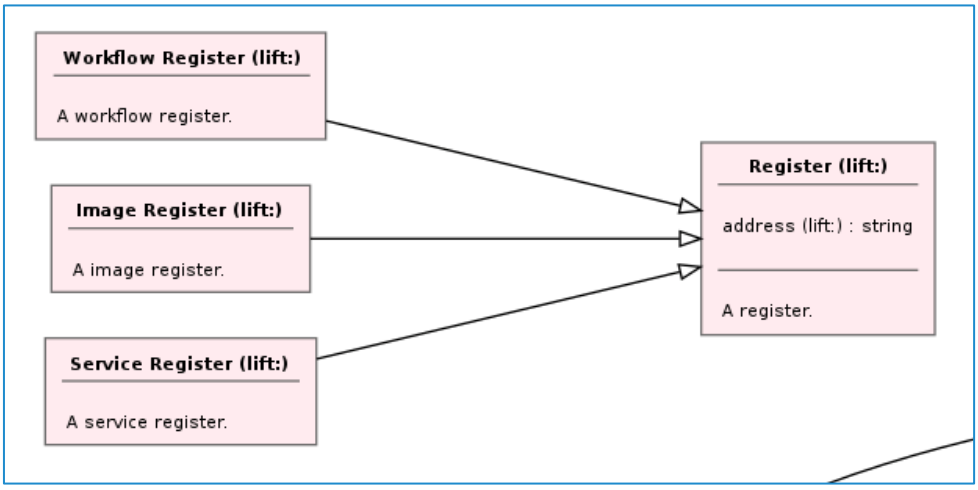


Figure 5.1.2: Excerpt of BRAINE vocabulary lift highlighting Workflow,

Workflow Definition For being open source and tightly integrated with Kubernetes, Argo was chosen as the service workflow execution framework. In addition, it offers all required functionalities in the project scope and users can describe workflows in a declarative way using manifests in a similar fashion to those of Kubernetes and Docker.

With the addition of the Argo framework to the BRAINE software stack, it is possible to define workflows through Argo workflow definition language (Listing 5.1.3). Argo can be easily coupled with the overall project architecture and can be easily managed by the user as well as by the system. Argo's tight integration with Kubernetes, its declarative workflow definitions, and its support for event processing make it the most suitable candidate. It is likely that effort will be required to integrate Argo into the BRAINE architecture and develop its functionality further; but this is preferable to building a custom BRAINE solution from scratch and may provide useful contributions to the

opensource solution. The full vocabulary is available under Creative Common CC-BY-4.0 license at https://github.com/eccenca/braine-vocab.

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: hello-world-
  labels:
    workflows.argoproj.io/archive-strategy: "false"
  annotations:
    workflows.argoproj.io/description: |
      This is a simple hello world example.
      You can also run it in Python: https://couler-proj.github.io/couler/examples/#hello-world
spec:
  entrypoint: whalesay
  templates:
  - name: whalesay
    container:
      image: docker/whalesay:latest
      command: [cowsay]
      args: ["hello world"]
```

Listing 5.1.3: Example of workflow definition.

5.2. Registry Interfaces

In addition to the schema extension, we have also further developed a BRAINE management webclient that allows the management of services, images, workflows and their respective registries (Figure 5.2.1). Figure 5.2.2 displays the Docker Image register window in CMEM, it allows users to register Docker images for deployments. Figure 5.2.3 displays the Service Profile Register Window that allows the registering of Services through Kubernetes Deployment description files. In both windows there is an attribute *manifest* which is used to either register Kubernetes Deployment descriptor in case of Service Profile and Docker Image Descriptor in case of Docker Images.

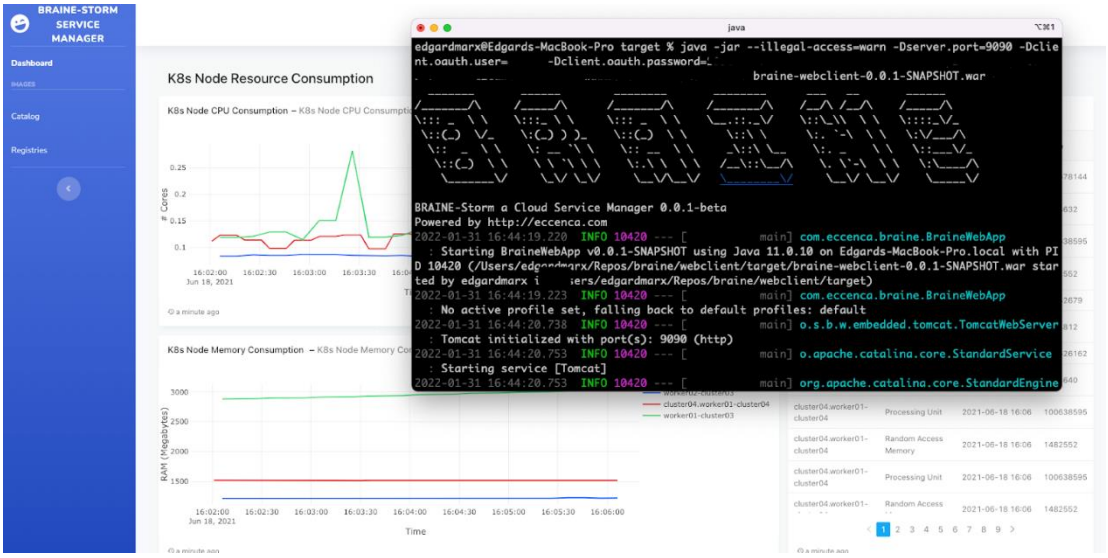


Figure 5.2.1: BRAINE webclient.

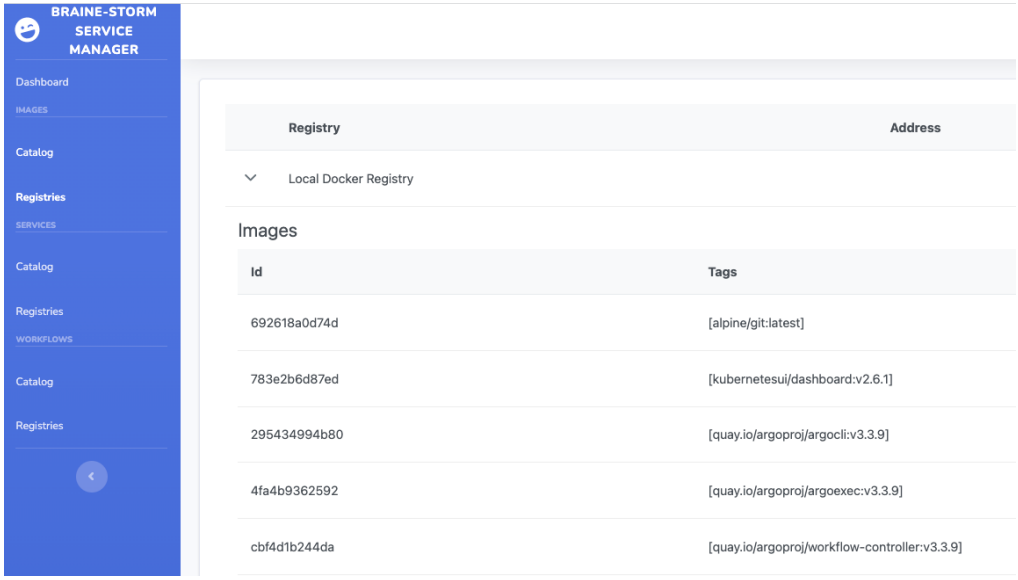


Figure 5.2.2: BRAINE Web Client Image Registry Web Interface.

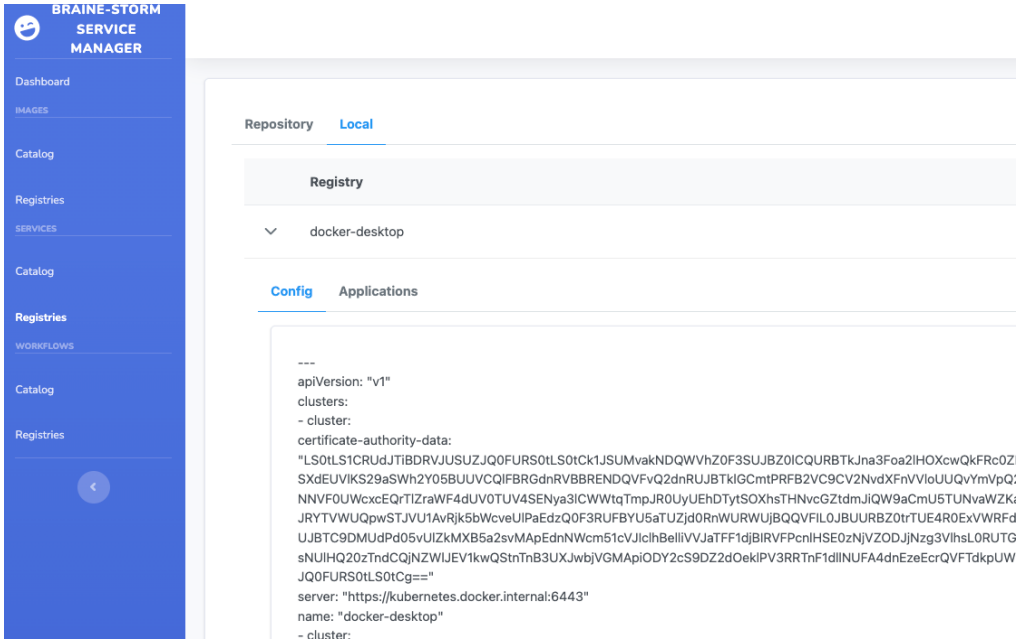


Figure 5.2.3: BRAINE webclient Service Registry Web Interface.

5.3. Progress beyond state-of-the-art

The last work months were pivotal for consolidating advances not just in the data model but as well in the architecture. We finish the first version of the full data model containing all concepts necessary for managing distributed workloads by adding concepts such as workflow registries as well as descriptors that allow us to describe and instantiate them. This approach differentiates the BRAINE platform from other vendor solutions that focus on different aspects of the distributed workload execution, BRAINE solution simplifies the management by allowing users to manage the whole data model in a single data-place.

Further we simplify the architecture from the previous version by developing a web-client application that communicates directly with the different components in the BRAINE architecture and simplify application usage by hiding complex communication APIs and protocols through friendly user interfaces. Figure 5.3.1 shows the old architecture while Figure 5.3.2 shows the new one. It is possible to see the removal of two components and the addition of the BRAINE Web Client. Besides the addition of the Workflow Engine, in the new architecture there is no need for the Service and Image Orchestrators since the BRAINE Web Client communicates directly with the Workflow Engine as well as Global Service and Image registries.

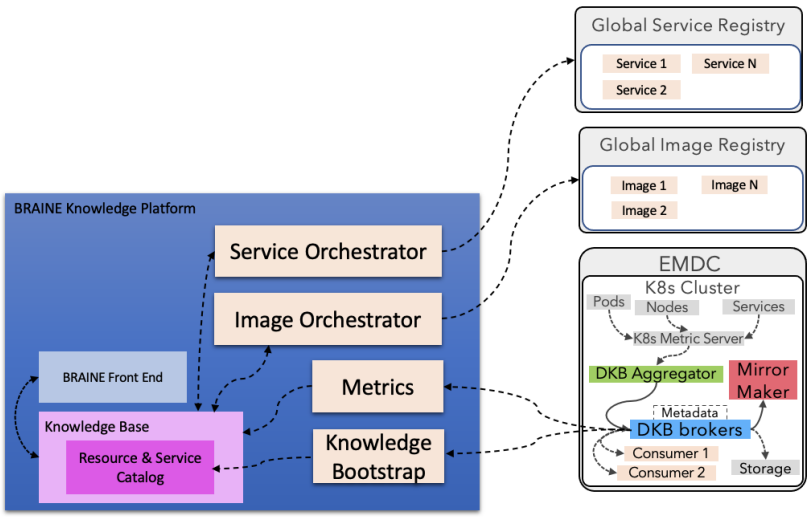


Figure 5.3.1: Old Architecture.

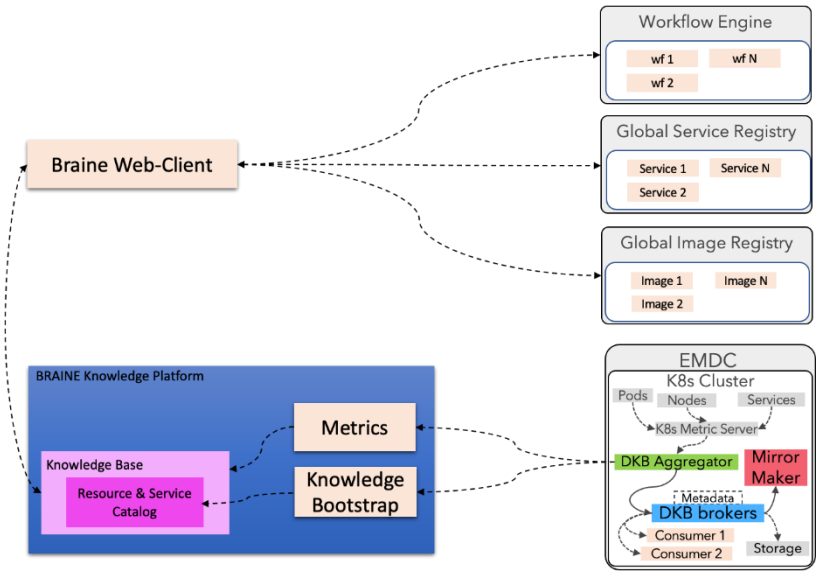


Figure 5.3.2: New Architecture.

5.4. Performance evaluation

To measure the performance of consuming data through Metrics EMDC consumer and to evaluate the scalability of the approach we measure the semantic lifting in four different scenarios. All data used in this scenario were synthetically generated, based on previously collected data. The synthetic generated dataset simulates ten clusters generating metrics data simultaneously. We measure the data lifting off on 10, 100, 1000, and 10000 data points. The graph in Figure 5.4.1 shows that the semantic lifting scale linearly growing from 10 to 10000 metrics while reducing the processing time per entry from approximately 0.5 seconds to 2 milliseconds at its pick. This discrepancy is due to Metrics relay network performance optimization that transmits the whole read metric data chunk at time instead of individual entries.

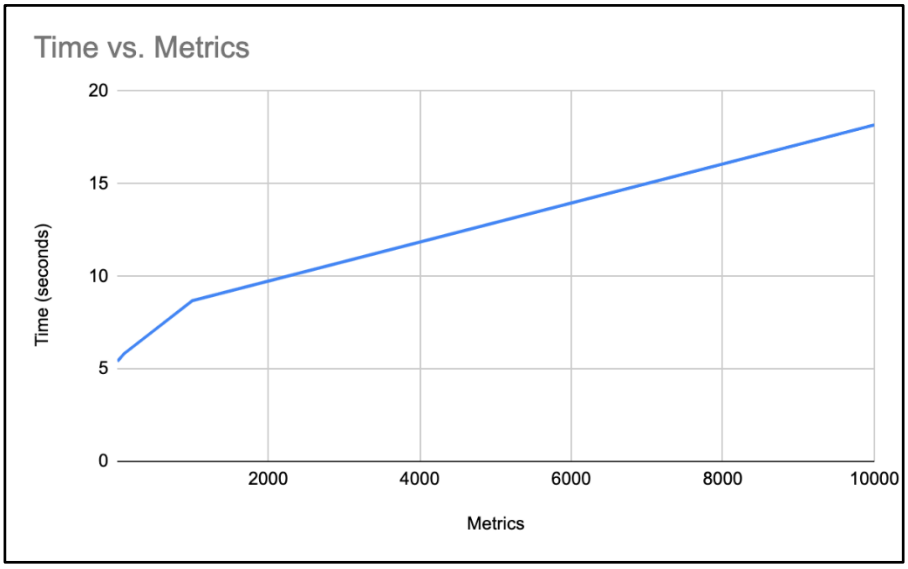


Figure 5.4.1: Evaluating the semantic lifting of metrics to Corporate Memory (CMEM) through Metrics relay.

6. L2 scheduler (slice scaling) for vRAN

6.1. Technical description

The main assumption is thus that the current EMDC has resources (CPU, MEM, storage) that are limited to a certain quota - per 5G vRAN cluster (or set of pods). But there are alternative EMDC's available in the federation where resources can be used to expand the current footprint of the vRAN. Here the vRAN would benefit from its disaggregation in order to offload only a part of its stack to the secondary EMDC.

The scaling (or L2 scheduling) function is considered after the 3GPP auto-scaling high level feature that is considered in order to perform slice on-the-flight reconfiguration when necessary. As such it works based on the architectural assumptions and capabilities of the BRAINE architecture. Therefore, the scaling is functionality made for the purpose of user data handling at scale. We are dealing solely with the CU-UP so the user-plane data is being offloaded to a EMDC in the current federation. At the current stage of 5G vRAN deployment it is already a function that poses a challenge for the dynamic and online adaptation of the vRAN that is not available on the market. The major interaction of the vRAN with AI/ML adjustment is via Workload Placement Agent in cooperation with the micro-orchestrator in order to a) find appropriate EMDC and b) to orchestrate an instance of CU-UP in that EMDC prior to user data request being redirected there from the primary EMDC. That is why the relatively high "look ahead time" from prediction models (Prediction Agent) would be beneficial in order to successfully prepare all the workload activities in the other EMDC in advance.

Therefore, the vRAN is consuming the inputs from the BRAINE Workload Placement Agent which is aware of the current traffic demand and the future predictions thereof. The predictions are provided by the AI/ML models of the Prediction Agent that is consuming telemetry data from vRAN and the K8 cluster/pods. The metrics are related to the RU/DU/CU components of the vRAN. As such Workload Placement agent is tightly coupled with the operation of the admission controller (AC) inside the vRAN stack, as well as the micro-orchestrator of the vRAN - the latter being the proprietary solution participating to the workload scheduling of selected vRAN component. The component currently selected for scaling is centralized unit (CU). CU contains protocols that are not real-time bound, they can operate even if latencies are relatively higher that in a real-time regime. That is why the CU itself can operate easily in the cloud - but this does not assume scaling mechanisms themselves so far (as this feature is under development).

From this perspective we are considering the interactions between 5G stack and the BRAINE architecture according to the figure (Figure 6.1.1) below.

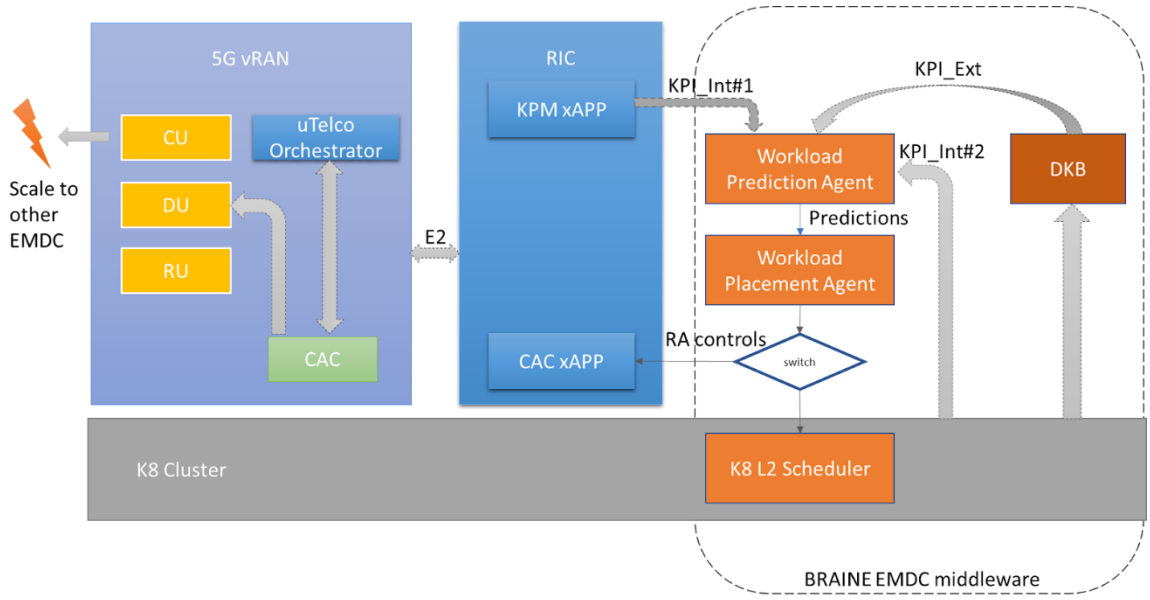


Figure 6.1.1: Conceptual diagram of the vRAN data handling framework.

In the figure there is virtual 5G network (with only vRAN and RIC presented for clarity) instantiated in the EMDC as K8 pods (part of a cluster). Inside the vRAN there are all the essential boxes (yellow ones) that constitute the 5G radio stack. The blue boxes represent the boundary of the 5G vRAN/network which constitutes the workload that is under study in both the WP3 and WP4 - so for scaling and for data handling at scale. The central element as regards the network scaling is the Workload Prediction Agent, cooperating with the Workload Placement Agent. The two components are collecting and processing metrics from multiple sources: a) the BRAINE DKB delivering CPU, RAM and in future also networking indicators of the external EMDC nodes, b) local resources of the EMDC K8 cluster where the 5G vRAN is deployed (as pods) and c) the radio resources provided by the Key Performance Metrics captured from the radio stack and delivered over the E2 interface to the KPM xApp and exposed outside vRAN to the BRAINE Workload Prediction and Workload Placement agents. Having the metrics collected, the decisions of the local admission controller (AC) together with the Workload Placement Agent, needs to cooperate in order to decide for the micro-orchestrator (telco aware). It is the micro-orchestrator that is responsible to deploy selected part of the disaggregated 5G when necessary.

In the above configuration the following programming languages are used:

- vRAN - C99
- xApps - JAVA
- Workload Prediction - python + pyTorch
- Workload Placement - python + pyTorch.

6.2. Workload prediction using AI/ML models

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.

Figure 6.2.1 demonstrates 5G DU CPU consumption for 2 hours. This scenario was used to train the long short-term memory (LSTM) model to forecast CPU consumption. In order to support data handling in such edge networks where high level of flexibility of load placement and migration are essential, we have performed some vRAN profiling sessions. During such profiling sessions we have focused mainly on the CPU usage, depending on the number of users.

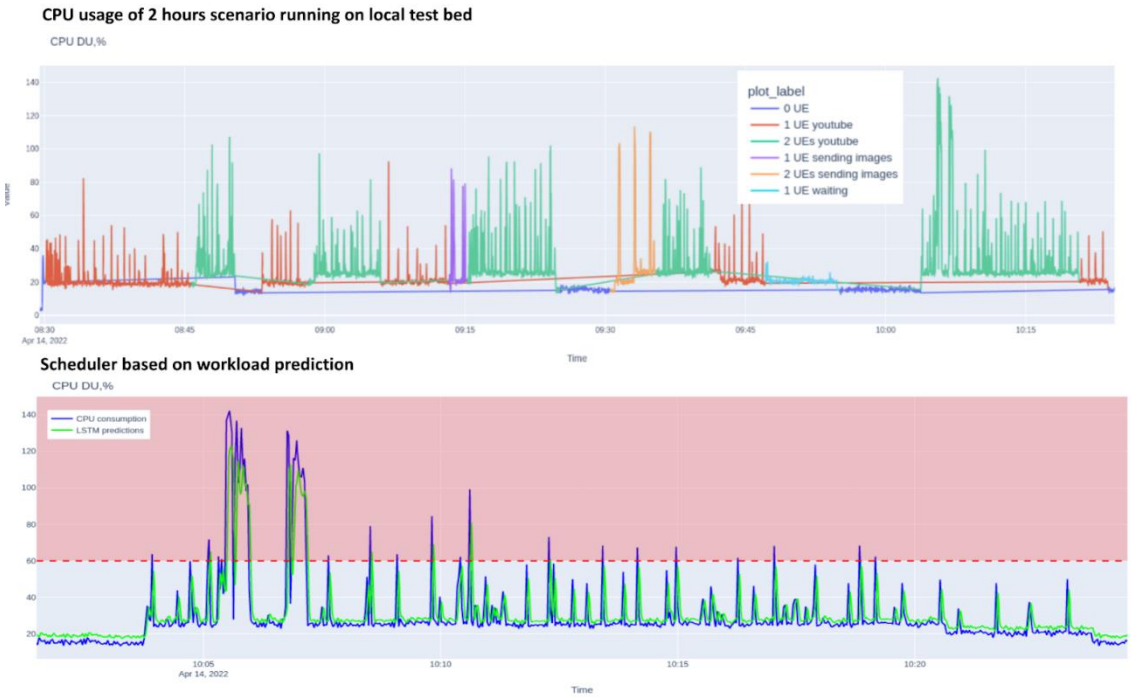


Figure 6.2.1: Data Usage for Workload Scheduling.

The top plot presents the DU unit CPU values evolution in time. It can be seen that variation of CPU is relatively high at times and can reach beyond the 100 percent - in case 2 UEs are actively using the YouTube sessions (e.g., from 10:05). The bottom plot on the other hand demonstrates the time evolution of the zoomed-in 2-minute CPU readings when the CPU usage is evolving towards exceeding the maximum thresholds (virtual machine quota). Having the predicted CPU behaviour ahead of it exceeding the threshold - the system could trigger the auto-scaling of vRAN CU e.g., to another EMDC server in the federation of servers.

This way owing to KPI metrics time evolution it is possible to apply AI/ML (e.g., LSTM, ARIMA (transfer learning), N-Beats or similar) to build prediction models. With such prediction models it is then possible to properly scale vRAN resources depending on the user's traffic envelope changes. In our case scaling is considering the horizontal-scaling of CU between edge servers. This way PDCP related, encryption based PDU processing can become more adapted to traffic changes according to a typical tidal effect.

6.3. Test results

In our experiment, the scenario model consisted of 6902 observations made each second, which is almost 2 hours of CPU usage from vRAN. This scenario observed several metrics such as throughput, primary resource block (PRBs), UEs connection, sending and receiving data etc.
We ran different size data pictures via email for transmitting purposes and also ran different quality YouTube videos ranging from 720p to 1440p for receiving purposes. We got the dataset from running the vRAN in this experimental setup.

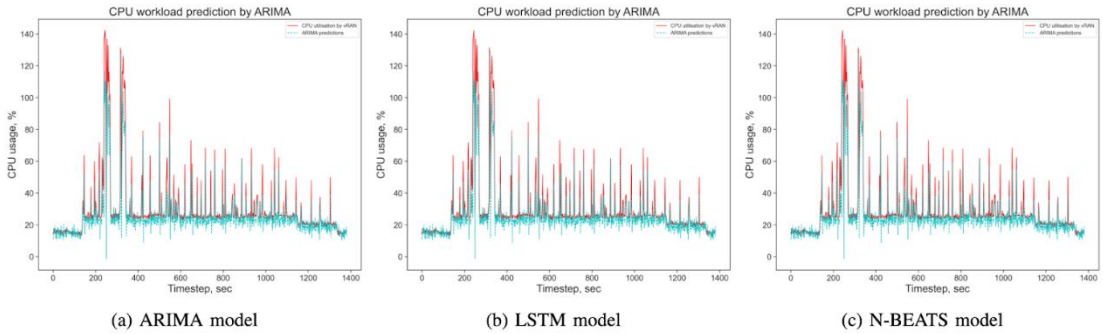
Obtained dataset was split to train and validation in 80:20 ratio, thus train data comprised 96 minutes of observation and for validation there was 24 minutes observation of CPU usage.

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. Based on the collection data, we have built several ML models for workload prediction and results will be described in the following sections.

In this section, 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.

We describe the modelling process of the ARIMA. Firstly, obtained data from the experiment was tested on autocorrelation and stationary to train the ARIMA model. Applying Dickey-Fuller test [24] to the data, we conclude that our series is stationary, thus order of differencing is set to (d=0). For the ARIMA model, we selected the autoregressive parameter (p) and moving average (q) as 2 and 20. As a result we obtained an ARIMA model with parameters (2,0,10) and it was trained on the training dataset.

On each step the model was trained with 20 historical observations and then trained models predicted future CPU usage. At the same time weights of the trained model were adjusted based on current observations. In other words, transfer learning of the ARIMA model (Figure 6.3.1-a) is an iterative process in which the model is retrained after prediction. Figure 6.3.1 presents the prediction on validation dataset of CPU utilization by vRAN. With transfer learning approach performance of the model is much better than predictions based on the model without retraining.



(a) ARIMA model (b) LSTM model (c) N-BEATS model

Figure 6.3.1: CPU workload prediction for proposed ML models.

The advantages of LSTM over recurrent neural networks (RNN) are the ability to forget or to take into account long-term dependencies. For training the LSTM model dataset was processed with a sliding window technique where the window was set to 151 records (2.5 minutes). After the transformation, the train dataset had 6599 windows with 151 records each and the validation dataset transformed to 1381 windows with the same number of observations.
Before training initial weights of the model were provided with uniform distribution $U_{[-0.08,0.08]}$. For this modelling, we set the hyper parameter values as follows: epoch to 600, learning rate to 0.03, hidden states to 55, recurrent layer to 1, and dropout to 0.1. Figure

6.3.1b shows the workload prediction for LSTM models. 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.
To train this model we used multivariate time series data which in addition to CPU usage parameter comprises number of connected user equipment (UEs) as presented as categorical variables. In the first stage of N-BEATS training vanilla model was trained to find optimal initial learning rate with learning rate range test [25]. From the test we obtained that initial learning rate equals (21 * 10^-4). For other hyper parameters, we set the values as follows: number of blocks to 2, number of fully connected layers to 4, size of fully connected layers to [300, 2048]. Workload prediction of CPU usage by N-BEATS model presented in Figure 6.3.1c and its result is the worst among the three models. In order to measure the accuracy of the specified models, we consider mean absolute error (MAE) and mean absolute percentage error (MAPE). MAE indicates the difference between predicted value and true value of an observation. Also, MAPE defines the statistical measurement accuracy of a ML algorithm for a fixed dataset. Both terms are referred to as a loss function for defining error by the model evaluation. To reflect the effectiveness of the ML models, we got the experimental values of 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.
Table 6.3.1 and Table 6.3.2 show the states of the presented components.

Component ID	**Component Name**	**Development**	**Owner**
C3.19	**Placement Agent**	**60%**	**ISW**
GitLab Repository: Containerized: Y Registered on BRAINE platform image registry: Y Deployed as a pod and functional on BRAINE platform: Y Integrated with other platform components: Y – Status Report: Placement agent is under development, it will be integrated with the mini-orchestrator of the 5G vRAN. The focus now is on mechanisms of the “scheduling” (scaling) between machines, as they require certain activities on the side of stack (relinking the DU module with CU-UP / CU-CP modules) for the 3GPP F1 and E1 interfaces.			

Table 6.3.1: Component status (C3.19).

Component ID	**Component Name**	**Development**	**Owner**
C3.22	**Baseline Scheduler 2**	**100%**	**ISW**
GitLab Repository: Containerized: Y Registered on BRAINE platform image registry: Y Deployed as a pod and functional on BRAINE platform: Y Integrated with other platform components: Y – Status Report: The scheduler is under development with our implementation team. Developments			

require the update of the 5G vRAN adjustments to make sure that when needed for 5G workload to scale, it will be possible to scale the CU-UP (user plane) functionalities of the CU (centralized unit). Additionally, the integration with the DKB agent will be required on our side.

Table 6.3.2: Component status (C3.22).

7. Conclusion

This report (i.e., the final report on the status of WP3 - part 1) concludes the efforts made by BRAINE WP3 partners in three areas: 1) Mapping of AI data-processing workloads in Edge Nodes, Blockchain algorithms for resource collaboration, and Repository of the edge node capabilities. Mainly, the report describes the workflow, interfaces, functionalities, readiness status, performance, and the comparison with the state-of-the-art for all the areas mentioned above. In addition, the report includes links to the software implementations in the BRAINE Gitlab account.

8. References

[Helsinger]: Helsinger, Aaron et al. “Cougaar: a scalable, distributed multi-agent architecture.” 2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583) 2 (2004): 1910-1917 vol.2.

[Omicini]: Omicini, Andrea. “L1 – JADE: Java Agent DEvelopment Framework.” (2015).

[Poslad]: Poslad, Stefan et al. “The FIPA-OS agent platform: Open Source for Open Standards.” (2006).

[Shafarenko1]: Shafarenko, A.. (2021). A PLS blockchain for IoT applications: protocols and architecture. Cybersecurity. 4. 4. 10.1186/s42400-020-00068-0.

[]Shafarenko2: Shafarenko, A.. (2021). Indexing structures for the PLS blockchain. Cybersecurity. 4. 10.1186/s42400-021-00101-w.

[Trendalyze1]: Time Series Intelligence and AI 3.0, whitepaper, https://trendalyze.com/wp-content/uploads/2019/12/Trendalyze-Introduction.pdf

[Trendalyze2]: Scientific Approach for Visual Motif Discovery, whitepaper, https://trendalyze.com/wp-content/uploads/2019/02/Scientific-Approach-of-Visual-Motif-Discovery.pdf