ERAIA - Enabling Intelligence Data Pipelines for IoT-based Application Systems

Aitor Hernandez, Bin Xiao, Valentin Tudor
Ericsson Research, Stockholm, Sweden,
Email: {aitor.hernandez.herranz, bin.xiao, valentin.tudor}@ericsson.com

***Abstract*—Establishing upon the connectivity layer provided by Internet of Things (IoT) platforms, modern industries are moving towards management and computation solutions which enable Artificial Intelligence (AI) services for data intensive applications. This raises two important challenges: first, the information carried by data should be refined and prepared for the various AI algorithms via data processing pipelines and; second, a distributed orchestration solution for data and AI computation resources featuring with migration capabilities is required to support the refining process. In order to address these challenges, this paper introduces ERAIA, an actor-based framework which provides a novel basis to build intelligence and data pipelines. ERAIA facilitates the deployment and migration of distributed AI computations for heterogeneous and dynamic IoT scenarios. An implementation description is accompanied by relevant performance evaluations to demonstrate the flexibility and scalability of the solution. ERAIA provides an interface to expand the scope of existing IoT systems as Application Enablement Platform (AEP), which hence accelerates the development of AI-based IoT solutions.**

***Keywords*-Internet of Things, industrial IoT, complex event processing, distributed processing, edge computing, artificial intelligence, machine learning, IoT landscape, reactive systems, actor model**

I. INTRODUCTION

Internet of Things (IoT) is a key enabler for industry digitization. With the industrialization of IoT, the explosion of data generated by devices is unavoidable. Hardware development (such as Tensor Processor Unit (TPU) and Graphics Processor Unit (GPU)) significantly increase systems' capabilities for processing big data and conduct Artificial Intelligence (AI) computations; nevertheless, frameworks with clear IoT targets to enhance the computation capabilities of distributed systems are still rare. This leads to extra demands on an IoT platform in order to enhance the intelligence computations while: 1) fulfilling latency requirements on AI computations and data processing for critical use-cases, 2) offering computation capabilities to provide quick insights through distributed data resources and, 3) supporting live configuration updates in dynamic environments.

This paper proposes a novel solution for the computation related demands in data-intensive IoT applications. ERAIA, a reactive system[1] built using an actor model, intends to provide a responsive, resilient and elastic system required by the IoT landscape [1] (from edge devices, to gateways, network infrastructure and data-center). By making use of all the end-to-end nodes for conducting divided computation tasks, AI computation and data processing are spread between components in the IoT landscape. Heavy raw data do not always need to travel from edge devices to the cloud, and by providing edge-centric data processing, latency can be reduced; communication resources, particularly the bandwidth, is optimized through filtered and processed data, and nodes' computation utilization is thus increased.

As main contributions, in this work we: a) present common challenges of horizontal Industrial IoT (IIoT) applications and solutions (cross-sectors), b) introduce a framework that simplifies the creation of intelligence and data pipeline for heterogeneous and dynamic scenarios, c) conduct the internal evaluation of an implementation that shows the flexibility and scalability of the solution.

The paper is structured as follows. In Section II, we present the common challenges that IIoT systems and applications share, followed by the related solutions. In Section III, the architecture of the proposed framework, ERAIA, is characterized in detail, and in Section IV a possible implementation is described followed by an internal validation of the solution. In Section V, we summarize the work and suggest directions for future improvement, and finally, we conclude this work in Section VI.

II. CHALLENGES AND RELATED WORK

Starting from the high-level aspects within different IIoT sectors, this section describes common characteristics and requirements that need to be satisfied by a replicable IIoT solution. Before enumerating the challenges, we describe previous work related to IoT background: a description of different IIoT sectors and their verticals is provided in [2], [3] presents a horizontal perspective of IoT applications, while [4] depicts how industrial applications can be built using common components with a focus on intelligence.

Our solution aims to enhance IoT platforms and Application Enablement Platform (AEP) by enabling capabilities in all the horizontal categories i.e. Extract, Transform, Load (ETL), predictions, pattern detection and classification [5], event detection and notifications, visualization, security and privacy [6], and Cyber-Physical System (CPS) [7]. The following subsections cover challenges specific to the IIoT, which are not fully addressed by the current state of the art, together

[1] https://www.reactivemanifesto.org/

978-1-7281-4657-7/20/$31.00 ©2020 IEEE

with proposed approaches to evaluate how these challenges are tackled.

A. Data management

Enabling intelligence of IoT raises requests to process the data generated by the sensors for discovering patterns and extracting knowledge, which hence needs to manage the data effectively. Among data management topics in heterogeneous IoT systems, data ingestion, data serving and data processing becomes relevant to extract, understand and expose data between different entities. Ingesting or serving data can be done via various protocols, message brokers, or by querying from databases. Processing together with serving those data online improves IoT systems' functionality via transformation, accumulation, serialization, aggregation and/or compression, and/or even advanced calculation such as Machine Learning (ML), which further enriches data utility.

To understand the impact of heterogeneous systems and how they affect data serving and processing, we identify two aspects for the evaluation: performing a profiling to understand the dynamics of the implementation, and measuring the latency or delay introduced from the application layer.

B. Latency and throughput

For data-intensive systems like IIoT, the latency affects the processing efficiency of complex algorithms, such as deep neural networks. Reducing the latency enables the implementation of real-time applications (i.e. CPS) and provides quicker delivery to stakeholders of insights extracted from sensor data.

Regardless the improvement of communication and processing hardware, software architectures with clear IoT targets for efficient orchestration would bring a large benefit in tackling this challenge. Splitting the total latency and throughput cost into individual shares attached to the various IoT solution components provides insights into how they can be further optimized via resource orchestration.

C. Artificial Intelligence computations

AI methods (e.g. ML) are extensively used for IoT data analysis to support various smart service use cases.

In IoT, there are strong needs to deploy and run the algorithms in a flexible way to immediately retrieve insights from the data streams. Therefore, AI orchestration in IoT systems has become rather important, which optimizes the resource utilization of the systems conformed by heterogeneous resource-constrained components.

In order to evaluate AI composition, it is important to examine the footprint requirements, the possibilities to ingest and expose data using different protocols, and the downtime when a model is updated or re-deployed in another node.

D. Distributed computing and orchestration

In order to exploit and benefit from the distributed nature of IoT systems, challenges such as weak scalability and weak interoperability due to the heterogeneity need to be dealt with. One way to accomplish this is to employ virtualization techniques, such as docker containers [8]–[10], unikernels [11], cloud resource containers [12] and other lightweight options such as node-red [13], [14] or actors [15]. These solutions play an essential role in managing and orchestrating the computation capabilities among the distributed components.

To evaluate the efficiency of distribution and orchestration, it is important to analyze the scalability enabled by the architecture. Specifically, it can be measured by how the system is improved when adding more resources to an existing node (namely scale-up or vertical scaling); and/or how the system is improved when more nodes are added (namely scale-out or horizontal scaling).

E. Live reconfiguration

One of the main characteristics of IoT is the dynamism of the environments and the need to adapt to changes rapidly while minimizing the impact on the application. The live reconfiguration challenge covers changes such as protocol and/or data adaptation, migration or relocation of computation loads, algorithm changes or ML model updates.

In order to evaluate the impact of the reconfiguration, it is possible to measure the influence of latency or packets loss on the application layer.

F. Intelligence and data composition

The IIoT challenges covered in this section can be grouped under the common umbrella of intelligence and data composition. This set of challenges can be tackled by solutions employing intelligence pipelines which handle system heterogeneity (Subsection II-A), intelligence extraction (Subsection II-C) and computation distribution on remote nodes (Subsection II-D), while providing a quick response (Subsection II-B) and allowing online changes (Subsection II-E).

A number of studies and proposed solutions partially covering these challenges have been identified. Young et al. [16] present an architecture using open source components that allows building intelligence and data pipelines. Nakamura et al. [17] propose a solution called Information Flow of Things (IFoT) that performs distributed processing and analysis of data streams near their sources based on recipes. Cheng et al. [18] suggests a solution for on-demand edge analytics based on the orchestration of docker containers as wrappers of the different operations. Najdataei et al. [19] introduces a framework which distributes and parallelizes the processing based on results from a bottle-neck detection algorithm, enabling continuous data stream processing. Dias et al. [20] provide an extensive survey for distributed data streams processing frameworks.

Table I compares the most extensively used solutions which covering the challenges mentioned above. It gives a snapshot of the positioning of ERAIA in the landscape of the existing state of the art IoT frameworks and solutions.

From the data management perspective, existing solutions target either data ingestion and serving or data processing with limited data serving. ERAIA integrates data ingestion, processing and serving to better adapt protocol serializations and encoding, or to modify the data for easier exposure to

other services. From the model serving perspective, existing solutions provide a way to pack up each model (i.e. in a single docker container) with limited interfaces to execute the model, using HTTP and/or gRPC. In contrast, ERAIA aims to provide flexible data integration for ingesting and exposing the results. Additionally, re-utilization of the libraries along with the worker package is beneficial for scaling-up. The configuration is done at runtime via an Application Programming Interface (API) that allows changes in real-time for either data ingestion/serving or processing of any computation including AI. With that, the actor-based toolkit simplifies the migration of the elements from one worker to other different workers. Section III will provide a specific description of ERAIA's architecture.

In addition to the projects in Table I, there are also some commercial solutions targeting some of these challenges: a) Foghorn[11], a product for IIoT real-time edge intelligence covering data ingestion and data processing with AI capabilities, b) Keptware[12], a product providing IIoT data ingestion supporting many of the industrial and proprietary protocols, c) EdgeX Foundry[13] and d) Mainflux[14], open source projects providing IoT platforms to the edge. Multiple analyst reports cover available commercial products in more detail [21]–[23].

III. ARCHITECTURE OF THE SOLUTION

Once the main challenges have been discussed, this section proposes ERAIA, a solution with the following features: i) a distributed system that can dynamically expand across multiple nodes ranging from edge to cloud in the IoT landscape, ii) a flexible system which can be reused in various scenarios and be integrated into heterogeneous infrastructures, iii) seamless online data ingestion and data processing with live reconfiguration, iv) highly interoperable with state of the art protocols, which facilitates the integration with other solutions and technologies, v) a well-defined API exposed to other systems (e.g. IoT platforms, AEP) that allows for application functionality deployment in a distributed fashion.

ERAIA is a reactive system[15] and therefore inherits its features: responsive, resilient, elastic and message-driven. Built using an actor model, it provides a decentralized management system to deploy high performance distributed computations that can scale up using the resources efficiently, and scale out with the addition of new nodes.

Figure 1 depicts the architecture of the solution divided in four categories of components:

external provides the API to compose/decompose the intelligence and data pipeline based on ERAIA, and enables external applications to interact with the framework.

cluster manager creates and manages a fault-tolerant peer-to-peer cluster. This component takes the responsibilities to distribute and place the workloads using a scheduler and/or location based policies.

worker is the component that executes the pipeline elements provided via the external API. It is the component responsible for the data plane, which establishes the connection to the devices and/or the message brokers.

user interface facilitates the visualization and management of the system exposed via the API.

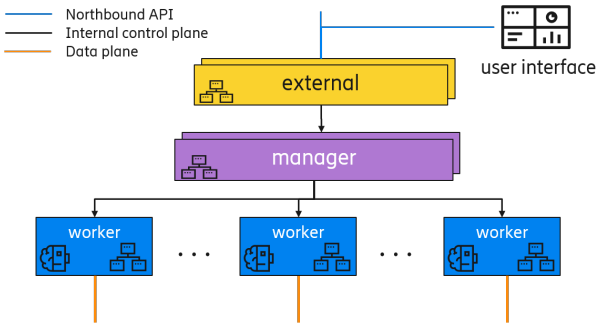


Fig. 1: ERAIA architecture with four classes of components and the different types of interfaces

The following subsections will provide detailed descriptions on the system components and how they tackle the challenges presented in Section II.

A. The deployment and infrastructure

The deployment, together with the infrastructure, need to be considered when dealing with the heterogeneity of the components, in order to provide interoperability while supporting scalability. The proposed solution provides the following enriched capabilities: i) it supports multi-platform environments (e.g. amd64, arm and arm64), ii) it is easily deployable and packageable, iii) it builds up distributed peer-to-peer computation, iv) it dynamically reconfigures the computation within the cluster, v) it has no restrictions for virtualization technologies (e.g. docker images, unikernels, Virtual Machines (VMs)). Furthermore, it supports two deployment modes: a) a standalone mode whereby all the required components run in the same deployment package, b) a distributed mode whereby each component runs on different deployment packages among remotely distributed nodes.

B. The interfaces

The proposed solution provides three different types of interfaces as shown in Figure 1:

northbound API exposes the system capabilities by facilitating the integration with existing IoT platforms. Furthermore, it enhances the capabilities of the AEP by onboarding AI solutions, based on a distributed deployment involving components ranging from small gateways to data centers (e.g. HTTP, gRPC, or a message bus);

control plane used to configure, manage and keep the cluster alive (e.g. TCP/IP);

data plane used for the processed and transformed data from users and applications initiated or terminated exclusively

[11]https://www.foghorn.io/
[12]https://www.kepware.com/
[13]https://www.edgexfoundry.org/
[14]https://www.mainflux.com/
[15]https://www.reactivemanifesto.org/

TABLE I: Comparison of main features

		DATA SERVING	*DATA INGESTION*		*DATA PROCESSING*		*MANAGEMENT*		
	Name		*Multi-protocol*	*IoT protocols*	*Transform.*	*AI-capable*	*Interfaces*	*Config.*	*Distributed*
	ERAIA	✓	✓	✓	**by code**	**external libraries***	**REST API and/or UI**	**at runtime**	✓
DATA INGESTION	Apache NiFi[2]	✓	✓	limited	by libraries	-	REST API and/or UI	at runtime	with zookeeper
	Apache Storm[3]	-	✓	limited	multi-language	Samoa or S4	Jar package	at deployment	with zookeeper
	Apache Spark Streaming[4]	-	✓	-	Spark Engine	by code	multi-language API	at deployment	✓
DATA PROCESSING	Apache Flink[5]	-	connectors	-	by code	-	multi-language API	at deployment	with configuration
	Apache Spark[6]	-	connectors	-	by code	✓	multi-language API	at deployment	✓
MODEL SERVING	Apache Samoa[7]	-	-	-	by code	Apache Storm	Java API	at deployment	limited
	Kubeflow[8]	-	-	-	-	external libraries**	REST API and/or UI	at deployment	with kubernetes
	Clipper[9]	-	HTTP	-	-	Python libraries	CLI	at deployment	with kubernetes
	Seldon[10]	-	HTTP/gRPC	-	-	external libraries**	CLI	at deployment	with kubernetes

* included when packaging the framework, shared.
** included when packaging the application, not shared.
[2]http://nifi.apache.org/, [3]http://storm.apache.org/, [4]http://spark.apache.org/, [5]http://flink.apache.org/, [6]http://spark.apache.org/, [7]http://samoa.apache.org/, [8]https://www.kubeflow.org/, [9]http://clipper.ai/, [10]https://github.com/SeldonIO

on the worker components (e.g. temperature and humidity sensor values sent using a MQTT broker).

C. The protocols and communication

The communication components of the framework are established based on the underlying communication protocols selected and used in the data plane. The API is able to dynamically define and update the communication protocols that are used.

The data plane is managed by a reactive streaming data library which provides high-performance connectors for most of the available IoT protocols (and systems) that can be easily integrated with our solution: AMQP[16], Kafka[17], MQTT[18], HTTP, CoAP[19], LWM2M[20] or storage systems such as MongoDB[21] and Elasticsearch[22]. It also allows for integration of heterogeneous and brownfield[23] devices into the same platform.

D. The intelligence and data composition

The intelligence and data composition refers to the process of on-loading intelligence computations by creating and configuring a series of computations on the ingested data resources. To make such process flexible and re-configurable, the composition is based on a set of intelligence processor units, configured as a set of distributed and organized flow of computations forming the intelligence and data pipeline. Given the decoupling of control and data plane, each of the communication endpoints are initiated or terminated on the processor units, and not on the manager component. This allows for each of the processor units to be deployed on different worker nodes without impacting the communication. Therefore, it facilitates a rapid migration and redeployment of the individual units, fulfilling the requirements of a vivid environment. Based on this, we define:

1) The Intelligence and Data Pipeline (IDP): an end-to-end data pipeline composed of one or more individual execution units called Intelligence Processor Units (IPUs);

[16]http://www.amqp.org/
[17]https://kafka.apache.org/
[18]http://mqtt.org
[19]http://coap.technology/
[20]http://www.openmobilealliance.org
[21]https://www.mongodb.com/
[22]https://www.elastic.co/products/elasticsearch

[23]legacy devices and protocols with or without possibilities to be updated

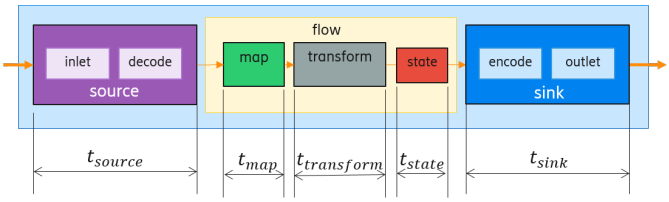


Fig. 2: Intelligence Processor Unit (IPU) decomposition

2) The Intelligence Processor Unit (IPU): the atomic unit used to compose the pipeline, which can be distributed among workers deployed in the IoT landscape, where each unit conducts computations for certain partitions of the whole AI task. The IPU is depicted in Figure 2.

An IPU consists of several blocks:

source is the data ingestion block: *inlet* encloses the data from the connector library (e.g. MQTT library) to an object. Subsequently, *decode* converts the data based on the serialization, and exposes it as the expected data type (e.g. received data from the topic X at the MQTT broker Y with JSON+SenML).

map provides Complex Event Processing (CEP) mechanisms in conjunction with defined intervals that control the output flow (i.e. accumulate, sum, minimum, maximum, first, last, standard deviation). For example, accumulate each element received during a 10 second interval, and provide the list to the next block.

transform executes a generic or AI function (e.g. enhance data semantics by metadata annotation, rule-based reasoning), or ML functions (e.g. deep neural networks and Bayesian models). This block is compatible with external libraries (and/or tools) for creating customized ML or AI functions.

state manages the persistency of data required for some algorithms on consecutive executions. For example, output data from the previous execution can be used as an input to the current execution.

sink is the data serving block: *encode* converts the data based on the expected serialization. Subsequently, *outlet* packages the data to be sent by the specific connector. For example, send data to the topic X in the Kafka broker Y. An *interval* parameter throttles the outgoing data flow.

Figure 3 shows an example of the intelligence and data composition: (1) sensor data is sent to a message broker; (2) a processor receives, cleans and accumulates sensor data every minute; (3) another processor receives the accumulated data, applies a function to compute the value required to control an engine and sends the output to the message broker; (4) another processor receives the accumulated data, applies a regression model to predict future state and forwards the result to another message broker; (5) an actuator device receives the output from (4) and applies it to the actuator of an engine; (6) data is received by an external entity (e.g. for reconfiguration and optimization, visualization, or intelligent consumers); and (7) alternatively, sensor data could be sent directly to the IDP..

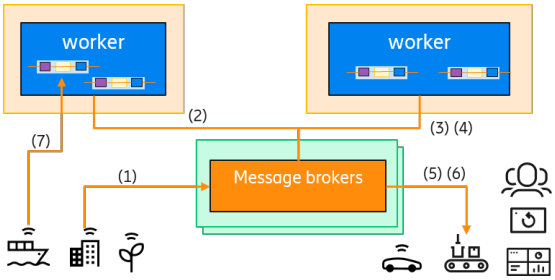


Fig. 3: Intelligence and data composition with a pipeline of four IPUs distributed across two different workers

IV. STUDY ON IMPLEMENTATION

In this section we present one specific implementation of ERAIA based on the architecture described in the previous section. ERAIA employs a reactive system actor model and it is implemented on the Java Virtual Machine (JVM) and akka[24]. It can be seamlessly integrated with: 1) akka-cluster, providing a decentralized peer-to-peer cluster, 2) akka-persistency, simplifying the recovery after failure, 3) akka-streams, enabling asynchronous, non-blocking, and backpressured stream processing, and 4) Alpakka, a reactive enterprise integration library.

The transformation functions can be implemented using either compiled (e.g. Scala or Java) or interpreted (e.g. Python) programming languages. Scala, a functional programming language in Java, is used as the default since ERAIA's implementation is done in Scala [25]. Python[26] is used for its access to an extensive number of AI libraries and to facilitate their deployments by data scientists and data engineers.

A. Performance evaluation

We have conducted an evaluation of the previously described implementation, based on the following experiments: i) a software profiling analysis to measure the execution time of the processor units, ii) the scalability of the system in terms of the maximum number of supported devices and data rate, and iii) an analysis of the migration of processor units to quantify the impact of moving a processor unit from one worker to another.

1) Evaluation setup: Figure 4 shows the infrastructure setting for the evaluation. It employs VMs (*cXmY*), which are deployed on Openstack (KVM Hypervisor) in a private cloud located in Lund, Sweden. The rest of the IoT components, RaspberryPi based devices (*rpi3-arm/arm64*) and Intel UP (*up-amd64*), are located in our labs in Stockholm, Sweden. For all platforms and architectures, the workers run in docker containers for a seamless deployment. Table II summarizes the characteristics of the hardware used.

In addition to the hardware used for ERAIA's implementation, several VMs are used to provide additional services: a device generator and several message brokers. The device

[24]https://akka.io
[25]Just-In-Time (JIT) compilation is used for the Scala functions.
[26]using Java Embedded Python (JEP), https://github.com/ninia/jep

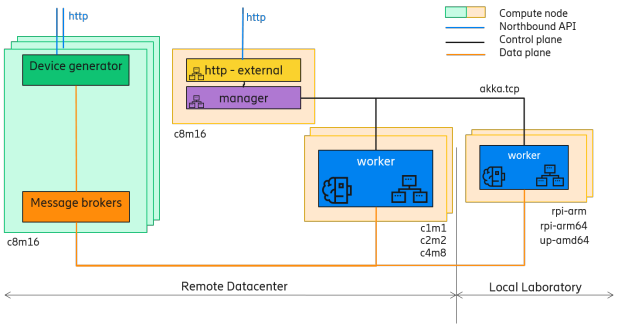


Fig. 4: Infrastructure for the performance evaluation

generator is employed to emulate devices and expose the values of data generated by those devices in the specified communication protocol. Also, an API is provided to create/remove devices, which can currently support the following communication protocols: AMQP, HTTP, Kafka, CoAP and MQTT. For the evaluations, the impact of the virtualization layers is neglected, as demonstrated in [24]. Given that the evaluations have low I/O operations, their impact is neglected as well, as demonstrated in [25].

TABLE II: List of hardware used in the evaluation

Name	Processor	Arch.	Num. CPUs	RAM [GB]	Num. Threads	Power [W][1]
c1m1	Intel Xeon E5-26xx [2]	amd64	1	1	7686	8.75
c2m2			2	2	15748	17.5
c4m8			4	8	63614	35
c8m16			8	16	128115	70
rpi3-arm	ARM Cortex-A53	armv7l	4	1	13655	2
rpi3-arm64		aarch64	4	1	6638	2
up-amd64	Intel Atom x5-Z8350	amd64	4	8	61502	4

[1] Thermal Design Power (TDP) used as reference. For *cXmY*, $power_{processor}/CPUs$ is used
[2] Within our data center, multiple types of processors are used. Selected E5-2628L as reference with 70W and 8 cores

2) Software profiling: This evaluation provides a dynamic program analysis to measure the duration of the stages in the processor unit, depicted in Figure 2. The evaluation results deliver information regarding how the implemented solution behaves across varying platforms, CPUs, and memory. The total or processing time of a processor is defined as:

$$t_{processing} = t_{source} + t_{map} + t_{transform} + t_{state} + t_{sink}$$

where t_{source}, t_{map}, $t_{transform}$, t_{state} and t_{sink} are the timings of the IPU blocks described in Section III-D.

It is worth mentioning that $t_{processing}$ is not the delay introduced by the IoT application employing the intelligence pipeline, since it only refers to the IPUs. Extra delay could be caused by the connectors, network or message brokers. This is covered in more detail in Subsection IV-A4.

In order to measure the time cost for the execution of multiple processor units during the profiling analysis, an outbound interface is used to expose them periodically. The overhead introduced by the interface is neglected for the purpose of this evaluation. Data is collected during a period of 20 minutes per platform and communication type. During this period, a device sends data every $100ms$. [27]

Figure 5 shows the impact of different communication and processor types employing a transformation function that forwards the message, either in Scala ({`x: Map[String, Any] => x`}) or Python (`output = input`). Using a message broker (i.e. AMQP, Kafka or MQTT) with Scala, the internal delay is around $1ms$; in contrast, using the same message broker with Python processors, the time can vary between $60ms$ and $75ms$. In Scala, the transformation function introduces low value delays ($2\mu s$) due to its compiled code, compared to the $60ms$ delay for Python. When using Python, the transformation includes i) time cost of Python interpreter and ii) time cost for converting data types from Scala to Java and then from Java to Python.

As observed for both processor types, the protocols HTTP, HTTP-bus[28] and CoAP have a larger t_{sink} compared to the message brokers. This is due to the synchronous requests to the corresponding servers. The same result also applies to t_{source} for HTTP. HTTP-bus achieves a low t_{source} given that the HTTP server runs in the processor unit by itself. CoAP achieves a low t_{source} because the Observe mechanism is used, and the request is not included in the time measurement.

Figure 6 shows the t_{sink} for some of the data serving protocols that ERAIA supports. For the HTTP, t_{sink} includes the HTTP request time; for the LWM2M, it includes the time cost for mapping the original value to the IPSO object; and for MQTT-SenML, it includes the time cost for converting to a non-optimized SenML encoder. For any other case, it is assumed that a JSON encoder is used.

Figure 7 shows the total time cost for executing a Scala processor unit and its impact on different platforms and architectures. All the time costs for the VMs (cXmY) have similar levels given that the same CPU is used constantly during the evaluation. There is a small increase on the *rpi-arm64* and *up-amd64*, approximately $2ms$ and $1ms$ respectively compared to the VM; meanwhile, the *rpi-arm* has a bigger penalty of $10ms$.

3) Scalability: It can be defined in two ways: 1) scale-up or vertical scaling, which refers to the impact of the performance when there is an increase in the memory size and/or number of CPUs, and 2) scale-out or horizontal scaling, which refers to the impact of performance when further nodes are added.

Given the distributed architecture of the solution, an approximation of the scale-out capabilities could be defined as:

$$capacity_{total} = \sum_{i=0}^{num_{workers}} capacity_{worker_i}$$

To analyse the scale-up capabilities of the system, an outbound interface is used. During the execution of the ex-

[27] Outliers are removed after standardizing the variables and using an absolute threshold of 3 for the z-scores $z = \frac{x-\mu}{\sigma}$ and $|z| > 3$

[28] An HTTP-bus refers to a HTTP server running inside the processor unit, while HTTP refers to an external HTTP server

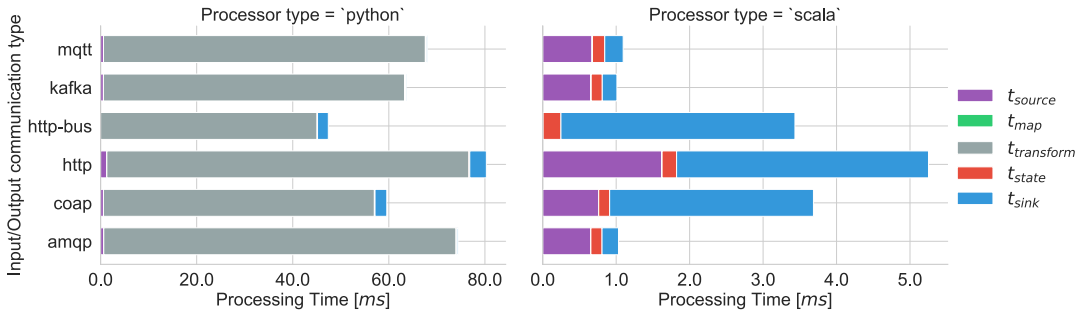


Fig. 5: Processing times per communication type and processor type on a VM with 8 CPUs and 16GB RAM (c8m16)

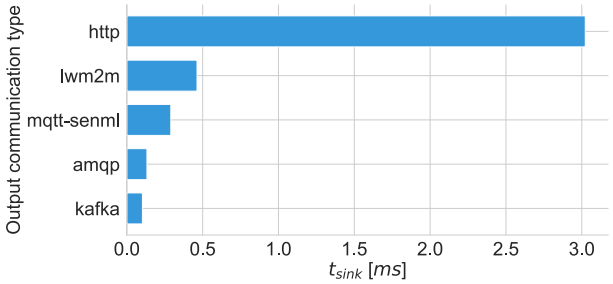


Fig. 6: t_{sink} comparison for CoAP data forwarded to different output communication types using Scala processor

periments, a monitoring device sends annotated data every second, whose processors units are adding a timestamp[29] and additional information to the packet[30], and then storing the information into a database. All the other devices have only forwarding processors units associated. The execution of the experiment is based on a periodic increase of the number of devices in a given time period. The following values are used: devices-step=50, max-devices=5000, timeout-step=2min. The devices generating data have the following characteristics: communication-type=MQTT, interval=100ms, data-encoding=JSON, packet-size≈40 bytes. Three device generators and three MQTT brokers are deployed in order to simulate a high load[31].

Values are collected until the established maximum number of devices is reached or the worker component terminates. The maximum values presented in Table III reflect the number of devices when the packet loss is above the threshold of 5 % over a period of 30 seconds, or the delay is above 1 second[27].

Table III and Figure 8a present the scale-up capabilities of the workers. As expected, the lower the CPU and memory, the lower the number of devices that is supported. Furthermore, it is interesting to notice the relation between the number of threads and the number of devices: there is a direct proportionality between memory and number of threads, since the thread structures are allocated in memory[32].

Figure 8b shows the normalized packet rate values based on the power consumption of the processors (described in Table II). It is noticeable that the low-power processors (i.e. *rpi3-amd** and *up-amd64*) have a better normalized packet rate compared to the ones running in the data center.

TABLE III: Scale-up capabilities

Platform	Threads	Max. Devices	Rate [$10^3 pkts/s$]	Throughput* [$Mbps$]
rpi-arm	13 655	168	16.8	5.4
rpi-arm64	16 638	221	22.1	7.1
c1m1-amd64	7 685	251	25.1	8.0
c2m2-amd64	15 748	804	80.4	25.7
up-amd64	61 502	2 162	216.2	69.2
c4m8-amd64	63 614	3 168	316.8	101.4

*equivalent throughput given the maximum number of devices

4) Migration of processor units: This analysis shows the delay and packet loss introduced by the migration of the processor units from one worker to another. To approximate the delay at the application layer, the device generator sets a timestamp when the packets are built (marked as t_0), and then the processor unit sets a timestamp when the transformation function is executed (marked as t_1). The delay is defined as:

$$t_1 - t_0 = t_{sink_{generator}} + t_{broker} + t_{network} + t_{source_{processor}} + t_{map}$$

During this experiment, one processor unit migrates from one worker to another every minute, while one monitoring device with the same characteristics as the one used for scalability sends data every 100ms. Figure 9 depicts a migration covering workers c1m1, c2m2 and c4m8 respectively. The packet loss and delay results are based on a moving window of 100 samples, and the output is smoothed by re-sampling the data every second. The result shows a small impact on the packet loss of $\leq 1\%$, which corresponds approximately to the loss of one packet.

[29]all workers are synchronized using Network Time Protocol (NTP)

[30]transform: `{x: Map[String, Any] => {x + ("t1" -> System.currentTimeMillis, "r1" -> "cXmY", "n1" -> x("n0"))}}`

[31]The MQTT brokers are used on a round-robin basis for load-balancing.

[32]From `kernel/fork.c`: maxthreads = mempages / (8 * THREADSIZE / PAGESIZE)

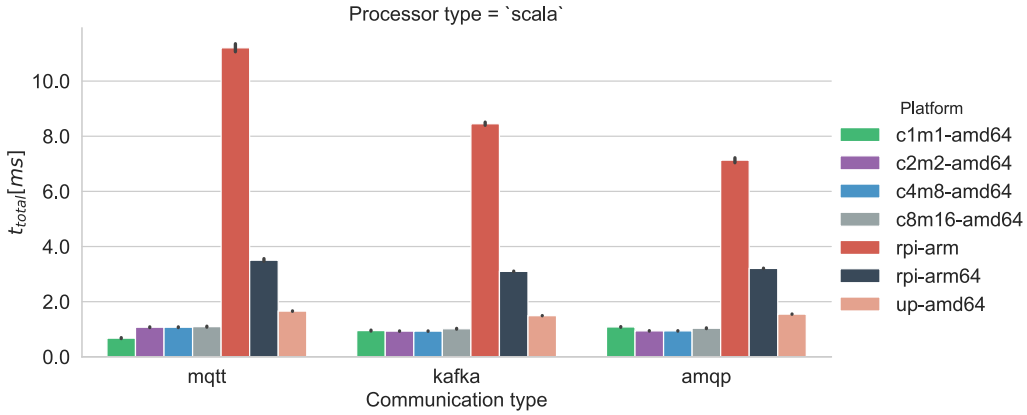
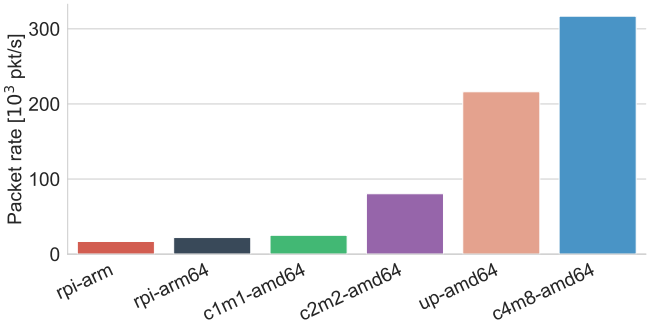
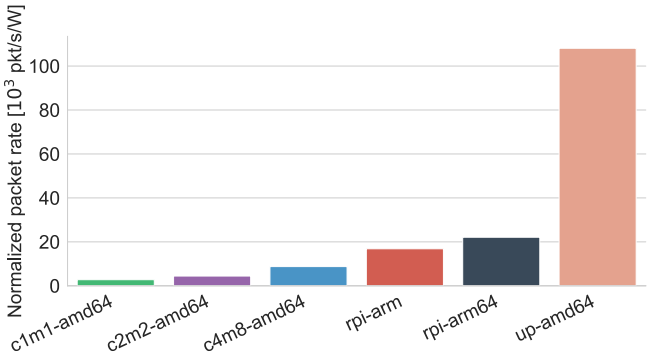


Fig. 7: Comparison of the t_{total} for multiple platforms



(a) Packet rate $[10^3 pkt/s]$



(b) Normalized packet rates with processor consumption $[10^3 pkt/s/W]$

Fig. 8: Packet rates supported by different platforms and architectures

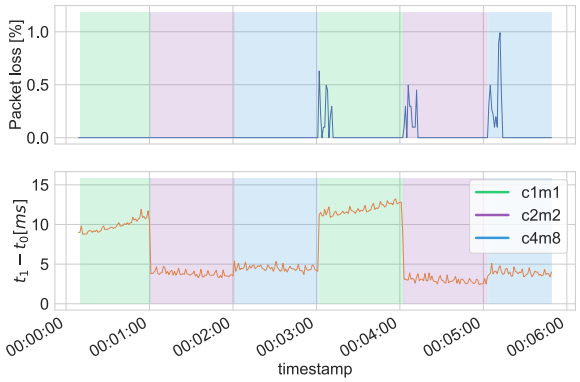


Fig. 9: Packet loss and delay during migration of a processor unit between workers

V. SUMMARY AND FUTURE WORK

This paper describes the architecture of ERAIA, whose implementation has been successfully employed in different use-cases [26] and scenarios [27]. ERAIA exposes a simple API that allows composition of data management pipelines with support for real-time information and intelligence extraction. It offers the possibility to distribute computation based on specific IIoT application requirements such as power consumption, bandwidth, latency, availability of computational power or dedicated hardware, availability of data source, security and privacy of data, etc. The solution integrates open source components and expands their functionality by enabling execution within actors. In the future, we will work on a number of features and improvements leading to the productification of ERAIA. As a brief mention:

Life cycle management of ML models: It covers a distributed management of the models within the framework to allow the processors running the models to be stopped, started, updated and migrated to other workers.

Robustness features: It covers the improvements of the control plane, in terms of the distribution and redundancy of the manager components, the persistency of the worker deployments to support advanced failures, and the interconnections among workers through improved cluster management capabilities.

State replication improvement: It covers the improvement of state management for the processors, enabling advanced data pipeline composition and advanced migration capabilities

between the different actors, beyond the local environment.

Processor scheduling capabilities: It aims to explore how the processor units could self-optimize their placement within workers to improve certain Key Performance Indicators (KPIs), such as context-aware data and tasks scheduling solutions [28].

VI. CONCLUSION

In this paper we introduce ERAIA, a new framework that supports scaling, handles heterogeneity, and provides intelligence and data composition. The proposed solution targets two of the main challenges of Industrial IoT (IIoT) applications, namely the orchestration of data and computational resources and the refining and preparing of data for intelligence extraction. We present how ERAIA, our actor-based framework enabling intelligence and data pipelines, addresses these challenges, together with a detailed architecture description, an implementation and a performance evaluation, proving its capabilities on a series of hardware platforms covering the IoT ecosystem.

We identify future directions and functionality to better serve the IIoT application ecosystem. Although focused on the IIoT, our solution could easily be integrated into other IoT platforms to provide extended features for the Application Enablement Platform (AEP) in a distributed manner.

REFERENCES

[1] Machnation, "Worldwide Embedded and Intelligent Systems Forecast, 20182022: Data Transformation and the Journey of Data Across the Internet Landscape from the Physical to the Digital," 2018. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=US43690318

[2] R. A. Martin and A. Soellinger, "The emerging iic verticals taxonomy landscape," 2016. [Online]. Available: https://iiconsortium.org/news/joi-articles/2016-June-The-Emerging-IIC-Verticals-Taxonomy-Landscape.pdf

[3] A. Shukla, S. Chaturvedi, and Y. Simmhan, "RIoTBench: An IoT benchmark for distributed stream processing systems," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 21, p. e4257, 2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4257

[4] J. Höller, V. Tsiatsis, and C. Mulligan, "Toward a Machine Intelligence Layer for Diverse Industrial IoT Use Cases," *IEEE Intelligent Systems*, vol. 32, no. 4, pp. 64–71, 2017.

[5] S. Sridhar and M. E. Tolentino, "Evaluating the impact of pushing voice-driven interaction pipelines to the edge," in *Proceedings of the 15th ACM International Conference on Computing Frontiers*, ser. CF '18. New York, NY, USA: ACM, 2018, pp. 45–53. [Online]. Available: http://doi.acm.org/10.1145/3203217.3203242

[6] V. Tudor, V. Gulisano, M. Almgren, and M. Papatriantafilou, "BES: Differentially private event aggregation for large-scale iot-based systems," *Future Generation Computer Systems*, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17322823

[7] P. Simoens, M. Dragone, and A. Saffiotti, "The internet of robotic things: A review of concept, added value and applications," *International Journal of Advanced Robotic Systems*, vol. 15, no. 1, p. 1729881418759424, 2018. [Online]. Available: https://doi.org/10.1177/1729881418759424

[8] P. Mendki, "Docker container based analytics at iot edge video analytics usecase," in *2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU)*, Feb 2018, pp. 1–4.

[9] R. Morabito and N. Beijar, "Enabling data processing at the network edge through lightweight virtualization technologies," in *2016 IEEE International Conference on Sensing, Communication and Networking (SECON Workshops)*, June 2016, pp. 1–6.

[10] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan, "Osmotic computing: A new paradigm for edge/cloud integration," *IEEE Cloud Computing*, vol. 3, no. 6, pp. 76–83, Nov 2016.

[11] V. Cozzolino, A. Y. Ding, and J. Ott, "Fades: Fine-grained edge offloading with unikernels," in *Proceedings of the Workshop on Hot Topics in Container Networking and Networked Systems*, ser. HotConNet '17. New York, NY, USA: ACM, 2017, pp. 36–41. [Online]. Available: http://doi.acm.org/10.1145/3094405.3094412

[12] C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi, "Cloud container technologies: a state-of-the-art review," *IEEE Transactions on Cloud Computing*, vol. PP, pp. 1–1, 05 2017.

[13] R. Jain and S. Tata, "Cloud to edge: Distributed deployment of process-aware iot applications," in *2017 IEEE International Conference on Edge Computing (EDGE)*, June 2017, pp. 182–189.

[14] N. K. Giang, R. Lea, M. Blackstock, and V. C. M. Leung, "Fog at the edge: Experiences building an edge computing platform," in *2018 IEEE International Conference on Edge Computing (EDGE)*, July 2018, pp. 9–16.

[15] P. Persson and O. Angelsmark, "Calvin–merging cloud and iot," *Procedia Computer Science*, vol. 52, pp. 210–217, 2015.

[16] R. Young, S. Fallon, and P. Jacob, "An architecture for intelligent data processing on iot edge devices," in *2017 UKSim-AMSS 19th International Conference on Computer Modelling Simulation (UKSim)*, April 2017, pp. 227–232.

[17] Y. Nakamura, H. Suwa, Y. Arakawa, H. Yamaguchi, and K. Yasumoto, "Design and implementation of middleware for iot devices toward real-time flow processing," in *2016 IEEE 36th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, June 2016, pp. 162–167.

[18] B. Cheng, A. Papageorgiou, and M. Bauer, "Geelytics: Enabling on-demand edge analytics over scoped data sources," in *2016 IEEE International Congress on Big Data (BigData Congress)*, June 2016, pp. 101–108.

[19] H. Najdataei, M. Subramaniyan, V. Gulisano, A. Skoogh, and M. Papatriantafilou, "Adaptive Stream-based Shifting Bottleneck Detection in IoT-based Computing Architectures," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2019, pp. 993–1000.

[20] M. D. de Assuno, A. da Silva Veith, and R. Buyya, "Distributed data stream processing and edge computing: A survey on resource elasticity and future directions," *Journal of Network and Computer Applications*, vol. 103, pp. 1 – 17, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804517303971

[21] Machnation, "2019 IoT Edge Score Card," 2019. [Online]. Available: https://www.machnation.com/2019/02/12/machnation-rates-17-iot-vendors-2019-iot-edge-scorecard

[22] ——, "2019 IoT Application Enablement ScoreCard," 2019. [Online]. Available: https://www.machnation.com/2019/01/23/2019-application-enablement-aep-scorecard-rates-20-vendors/

[23] IDC, "Extending the IoT Platform to the Edge," 2018. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=US43303518

[24] R. Morabito, "A performance evaluation of container technologies on internet of things devices," in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2016, pp. 999–1000.

[25] R. Morabito, J. Kjllman, and M. Komu, "Hypervisors vs. lightweight virtualization: A performance comparison," in *2015 IEEE International Conference on Cloud Engineering*, March 2015, pp. 386–393.

[26] S. S. Arumugam, V. Umashankar, N. C. Narendra, R. Badrinath, A. P. Mujumdar, J. Höller, and A. Hernandez, "IoT enabled smart logistics using smart contracts," in *2018 8th International Conference on Logistics, Informatics and Service Sciences (LISS)*, Aug 2018, pp. 1–6.

[27] S. S. Arumugam, R. Badrinath, A. H. Herranz, J. Höller, C. R. B. Azevedo, B. Xiao, and V. Tudor, "Accelerating industrial iot application deployment through reusable ai components," in *2019 Global IoT Summit (GIoTS)*, June 2019, pp. 1–4.

[28] M. Breitbach, D. Schäfer, J. Edinger, and C. Becker, "Context-aware data and task placement in edge computing environments," in *2019 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, March 2019, pp. 1–10.