Design Guidelines for Apache Kafka Driven Data Management and Distribution in Smart Cities

Theofanis P. Raptis*, Claudio Cicconetti*, Manolis Falelakis†, Tassos Kanellos‡, Tomás Pariente Lobo§

*Institute of Informatics and Telematics, National Research Council, Pisa, Italy. Email: {name.surname}@iit.cnr.it
†Netcompany-Intrasoft, Athens, Greece. Email: manolis.falelakis@netcompany-intrasoft.com
‡ITML, Athens, Greece. Email: tkanellos@itml.gr
§Atos Spain, Madrid, Spain. Email: tomas.parientelobo@atos.net

Abstract—Smart city management is going through a remarkable transition, in terms of quality and diversity of services provided to the end-users. The stakeholders that deliver pervasive applications are now able to address fundamental challenges in the big data value chain, from data acquisition, data analysis and processing, data storage and curation, and data visualisation in real scenarios. Industry 4.0 is pushing this trend forward, demanding for servitization of products and data, also for the smart cities sector where humans, sensors and devices are operating in strict collaboration. The data produced by the ubiquitous devices must be processed quickly to allow the implementation of reactive services such as situational awareness, video surveillance and geo-localization, while always ensuring the safety and privacy of involved citizens. This paper proposes a modular architecture to (i) leverage innovative technologies for data acquisition, management and distribution (such as Apache Kafka and Apache NiFi), (ii) develop a multi-layer engineering solution for revealing valuable and hidden societal knowledge in smart cities environment, and (iii) tackle the main issues in tasks involving complex data flows and provide general guidelines to solve them. We derived some guidelines from an experimental setting performed together with leading industrial technical departments to accomplish an efficient system for monitoring and servitization of smart city assets, with a scalable platform that confirms its usefulness in numerous smart city use cases with different needs.

Index Terms—Smart cities, Apache Kafka, Apache NiFi, Data Management, Industry 4.0

I. INTRODUCTION

During the last 10–15 years, there has been an explosion of enabling technologies for the realization of the Internet of Things (IoT), including sensors, actuators, embedded devices with computation capabilities, software platforms, and communication protocols [1], [2]. This phenomenon was driven initially by the huge potential foreseen in the automation and digitization of industrial applications [3] and personal health systems [4], but it benefited many other segments through spillover effects. One of the most important outlets of the growing IoT ecosystem has been the smart city market [5], which was ready to incorporate new technologies to supply citizens, as well as city councils, with new services or more efficient realizations of existing ones.

In the early developments of smart cities, each service relied on its own devices that could operate only with a dedicated proprietary platform in a vertical manner. Typically, the platforms offered Application Programming Interfaces (APIs) for the consolidation of data across multiple services in the cloud, e.g., for integrated user dashboards or big data analysis of historical data. Indeed, many studies have focused on supporting semantic interoperability of data only *after* they have been safely stored in a common repository (currently referred to as "data lake") [6], [7]. However, such a compartmentalized structure had limitations, especially in terms of redundant deployed resources and inefficient management. Therefore, the community has moved towards a horizontal approach, where a common platform is able to communicate with all kinds of devices; most often sensors in smart city applications [8].

This evolution is illustrated in Fig. 1, which also shows the high-level architecture of the Data Management Platform (DMP) defined the H2020 MARVEL project, which aspires to define a comprehensive solution for multi-modal real-time analytics applications. Such applications derive from the analysis of the requirements and expectations in several use cases of practical interest and high impact of the quality of life of citizens, and they will be validated in three small-scale field trials across Europe, i.e., in Malta, Serbia, and Italy [9]. In the project, we exploit the recent trend of breaking down the computation elements of the system into three layers in a hierarchy [10], [11]: the *edge* layer is closest to the sensing and embedded computation devices, but it consists of devices with modest capabilities in terms of computation, connectivity, and storage; the *fog* layer has more powerful capabilities and we treat it like a small private cloud, which is however under the control of the end user; and, finally, the *cloud* layer is hosted on public remote data centers, which have virtually infinite capacity but incur a high latency and usage costs.

The H2020 MARVEL project covers all aspects, from the development of new sensors, e.g., directional microphone arrays, to the efficient training of Artificial Intelligence (AI)/Machine Learning (ML) models on devices with limited capabilities, to the ethics of data collection and analysis. However, in this paper we focus only on the DMP, which is the core of the project's software architecture and consists of the following components:

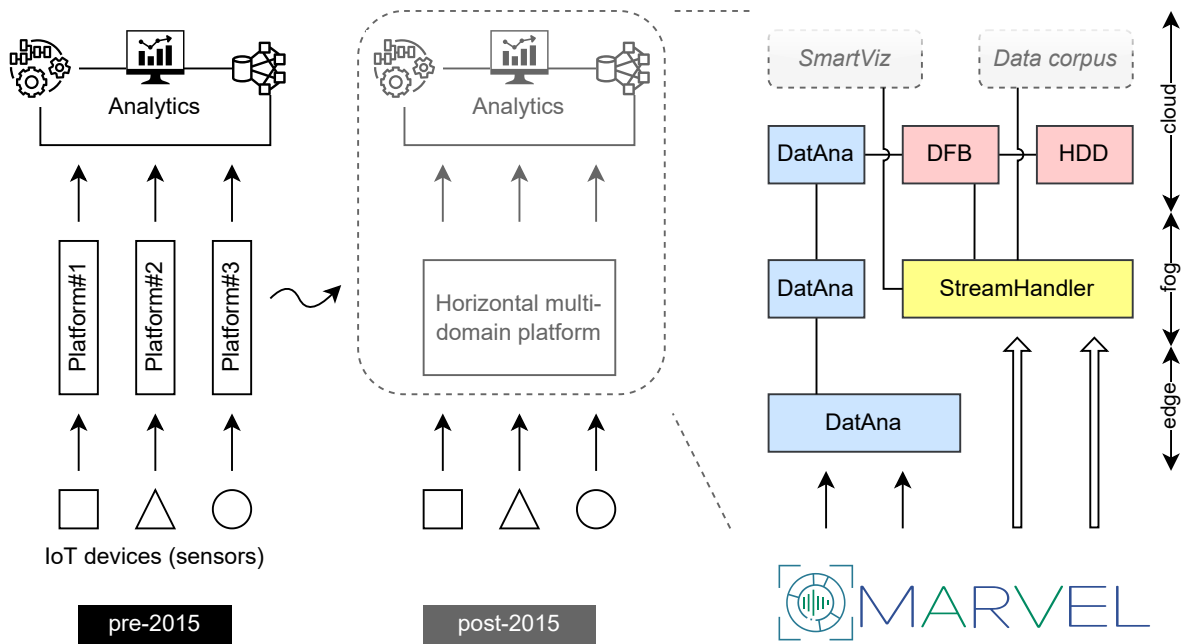– **DatAna**, for the processing and transmission of structured

Fig. 1: Transition of smart city data management and distribution systems from vertical silos to horizontal (cloud-based) platforms and positioning of the H2020 MARVEL project's proposition for multi-layer Edge to Fog to Cloud (E2F2C) environments.

data produced by AI components to all the layers in the inference pipeline;

– **DFB**, which is in charge of managing heterogeneous data across multiple components in the cloud;
– **StreamHandler**, which processes, stores and delivers real-time AV data at the fog layer toward the processing servers and the Data corpus;
– **HDD**, which can optimize the data management based on the available resources and current workload;

Those core components work in close connection to the **Data corpus**, which is the repository of the data collected from the sensors, mainly consisting of microphones and cameras, for visualization and augmentation, and the **SmartViz**, which provides the Human Machine Interface (HMI) for visualization and analysis.

The rest of the paper is structured as follows. First, we introduce the background and foundations of the H2020 MARVEL project, which are needed to understand the concepts and terminology used in the rest of the paper, in Sec. II. This section also includes an overview of the essential state of the art on data/resource management in edge/fog systems. In Sec. III we then illustrate all the main components of the H2020 MARVEL project's DMP. Handling Audio-Video (AV) data was found to be challenging in particular, hence it is discussed separately in Sec. IV. Preliminary results obtained during the mid-project integration tests are reported in Sec. V, while Sec. VI presents some useful design guidelines which directly reflect our experience.
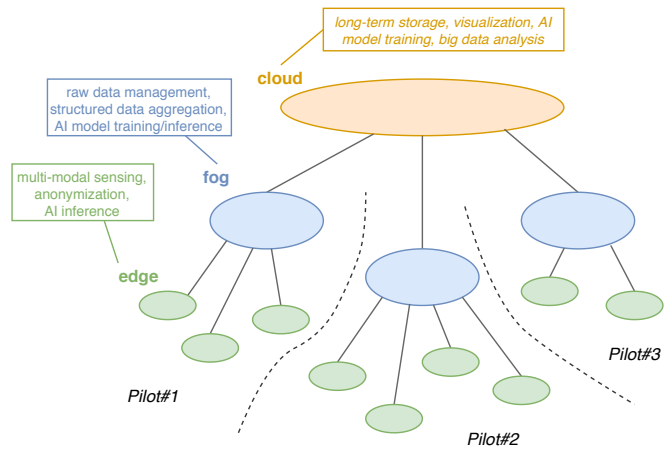

Fig. 2: E2F2C view of the H2020 MARVEL project's pilots.

II. BACKGROUND

In this section we provide an overview of the aspects of the H2020 MARVEL project that are relevant to the design and implementation of the DMP, which is the main subject of this work.

A layered view of the physical deployment of the project's pilots is illustrated in Fig. 2. The lowest layer is the *edge*, which contains the sensors and embedded devices to perform on-site operations. We have two types of sensors: microphone arrays, producing audio streams, and cameras, producing AV streams. All the sensors are expected to operate continu-

ously during the service lifetime. The embedded devices, i.e., Raspberry Pis and NVIDIA Jetson boards, perform operations directly at the edge, which include anonymization of the AV streams and basic inference operations. The intermediate layer is called the *fog*[1], which includes more powerful computation resources that are shared by multiple edge sites, e.g., workstations and rack-mountable servers with Graphics Processing Units (GPUs), which are are suitable for not only managing the different data streams but also performing more advanced inference tasks, as well as training of AI models. Finally, the *cloud* is hosted in an infrastructure provided by a project partner and it is common for all the pilots. The cloud provides long-term storage of data and all the services for HMI, i.e., visualization and real-time (on-demand) analysis.

In Fig. 2 we distinguish between raw data vs. structured data:

– The **raw data** are the AV streams generated by the sensors, irrespective of whether they have been anonymized or not. Their formats and characteristics are heterogeneous because they depend on the physical devices installed (e.g., may use different codec or sample AV at different rates). In any case, the throughput is generally high, especially for video, which requires carefully provisioned bandwidth, long-term storage, and configuration of the data distribution services.

– The **structured data**, instead, are data generated by the analytics applications, i.e., the inference components. In the project, have different data models for each AI component, but all of them have been based on the fully Smart-Data-Models (SDM)-compliant[2] data models that DatAna is producing. Furthermore, their throughput is generally much lower than than of raw data.

A schematic of a typical multi-modal application in the H2020 MARVEL project is illustrated in Fig. 3. We have used "thick" arrows to represent the raw data flows, while "thin" arrows refer to structured data flows, to reflect in a pictorial manner their different bandwidth requirements. The diagram is intended to provide the reader with a sketch of the components that are involved in the deployment of a service, which can differ in practice depending on the specific use case and pilot. The services we support in the project include surveillance applications (e.g., detecting anomalies in public spaces), emotion recognition (with aerial images taken by drones), road safety (e.g., monitoring junctions with mixed cars and bike traffic).

A. Related work

The topic of efficient allocation of resources in E2F2C has been extensively studied in the scientific literature (e.g., [12]). The vast majority of the works propose highly simplified mathematical models, which are then solved to maximize a given objective function, but they do not linger on the practical implementation of their solution using industry-grade tools. On the other hand, in the H2020 MARVEL project our goal was to realize a solution based on widely-used, reliable, and open source software. Furthermore, some works have focused on the specific aspect of efficient data distribution. In [13], the authors have proposed Peer-to-Peer (P2P) as a means to distribute data in a robust and decentralized manner among agents at the edge. Such an approach only partially covers the needs of the H2020 MARVEL project, since AV real-time streams cannot be stored efficiently in a P2P overlay. An in-network storage management is put forward in [14], to place both raw and structured data on distributed resources using the Google File System, which is however proprietary. Another perspective is taken in [15], which builds on top of the distributed database Apache Cassandra by adding geolocalization tags that help distributing the load across edge/fog nodes; we have not considered this feature since, at least in the project pilots, the edge/fog nodes are all deployed in close proximity. Finally, in [16], the authors distribute the execution of functions following a Function as a Service (FaaS) approach, which is adopted in production only in cloud systems, but has shown some potential also for edge computing [17], even though further research may be needed. The solutions mentioned can be gradually incorporated in the DMP at a later stage of the project, which is still ongoing with expected completion date at the end of 2023.

More specifically, Apache Kafka has also been widely used in the streaming applications domain. In [18], the authors replicate Apache Kafka logs for various distributed data-driven systems at LinkedIn, including source-of-truth data storage and stream processing. In [19], the authors design a distributed cluster processing model based on Apache Kafka data queues, to optimize the inbound efficiency of seismic waveform data. In [20], the authors extend Apache Kafka by building an in-memory distributed complex event recognition engine built on top of Apache Kafka streams. In [21], the authors design a simulation platform enabling evaluations of future mobility scenarios, based on an Apache Kafka architecture. In [22], the authors break the streaming pipeline into two distinct phases and evaluate percentile latencies for two different networks, namely 40GbE and InfiniBand EDR (100Gbps), to determine if a typical streaming application is network intensive enough to benefit from a faster interconnect. Moreover, they explore whether the volume of input data stream has any effect on the latency characteristics of the streaming pipeline, and if so, how does it compare for different stages in the streaming pipeline and different network interconnections. In [23], the authors propose a distributed framework for the application of stream processing on heterogeneous environmental data, which addresses the challenges of data heterogeneity from heterogeneous systems and offers real-time processing of huge environmental datasets through a publish/subscribe method via a unified data pipeline with the application of Apache Kafka for real-time analytics. In [24], the authors find that filtering on large datasets is best done in a common up-stream point instead of being pushed to, and repeated, in downstream components. To demonstrate the advantages of such an approach, they modify Apache Kafka to perform

---

[1]We not that in the scientific literature and in the market press the terms "edge" and "fog" do not have universally accepted meanings. Sometimes they are even used interchangeably.
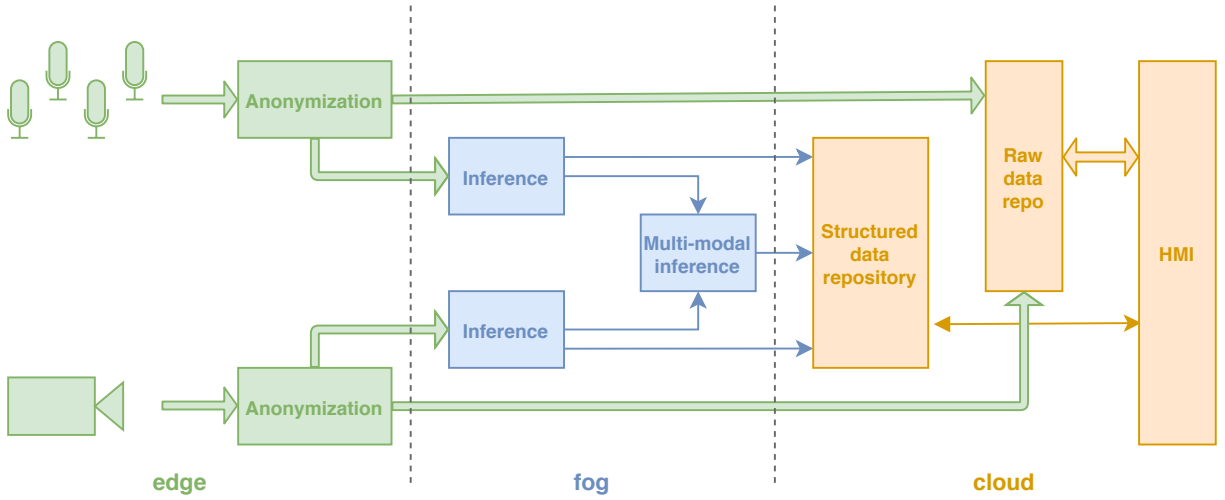
[2]https://smartdatamodels.org/

Fig. 3: Schematic of a typical multi-modal, i.e., audio + video, real-time analytics application in H2020 MARVEL project. The mapping of components to the edge/fog/cloud layers is only indicative: in real applications this may depends on physical, environmental, and administrative constraints.

limited native data transformation and filtering, relieving the downstream Spark application from doing this. Their approach outperforms four prevalent analytics pipeline architectures with negligible overhead compared to standard Kafka. In the next sections, we illustrate the high-level design and current status of development of H2020 MARVEL project's DMP.

III. DMP SOFTWARE ARCHITECTURE DESIGN

The main components of the DMP are introduced briefly in Sec. I (see also Fig. 1). Below we provided an overview, followed by component details in dedicated sub-sections.

DatAna is responsible for collecting the inference results from all AI components from all layers through its instances residing at each layer, transforming them into SDM-compliant counterparts and then transferring them to higher layers in the E2F2C continuum. The DatAna cloud layer thus aggregates all transformed inference results and relays them to the DFB, which also resides at the cloud. **DFB** persistently stores all SDM-compliant inference results it receives, but also makes them available in real time to SmartViz and Data Corpus. DFB also exposes a REST API to SmartViz to allow it to access all archived inference results in its ElasticSearch database. In addition, the DFB receives user-generated inference result verification messages from SmartViz and use this information to update the corresponding inference results stored in its database. **HDD** interacts exclusively with the DFB to receive information on current Kafka topic partitioning and associated performance metrics and to send updated, optimised Kafka topic partitioning recommendations. In parallel, **StreamHandler** receives information on active AV sources after requesting it from a component called *AV Registry* via a REST call, and it uses that information to connect to all active AV sources and receive their AV data streams to segment them and store them persistently. StreamHandler also exposes a REST API that is accessed by SmartViz to request archived AV data

from specific sources and points in time. The Data Corpus resides at the cloud and is subscribed to all DFB Kafka topics where DatAna publishes SDM-compliant inference results to receive them in real time and archive them internally to make them available for further AI training purposes along with the associated AV data it collects from StreamHandler. The Data Corpus is also subscribed to the DFB Kafka topic that is used by SmartViz to publish user-generated inference result verifications to receive them in real time and update the corresponding archived inference results accordingly. The Data Corpus is also connected to StreamHandler, from which it receives AV data as binary files that are a result of AV stream segmentation.

A. DatAna

DatAna is a component distributed across all three E2F2C layers, with a separate instance deployed at each infrastructure node. DatAna is complemented by an MQTT message broker, which is also deployed at each infrastructure node, alongside DatAna. Each instance of the MQTT message broker is responsible for collecting structured data, i.e., inference results, from the AI components residing on the same layer as the respective MQTT instance. Specifically, AI components publish their raw inference results to dedicated MQTT topics in real time as they are being produced through the analysis of the AV data streams they receive. The input inference results of each AI component are formatted as JSON documents according to a dedicated distinct data model that fits the requirements of each AI component. The following are the most notable fields in these data models:

– AV source id. The id of the AV source that produced the stream that was analysed to produce the inference result.
– Inference result id. A unique identifier for the inference result.

– Timestamps. In case the inference result refers to an instant in time a single timestamp is provided. In case the inference result refers to a period in time, two timestamps are provided, corresponding to the start and end of the time period of the result. All time information is absolute and following the ISO 8601 UTC format.

Besides the above, the raw inference results contain other fields that are specific to the needs of each AI component.

Each DatAna instance residing on the same infrastructure node as an MQTT broker subscribes to the broker's topics to receive all incoming input AI inference results. Subsequently, DatAna transforms the input inference results into SDM-compliant counterparts. Three data models that belong in the collection of smart data models of the SDM standard have been identified to be relevant to MARVEL, which have been modified by adding additional fields to account for the project's needs:

– **MediaEvent**: to describe general AI inference results.
– **Alert**: to describe AI inference results that should be perceived as alerts.
– **Anomaly**: to describe AI inference results that should be perceived as detected anomalies.

DatAna selects autonomously the most appropriate data model to perform the transformation, whose output is then relayed to higher-level layers.

Specifically, the SDM-compliant inference results produced by DatAna at the edge layer are relayed to DatAna at the fog layer and the SDM-compliant inference results produced by DatAna at the fog layer are relayed to DatAna at the cloud layer. The DatAna instance at the cloud layer is responsible for relaying the SDM-compliant inference results it collects from all layers to the DFB by publishing them to the appropriate DFB Kafka topics.

B. DFB

The DFB resides at the cloud and receives all SDM-compliant inference results published by the DatAna cloud instance and stores them persistently in its ElasticSearch database. The DFB also exposes a REST API to SmartViz to allow it to access all archived inference results in its ElasticSearch database. The DFB receives user-gerenated verifications of inference results from SmartViz when they are published to a dedicated DFB Kafka topic and uses them to update the respective archived inference result entries accordingly. The DFB also accesses a REST API at the HDD for dispatching the currently applied Kafka topic partition information along with associated performance measurements to it. Using the same REST API, the DFB can also receive updated Kafka topic partition allocation that is recommended by the HDD. SmartViz is subscribed to all DFB Kafka topics where DatAna publishes SDM-compliant inference results to receive them in real time and present them to the user. SmartViz also allows users to verify the inference results they are presented with. SmartViz transmits these user-generated verifications to the DFB by publishing them to a dedicated Kafka topic available at the DFB.

C. StreamHandler

StreamHandler resides at the fog and receives AV data streams from all active AV sources (CCTV cameras, network-enabled microphones, AudioAnony and VideoAnony instances) via RTSP. During initialisation, StreamHandler accesses the REST API of the AVRegistry to discover the active AV sources and their details. During operation, StreamHandler consumes the AV RTSP streams and segments them according to a pre-specified time intervals to generate binary documents, suitable for persistent storage. StreamHandler archives the generated AV data files and also exposes a REST API to accept requests from SmartViz about the transmission of AV data from specific AV sources (reference to AV Source id) and from specific points in time. Upon such requests, StreamHandler retrieves the necessary binary files, compiles a unified/edited version of the stream that corresponds to the timeframe requested and generates a link to the said binary file which is to be consumed by SmartViz.

D. HDD

The HDD exposes a REST API to allow the reception of the currently applied DFB Kafka topic partition information along with associated performance measurements from the DFB. The HDD uses this information as input to calculate an optimised Kafka topic partition allocation and subsequently makes it available to the DFB via its REST API. The exact optimisation method that is implemented by the HDD can be found in [25].

The DMP has been applied in 5 use cases defined for the needs of the initial version of the MARVEL Integrated framework.

IV. AV DATA

In the context of the MARVEL framework design activities, certain similarities and overlaps were identified between the functionalities of StreamHandler and those of DFB and DatAna with regards to big data management. However, following an in-depth analysis of the MARVEL framework requirements that the DMP should satisfy, a gap was identified that could not be covered by the DFB and DatAna solutions. This gap was related to the management of audio-visual data. More specifically, the following requirements were established:

– Receive and efficiently archive live streams of audiovisual binary data from all relevant MARVEL sensors, devices and components during system operation.
– The persistent storage of archived AV data should comply with high data security standards and data privacy requirements.
– Provide access to archived audiovisual binary data to the MARVEL UI (SmartViz) by streaming requested archived audiovisual data upon demand in order to present them to the end-user and in association with relevant inference results produced by MARVEL AI components.
– Support the expansion of the data set of the Data Corpus by relaying selected archived audiovisual data to it.

StreamHandler was found to be in a position to be able to satisfy these requirements and fill the gap by extending its supported data source types, its connectors and data storage capabilities. This course of action was aligned with INTRA's strategic plan to expand the StreamHandler platform in the direction of audiovisual data management for increased inter-operability in order to address additional business cases and reinforce its position in the big data management and smart cities domains.

V. EVALUATION OF THE SETUP

In this section we summarize the results obtained during the first system integration tests of H2020 MARVEL project.

A. DatAna

During the tests performed during the MVP, the performance metrics of a single NiFi instance was measured. Table I summarises the collected measurements for the specified metrics.

TABLE I: Results of the measurements for DatAna component

Metric	Value
Data loss rate	0
Service availability-failed request	100% availability
Data access restriction	None
Data throughput	1.1 MB/s
Response time	47.1 ms
Number of cluster nodes	1

For a more in-depth analysis of performance metrics, there is this Cloudera study [26], which reports how NiFi behaves in terms of scalability and performance (data rates) using very demanding workloads.

B. DFB

For DFB the following high-level performance indicators were considered:

– **Data Integrity**: to confirm that advanced encryption mechanisms over end-to-end data transfer will guarantee data integrity. Metric: Data loss rate.
– **Scalability**: to increase the number of modality data streams and verify that performance metrics improve or at least stay the same. Metric: HW speed up.
– **Availability**: to verify that DFB resources are available and discoverable. Metrics: Service availability-failed request, data access restriction.
– **Performance** (for high volume, heterogeneous data streams): to measure different performance metrics under different execution conditions. Metrics: Data transfer latency, data throughput, response time, number of cluster nodes.

Table II summarises the collected measurements for the specified metrics.

TABLE II: Results of the measurements for DFB component

Metric	Value
Data loss rate	0
HW speed up	-
Service availability-failed request	100% availability
Data access restriction	None
Data transfer latency	5 ms (200 MB/s load)
Data throughput	605 MB/s
Response time	5 ms (200 MB/s load)
Number of cluster nodes	3

C. StreamHandler

Preliminary testing has indicated that StreamHandler is capable of processing at least 3 Full HD AV data streams in parallel with no performance lag when deployed on an infrastructure with 2 CPU cores allocated.

D. HDD

For the purposes of evaluating the efficiency of HDD, we took into account the industrial best practices in the related application sectors. We identified Kafka setup guidelines used by credible industrial service providers. For example, Microsoft, recommends that it would be better to constrain the existing partitions per broker (including replicas) to a number not more than 1000. In another example, Confluent recommends to set the number of partitions per broker to at least $100 \cdot B$. Consequently, combining the essence of these configuration recommendations, we arrive at the following benchmark method, called MS-CNFL: $P = \min\left(P \in_R [1...\frac{1.000 \cdot B}{r}], P \in_R [1...100 \cdot B]\right)$ and $b \in_R [1...B]$, where $\in_R$ denotes uniformly random selection. We measure the system throughput, captured by the ultimate number of partitions selected by each algorithm (our algorithms being BroMin and BroMax of [25]), the replication latency, captured the amount of time that is needed to process each message, in the sense of time required for data to be stored or retrieved, the numbers or costs of the application's infrastructure, captured by the number of brokers used in the Apache Kafka cluster, the OS load metric via the open file handles and the unavailability metric via the unavailability time. We perform the measurements for variable number of consumers. Indicatively, we display the performance in terms of throughput (number of partitions) and replication latency, in Fig. 4. We can see that HDD maintains equivalent numbers of partitions (and therefore throughput), but, at the same time, does not violate the latency constraint (like the benchmark is doing).

VI. DESIGN GUIDELINES

In this Section, we report some useful guidelines which reflect the experiences that we had when building our platform.

- Distil the data exchange requirements of the involved components to consolidate the necessary I/O interfaces as much as possible and consequently reduce integration complexity.
- Decouple as much as possible the direct data exchange between pairs of individual component instances to reduce integration complexity, i.e., avoid the use of REST
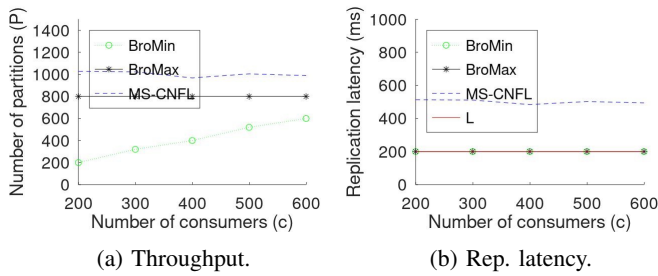
(a) Throughput.   (b) Rep. latency.

Fig. 4: Results of the simulated measurements for HDD component.

APIs wherever possible and promote the use of pub/sub distributed messaging systems.

- Implement open, industry-standard approaches for increased interoperability, scalability and expandability.
- Align the data models used for handling and storing the inference results with the SDM standard in order to improve the visibility and acceptance of the envisioned results.
- Achieve a versatile, yet consistent and coherent solution that can support a multitude of different use cases and scenarios and operate on different infrastructure configurations. In our case, this is achieved through the design of the DMP and the specification of an adaptive reference "AI Inference Pipeline" architecture. The DMP is fully scalable and interoperable as it can be adapted to incorporate virtually any number of edge and fog nodes, while it can handle data emerging from any MARVEL component (e.g., anonymisation components, AI components) at any layer of the E2F2C continuum.
- Handle multimodal raw (AV) and structured (inference results) data by collecting from and distribuing among multiple endpoints both in real time and asynchronusly via persistent storage mechanisms.
- Maintain an up-to-date comprehensive documentation of the specifications for all implemented I/O interfaces and data models using version control. In our case, a GitLab repository was used for this purpose.

REFERENCES

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[2] J. Lin, W. Yu *et al.*, "A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1125–1142, 2017.

[3] L. D. Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[4] S. M. R. Islam, D. Kwak, M. H. Kabir, M. Hossain, and K.-S. Kwak, "The internet of things for health care: A comprehensive survey," *IEEE Access*, vol. 3, pp. 678–708, 2015.

[5] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22–32, 2014.

[6] C. Perera, A. Zaslavsky *et al.*, "Sensor search techniques for sensing as a service architecture for the internet of things," *IEEE Sensors Journal*, vol. 14, no. 2, pp. 406–420, 2014.

[7] P. Desai, A. Sheth, and P. Anantharam, "Semantic gateway as a service architecture for iot interoperability," in *2015 IEEE International Conference on Mobile Services*, 2015, pp. 313–319.

[8] T. P. Raptis, A. Passarella, and M. Conti, "Distributed data access in industrial edge networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 5, pp. 915–927, 2020.

[9] D. Bajovic, A. Bakhtiarnia *et al.*, "Marvel: Multimodal extreme scale data analytics for smart cities environments," in *2021 International Balkan Conference on Communications and Networking (BalkanCom)*, 2021, pp. 143–147.

[10] E. Ahmed, A. Ahmed *et al.*, "Bringing computation closer toward the user network: Is edge computing the solution?" *IEEE Communications Magazine*, vol. 55, pp. 138–144, 2017. [Online]. Available: http://ieeexplore.ieee.org/document/8114564/

[11] P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttila, and T. Taleb, "Survey on multi-access edge computing for internet of things realization," *IEEE Communications Surveys & Tutorials*, vol. PP, p. 1, 2018. [Online]. Available: http://arxiv.org/abs/1805.06695

[12] P. Wang, Z. Zheng, B. Di, and L. Song, "Hetmec: Latency-optimal task assignment and resource allocation for heterogeneous multi-layer mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 18, pp. 1–1, 2019.

[13] S. Becker, F. Schmidt, and O. Kao, "Edgepier: P2p-based container image distribution in edge computing environments." IEEE, 10 2021, pp. 1–8. [Online]. Available: https://ieeexplore.ieee.org/document/9679447/

[14] A. C. Nicolaescu, S. Mastorakis, and I. Psaras, "Store edge networked data (send): A data and performance driven edge storage framework," *Proceedings - IEEE INFOCOM*, vol. 2021-May, 2021.

[15] H. Gupta, Z. Xu, and U. Ramachandran, "Datafog: Towards a holistic data management platform for the iot age at the network edge," *USENIX Workshop on Hot Topics in Edge Computing, HotEdge 2018, co-located with USENIX ATC 2018*, 2018.

[16] C. Cicconetti, M. Conti, and A. Passarella, "A decentralized framework for serverless edge computing in the internet of things," *IEEE Transactions on Network and Service Management*, vol. 18, pp. 2166–2180, 2020.

[17] M. S. Aslanpour, A. N. Toosi *et al.*, "Serverless edge computing: Vision and challenges." ACM, 2 2021, pp. 1–10. [Online]. Available: https://dl.acm.org/doi/10.1145/3437378.3444367

[18] G. Wang, J. Koshy *et al.*, "Building a replicated logging system with apache kafka," *Proc. VLDB Endow.*, vol. 8, no. 12, p. 1654–1655, aug 2015.

[19] X.-C. Chai, Q.-L. Wang *et al.*, "Research on a distributed processing model based on kafka for large-scale seismic waveform data," *IEEE Access*, vol. 8, pp. 39 971–39 981, 2020.

[20] S. Langhi, R. Tommasini, and E. D. Valle, "Extending kafka streams for complex event recognition," in *2020 IEEE International Conference on Big Data (Big Data)*, 2020, pp. 2190–2197.

[21] M. Gütlein and A. Djanatliev, "On-demand simulation of future mobility based on apache kafka," in *Simulation and Modeling Methodologies, Technologies and Applications*, M. S. Obaidat, T. Oren, and F. D. Rango, Eds. Cham: Springer International Publishing, 2022, pp. 18–41.

[22] M. H. Javed, X. Lu, and D. K. D. Panda, "Characterization of big data stream processing pipeline: A case study using flink and kafka," in *Proceedings of the Fourth IEEE/ACM International Conference on Big Data Computing, Applications and Technologies*, ser. BDCAT '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1–10.

[23] A. Akanbi, "Estemd: A distributed processing framework for environmental monitoring based on apache kafka streaming engine," in *2020 the 4th International Conference on Big Data Research (ICBDR'20)*, ser. ICBDR 2020. ACM, 2020, p. 18–25.

[24] E. Falk, V. K. Gurbani, and R. State, "Query-able kafka: An agile data analytics pipeline for mobile wireless networks," *Proc. VLDB Endow.*, vol. 10, no. 12, p. 1646–1657, aug 2017.

[25] T. P. Raptis and A. Passarella, "On efficiently partitioning a topic in apache kafka," in *The 2022 International Conference on Computer, Information and Telecommunication Systems (CITS)*. IEEE, 2022.

[26] M. Payne, "Processing one billion events per second with NiFi," https://blog.cloudera.com/benchmarking-nifi-performance-and-scalability/, 2020, [Online; accessed 21-June-2022].