Scalable and Reliable Data Framework for Sensor-enabled Virtual Power Plant Digital Twin

Amritpal Singh, *Member, IEEE*, Umit Demirbaga, Gagangeet Singh Aujla, *Senior Member, IEEE*, Anish Jindal, *Member, IEEE*, Hongjian Sun, *Senior Member, IEEE*, and Jing Jiang, *Senior Member, IEEE*

***Abstract*— Sensor-enabled distributed energy resources (DERs) provide various advantages, including a lower carbon footprint, yet effective management of millions of DERs is still an issue. Virtual power plants (VPP) can integrate several DERs into a unified operational digital twin to enable real-time monitoring, analysis, and control. VPP may utilize advanced solutions to improve operational efficiency by combining substantial measurement data from DERs. However, effectively managing the quantity and complexity of data flows, whether streaming data or high-impact low-frequency data, is essential in maintaining the performance of DERs at sustained levels. The vast amounts of diverse data generated from various DERs pose significant challenges for storage, processing, and resource management. This paper proposes a comprehensive framework that employs a distributed big data cluster to ensure scalable and reliable data storage and utilizes a robust message broker system for efficient data queuing. Additionally, we present innovative load-balancing strategies within the VPP Digital Twin system. A decision tree algorithm is implemented to calculate the forthcoming workload collected by various deployed sensors at various DERs. The required resources are identified per workload, and the numbers are forwarded to the Orchestrator. The Orchestrator scales up and down resources based on resource utilization suggested by the decision tree algorithm when the resources or nodes are insufficient to handle the sensor data, optimizing the utilization of computing resources. The framework also features a failure detection component that performs root cause analysis to provide actionable insights for system optimization. Experimental results show that this framework ensures high efficiency, reliability, and real-time operational capability in VPP digital twin by addressing critical challenges in data storage, streaming data analysis, and load balancing.**

***Index Terms*— Digital Twin, Sensors, Streaming Data Processing, Load Balancing, Real-Time Analytics, Virtual Power Plant.**

This work was supported by the Engineering and Physical Sciences Research Council [grant number EP/Y005376/1] – VPP-WARD Project, https://www.vppward.com. This work was also supported by the CHEDDAR: Communications Hub for Empowering Distributed ClouD Computing Applications and Research funded by the UK EPSRC under grant numbers EP/Y037421/1 and EP/X040518/1.

A. Singh is with the Department of Computer and Information Sciences, Northumbria University, Newcastle Upon Tyne, United Kingdom (e-mail: amritpal2.singh@northumbria.ac.uk).

U. Demirbaga is with the Department of Computer Engineering, Bartin University, Bartin, Türkiye (e-mail: udemirbaga@bartin.edu.tr).

G.S. Aujla and A. Jindal are with the Department of Computer Science, Durham University, Durham, United Kingdom (e-mail: gagangeet.s.aujla@durham.ac.uk and anish.jindal@durham.ac.uk).

H. Sun are with the Department of Engineering, Durham University, Durham, United Kingdom (e-mail: hongjian.sun@durham.ac.uk).

J. Jiang is with the Department of Mathematics, Physics & Electrical Engineering, Northumbria University, Newcastle, UK (e-mail: Jing.Jiang@Northumbria.ac.uk).

I. INTRODUCTION

Distributed energy resources (DERs), such as dwellings equipped with solar and battery systems, are essential for attaining Net Zero emissions [1], [2]. Despite the numerous advantages of DERs, including a reduced carbon footprint, the optimal management of millions of DERs remains an unresolved issue, particularly concerning enhancing grid resilience against extreme phenomena (e.g., storms and heat waves) [3]. The latest "Holistic Transition" Future Energy Scenario[1] anticipates that annual energy generation from onshore wind will rise from 36 TWh in 2023 to 104 TWh by 2050, while solar energy is projected to expand from 16 TWh to 116 TWh. This transition may present considerable hurdles to power grid operations since managing millions of sensor-enabled DERs with the existing, predominantly centralized architecture will be exceedingly challenging, if not unfeasible [4], [5]. In this context, enabling Virtual Power Plants (VPP), which can consolidate several DERs into a singular operational "virtual" power plant digital twin, can effectively manage multiple DERs in a resilient and scalable manner [6], [7]. VPP digital twins, coupled with sensor-enabled applications, have the potential to reinvent the energy sector by enabling the creation of virtual replicas for physical DERs to facilitate real-time monitoring, analysis, and control [8], [9].

According to Ofgem facts[2], there are more than 1 million homes equipped with solar panels in the UK. By 2050, if each of the approximately 29 million households and businesses in the UK own one DER, it might result in tens of millions of DERs integrated into the UK power grid, generating substantial data volumes. These DERs may witness increased irregularities within the energy network if not properly controlled [10]. Furthermore, DERs can use advanced reactivity, efficiency, and intelligence made possible by sensor-driven digital twins [11]. For this, power grids rely on sophisticated Artificial Intelligence (AI) solutions to exhibit resilience against extreme events and uncertainties that require a substantial quantity of high-quality training data. They should ideally be supplied with data from millions of DERs

[1]https://www.neso.energy/document/321041/download

[2]https://www.ofgem.gov.uk/sites/default/files/docs/2018/12/ofg1050_riio_fast_facts_web.pdf

along with their corresponding distribution networks as well as climatic and environmental data. While digital solutions may enhance operational efficiency by aggregating extensive measurement data from DERs, the processing and storing these vast data volumes provide an escalating challenge. Nonetheless, this task is more challenging than anticipated, as it is costly to execute and because extreme events are infrequent and sudden—commonly referred to as high-impact, low-frequency occurrences, requiring processing systems to adapt quickly within a short turnaround time. Most of the time, traditional data management approaches are inefficient in managing the scale and complexity of data flows and hence become a bottleneck in data management [12].

The massive volume and variety of data generated by DERs require holistic solutions for efficiently handling and storing the data to keep the performance of the DERs at acceptable levels [13], [14]. Moreover, the requirement for real-time data processing worsens matters because any delay or inaccuracy in data analysis can severely impact the decision-making process and operational efficiency of VPP digital twin [15]. It also requires the system to be dynamic in resource management because static resource allocation may cause a dip in performance and escalate expenses. Advanced machine learning and AI techniques have shown good potential in predicting resource demands and optimising task scheduling.For instance, Rosenberger *et al.* [16] demonstrated the effectiveness of AI-based predictive algorithms in managing distributed resource allocation for real-time systems, showcasing their utility in improving decision-making processes. However, their integration into real-time, scalable solutions remains a challenge [17]. All these necessitate a holistic framework that should support scalable and reliable data storage and include effective data queuing, real-time processing, and dynamic auto-scaling mechanisms to make the overall performance and reliability of VPP digital twin environments more effective.

A. Research Problem and Contributions

Consider an example in Fig. 1, where DER data is forwarded to the queuing system with a consistent input and output data rate of X kbps. The DERs act as 'producers', while the processing system is the 'consumer' ingesting the incoming data for storage or processing.

In a **normal case** (as depicted in Fig. 1(a)), the queuing system handles an input rate of X kbps, which is further efficiently ingested by the processing system, ensuring synchronisation between the input and output data rates. However, this scenario can witness two basic abnormal cases, i.e., a) bottleneck at the queuing level and/or b) bottleneck at the processing level. In the **first case**, with an increase in the DER data (maybe due to additional DERs or a change in data generation patterns), let's say the input data rate increases from X kbps to $X + Y$ kbps (as shown in Fig. 1(b)). Now, it may be possible that the queuing channel may be sufficient to handle this sudden change in data rate, however, if the queuing system with one channel has resources to handle an input rate of X kbps, then this will lead to a mismatch between the IN rate ($X + Y$ kbps) and the OUT rate (X kbps). Thus, the queuing system becomes

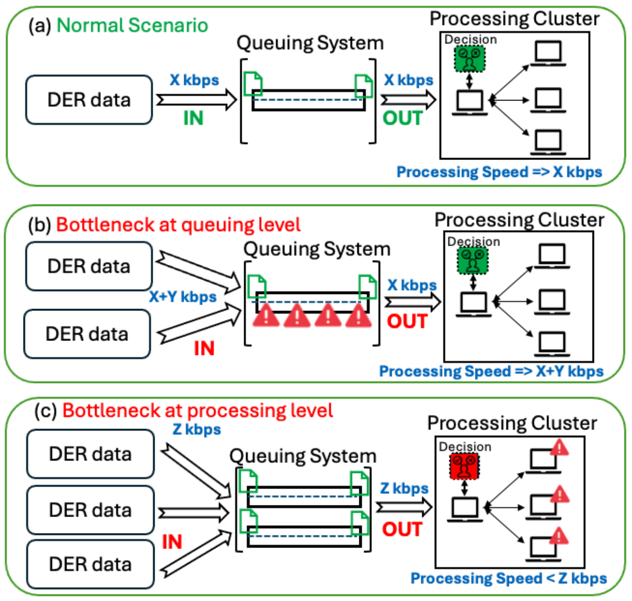


Fig. 1: Problem description; (a) normal scenario, (b) bottleneck at queuing level, and (c) bottleneck at processing level

overwhelmed, leading to delays or possible data loss. In the **second case**, as depicted in Fig. 1(c), let's say an increase in DER data results in the input data rate rising to Z kbps, which matches the queuing system's capacity due to the activation of one more channel. In this case, the queuing system's IN and OUT rates are synchronised. However, the issue arises at the processing end, which cannot ingest the Z kbps rate. This causes a bottleneck within the processing system, where incoming data accumulates faster than it can be processed or stored. Over time, if the processing system does not scale up or the data flow is not regulated, the queue will eventually saturate, causing a secondary bottleneck in the queuing system. Addressing the above bottlenecks is critical for enhancing VPP digital twin's overall performance and scalability. Therefore, this paper investigates the following research questions (RQs):

RQ1: What methodologies can be employed to effectively store and manage the vast amounts and diverse types of data generated by DERs and their corresponding distribution networks, as well as climatic and environmental data? The enormous DER data streams greatly impact the VPP digital twin performance. If high-frequency data is stored and processed inefficiently, this leads to inefficiencies or system failures. The challenge has two different parts. One is to architect solutions that can scale up and out while maintaining high availability for storing and processing various datasets in real-time so that decisions taken within the VPP ecosystem happen promptly.

RQ2: What techniques and frameworks are most effective for analysing the forthcoming workload collected from DER sensors deployed within VPP digital twin? Understanding the accurate forecast of data load is necessary for optimal resource management. While DER sensors generate a continuous data stream, optimising energy consumption and resource allocation is essential. AI models that review historical and real-time data can be deployed to predict future demand behaviour, making VPP operations smooth.

RQ3: Which load balancing strategies can optimally utilise computing resources in a distributed sensor-enabled VPP Digital Twin system, ensuring efficiency and reliability? Effective load balancing is essential in a distributed system such as the VPP to prevent bottlenecks and maximise resource utilisation. Dynamic workloads and poor resource allocation may lead to performance degradation and energy waste. Dynamic workload management strategies, informed by real-time feedback, are crucial to resource scaling and the robustness of the system as a whole.

To address the above-discussed RQs, the key contributions of this paper are outlined below:

- This paper proposes a comprehensive methodology for effectively storing and managing the vast amounts and diverse types of data generated by sensor-enabled DERs within a VPP digital twin. The framework ensures scalable and reliable data storage by deploying a cloud-based big data cluster. Additionally, the system utilises RabbitMQ for efficient data queuing and continuous monitoring to enable dynamic data management and storage optimisation.
- This paper introduces a machine learning approach to analysing the forthcoming workload collected by sensors from various DERs for optimal selection of the required resources. This approach analyses the historical and real-time workload of the various deployed sensors to collect data from various DERs and use the data to identify the futuristic requirements of the resources.
- The paper presents innovative load-balancing strategies within the distributed VPP digital twin system to utilise computing resources fully. The Orchestrator scales resources up and down based on the feedback when nodes cannot handle the sensor-generated data, optimising resource utilisation in the VPP digital twin.

B. Organisation

The rest of the paper is organised as follows. Section II describes the problem formulations, followed by system framework in Section III. Section IV discussed the proposed approach in detail, while the results are illustrated in Section V. The paper is finally concluded in Section VI.

II. Preliminaries and Problem Formulation

Mathematically, the proposed scheme is represented with a graph $\mathcal{G} = (\mathcal{C}, \mathcal{Q}, \mathcal{S})$; where $\mathcal{C}$ is a cluster comprised of storage and processing nodes, $\mathcal{Q}$ is a set of queues to manage the flow of the data, and $\mathcal{S}$ is a set of sensors configured to collect the data from various DERs. Each cluster $\mathcal{C} = (\mathcal{R}^{avl}, \mathcal{R}^{al})$ in the network is characterised by available ($\mathcal{R}^{avl}$) and allocated ($\mathcal{R}^{al}$) resources including memory capacity, CPU processing power, and bandwidth, which collectively influence its ability to handle data traffic, process requests, and manage queues.

The objective function ($\mathcal{OF}$) of the proposed work is segregated into two sub-problems, i.e., $\mathcal{OF} \in \{\mathcal{SP}_1, \mathcal{SP}_2\}$. In sub-problem I ($\mathcal{SP}_1$), we aim to minimize two problem indicators ($\mathcal{PI}_{1.1}, \mathcal{PI}_{1.2}$), i.e., queuing delay ($\mathcal{T}^{queue}$) and data loss ($\mathbb{D}^{loss}$) with an overall goal to have a dynamic queuing system. In sub-problem II, we consider one problem indicator ($\mathcal{PI}_2$) that focuses on improving the performance and reliability of the processing system. The combined objective function ($\mathcal{OF}(\aleph)$) is provided below:

$$\mathcal{OF}(\aleph) = \begin{cases} F_1 \Leftrightarrow \mathcal{SP}_1 : \begin{cases} \mathcal{PI}_{1.1} = \min\ F_{1.1}(\aleph) \\ \mathcal{PI}_{1.2} = \min\ F_{1.2}(\aleph) \end{cases} \\ F_2 \Leftrightarrow \mathcal{SP}_2 : \begin{cases} \mathcal{PI}_{2.1} = \max\ F_2(\aleph) \end{cases} \end{cases} \tag{1}$$

A. Sub-problem 1 ($\mathsf{F}_1(\aleph)$): Queuing Model

The queuing system adopted in the VPP digital twin aims to enhance the network's resource management and performance. The queuing system dynamically monitors the usage of the network resources, guarantees optimal resource allocation, and helps avoid any bottleneck in the network. The two objective functions at the queuing level, including minimal queuing delay and data loss, are described below.

$$F_1(\aleph) = \{F_{1.1}(\aleph), F_{1.2}(\aleph)\} \tag{2}$$

The formal description of these functions is provided below.

a) Queuing Delay ($F_{1.1}(\aleph)$): Let us consider that k^{th} DER sensor collect i^{th} data packet. The generated data is forwarded to the designated queues. Later, the queue data at t time is forwarded to the j^{th} processing cluster. The queues dynamics at time $(t+1)$ focusing i^{th} data packet forwarded to j^{th} processing cluster is given below [18]:

$$\mathcal{Q}_i^j(t+1) = max[\mathcal{Q}_i^j(t) - \mathcal{Q}^j(t), 0] + \zeta_i^j(t) \tag{3}$$

where, ζ_i at time t is the number of i^{th} packets/data forwarded to the j^{th} processing cluster.

Queuing delay occurs when the rate of incoming data (IN rate) surpasses the rate at which data is transmitted (OUT rate). If the IN data rate is synchronised with the OUT data rate (as highlighted in Fig. 1(a)), i.e. X kbps, there is no queuing delay in the underlying network. However, when the arrival rate (IN rate) and transmission rate (OUT rate) are not synchronised (as shown in Fig. 1(b)), a queuing delay occurs in the network. This delay can be formulated as below.

$$\mathcal{T}^{queu} = \frac{\mathcal{L}^{queu}}{\chi^{OUT}} \tag{4}$$

where, $\mathcal{L}^{queu}$ is the length of the queue, and χ^{OUT} is the data output rate from the queue. Here, $\mathcal{L}^{queu}$ is calculated using the following formulation.

$$\mathcal{L}^{queu} = (\chi^{IN} - \chi^{OUT}) \times t \tag{5}$$

where, χ^{OUT} is the data output rate, and χ^{IN} is the data input rate at time t.

Now, in this sub-problem, the objective function ($F_{1.1}(\aleph)$) and constraints ($\mathbb{C}1.1 - \mathbb{C}1.4$) are provided below.

$$F_{1.1}(\aleph) = \min\left(\mathcal{T}^{queu}\right) \tag{6}$$

$$\textit{such that:}$$

$$\mathbb{C}1.1 : \mathcal{L}^{queu} \geq 0$$

$$\mathbb{C}1.2 : \chi^{OUT} > 0$$

$$\mathbb{C}1.3 : \mathcal{L}^{queu} \leq \mathcal{L}^{max}$$

$$\mathbb{C}1.4 : \mathcal{T}^{queu} \geq 0$$

where, $\mathcal{L}^{max}$ is the maximum allowable queue length.

b) Data loss ($F_{1.2}(\aleph)$): As shown in Fig. 1(b), a mismatch in the data IN and OUT rates leads to data accumulation in the queue. Due to resource limitations, the queue/channel can only store a finite amount of data packets ($\mathcal{L}^{max}$). Let us say that $\mathbb{D}^{prod}$ is the input data rate from DERs, and $\mathbb{D}^{cons}$ is the data rate consumed at the processing layer. The total traffic load ($\mathbb{T}^{ld}$) in the channel at time t is calculated as below.

$$\mathbb{T}^{ld}(t) = \int_0^t \mathbb{D}^{prod}(t)dt - \int_0^t \mathbb{D}^{cons}(t)dt \tag{7}$$

The data loss ($\mathbb{D}^{loss}$) is the amount of data at time t, that is not transmitted to the consumer layer, i.e., that data is lost. The $\mathbb{D}^{loss}$ at time t is calculated below.

$$\mathbb{D}^{loss}(t) = \mathbb{T}^{ld}(t) - \mathcal{L}^{max} \tag{8}$$

The overall data loss ($\mathbb{D}_{tot}$) in time interval $[0, T]$ is given as below.

$$\mathbb{D}_{tot} = \int_0^T \mathbb{D}^{loss}(t)dt \tag{9}$$

Accordingly, in this sub-problem, the objective function ($F_{1.2}(\aleph)$) and constraints ($\mathbb{C}1.5 - \mathbb{C}1.9$) are defined as below.

$$F_{1.2}(\aleph) = \min\left(\mathbb{D}_{tot}\right) \tag{10}$$

such that:

$$\begin{aligned}
&\mathbb{C}1.5 : \mathbb{D}^{loss} \geq 0\\
&\mathbb{C}1.6 : t > 0\\
&\mathbb{C}1.7 : \mathbb{D}^{prod} \geq 0\\
&\mathbb{C}1.8 : \mathbb{D}^{cons} \geq 0\\
&\mathbb{C}1.9 : \mathbb{D}^{loss} \leq \mathbb{D}^{max}
\end{aligned}$$

where, $\mathbb{D}^{max}$ is maximum allowable data loss in the network.

B. Sub-problem 2 ($F_2(\aleph)$): Processing Model

Limited computational power can cause the system to be unable to handle and analyse real-time data streams, causing delays in processing and updations in digital twins. The proposed multi-tiered framework aims to solve the problem of scarce processing resources in a VPP digital twin. With this method, computing resources are dynamically allocated in response to real-time demands, ensuring scaling up when resources are limited concerning incoming data from the queues. Additionally, it scales down the resources when the data stream is low, utilizing optimal resources and ensuring reliable and robust data management. The formal description of the objective function ($F_2(\aleph)$) to achieve the above elaborated dynamic resource management is provided below.

a) Objective Function 2.1: Reliability Indicator: A reliability indicator (RI) is designed to measure the proposed scheme's performance regarding resource availability for the end users. In the VPP scenario, i^{th} request is generated for processing data generated from DERs at n processing resources. The resources required to process i^{th} request ($\mathcal{R}_i^{req}$) are presented below.

$$R_i^{req} = \left\{ \begin{matrix} \mathcal{R}_{1,1}^{req} & \mathcal{R}_{1,2}^{req} & \cdots & \mathcal{R}_{1,i}^{req} \\ \mathcal{R}_{2,1}^{req} & \mathcal{R}_{2,2}^{req} & \cdots & \mathcal{R}_{2,i}^{req} \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{R}_{n,1}^{req} & \mathcal{R}_{n,2}^{req} & \cdots & \mathcal{R}_{n,i}^{req} \end{matrix} \right\} \tag{11}$$

Now, the total required resources $\mathcal{R}_{tot}^{req}$ for all generated requests are calculated by using:

$$\mathcal{R}_{tot}^{req} = \left\{ \begin{matrix} \sum_{i=1}^{I} \mathcal{R}_{i,1}^{req} \\ \sum_{i=1}^{I} \mathcal{R}_{i,2}^{req} \\ \vdots \\ \sum_{i=1}^{I} \mathcal{R}_{i,j}^{req} \end{matrix} \right\} \tag{12}$$

In an active resource cluster, the available resources $\mathcal{R}^{avl}$ can be mapped as below.

$$\mathcal{R}^{avl} = \left\{ \begin{matrix} \mathcal{R}_1^{avl} \\ \mathcal{R}_2^{avl} \\ \vdots \\ \mathcal{R}_j^{avl} \end{matrix} \right\} \tag{13}$$

In the above context, if ($\mathcal{R}_{tot}^{req}$) exceeds ($\mathcal{R}_j^{avl}$) at j^{th} processing cluster, there is a degradation in the RI. Considering the factors, the main objective is to maximise RI, thereby improving the robustness and reliability of the overall processing framework. There are two cases to realise this goal, depicted below.

$$RI = \begin{cases} \text{if } \mathcal{R}^{\text{avl}} < \mathcal{R}_{tot}^{\text{req}} & \textit{Identify Stragglers or Scale UP} \\ \text{if } \mathcal{R}^{\text{avl}} \geq \mathcal{R}_{tot}^{\text{req}} & \textit{Check Possible Scale DOWN} \end{cases} \tag{14}$$

In this sub-problem, the objective $F_2(\aleph)$ and constraints ($\mathbb{C}2.1 - \mathbb{C}2.2$) are mentioned below:

$$F_2(\aleph) = \max\left(RI\right) \tag{15}$$

such that:

$$\begin{aligned}
&\mathbb{C}2.1 : \sum_i \mathcal{R}_i^{req} > o\\
&\mathbb{C}2.2 : \sum_j \mathcal{R}_j^{avl} > o\\
&\mathbb{C}2.3 : \sum_{i,j} \mathcal{R}_i^{req} \leq \mathcal{R}_j^{avl}; \forall(j)\\
&\mathbb{C}2.4 : \sum_i \mathcal{R}_i^{req} \leq \mathcal{R}_j^{avl}; \forall(j)
\end{aligned}$$

Combined Objective Function: The objective function $\mathcal{OF}(\aleph)$ is achieved if and only if both $F_1(\aleph)/\mathcal{SP}_1$ and $F_2(\aleph)/\mathcal{SP}_2$ are achieved. The combined objective function is formulated as below.

$$\min \mathcal{OF}(\aleph) = f(F_{1.1}(\aleph), F_{1.2}(\aleph), -F_2(\aleph)) \tag{16}$$

III. System Framework

The VPP digital twin framework comprises of various layers. These layers are shown in Figure 2.

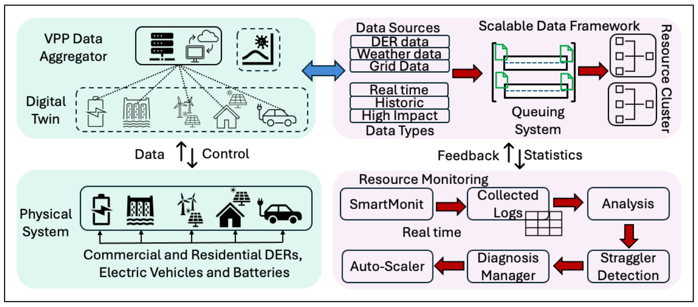


Fig. 2: VPP digital twin framework

The input layer consists of various DERs (rooftop photovoltaic, battery systems, small hydro systems, electric vehicles) deployed at several locations. These sensor-enabled DERs are connected to a virtual layer within the VPP digital twin. This virtual layer includes an aggregator that collects and aggregates the data generated from DERs, stores and processes it as and when required to accomplish various DER targets. Now, the collected data is forwarded to the processing system through an intermediary queuing. This system provides a reliable and efficient data flow between the source and destination systems. The queuing layer helps to control data loss caused by the limited bandwidth of the transmission medium. The managed/queued data is forwarded to the processing layer for analysis and decision-making. The data storage component amalgamates the functionality of Hadoop distributed storage and Spark's in-memory data processing capabilities [19]. The data stored on the configured HDFS framework [20] is further used for processing and making the necessary decisions at the virtual layer. The monitoring system (SmartMonit) is deployed to collect the resource utilization logs of the deployed resources in the processing and queuing layers. The collected resource utilisation logs are analysed, and accordingly, stragglers in the processing resources are identified by a diagnosis system. Based on the diagnosis, the scale-up and scale-down policies are configured at the processing and queuing layers. The proposed scheme works in different phases depicted in Figure 3.

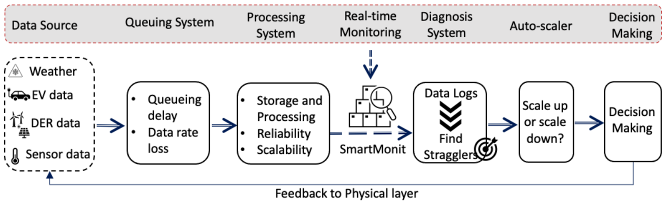


Fig. 3: Various phases of the proposed approach

A. Streaming Sensor Data Collection

In our proposed framework, the collection of streaming data begins at the input layer, where various sensors are deployed to collect data from multiple sources, such as DERs of hydro-power, solar energy, and wind energy deployed in various locations. These data are transmitted through a message broker system, RabbitMQ[3], an intermediary between the input and queuing layers. RabbitMQ employs a producer-consumer model: sensors function as producers, sending data encapsulated in messages to RabbitMQ's message queues and temporarily storing the data. Consumers, comprising computing devices and other processing units in the queuing layer, retrieve and process these messages. This queuing mechanism guarantees reliable, scalable, and fault-tolerant streaming data collection for sensor-enabled digital twins. It succeeds in dealing with low-idleness and lightweight jobs and offers progressed steering abilities for exact and effective message dispersion.

[3]https://www.rabbitmq.com/

B. Message queuing model

The message queuing model, depicted in Fig. 4, ensures efficient and reliable data transmission by using RabbitMQ servers deployed on multiple nodes within the Spark cluster. The workflow of the configured components of the queuing model is highlighted in the figure, considering the spark cluster and the management node. These RabbitMQ servers utilise a producer-consumer paradigm to manage the data flow between the cluster and queuing layers. Sensors, acting as producers, collect data encapsulated in messages, which are then sent to RabbitMQ's message queues on the nodes for temporary storage and systematic data flow management.

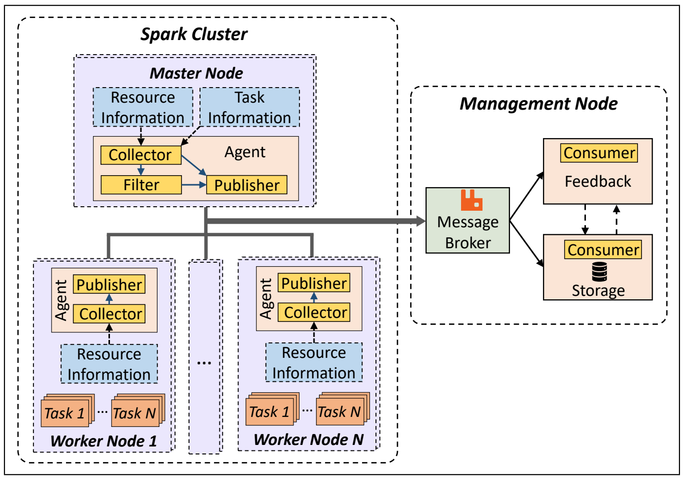


Fig. 4: Message queuing in the monitoring framework

The key components of the RabbitMQ server are defined below:

- ***Producer (Agent/Publisher):*** The producer, sometimes called an agent or publisher, creates and distributes messages. As producers would do, messages are sent to an exchange that determines how to route them rather than directly to a queue. Therefore, additional delay must be added to the total delay incurred in the network.
- ***Filter:*** The filter performs real-time preprocessing on all the collected logs to optimise the data flow and reduce data redundancy before sending them to the message broker system.
- ***Message Broker:*** RabbitMQ, as a message broker, receives, stores, and forwards messages between producers

and consumers. It also controls message routing, exchanges, and queues.

- ***Exchange:*** Messages from producers must be routed through an exchange according to the specified routing criteria to the relevant queue or queues. RabbitMQ supports different types of exchanges: direct exchange, fanout exchange, topic exchange, and header exchange.
- ***Diagnosis System*:** The term "auto-diagnosis" (inspired from AutoDiagn [21]) in RabbitMQ refers to built-in tools and capabilities that assist with identifying and reporting problems with the RabbitMQ system [21]. Numerous techniques have been incorporated into RabbitMQ to automatically manage, monitor, and troubleshoot potential issues in real time.
- **Consumer:** The messages routed from the message exchange are forwarded to the consumer for further processing. The consumer subscribes to the relevant exchange by binding to the correct queue to ensure the specific data type.

C. Data Processing

The DERs relay real-time operating data by constantly monitoring parameters, including voltage, temperature, and energy output. A RabbitMQ queuing system receives these sensor data and is the middleman message broker. RabbitMQ effectively controls the flow of the data stream, guaranteeing dependable communication between DERs and the processing framework. The data from RabbitMQ queues is consumed by the Hadoop-Spark cluster. The cluster processes the incoming data in real time, performing the required transformations and analysis using Apache Spark Streaming. After that, these data are sent to the DERs' digital twin, which operates inside the Spark environment. The behaviour and state of the real DERs are replicated in the digital twin, allowing for monitoring and simulation.

IV. Proposed Approach

The proposed approach comprises several fundamental components, elaborated in the subsequent sections.

A. Resource Log Collection and Analysis

SmartMonit [22] monitors processing cluster and queuing system to track infrastructure metrics such as CPU utilisation, memory utilisation, and network statistics. The RabbitMQ messaging platform captures the logs generated by the SmartMonit. RabbitMQ, acting as the intermediary message broker, makes it easier to collect and route these logs. After being enqueued, the logs are sent to a specific InfluxDB[4] instance after being methodically analysed. The logs are stored in InfluxDB, a time-series database for high-performance data intake and querying. This combination provides effective log management and analysis by utilising InfluxDB's specialised storage and retrieval mechanisms to efficiently retain and query the historical log data and RabbitMQ's strong queuing capabilities to handle large volumes of log data. As seen in the accompanying Fig. 5, the structure demonstrates how log data flows from the RabbitMQ system to the InfluxDB database, emphasising how simple it is to monitor and analyse resource utilisation and system performance.

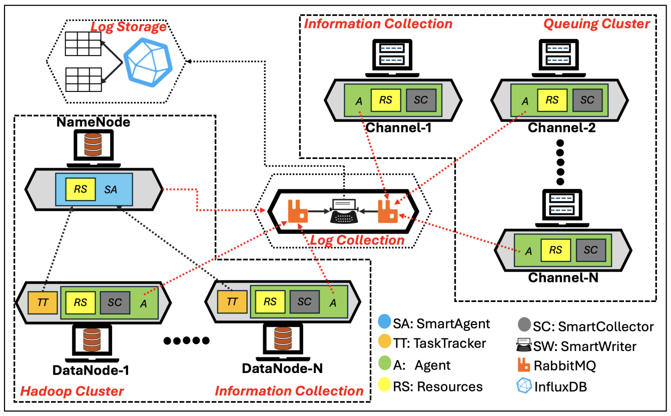


Fig. 5: Framework for storing Real-Time Resource Utilisation

In Alg. 1, the Hadoop/Spark cluster and Queuing system logs are stored in the Influx database until the Cluster Federation (group of clusters) is active. Each activated Hadoop cluster $\mathbf{CL_i}$ and Queuing Cluster $\mathbf{CC_i}$ are designated with active Agent $\mathbf{A}$ to forward the logs to the connected channel $\mathbf{RQ}$. Later, the logs are stored in the designated $\mathbf{TB}$. The Table I is used to store the logs of $\mathbf{CL}$ and Table II to store the logs of $\mathbf{CC}$ for futuristic analysis. The considered metrics to analyse the utilisation of the resources are highlighted in the Table I, II. The individual resource metrics are stored in the labelled table to monitor the performance of both activated clusters for data storage, processing and queuing.

Algorithm 1: Real-time Resource Utilisation Logger

```
Input:  $\mathbf{CF}$: Cluster Federation,
        $\mathbf{CL}$: Cluster of Hadoop/Spark,
        $\mathbf{NN}$: Node Name,
        $\mathbf{RQ}$: RabbitMQ Queue,
        $\mathbf{CC}$: Cluster of Channels (RabbitMQ),
        $\mathbf{ST}$: Sleep Time to re-collect logs
Output: $\mathbf{DB}$: Influx Database;
        $\mathbf{TB} \ni$ (Table $I$, Table $II$): Influx Database Tables
1  // Collect logs until the clusters are not $OFF$
2  while $\{\mathbf{CL} \neq OFF\}$ do
3      for each $\mathbf{CL_i}$ & $\mathbf{CC_i}$ in $\mathbf{CF}$ do
4          for each $\mathbf{NN_j}$ in $\mathbf{CL_i}$ & $\mathbf{CC_i}$ do
5              Forward Logs: $A_j \xrightarrow{forward} \mathbf{RQ}$   ▷ $\mathbf{A}$ : Agent of $\mathbf{NN_j}$
               //Forward the logs to the respective table
6              if $(NN_j \in CL)$ then
7                  $\mathbf{RQ} \xrightarrow{Forward} TableI;$
8              end
9              else
10                 $\mathbf{RQ} \xrightarrow{Forward} TableII;$
11             end
12         end
13     end
14     Reset($\mathbf{ST}$)                              ▷ Reset $(i, j)$=0 after $ST$
15 end
```

The analysis of the health status of the entire system is shown in Fig. 6. This layer adopts AutoDiagn [21], designed to perform root cause analysis by examining the logs collected by SmartMonit [22], which utilises advanced predefined functions to analyse the information of big data

[4]https://www.influxdata.com/

TABLE I: Processing utilisation metrics

Time	Measurement	Tag (role)	Field (CPU)	Field (memory)	Field (disk)
xxxx	Processing_cluster	Master	15.2%	60%	45%
xxxx	Processing_cluster	Worker_1	25.2%	56%	40%
xxxx	Processing_cluster	Worker_2	27.6%	58.5%	46.4%

TABLE II: Queuing utilisation metrics

Time	Measurement	Tag (role)	Field (CPU)	Field (memory)	Field (disk)
xxxx	Queuing_cluster	Master	21.4%	68%	49.5%
xxxx	Queuing_cluster	Worker_1	29.6%	51.9%	47.5%
xxxx	Queuing_cluster	Worker_2	31.6%	64.3%	54.7%

tasks (i.e., status, progress, execution time and data locality of mapper tasks) and infrastructure metrics such as CPU utilisation, memory utilisation, and network statistics of each node to identify potential issues and their underlying causes. By conducting detailed log analysis via *Diagnosers*, this layer can pinpoint anomalies, detect patterns indicative of failures, and provide actionable insights for troubleshooting and system optimisation to minimise downtime and maintain the overall reliability and performance of the digital twin environment and the explanation of the proposed scheme is elaborated in the defined algorithm. All this information is combined by *Diagnosis of the symptoms*, and then is forwarded to the *Diagnoser Manager* to update the *Feedback* for the system manager to take an action accordingly.

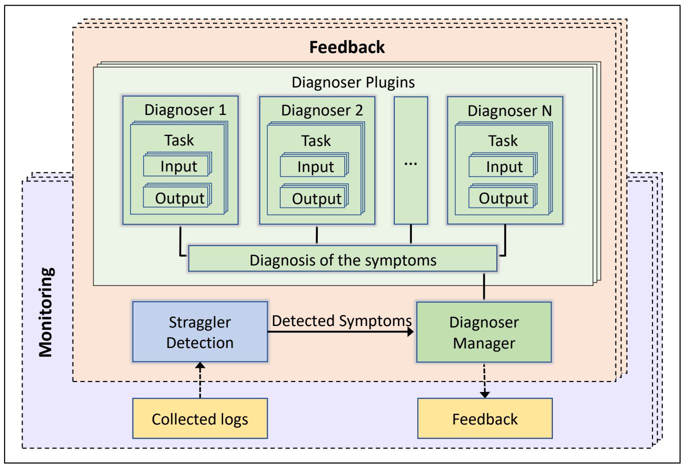


Fig. 6: Decision making for the feedback

In the Real-time Resource Utilisation Analyser approach defined in Alg. 2, the Influx Database tables $\mathbf{TB}$ are fetched from the output of Alg. 1. The maximum and minimum CPU utilisation threshold values $\mathbf{CPU^{th}_{max}}$ and $\mathbf{CPU^{th}_{min}}$ are defined as per the clock speed of the CPU. Similarly, the maximum and minimum Memory utilisation threshold value $\mathbf{Mem^{th}_{max}}$ and $\mathbf{Mem^{th}_{min}}$ are defined as per the capability of the resources. The data entries $\mathbf{D}$ from the $\mathbf{TB}$ are fetched to find any occurrence of the abnormality in the underlying network and resources. The $\mathbf{D}$ is forwarded to the Symptom Detection Engine ($\mathbf{SDE}$) to analyse the pattern of the record of the resources. The connected Diagnoser Manager ($\mathbf{DM}$) analyse the current CPU Utilisation ($\mathbf{CPU^{utl}}$) and Memory Utilisation ($\mathbf{Mem^{utl}}$) of the active configured resources. Along with the current utilisation of the resources, it also stores a flag ($\mathbf{id}$) to identify whether $\mathbf{CPU^{utl}}$ and $\mathbf{Mem^{utl}}$ belongs to $\mathbf{CL}$ or $\mathbf{CC}$. The evaluated value $\mathbf{CPU^{utl}}$, $\mathbf{Mem^{utl}}$, and $\mathbf{id}$ is stored in the $\mathbf{BM^{fdbk}}$. Later, the defined $\mathbf{CPU^{th}_{max}}$, $\mathbf{Mem^{th}_{max}}$, and $\mathbf{id}$ is matched with the values stored in the $\mathbf{BM^{fdbk}}$. In case, the $\mathbf{id}$ belongs to $\mathbf{CL}$ and the $\mathbf{CPU^{utl}}$, $\mathbf{Mem^{utl}}$ is equals and greater than the $\mathbf{CPU^{th}_{max}}$, $\mathbf{Mem^{th}_{max}}$, Root-cause Analysis ($\mathbf{RCA}$) buffer is updated with decision to scale-up the resources ($\mathbf{R^{CL}_{up}}$) in the Hadoop cluster and relevant objective functions are activated. In case, the $\mathbf{id}$ belongs to $\mathbf{CC}$ and the $\mathbf{CPU^{utl}}$, $\mathbf{Mem^{utl}}$ is equals and greater than the $\mathbf{CPU^{th}_{max}}$ and $\mathbf{Mem^{th}_{max}}$, the Root-cause Analysis ($\mathbf{RCA}$) buffer is updated with decision to scale-up the resources ($\mathbf{R^{CC}_{up}}$) in the RabbitMQ cluster and the corresponding objective function is activated. In another case, if again $\mathbf{id}$ belongs to $\mathbf{CL}$ and the $\mathbf{CPU^{utl}}$, $\mathbf{Mem^{utl}}$ is equals and less than the $\mathbf{CPU^{th}_{min}}$, $\mathbf{Mem^{th}_{min}}$, Root-cause Analysis ($\mathbf{RCA}$) buffer is updated with decision to scale-down the resources ($\mathbf{R^{CL}_{down}}$) in the Hadoop cluster.

Algorithm 2: Real-time Resource Utilisation Analyser

```
Input: $\mathbf{DB}$: Influx Database,                                    ▷ Alg. 1
       $\mathbf{TB} \in (\mathbf{TB^{CL}}, \mathbf{TB^{CC}})$: Influx Database Tables          ▷ Alg. 1
       $\mathbf{CPU^{th}_{max}}$ : Maximum CPU threshold value
       $\mathbf{Mem^{th}_{max}}$ : Maximum Memory threshold value
       $\mathbf{CPU^{th}_{min}}$ : Minimum CPU threshold value
       $\mathbf{Mem^{th}_{min}}$ : Minimum Memory threshold value
Output:
$\mathbf{RCA} = \{F_1, F_2, \mathbf{R^{CL}_{down}}, \mathbf{R^{CC}_{down}}, 0\}$ : Root Cause Analyser
1  // Read entries from $\mathbf{TB}$
2  while $\{\mathbf{TB} \neq NULL\}$ do
3      Fetch data: $\mathbf{D} \xleftarrow{fetch} \mathbf{TB}$                      ▷ $\mathbf{D}$ : Data
4      $\mathbf{D} \xrightarrow{fwd} \mathbf{SDE}$                   ▷ $\mathbf{SDE}$ :System Detection Engine
5      $\mathbf{SDE} \xrightarrow{data} \mathbf{DM}$                       ▷ $\mathbf{DM}$ :Diagnoser Manager
6      //Diagnoser Manager (DM) analyse the data as per policy
7      Store feedback: $\mathbf{BM^{fdbk}} \xleftarrow{feedback} \mathbf{DM}$ ▷ $\mathbf{BM}$ : Buffer Memory
8      $\mathbf{BM^{fdbk}} \ni \{\mathbf{CPU^{utl}}, \mathbf{Mem^{utl}}, id\}$          ▷ $id \ni \{\mathbf{CL}, \mathbf{CC}\}$
9      if $(id == \mathbf{CL}\&\mathbf{CPU^{utl}} \geq \mathbf{CPU^{th}_{max}}\&\mathbf{Mem^{utl}} \geq \mathbf{Mem^{th}_{max}}$
        $)$ then
10         Set objective function: $F_{21}(\aleph) = F_{21}(1)$;
11         Set objective function: $F_{22}(\aleph) = F_{22}(1)$;
12         Store result: $\mathbf{RCA} = F_2$                  ▷ Use Eq. 14 ;
13     end
14     else if
        $(id == \mathbf{CC}\&\mathbf{CPU^{utl}} \geq \mathbf{CPU^{th}_{max}}\&\mathbf{Mem^{utl}} \geq \mathbf{Mem^{th}_{max}}$
        $)$ then
15         Set objective function: $F_{11}(\aleph) = F_{21}(1)$;
16         Set objective function: $F_{12}(\aleph) = F_{22}(1)$;
17         Store result: $\mathbf{RCA} = F_1$                   ▷ Use Eq. 2;
18     end
19     else if
        $(id == \mathbf{CL}\&\mathbf{CPU^{utl}} \leq \mathbf{CPU^{th}_{min}}\&\mathbf{Mem^{utl}} \leq \mathbf{Mem^{th}_{min}}$ $)$
        then
20         Store result: $\mathbf{RCA} = \mathbf{R^{CL}_{down}}$;
21     end
22     else if
        $(id == \mathbf{CC}\&\mathbf{CPU^{utl}} \leq \mathbf{CPU^{th}_{min}}\&\mathbf{Mem^{utl}} \leq \mathbf{Mem^{th}_{min}}$
        $)$ then
23         Store result: $\mathbf{RCA} = \mathbf{R^{CC}_{down}}$;
24     end
25     else
26         Store result: $\mathbf{RCA} = 0$;
27                                                       ▷ 0: No Action
28     end
29 end
```

In case, the $\mathbf{id}$ belongs to $\mathbf{CC}$ and the $\mathbf{CPU^{utl}}$, $\mathbf{Mem^{utl}}$ is equals and less than the $\mathbf{CPU^{th}_{min}}$ and $\mathbf{Mem^{th}_{min}}$, the Root-cause Analysis ($\mathbf{RCA}$) buffer is updated with decision to scale-down the activated resources ($\mathbf{R^{CC}_{up}}$) in the RabbitMQ cluster. By default, the value of the $\aleph$ is set to 0. If a need is to target any objective function, the value of $\aleph$ is set to 1. Further, the $\mathbf{RCA}$ is set to F_1 or F_2 as per the decision made by the algorithm. In standard scenarios, the label $\mathbf{RCA}$ is updated with 0, and no further action is required. In this case, all the objective functions $\aleph$ index is set to 0, which means all the objective functions are already satisfied.

B. Intelligent Orchestrator: Auto-scaler

An innovative load-balancing approach strategy in the distributed digital twin system is mentioned in Alg. 3. An orchestrator is designed to achieve the activated objective functions by scaling up or scaling down the resources as per the decision made by the deployed ML model by analysing the workload generated by deployed sensors.

The resource prediction model is developed to dynamically forecast resource scaling decisions to ensure efficient system resource management under varying workloads. It is trained by utilising the log data collected via SmartMonit, including CPU and memory utilisation, the details of the tasks hosted by the machine, queuing delay, data input/output rates, and timestamps in the big data environment. This model deploys the Decision Tree algorithm, alerting possible future resource constraints. CPU utilisation thresholds are usually 70% to 80%, considered moderately loaded on a system [23]. The system acts quickly and scales out when CPU utilisation crosses 85%; if we wait for it to shore past the critical point, performance will go down, and bottlenecks will occur. With memory, autoscaling is usually implemented when utilisation hits 70%-75%, and over the top of that (from about 85%) [24], it implies a potential new memory bottleneck scenario where latency rises, and many disk swapping use cases start to affect performance. Using these thresholds, the model forecasts resource crunch and advises Orchestrator to spin up more Spark clusters as demands go high.

Algorithm 3: Orchestrator: Auto-scaling approach

```
Input:  $\mathbf{RCA} = \{\mathbf{R^{CL}_{up}}, \mathbf{R^{CC}_{up}}, \mathbf{R^{CL}_{down}}, \mathbf{R^{CC}_{down}}, 0\}$   ▷ Alg. 2
        $\delta_{pool} \ni \{\mathbf{CL}_{sleep}, \mathbf{CC}_{sleep}\}$: Dormant cluster
Output: Achieve objective functions: $\mathbf{RS} \leftrightarrow F_1, F_2$
1  // Fetch label from RCA
2  if ($\mathbf{RCA} \ni \{\mathbf{F_2} \vee \mathbf{F_1}\}$) then
3      Calculate: $\mathcal{R}^{req}_{tot}$ using ML model
4      Activate optimal cluster $\leftarrow \delta_{pool}$
5  end
6  else if ($\mathbf{RCA} \ni \{\mathbf{R}^{CL}_{down} \vee \mathbf{R}^{CC}_{down}\}$) then
7      Calculate: $\mathcal{R}^{req}_{tot}$ using ML model
8      Put extra cluster on sleep mode from $\mathbf{CF}$
9      Shift the data on the active clusters
10 end
11 Update Influx Database Table: $\mathbf{TB}$ using Alg. 1
12 Update Root Cause Analyser: $\mathbf{RCA}$ using Alg. 2
13 if $\mathbf{RCA} == 0$ then
14     Declare System: $\rightarrow \mathbf{RS}$   ▷ Reliable Stage
15 end
16 else
17     Repeat steps 1
18 end
```

The label $\mathbf{RCA}$ is fetched from Alg. 2 to target the specified objective functions by auto-scale the resources to make a reliable system in a digital twin environment. If the fetched labels are $\mathbf{R}_{up}$, then the orchestrator targets the objective functions ($\mathbf{F}_1 \vee \mathbf{F}_2$). The trained machine learning (ML) model is used to dynamically predict the optimal resource allocation required to achieve specified objectives. By leveraging predictive insights from the ML model, resource selection becomes significantly more efficient, ensuring that computational and storage resources are provisioned to minimise overhead while meeting performance and scalability requirements. Further, activate the optimal cluster as per $\mathcal{R}^{req}_{tot}$ from the list of Dormant Cluster (δ_{pool}) to normalise the requirement of the resources for processing or storage. In another way, if the fetched labels are $\mathbf{R}_{down}$, the activated resources are more than the requirement. Therefore, there is a need to put a few of the resources into sleep mode. To normalise the under-utilisation of resources, an optimal cluster is deactivated with the output provided by the Decision Tree algorithm, and the cluster index is updated in the δ_{pool} for futuristic use. Further, the Alg. 1 is called to collect the logs of the activated clusters. The resource utilisation is analysed using Alg. 2, and resultant labels are forwarded to the Orchestrator for further decision-making. If the updated label is 0, it depicts the current system is at Reliable Stage ($\mathbf{RS}$); otherwise, step 1 is called again for further actions.

The primary objective of the orchestrator is to achieve the combined objective function for better decision-making on the digital twin running on the Spark cluster. Accordingly, the system's decision or feedback is forwarded to the physical system (i.e., DERs) to enhance its performance.

V. Simulation and Result analysis

This section provides a comprehensive evaluation of the proposed scheme using varied scenarios and metrics. We also performed the power model validation for the proposed framework.

A. Real-time Set Up

The real-time setup comprises a range of powerful systems and software tools designed for high-performance computing and monitoring. At its core is the 3XS Development Box Pro G1-32C, which features an AMD Ryzen Threadripper PRO processor with 32 cores, 64 GB of RAM, and an NVIDIA GeForce RTX 4090 graphics card with 24 GB of VRAM, all running on Linux. This machine is ideal for cluster formation due to its processing power. Another machine, MacBook M1 pro, embedded with an M1 chip with an 8-core CPU and 16 GB of RAM. This machine is embedded with an internal 8-core GPU, providing exceptional performance while forming another cluster for storage and processing the allocated tasks. Additionally, the Sony i3 system with Intel I3 processor, and 8 GB of RAM is used for deploying DERs as a data source in the digital twin environment. A queuing system known as RabbitMQ, a messroker, is also configured to the environment supporting the AMQP protocol in the environment. This tool provides features such as high availability, clustering, a management UI, and a plugin system operating on Linux. Lastly, SmartMonit is configured to monitor the deployed resources in the environment. The real-time setup details are also highlighted in Table III.

TABLE III: Real-Time Setup Specifications

Component	Specifications
3XS Development Box Pro G1-32C	**Processor:** AMD Ryzen Threadripper PRO (32-core) **RAM:** 64 GB **Graphics:** NVIDIA GeForce RTX 4090 (24 GB VRAM) **Operating System:** Linux
MacBook Pro M1	**Processor:** Apple M1 Chip (8-core) **RAM:** 16 GB **Graphics:** Integrated 8-core GPU **Operating System:** macOS
Sony i3 System	**Processor:** Intel Core i3 **RAM:** 8 GB **Graphics:** Integrated Intel HD Graphics **Operating System:** Windows
RabbitMQ	**Type:** Message Broker **Protocol:** AMQP **Features:** High availability, Clustering, Management UI, Plugin system **Operating System:** Linux
SmartMonit	**Type:** Monitoring Tool **Features:** Real-time performance metrics, Alerting, Historical data analysis **Operating System:** Linux

B. System Overheads

We evaluate the system overhead introduced by the deployed monitoring and root cause analysis tool by measuring the CPU and memory usage of the Monitoring and Root cause analyser. Table IV shows that *Monitoring* uses approximately 2.74% memory and 4.84% CPU, while *Root cause analyser* consumes 2.62% memory and 3.71% CPU.

TABLE IV: Resource overhead caused by components

Components	Mem (%)	CPU (%)
Monitoring	2.74	4.84
Root cause analyser	2.62	3.71

C. Evaluation of the Prediction Model

Fig. 7 shows the model evaluation for F1 Score, Precision, Recall and Accuracy. The F1 score is 0.912, balanced between Precision and Recall, and is best suited for both high false positives and high false negatives. Precision is slightly off with 0.874, meaning the model performs well generally, but it could potentially give a few false positives. The recall is 0.913, indicating that the model effectively finds positive cases and has performed well in preventing false negatives. The model finally had an accuracy of 0.927, which means the share of right predictions over all the predictions made.

D. Validation of the Auto-scaler approach

Fig. 8 shows the impact of increasing the number of nodes on energy consumption, straggler frequency, and execution time. Using more resources as the number of nodes grows

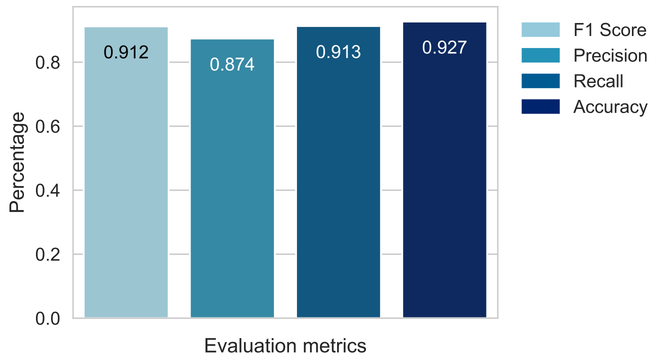


Fig. 7: Performance evaluation values of the ML model

leads to energy consumption. Nevertheless, straggler frequency drops since the workload is shared across more nodes, yielding higher efficiency. Also, total execution time decreases with increased nodes, indicating better task completion performance.

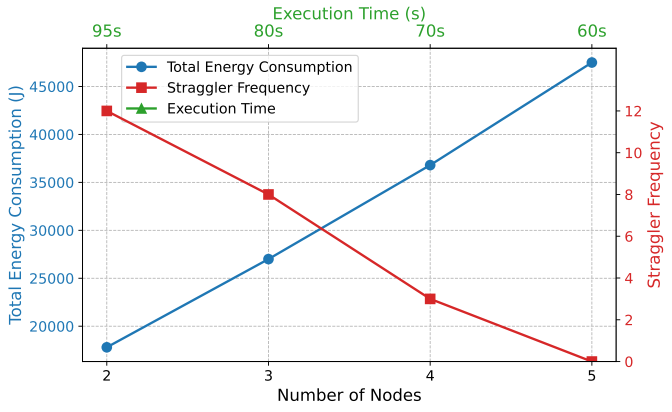


Fig. 8: Impact of node count on energy, stragglers, and time

Fig. 9 illustrates the relationship between the number of sensors transmitting data and the corresponding message rates and data size. As the number of sensors increases, message rates and data size grow linearly. This indicates that as more sensors send data simultaneously, the overall load on the system rises proportionally regarding the number of messages per second and the volume of data transmitted. This is a linear relationship since each sensor sends a fixed amount of information per transmitted message at constant intervals. Hence, the message rate scales up proportionately with the number of sensors, leading to a proportional increase in the total data size. Such issues underline the inherent challenges of scaling the management of large sensor networks, where the data burden may hinder system performance and resource needs.

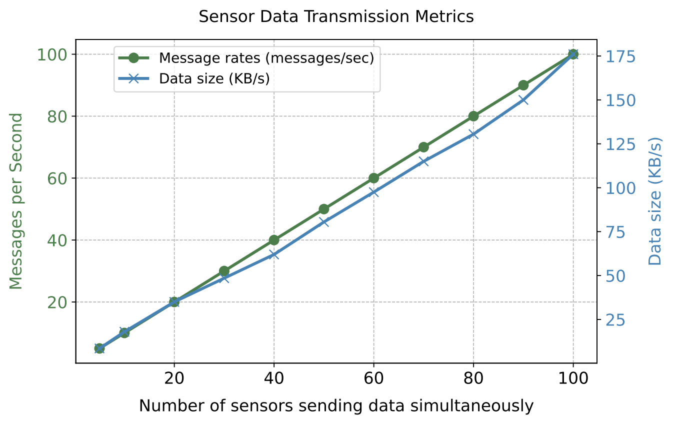


Fig. 9: Data size and Message rates vs. Number of sensors

Fig. 10 demonstrates the data rate against delay and packet drop under fixed-cluster configuration without dynamic scaling mechanisms. The result of increasing the data rate from 10 MB/s to 100 MB/s is that both the delay and packet drop increase sharply, illustrating that the fixed cluster cannot handle such heavy work for this configuration. As the data rate increases, this leads to much longer processing time and higher packet drop rates. These results emphasize the constraints of a fixed resource allocation strategy and justify that an auto-scaling mechanism is required. Scaling up the cluster adds more capacity in these cases, which helps reduce those delays, request processing and keep packets intact since resources can be adjusted in real time.

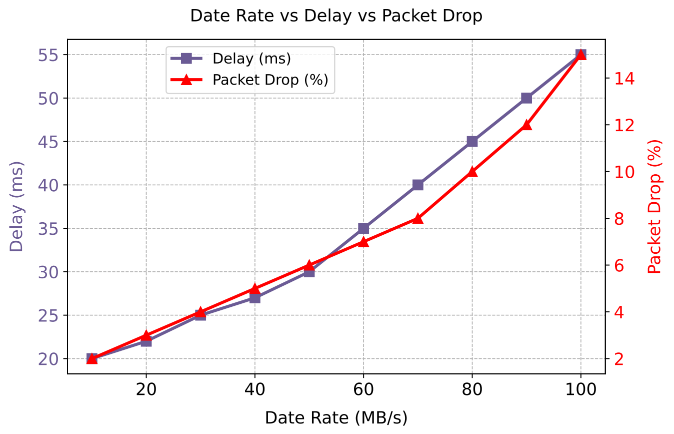
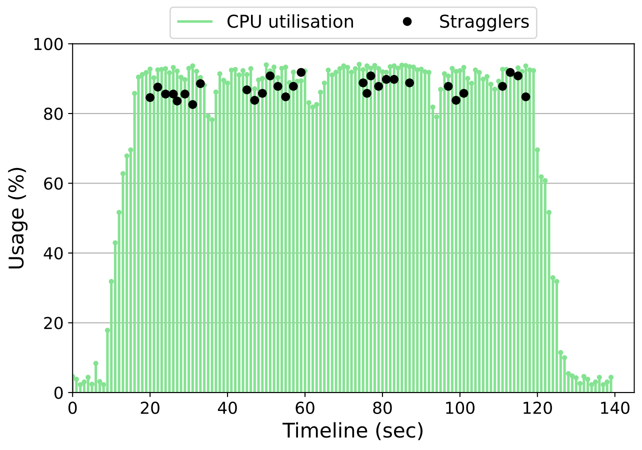
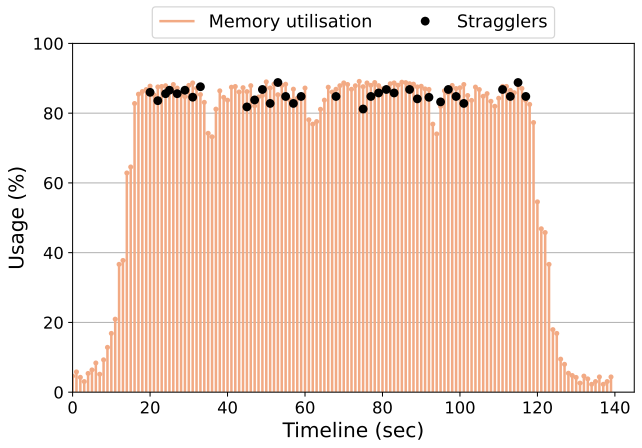


Fig. 10: Effect of Increasing Data Rate on Delay and Packet Drop

Fig. 11 shows occurrences of stragglers in a big data cluster during high CPU and memory utilisation. Stragglers appear under heavy CPU load in Fig. 11a and during high memory usage in Fig. 11b. The appearance of stragglers in both scenarios underscores the performance challenges faced during periods of high resource demand. The findings show that if proper dynamic resource allocation strategies are not in place, the stragglers can cause a significant loss of system performance. A powerful scale-up and scale-down approach is proposed to resolve these issue, which uses adaptive resource management to reduce delays due to overloaded nodes.



(a) Stragglers during high CPU utilisation

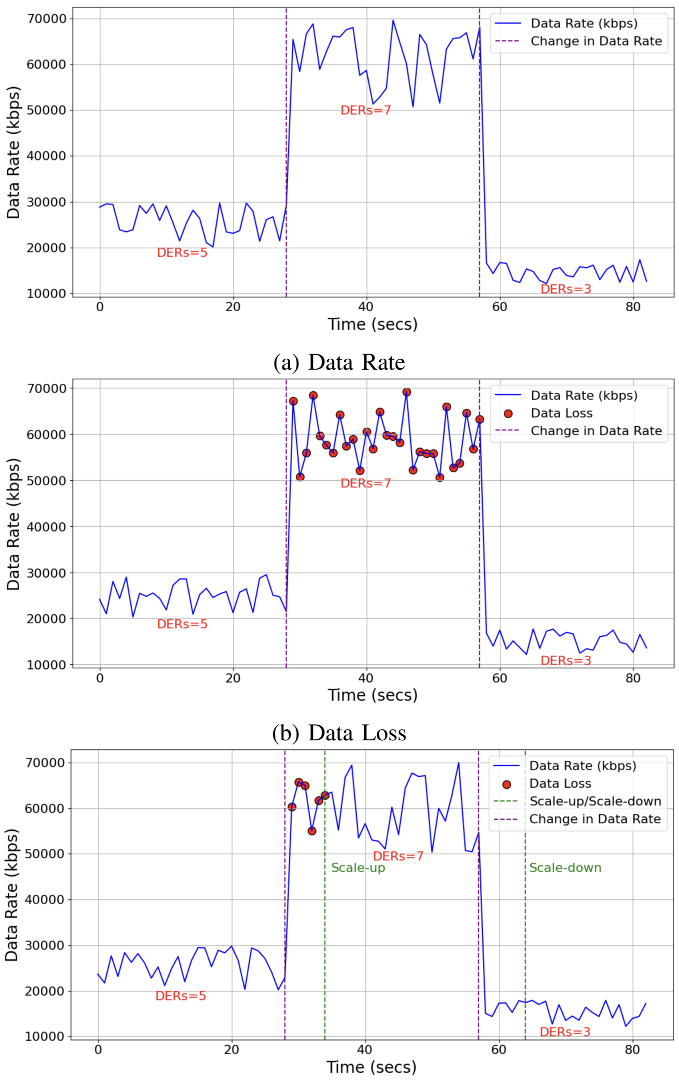


(b) Stragglers during high memory utilisation

Fig. 11: Occurrences of Stragglers under High Utilisation

E. Case Study

A system comprising a 3-node cluster and another with a 4-node cluster is configured for a case study. To validate the proposed scheme, Orchestrator: Auto Scaler, the system undergoes testing at two levels: a) **Queuing Level** and b) **Processing Level**. In the first level, a single channel is configured on a Linux system equipped with 2GB of memory and a dual-core processor to handle the data collected from deployed sensors. An additional channel with identical specifications is created for back-up purposes but remains in sleep mode. The validation of this scenario is illustrated in Fig. 12. As depicted in Fig. 12a, variable data rates are generated from various DERs and collected through the deployed sensors across different levels. With five active DERs, the operational channel manages the forwarded data without any loss. However, as two additional DERs are activated, the change in data rate becomes apparent, as shown in Fig. 12b. The configured channel can accommodate the data rate to a certain extent; beyond this threshold, data loss occurs, as highlighted by the red circles in Fig. 12b. Additional resources are required within the framework to manage the increased data rate effectively. Consequently, the Orchestrator activates the necessary resources recommended by the ML model. The channel initially in sleep mode is subsequently activated, redirecting the data collected by the deployed sensors to this newly created channel to mitigate data loss, as depicted in Fig. 12c. This transition restores the system to a reliable state.



(a) Data Rate

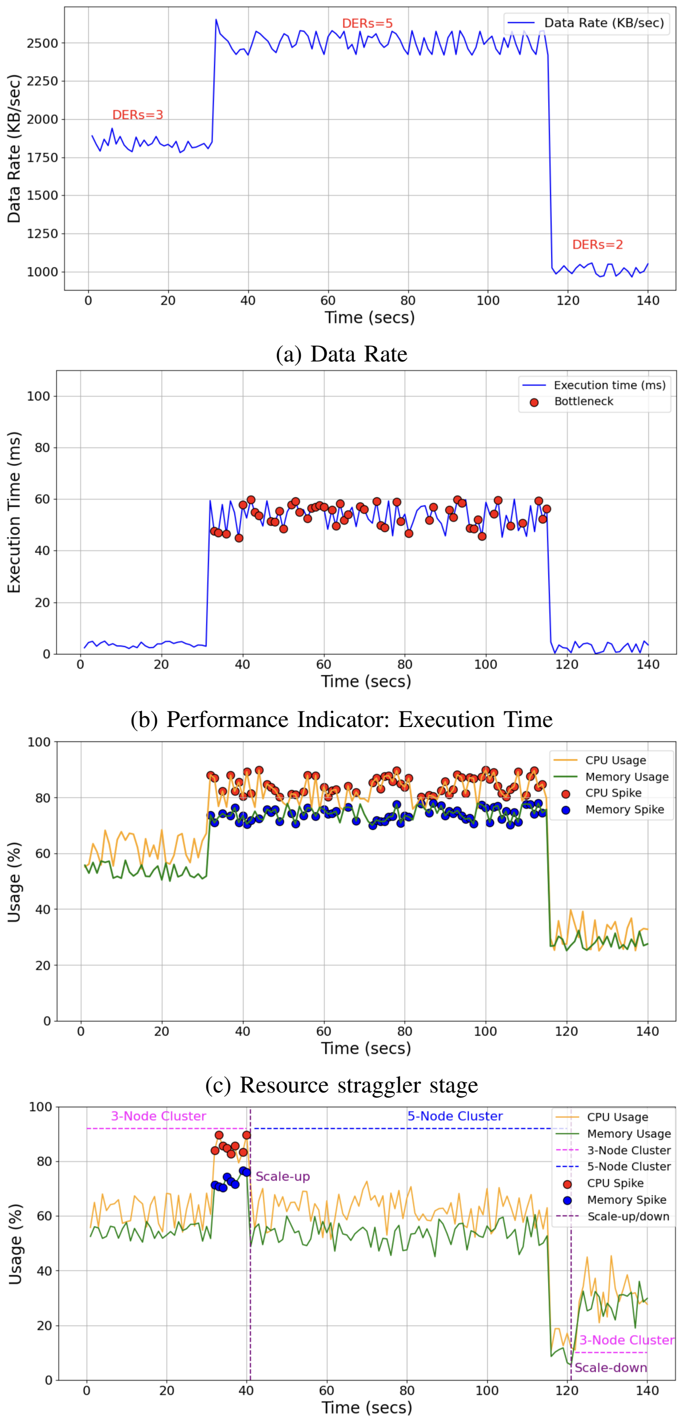
(b) Data Loss

(c) Reliable stage (Auto-scaler)

Fig. 12: Orchestrator: Queuing Auto-scaler

In another scenario; **Processing Level**, Fig. 13 illustrates the overall performance of the **Orchestrator: Processing Auto-scaler**.

Fig. 13a displays the data rates from various deployed DERs. Initially, the framework activates two DERs for real-time data streaming. After a few seconds, three additional DERs are activated, and the resulting impact on the data flow rate is evident in the figure. Subsequently, two DERs are deactivated to demonstrate the variations in the streaming data flow and its influence on the proposed system. With the change in the data rate, the execution time of the data is also changed subject to the capacity of the underlying system, as highlighted in Fig. 13b. In the case of sensitive applications, a delay in the execution of the data is not tolerable, can create a bottleneck situation for the system and directly impact



(a) Data Rate

(b) Performance Indicator: Execution Time

(c) Resource straggler stage

(d) Scale-up/down approach

Fig. 13: Orchestrator: Processing Auto-scaler

the performance indicator set for the configured system as highlighted in Fig. 13b. Therefore, the Orchestrator follows the root cause analysis and finds the stragglers in the running system and comes up with the results shown in Fig. 13c. The Figure highlights resource utilisation in CPU and Memory performance within a three-node cluster. The workload is minimal at the outset, and the deployed resources are not over-utilised. When all DERs are activated, the streaming data rate increases significantly, as indicated by the red circles in

Figure 13c. After some time, the two DERs are deactivated, normalising resource utilisation as shown in the figure. At this point, there is a risk of losing streaming data due to over-utilisation of the deployed resources. To address this, the Orchestrator runs the decision tree algorithm to calculate the resources required to manage the streaming data, and an optimised cluster is activated accordingly, as illustrated in Figure 13d. At the 42^{nd} second mark of the streaming data, the Orchestrator activates the optimal cluster, resulting in normalised resource utilisation.

F. Evaluation of the Power Model

In distributed systems, the resource utilisation of computing nodes, such as processors, memory, storage, and network components, significantly affects their energy consumption. Among these resources, the processor is the largest energy consumer. Furthermore, processor utilisation typically represents the overall load on the machine [25]. Therefore, this model focuses on tracking processor energy usage.

- **Resource Utilisation Assessment:** Efficient resource utilisation in distributed systems substantially impacts overall performance and cost-effectiveness. High resource utilisation ensures the effective use of computational resources, thereby increasing profits and reducing energy consumption by minimising idle resources. Consequently, a resource management technique should be evaluated based on resource utilisation using the following equation [26]:

$$U_m(H,\tau) = \sum_{i=1}^{W} h_{ii} \times \frac{\mathrm{R}_{\mathrm{CPU}i}(\tau)}{\mathrm{CPU}_m} \tag{17}$$

  Equation 17 shows the utilisation $U_m(H,\tau)$ of a server N_m at a specific time τ. H represents the allocation of virtual machines (VMs), while h_{ii} denotes whether a VM V_i is hosted on the server N_m. If V_i is hosted on N_m, the value of h_{ii} is one; otherwise, it is zero. CPU_m signifies the total computational capacity of N_m, and $\mathrm{R}_{\mathrm{CPU}i}(\tau)$ is the CPU capacity required by V_i at a given time.
- **Energy Consumption Calculation:** As outlined in the previous section, resource utilisation directly influences energy efficiency. Higher resource utilisation improves energy efficiency by ensuring servers actively process tasks instead of idling. To compute the power consumption of a specific server at a given time τ with an allocation H, the following equation is used [25]:

$$Q_m(H,\tau) = 0.7Q_m^{\max} + 0.3Q_m^{\max} \times U_m(H,\tau) \tag{18}$$

  where $Q_m^{\max}$ is the maximum power consumed by a server when it is fully utilised, and $U_m(H,\tau)$ is the utilisation of the server at time τ.
  To calculate the total power consumption of all servers between time τ_1 and τ_2, the following equation is used:

$$\mathrm{Power}(H,\tau_1,\tau_2) = \sum_{n=1}^{N} \int_{\tau_1}^{\tau_2} Q_m(H,\tau)\, d\tau \tag{19}$$

  where N is the total number of servers, H is the allocation of the servers, and $Q_m(H,\tau)$ is the power consumption of the m-th server at time τ.

As discussed above, we consider energy consumption based on CPU, memory, and network utilisation. The evaluation of our energy model involves analysing dynamic power and total energy consumption concerning the makespan (total task execution time). Power consumption multiplied by the time of execution provides the total energy. Table V summarises the key findings. Dynamic power consumption ranges from 50 to 70 watts, clustering around 55-65 and 65-70 watts, indicating variability due to task complexity and resource allocation. Power consumption becomes more consistent as the makespan increases from 60 to 95 units. Total energy consumption ranges from 8500 to 9900 joules, clustering between 8900 and 9500 joules. Energy consumption generally increases with makespan, reflecting the direct relationship between execution time and total energy usage.

The effect of increasing the number of nodes on energy consumption, straggler frequency, and execution time is shown in Figure 8. We derive these trends through a power model, which relates energy consumption to resource utilisation and workload distribution. With more processors in the system, the energy consumption (due to the additional active resources) increases with the number of nodes, while the straggler frequency and execution time decrease due to better workload distribution.

TABLE V: Summary of dynamic power and energy consumption per machine

Metric	Observation
Dynamic Power (W)	Node: 55-70 W
Makespan (units)	60-95 sec.
Total Energy (J)	Node: 8900-9500 J
Energy vs. Makespan	More time, more energy

VI. CONCLUSION

This work introduces a comprehensive data framework to tackle the substantial challenges of data storage, processing, and resource management in VPP digital twins integrated with sensor-enabled DERs. By deploying a cloud-based big data cluster, the framework ensures scalable and reliable storage, complemented by RabbitMQ for efficient data queuing and continuous monitoring to support dynamic data management. Furthermore, our innovative load-balancing strategies dynamically manage network traffic using the queuing system. At the same time, the Orchestrator optimises resource utilisation by scaling resources using the decision tree algorithm. Including a failure detection component that performs root cause analysis ensures actionable insights for continuous system optimisation. Experimental results validate the framework's effectiveness in achieving high efficiency, reliability, and real-time operational capability, addressing critical issues in data storage, streaming data analysis, and load balancing. In future, coordinating edge technology with the proposed framework fundamentally upgrade the framework by empowering restricted information handling near DERs, resultant into reduce latency, improve real-time decision-making, and alleviate the burden on central cloud systems.

References

[1] J. Han, Q. Hong, M. H. Syed, M. A. U. Khan, G. Yang, G. Burt, and C. Booth, "Cloud-edge hosted digital twins for coordinated control of distributed energy resources," *IEEE Transactions on Cloud Computing*, vol. 11, no. 2, pp. 1242–1256, 2023.

[2] T. Wang, D. O'Neill, and H. Kamath, "Dynamic control and optimization of distributed energy resources in a microgrid," *IEEE Transactions on Smart Grid*, vol. 6, 09 2014.

[3] M. A. Shuvra and B. Chowdhury, "Load management system and control strategies of distributed energy resources in an islanded microgrid," in *2019 IEEE 16th International Conference on Smart Cities: Improving Quality of Life Using ICT & IoT and AI (HONET-ICT)*, 2019, pp. 100–104.

[4] C. B. K. Yadav and D. Dash, "An efficient partial charging and data gathering strategy using multiple mobile vehicles in wireless rechargeable sensor networks," *Cluster Computing*, pp. 1–22, 2024.

[5] N. Murugan, G. Devarajan, S. M, R. V, A. Bashir, and A. Ali, "Artificial intelligence based zero trust security approach for consumer industry," *IEEE Transactions on Consumer Electronics*, vol. PP, pp. 1–1, 01 2024.

[6] D. Ochoa, F. Galarza-Jimenez, F. Wilches-Bernal, D. Schoenwald, and J. Poveda, "Control systems for low-inertia power grids: A survey on virtual power plants," *IEEE Access*, vol. 11, pp. 20 560–20 581, 01 2023.

[7] A. Eggebeen, M. Vygoder, G. Oriti, J. Gudex, A. L. Julian, and R. M. Cuzner, "The use of digital twins in inverter-based ders to improve nanogrid fault recovery," in *2023 IEEE Energy Conversion Congress and Exposition (ECCE)*, 2023, pp. 734–741.

[8] Y. Tao, J. Wu, Q. Pan, A. K. Bashir, and M. Omar, "O-ran-based digital twin function virtualization for sustainable iov service response: An asynchronous hierarchical reinforcement learning approach," *IEEE Transactions on Green Communications and Networking*, 2024.

[9] K. Sun, J. Wu, A. K. Bashir, J. Li, H. Xu, Q. Pan, and Y. D. Al-Otaibi, "Personalized privacy-preserving distributed artificial intelligence for digital-twin-driven vehicle road cooperation," *IEEE Internet of Things Journal*, 2024.

[10] A. Garg, G. S. Aujla, and H. Sun, "Analyzing impact of data uncertainty in distributed energy resources using bayesian networks," in *2023 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. IEEE, 2023, pp. 1–6.

[11] C. Perera, Y. Qin, J. C. Estrella, S. Reiff-Marganiec, and A. V. Vasilakos, "Fog computing for sustainable smart cities: A survey," *ACM Computing Surveys (CSUR)*, vol. 50, no. 3, pp. 1–43, 2017.

[12] S. Y. Teng, M. Touš, W. D. Leong, B. S. How, H. L. Lam, and V. Máša, "Recent advances on industrial data-driven energy savings: Digital twins and infrastructures," *Renewable and Sustainable Energy Reviews*, vol. 135, p. 110208, 2021.

[13] A. Gharaibeh, M. A. Salahuddin, S. J. Hussini, A. Khreishah, I. Khalil, M. Guizani, and A. Al-Fuqaha, "Smart cities: A survey on data management, security, and enabling technologies," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2456–2501, 2017.

[14] A. Jindal, N. Kumar, and M. Singh, "A unified framework for big data acquisition, storage, and analytics for demand response management in smart cities," *Future Generation Computer Systems*, vol. 108, pp. 921–934, 2020.

[15] Y. Pan, T. Qu, N. Wu, M. Khalgui, and G. Huang, "Digital twin based real-time production logistics synchronization system in a multi-level computing architecture," *Journal of Manufacturing Systems*, vol. 58, pp. 246–260, 2021.

[16] J. Rosenberger, M. Urlaub, F. Rauterberg, T. Lutz, A. Selig, M. Bühren, and D. Schramm, "Deep reinforcement learning multi-agent system for resource allocation in industrial internet of things," *Sensors*, vol. 22, no. 11, p. 4099, 2022.

[17] W. Chen, Z. Milosevic, F. A. Rabhi, and A. Berry, "Real-time analytics: Concepts, architectures and ml/ai considerations," *IEEE Access*, 2023.

[18] G. S. Aujla and N. Kumar, "Mensus: An efficient scheme for energy management with sustainability of cloud data centers in edge–cloud environment," *Future Generation Computer Systems*, vol. 86, pp. 1279–1300, 2018.

[19] J. C. Perafan-Villota, O. H. Mondragon, and W. M. Mayor-Toro, "Fast and precise: Parallel processing of vehicle traffic videos using big data analytics," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 8, pp. 12 064–12 073, 2022.

[20] A. D'Alconzo, I. Drago, A. Morichetta, M. Mellia, and P. Casas, "A survey on big data for network traffic monitoring and analysis," *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 800–813, 2019.

[21] U. Demirbaga, Z. Wen, A. Noor, K. Mitra, K. Alwasel, S. Garg, A. Y. Zomaya, and R. Ranjan, "Autodiagn: An automated real-time diagnosis framework for big data systems," *IEEE Transactions on Computers*, vol. 71, no. 5, pp. 1035–1048, 2021.

[22] U. Demirbaga, A. Noor, Z. Wen, P. James, K. Mitra, and R. Ranjan, "Smartmonit: Real-time big data monitoring system," in *2019 38th symposium on reliable distributed systems (SRDS)*. IEEE, 2019, pp. 357–3572.

[23] E. Casalicchio, "A study on performance measures for auto-scaling cpu-intensive containerized applications," *Cluster Computing*, vol. 22, no. 3, pp. 995–1006, 2019.

[24] C. Li, J. Zheng, H. Okamura, and T. Dohi, "Performance evaluation of a cloud datacenter using cpu utilization data," *Mathematics*, vol. 11, no. 3, p. 513, 2023.

[25] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future generation computer systems*, vol. 28, no. 5, pp. 755–768, 2012.

[26] S. Mustafa, B. Nazir, A. Hayat, A. u. R. Khan, and S. A. Madani, "Resource management in cloud computing: Taxonomy, prospects, and challenges," *Computers & Electrical Engineering*, vol. 47, pp. 186–203, 2015.