Scalable and Interoperable Distributed Architecture for IoT in Smart Cities

VJS Pranavasri*, Leo Francis*, Ushasri Mogadali, Gaurav Pal,
SVSLN Surya Suhas Vaddhiparthy, Anuradha Vattem, Karthik Vaidhyanathan, Deepak Gangadharan
Smart City Research Centre, *IIIT Hyderabad* India
vjs.pranavasri@research.iiit.ac.in, leojfrancis.now@gmail.com,
{ushasree.mogadali, gaurav.pal, suhas.vaddhipar, anuradha.vattem}@research.iiit.ac.in
{karthik.vaidhyanathan, deepak.g}@iiit.ac.in

***Abstract*—The increase of IoT devices and the emergence of smart cities have revolutionized urban development, offering numerous benefits while addressing environmental concerns. This has caused an increase in the usage of IoT frameworks, and the need for efficient architecture and standardized ontology is imminent. In this regard, we propose a distributed, multi-layered data platform architecture comprising the Data Monitoring Layer (DML), Data Storage Layer (DSL), Data Enhancement Layer (DEnL), and Data Exchange Layer (DEL). Our architecture achieves interoperability, facilitates data transfer between nodes, enables telemetry data retrieval, and ensures cross-platform and cross-device compatibility. It addresses the challenges of handling increased sensor data and user demands by providing high throughput and scalability support. We investigated Smart City Living Lab at IIIT Hyderabad an existing large-scale system deployed within a 66-acre campus. This system consists of 291 nodes. By studying this deployed system, we were able to gather valuable real-world data, allowing us to analyze the challenges and potential solutions related to data architecture. Our results show improvements of up to 41.23% in throughput and a decrease in latency by 29.19% for data insertion from the sensor nodes. The retrieval by the data client gives an increase of over 800% in both throughput and number of requests through DENL. These metrics are compared to a centralised data platform architecture. We conclude by discussing the implications of our findings and suggesting future work.**

***Index Terms*—smartcity; data platform architecture; oneM2M; IoT; distributed; performance analysis;**

I. Introduction

The increasing population and demand for urbanization in smart cities necessitate efficient connectivity and automation solutions. These processes become crucial in addressing key challenges such as energy utilization, waste management, water quality, and pollution [1] [2]. Smart cities aim to seamlessly integrate sectors through interconnected systems, enabling real-time monitoring, analysis, and efficient management of critical infrastructure. These advancements empower cities to make data-driven decisions, enhance operational efficiency, foster innovation, and engage citizens in decision-making processes [3] [4].

Smart cities successfully address issues such as carbon footprint [5] and resource utilization through technologies such as the Internet of Things (IoT), data analytics, and connectivity to enhance sustainability and improve infrastructure efficiency. These deployments encompass key features like data collection, storage, IoT device control, and access control, among others. However, the adoption of smart city technologies introduces security and data management challenges. These challenges include non-streamlined data accumulation, data silos that hinder integration, sensor inaccuracies resulting in erroneous data, and the need for standardized semantic details to facilitate seamless data exchange among stakeholders [6].

To address the aforementioned challenges in smart cities, we propose a distributed multilayered data platform architecture for an interoperable IoT system. This architecture consists of four layers: the Data Monitoring Layer (DML), Data Enhancement Layer (DEnL), Data Storage Layer (DSL), and Data Exchange Layer (DEL). Each layer plays a crucial role in enabling efficient data processing, storage, and exchange, improving overall performance and scalability.

Our contributions to smart city interoperability architectures include proposing this distributed multilayered data platform architecture that addresses data management, security, and standardization challenges. We implemented the architecture using distributed instances of OM2M as the interoperability platform and conducted an in-depth analysis of key performance indicators such as insertion, retrieval, and latency. This evaluation provides insights into the effectiveness and efficiency of the architecture.

In the following sections, we provide a detailed description of each layer of the proposed architecture (Section III), discuss the implementation (Section IV), and explain the use of distributed instances of OM2M [7] as the interoperability platform. We analyze key performance indicators, including insertion, retrieval, and latency to provide insights into the architecture's effectiveness in Section V. The paper concludes with a discussion of the implications of the proposed architecture and outlines future research directions in Section VII.

This research was conducted on the Smart City Living Lab[1], a large-scale system deployed at IIIT Hyderabad. By studying this system, we were able to gather valuable real-world data allowing us to evaluate the effectiveness of our proposed architecture. The experiments conducted within the Smart City Living Lab demonstrated efficient results and highlighted the

*VJS Pranavasri and Leo Francis contributed equally to this work

[1]https://smartcitylivinglab.iiit.ac.in/

potential of our architecture in addressing the complexities of smart city deployments.

II. Related Works

Several works have made significant contributions to the field of data platform architecture for smart cities, addressing diverse aspects, challenges, and requirements. One notable work by Sethi and Sarangi [8] proposed a taxonomy for IoT technologies based on architectural elements, emphasizing scalability and standards compliance. Our architecture aligns with existing implementations [9] while incorporating these principles, supporting diverse industry scenarios and enforcing a distributed system. Ibrahim et al. [10] highlighted the significance of IoT architecture and associated challenges like security, interoperability, QoS, and scalability, some of which our architecture addresses, offering solutions to enhance scalability, interoperability, and resource management. Srdjan et al. [11] presented an architectural reference model (ARM), inspiring our architecture's adoption of a common ontology and promoting semantic interoperability. Guth, Jasmin et al. [12] compares different IoT Platform architectures. If we observe the data architecture of these various vendors they do not have a standard ontology, are not a distributed system, and are mainly a three-layer system which makes them less scalable and have a lower ability to handle load for city-wide deployments. Seungmyeong et al. [13] underscored the need for interoperable and maintainable smart city data platforms, aligning with our modular architecture design that supports data ingestion, core management, services, and analytics.

A Multi-Layer Data Platform Architecture for Smart Cities using oneM2M and IUDX [14] introduces a comprehensive architecture for smart city applications, incorporating a Data Monitoring Layer, Data Storage Layer, and Data Exchange Layer. It demonstrates interoperable data management through a proof of concept implementation and evaluates feasibility and sustainability. However, the architecture in the paper is not of a distributed nature and is not scalable. Building upon the foundational work of Mante et al. [14], our research advances this architecture with a distributed and scalable approach. We compare our final implementation against the works of this paper.

III. Proposed Architecture

Our proposed architecture consists of four key layers: Data Monitoring Layer (DML), Data Enhancement Layer (DEnL), Data Storage Layer (DSL), and Data Exchange Layer (DEL) as illustrated in Fig.1. The DML receives data from IoT nodes through a load balancer and forwards it to the distributed oneM2M instances. This Layer then forwards data to DEnL using the Common Service Function (CSF) of oneM2M. The DSL comprises a Historical Data Archive for older data and a temporary Data store for recent data, which are populated through the Extract, Transform, Load (ETL) process from the DEnL. The DEL includes an Authentication server, resource server, and REST APIs for secure data retrieval.

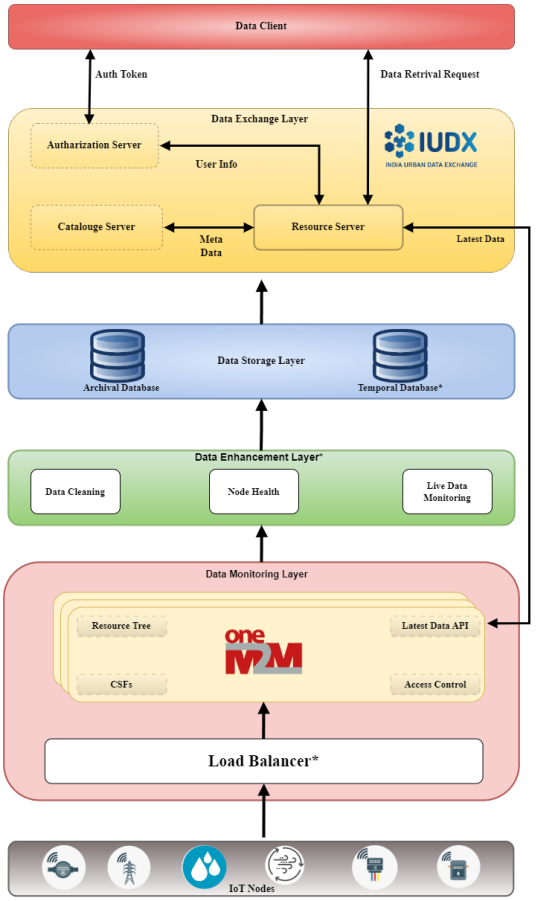

Fig. 1. Proposed architecture of the Distributed OM2M

A. Data Monitoring Layer (DML)

The DML acts as the entry point for data in the architecture. It receives data from a cluster of IoT nodes at predefined frequencies, with a load balancer efficiently distributing the incoming traffic among multiple oneM2M framework instances making it a distributed system. The oneM2M interoperability layer collects and stores data using the built-in Common Service Functions (CSF). This information is stored as a `<contentInstance>` resource within the relevant `<container>` resource. These resources can be controlled collectively through `<group>` resources. Application Entities (`<AE>`) represent specific sensor network categories which are the parent of Container resource. Data forwarding to the DSL occurs via `<Subscription>` resources using a publish-subscribe mechanism. Each oneM2M instance internally maintains a database for metadata storage, resource management, and sensor data.

B. Data Enhancement Layer (DEnL)

The DEnL receives data from the DML and performs various data enrichment operations. It is equipped with functionalities to process and transform the incoming data. This layer may involve data cleansing, normalization, aggregation,

or other operations to enhance the quality and value of the data. The DEnL layer shields the underlying components and technologies used by encapsulating the logic and operations required for data preprocessing and transformation. This layer eliminates the need for repetitive reformatting during data retrieval which will be required if this layer doesn't exist, Ultimately optimizing the overall data flow and facilitating seamless integration with downstream systems. The DEnL enables the system to handle diverse data sources and adapt to evolving requirements without tightly coupling it to specific technologies or components.

C. Data Storage Layer (DSL)

The DSL is a critical component that is responsible for securely storing and managing data obtained from the DEnL. The DSL adopts a multi-tenant architecture with a focus on efficient and reliable storage. It consists of two main components: a Historical Data Archive and a temporary Data Store. The Historical Data Archive serves as a long-term storage solution for older data, ensuring data retention and compliance with regulatory requirements. On the other hand, the temporary Data Store holds recent data, enabling quick and efficient access for applications. The DEnL forwards data to both the Historical Data Archive and the temporary Data Store in parallel, ensuring redundancy and availability.

D. Data Exchange Layer (DEL)

The DEL acts as a secure interface for data clients to access the stored data within the architecture. It comprises an Authentication server, a resource server, and REST APIs. The Authentication server handles user registration, access policy creation/deletion, and token generation for data retrieval which ensures that only authorized clients can access the data by verifying their credentials. The Resource server hosts the stored data and provides the necessary functionalities for data retrieval. REST APIs are employed to facilitate communication and enable data clients to query the stored data efficiently.

IV. Implementation

This section discusses the technology stack used to implement our proposed architecture. For our experiments, We investigated the Smart City Living Lab an existing large-scale system deployed within a 66-acre campus which comprises 291 nodes categorized into 10 groups known as verticals each with its own Application Entities (AE's). These verticals cover a wide range of functionalities, including water quality monitoring, air quality monitoring, and more. In our experiment, we simulated three verticals to emulate nodes sending requests to the data architecture. The architecture implementation can be seen in Figure 2. To handle the data from these nodes and to our data architecture, we deployed three similarly configured systems, each with three instances of IN-CSEs. These instances received data from a load balancer, ensuring efficient distribution of data across the system. To enhance data quality, we integrated Thingsboard for data cleaning. For seamless data streaming, we employed the Kafka Engine, which facilitated the transfer of data to Elasticsearch [15] and Postgres. Elasticsearch served as the temporary data archive, while Postgres acted as the historical data archive.

A. Data Monitoring Layer

In the Data Monitoring Layer, we employed a load balancer as an entry point from the sensor nodes. It distributes incoming sensor data and delivers it to distributed OM2M instances, i.e. distributed IN-CSEs. The distributed IN-CSEs are physically apart from each other. In this scenario, we used Round Robin (RR) and the least connection (LC) approach separately to input data into the dispersed OM2M IN-CSEs to evaluate its performance characteristics. As in the prior method, one MongoDB instance is utilised to accomplish the IN-CSE, resulting in data congestion, thus In the current design, each IN-CSE has its own MongoDB instance as its Database. We use the oneM2M `<subscription>` resource functionality to transfer data to the Data Enhancement Layer.

B. Data enhancement Layer

The Data Enhancement Layer seamlessly processes incoming data from various sources within the oneM2M framework using Thingsboard. It connects with the onem2m platform to receive real-time data in JSON format. The layer's main function is to extract and parse relevant telemetry information from the JSON payload, specifically focusing on the content information (`<con>`). Once extracted, the enhancement layer assigns appropriate parameters to the data to ensure compatibility with downstream layers and optimize data cleaning/preprocessing. The processed data is then seamlessly forwarded to the Data Storage Layer for transmission and storage. The design and implementation of the enhancement layer offer flexibility and extensibility, encapsulating device functionalities and rule chains to achieve a modular and scalable architecture. Fig 3 shows an example rule chain done in Thingsboard. The system's core functionalities include data ingestion, parsing, parameter assignment, and integration with the data storage layer, emphasizing the role of the enhancement layer.

C. Data Storage Layer

We continue the workflow from Thingsboard by sending formatted data to Kafka through a publish/subscribe mechanism. Kafka Connect API [16] is then leveraged to transmit this data to both Elasticsearch and PostgreSQL. Elasticsearch is our temporary data store, while PostgreSQL is our historical data archive. Elasticsearch is a state-of-the-art search store for faster data querying and searching and is one of the best options to use as a temporary database. It is one of the most popularly used data stores for efficient data ingestion and retrieval as suggested by Vandikas [17] and Thacker [18] in their research.

Thingsboard in the DEnL formats data before storing it in Elasticsearch, eliminating the need for redundant reformatting when retrieving data. For each similar node, there will be a topic created in Kafka and a respective table in Postgres and an index in Elasticsearch. A Kafka connector would be created

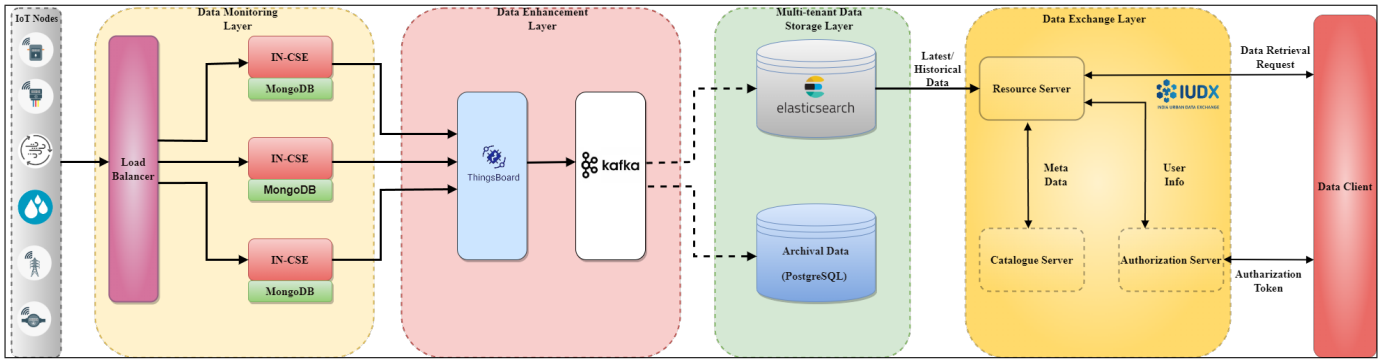

Fig. 2. Implementation of the Distributed OM2M

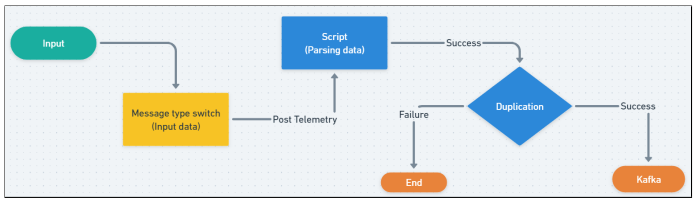

Fig. 3. Thingsboard Rule Chain

for each topic to ensure the data is being stored in both the data stores successfully. This provides an efficient and reliable data storage layer for our implementation. The code for our implementation is available on GitHub [2]

D. Data Exchange Layer

In our most recent implementation, we included a resource server with a set of services. An authorization service that enables the user to request a token for data access. The data flow within the implementation can be seen in Fig.2. The APIs fetch data from the DML using the Resource service that decides which data store to perform the query based on the data requested from the data client. This enables the framework to maintain data freshness when querying live data and also supports archival retrievals. Further, we have leveraged a Python service based on Flask API (Gunicorn as the gateway) to implement the data exchange layer with Nginx as the open-source web server.

V. Results

We performed experiments to evaluate the effectiveness and efficiency of the proposed architecture. We employed three identical workstations as infrastructure nodes with common service entities (in-cse). These machines featured an x86_64 architecture, 6 cores, an Intel i5 CPU, a minimum of 16GB RAM, and a 64-bit Ubuntu 18.04 operating system. Throughput and latency were measured to assess the architecture's performance under various workloads, including data insertion and retrieval.

[2]https://github.com/vjspranav/KafkaConnect

A. Improvements in the Data Monitoring Layer

In this experiment, our primary objective was to replicate real-time data entry and retrieval events in the Interoperability Layer. We conducted performance tests for data insertion and retrieval to the DML over a duration of 3 hours for insertion and 1 hour for retrieval. Throughout the tests, we systematically increased the number of simultaneous requests/users while capturing various parameters [3].

For data retrieval, we tested varying parallel user counts in steps of 25 and requested data points in steps of 200 (ranging from 200 to 600). The results showed that our distributed architecture could effectively handle **95** concurrent users using Round Robin Load balancing and **116** concurrent users using the Least Connection Load balancing, with 0% downtime.

The architecture exhibited a remarkable **222.70%** increase in overall throughput and an average **69.08%** decrease in latency compared to the existing centralized architecture[4] as seen in Table I. Fig 4 shows a detailed comparison of performance across different load balancers when compared with the centralised architecture for data retrieval.

In the data-insertion scenario, the proposed distributed architecture with LC load balancer effectively handled **51** concurrent devices, resulting in up to a **41.23%** improvement in throughput and an average **29.19%** decrease in latency compared to the centralized approach as shown in Table II. This improvement was achieved by utilizing the LC load balancer, which dynamically allocates requests based on server load, optimizing resource utilization and employing a distributed system.

TABLE I
Comparision of performance across different load balancers for retrieval

LoadBalancer	Throughput/s	Average Latency (ms)	Median Latency
None	36.11	81329.23	8062.00
RR	95.98	30571.24	185.00
LC	116.53	25143.10	804.50

[3]Detailed results are available at: https://vjspranav.github.io/om2mreports

[4]This performance evaluation focuses on the om2m retrieval in a distributed setting

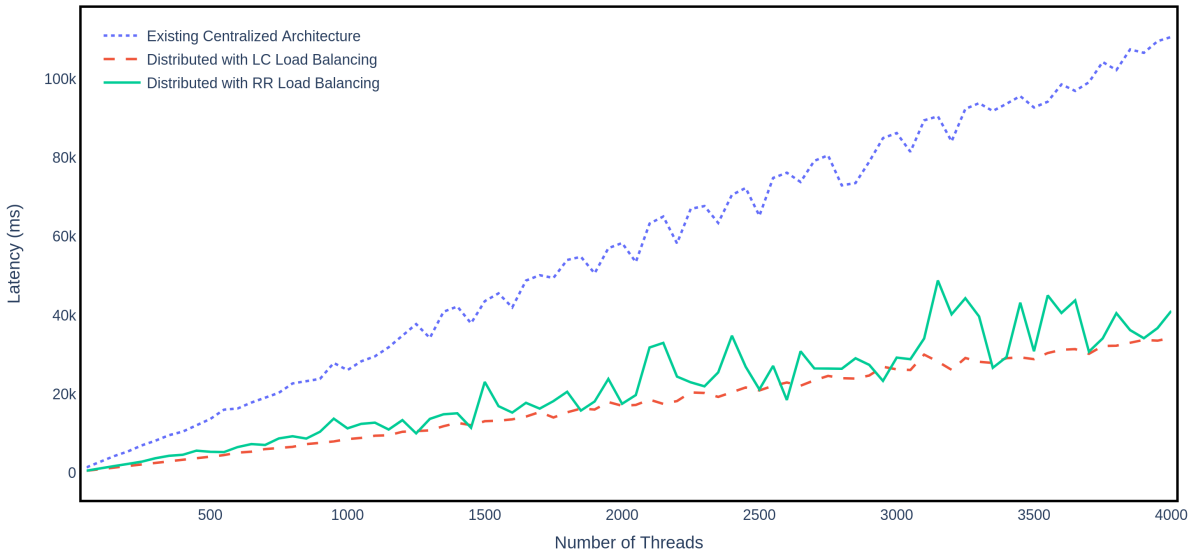

Fig. 4. OM2M retrieval performance characteristic

TABLE II
COMPARISION OF PERFORMANCE ACROSS DIFFERENT LOAD BALANCERS FOR INSERTIONS

LoadBalancer	Throughput/s	Average Latency (ms)	Median Latecy
None	36.45	2364.75	5185.50
RR	51.48	1674.28	47.00
LC	49.43	1742.75	924.00

TABLE III
COMPARISON OF INSERTION PERFORMANCE THROUGH RESTFUL API VS OUR DENL CONSISTING OF DATA PRE-PROCESSING AND PUBLISH SUBSCRIBE

Architecture	Throughput	Total transactions	Avg Latency
RESTful API	153.55	730731	147.93
Publish-Subscribe	176.94	842024	70.39

B. Improvements in Data Enhancement and Control Layer

To provide a comparative perspective, we considered a direct alternative approach of utilizing a RESTful API to obtain the data and subsequently push it to the respective databases. In this context, we implemented a fastAPI-based middleware service to directly push the data into PostgreSQL and Elasticsearch. However, a performance analysis revealed that this direct alternative, while functional, exhibited slightly inferior performance when compared to our proposed DEnL.

Our DEnL, which involves data preprocessing and streaming through a publish-subscribe architecture, consisting of ThingsBoard and Kafka, showcased notable performance improvements across various aspects. The results depicted in Table III affirm the enhanced performance achieved by our implementation while retaining all the additional functionality provided by the rule chain of ThingsBoard. Thus, our chosen approach not only ensures data integrity but also delivers improved performance compared to a conventional CRUD (Create, Read, Update, Delete) method.

C. Improvements in Data Exchange Layer

In the original implementation, the Data retrieval by the client happened directly through OM2M which sends the latest data through REST API. Now that we propose using a temporary Data Store as our source for data, we compare how beneficial retrieval through this temporary Data Store would be when compared to retrieving directly through OM2M. We implement a simple service that will act as the middleware allowing us to query the latest data and do a fair data retrieval comparison.

We compare retrieval through our implementation of this latest data middleware directly with retrieval in OM2M which allows us to showcase the real-life benefit of our new architecture. Retrieval through our new middleware gives a substantially high throughput. We see an increase of over 800% in both throughput and total number of requests. By utilizing ElasticSearch in our architecture, the query response time is significantly reduced, resulting in the observed substantial improvement. This comparison demonstrates the effectiveness of our architecture in overcoming performance limitations and achieving enhanced retrieval capabilities. The results can be seen in Table IV.

TABLE IV
COMPARISON OF RETRIEVAL PERFORMANCE FROM OM2M TO THE IMPLEMENTED MIDDLEWARE FOR THE LATEST DATA

Service	Throughput/s	Total transactions
OM2M	36.11	375556
Proposed middleware	332.36	3456209

VI. Discussion

A. Lessons Learned

The results presented in this paper demonstrate the enhanced performance and scalability achieved by our distributed architecture, as shown in the performance comparison tables and graphs (Section V). The specific technology stack utilized was chosen to ensure a fair comparison with an existing non-distributed architecture that faced challenges when the number of users increased [14]. The aim was to address these issues and provide a more scalable solution. The architecture showcased in this paper exhibits notable improvements in handling increased user load, as evidenced by the results obtained in insertion and retrieval operations as shown in Section V-A. The findings highlight the advantages of the proposed architecture in terms of improved performance and reduced load, even under conditions of increased user activity. We believe that this architecture can be improvised further by moving it into a micro-services-based architecture ensuring we have very few dependencies across multiple services provided by the oneM2M standard.

B. Threats to Validity

Threats to *internal validity* concern the selection of workload parameters and measuring performance metrics. To this end, real workloads of the system have been used. Further, the same system configuration was used for the different experiments to prevent bias arising from any background processes. Threats to *external validity* concern the generalizability of the study across different IoT systems. Although our approach has been applied to a specific smart city IoT system, it uses techniques that can be generalized to more complex systems. Moreover, while this experiment provides valuable insights into the performance metrics of different architectures, the system configurations may impact them. To this end, we ensured the same base configuration was used across different experiments to ensure consistency.

VII. Conclusion and Future Scope

This study describes an improved architecture for a distributed and scalable system based on the oneM2M smart city standards. We proposed a multi-layered architecture consisting of a Data Monitoring Layer (DML) which is distributed and oneM2M compliant, a Data Storage Layer (DSL) consisting of temporary and historical data archives, and a Data Enhancement Layer (DEnL) which formats data before storage and the Data Exchange Layer (DEL) which serves as a robust interface where the data clients can securely access the data. We have conducted the performance analysis and provided context on the increase in scalability and load-handling capability in the system.

One possible future direction would be to implement the architecture from this paper in different tech stacks and do a comparison. By exploring alternative technology stacks, researchers and practitioners can gain a deeper understanding of how different frameworks impact the performance and scalability of the proposed architecture. Also as mentioned in section VI a micro-service-based architecture based on the proposed architecture could lead to much higher performance improvements.

VIII. Acknowledgement

This study was funded in part by the Ministry of Electronics and Information Technology (MEITY) as part of the Smart City Living Lab initiative under grant number 3070665 (2020). We really appreciate the IUDX team's assistance in creating the Data Exchange Layer.

References

[1] B. Mohanty, *Strategic Investments Towards Resource Efficient Cities*. United Nations Environment Programme (UNEP), 01 2015, pp. 140–155.

[2] M. S. Camaren Pete, *Sustainable, Resource efficient cities - Making it happen!* United Nations Environment Programme, 2012.

[3] T. Nam and T. Pardo, "Conceptualizing smart city with dimensions of technology, people, and institutions," in *Value Co-Creation Practices in Smart City Ecosystem*, 06 2011, pp. 282–291.

[4] R. Giffinger, C. Fertner, H. Kramar, R. Kalasek, N. Milanović, and E. Meijers, *Smart cities - Ranking of European medium-sized cities*. Vienna University of Technology, 01 2007.

[5] N. D. K. R. Chukka, A. Arivumangai, S. Kumar, R. Subashchandrabose, Y. B. S. Reddy, L. Natrayan, and G. C. Debela, "Environmental impact and carbon footprint assessment of sustainable buildings: An experimental investigation," *Adsorption Science & Technology*, Mar 2022.

[6] L. Farhan, S. Shukur, A. E. Alissa, M. Alrweg, U. Raza, and R. Kharel, "A survey on the challenges and opportunities of the internet of things (iot)," 12 2017, pp. 1–5.

[7] M. Alaya, Y. Banouar, T. Monteil, C. Chassot, and K. Drira, "Om2m: Extensible etsi-compliant m2m service platform with self-configuration capability," *Procedia Computer Science*, vol. 32, pp. 1079–1086, 12 2014.

[8] P. Sethi and S. Sarangi, "Internet of things: Architectures, protocols, and applications," *Journal of Electrical and Computer Engineering*, vol. 2017, pp. 1–25, 01 2017.

[9] X. Liu, A. Heller, and P. S. Nielsen, "Citiesdata: a smart city data management framework," *Knowledge and Information Systems*, vol. 53, no. 3, pp. 699–722, Dec 2017. [Online]. Available: https://doi.org/10.1007/s10115-017-1051-3

[10] I. Yaqoob, E. Ahmed, I. Hashem, A. I. A. Ahmed, A. Gani, M. Imran, and M. Guizani, "Internet of things architecture: Recent advances, taxonomy, requirements, and open challenges," *IEEE Wireless Communications*, vol. 24, 06 2017.

[11] S. Krčo, B. Pokrić, and F. Carrez, "Designing iot architecture(s): A european perspective," in *2014 IEEE World Forum on Internet of Things (WF-IoT)*, 2014, pp. 79–84.

[12] J. Guth, U. Breitenbücher, M. Falkenthal, P. Fremantle, O. Kopp, F. Leymann, and L. Reinfurt, "A detailed analysis of iot platform architectures: Concepts, similarities, and differences," in *Internet of Everything*, 2018.

[13] S. Jeong, S. Kim, and J. Kim, "City data hub: Implementation of standard-based smart city data platform for interoperability," *Sensors*, vol. 20, no. 23, p. 7000, Dec 2020. [Online]. Available: http://dx.doi.org/10.3390/s20237000

[14] S. Mante, S. Vaddhiparthy, R. Muppala, D. Gangadharan, A. Hussain, and A. Vattem, "A multi layer data platform architecture for smart cities using onem2m and iudx," 06 2023.

[15] R. Kuc and M. Rogozinski, *Elasticsearch Server*, ser. Community experience distilled. Packt Publishing, 2013. [Online]. Available: https://books.google.co.in/books?id=PEFK3MuwBsIC

[16] "Kafka connect — confluent documentation." [Online]. Available: https://docs.confluent.io/platform/current/connect/index.html

[17] U. Thacker, M. Pandey, and S. Rautaray, *Review of Elasticsearch Performance Variating the Indexing Methods*, 01 2018, pp. 3–8.

[18] K. Vandikas and V. Tsiatsis, "Performance evaluation of an iot platform," in *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies*, 2014, pp. 141–146.