MSMIoT: An Efficient Microservice-based Middleware architecture for the Internet of Things

Tushar Champaneria[1†] , Ashwin Makwana[2††] and Sunil Jardosh[3†††]
tac@ldce.ac.in
[†]Research Scholar, [††]Associate Professor, U & P U. Patel Department of Computer Engineering, Chandubhai S. Patel Institute of Technology, Charotar University of Science and Technology (CHARUSAT), Changa, India.
[†††]Software Architect, Progress Software, Hyderabad, India

Abstract
The Internet of Things (IoT) is becoming prevalent in the most promising domains like smart cities, healthcare, industrial automation, etc. Hence, the scalable and reliable middleware architecture design is the prime need for the adoption of IoT. An essential task of middleware is to abstract the underlying complexity of hardware to the services by facilitating the developer to develop software and services. The service orientation middleware approach is promising for addressing the challenges of the middleware of IoT. Microservices are radically popular in implementing SOA with different perspectives and goals. In this paper, we present a MicroService-based Middleware architecture for IoT (MSMIoT) that is modular and provides an accessible interface to applications and addresses known challenges like heterogeneity, scalability, interoperability, reliability, availability, context awareness, security, transparency, and abstraction to applications. We evaluated our proposed architecture for different traffic rates. Finally, the results show that the proposed architecture outperforms the traditional approach by a gain of 10-15 % in the case of throughput and a decrease in latency by 15-20 %.

Keywords:
IoT middleware architecture, middleware challenges, service-oriented architecture, microservices

1. Introduction

IoT is envisioned as the future of the internet, where every real-world thing is connected and communicates. Various technologies like sensors, communication protocols, and cloud computing play a vital role in making IoT a reality. IoT has characteristics like heterogeneity of devices and connectivity, low resources, sudden interactions between entities, a massive number of devices, no fixed infrastructure, context awareness, smartness, geo-location awareness, and distributed environment [1][2]. However, due to ultra-large-scale connected devices, IoT has its own challenges. IoT infrastructure must have high availability, reliability and must scale massively, IoT devices, software, protocols, and apps must have standardization and interoperable. As in most scenarios of IoT devices moving from one place to another, dynamic management of devices, i.e., Mobility, is also critical. Other significant challenges are naming and identity management of devices and services, QoS parameters [3][4], resilience to faults, performance measurement of devices and services, security privacy of the user and his data, access control for users, trust management for data, data confidentiality, data integrity, data mining, knowledge creation, and big data, greening of IoT which means using minimum energy for IoT infrastructure, design of SOA for IoT, design of middleware [5-14].

Above mentioned characteristics and challenges raises the requirement for middleware that acts as a gluing layer between IoT devices and software applications. It facilitates the programmer to develop IoT solutions using APIs and abstracts the device layer. IoT middleware inherits the challenges of IoT like interoperability, scalability, abstraction provisioning, spontaneous interaction, Adhoc infrastructure, multiplicity, security & privacy, and context awareness. In addition, it includes the detection and necessary actions on some context management of data with big data technologies, trust management, mobility management, random topology, multiplicity, unknown data point availability, extensibility, and modularity [15][16][17].

For designing the middleware of IoT, there are specific key requirements that need to be addressed like scalability where heterogeneity needs to be handled at a massive scale, reliability in the case of fault occurrence, availability within given constraints and ability to recover from faults, inter- operability within various actors of IoT whether M2M or M2H, context awareness to take the right decision at the right time, security and privacy are essential and major requirements of IoT middleware which are vertical across all layers[1].

Manuscript received September 5, 2022
Manuscript revised September 20, 2022
https://doi.org/10.22937/IJCSNS.2022.22.9.90

The rest of this paper is organized as follows: In Section II, background and related works of IoT middleware architectures are presented. In Section III proposed architecture, with its implementation details discussed. Section IV evaluates the proposal. Finally, section IV concludes the work and highlights the future scope.

2. Background and Related Work

Middleware plays an essential role in implementing IoT solutions. There are various types of middleware proposed in the literature, like message-oriented middleware, context-aware middleware, robotics middleware, semantic middleware, and pub/sub middleware. Most of the middlewares in the literature target a different set of requirements. Among them, the service-oriented approach can address major key requirements which are discussed in the previous section. Service-oriented middleware is a widely accepted approach and is taken into consideration for reference.

2.1 SoA Middleware for IoT

To address the above middleware challenges, various approaches and classifications for middleware implementation are reviewed in this literature which include message-oriented middleware, semantic middleware, resource-oriented, context-aware, publish/subscribe, and service-oriented middleware (SOA) architectures [17][18]. SOA presents business processes and functionality as a service to the consumer. Service orientation helps to address the key challenges of the middleware like abstraction, modularity, interoperability, scalability, heterogeneity, security and privacy [19]. Service-oriented architecture can be implemented using SOAP-based web services and/or RESTFul web services approach. Middleware architecture for IoT mostly follows SOA [11][20], as shown in figure 1, which depicts a typical implementation of service-oriented architecture. From various studies, it has been well established that SOA middleware architectures are suitable for the IoT arena. [21][22][23][24].

As the majority of the web has become RESTFul, and studies have also shown that the RESTFul approach in service-oriented architecture helps in achieving easier integration, faster application development, less overhead in processing and communication, easy to use, scalability, programming smart environment, etc. [25][26][27][28][29].

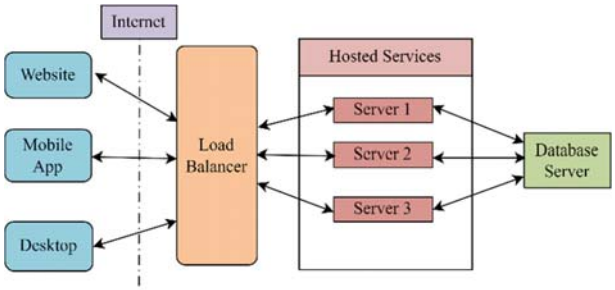


Fig. 1: Implementation of Service Orientation

2.2 Monolithic v/s Microservices SOA Approach for IoT

Currently, RESTful-based service-oriented middleware architectures are typically designed to be modular but packaged and deployed as a single application, i.e., a monolithic approach. Figure 2 shows that SOA-based implementation uses three-tier architecture. SOA, with the monolithic approach, negatively impacts reliability and scalability, especially in IoT smart city scenarios with added business functionalities and feature development. Due to the lack of interoperability, adopting the new technologies and frameworks in such monolithic architecture is not easy. Also, frequent deployment is cumbersome due to the increase in application size [30]. In addition to that, a monolithic approach typically has a larger team size where effective communication among teams might be an issue.

The solution to such drawbacks and challenges motivates the adoption of microservices in developing middleware for the IoT. It is a new paradigm for implementing a service-oriented architecture. A microservice is the process responsible for performing a single independent task that is typically built to perform a specific business capability. For example, given an application for recommendations, one microservice can be responsible for implementing a search feature while another microservice can be implemented to perform recommendations or ratings. Microservices are developed and exist independently, but ultimately, they are composed together to provide the overall functionality of an application. Each microservice has a well-defined interface or API that informs other microservices how they can be used and communicated. Microservices can be considered a

variation of SOA applied on an application scale rather than an enterprise scale.

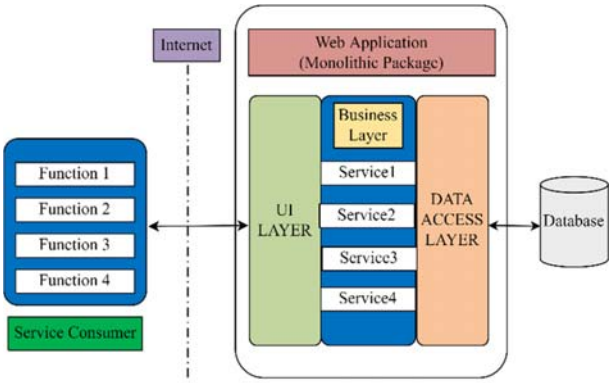


Fig. 2: SOA: Monolithic packaging

Microservices aid in addressing the middleware challenges like handling heterogeneity, interoperability, reliability, availability, scalability, and security. Microservices are an evolution of SOA over time, and it is SOA done well. It is an architectural pattern and can be considered as a particular case of service orientation, which helps realize service orientation in the real sense. Microservices have characteristics like functional decomposition, technology agnostic, and polyglot in the implementation.

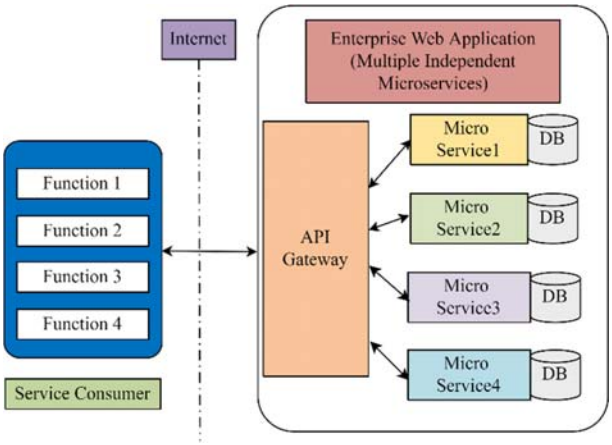


Fig. 3: SOA: Microservices Way

Microservices typically work by dividing the bigger application into smaller manageable applications, which interact with each other. As the application is divided into smaller parts, complexity can be handled effectively, and modularity is imposed indirectly. Development & maintenance of the smaller application is easy compared to a larger system. Individual services can be developed using different tools and technologies until they can communicate using endpoints that leverage reusability and interoperability. Microservices are reliable because for each mini-application, a dedicated team is responsible for each aspect such as development, build and test, etc. Microservices can be implemented in the same way as regular web services. It can run on the physical machine as well as on a virtual machine in the cloud; hence horizontal scalability can be achieved. The Updating of an application can be done by updating each microservices independently that ensure minimal downtime of an application. The communication in microservices is typically done using a message queue rather than direct access. Due to that, service functionality and data communication is decoupled which results in achieving scalability and more functionality can be added. Using microservices, security can be provided by authentication, access control policy, and encryption [30]. In a nutshell, microservice architecture is where the application is functionally decomposed into multiple, standalone, independently deployable, and scalable services. Figure 3 shows SOA implementation using microservice. Furthermore, the authors of [31] propose an IoT platform SEnviro Connect that uses microservice architecture and strives to provide stability and interoperability to minimize operating costs. The platform's capabilities are validated by using the smart farming use case. The above-discussed middleware design techniques focused on interoperability and stability, but scalability is compromised. Hence, we design scalable microservice-based middleware to overcome the limitations.

3. Proposed MSMIOT Architecture

This section describes the proposed architecture along with its components involved. Figure 5 shows the architecture which has three parts. i) Device layer ii) Middleware gateway iii) Cloud middleware. The device layer contains all the sensors and actuators communicating with the gateway. The middleware gateway communicates with cloud middleware, and cloud middleware hosts different microservices & supporting software. The above middleware architecture receives data from the device layer via the

middleware gateway. Various tools, technologies, and custom code are used to realize the design practically.

3.1 Device Layer

The device layer hosts the sensors & actuators pool that connects devices to a middleware gateway to send and receive the data. Sensors that can communicate directly can send the data to an IoT gateway. The sensors lacking a communication stack can be hooked to any computing board (e.g Raspberry Pi or ESP8266) and send data to the IoT gateway. To put architecture in action, data ingestion is required. Therefore, from the device layer, data is generated continuously and sent to the middleware gateway. For generating data, two different approaches are used. i) *Actual sensors:* Use two sensors of type Temperature and Humidity sensor, i.e., DHT22, one GAS sensor, i.e. MQ-2 and PIR sensor. All of the above sensors are attached to the nodemcu ESP8266 module. All nodemcu ESP8266 modules (hereafter referred to as devices) are connected to the router and configured to read the values from the respective sensor and send them to the middleware gateway. Devices send the data to the middleware gateway via the HTTP POST method. ii) *Simulated sensors:* Use custom code to simulate the sensor's data using bounded randomized data and to reflect the dynamic behavior of data. The custom code can be configured with the number of sensors, type of sensors, location, and frequency. The simulated sensors send the data to the middleware gateway using the HTTP POST method. Figure 4 shows an example data format sent by devices to the gateway. It consists of two types of sensor messages, namely device registration and keep-alive message and other with sensor details including following

```
Message: sensors,sensorId=S1 isAlive=True
//To send registration and keep alive message

Message:  sensorData,sid=S2,loc=IN-GUJ-W1
val=30
//To send sensors information.

Message:  simsensors,sid=S3,loc=IN-GUJ-W3
val=56
//To send data with simulated values..
```

Fig. 4: Data format used by devices

details: sid i.e., sensor id, i.e., location and val i.e., the value of sensors.

3.2 Middleware gateway

The middleware gateway is an extension of middleware that consists of various components. The middleware gateway is an integral part of the proposed middleware, and all the devices are connected via it. It runs on the Raspberry Pi 3B v1.2 single board computer. It performs various functions like registering devices that send data, checking whether the device is connected, receiving data from the devices, and providing local storage for data redundancy. In addition, it has a gateway manager component to communicate to cloud middleware for sending gateway health and receiving the command. The middleware gateway consists of a data aggregator, time-series database, cloud data broker, and gateway manager.

Each component performs a specific task, i.e., i) Data aggregator: It receives and merges data from different sources and prepares the data for sending to the middleware in the cloud. ii) Time-series database: The data aggregator sends data to a time-series database to store time-stamped raw data. Time-stamped data helps track the data coming from various devices from the device layer. iii) The cloud data broker reads the time-series data and sends it over to middleware services hosted in the cloud for further processing and communication with the cloud and sends data to the cloud using a message queue [32]. It routes the data to a particular predefined queue. iv) The Gateway Manager: It controls and manages all gateway components and communicates with the specific microservices in the middleware. The gateway manager sends and receives different information. It registers the gateway to the middleware in the cloud and keeps track of connected devices. It sends CPU and memory utilization to the middleware microservice. It is also responsible for routing actuation and control messages from microservices hosted in cloud middleware.

For the realization of the middleware gateway implementation following components plays an important role. Firstly, it contains an HTTP listener service, which listens on a port for data coming from the physical and simulated sensors and forwards it for storage.

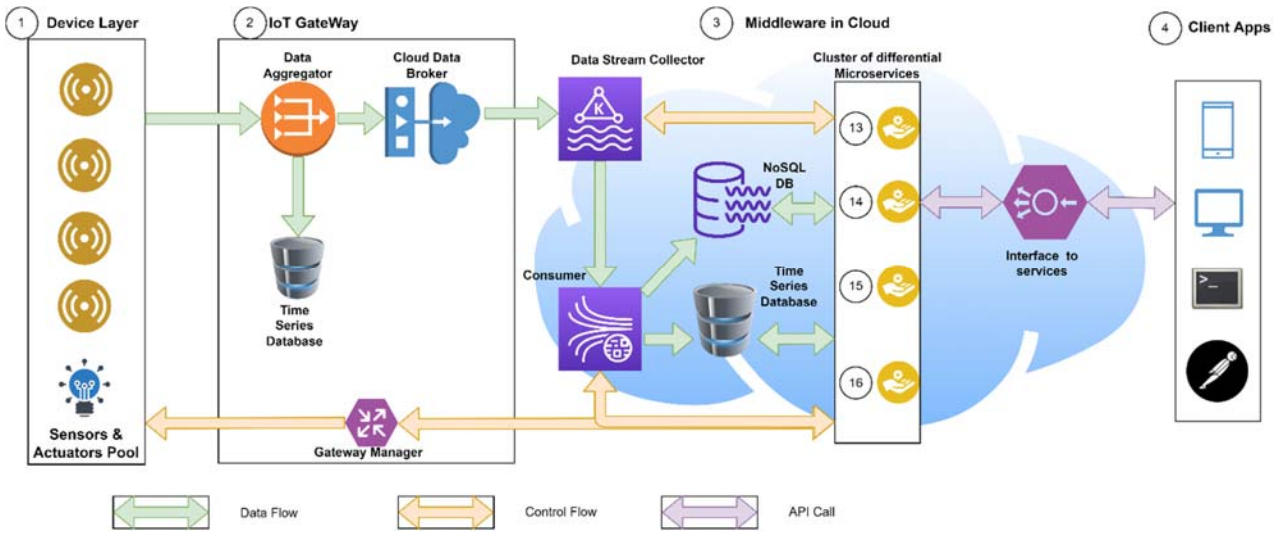


Figure 5: Proposed MSMIoT architecture

After receiving the data, it will be sent to the time-series database (e.g., InfluxDB [33]) for storage using line Protocol format as mentioned in figure 4. Also, data is sent to the message broker, which sends it to the server message queue (e.g., rabbitmq [34]) using AMQP. The Gateway manager service performs the following tasks. It keeps checking for any disconnected sensor by querying the sensor database. Its monitors the CPU and memory utilization of the gateway by using system API and sends it to the middleware server at a particular interval. It routes any actuation message coming from cloud middleware to the appropriate actuator. The gateway manager service first registers itself using its MAC address and ID with the middleware server. After that, it can communicate with the server's gateway management service using bi-directional WebSockets. While communicating with the server gateway uses JSON objects. An example of the structure of the JSON object is shown in figure 6.

```
‘msgType’:’sensorDetails’,
‘msg’:
{‘macaddress’: ‘56:9f:a3:0f:6f:32’,
  ‘gatewayid’  : ‘gateway’,
  ‘noofsensor’ : ‘4’,
  ‘sensorlist’   : [{' sid’:’101’,’stype’:’temperature'},
                   {' sid’:’102’,’stype’:’humidity'},
                   {' sid’:’103’,’stype’:’gas'},
                   {' sid’:’104’,’stype’:’motion'}]
 }
```

Fig. 6: Structure of JSON Object

The middleware gateway and server communicate based on the msgType for sending and receiving data and commands from the gateway to the server and vice versa.

3.3 Cloud middleware

Cloud middleware contains the components like stream collector, consumer, databases, and a set of differential microservices. As discussed in section II, the idea behind using microservices as the core middleware functionality is to utilize the advantages of microservices. It is the heart of the middleware that hosts a variety of functions. Cloud middleware runs on a VM hosted in Google Cloud Platform (GCP) running ubuntu 18.04 LTS server Operating system. It is essentially a set of components working together to achieve the goal of middleware. The following are the essential components: i) Stream collector hosts the message queue and receives the data from the middleware gateway in a particular message queue. It receives the sensorData sent over AMQP by one or more gateway(s). RabbitMQ server receiving messages at the exchange on a message queue. ii) Consumer: Messages the message queue receives must be consumed. It consumes the data and organizes it into two different databases, i.e., the time-series database & NoSql database. The time-series database captures the raw data. It is the main store of the

device's data received at middleware and stores Line Protocol. The consumer program monitors the message queue, extracts messages from it, and stores them in the time-series database (e.g., InfluxDB). The consumer also stores data gateway and sensors-related data in Nosql. iii) Differential Microservices: The next component in the cloud middleware is a set of differential microservices hosted in the cloud server. Various microservices are implemented to process the data. The following are examples of different microservices in the middleware. a) Authorization: It authorizes the service that wants to read and write the database in middleware. It authorizes other microservices/users to access the database. It checks user credentials and service rights to access the database and authenticate accordingly. b) DB access: Its facilities the client apps and acts as an intermediary for reading and writing from and to the database. All the available database operation is done through this microservice. It provides an interface to with database to other microservices. It exposes APIs to interact with the database and provide access. In the implementation, two databases are primarily used, i.e., NoSQL DB and time-series database, each with different purposes. NoSQL DB is used to store data related to the configuration of gateways. Sensors attached to a particular gateway and no of microservices and their functions. Time-series database, on the other end, stores the data of the sensor coming in from one or more gateway sources. c) Alert generation: It monitors and receives the alert from the database management tool, generates an appropriate message, and sends it to the gateway management service. It monitors alerts generated from the time-series database regarding anomalies in data like out-of-range data readings and continuously missing data and sends alerts to the telegram bot. It generates the required actuation message and sends it to an appropriate actuation service for sending the action back to the gateway to devices. d) Actuation function: When any actuation message is received, it routes that message to the appropriate gateway, which sends the message to the actuator for the corresponding action. e) Gateway management functions: Gateway management microservices are a set of microservices used for communicating with one or more gateways. It is essential for communicating with the sensor devices to and fro. Gateway management functions include receiving the CPU and memory utilization of the gateway, the number of sensors connected, the disconnected sensor information, the alert from the alert service, and sending alerts and commands gateway.

Moreover, custom microservices that are responsible for data and processing can be implemented and hosted in this layer to the above-listed services. Using microservices in the middleware helps to overcome challenges like scalability, heterogeneity, interoperability and high availability by duplicating microservices using containerization. Microservice operates on data and finds the context of data to make context-aware decisions. Data management functions can be implemented at this layer which helps in ensuring fault tolerance.

4. Results

This section covers the performance evaluation of the MSMIoT. Both deployment and testing of the MSMIoT are done on the google cloud platform to evaluate its performance [35]. Table 1 depicts the specifications and experimentation parameters for the deployment and testing of the proposed architecture. On the server-side, microservices and related software are deployed, and for testing, the JMeter software tool is used [36]. The test computer has been used to generate random data and simulate IoT devices as publishers and users to retrieve the observations using the web app. Experimentation aims to explore and evaluate the IoT platform from the scalability aspects. We have compared our proposed approach with SEnviro Connect architecture [31].

Table 1: Specification of environment used for performance analysis.

Specification	Deployment Mechine	Test Machine
GCP Instance	e2-standard-8	e2-standard-4
CPU	8x Intel(R) Xeon(R) CPU V4@2.20 GHz	4x Intel(R) Xeon(R) CPU V4@2.20 GHz
Memory	16 GB	16 GB
OS	Ubuntu 20.04.4 LTS	Microsoft Windows Server 2012 R2 Datacenter
Software	Microservices, influxdb, mongoDB, grafana server	Jmeter

4.1 Performance Metrics

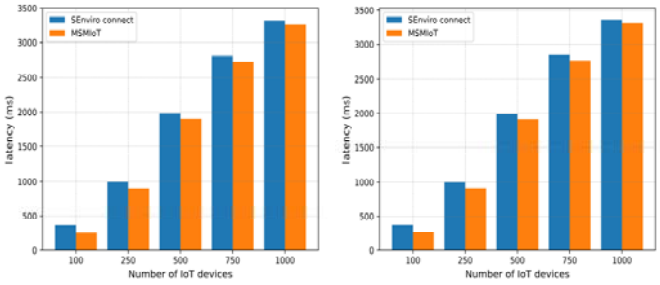
To assist time-critical applications, the IoT platform should support the management of thousands of IoT devices, simultaneously publishing observations with as little lag as possible. Performance metrics such as throughput and response time can be

used as reliable indicators of how scalability is preserved.

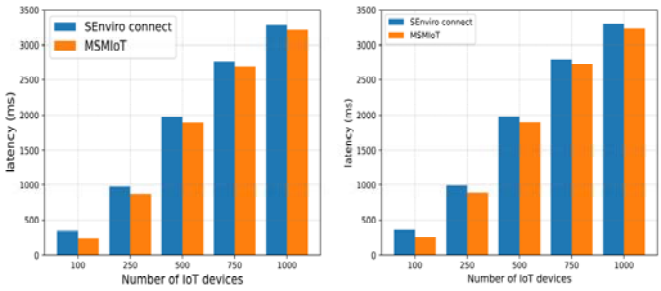
Throughput: It indicates the number of messages received and processed per second by the IoT platform. A higher throughput indicates the efficacy of the proposed architecture.

Latency: It refers to the delay incurred in processing the requests. The result shows that the proposed platform processes the requests and responses in a minimal time.

Figure 7 shows latency analysis of MSMIoT where the x-axis shows the number of IoT devices and the y-axis shows the latency in mili-seconds. It shows the impact of increasing the number of IoT devices operating at varying rates of message traffic, i.e., λ=5,10,15,20. It is observed that with the increase in the number of IoT devices, latency remains low for a lesser number of IoT devices and increases as IoT devices number increase. However, Still, MSMIoT outperforms by approximately 5-10%.



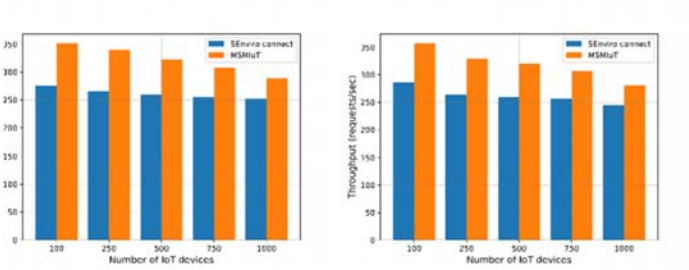
(a) (b)



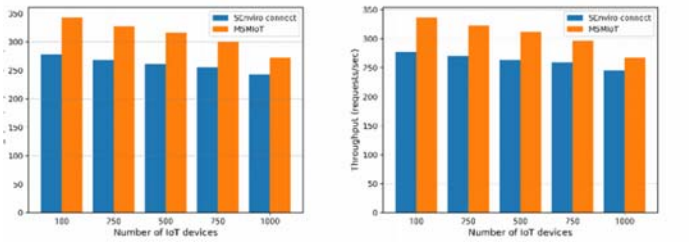
(c) (d)

Figure 7: Impact of increasing number of IoT devices on latency for rate a) λ = 5, b) λ = 10, c) λ = 15, d) λ = 20

Figure 8 shows the throughput analysis of MSMIoT where the x-axis shows the number of IoT devices and the y-axis shows throughput in terms of messages per second processed. It shows the impact of increasing the number of IoT devices operating at varying rates of message traffic i.e., λ=5,10,15,20. It is observed that the increase in the number of IoT devices throughput is higher for less number of IoT devices and decreases uniformly. Concerning throughput also, MSMIoT outperforms the traditional approach.



(a) (b)



(c) (d)

Figure 8: Impact of increasing number of IoT devices on throughput for rate a) λ = 5, b) λ = 10, c) λ = 15, d) λ = 20

4. Conclusion

This paper proposed a generic, scalable, and reliable microservice-based middleware architecture. First, we compared our approach with SEnviro connect approach. Next, we discussed detailing the components involved in the proposed middleware MSMIoT. The various use cases of the MSMIoT range from domains like healthcare, smart cities, smart homes, etc. The performance evaluation results show the efficacy of the proposed architecture. The result shows an improvement in throughput and latency by almost 10-15%. The future direction motivates for incorporating the data management functionality in the MSMIoT framework.

References

[1] Mohammad Abdur Razzaque et al. "Middleware for internet of things: A survey". In: IEEE Internet of Things Journal 3.1 (2016), pp. 70–95. ISSN: 23274662. DOI: 10.1109/JIOT.2015.2498900.

[2] Ruhul Amin et al. "A light weight authentication proto- col for IoT-enabled devices in distributed Cloud Computing environment". In: Future Generation Computer Systems 78 (2018), pp. 1005–1019.

[3] Selma Dilek et al. "QoS-aware IoT networks and protocols: A comprehensive survey". In: International Journal of Communication Systems 35.10 (2022), e5156. DOI: https://doi.org / 10. 1002 / dac. 5156. eprint: https://onlinelibrary. wiley. com / doi / pdf / 10. 1002 / dac. 5156. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.5156.

[4] Malaram Kumhar and Jitendra Bhatia. "Emerging communication technologies for 5G-Enabled internet of things applications". In: Blockchain for 5G-Enabled IoT. Springer, 2021, pp. 133–158.

[5] Andrew Whitmore, Anurag Agarwal, and Li Da Xu. "The Internet of Things???A survey of topics and trends". In: Information Systems Frontiers 17.2 (2015), pp. 261–274. ISSN: 13873326. DOI: 10. 1007 / s10796 -014-9489-2.

[6] Eleonora Borgia. "The Internet of Things vision: Key features, applications and open issues". In: Computer Communications 54 (2014), pp. 1–31. ISSN: 0140-3664. DOI: 10.1016/j.comcom.2014.09.008. URL: http://dx.doi.org/10.1016/j.comcom.2014.09.008.

[7] Shancang Li, Li Da Xu, and Shanshan Zhao. "The internet of things: a survey". In: Information systems frontiers 17.2 (2015), pp. 243–259.

[8] In Lee and Kyoochun Lee. "The Internet of Things (IoT): Applications, investments, and challenges for enterprises". In: Business Horizons 58.4 (2015), pp. 431–440. ISSN: 00076813. DOI: 10.1016/j.bushor.2015.03.008.URL:http://dx.doi.org/10.1016/j.bushor.2015.03.008.

[9] Louis Coetzee and Johan Eksteen. "The Internet of Things-promise for the future? An introduction". In: 2011 IST-Africa Conference Proceedings. IEEE. 2011, pp. 1–9.

[10] John A. Stankovic. "Research directions for the internet of things". In: IEEE Internet of Things Journal 1.1 (2014), pp. 3–9. ISSN: 23274662. DOI: 10.1109/JIOT. 2014.2312291.

[11] Daniele Miorandi et al. "Internet of things: Vision, applications and research challenges". In: Ad Hoc Net- works 10.7 (2012), pp. 1497–1516. ISSN: 15708705. DOI: 10. 1016 / j. adhoc. 2012. 02. 016. URL: http: / / dx. doi.org/10.1016/j.adhoc.2012.02.016.

[12] Debasis Bandyopadhyay and Jaydip Sen. "Internet of things: Applications and challenges in technology and standardization". In: Wireless Personal Communications 58.1 (2011), pp. 49–69. ISSN: 09296212. DOI: 10.1007/s11277-011-0288-5. arXiv: 1105.1693.

[13] Ala Al-Fuqaha et al. "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications". In: IEEE Communications Surveys and Tutorials 17.4 (2015), pp. 2347–2376. ISSN: 1553877X. DOI: 10.1109/ COMST.2015.2444095.

[14] Rafiullah Khan et al. "Future internet: the internet of things architecture, possible applications and key challenges". In: 2012 10th international conference on frontiers of information technology. IEEE. 2012, pp. 257–260.

[15] Moumena A. Chaqfeh and Nader Mohamed. "Challenges in middleware solutions for the internet of things". In: Proceedings of the 2012 International Conference on Collaboration Technologies and Systems, CTS 2012 (2012), pp. 21–26. DOI: 10.1109/CTS.2012.6261022.

[16] Soma Bandyopadhyay et al. "A survey of middleware for Internet of things". In: Communications in Computer and Information Science 162 CCIS (2011), pp. 288–296. ISSN: 18650929. DOI: 10.1007/978-3-642-21937-5 27.

[17] Ghofrane Fersi. "Middleware for internet of things: A study". In: Proceedings - IEEE International Conference on Distributed Computing in Sensor Systems, DCOSS 2015 (2015), pp. 230–235. DOI: 10. 1109 / DCOSS.2015.43.

[18] Zhen Peng, Jingling Zhao, and Liao Qing. "Message oriented middleware data processing model in Internet of things". In: Proceedings of 2nd International Conference on Computer Science and Network Technology, ICCSNT 2012 (2012), pp. 94–97. DOI: 10. 1109 / ICCSNT.2012.6525898.

[19] Chayan Sarkar et al. "DIAT: A scalable distributed architecture for IoT". In: IEEE Internet of Things Journal 2.3 (2015), pp. 230–239. ISSN: 23274662. DOI: 10.1109/ JIOT.2014.2387155.

[20] Luigi Atzori, Antonio Iera, and Giacomo Morabito. "The Internet of Things: A survey". In: Computer Net- works 54.15 (2010), pp. 2787–2805. ISSN: 13891286. DOI: 10.1016/j.comnet.2010.05.010. URL: http://dx.doi. org/10.1016/j.comnet.2010.05.010.

[21] Patrik Spiess et al. "Soa-based integration of the internet of things in enterprise services". In: 2009 IEEE International Conference on Web Services, ICWS 2009 (2009), pp. 968–975. DOI: 10.1109/ICWS.2009.98.

[22] Dominique Guinard et al. "Interacting with the SOA- based internet of things: Discovery, query, selection, and on-demand provisioning of web services". In: IEEE Transactions on Services Computing 3.3 (2010), pp. 223–235. ISSN: 19391374. DOI: 10.1109/TSC.2010.3.

[23] Vale'rie Issarny et al. "Service-oriented middleware for the Future Internet: State of the art and research directions". In: Journal of Internet Services and Applications 2.1 (2011), pp. 23–45. ISSN: 18690238. DOI: 10.1007/s13174-011-0021-3.

[24] Jameela Al-Jaroodi and Nader Mohamed. "Service- oriented middleware: A survey". In: Journal of Network and Computer Applications 35.1 (2012), pp. 211–220. ISSN: 10848045. DOI: 10.1016/j.jnca.2011.07.013.URL:http://dx.doi.org/10.1016/j.jnca.2011.07.013.

[25] Dominique Guinard, Iulia Ion, and Simon Mayer. "In search of an internet of things service architecture: REST or WS-*? A developers' perspective". In: Lecture Notes of the Institute for Computer Sciences, Social- Informatics and Telecommunications Engineering 104 LNICST (2012), pp. 326–337. ISSN: 18678211. DOI:10.1007/978-3-642-30973-1 32.

[26] Yuchen Yang et al. "A Survey on Security and privacy issues in Internet-of-Things". In: 4662.c (2017), pp. 1–10. DOI: 10.1109/JIOT.2017.2694844.

[27] Eric J Bruno et al. "The Intelligent Internet of Things with Axeda and Oracle Java Embedded". In: June (2014).

[28] Javier Poncela et al. "Smart cities via data aggregation". In: Wireless Personal Communications 76.2 (2014), pp. 149–168. ISSN: 09296212. DOI: 10. 1007 / s11277 -014-1683-5.

[29] Federica Paganelli, Stefano Turchi, and Dino Giuli. "A Web of Things Framework for RESTful Applications and Its Experimentation in a Smart City". In: IEEE Systems Journal 10.4

(2016), pp. 1412–1423. ISSN: 19379234. DOI: 10.1109/JSYST.2014.2354835.
[30] Long Sun, Yan Li, and Raheel Ahmed Memon. "Sun2017". In: (2016), pp. 154–162.
[31] Sergio Trilles, Alberto Gonza´lez-Pe´rez, and Joaqu´ın Huerta. "An IoT platform based on microservices and serverless paradigms for smart farming purposes". In: Sensors (Switzerland) 20.8 (2020). ISSN: 14248220. DOI: 10.3390/s20082418.
[32] Khushi Shah, Preet Modi, and Jitendra Bhatia. "Data Processing and Analytics in FC for Healthcare 4.0". In: Fog Computing for Healthcare 4.0 Environments. Springer, 2021, pp. 131–154.
[33] InfluxDb. URL: https://www.influxdata.com/ (visited on 05/31/2022).
[34] RabbitMQ. URL: https://www.rabbitmq.com/ (visited on 05/31/2022).
[35] Google Cloud Platform. URL: https://cloud.google.com/gcp (visited on 05/31/2022).
[36] Apache Jmeter URL: https://jmeter.apache.org/ (visited on 05/31/2022).

Tushar Champaneria has more than ten years of academic experience and two years of industrial experience. Since 2011, he has been working as an Assistant Professor in the Department of Computer Engineering at L.D.College of Engineering Ahmedabad, Gujarat. He received his B.E degree in Computer Engineering from C.U.Shah College of Engineering and Technology, Saurashtra University, in 2006, and his M.E degree in Computer Engineering from the BITS Pilani, Pilani campus, in 2008. He is currently pursuing a Ph.D. degree in computer science at Charotar University of Science and Technology (CHARUSAT). His research interests include the Internet of Things, software design, and Natural language processing. He has published several papers in refereed journals and conferences in these fields. He is a member of ACM, ISTE, and CSI.

.

Dr. Ashwin Makwana has more than eighteen years of academic and research experience. He is currently working as an Associate Professor at the Charotar University Of Science and Technology (CHARUSAT). He completed his Ph.D. in Computer Engineering from Charotar University Of Science and Technology (CHARUSAT) in 2020. He did his master of engineering from Dharamsinh Desai Institute of Technology in 2004 and a bachelor of engineering from Gujarat University in 2002. He has worked in different roles at Charotar University of Science and Technology (CHARUSAT) during his academic career. His research interests include Semantic Web, AI, Machine Learning, Deep Learning, and NLP. His research works are recognized by refereed journals and conferences in these fields.

Dr. Sunil Jardosh has more than ten years of industrial research experience and three years of academic experience. He is currently working as a Software Architect at Progress Software Hyderabad. He completed his Ph.D. in Information and Communication Technology at Dhirubhai Ambani Institute of Information and Communication Technology in 2011. He did his master of engineering from Dharamsinh Desai Institute of Technology in 2004 and a bachelor of engineering from L. D. College of Engineering, Gujarat University, in 2002. Before joining Progress Software Hyderabad, he worked as an assistant professor in an academic institution and as a research intern at the institute of plasma research. His research interests include Databases, Networks, WSN, Architecture Design, Middleware Design, and the Internet of Things. His research works are recognized by refereed journals and conferences in these fields.