Journal of Advanced Research in Applied Sciences and Engineering Technology

Journal homepage:
https://semarakilmu.com.my/journals/index.php/applied_sciences_eng_tech/index
ISSN: 2462-1943

Empowering Industry 4.0: Revolutionizing Data Exchange with Reactive Microservices-based Data Connector

Sze-Kai Gan[1,*], Thein-Lai Wong[1], Ching-Pang Goh[1], Dani-Eka Saputra[2]

[1] Faculty of Computing and Information Technology, Tunku Abdul Rahman University of Management and Technology, Kuala Lumpur, 53300, Malaysia
[2] Computer Science Department-BINUS University, Indonesia

ARTICLE INFO

Keywords:
Industry 4.0; Interoperability; Microservices architecture; Reactive programming; Connector development; Real-time data integration

ABSTRACT

In the rapidly evolving landscape of Industry 4.0, achieving seamless interoperability among diverse systems and technologies is paramount for organizations seeking to enhance operational efficiency and maintain competitiveness. This paper presents a ground-breaking solution, the reactive microservices-based connector, designed to address the intricate challenges of data exchange and communication within Industry 4.0 environments. By leveraging the principles of microservices architecture and reactive programming, the connector offers a scalable, resilient, and agile framework for facilitating real-time data integration across heterogeneous systems. Drawing upon a comprehensive literature review, this paper establishes a theoretical foundation by examining key concepts of Industry 4.0 and microservices architecture. Methodologically, the research adopts a Design Science Research approach, guiding the systematic development, implementation, and evaluation of the connector. Detailed insights into the design and implementation aspects of the connector, including its architecture, data processing pipeline, and integration with Akka Cluster and Akka Streams, are provided. Testing results offer valuable insights into the capabilities and limitations of the connector, particularly emphasizing observations from testing in virtualized environments. The paper concludes by highlighting the significance of the proposed approach in advancing interoperability efforts within the Industry 4.0 landscape and offering recommendations for future research directions.

1. Introduction

In the landscape of Industry 4.0, where digital transformation is reshaping manufacturing processes, achieving seamless interoperability among diverse systems remains a paramount challenge. The proliferation of IoT devices, sensors, and automation systems has led to an exponential increase in data generation, necessitating robust solutions for effective data exchange and communication across industrial ecosystems. Horizontal data sharing has emerged as a critical aspect of Industry 4.0, where various systems and stakeholders need to exchange data seamlessly to optimize processes and drive innovation.

* *Corresponding author.*
E-mail address: gansk-wr21@student.tarc.edu.my

https://doi.org/10.37934/araset.63.1.120142

To address these challenges, the development of innovative technologies is crucial to empower organizations with the capability to optimize operational efficiency and maintain competitiveness in the digital era. A key component in enabling horizontal data sharing is the implementation of connectors that facilitate the seamless exchange of data between disparate systems. These connectors play a vital role in orchestrating real-time data exchange and ensuring interoperability across heterogeneous systems.

This paper introduces a pioneering solution designed to address the complexities associated with data integration and interoperability in Industry 4.0 environments. The reactive microservices-based data connector is specifically developed to enable horizontal data sharing and streamline data exchange processes across industrial ecosystems. By leveraging the principles of microservices architecture and reactive programming, the connector offers a flexible and scalable framework for orchestrating real-time data exchange between diverse systems.

The significance of this research lies in its potential to revolutionize the approach to data interoperability challenges in Industry 4.0. With a focus on data integration, real-time data exchange orchestration, and data interoperability, the reactive microservices-based data connector aims to address the pressing need for seamless data exchange and communication in modern manufacturing environments. Through its innovative design and capabilities, the connector seeks to enhance operational efficiency, drive innovation, and ensure competitiveness in the digital era.

Aligned with the challenges identified in transitioning from Industrial Revolution 3.0 (IR3) to Industry 4.0, this research sets out to explore key objectives aimed at addressing these challenges. The objectives of the research are as follows:

i) Flexibility: The research aims to ensure that all components of the data connector are decoupled and autonomous, allowing the system to adapt to different manufacturing environments and connect with various modern protocols and applications.
ii) Scalability: Another crucial objective is to achieve high availability and scalability within the data connector. This entails implementing robust error handling, recovery mechanisms, and failover strategies to mitigate the risk of data loss or service interruption, particularly during peak operational hours.
iii) Enhanced Data Processing Efficiency: A key focus area is to enhance data processing efficiency within the data connector. By optimizing data processing pipelines and resource allocation strategies, the system aims to facilitate the seamless handling of high-volume data streams without overburdening computational resources.

Through the pursuit of these objectives, the research endeavors to develop a reliable and flexible data connector capable of meeting the demands of Industry 4.0 and addressing the identified challenges in data interoperability and integration.

2. Literature Review

2.1 Industry 4.0 and the International Data Spaces Reference Architecture Model (IDS-RAM)

The fourth industrial revolution, commonly referred to as Industry 4.0, emerged in Germany in 2011 as the next evolutionary step building upon the previous three industrial revolutions. At its core, Industry 4.0 is characterized by the seamless integration of cyber-physical systems (CPS)[1], the Internet of Things (IoT), and cloud computing technologies.

A pivotal framework for realizing Industry 4.0 initiatives is the Reference Architecture Model Industrie 4.0 (RAMI 4.0)[2], developed by the German Electrical and Electronic Manufacturers'

Association (ZVEI). This comprehensive three-dimensional model encompasses Hierarchy Levels, Life Cycle Value Stream, and Cross-Cutting Aspects, providing a structured approach to implement Industry 4.0 functionalities within factories and manage product lifecycles[3]. RAMI 4.0 facilitates communication between participants and workpieces through IoT or cloud technologies, making it a valuable tool for businesses adopting Industry 4.0. However, challenges remain, such as achieving interoperability and standardization across different systems and components. Efforts like "Constructing a Real-Time Value-Chain Integration Architecture for Mass Individualized Juice Production" [4], built on RAMI 4.0, demonstrate the potential for further evolution towards microservices architectures[5] .

In today's data-driven landscape, ensuring data sovereignty and interoperability is crucial. The International Data Spaces (IDS) initiative [6] aims to establish a secure and reliable data exchange infrastructure, facilitating data interoperability between organizations. IDS is based on creating secure data spaces for efficient data exchange, comprising four key layers: application, transport, data, and governance. The IDS architecture is technology-neutral, enabling organizations to implement IDS-compliant solutions using their preferred technologies while providing data sovereignty and interoperability.

A key component of both IDS-RAM 3.0 and 4.0 is the IDS Connector, serving as a standardized interface for communication between Industry 4.0 system components. In IDS-RAM 3.0, the connector is based on the OPC UA protocol, providing a uniform data model to enable data and information exchange. However, IDS-RAM 4.0[7] further enhances the connector with additional features like cross-domain communication, enabling secure and efficient interaction between different Industry 4.0 systems, while still leveraging the OPC UA protocol with Industry 4.0-specific extensions. Fig. **1** illustrates the structure of the IDS connector as described in IDS-RAM 4.0.

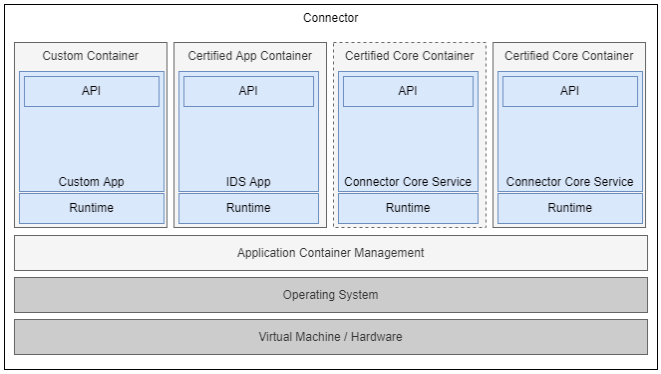

Fig. 1 IDS Connector in IDS-RAM 4.0

2.2 Contrasting Architectural Paradigms: Monolithic, Microservices, SOA, and Reactive Microservices

The discussion on the pivotal role of the IDS Connector in enabling standardized communication within Industry 4.0 systems underscores the importance of selecting the appropriate architectural paradigm for its implementation. A key consideration arises: whether the IDS Connector should be realized within a monolithic, microservices[8], or a more advanced reactive[9] microservices architecture.

Monolithic architecture is a traditional approach characterized by a single, tightly-coupled application where all components are bundled together. While this architecture simplifies development and testing, it can become challenging to scale and maintain as the application grows in size and complexity. In contrast, microservices architecture is a modern paradigm that comprises a suite of small, independent services communicating via lightweight protocols. Microservices

architecture is a variant of Service-Oriented Architecture (SOA)[10], which is built from a collection of services. Each microservice contains minimal functions and runs autonomously on its own process, enabling greater flexibility, scalability, and resilience compared to monolithic architectures.

Reactive microservices further enhance standard microservices by providing more isolation and autonomy through their consensus protocol and asynchronous non-blocking messaging architecture [11]. According to Boner [12,13], for a system to be considered reactive, it needs to comply with a set of principles to build applications meeting the requirements of responsiveness, even in the occurrence of failures and high demand. These principles stipulate that the system design must be responsive, resilient, elastic, and message-driven. Isolation is a prerequisite for resilience and elasticity, requiring asynchronous communication between services to decouple them in terms of time (allowing concurrency), space (allowing distribution and mobility), failure (allowing failure handling without cascading effects), and state (allowing each microservice to take sole responsibility for its own state and persistence).

Microservices that adhere to these principles are termed reactive microservices. Reactive microservices must adapt to the availability or unavailability of surrounding services, handling failures, acting independently, and cooperating with other microservices as required. A reactive microservice uses asynchronous messaging to interact with its peers and must implement recovery or compensation strategies whenever a failure occurs.

Recent studies [14] have explored utilizing reactive and asynchronous programming paradigms to improve service availability in distributed IIoT networks, although further research is needed to address timely data exchange requirements in manufacturing environments.

A key advantage of microservices and reactive microservices architectures is its ability to provide better fault isolation. If one service fails, it does not impact the entire application, unlike in monolithic architectures where a single module's failure can cause a system-wide crash. Furthermore, the microservices approach enables greater agility, allowing organizations to release new features and functionality at a faster pace. However, this architecture also presents challenges, such as increased complexity, higher operational costs, and the need for a robust DevOps culture to effectively manage and operate the independent services.

Recent research works, such as [15-17] have compared monolithic and microservices architectures. These studies have reported that microservices offer better scalability, fault isolation, and agility, while acknowledging the increased complexity and the need for specialized expertise and robust DevOps practices to effectively manage microservices architectures.

2.3 Clustering and Consensus Protocols for Enhancing Microservices Scalability and Availability

Microservices architecture offers scalability benefits through horizontal scaling, allowing individual services to be scaled independently. However, the scalability potential of microservices can be constrained by blocking code, which limits concurrency. Gunther's Law, also known as the Universal Scalability Law shown in Fig. **2**, highlights how blocking code can hinder system scalability by reducing concurrency. External factors such as network latency and hardware resources further impact microservices scalability.

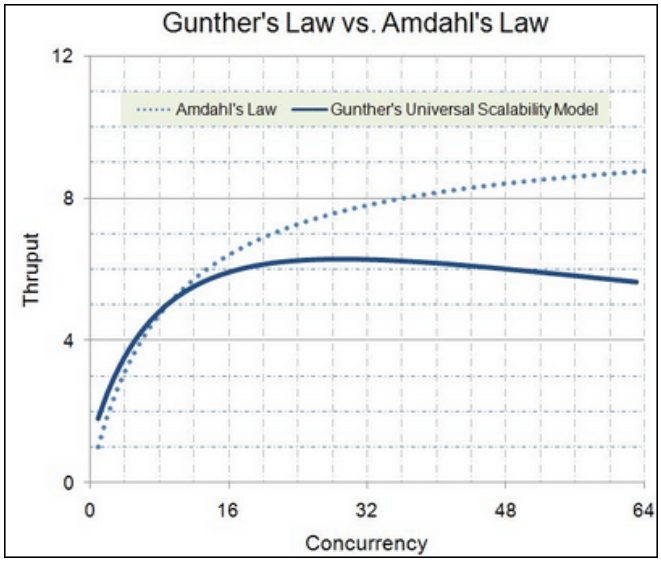

Fig. 2 Amdahl's Law and Gunther's Law[18]

Communication between microservices typically occurs via TCP or UDP protocols, which are constrained by hardware bandwidth and resources. To enhance scalability, it's essential to minimize the blocking fraction of the program and adopt reactive architecture principles. Selecting the appropriate clustering management model is also crucial for achieving scalability and availability.

Clustering management models can be centralized or decentralized. Centralized models rely on centralized services for node management, while decentralized models use consensus protocols within the application. Consensus protocols ensure agreement among multiple servers, facilitating fault-tolerant distributed systems. Fig. **3** illustrate how the cauterized service manage the metadata of the members in the cluster.

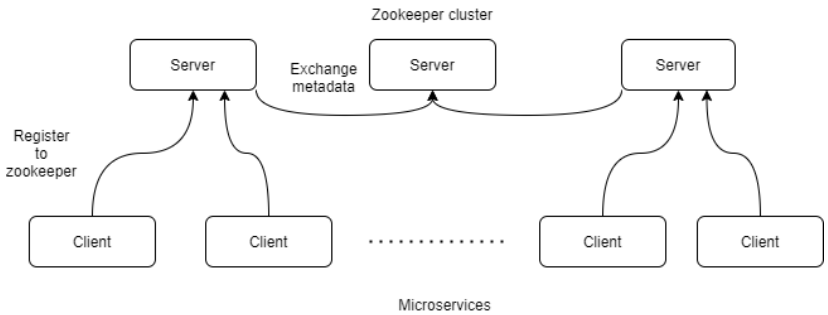

Fig. 3 Clustering with Centralized Service.

Examples of consensus protocols include Raft [19], Gossip protocol [20], and SWIM [21]. Raft operates on a single-leader model and uses elections to select leaders and followers. Gossip protocol employs an infection approach by broadcasting state to nearest neighbours. SWIM maintains membership and status through "ping" messages. Fig. **4** illustrate how the nodes communicate to each other without the centralized server.

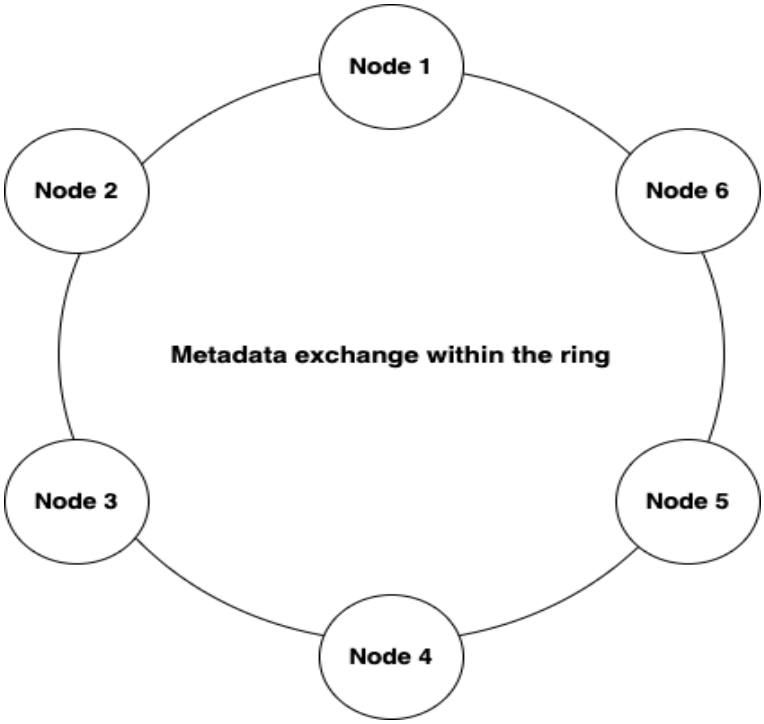

Fig. 4 Decentralized Cluster Management with Gossip Protocol

2.4 Motivation for Using Akka Cluster

Akka Cluster, a JVM-based cluster management tool from Lightbend, is based on the gossip protocol with membership. It implements the actor pattern as a basic building block of concurrency systems, abstracting complex algorithms and network protocols. This decentralized cluster management model offers several advantages over centralized models. In centralized cluster management, a middleman (e.g., ZooKeeper) is responsible for managing nodes, registering members, sharing state and metadata, and managing clustering-related activities such as adding or removing members and load balancing. While this approach provides higher consistency, it introduces overhead due to the need to communicate with the middleman to get the status and manage the cluster. In contrast, Akka Cluster's decentralized model allows the cluster to self-manage and react to changes immediately, reflecting the principles of the Reactive Manifesto. Specifically, Akka Cluster utilizes the Gossip protocol, which ensures high availability and eventual consistency through an infection-style process group membership. This decentralized, reactive approach reduces overhead and enhances the system's ability to handle dynamic and distributed environments effectively. Moreover, consensus protocols are not limited to microservices but are also relevant in blockchain systems. For instance, the paper "Evaluation of an Actor Model-based Consensus Algorithm on Neo Blockchain" [22] demonstrates the importance of consensus protocols in ensuring the integrity and reliability of blockchain networks. The use of consensus protocols, exemplified by Akka Cluster, highlights their significance in enhancing scalability and availability in distributed systems architecture.

Akka Cluster is just one of the toolkits that can provide decentralized cluster management, and its creator is one of the co-authors of the Reactive Manifesto, making it a good candidate to demonstrate how decentralized clusters work. The comparison and optimization with other decentralized cluster management toolkits are beyond the scope of this research. Conducting such comparisons would involve setting up complex testing environments for each protocol, which requires considerable time and expertise. However, such cluster optimization could be a focus for

future studies, providing a broader understanding of the performance and scalability of various decentralized cluster management tools. This comprehensive approach to cluster management and the alignment with reactive principles make Akka Cluster a robust and efficient choice for developing a scalable, flexible, and highly available data connector in Industry 4.0 environments. It should also be noted that as of October 2022, Akka has moved to a commercial license model, which may impact its accessibility for some users. Future research can build upon this foundation to explore and compare other decentralized cluster management solutions that remain open source.

While microservices and reactive microservices architectures offer greater flexibility, scalability, and maintainability compared to monolithic architectures, they require additional effort in designing and managing the independent services, as well as ensuring reliable and secure communication between services. Addressing these challenges aligns with the core objectives of our research. The connector prioritizes flexibility by ensuring that all components are decoupled and autonomous, allowing the system to seamlessly adapt to different manufacturing environments and connect with various modern protocols and applications. High availability and scalability are critical in Industry 4.0 environments, driving the incorporation of robust error handling, recovery mechanisms, and failover strategies in the connector to mitigate the risk of data loss or service interruption, especially during peak operational hours. Additionally, the emphasis on scalability is motivated by the need to support dynamic and growing demands in modern manufacturing systems. Moreover, the connector focuses on enhancing data processing efficiency by facilitating the seamless handling of high-volume data streams without overburdening computational resources. Through optimized data processing pipelines and resource allocation strategies, the connector ensures efficient utilization of hardware resources while maintaining responsiveness and reliability. This objective underscores the importance of efficient data processing in enabling real-time decision-making and analytics in Industry 4.0 environments.

3. Research Methodology

This study employed a hybrid approach that integrated the Design Science Research Methodology (DSRM) [23,24] with Agile practices [25]. This combined methodology was chosen for its suitability in addressing the research objectives of designing, implementing, and evaluating a reactive microservices-based connector to enable seamless interoperability in Industry 4.0 environments.

The rationale behind this methodological choice lies in its ability to provide practical solutions to real-world problems through an iterative development process and close collaboration with stakeholders. While DSRM offered a structured framework for creating innovative artifacts, the integration of Agile practices fostered flexibility, iterative development cycles, and adaptability to evolving requirements and stakeholder feedback.

The key steps of the DSRM process employed in this study included:

i) Problem Identification: Identifying the challenge of achieving seamless interoperability between heterogeneous systems in Industry 4.0 environments.
ii) Objective Definition: Defining clear objectives to guide the design and development process, including ensuring flexibility, scalability, and enhanced data processing efficiency.
iii) Design and Development: Creating an innovative artifact, the reactive microservices-based connector, leveraging microservices architecture, reactive programming principles, and modern communication protocols.

iv) Evaluation: Rigorously assessing the artifact's effectiveness in addressing the identified problem and achieving the defined objectives, based on criteria such as functionality, performance, reliability, and usability.
v) Reflection: Reflecting on the design process and outcomes to identify lessons learned and areas for improvement, informing future iterations of the design process.

3.1 System Architecture Design

The system architecture design played a crucial role in the development of the microservices data connector, providing a high-level representation of the system's structure and identifying key components that enable seamless integration into existing manufacturing environments.

The microservices architecture was well-suited for the data connector system, adhering to the principle of "do one thing and do it well" [26]. This approach ensured that the service was solely responsible for data transfer, without additional tasks. Microservices were designed to be autonomous and independent, allowing for the use of any programming language, framework, or platform, and facilitating easier upgrades or enhancements.

The reactive microservice architecture variant was adopted to achieve high availability and enhanced data processing efficiency. It enabled microservices to react to changes in the environment, failures, and limitations, ensuring a highly responsive and efficient data connector.

3.2 Enabling Clustering with Akka Cluster, Akka HTTP and Akka Streams

To achieve high availability, fault tolerance, and scalability, the microservices data connector employed Akka Cluster to enable clustering of its microservices. Akka Cluster allowed the microservices to form a distributed cluster and work together as a cohesive unit, enabling high availability and scalability. Any node within the cluster could handle requests and provide services, while clustering ensured fault tolerance, allowing the system to continue functioning even if one or more nodes encountered failures.

Akka HTTP was utilized to enable clustering among the microservices, providing a robust and efficient foundation for building RESTful services integrated into the microservices architecture. By leveraging Akka HTTP, the connector's microservices could communicate using HTTP-based protocols, facilitating clustering and distributing workloads.

Reactive streaming with Akka Streams and Akka HTTP transformed the microservices data connector into a highly responsive and efficient data processing system. Akka Streams empowered the creation of asynchronous, backpressure-aware data processing pipelines, enabling real-time data processing, efficiency and resource optimization, resilience and fault tolerance, and scalability through distributed stream processing.

3.3 Testing Methodologies

Testing was a crucial aspect of the development process, ensuring the reliability, performance, and correctness of the microservices data connector. The testing methodologies employed included:

Data Collection: Collaborating with domain experts to understand the existing data sources, formats, and challenges within the manufacturing environment, identifying key stakeholders, and assessing data quality.

Testing the Clustering Feature: Evaluating the system's ability to adapt to node failures, handle leader election, maintain data consistency across the cluster, and assess the impact of increased loads on the cluster's scalability.

Testing Reactive Streaming: Evaluating the efficiency and reliability of the data processing pipeline, measuring the system's ability to handle real-time data streams with low latency and high throughput, conducting load testing, and examining the integration of reactive streaming with the clustering feature.

Through rigorous testing of the clustering feature, reactive streaming, and overall system performance, the microservices data connector's reliability, resilience, and ability to meet the defined objectives were thoroughly evaluated, serving as a basis for making necessary adjustments and improvements.

4. Design and Implementation

The design and implementation of the microservices data connector within the context of Industry 4.0 environments require careful consideration of various architectural aspects and design patterns. This chapter provides an overview of the design and implementation process, highlighting key decisions and considerations made throughout the development lifecycle.

4.1 Connector in Microservice Architecture

The microservices data connector serves as a critical component within the overall architecture, facilitating seamless interoperability between disparate systems and services. Built upon microservices principles, the connector architecture embodies modularity, flexibility, and scalability, enabling it to adapt to diverse environments and evolving requirements. At its core, the connector implements a range of features to ensure robust connectivity and efficient data exchange.

Security is paramount in the design of the microservices data connector. Leveraging the capabilities of the API Gateway, the connector incorporates authentication and authorization services, generating OAuth tokens from an OAuth server to enable secure communication with external systems. Furthermore, the connector can validate these tokens via the OAuth server, ensuring that only authorized entities can access its services.

The deployment context of the connector is carefully managed to optimize scalability and manageability. It provides APIs to share crucial information about its deployment, including location, type, participants, and other metadata. Alternatively, this metadata can be centrally managed by a service registry, streamlining the deployment process, and enhancing system scalability.

To facilitate resource discovery and utilization, the connector incorporates a catalogue feature, allowing users to query resource metadata from a broker. This functionality enables efficient access to resources, enhancing overall system interoperability. In terms of host connectivity, the connector is designed to connect to different hosts using standard protocols such as HTTP or custom protocols supported by third-party libraries. This flexibility ensures compatibility with diverse environments and systems, enabling seamless integration and communication.

Data pre-processing is seamlessly integrated into the core functionality of the connector, leveraging the Reactive Stream API. This allows for efficient and scalable data processing, enabling real-time analysis and transformation of incoming data streams. API exposure is managed through the API Gateway, which controls the exposure of the connector's API endpoints based on predefined policies and configurations. This ensures controlled access to the connector's services, enhancing security and governance. By incorporating these features into its design and implementation, the

microservices connector achieves a high level of flexibility, scalability, security, and efficiency, making it well-suited for addressing the interoperability challenges inherent in Industry 4.0 environments.

4.2 Designing the Microservices Data Connector

The architecture of the microservices data connector is designed to facilitate seamless integration and data exchange between various manufacturing systems, applications, and protocols. At a high level, the architecture is based on a distributed and modular approach, enabling each component to operate independently while collaborating efficiently as a whole system. The primary architectural components of the microservices data connector include:

Microservices: The core of the architecture is built on the microservices paradigm. Each microservice represents a self-contained and independent functional unit responsible for specific tasks, such as data ingestion, data processing, and data delivery. The microservices are designed to be lightweight and loosely coupled, promoting flexibility, maintainability, and scalability.

Data Processing Pipeline: The data processing pipeline plays a crucial role in efficiently handling data streams and enabling real-time data processing. It consists of multiple stages, each performing specific data transformations and filtering operations. The pipeline ensures that data is efficiently processed and forwarded to the appropriate microservices for further actions.

Clustering Management: Clustering management is a critical aspect of the microservices data connector architecture, enabling high availability, fault tolerance, and scalability. This feature allows multiple instances of microservices to work together as a cohesive cluster, ensuring continuous service availability even in the face of failures and increasing the system's capacity to handle a higher load of incoming data.

Reactive Streaming Integration: Reactive streaming is incorporated into the architecture through Akka Streams, providing a powerful tool for handling large volumes of data streams efficiently. Reactive streams facilitate non-blocking and asynchronous processing, essential for managing high-throughput real-time data. By adopting reactive streaming, the microservices data connector can efficiently process and transmit data, ensuring continuous handling of incoming data without delays or bottlenecks. Key benefits include backpressure handling, asynchronous processing, and memory efficiency.

The integration of Akka Streams involves defining streams using a fluent DSL (Domain-Specific Language) and applying various transformations and operations to process the data streams. Within the data processing pipeline, reactive streaming enables efficient handling of the continuous flow of incoming data. As data is ingested, it traverses through a series of Akka Streams' Flows, where various processing operations occur, such as data transformation, filtering, and enrichment. The reactive nature of Akka Streams ensures asynchronous data processing with backpressure handling, preventing data congestion and resource exhaustion.

This design choice results in an efficient and responsive data processing pipeline, enabling the microservices data connector to handle data in real-time with low latency. By integrating reactive streaming with Akka Streams, the microservices data connector achieves enhanced data processing efficiency, facilitating the seamless handling of high data volumes without overburdening computational resources.

4.3 System Development

The system development phase involves selecting appropriate technology stacks and tools to implement the designed architecture. Build management, microservices implementation, integration

with Akka Cluster and Akka Streams, as well as configuration and deployment processes, are essential aspects of system development.

To address the identified problem, the proposed solution involves the development of a proof of concept using microservice architecture. This approach follows a structured software development life cycle encompassing stages such as development, unit testing, and system testing[25], [27]. During development, essential components of the microservice architecture, including the connector, cluster management system, and streaming system, are constructed with careful consideration of their intended functionalities.

Unit testing plays a crucial role in ensuring the correctness and reliability of each component. Rigorous testing procedures are employed to identify and address potential issues or bugs promptly, guaranteeing that the components function as intended. Subsequently, system testing is conducted to comprehensively assess the overall performance and efficacy of the developed system, measuring its effectiveness in addressing the identified problem and handling real-world scenarios seamlessly.

The microservices connector is built using a combination of technologies and tools that support the implementation of a robust and scalable system. The primary components of the technology stack include the Scala programming language and the Akka toolkit. Scala's versatility and expressive nature make it an ideal choice for building distributed systems, while Akka provides libraries for creating reactive, concurrent, and distributed applications.

Akka Cluster facilitates clustering management, enabling high availability and fault tolerance, while Akka Streams are utilized for efficient stream processing. Simple Build Tool (SBT) is employed for build management, offering functionalities to compile, test, package, and deploy Scala projects. External configuration files are utilized for configuring the microservices data connector, allowing easy adjustment of parameters and settings without code changes.

The microservices data connector is implemented as a collection of loosely coupled microservices, each responsible for specific tasks such as data ingestion, processing, streaming, storage, and retrieval. Integration with Akka Cluster and Akka Streams enables effective clustering management and efficient stream processing, respectively. Kubernetes orchestrates deployment, ensuring consistent and reliable deployment across various environments.

By utilizing this technology stack and implementing microservices as described above, the microservices data connector achieves a flexible, scalable, and efficient architecture capable of handling real-time data streams and meeting specified research objectives.

4.4 Testing Environment and Methodology

Establishing a robust testing environment and methodology is crucial for ensuring the reliability and performance of the microservices data connector. The testing environment is carefully configured to simulate real-world scenarios, and various testing methodologies, including Akka Cluster tests and reactive streaming tests, are employed to evaluate the system's behaviour under different conditions. By conducting comprehensive tests, developers can identify and rectify potential issues, guaranteeing a high-quality product that meets the desired standards of functionality and reliability.

4.4.1 Testing Environment

To assess the effectiveness and performance of the microservices data connector, a comprehensive testing environment and methodology were employed. The evaluation aimed to

measure the system's ability to achieve the defined objectives and verify its efficiency, scalability, and responsiveness.

The testing environment was meticulously configured to simulate real-world conditions while ensuring flexibility and ease of deployment. It comprised virtual machines running on high-end servers, allowing for the simulation of multiple microservices instances and the deployment of the microservices data connector. The hardware specifications used in the research are detailed in Table 1.

Table 1
Hardware Specification used for this testing

Specification	Description
Model	HX240C-M5SX
Processor	Intel Xeon Platinum 8168 CPU @ 2.7GHz
Hypervisor	VMware ESXi, 6.7.0.14320388
Logical Processors	96
Virtual Machines	39 created and 8 is used for this research
Operating System in VM	Ubuntu
JVM	Java 11

To address the challenges associated with testing microservices at scale, 39 virtual machines running either Ubuntu or Windows operating systems were utilized. Each virtual machine was equipped with 4 CPUs, 8GB of memory, and 100GB of storage. However, it's important to note that these machines were shared among multiple researchers, leading to unpredictable CPU load variations. Despite these limitations, the HX240C-M5SX machines provided an exceptional platform for microservices development and research.

4.4.2 Testing Methodology

The testing methodology comprised a series of controlled experiments designed to assess specific aspects of the microservices data connector's performance. Key areas of focus included clustering performance and reactive streaming efficiency.

Testing the Akka clustering feature involved a range of methodologies, including telemetry monitoring, automated and manual testing, and chaos testing. Telemetry monitoring enabled developers to gain insights into the cluster's behavior and performance, facilitating prompt issue identification and resolution. Unit tests and integration tests were employed to assess individual components and interactions between different microservices, respectively. Load testing tools like JMeter were utilized to evaluate performance and scalability across varying traffic levels. Chaos testing involved intentionally introducing faults and failures to assess system resilience and identify potential weaknesses.

The testing of reactive streaming focused on assessing performance in terms of data delivery and processing efficiency compared to conventional HTTP request-response approaches. When configured optimally, reactive systems exhibited drastically increased throughput during load tests until reaching maximum efficiency. The backpressure mechanism ensured that the system was not overloaded, preventing resource exhaustion and system halts. Fig. **5** illustrates the expected pattern of throughput versus time in reactive systems, considering external factors such as Java garbage collection and operating system activities.

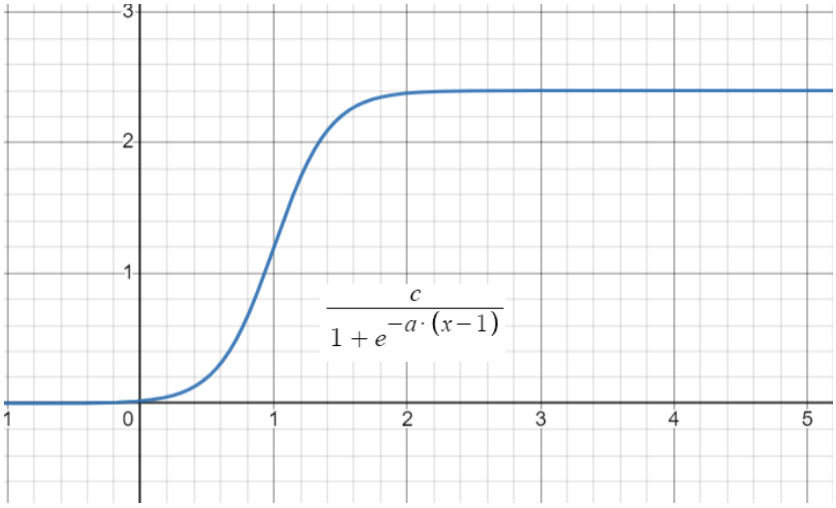

Fig. 5 Sigmoid Graph describes the pattern of the maximum throughput cap by the systems limitation. c is the maximum efficiency, a is the request rate or processing rate. Y-Axis is throughput, X-axis is time.

The design and implementation of the microservices data connector represent a significant milestone in addressing the interoperability challenges in Industry 4.0 environments. By leveraging microservices architecture, design patterns, and modern technologies, the connector demonstrates the potential to facilitate seamless data exchange and integration across heterogeneous systems.

4.5 IDS Connector Transformation

In the process of evaluating the IDS Connector, it's essential to delve into its transformation, particularly focusing on the shift towards a microservices architecture. This transformation marks a significant evolution in the architecture, moving away from monolithic systems towards a more modular and scalable approach.

The IDS Connector transformation involves restructuring the architecture to leverage microservices governance components effectively. These components include the Gateway, Service Discovery, and Authentication mechanisms, each playing a vital role in ensuring the reliability, security, and flexibility of the system.

The Gateway serves as a centralized entry point, facilitating communication between external systems and the microservices within the IDS Connector. By consolidating external interactions through the Gateway, the system gains greater control over access, security policies, and traffic management. This centralized approach enhances security and simplifies the management of incoming requests.

Service Discovery emerges as a critical component in dynamic ecosystems, enabling microservices to locate and communicate with each other seamlessly. In the context of the IDS Connector, Service Discovery facilitates the dynamic discovery of microservices, allowing them to adapt to changes in the environment, such as scaling, updates, or failures. This dynamic nature ensures the resilience and scalability of the system, even in complex and rapidly changing environments.

Authentication mechanisms play a crucial role in ensuring secure interactions between microservices and external systems. By implementing robust authentication protocols, such as OAuth or token-based authentication, the IDS Connector can authenticate and authorize requests effectively, safeguarding sensitive data and resources from unauthorized access.

Furthermore, the use of Akka Cluster introduces an alternative approach to service registration and management within the microservices architecture. Akka Cluster provides robust cluster management capabilities, allowing microservices to operate in a clustered environment with high availability and fault tolerance. This alternative approach enhances the scalability and resilience of the IDS Connector, ensuring uninterrupted service delivery even in the face of failures or increased demand. There are many forms setup of microservice architecture, Fig. **7** illustrate one of the common use patterns to transform the original IDS Connector illustrate in Fig. **6**.

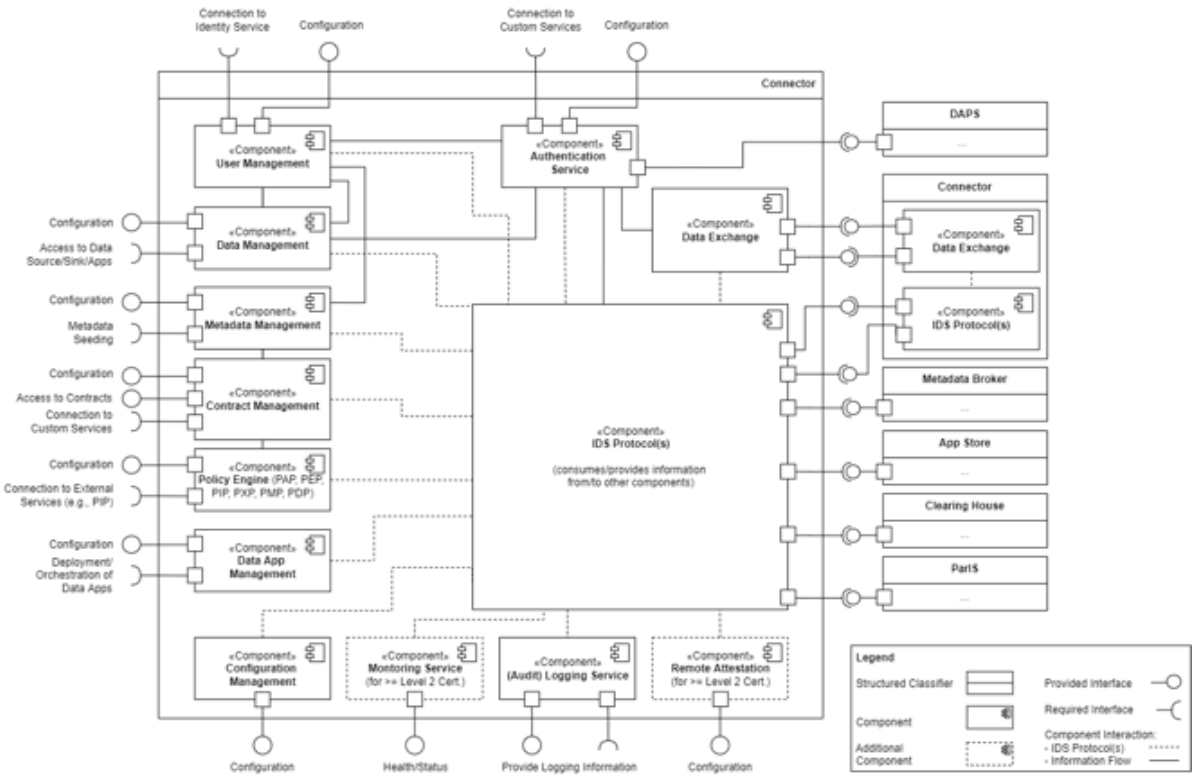

Fig. 6. The IDS Connector Architecture describe in IDS-RAM 4.0

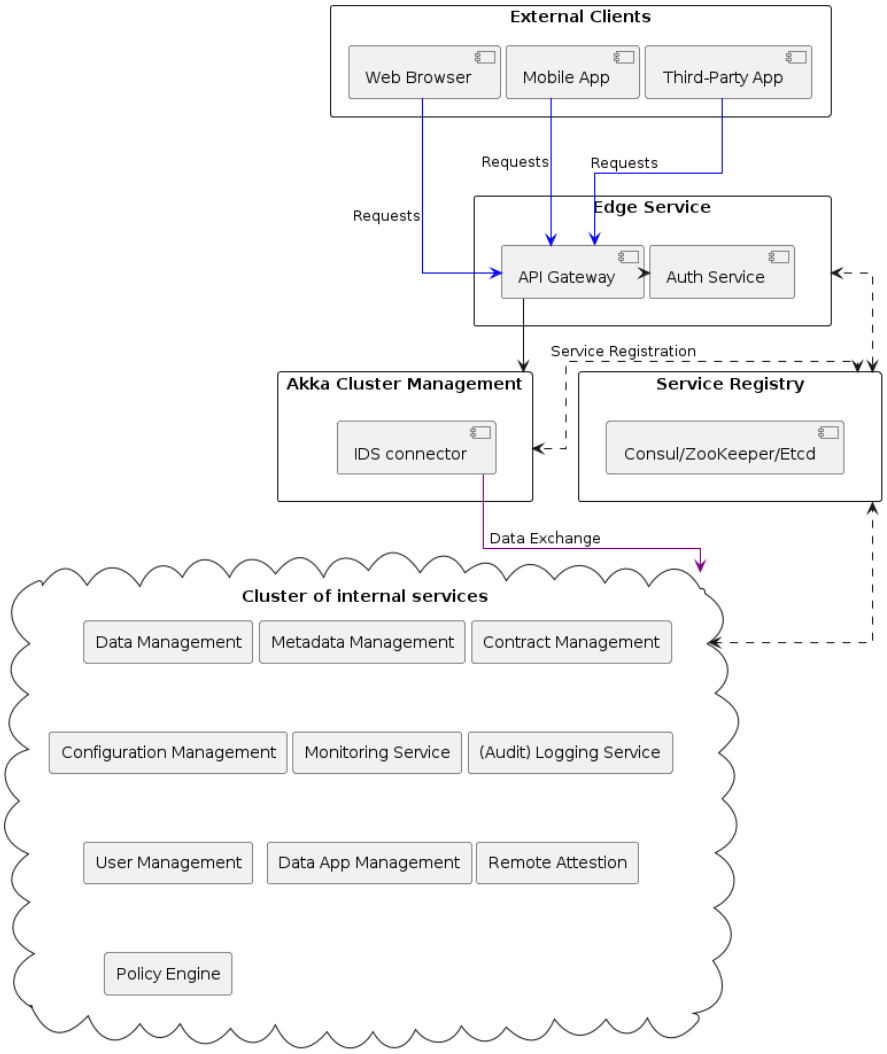

Fig. 7. IDS Connector after transform into Microservice Architecture by including the Gateway, Service Discovery and Authentication module

In conclusion, the transformation of the IDS Connector towards a microservices architecture is a strategic move aimed at enhancing its flexibility, scalability, and security. By leveraging microservices governance components such as the Gateway, Service Discovery, and Authentication mechanisms, along with Akka Cluster for cluster management, the IDS Connector can evolve into a more resilient, adaptable, and efficient system, capable of meeting the demands of dynamic manufacturing environments.

5. Evaluation and Discussion

5.1 Clustering for High Availability

In the evaluation and analysis of the IDS Connector, the results of two critical experiments conducted on a microservices system within a VMware virtual machine are presented. These experiments aimed to assess the clustering management and compare the traditional HTTP request-response protocol with Akka reactive streaming, utilizing the HTTP stream protocol.

The clustering management experiment demonstrated the configuration options available for Akka cluster member lookup. Two approaches were explored: defining seed nodes and predefining a list of members in a properties file. The results in Fig. **8** showcased the system's ability to form

cohesive clusters, manage cluster membership, and handle node failures effectively. Notably, the configuration with predefined cluster members provided a static and fixed number of nodes, ideal for illustrating the behavior of reactive streaming. Table 2 summarized of the activities from these observations.

Fig. 8. Example of Activities like Joining, Leaving and Removing from the cluster.

Table 2
Summary of the activities shown in Fig. **8**

Time		Reachable node	Unreachable node	Activity
From	To			
19:39	19:44	3	0	Start-up 3 nodes
19:44	19:46	5	0	Start-up 2 nodes, it takes a while to accept to the cluster
19:46	19:46	4	0	Shutdown 1 node
19:47	19:47	3	0	Shutdown 1 more node
19:48	19:48	2	0	Shutdown 1 more node
19:51	19:55	5	0	Start back 3 nodes
19:55	19:59	3	2	2 nodes not reachable, removed from cluster. Unreachable node will force shutdown after 30s
19:59	20:01	5	0	Start back 2 nodes
20:01	20:01	4	1	1 node detected unreachable and rejoin back within 30s.

The evaluation of clustering performance focused on scalability, resilience, and handling node failures. The Akka Cluster demonstrated its ability to form cohesive clusters, manage membership effectively, and redistribute responsibilities in the event of node failures. Load testing revealed that the clustering mechanism efficiently distributed tasks among cluster members, maintaining high availability and scalability under heavy traffic conditions.

Overall, the results of the clustering management experiment and the evaluation of clustering performance demonstrate the effectiveness of the Akka Cluster in achieving high availability, scalability, and resilience in microservices architectures. These findings validate the suitability of Akka clustering for building robust and reliable distributed systems, such as the IDS Connector, and highlight its importance in ensuring the system's stability and performance.

5.2 Streaming versus non-Streaming

The testing aims to showcase the effectiveness of reactive streaming in processing data in chunks, preventing server crashes caused by resource depletion, a common issue in traditional HTTP request-response models. Using Apache JMeter, post requests with large payloads are generated to a microservice API with reactive streaming enabled, and another test is conducted with an API handling normal HTTP request.

The results in Fig. **9**, Fig. **10** and Fig. **11** highlight the benefits of reactive streaming in preventing server crashes and enhancing performance. In the streaming API test, JMeter sends requests to the HTTP stream endpoint, resulting in efficient processing of data in chunks. The microservice demonstrates high throughput, handling over 100 transactions per second with dynamic resource allocation based on available threads. This showcases the and efficiency of Akka reactive streams in handling continuous data streams.

Conversely, in the HTTP API test, requests are sent to a standard HTTP endpoint. The microservice struggles to handle the large volume of data, resulting in out-of-memory errors and failed requests. Even with reduced data size, the microservice fails to process all requests efficiently, indicating limitations in handling large payloads with traditional HTTP request-response models.

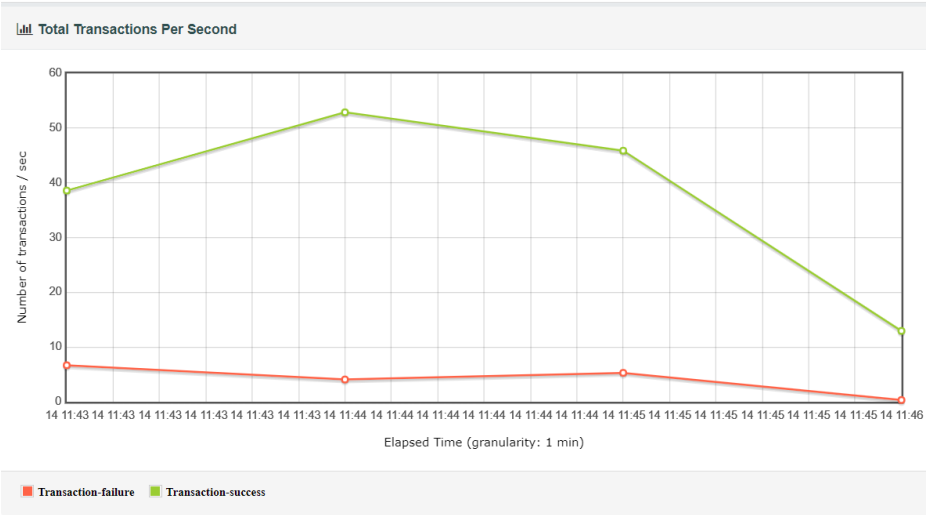

Fig. 9 JMeter Test Result for Reactive Streams with 1000 Threads 2000 Data and 100 Iterations

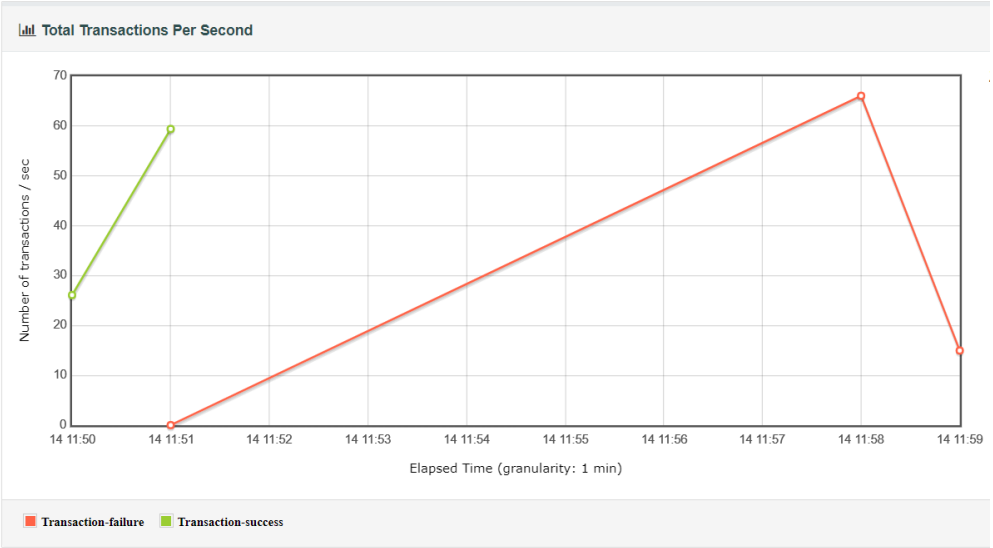

Fig. 10 JMeter Test Result for Http Post with 1000 Threads, 2000 Data and 100 Iterations

```
Created the tree successfully using HTTP_post2.jmx
Starting standalone test @ 2022 Sep 14 11:50:44 UTC (1663156244040)
Waiting for possible Shutdown/StopTestNow/HeapDump/ThreadDump message on port 4445
Warning: Nashorn engine is planned to be removed from a future JDK release
summary +   1568 in 00:00:16 =   98.7/s Avg:    46 Min:     2 Max:   317 Err:     0 (0.00%) Active: 420 Started: 433 Finished: 13
summary +   3537 in 00:00:30 =  118.6/s Avg:   219 Min:     2 Max: 11929 Err:     1 (0.03%) Active: 840 Started: 1000 Finished: 160
summary =   5105 in 00:00:46 =  111.7/s Avg:   166 Min:     2 Max: 11929 Err:     1 (0.02%)
summary +   3997 in 00:07:30 =    8.9/s Avg: 93182 Min:     0 Max: 445743 Err:  3967 (99.25%) Active: 402 Started: 1000 Finished: 598
summary =   9102 in 00:08:16 =   18.4/s Avg: 41013 Min:     0 Max: 445743 Err:  3968 (43.59%)
summary +    898 in 00:00:03 =  281.1/s Avg:     3 Min:     0 Max:    33 Err:   898 (100.00%) Active: 0 Started: 1000 Finished: 1000
summary =  10000 in 00:08:19 =   20.0/s Avg: 37330 Min:     0 Max: 445743 Err:  4866 (48.66%)
Tidying up ...    @ 2022 Sep 14 11:59:03 UTC (1663156743205)
... end of run
```

Fig. 11 Higher rate or error due to Http Post approach not able to accept connections

The analysis of reactive streaming efficiency highlights the superior performance of Akka Streams in processing real-time data streams. The system achieves low latency and high throughput, efficiently handling continuous data streams without compromising performance. The reactive nature of Akka Streams optimizes resource utilization, preventing memory overflows and ensuring stability during data processing.

5.3 Experimentation, Data Types and Environment

This study primarily focused on evaluating the performance and scalability of the reactive microservices-based data connector using HTTP-based requests. The choice of HTTP was deliberate—it is a widely adopted protocol, straightforward to implement, and well-suited as a baseline for this initial evaluation. One of the main reasons for limiting the scope to HTTP requests was the complexity involved in setting up testing environments for each protocol. Different protocols come with unique configuration requirements and learning curves, which demand significant time and effort to master. Other protocols like gRPC, MQTT, or WebSocket may have their strengths in specific scenarios, their inclusion was beyond the scope of this phase of the research.

The testing environment relied on virtual servers due to cost considerations. Virtualization provided a flexible and scalable setup for development and testing, but it's important to acknowledge the trade-offs. Virtual servers introduce an additional layer of abstraction, which can result in higher latency and reduced computational efficiency compared to bare-metal servers. As highlighted by George from FS.com[28], bare-metal servers are better suited for environments demanding consistent, high performance, as they eliminate the overhead associated with virtualization. While the use of virtual servers allowed us to explore scalability and fault tolerance effectively, future

testing on physical hardware will be essential to validate the system's real-world performance in industrial settings.

5.3.1 Error Handling and Recovery

The error handling and recovery mechanisms in this study are primarily managed by Akka Cluster. Akka Cluster provides robust features for handling node failures, redistributing tasks, and maintaining cluster integrity. Key mechanisms include gossip-based communication for state sharing, automatic leader election, and fault-tolerant recovery strategies. These features enable the system to detect and recover from intra-cluster communication failures efficiently, manage latency during peak loads through asynchronous processing, and restore operations seamlessly after outages. Future enhancements could involve more granular error reporting and advanced recovery strategies tailored for specific industrial use cases. Figure 12 shown the life cycle of Gossip Protocol.

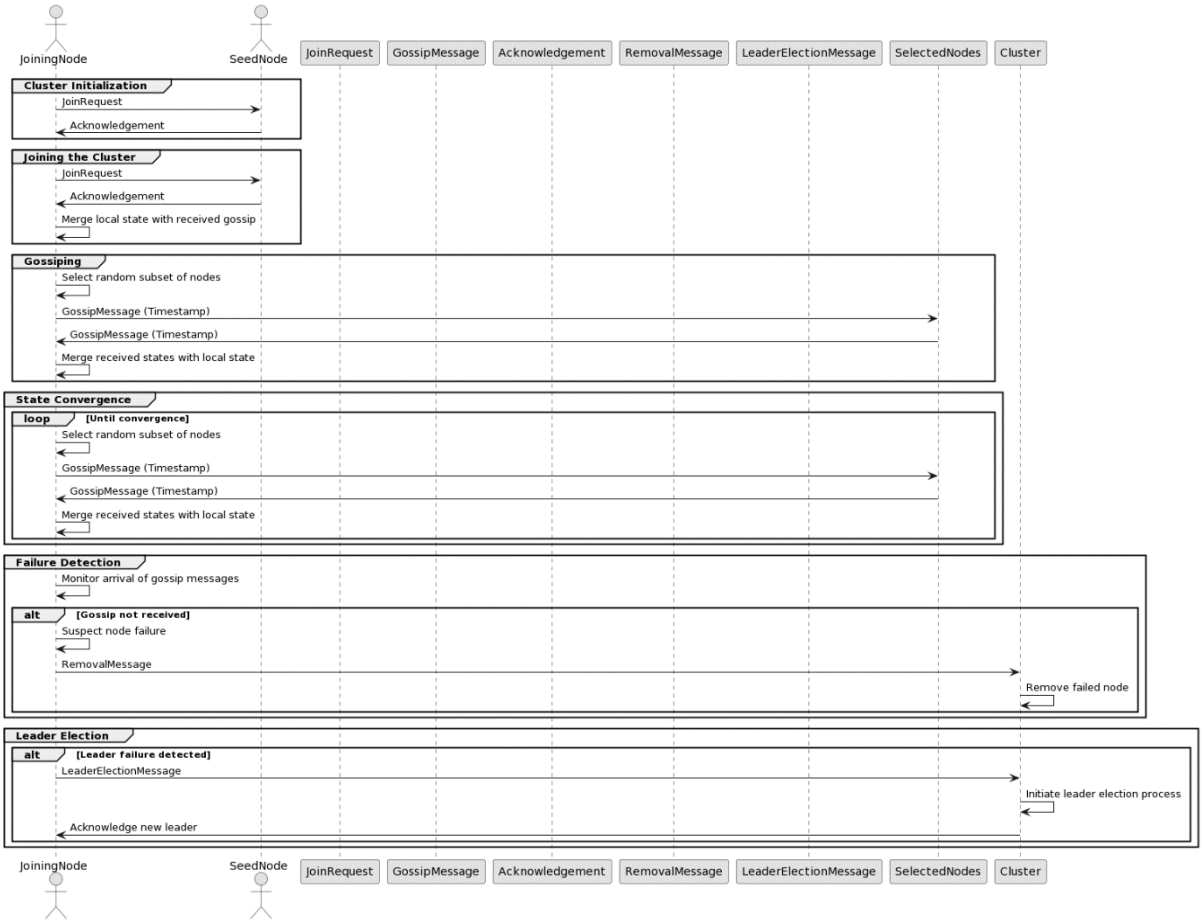

Fig. 12 Gossip Protocol Life Cycle in Akka Cluster

5.3.2 Evaluation Limitations and Future Research

It's important to reflect on some limitations of this study and the directions for future work. Testing was conducted in a virtualized environment, which, while cost-effective and flexible, may not fully capture the nuances of real-world deployments on physical hardware. Future research will address this gap by conducting tests on bare-metal servers, focusing on key performance indicators like latency and throughput under industrial workloads.

Additionally, while HTTP served as a practical starting point, protocols like gRPC, MQTT, and WebSocket each bring unique capabilities that could enhance the system's versatility:

gRPC: Known for its efficient serialization and low latency, gRPC is well-suited for high-throughput scenarios requiring rapid data exchange.

MQTT: A lightweight protocol ideal for IoT applications, particularly in environments where devices have limited resources or network constraints.

WebSocket: Designed for bidirectional, real-time communication, WebSocket is especially useful for continuous data streaming or interactive use cases.

Comparing these protocols will provide a clearer picture of the connector's adaptability across different use cases. Metrics such as throughput, latency, and fault tolerance will be carefully measured to identify the strengths and weaknesses of each protocol. By exploring these alternatives, we aim to demonstrate the connector's ability to meet the diverse demands of Industry 4.0 environments. This will allow us to refine the system further and optimize its performance for a variety of scenarios.

Despite these limitations, the findings from this study offer valuable insights into the connector's design and functionality. The results demonstrate its scalability and resilience within a controlled virtualized setup, laying a strong foundation for future enhancements and broader applicability in Industry 4.0 environments.

5.3.3 Case Study Integration

The proposed framework offers flexibility and scalability, making it suitable for various real-world scenarios. Some examples include:

i) Heat Transfer Optimization: The study by Fauzi *et al.,* [29] explores the use of nanofluids and regression analysis to improve heat transfer efficiency in complex fluid systems. Applying the reactive microservices-based data connector to similar domains could enhance data processing for real-time monitoring and optimization of heat transfer systems. The framework could facilitate efficient data exchange between sensors, analytical models, and control systems in such applications.

ii) Knowledge-Enhanced Educational Tools: Lu *et al.,* [30] demonstrated the potential of large language models, such as ERNIE Bot, to enhance educational outcomes in automotive marketing. By integrating AI-driven systems into the proposed data connector, real-time knowledge dissemination and decision-making could be enabled, supporting smart training systems and advanced learning platforms within Industry 4.0 environments.

iii) Geoinformation Systems for Landslide Mitigation: The geoinformation system for landslide mitigation presented by Yanuarsyah *et al.,* [31] could benefit from the flexible, scalable data exchange enabled by the reactive microservices-based data connector discussed in this paper. Integrating this framework could enhance system responsiveness and interoperability, especially when processing geospatial data in real time.

iv) IoT-Enabled GPS Tracking Systems: Similarly, the IoT-enabled GPS tracking system for school student safety described by Hanafi *et al.,* [32] aligns with the objectives of this research. By leveraging the proposed connector's scalability and real-time streaming capabilities, such systems could efficiently handle dynamic data streams while ensuring system reliability and scalability.

These case studies highlight the potential applications of the proposed connector in diverse fields, emphasizing its adaptability and broad impact.

6. Conclusion

In this paper, we have presented a comprehensive evaluation of a microservices data connector system, focusing on its transformation, performance, and efficiency. Through rigorous testing and analysis, we have gained valuable insights into the capabilities and limitations of the system, shedding light on its suitability for modern distributed environments.

The transformation of the microservices data connector, particularly in the context of governance and service orchestration, has been a key area of focus. By leveraging gateway, service discovery, and authentication mechanisms, we have demonstrated how the system can adapt to dynamic ecosystems, ensuring secure and efficient interactions between microservices. The integration of Akka Cluster as an alternative service registration mechanism further enhances the system's flexibility and scalability.

Our evaluation of clustering performance has highlighted the system's robustness in managing cluster membership and handling node failures. Through scalable configurations and effective split-brain resolution strategies, the system has exhibited resilience and high availability, essential characteristics for distributed environments. Furthermore, our analysis of streaming versus non-streaming approaches has underscored the importance of reactive streaming in modern data processing applications. By comparing the performance of reactive streaming with traditional HTTP request-response models, we have demonstrated the efficiency and scalability advantages offered by Akka Streams. The ability to process real-time data streams with low latency and high throughput positions the system as a reliable choice for Industry 4.0 environments.

Looking ahead, one potential area of exploration is the integration of advanced analytics and machine learning algorithms into the connector, enabling real-time data analysis and predictive maintenance capabilities. Additionally, the connector could be extended to support edge computing architectures, allowing for distributed data processing closer to the data source. These enhancements would further optimize operational efficiency and facilitate data-driven decision-making in Industry 4.0 environments.

However, the successful implementation of the microservices data connector also revealed certain limitations and challenges. One notable limitation stemmed from the use of a virtualized testing environment, potentially limiting the representation of real-world performance on physical hardware. To obtain a more accurate assessment of the system's performance, future work should focus on conducting tests on physical hardware. Additionally, further research and improvements are opportunities to boost the system's performance and adaptability in complex manufacturing environments.

During the development and evaluation of the microservices data connector, several challenges were encountered, and solutions were devised to overcome them. Fine-tuning of the cluster management configuration, optimization of the reactive streaming pipeline, and iterative development were key steps taken to address these challenges. These experiences underscore the importance of continuous improvement and adaptation in the rapidly evolving landscape of distributed systems architecture.

In conclusion, the microservices data connector has effectively addressed Industry 4.0 challenges, exhibiting flexibility, high availability, scalability, and enhanced data processing efficiency. The implementation of microservices architecture, clustering with Akka Cluster, and reactive streaming with Akka Streams has demonstrated effectiveness in creating a robust and efficient data connector. However, ongoing research and development efforts will be essential to further enhance the system's capabilities and meet the evolving demands of modern distributed environments.

References

[1] Lasi, Heiner, Peter Fettke, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. "Industry 4.0." *Business & information systems engineering* 6 (2014): 239-242. https://doi.org/10.1007/s12599-014-0334-4

[2] Federal Ministry for Economic Affairs and Energy, "Plattform Industrie 4.0 - RAMI4.0 – a reference framework for digitalisation," Plattform Industrie 4.0, 2019.

[3] "DIN SPEC 91345," vol. 0, no. April, Berlin, Heidelberg: Springer Berlin Heidelberg, 2016

[4] Hang, Jen Hin, Donovan Sheldon Charles, Zheng Hung Gan, Sze Kai Gan, Yee Mei Lim, Wah Pheng Lee, Thein Lai Wong, and Ching Pang Goh. "Constructing a real-time value-chain integration architecture for mass individualized juice production." *Information* 13, no. 2 (2022): 56. https://doi.org/10.3390/info13020056

[5] Gan, Sze-Kai, Thein-Lai Wong, Ching-Pang Goh, Wah-Pheng Lee, and Yee-Mei Lim. "A Review on the Development of Dataspace Connectors using Microservices Cross-Company Secured Data Exchange." In *International Conference on Digital Transformation and Applications (ICDXA)*, vol. 25, p. 26. 2021.

[6] Otto, Boris, Sebastian Steinbuß, Andreas Teuscher, Steffen Lohmann, S. Auer, S. Bader, H. Bastiaansen et al. "International Data Spaces: Reference Architecture Model Version 3." *International Data Spaces Association–IDSA, URL: https://www. internationaldataspaces. org/wp-content/uploads/2019/03/IDS-Reference-Architecture-Model-3.0. pdf (visited on 29/11/2019)* (2019).

[7] Sebastian Steinbuss, "IDS-RAM 4.0," International Data Spaces Associationv. 2024.

[8] Fowler, M. and J. Lewis, "Microservices."

[9] Bonér, J., D. Farley, R. Kuhn, and M. Thompson, "The Reactive Manifesto (Version 2.0)," Reactivemanifesto.Org, vol. 2, no. 16, (2014):1-2

[10] Xiao, Zhongxiang, Inji Wijegunaratne, and Xinjian Qiang. "Reflections on SOA and Microservices." In 2016 4th International Conference on Enterprise Systems (ES), pp. 60-67. IEEE, 2016. https://doi.org/10.1109/ES.2016.14

[11] Mansi Babbar, "Diving Into Reactive Microservices," Dzone.com. 2024.

[12] Bonér, J. Reactive Microsystems. 2017.

[13] Bonér, J. Reactive Microservices Architecture. 2016.

[14] Santana, Cleber, Leandro Andrade, Flávia C. Delicato, and Cássio Prazeres. "Increasing the availability of IoT applications with reactive microservices." Service Oriented Computing and Applications 15, no. 2 (2021): 109-126. https://doi.org/10.1007/s11761-020-00308-8

[15] and N. U. R. A. Khan, N. A. Khan, K. Salah, A. H. Abdullah, "A Comparative Study of Monolithic and Microservices-based Architecture," 2020 IEEE 4th Middle East Conference on Biomedical Engineering (MECBME), pp. 340–344, 2020.

[16] and S. L. W. Zhang, J. Zhang, "A Comparative Study of Microservices Architecture and Monolithic Architecture," 2019 18th IEEE International Conference on Cognitive Informatics & Cognitive Computing (ICCI*CC), pp. 248–253, 2019.

[17] and M. F. Z. M. Idrees, S. S. Riaz, M. A. Ali, A. Imran, "A Comparative Analysis of Microservices Architecture with Monolithic Architecture for E-Commerce Systems," 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), pp. 0252–0258, 2019.

[18] Hussain, Ishfaq. "Ant Colony Optimization based Design Space Exploration Engine (ACODSEE) for Balanced Energy and Throughput." (2015).

[19] Ongaro, Diego, and John Ousterhout. "In search of an understandable consensus algorithm." In 2014 USENIX annual technical conference (USENIX ATC 14), pp. 305-319. 2014.

[20] Birman, Ken. "The promise, and limitations, of gossip protocols." ACM SIGOPS Operating Systems Review 41, no. 5 (2007): 8-13. https://doi.org/10.1145/1317379.1317382

[21] Das, Abhinandan, Indranil Gupta, and Ashish Motivala. "Swim: Scalable weakly-consistent infection-style process group membership protocol." In Proceedings International Conference on Dependable Systems and Networks, pp. 303-312. IEEE, 2002. https://doi.org/10.1109/DSN.2002.1028914

[22] Suliyanti, Widya Nita, Muhammad Salman, and Riri Fitri Sari. "Evaluation of an actor model-based consensus algorithm on NEO blockchain." In 2021 31st International Telecommunication Networks and Applications Conference (ITNAC), pp. 60-64. IEEE, 2021. https://doi.org/10.1109/ITNAC53136.2021.9652155

[23] Vom Brocke, Jan, Alan Hevner, and Alexander Maedche. "Introduction to design science research." *Design science research. Cases* (2020): 1-13. https://doi.org/10.1007/978-3-030-46781-4_1

[24] Peffers, Ken, Tuure Tuunanen, Marcus A. Rothenberger, and Samir Chatterjee. "A design science research methodology for information systems research." *Journal of management information systems* 24, no. 3 (2007): 45-77. https://doi.org/10.2753/MIS0742-1222240302

[25] Laoyan, Sarah. "What is Agile methodology?(A beginner's guide)." Asana, Oct 15 (2022).

[26] Amey Dhavle, "What is a microservice?," medium.com. 2024.

[27] Matthew Martin, “Software Development Life Cycle (SDLC) Phases & Models,” Guru99. 2024.
[28] George, “Bare Metal Servers vs. Virtual Servers: How to Choose?”
[29] Fauzi, Farahanie, Abdul Rahman Mohd Kasim, Siti Farah Haryatie Mohd Kanafiah, Syazwani Mohd Zokri, Adeosun Adeshina Taofeeq, and Siti Hanani Mat Yasin. (2024). Advancing Heat Transfer: Exploring Nanofluids and Regression analysis on Lower Stagnation Point of a Horizontal Circular Cylinder for Brinkman-Viscoelastic Fluid. Semarak Engineering Journal, 7(1), 1–10. https://doi.org/10.37934/sej.7.1.110
[30] Lu, Hong Kun, Fei Song, Muhamad Mat Noor, and Bingli Wu. "Exploring the Application of Knowledge-Enhanced Large Language Models in Automotive Marketing Education: A Case Study of ERNIE Bot." Journal of Advanced Research in Technology and Innovation Management 13, no. 1 (2024): 1-12. https://doi.org/10.37934/jartim.13.1.112
[31] Yanuarsyah, Iksal, Syarbaini Ahmad, and Nurkaliza Khalid. "The Designing of Geoinformation Backend-Frontend to Improve Landslide Mitigation Application." *Journal of Advanced Research in Computing and Applications* 37, no. 1 (2024): 21-33. https://doi.org/10.37934/arca.37.1.2133
[32] Hanafi, Hafizul Fahri, Muhamad Hariz Adnan, Miftachul Huda, Wan Azani Mustafa, Miharaini Md Ghani, Mohd Ekram Alhafis Hashim, and Ahmed Alkhayyat. "IoT-Enabled GPS Tracking System for Monitoring the Safety Concerns of School Students." *Journal of Advanced Research in Computing and Applications* 35, no. 1 (2024): 1-9. https://doi.org/10.37934/arca.35.1.19