MESSAGE BROKER DESIGN FOR IOT AND COMPARING DATA PROCESSING TIMES WITH MOCK DATA

Süha UĞUZ TURGUT

Sivas Cumhuriyet University

20239257005@cumhuriyet.edu.tr - 0009-0006-0660-5918

Ahmet Turan KARAKUŞ

Sivas Cumhuriyet University

20239257014@cumhuriyet.edu.tr - 0009-0001-2941-3250

Ahmet Gürkan YÜKSEK

Sivas Cumhuriyet University

agyuksek@cumhuriyet.edu.tr – 000-001-7709-6360

ABSTRACT

The Internet of Things, known as IoT for short, is a deep technology framework and a globalizing field. The devices used with this technology exchange data in different ways. The IoT ecosystem needs effective data management and strong communication strategies to cope with the ever-increasing volume of data. The increasing number of IoT devices that collect instant or regular data, especially by utilizing machine learning and embedded technologies, and the increasing processing time of this collected data in proportion to the new incoming data have led to the development of new system designs. The collection and processing of this massive amount of data is still a subject that is being studied and improved. An example solution to control this data density is Message Broker designs. Especially in real-time systems such as artificial intelligence-supported embedded systems that aim to help visually impaired individuals who need instant data processing, vehicle autonomous systems, abnormal situation detection systems based on location tracking, it has become very important to coordinate communication between devices and optimize data flow. The standard Relational Database Systems (RDMS), Standard Message Queuing systems (such as RabbitMQ, Apache Kafka) are sometimes inadequate in the process of processing this huge amount of IoT data in a coordinated and performant manner. This paper aims to address different broker designs and analyze the processing times using Mock Data on a NoSQL database. It also examines the performance metrics of replicated Message Broker designs using data virtualization technology. The results are visualized and presented in an analytical and interactive way through Grafana. This enables meaningful evaluation of different broker designs. This enables meaningful evaluation of different broker designs. It also aims to measure the performance differences between different broker designs with visualized metrics such as memory, bandwidth, processor utilization, etc.

Keywords: IoT, Message Broker, Mock Data, Data Processing, Data Benchmarking

1.INTERNET OF THINGS AND BROKER SYSTEMS

Although the Internet of Things has been a well-known concept for many years, in recent years it has risen again with artificial intelligence as one of the fastest growing technologies. The Internet of Things refers to all technologies that enable communication both between devices and the cloud and between devices themselves [1]. This technology is applied in a wide range of applications, from industrial systems to home automation, from healthcare to agriculture. The Internet of Things is a global network of objects with sensing and data processing capabilities that can communicate with each other through different communication protocols [2]. The features and usage areas of different protocols determine their preference according to the designed system.

Data communication and management is an important issue in IoT systems. This is where broker systems come into play. Broker systems are middleware that facilitate data exchange between IoT devices. They regulate data flow, ensure security and manage data processing. For example, an IoT device can transmit data from sensors to a target device through a broker.

Some of the existing broker systems are solutions based on protocols such as MQTT (Message Queuing Telemetry Transport), AMQP (Advanced Message Queuing Protocol) and CoAP (Constrained Application Protocol), RabbitMQ, Apache Kafka and ActiveMQ. These systems are designed for different needs and generally offer features such as low power consumption, low bandwidth utilization and high scalability.

In this study, the importance and common uses of broker systems in IoT systems will be analyzed and examples of existing broker systems will be given. This analysis plays an important role in the widespread adoption of IoT technologies and further integration in daily life.

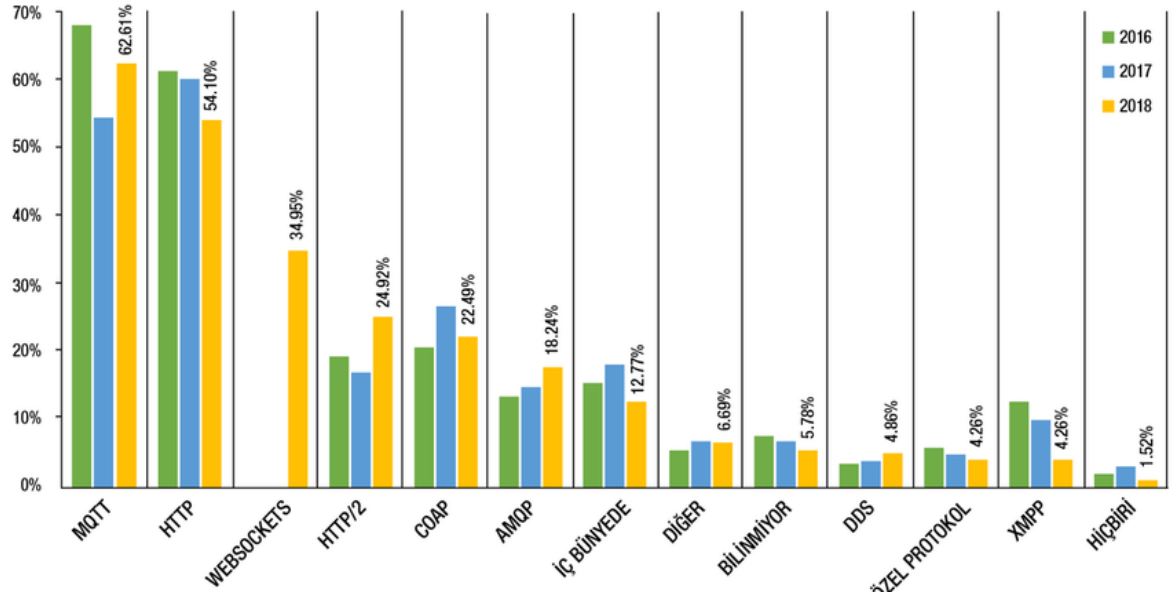

Image 1. Preferred Protocols in IoT Applications [3]

1.1. DETAILED REVIEW OF THE PROTOCOLS

1.1.1 MQTT Protocol (Message Queuing Telemetry Transport)

A standards-based messaging protocol or set of rules. It is used for device-to-device communication. The protocol is simple to implement. Since it can easily transmit IoT data, IoT devices use MQTT for such data transmission [4].

1.1.2. HTTP Protocol (Hyper-Text Transfer Protocol)

It is the most widely used protocol in the world. The most familiar use of HTTP is as a protocol for viewing web pages on the internet [5].

1.1.3. Web Sockets Protocol

It is a protocol developed to create a connection between a client and a server - usually a browser and a server - and to enable real-time communication between them [6].

1.1.4. HTTP/2 Protocol (Web 2.0)

The HTTP/2 protocol is a protocol that regulates the rules and methods for exchanging information between the accessed device and the server [7]. It is also known as Web 2.0.

1.1.5. CoAP Protocol (Constrained Application Protocol)

It is an application communication layer protocol designed to enable communication between IoT devices with low power consumption and low resource utilization [8].

1.1.6. AMQP Protocol (Advanced Message Queuing Protocol)

It is designed to exchange data securely and efficiently between messaging applications. It is an open source messaging protocol [9].

1.1.7. DDS Protocol (Data Distribution Service)

It was published by the Object Management Group (OMG) to facilitate the development of large-scale distributed systems. It is a data-driven middleware supporting publish/subscribe architecture [10].

1.1.8. XMPP Protocol (Extensible Messaging and Presence Protocol)

Jabber, formerly known as Jabber, is an open-source protocol that allows two ends of the Internet to transfer any structured information between each other mutually and almost simultaneously [11].

2.MESSAGE BROKER DESIGN

The basic components of broker systems are designed to manage data flow, provide security and offer high performance as well as efficiency throughout the system. In our study, the broker

system developed on Java coding language using RabbitMQ, Apache ActiveMQ and Apache Kafka technologies will be discussed. It has been developed to perform the functions of collecting, processing and distributing data from IoT devices.

Brokers have different characteristics in terms of their purpose and use cases. Apache ActiveMQ adopts the Java Message Service (JMS). It provides high performance in messaging and security. Apache Kafka is used to manage high volume data streams. In addition to Apache Kafka, which stands out with its durability and speed, RabbitMQ is preferred especially for complex messaging needs. It is a flexible and reliable middleware. In the developed broker design, the protocols and algorithms that regulate the data flow are harmonized through these two systems - Apache ActiveMQ and RabbitMQ. Measures such as encryption, access control lists and message authentication have been taken to ensure security and data integrity. This design maximizes data security and protects data integrity, which is critical for IoT applications. It also provides valuable insights on how broker systems can be optimized and securely managed in large-scale IoT applications.

2.1. CREATING MOCK DATA

Mock data plays a critical role in software development and testing, as it is used to mimic the complexity and behavior of real system data. Creating mock data can be done using predefined data sets, or it can be generated completely randomly. Random data generation aims to push the limits of systems by providing a wider variety of data scenarios.

Similarity to real data can be either structural or behavioral similarities. Structural similarity involves mock data mimicking the structural elements of real data sets, such as format, types and relational links. Behavioral similarity aims to model the behavior of the data as it changes over time or as a result of specific user interactions. These approaches help to understand how robust the systems under test are to real-world conditions.

There is a wide range of mock data creation tools and technologies. For example, the Faker library for the Python programming language allows developers to create realistic data sets with various localization options. JSON Server for JavaScript enables rapid prototyping for REST APIs. At the same time, the GenFu library is a library for creating data sets for the C# language. In addition, API testing and development tools such as Postman create mock services and facilitate API design and testing processes. These tools are particularly useful for testing networked systems, such as in IoT projects. This is because simulating realistic data flow simulations can help predict how systems will perform under real conditions. With these detailed simulations, developers can identify system bugs in advance and strengthen their applications against them.

As a strategy to achieve similarity to real data, our system does not require the use of any special method or technology. Instead, cities were set as fixed and a mechanism was developed to randomly generate distance and proximity distances. These distances were generated as data files in JSON format, processed through data brokers and then stored in a Cassandra database. This method minimizes the complexity and costs of the system while creating conditions similar

to real-world data. This simple yet effective approach increases efficiency in testing and development processes. It also allows for accurate simulation.

2.2. TECHNOLOGIES USED AND INTRODUCTION OF THESE TECHNOLOGIES

The core technologies used in this project include Kubernetes (K8s). Kubernetes is a platform designed to deploy, manage and automate container-based applications in a scalable and manageable way. K8s can automatically deploy, align and manage applications, and offers features such as load balancing and auto-scaling. In this project, K8s is used to manage and deploy applications in containerized environments.

Messaging systems such as RabbitMQ, Apache ActiveMQ and Apache Kafka play an important role for asynchronous data communication. RabbitMQ implements the AMQP (Advanced Message Queuing Protocol) protocol. It facilitates communication between various programming languages and platforms. Apache ActiveMQ is an open source messaging tool and offers broad protocol support, with a particular focus on JMS (Java Message Service). Apache Kafka is a distributed, high-performance and robust streaming platform. Kafka is designed to handle large-scale data streams. It facilitates data integration by processing real-time data streams.

Apache Cassandra is a highly scalable NoSQL database. It allows for a large number of low latency write and read operations. Moreover, thanks to its distributed architecture, it can be used across data centers and in cloud environments. In this project, Cassandra database was chosen to manage and store large-scale data streams.

These technologies are used together to improve the reliability, scalability and performance of the application. The combined use of RabbitMQ [12], Kubernetes [13], Apache ActiveMQ [14], Apache Kafka [15] and Apache Cassandra [16] creates a strong foundation to meet the requirements of the application. It facilitates the management and deployment of complex systems. Furthermore, this project uses a powerful data visualization tool such as Grafana. Grafana takes data from various data sources and presents it in interactive and customizable visualizations. Grafana's flexible and user-friendly interface facilitates data analysis and monitoring processes.

3. PROPOSED MESSAGE BROKER ARCHITECTURE SENSITIVE TO DATA LOSS AND FOCUSED ON PERFORMANCE

In large-scale or big data processing software projects, third-party applications such as Kubernetes (K8s), RabbitMQ, Apache ActiveMQ, Apache Kafka, Apache Cassandra are used to ensure that the system designed is secure, robust, accessible and scalable. In fact, the creation of big data and performance-oriented systems in accordance with these concepts depends on third-party software having the right infrastructure design and selecting the appropriate tools to be used. In this sense, the use of virtualization and open source projects provides the opportunity to work in harmony with other systems easily thanks to the cloud system [17]. With this opportunity, it becomes an important problem for IoT devices and data processing systems, which are increasing in today's world, to work in different locations and to access data quickly

and securely from different locations. To solve this problem, Cassandra technology, which can work on a data center basis and can manage large amounts of data with replications at all times, is used.

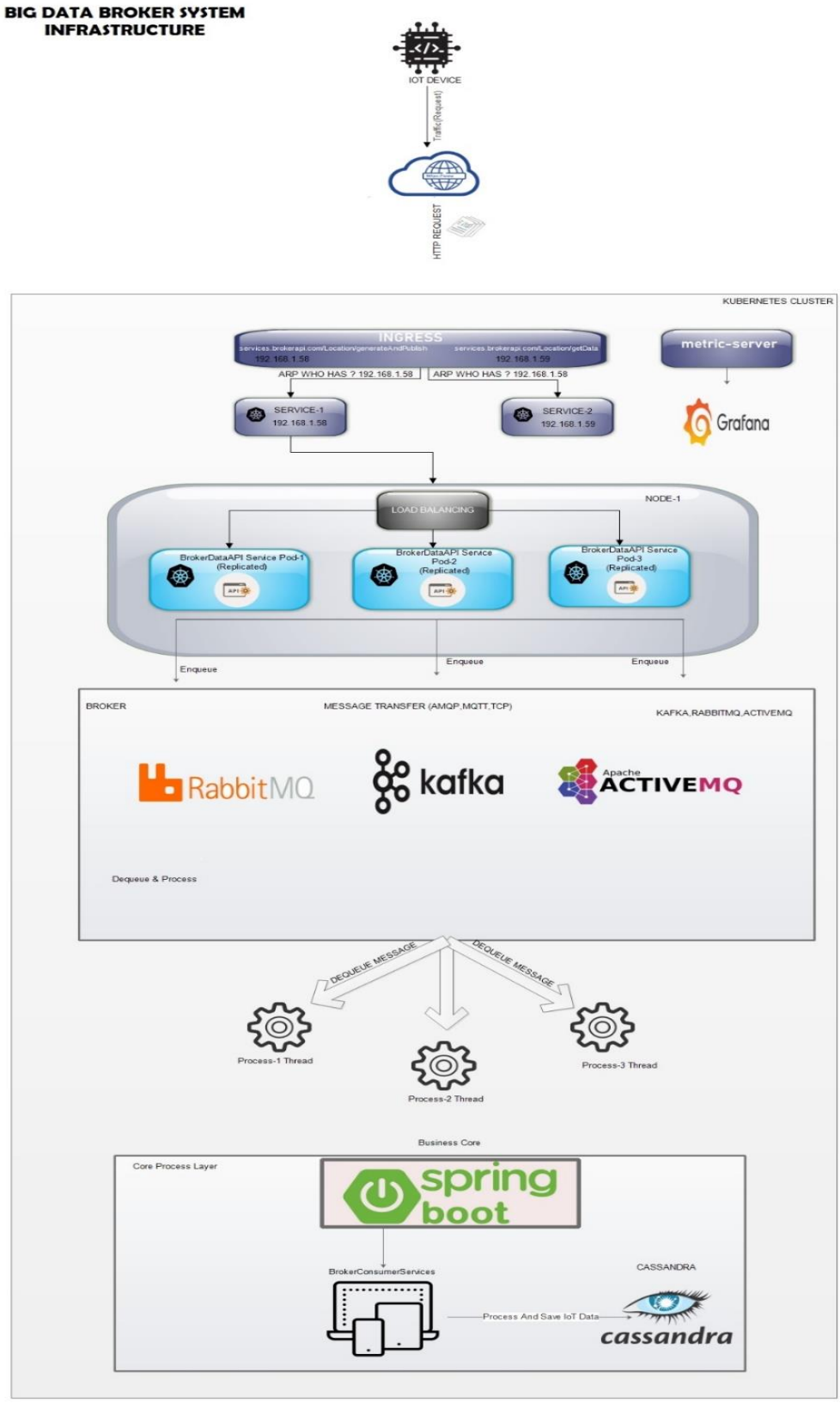

Image 2. Broker System Architecture

Broker systems such as ActiveMQ, RabbitMQ and Kafka are designed for different usage scenarios. For example, ActiveMQ is often preferred in traditional message queuing scenarios, while RabbitMQ is used as a flexible and general purpose message broker. Apache Kafka, on the other hand, is designed to handle large data streams and is used in real-time data processing scenarios. Any big data scenario often raises concerns about storage scalability and security of the data pipeline. Modern solutions can be effective in multiple areas. However, the selection of different broker systems ignores the resources used by the broker systems mentioned here to automate big data. While scaled systems have features such as providing high performance on a regular basis, resources such as bandwidth, memory, CPU and IO utilization are ignored. At this point, the proposed system uses Kubernetes and Docker components as well as a NoSQL-based data storage system to achieve full functionality. Since the data volume will increase due to the increase in the number of devices in the IoT network, Kubernetes Orchestration Tool is recommended to manage and load balance the system to meet the increasing data volume. Docker technology, a powerful tool in virtualization infrastructure, is recommended to orchestrate the management of this proposed system, as it is independent of operating systems and can easily adapt to any system. The main application-based component for combining and using the proposed tools is systematized on a Java application developed with Spring Boot. This application recommends the routing of data from IoT devices through Spring Boot Web to broker tools such as Apache Kafka, RabbitMQ, Apache ActiveMQ and Cassandra database for storing this data. Docker Compose file, which manages the dependencies of the application to be used in local working or remote server working environments, is used to specify the different services of the application and how these services will work together. Thanks to Docker technology and the Compose file, which ports each service will use, which environment variables it will have and its dependencies on other services are determined; this file is referenced for use in Kubernetes technology. Thus, this architecture provides an ideal product environment for applications that process big data and require high availability, as shown in Image 2. Since each service fulfills a specific task and can provide interoperability, it provides sustainable-scalable application functionality in the data processing process written in Java.

Based on the system architecture in Image 2, the following conclusions can be drawn. Kubernetes is a container orchestration system that, through an API (Application Programming Interface), can execute applications that scale dynamically within specified limits or on a Kernel base. This system isolates the libraries used by applications within containers, abstracting them from the operating system. Furthermore, Kubernetes can scale these microservice threads as needed and deploy location-independent applications. The system enables different applications to work together through VLANs. Thus, it manages and streamlines applications. Kubernetes is actually a cluster of master controllers and slave worker nodes that work together to run containers. The master node is the node that manages the workers and pods (PODs) of the current cluster. The applications we want to run are run on this node, called the worker node.

After containerizing the applications, we can implement our application in master and worker clusters through Kubernetes to orchestrate them on a microservice basis, and through the Kubectl Command Line Tool, we can communicate with this cluster and perform functions

such as timed expansion of related applications and running duplicate applications under load. Kubernetes is a tool that organizes the maintenance of systems that need to operate at high capacity, such as IoT. This tool can automatically extend systems when needed and fulfill timed orders. It can also dedicate certain features of hardware to specific applications. Kubernetes can simultaneously run and manage applications on servers in another remote environment in any disaster scenario. These features are managed by the system administrator. It provides scalability and extensibility of applications. In this project, it is proposed to place the Kubernetes tool, which will act as a central brain to direct/manage the execution processes of the application, in the skeleton of the system architecture.

Within the framework of the proposed system, it presents a system design that helps to process and store incoming data in IoT environments where real-time data flow increases instantaneously or where there is a continuously intense data flow, and a performance comparison of the broker systems proposed in this system design.

The working principle of the proposed system starts with the incoming data from IoT devices. The incoming data is first received with the BrokerDataAPI Restful service written in Java Spring Boot Web. This data is transferred to the queue through a system called "Producer", which is parsed according to the desired broker type and customized for each broker type. According to the customized configuration states given to the queue (RabbitMq, Apache ActiveMQ, Apache Kafka), structures called "Consumer" read the data in the queue. The desired operations are performed and stored in the Cassandra database. The designed system is virtualized with Docker and can process data in parallel by creating multiple copies with the Kubernetes tool. Again, the control and load balance distribution of this parallelism is provided by the "Ingress plugin" of the Kubernetes tool. The working principle of this system is shown in Image 2.

4. PERFORMANCE COMPARISONS AND RESULTS

4.1. PERFORMANCE METRICS AND EVALUATION CRITERIA

The Grafana tool is used in the proposed system design to evaluate performance metrics. Grafana open source software provides metrics, system logs, query traces and visualizations stored in any location. Grafana provides tools to transform data into comprehensible visualizations with a time series database (TSDB) [18].

Grafana provides many metrics and visualizations to monitor and analyze the system performance as described in the contributors' and officially published description. It allows us to obtain metrics such as how much bandwidth, memory utilization, processor utilization, disk utilization, etc. different brokers of the proposed system design use in the system, and reveals the purpose of the proposed system and the researcher. It also provides comparison metrics.

As evaluation criteria, values such as bandwidth utilization, memory utilization and CPU utilization are considered on Grafana. With the data coming from the IoT device, the recording time is taken into account and the values shown in Image 2 are extracted within the framework

of the processing time criteria, which will help us in our goal of determining which broker the proposed system works better on.

4.2. PERFORMANCE COMPARISONS

This section presents the metrics used to compare the performance of different Message Broker designs and the results obtained. According to our results, the most striking result is that the number of parallel runs of the broker systems, the number of computational operations on the incoming data and the processing time at the point of access to the database (TimeToFindNearPoint) have a bottlenecking effect according to the given configurations.

4.2.1. METRICS USED IN THE COMPARISON

Here is a one sentence description of each column in the table:

Id: Unique number of randomly generated location information for the broker to process.

City Name: An indication of which city the data belongs to.

Neighborhood Name: Information indicating the name of a specific neighborhood or district within the city.

Latitude: A numeric value representing the coordinate of a specific location on the north-south axis.

Longitude: A numeric value representing the coordinate of a specific location on the east-west axis.

Created Date: A timestamp indicating the date and time the data was created.

Broker Type: An indication of which broker technology was used to receive or publish the data.

Near Point Id: The other Location Id information that the randomly generated location information is close to.

Time to Find Near Point Id: Time in milliseconds indicating how far away a randomly generated location is from another location and how long it took to process the Near Point Id value.

Test Group Id: The unique alphanumeric value of the test groups used to differentiate scenarios in bulk data submission during comparison.

4.2.2. DEFINING THE BENCHMARK ENVIRONMENT AND TEST CASES

Since metrics such as data processing time, memory utilization, CPU utilization will also be affected by the features of the system used in the benchmarking process, the proposed broker system was integrated on an SSD system with 32 GB Memory, 12th Gen Intel(R) Core(TM) i7-12700H 4.7GHz, 5000mb/s read/write speed. In order to isolate the processing time and other usage metrics of the broker systems with other broker systems in the benchmarking processes, the test and broker system isolation is provided with the "BROKERTYPE" parameter with the Compose file of the Docker Container system in our core application, and the metrics to be

obtained are aimed to be independent and reliable. The processing time of the algorithm for finding the closest location to the latitude and longitude data, which will calculate the distance of each location data read by the broker systems to the other location data, is specified as the "Time to Find Near Point Id" column in the test process in order to ensure that the test processes in each broker type provide reliable and accurate results. The purpose of this algorithm is to measure the impact of each message on the processing time in the broker systems with an independent parameter. In addition, since each broker system operates on the same database in different time periods, the Cassandra database was run in 2 copies during the Type 1 and Processing time comparison to ensure that the metrics obtained are not affected by the data processing and maximum number of messages that the broker systems can transfer. In addition, in order to simulate IoT systems, Apache JMeter test tool was used to specify the number of IoT devices accessing the core application at the same time with the Thread Group system and the number of data coming from the same IoT device at the same time was kept constant as 10 and the "Number Of Threads" number was kept constant as 10 and the "Ramp-Up Periods" value as 1 in order to send 10 data every second.

In order to compare different broker systems, 4 different test scenarios were defined;

1- Broker type and processing time of each IoT data
2- Number of broker copies and maximum number of message transfers
3- The effect of the number of database copies on the broker system by increasing the number of Cassandra copies of the broker system that gives the 1st test result with the best performance
4- Impact on message processing time and maximum number of transfers by parallel execution of Broker core code with Thread routine

In addition, it is desired to reach the most performant and usable broker system by presenting usage metrics based on bandwidth, memory usage, CPU usage of each test number with Grafana around the conternization of the test scenarios and broker designs defined above.

4.3. PERFORMANCE COMPARISON RESULTS

Chart 1. Broker type and processing time of each IoT data

Broker Type	***Near Point Find Avarage / Yakınlık Derecesi Bulma Hız Ortalaması***	***Broker Core Application Replication / Ana broker uygulamasının kopya sayısı***	***Broker Application Replication / Broker uygulamasının kopya sayısı***
ACTIVEMQ	171 Milisaniye	1	1
RABBITMQ	486 Milisaniye	1	1
KAFKA	260 Milisaniye	1	1

The results in Chart 1 are obtained by defining the "BROKER_TYPE" environment variable defined in the core application for the generation and transfer of 10,000 falsely generated location information to the broker for each broker type and for each broker test. In addition, the description of each environment variable defined on the core application is shared on the project as an open source ReadMe marked file [18].

There are processes such as processing location data from IoT etc. devices in artificial intelligence modules or LLM modules. In this proposed system, the Near Point Find Avarage (Near Point Find Avarage) value has been created to mimic these processes. With this value, the performance of the proposed system is presented with the processing time of each IoT data. It is observed that the queuing and processing time of the data in the RabbitMQ broker model gives a considerably longer processing time compared to other broker systems. At maximum performance, it is determined that ActiveMQ broker system provides faster access to data than RabbitMQ broker system in terms of the number of message transfers.

Number of broker copies and maximum number of message transfers

Broker Type	***Broker Kopya Sayısı***	***Maksimum Performansta Mesaj Transfer Sayısı***
ACTIVEMQ	1	82.000
RABBITMQ	1	100.000
KAFKA	3	310.000

Chart 3. Number of broker copies and maximum number of message transfers

Broker Type	***Broker Kopya Sayısı***	***Maksimum Performansta Mesaj Transfer Sayısı***
ACTIVEMQ	2	95.000
RABBITMQ	2	120.000
KAFKA	4	360.000

Although there is no limit to the maximum number of message transfers in each broker, depending on the load on the broker systems, the time it takes to transfer messages may be longer. This time may vary depending on the protocols, design patterns, queuing models, priority types of the queue, etc. of the broker designs. With this comparison, the number of message transfers that brokers can receive at maximum performance according to the message reading times of the brokers according to the increasing message volume by not going beyond the proposed broker systems is determined, and the management panels offered by the brokers are utilized in this determination.

Chart 4. The effect of the number of database copies on the broker system by increasing the number of cassandra copies of the broker system with the 1st best performance test result

Best Effort Broker Type	*Cassandra Replication*	*Near Point Find Avarage*
ACTIVEMQ	3	98.57 Milisaniye

As the number of copies of Cassandra's database implementation increases, the processing time decreases, allowing the load to be better distributed, reducing the processing overhead. This is due to the fact that Cassandra keeps the data in Partition. Since each copy sends data in different columns, it allows for faster processing. In addition, unlike other NoSQL databases, Cassandra does not have a single "Writer Node" and each copy can take both write and read tasks, allowing writes to be made to each copy without delay.

These features allow for a more efficient distribution of the processing load and reduce the data processing time of each replica. As a result, the increased number of database copies of Cassandra contributes to lower overall processing times and higher performance. These features offer significant advantages for faster and more efficient performance in data access and transactions in distributed database systems.

Chart 5. The effect of parallel execution of the Broker core code with the Thread procedure on message processing time and maximum number of transfers

Broker Type	*Near Point Find Avarage / Yakınlık Derecesi Bulma Hız Ortalaması*	*Broker Core Application Replication / Ana broker uygulamasının kopya sayısı*	*Broker Application Replication / Broker uygulamasının kopya sayısı*	*Message Count / Mesaj Transfer Sayısı*
ACTIVEMQ	140 Milisaniye	2	1	10.000
RABBITMQ	376 Milisaniye	2	1	10.000
KAFKA	214 Milisaniye	2	1	10.000

Examining the effect of parallel execution of the broker core code via Thread on the message processing time and the maximum number of transfers, we observe a significant improvement in the metrics obtained. Depending on the size of the main broker application, i.e. the application where the queues are consumed, parallel processing reduces the processing time. This is due to the fact that the kernel code can process multiple threads at the same time. Processing each message in a separate thread allows processing times to be executed in parallel. This reduces the total processing time.

It is also expected to observe a significant increase in the maximum number of transfers. However, since the focus is on the change of the average speed, this parameter is kept constant. The reason for the increase is that with parallel processing, more messages can be processed and transferred at the same time. This increases the maximum transfer capacity of the system.

After the system utilization metrics of the test scenarios are obtained visually through Grafana, the data is digitized and made suitable for comparison as shown in Chart 6 to show that the proposed system offers the highest capacity at the lowest utilization by obtaining the hardware metrics used to reach the most suitable broker system for IoT environments in today's systems.

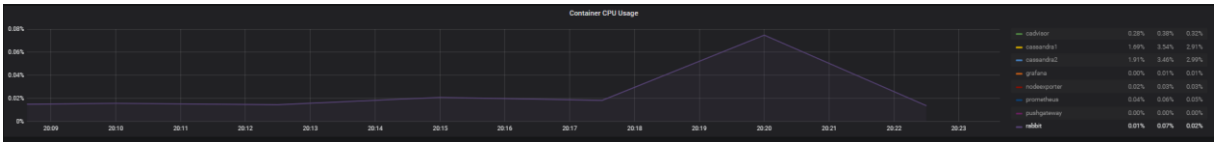

Image 3. Memory - CPU - Bandwidth Utilization of the Proposed Systems, with Grafana

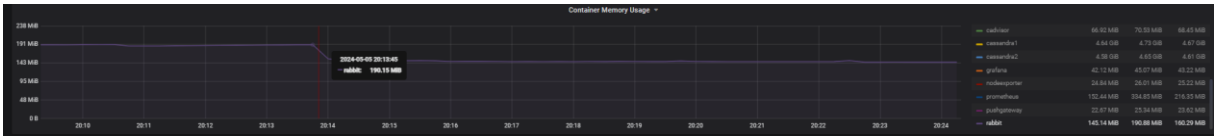

Image 4. Memory - CPU - Bandwidth Utilization of the Proposed Systems, with Grafana

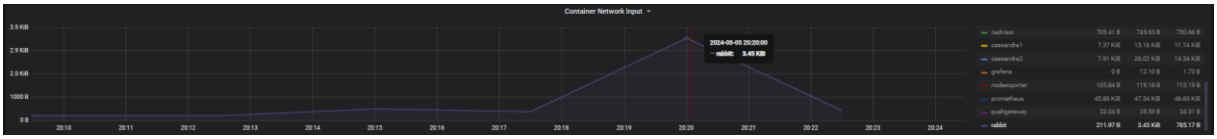

Image 5. Memory - CPU - Bandwidth Utilization of the Proposed Systems, with Grafana

Chart 6. System Utilization Values

Tablo Test Numarası	*Broker Tipi*	*Bant Genişliği Kullanımı (Avg.)*	*Hafıza Kullanımı(Avg.)*	*CPU kullanımı(Avg.)*
1	ACTIVEMQ	14.62 Kib	554.60 Mib	%1.05
1	RABBITMQ	556.53 Kib	179.30 Mib	%0.02
1	KAFKA	575.39 Kib	1.712 Mib	%0.28
3	ACTIVEMQ	28.33 Kib	680.38Mib	%1.96
3	RABBITMQ	953.68 Kib	245.34 Mib	%0.05
3	KAFKA	895.78 Kib	2.485 Mib	%0.36
4	ACTIVEMQ	15.42 Kib	542.71Mib	%1.04

As a result of the comparative tests, significant results were obtained on both the processing time and the maximum number of data that broker systems can transfer for IoT systems that require instant data processing, post-processing IoT systems that perform intensive data flow, and potential asynchronous messaging processes. The metric values referenced in the tables are shared on the public figure [19].

According to the results, it was observed that as the number of copies of each test case increases, the maximum number of data that can be transferred can be increased and the average processing time decreases. The parallel execution of the core application within the framework of the proposed system, both with Spring Framework support and with Reactive Programming mechanisms developed by the application developer, has a significant positive impact on the performance of the proposed system. By evaluating the system utilization metrics created by the proposed high performance and extensible broker system, the values in Chart 6, which are obtained by evaluating which hardware metrics the proposed systems affect, reveal the requirements of the environments in which the proposed systems can be run.

In our proposed system design, the results show that the RABBITMQ broker system can process faster and more data than other broker systems, uses less system requirements, and outperforms other systems in terms of processing performance/requirement values.

As a result of the findings of this study, it is proved that the broker system proposed in the paper "Efficient and Scalable Broker Design for the Internet of Things Environments" by Yasin

Görmez et al. [20] can be improved and more performance and data loss-resistant system designs that are faster in accessing data can be proposed.

5. CONCLUSION AND FUTURE WORK

This paper addresses an important problem in the Internet of Things (IoT) domain and investigates the performance of different broker designs. The results obtained can contribute to many scientific researches. It clearly demonstrated the impact of various broker designs on trading times. An analytical and interactive method was used to compare the performance of these designs. Visualizations were made available through Grafana. This enabled a meaningful evaluation of different broker designs.

In terms of future work, one could focus on the gaps and potential improvements in this research area. For example, more broker designs could be studied and their performance evaluated in different scenarios. More data could be collected and comparisons could be made simultaneously. Also, more data could be collected on how these broker designs perform in real-world applications. However, this study was limited to the analysis on a NoSQL database, so the study of relational databases and other data storage methods can also be considered in future research.

In conclusion, this study focused on an important problem in the IoT domain and contributed to research in this area by comparing the performance of different broker designs. Future work can increase the impact of this research by expanding knowledge in this area and focusing on new, powerful technologies.

SOURCE

[1] https://aws.amazon.com/tr/what-is/iot/ - (Date of Access: 01.05.2024)

[2] https://acikerisim.sakarya.edu.tr/bitstream/20.500.12619/79673/1/T08082.pdf - (Date of Access: 05.05.2024)

[3] https://gsl.com.tr/mqtt-endustriyel-nesnelerin-interneti-uygulamalarinda-verimli-ve-esnek-bir-haberlesme-yapisi.html - (Date of Access: 01.05.2024)

[4] https://aws.amazon.com/tr/what-is/mqtt/ - (Date of Access: 01.05.2024)

[5] https://www.tercihyazilim.com/blog/http-nedir – (Date of Access: 25.04.2024)

[6] https://appmaster.io/tr/blog/websockets-nedir-ve-nasil-olusturulur – (Date of Access: 25.04.2024)

[7] https://www.hosting.com.tr/bilgi-bankasi/http-2-nedir-ne-ise-yarar/ - (Date of Access: 25.04.2024)

[8] https://prodiot.com/tr/coap-protokol-ozellikleri-nedir-/article/6 – (Date of Access: 26.04.2024)

[9] https://www.ozztech.net/genel/message-broker-nedir-2/ - (Date of Access: 05.05.2024)

[10] https://www.ariteknokent.com.tr/tr/haberler/milsoft-seminerine-davetlisiniz - (Date of Access: 26.04.2024)

[11] https://tr.wikipedia.org/wiki/XMPP – (Date of Access: 26.04.2024)

[12] https://www.rabbitmq.com/ - (Date of Access: 26.04.2024)

[13] https://kubernetes.io/ - (Date of Access: 05.05.2024)

[14] https://activemq.apache.org/ - (Date of Access: 26.04.2024)

[15] https://kafka.apache.org/ - (Date of Access: 26.04.2024)

[16] https://cassandra.apache.org/_/index.html – (Date of Access: 26.04.2024)

[17] https://docs.docker.com/build/cloud/ - (Date of Access: 05.05.2024)

[18] https://grafana.com/docs/grafana/latest/introduction/ - (Date of Access: 05.05.2024)

[19] https://github.com/sogz/bigdatabrokersystem/ - (Date of Access: 07.05.2024)

[20] Y. Görmez, H. Arslan and Ö. F. Kelek, "Efficient and Scalable Broker Design for the Internet of Things Environments," 2020 28th Signal Processing and Communications Applications Conference (SIU), Gaziantep, Turkey, 2020 (Date of Access:11.05.2024)