FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING
DEGREE PROGRAMME IN ELECTRONICS AND COMMUNICATIONS ENGINEERING

MASTER'S THESIS

DISTRIBUTED TRUSTWORTHY SENSOR DATA MANAGEMENT ARCHITECTURE

Author:	Niko Haiko
Supervisor:	Erkki Harjula
Second Examiner:	Tanesh Kumar
Technical Advisor:	Tanesh Kumar

June 2022

Haiko N. (2022) Distributed Trustworthy Sensor Data Management Architecture. University of Oulu, Faculty of Information Technology and Electrical Engineering, Degree Programme in Electronics and Communications Engineering, 40 pages.

ABSTRACT

Growth in Internet of Things (IoT) market has led to larger data volumes generated by massive amount of smart sensors and devices. This data flow must be managed and stored by some data management service. Storing data to the cloud results high latency and need to transfer large amount of data over the Internet. Edge computing operates physically closer to the user than cloud, offering lower latency and reducing data transmission over the network. Going one step forward and storing data locally to the IoT device results smaller latency than cloud and edge computing.

Utilizing isolation technique like virtualization enables easy to deploy environment to setup the needed software functionalities. Container technology works well on lightweight hardware as it offers good performance and small overhead. Containers are used to manage server-side services and to give clean environment for each test run.

In this thesis two data management platforms, Apache Kafka and MySQL-based MariaDB are tested on a IoT platform. Key performance parameters considered for these platforms are latency and data throughput while also collecting system resource usage data. Variable amount of users and payload sizes are tested and results are presented in graphs. Kafka performed similarly to the SQL-based solution with small differences.

Keywords: Container, Data management, Event streaming, IoT, Latency, Throughput, Virtualization

Haiko N. (2022) Hajautettu luotettava anturidatan hallintajärjestelmä. Oulun yliopisto, tieto- ja sähkötekniikan tiedekunta, elektroniikan ja tietoliikennetekniikan tutkinto-ohjelma. Diplomityö, 40 sivua.

TIIVISTELMÄ

IoT-markkinoiden kasvu on johtanut suurempien datamäärien luontiin IoT-laitteiden toimesta. Tuo datavirta täytyy hallita and varastoida datan käsittelypalvelun toimesta. Datan tallennus pilvipalveluihin tuottaa suuren latenssin ja tarpeen suurien datamäärien siirrolle Internetin yli. Fyysisesti lähempänä loppukäyttäjää oleva reunapalvelu tarjoaa pienemmän latenssin ja vähentää siirrettävän datan määrää verkon yli. Kun palvelu tuodaan vielä askel lähemmäksi, päästään paikalliseen palveluun, mikä saavuttaa vielä pienemmän latenssin kuin pilvi- ja reunapalvelut.

Virtualisointitekniikka mahdollistaa helposti jaettavan ympäristön käyttöönottoa, mikä mahdollistaa tarvittavien ohjelmiston toimintojen asennuksen. Virtualisointitekniikoista kontit nousivat muiden edelle, koska IoT-laitteet omaavat suhteellisesti vähän muistia ja laskentatehoa. Kontteja käytetään palvelinpuolen palveluiden hallintaan sekä tarjoamaan puhtaan vakioidun ympäristön jokaiselle testikierrokselle.

Tämä diplomityö käsittelee kahden tiedonhallinta-alustan: Apache Kafka ja MySQL pohjaisen MariaDB-tietokannan suorituskykyeroja IoT-alustan päällä. Kerätyt suorituskykymittaukset ovat latenssi ja tiedonsiirtonopeus mitaten samalla järjestelmän resurssien käyttöasteita. Vaihtelevia määriä käyttäjiä ja hyötykuormia testataan ja tulokset esitetään graafeissa. Kafka suoriutui yhtä hyvin kuin SQL ohjelmisto näissä testeissä, mutta pieniä eroja näiden välillä havaittiin.

Avainsanat: Datan hallinta, Kontti, Tapahtumavirtaus, IoT, Latenssi, Tiedonsiirtonopeus, Virtualisointi

TABLE OF CONTENTS

ABSTRACT
TIIVISTELMÄ
TABLE OF CONTENTS
FOREWORD
LIST OF ABBREVIATIONS AND SYMBOLS
1 INTRODUCTION 8
1.1 Thesis Structure 8
2 RELATED WORK 10
2.1 Internet of Things 10
2.1.1 Use Cases 10
2.1.2 Hardware Resources for IoT 11
2.1.3 Software Solutions for IoT 12
2.1.4 Communication Protocols for IoT 12
2.2 Distributed Ledger Technology 14
2.2.1 Blockchain 15
2.2.2 Blockchain Platforms 15
2.3 Databases 15
2.3.1 Relational Database 16
2.3.2 Non-Relational Database 17
2.3.3 Event Streaming 17
2.4 Virtualization 18
2.4.1 Desktop Virtualization 19
2.4.2 Service Virtualization 19
2.4.3 Containers 20
3 SYSTEM DESIGN 21
3.1 Test Hardware 21
3.2 Test Software 21
3.3 Test Setup 22
4 EVALUATION 24
4.1 Latency 24
4.2 Throughput 28
5 DISCUSSION 33
5.1 Future Work 33
6 SUMMARY 35
7 REFERENCES 36

FOREWORD

Thesis was done with help of Center for Wireless Communications - Networks and Systems (CWC-NS) research group in University of Oulu. I would like to thank my supervisors, assistant professor D.Sc. Erkki Harjula and D.Sc. Tanesh Kumar for technical support and keeping me right track with the idea of the thesis.

Oulu, June 21st, 2022

Niko Haiko

LIST OF ABBREVIATIONS AND SYMBOLS

3GPP	Third Generation Partnership Project
6LoWPAN	IPv6 over Low-Power Wireless Personal Area Network
AES	Advanced Encryption Standard
API	Application Programming Interface
BLE	Bluetooth Low Energy
CPU	Central Processing Unit
CoAP	Constrained Application Protocol
DBMS	Database Management System
DDL	Data Definition Language
DL	Distributed Ledger
DLT	Distributed Ledger Technology
Dapps	Distributed applications
ETH	Cryptocurrency Ether
EVM	Ethereum Virtual Machine
Gb	Gigabit
GB	Gigabyte
GPIO	General-Purpose Input/Output
GPU	Graphical Processing Unit
GSM	Global System for Mobile communications
HDMI	High-Definition Multimedia Interface
HTTP	Hyper Text Transfer Protocol
I/O	Input/Output
IaaS	Infrastructure as a Service
IoT	Internet of Things
IIoT	Industrial Internet of Things
KVM	Kernel Virtual Machine
LAN	Local Area Network
LPWAN	Low-Power Wide-Area Network
LTE	Long-Term Evolution
LXC	Linux Containers
LoRaWAN	Long Range Wide Area Network
MQTT	Message Queuing Telemetry Transport
NIC	Network Interface Controller
NoSQL	Non-relational database
OS	Operating System
OSI	Open System Interconnections
PAN	Personal Area Network
PHY	Physical Layer
Paas	Platform as a Service
PoW	Proof of Work
QoL	Quality of Life
RAM	Random-Access Memory
RFC	Request for Comments
RFID	Radio Frequency Identification

RJ45	Registered Jack 45
SBC	Single-Board Computers
SC	Smart Contract
SD	Secure Digital
SQL	Structured Query Language
SaaS	Software as a Service
TCP	Transport Control Protocol
UDP	User Datagram Protocol
USB	Universal Serial Bus
UTC	Coordinated Universal Time
VDI	Virtual Desktop Infrastructure
VM	Virtual Machine
WLAN	Wireless Local Area Network
mW	Milliwatt
ms	Millisecond
us	Microsecond

1 INTRODUCTION

Internet has become colossal collection of networked devices that are able to communicate between each other. In a traditional sense we have home and work computers connected to servers across the Internet but it was only the beginning. Based on StatCounters Global stats in year 2021 mobile devices hold majority of web traffic market share with 57% compared to desktop computers at 42% leaving rest to tablet devices [1]. This means that small devices are big part of the current Internet.

Mobile devices, such as a smart phone or a tablet, are almost like a IoT device, but as they are designed to primarily handle human to machine connections, they are not directly considered to be IoT devices. Many simple wireless sensor devices are connected directly to the Internet making them an example of IoT devices. Rising usage of IoT leads to more data transmissions around us requiring different ways of managing the data. [2]

IoT environment have become more dynamic in terms of efficient program execution requiring dynamic response from the processing chain like edge, fog and cloud. This data flow needs to be managed and controlled in an efficient manner to keep the data flowing without any interruptions. Event stream processing came up as one of those technologies that are able to manage this kind of data flow. Event streaming offers way to split single monolithic application into chain of processes that can be distributed among multiple devices offering horizontal scalability. This kind of processing could be done in IoT realm reducing dataflow to the upstream. [3]

Industry have utilized IoT in form of Industrial Internet of Things (IIoT). IIoT is more machine-centric approach to the IoT environment than consumer IoT, which is more closely related to human interaction. Machine-centric approach tries to minimize human interaction between the machine and a human, generating autonomous environment to increase productivity. IIoT machines have stricter requirements in latency, reliability and security than consumer IoT machines like smart fridge.[4]

Usage of remote healthcare systems has been rising in the past decade bringing more business opportunities. Remote healthcare comes with its problems innate to the remote part, latency. Sometimes in healthcare it's critical that sensed data can be processed as fast as possible to avoid possible medical complications. Reducing this latency could be done in many ways but most effective is to cut the physical distance between a client and initial data processing. [5]

By offering remote service there are several security and trust concepts that need to be taken care of. Healthcare data is privacy centric and shouldn't be shared with a third party. This requires trustworthy data management to keep data private and immutable. Usage of distributed systems would reduce physical distance from server to user by distributing servers closer to the users. Offering data closer to the user would reduce the latency and require distributed platform technology like event streaming platform to streamline the data. [6]

1.1 Thesis Structure

Chapter 2 introduce related and used technologies in this project. Related information of IoT devices and their connections are presented. Technologies like blockchain and Distributed Ledger Technology (DLT) are considered in contrast of chosen event streaming platform.

System design, hardware and software configurations are presented in Chapter 3. Used devices and settings are laid out and chain of events in test process is explained. Main

focus is making possible to compare performance of Apache Kafka compared to MySQL-based MariaDB database. These tests are very simple and aiming to show possible differences between the platforms in resource usage, throughput and latency.

Test results are presented in Chapter 4 and are discussed in Chapter 5. Results are shown in graphical form to give easily comparable information between the platforms and possible deviations are discussed. Results are summarised in Chapter 6.

2 RELATED WORK

This chapter covers topics about the related work. General concepts are presented, and different possibilities explored.

2.1 Internet of Things

Internet of Things (IoT) is an environment, where different objects are able to sense and communicate with their surroundings. This can be interpreted as collection of networked objects that are able to send, receive or even process the data. IoT have become widely used term for networked autonomous devices. As they are networked, they need to be identifiable to know where the data is coming from. IoT device is able to sense surrounding environmental parameters by utilizing various sensors attached to it, for example change in temperature. This data can then be sent to other device or service to be processed usually to some cloud service. After data is processed and analysed the results can be offered on user friendly service to provide nice, ordered view of the data. [7]

IoT device can range from a networked coffee maker to traffic light controller. IoT devices are meant to improve everyday life in multiple forms as collecting sensor data, controlling systems or displaying information. Improving Quality of Life (QoL) by design have made this technology desired to wider audience. [8]

2.1.1 Use Cases

Many different use cases of IoT devices leads to big variation of IoT devices. One of the simplest cases is using the device to manage some sensor, collect the data and send it for processing. One use case for this kind of sensor devices is supplying weather data from multiple points in a large area. Figure 1. opens this by visually showing common application areas.

Smart home IoT would be one of the most visible parts of the IoT infrastructure. Basic home appliance like washing machine could be one of those. Smart washing machines for example can adjust needed detergent automatically based on the clothes inside the machine. Adding these machines to the local network or the Internet gives ability to monitor the status of the machines remotely and even start the machines over the network. This also enables possibility to collect information about the smart device for further product development. [10]

Smart factory utilizes IoT in many parts of the processing. Basic sensor metrics like temperature, humidity and dustiness of the environment can be detected by sensors leading into certain action based on the results. Access restriction and access control are common practices in factories, where IoT device could provide Radio Frequency Identification (RFID) sensing and processing for workers to gain access into the building. Machines itself can be networked and controlled remotely while also collecting useful metrics about the process. [11]

In healthcare IoT can be used for remote healthcare sensors to provide up to date information about the situation. This information can be used to automatically alarm professionals if sensor data exceeds certain limits. Getting instant feedback helps to solve the problem faster. There are also multiple other use cases for IoT that are not covered here. [12]

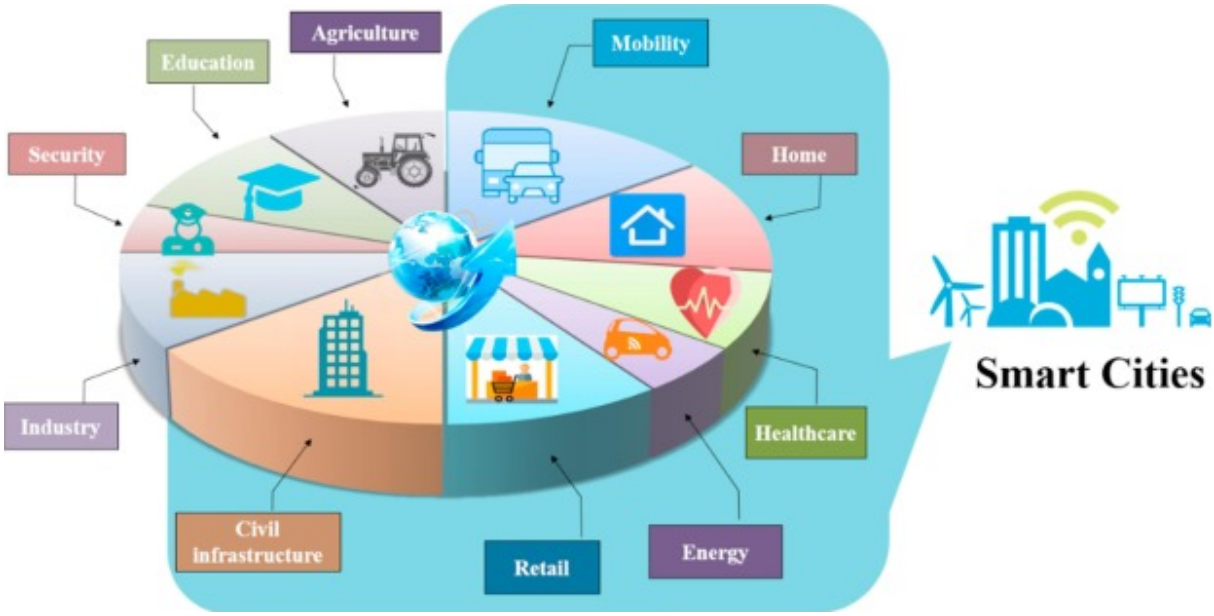

Figure 1. IoT use cases. [9]

2.1.2 Hardware Resources for IoT

There's big variety of different types of IoT devices. Hardware resources like processing power, memory and network connectivity can be very different between two IoT devices. Networking is one of the main things where wireless connectivity enables more open device placement and use case compared to wired connection. Using wireless connectivity gives opportunity to build system on battery leading to low power requirements. Low power requirements mean these devices are usually small and have limited capacity to do what is required keeping power consumption as low as possible.

Some common Single-Board Computers (SBC) are Raspberry Pi family, Arduino boards and ESP development boards. All these products offer General-Purpose Input/Output (GPIO) pins to connect other devices or sensors to the system. Some form of connectivity is available in wired or wireless form, wireless usually using Wi-Fi or Bluetooth. These types of development boards offer fast way of testing and building IoT system. [13][14][15]

Raspberry Pi is one of the most common SBC as it has interfaces to connect common peripherals. Raspberry Pi product line is currently in version 4 and the B model of this product is shown in Figure 2. This SBC can also work as a standalone desktop computer as it has Graphical Processing Unit (GPU) and micro High-Definition Multimedia Interface (HDMI) to deliver the picture. Also, other common interfaces like Universal Serial Bus (USB) ports for keyboard and mouse, Registered Jack 45 (RJ45) port for physical ethernet and wireless module for Wireless Local Area Network (WLAN) enable desktop computer experience. As Raspberry Pi also has GPIO interface, it can be used as simple standalone solution to test and develop a IoT solution. [16]

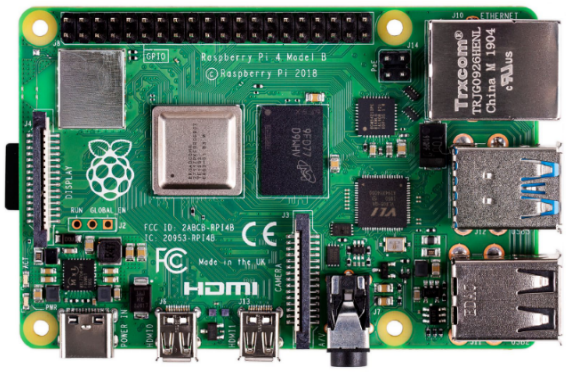

Figure 2. Raspberry Pi 4 B. [16]

2.1.3 Software Solutions for IoT

IoT devices tend to have restricted amount of Random-Access Memory (RAM). This greatly limits amount and types of software what can be run on specific device especially Operating Systems (OS). For general use it makes sense to have an operating system running to offer easy access to system devices, manage system resources and schedule events. Drawback is that not all devices have the luxury to run full operating system because of disk space or RAM limitations. This leads to more compact OS called embedded OS, or no operating system at all called bare metal.

Embedded OS is system that is limited to do certain tasks. This limitation comes from the fact that the system hardware is not capable running full desktop OS. Compared to regular desktop OS, embedded OS provides basic interfaces to handle at least memory management, Input/Output (I/O) and provide interrupts. [17]

When system resources are so limited that OS consumes most of available resources or induces too much latency, removal of the OS should be considered. More lightweight option is to run the code without OS, meaning that code would be running against bare metal. Bare metal means that raw compiled code is running on the limited device with bare minimum resource usage to fit in the device specs. This also gives opportunity to optimize the system performance to work without overhead of the OS. Downside of this approach is more complicated development and maintenance as there is no supporting OS. [18]

2.1.4 Communication Protocols for IoT

IoT devices can use traditional communication protocols for general purpose communications, but limited local resources require small protocol overhead. This leads to adaptation of certain types of protocols in IoT usage. Open System Interconnections (OSI) model of the communication protocols is shown in Figure 3. There we can observe high-level protocols working in the application layer and low-level protocols in the physical to network layer. For simplicity physical and data link layer are bundled together as many protocols shown provide both.

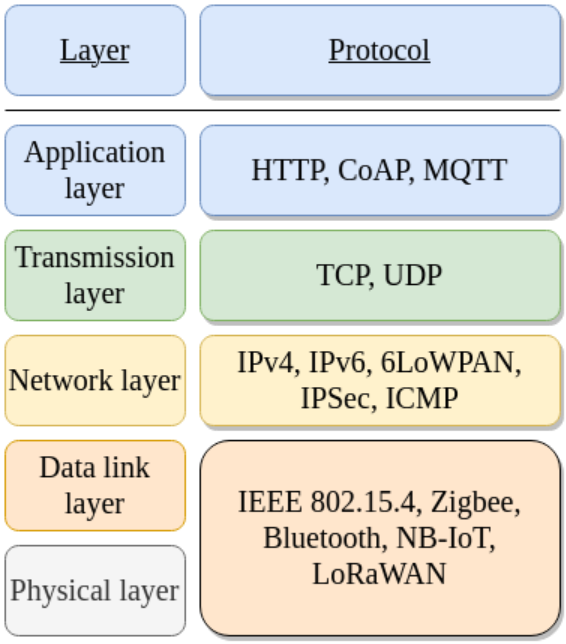

Figure 3. OSI layers of IoT protocols.

Defined protocols in OSI transmission layer are Transport Control Protocol (TCP) and User Datagram Protocol (UDP). TCP generates more overhead compared to UDP by confirming package delivery and keeping package order. This means TCP has many features that are not needed in all scenarios, making UDP more lightweight solution for IoT projects. Both protocols can be used in IoT environment like Message Queuing Telemetry Transport (MQTT) uses TCP and Constrained Application Protocol (CoAP) is built over UDP. [19]

Low-level protocols

There are several low-level communication protocols to help low power IoT devices to work longer and more efficiently. Network, data link and physical layers in OSI model are considered low-level protocols here. These protocols enable IoT devices to join nearby network or directly to the internet. Having low-level network connection allows IoT devices to share their data or receive new commands. Considering nature of the devices, physical data connections are usually made wirelessly utilizing several different wireless transmission standards.

IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) is one of these wireless low-level protocols defined in Request for Comments (RFC) 8138. 6LoWPAN works in adaptation layer between network and data link layer. It utilizes low-level protocol IEEE 802.15.4 for Physical Layer (PHY) and data link layer to provide low power wireless connectivity. From security perspective, data is encrypted with Advanced Encryption Standard (AES) 128. [20]

ZigBee also uses IEEE 802.15.4 base at its core. Unlike in 6LoWPAN, ZigBee does not provide IP networking, so it requires different devices like, ZigBee end devices, ZigBee network coordinators and ZigBee routers to build the infrastructure in order to support the network. Zigbee can have throughput up to 250 kpbs with only 10 to 100 milliwatts (mW) of power. This

low power limits the range of the technology to around 100 meters. Zigbee communication is also encrypted with AES 128. [21]

Bluetooth Low Energy (BLE) is special subcategory in Bluetooth family containing as name suggests low energy transmission protocol. BLE is designed to work in shorter range up to 10 m meaning that it is highly useful in Personal Area Network (PAN). Bluetooth 5 standard can offer up to 2Mb unencoded transmission bandwidth. [22]

Narrowband Internet of Things (NB-IoT) is a standard published by Third Generation Partnership Project (3GPP) operating in Low-Power Wide-Area Network (LPWAN) area. NB-IoT is transmitted over Global System for Mobile communications (GSM) or Long-Term Evolution (LTE) carrier. This enables long range connectivity to IoT devices with relatively low power consumption. [23]

Another long-range communication technology is Long Range Wide Area Network (LoRaWAN) defined by Lora Alliance. It is designed to offer long range connectivity to IoT devices. Usual single user bandwidth is around 0.3-50 Kbps. Range can potentially be tens of kilometres depending on the terrain. [24]

High-level protocols

High-level protocols are defined here as an application layer protocols in OSI model. High-level protocols offer services over the low-level protocols. Low-level protocols are used to transmit the data and high-level protocols are used to offer the service. High-level communication protocol, such as Hyper Text Transfer Protocol (HTTP), is a common way of transferring information over internet. IoT devices on the other hand have low processing power and memory leading to more compact application layer protocols. Widely adapted lightweight high-level protocols are CoAP and MQTT. [25]

Core protocol of CoAP is defined in RFC 7252. CoAP is generic web protocol aimed to provide more constrained communication than HTTP. It is designed to keep message overhead small in order to make packet sizes compact. Compact packet sizes reduce packet fragmentation when multiple devices communicate over restricted network. This enables resource constrained devices to more likely send the packets without fragmentation, saving bandwidth usage. [26]

MQTT is defined by open OASIS. It is publish-subscribe protocol to transfer data between nodes. In MQTT architecture server is called broker and publishers and subscribers are clients. Publisher could be thermometer and subscriber an info screen showing a temperature. Broker is the server between to handle connections. [27]

2.2 Distributed Ledger Technology

Distributed Ledger Technology (DLT) covers technologies using Distributed Ledger (DL) to provide decentralized database structure. Traditionally ledgers are centralized, but DLT offers way to decentralize the process while keeping the data trustworthy. Data is distributed and synchronized across multiple nodes in the network. As there's is no central administration, all participants in the network must come to agreement on singular truth. This is called consensus protocol provided by various consensus algorithms. [28]

In order to enable the trust between participants in the DLT network and participating nodes in the network should agree on a same version of truth. This is done by the consensus protocol making it a keystone of any DLT. There exists multiple different consensus protocols depending

on the need. One of the most used consensus protocol is Proof of Work (PoW), which requires some computational calculations to be done in order to validate the work. [29]

2.2.1 Blockchain

Blockchain is one of the most known and widely used DLT. In the blockchain, each block has knowledge of previous blocks, making a chain of trust. This chain is trusted since editing any previous block would require updating all subsequent blocks. Updating multiple blocks and gaining consent is an overwhelming task, which makes it hard to counterfeit data in a blockchain. First block in the chain is called Genesis block. This block has a unique name because it does not have a predecessor block to refer. [30]

Blockchains can be divided into three main categories based on privacy: public, private and permissioned. Public blockchain is open to everyone. It is free to use by the participants but keeping up the distributed ledger require high power consumption in terms of mining, like a public blockchain Bitcoin. In private blockchain each participant needs an invitation in order to join the network. This means that all participants are known to the network and selected by network administrators. Third option, permissioned blockchain is a mix of public and private blockchains. Certain groups of participants can be given certain permissions to use the network. [31]

Smart Contract (SC) is an automatic computer program inside a blockchain waiting for a condition to be met. This allows machines to automatically exchange data based on the contract integrating business side to the blockchain. SC could be a financial agreement between two parties on a price of product. SC would be automatically invoked when one party wants to make a transaction. [30]

2.2.2 Blockchain Platforms

Ethereum works as an open software platform on a blockchain offering its own cryptocurrency Ether (ETH), Distributed applications (Dapps) in form of a smart contract and Ethereum virtual machine (EVM). Ethereum networks has public and private option. Cryptocurrency ETH is used in the public side to verify all transactions. If developer opts to create private Ethereum, network transaction cost can be reduced or left out. [32]

Hyperledger fabric is part of Hyperledger project backed up by multiple companies including Linux Foundation. Hyperledger fabric is DLT platform designed for enterprise usage. Being enterprise-grade, it offers private or permissioned way of handling the DLT. [33]

2.3 Databases

Databases offer a way to store collection of data, such as financial records, inventory catalogues or even images. Commonly databases are managed by Database Management System (DBMS) that offers controlled access to the database itself. Database is a wide term that can be split into multiple different subcategories based of database type as visualized in Figure 4. Most common database type is relational database with Structured Query Language (SQL) as a querying language. Other common type is a Non-relational database (NoSQL). [34]

Usage of distributed databases provide way to scale the database size across multiple different computers. These machines could be close by or far away connected over networks. This also provides additional redundancy as not all data is stored on exactly same place. [34]

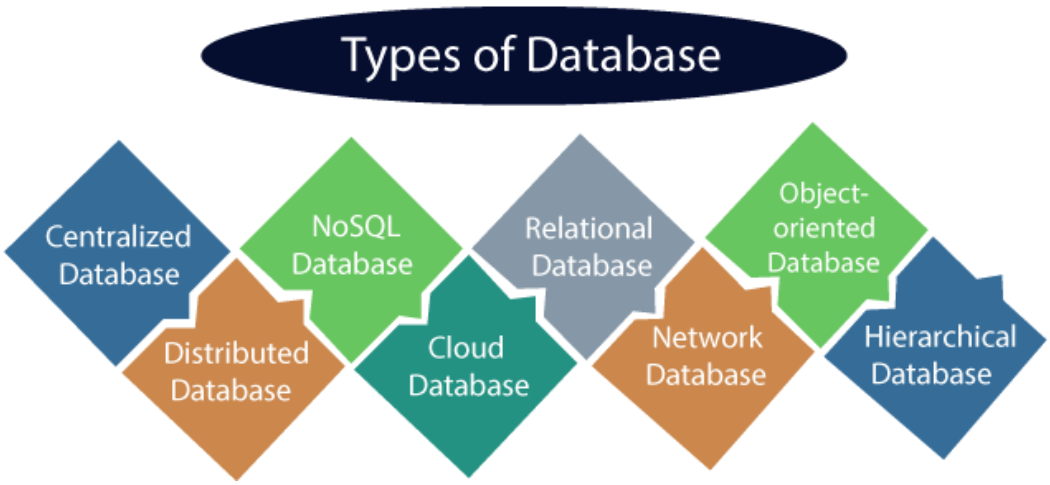

Figure 4. Types of databases. [35]

2.3.1 Relational Database

SQL-based relational databases offer easy access to structured data. Data is stored in table format in rows and columns that enables efficient way of handling the data. This predefined table format to store the data allows easy way to query and filter the required information. Relations between tables enable data to be stored in only one place reducing redundant information. Relational databases work well when data has relations between some of its elements. Company that sells some products to a customer could have a database for customers. This database could contain tables about customer personal information, customer transactions and customer agreements with the company. All of these tables have one thing in common, the customer. This creates relation between different tables of data that can be queried with customer as common factor, resulting reports with all the valuable information. Some of the most common SQL servers include: MySQL, MariaDB, Microsoft SQL server and PostgreSQL. [36]

SQL itself provides query language to access and manage database resources called Data Definition Language (DDL). For example, this allows user to create, edit and delete database tables by using DDL. All this can be done without shutting down database server, providing very flexible approach to data management. [36]

Relational databases could provide option to offer four key properties present in the acronym ACID. A stands for atomicity meaning data changes are made in a single operation providing change in all data as a part of transaction. This ensures that in scenario of data relocation, source and end point both have change in data. C stands for consistency refers to consistent state of data at the beginning and end of the transaction. This basically means that data should stay the same after a transaction. I stands for isolation meaning on-going transactions are not shown to other transactions. If data is moving from one place to another, it should be accessible only on

the source point or on the end point, not both at the same time. Lastly D stands for durability making sure that committed transactions are done even after system poweroff providing durable structure.[37]

2.3.2 Non-Relational Database

Non-relational database does not use relational model as the name stands instead, they can support very specific data type. As non-relational database does not have tables of rows and columns, data needs to be very specific in order to be added to the database. For example, data could be in form of a graph with nodes and edges generating this mesh network that couldn't be directly stored on the relational database. [38]

NoSQL is also called "not only SQL" as it evolved partly like SQL. Where SQL is designed to work with structured data, NoSQL has potential to store the unstructured data more efficiently. Data in NoSQL databases is stored in more abstract way as the information can be stored in different types of data structures, objects and files. Some NoSQL-based databases can provide ACID transactions like MongoDB [39]. Couple common NoSQL servers are Redis, MongoDB and Terrastore. [40]

Non-relational database can scale horizontally, meaning that the service scales by adding more servers to the cluster. This gives Non-relational databases great advantage handling large amounts of data as the service is horizontally scalable. Scaling is very useful for big websites that need to serve huge amount of data for multiple users.[39][41]

2.3.3 Event Streaming

Event streaming is form of data streaming meaning constant data flow. Events are records of something that happened within the system like user paying for a product in an online shop or user clicking to different page in a website. When a user browses website through clicking links and buttons it generates flow of events that can be considered event stream. This stream of events can be published to the event streaming platform to be processed and analysed by some other service. These other services are subscribed to the event streaming platform, and they will receive the stream of events as they arrive on the event streaming platform. [42]

Event streaming platform Apache Kafka is not quite a DLT platform but has some similarities. It's distributed scalable platform that keeps immutable commit log about data stored on the platform. This means that messages or order of messages in the platform cannot be changed. However, deletion of a topic is allowed but can be controlled. This enables multiple ways to use Kafka as example: log system to replace tradition logs, proxy between consumers and producers simplifying data pipelines or messaging platform that makes abstraction between a message and an event. [43][44]

Apache Kafka architecture contains Zookeepers, Kafka brokers, consumers and producers. Basic relation is shown in Figure 5. Core of the system is Zookeeper that manages the cluster of Kafka brokers and their metadata. Zookeeper can be deployed with each Kafka broker to provide fault tolerance against downtime, as Kafka broker require Zookeeper connection to keep cluster running. Multiple brokers may form a cluster where data is replicated over multiple brokers providing horizontal scaling.

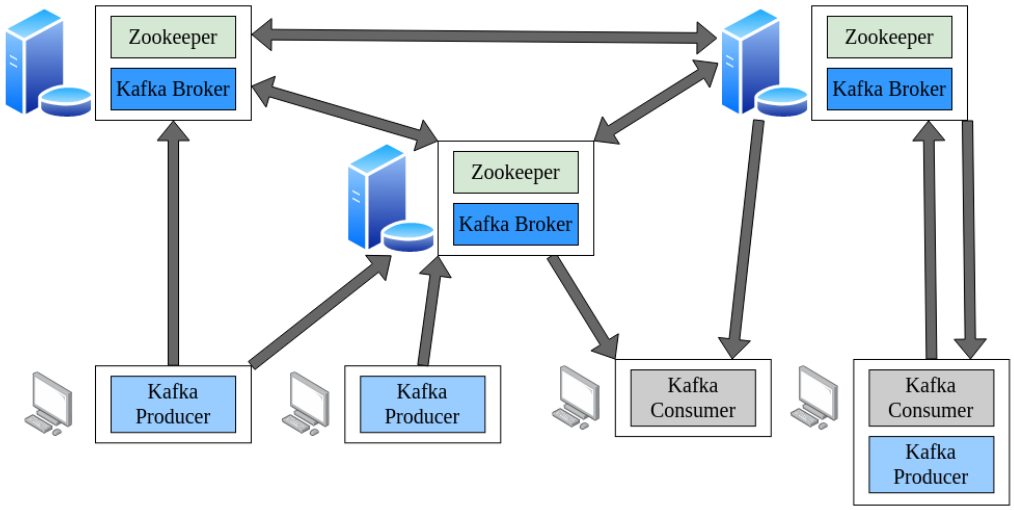

Figure 5. Kafka cluster architechture.

Kafka can be defined as a database but compared to more traditional databases it offers way less control over the data. As Kafka has append only log, data cannot be edited after it is added to the log. This type of behaviour increases the performance of the system. More complex data may not be efficient to store on Kafka alone but use Kafka as a gateway to other databases. [45]

2.4 Virtualization

To provide scalable and easy to setup deployment environment, use of virtualization comes up to ease the process. Virtualization offers way to emulate hardware resources generating virtual resource pool that can be shared for multiple Virtual Machines (VM). Difference between traditional bare metal approach and virtualization is abstraction between software and hardware. Same virtual environment can be run on multiple different physical hardware without any change in operating systems and services running in VMs. Traditional physical environment without virtualization would need operating system changes like different drivers when underlaying hardware changes. [46]

In IoT systems lightweight virtualization is preferred over full virtualization. There are three main types of lightweight virtualizations available for Linux: unikernel, container, and Kernel Virtual Machine (KVM). KVM being closest to a traditional virtual machine with the largest memory footprint because of actually virtualizing some hardware resources. Container being close to native performance by sharing same kernel with all guests, only limiting usage to hardware resources. Unikernel is the most lightweight option running each application on separate process with own specialized kernel giving better isolation than containers. Unikernel shows potential over other virtualizations but lack the wide adaptation as it does not have sufficient security measurements. Figure 6. shows how these three main approaches differ from resource isolation perspective. Containers are split into two different categories machine containers and process containers. Machine containers may have multiple services inside one container, whereas process containers may have only one service per container. [47]

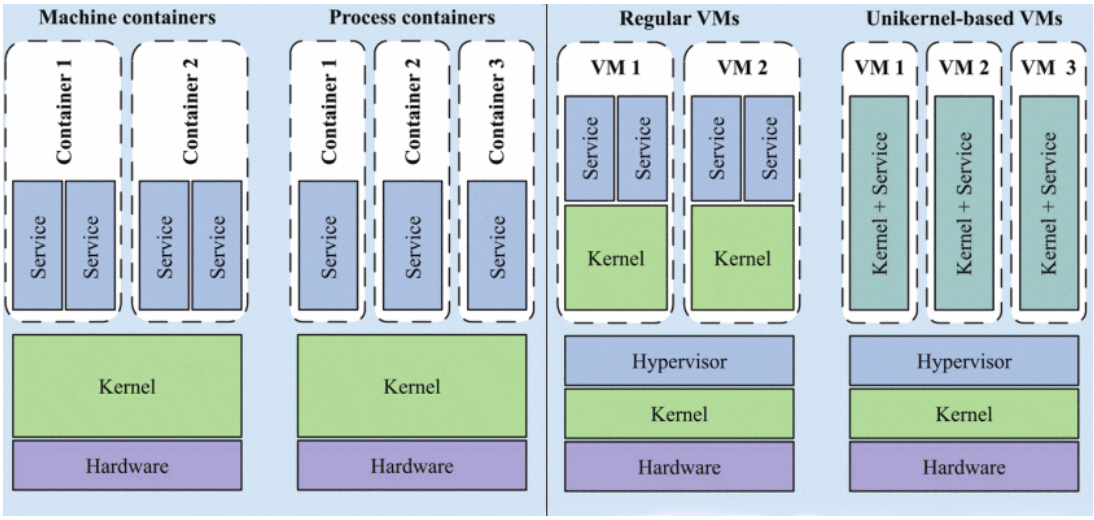

Figure 6. Visualization of resource isolation. [48]

2.4.1 Desktop Virtualization

Full desktop virtualization can be described as a traditional way of virtualization. This type of virtualization does not only virtualize the service and operating system but also the hardware. Idea is to virtualize most of hardware resources consuming a lot of computational power and memory. Full virtualization enables possibility to run totally different OS inside the virtual machine than the host OS. [49]

Virtual Desktop Infrastructure (VDI) is a popular way of providing desktop virtualization. It is based on client-server model where server runs virtual machines providing desktop virtualization for clients. These virtual machines can be served from persistent image or nonpersistent image. Persistent image can be saved and works as one could expect from normal desktop usage. Nonpersistent on the other hand reverts back to original form after user quits the session offering stable experience for the user. [50]

2.4.2 Service Virtualization

Service virtualization enables wide use of Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) types of services shown in Figure 7. These services offer companies alternative options compared to investing and managing local hardware resources. To deploy email service for the company one would need hardware to run it on and people to manage it. This has high initial investment and still requires upkeep costs. [51]

IaaS is the lower-level virtualization among the service virtualizations offering hardware, networking, storage and virtualization platform as a service. This gives customer control over the choice and management of operating system, runtime environment and application services. Flexibility of buying hardware resources as service enables customer to buy just what is needed and ability to expand resource as needed resulting small initial investment compared to on-premise approach. [51]

PaaS is level higher in hierarchy compared to IaaS. PaaS offers everything as IaaS but also operating system and some runtime for possible applications. This cuts customers management time as customer only needs to deploy an application as they wish. Because of this primary user base is developers who need platform to test and develop applications. [51]

SaaS is the highest level of service virtualization as the service provider basically offers everything. This approach gives no user control over the system or the application, but it offers end user access to the applications provided. Being the simplest solution to end user as they don't need to manage anything on the server side. SaaS offers application like email as a service over the internet eliminating initial investment and ongoing management of the software. [51]

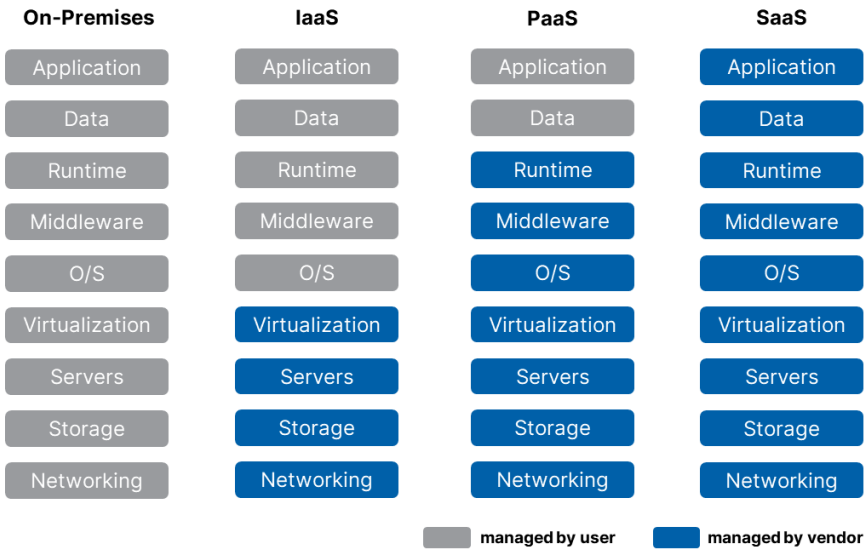

Figure 7. Visualization of resource isolation. [52]

2.4.3 Containers

Containers are lightweight virtualization technology which are good for IoT devices. Containers do not emulate hardware like a virtual machine would, but they virtualize the OS. Virtualization is provided using namespaces and control groups to isolate processes and data from the host system. All containers on a single host share the same kernel as the host. This is called single kernel virtualization. Docker and Linux Containers (LXC) are few of the well-known container platforms. [53][54]

Rapid deployment with easy to deploy structure are key concepts of containers. To enable rapid deployment some sort of process automation is required. Container orchestration offers this automation as it manages the deployment of the container with proper permissions, storage options and network settings. Kubernetes is one of the most popular container orchestration tools. Other mentionable tools are Docker Swarm and Apache Mesos. [55]

3 SYSTEM DESIGN

Requirements for the system design are straight forward. Usage of IoT device requires minimal data processing to produce data in a reasonable rate. This eliminates most of local processing and puts need for confidential data transmission, in this case Apache Kafka. In case of an emergency, extra steps in data processing chain would be beneficial to cut out requiring minimal data processing from IoT device and sending emergency signal if certain thresholds are crossed.

3.1 Test Hardware

Raspberry pi 4 is used as a test server to show capability of IoT device operating as a communication hub. Used Raspberry pi has 4 Gigabytes (GB) of RAM which is higher end of the Raspberry pi product line. Higher RAM count enables bigger memory buffers for data management resulting better performance in short bursts. Raspberry pi 4 has a Broadcom BCM2711 SoC with quad-core Cortex-A72 processor running at 1.5 Ghz. Raspberry pi 4 also has 1 Gigabit (Gb) network interface controller (NIC) for ethernet connecting Raspberry pi to local network. [16]

Desktop computer running test clients is a AMD based x86 platform running Linux based distribution as a operating system. It has AMD Ryzen 7 1700 eight-core processor running at 3.6 Ghz with 32 GB of RAM. This system is capable of simulating multiple users to test effect of multiple requests. This computer also has NIC for ethernet and it is connected to same network as Raspberry pi. Both machines are in the same Local area network (LAN) connected to 1 Gb switch with 1 Gb links show in Figure 8.

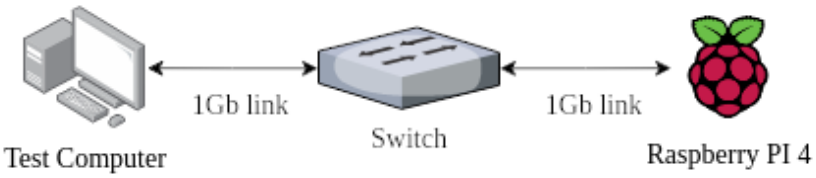

Figure 8. Network diagram of the test system.

3.2 Test Software

Raspberry Pi has Docker containers for Kafka broker and Zookeeper to provide clean environment for each run. Both Docker containers are based on same Debian Docker image with Kafka binaries and dependencies installed [56]. Desktop computer runs a single test client in LXC container to serve different packages than host system. The test itself ran inside the container by generating data to the Kafka platform via producers and receiving the data from the platform using consumers. Illustration of primary services running in both machines is shown in Figure 9.

Kafka server version 2.8.0 is used in these tests [57]. Client library librdkafka-1.6.1-r0 is used in the client side to provide high performance application programming interface (API) [58]. Kafka server and zookeeper are running with mostly default setting resulting only change

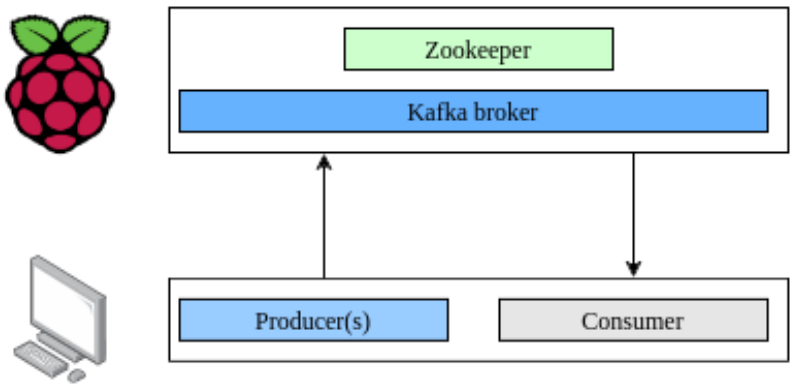

Figure 9. Kafka test setup.

to be faster topic removal time. After each test used test topic will be deleted and re-created giving new topic for each run. SQL tests are done against MariaDB version 10.4.17 inside a Docker container [59]. Resource usages are monitored by Docker stats utility where memory usage is determined by difference in committed memory.

3.3 Test Setup

Order of test setup is shown in Figure 10 that expresses single loop of the test process. In both SQL and Kafka tests the process is the same to perform tests for measuring latency, payload and throughput. To achieve more accurate timestamp, the times are inserted as part of the payload. In Kafka we have byte string, where first few bytes represent the timestamp so it can be easily extracted from the payload message. For SQL different column is used to give more natural separation between different data types as platform has ability to do that.

Generated payload is random data which is generated by test computer using kernels urandom character device [60]. Data is random and does not follow specific distribution. New random data is generated for each sent payload meaning that all users have different payload and each message sent by the users have a different payload. Each random call has set byte size based on the test requirement, so that payload size is kept constant over all messages in the tests. Randomizing data this way may generate differences in the results as randomness may produce repeating bytes or completely different bytes. Randomness is minimized by sending multiple messages and repeating tests multiple times reducing variance from data randomization.

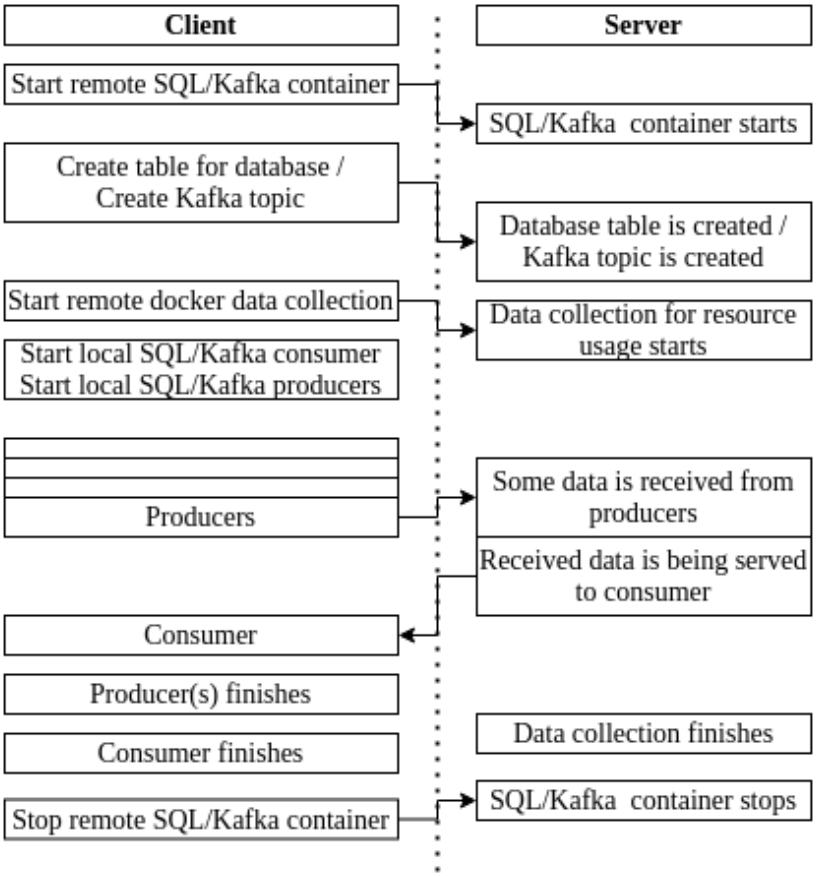

Figure 10. Test process.

Payload in latency tests contains also a timestamp. Unix time is used as a timestamp which indicates how many seconds are elapsed from Unix epoch. Unix epoch is 1st of January 1970 at 00:00:00 Coordinated Universal Time (UTC) [61]. As the used timestamp contains the Unix time in milliseconds with 2 decimal accuracy, its length stays static while testing. This increases actual payload by 8 bytes resulting biggest difference in payload tests where payload of size 1B is tested. This results actual payload of 9B to be transferred making difference to next step only two times instead of ten.

4 EVALUATION

Test results are presented here with short introduction to the test method. Goal is to test how well these platforms perform when run on a IoT device. Main attributes measured are latency and throughput as these reflect the responsiveness and performance of the system. Different number of users are swept for the latency tests and with fixed number of users with increasing payload are observed in terms of latency. In Throughput tests enough data is send to the platform so that it shows maximum constant data rate it can handle. Multiple system resources are measured and presented with each test.

4.1 Latency

In Kafka latency tests python wrapper for librdkafka is used to give easy to access API. These tests are run in batch with small delay in between. Single consumer process is started first and it will wait for producers to generate message to the server. Each single test contains 100-1000 individual producers (users) all with fixed payload size. All users send 60 messages to Kafka broker with inter-arrival time with mean of 1 second following normal distribution. Consumer will fetch the messages as fast as possible, and it compares timestamp of message creation time and current time to determine latency of the transmission. Both message creation time and current time are measured in 10 microsecond (us) accuracy. After the run, consumer will timeout and generate output of latencies, and the Docker containers are restarted to clear any data form previous runs. Each run is repeated 10 times and results will be averaged across all 10 runs.

SQL test scenario is similar to Kafka one, 100-1000 producers and single consumer are used. As this traditional SQL-based solution does not have event-based waiting for new data, the consumer needs to query for new data constantly. Consumer tries to send new query every millisecond to get new messages as soon as possible. Consumer queries for indexes greater than what it has received so that same data is not transferred multiple times. As with Kafka tests each producer sends 60 messages with the timestamp and payload keeping 1 second between each message.

Result of latency tests in Kafka scenario can be seen in Figure 11. This shows that Raspberry pi can serve multiple users with payload size of 100 B without compromising latency when using Kafka. We can observe the latency variance to be around 1 ms across all user scenarios. SQL-based platform performed like Kafka in this scenario. Variance between runs seems to be similar in both scenarios. Because of this variance, no clear increase in latency can be observed between 100 and 1000 users. Latency in both scenarios is reasonable and should be feasible in many applications.

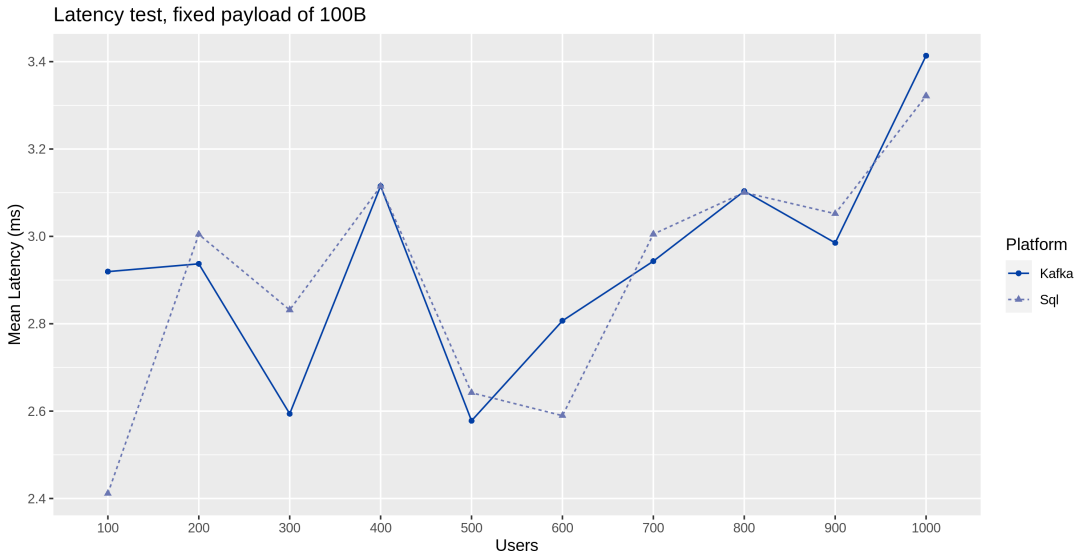

Figure 11. Average round trip latency of Kafka and SQL systems with 100 B payload while scaling users from 100 to 1000.

During the latency tests, system resources were monitored for key points of 100, 400, 700 and 1000 users in Figure 12. Central processing unit (CPU) usage and additional memory usage of containers running in the Raspberry pi are shown. CPU usage increases slowly with more users but is not over saturated resulting no increase in latency. Kafka seems to utilize CPU around 2 times more than SQL-based MariaDB in the server side. Memory increase in Kafka seems to be mild compared to SQL-based solution. In SQL memory usage increases with the user amount where in Kafka the difference is much smaller and not much increase between 700 and 1000 users. As payload is relatively small, it shouldn't increase memory usage by itself too much if it is cached in the memory during the testing period.

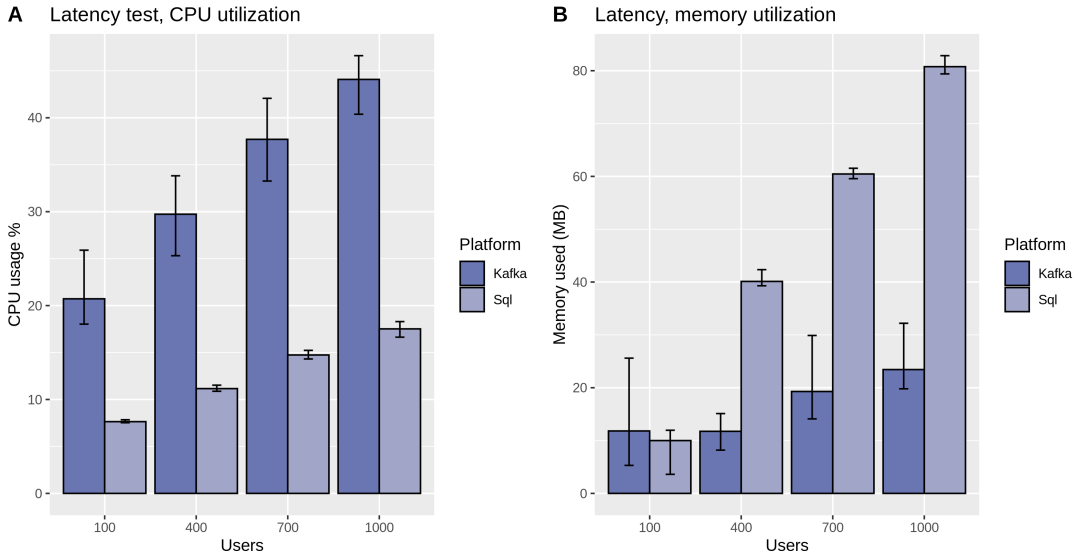

Figure 12. Cpu utilization and memory usage during latency tests with users 100, 400, 700 and 1000.

Total disk writes after a latency run shown in Figure 13. In the figure disk writes are shown and we can observe Kafka writing less data to the disk than a SQL counterpart. The difference between these platforms stay around 10 MB while both platforms have linear increase in writes while the user count is increased. Kafka seems to prolong the wait between disk writes with such of a small data amount as it lets OS to deal the disk writes and caching.

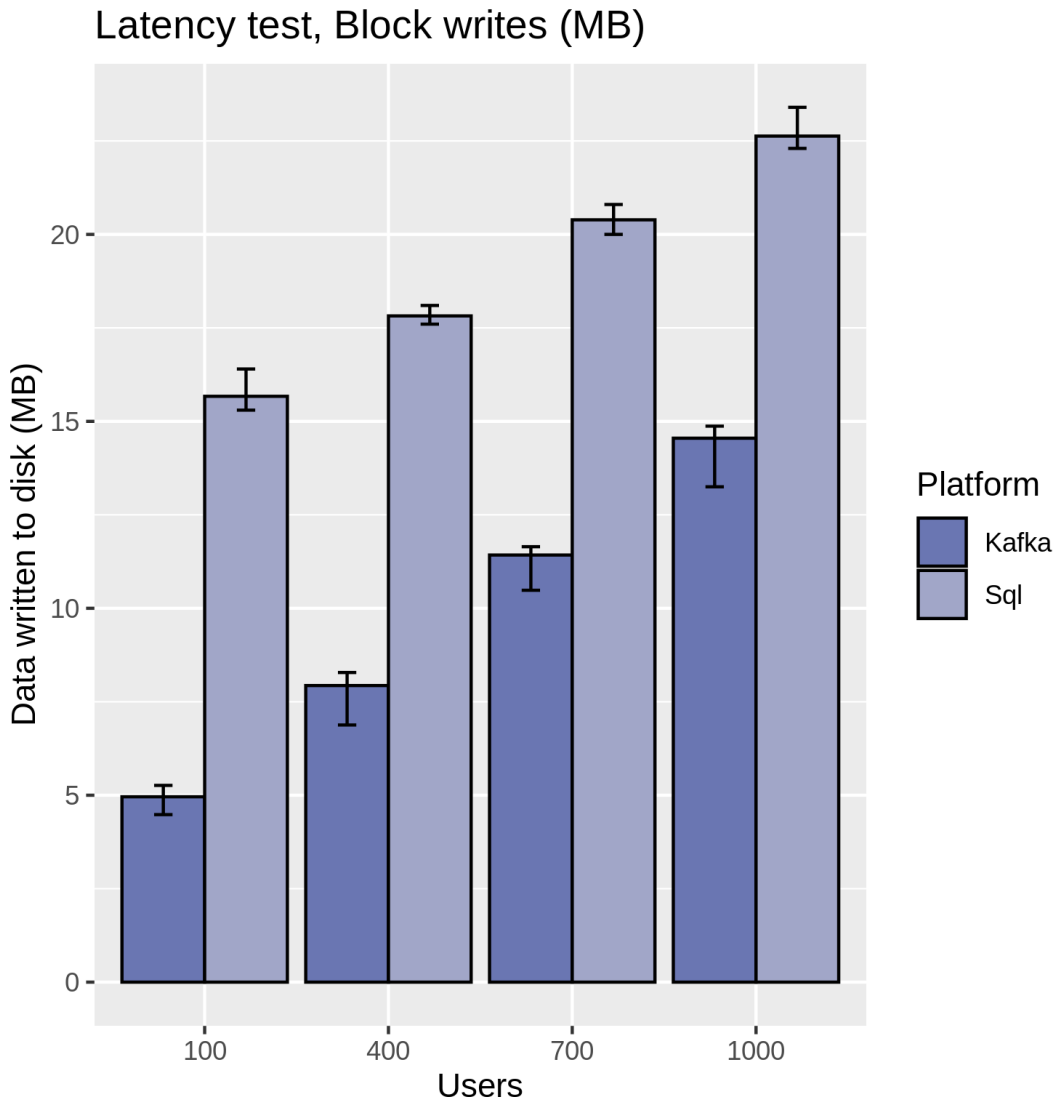

Figure 13. Disk writes during latency tests with users 100, 400, 700 and 1000.

Received and send network data after a latency run shown in Figure 14. In A) part of the figure, we can see incoming data amount in the network interface where Kafka seems to be receiving around 7 MB more data than SQL. Both platforms have linearly increasing amount of data coming in when user count is increased. Outgoing network data is shown in B) part of the figure and there we see similar behaviour compared to incoming data. While Kafka sends more data out its data amount increases more than SQL when serving more users.

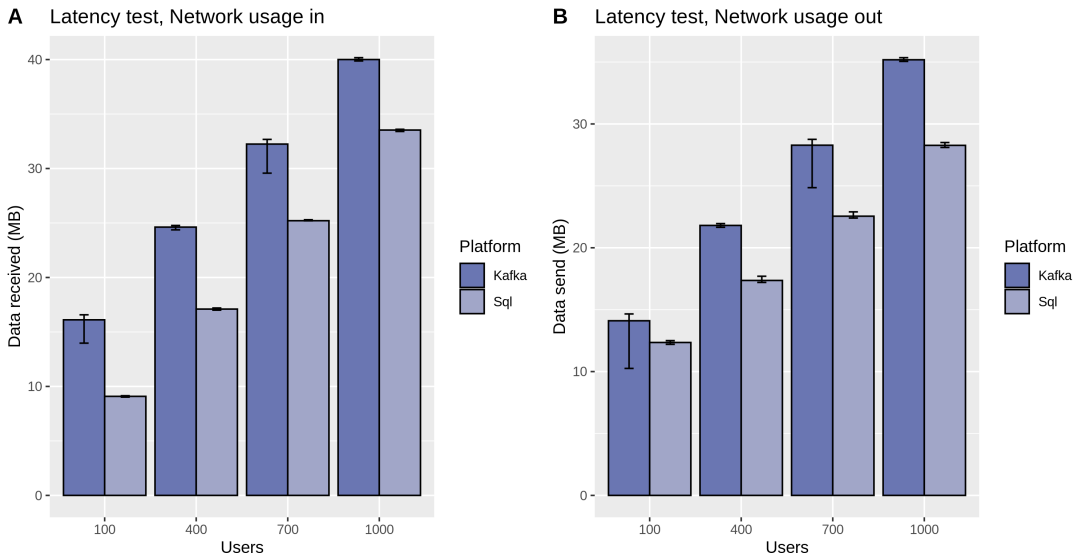

Figure 14. Network received / send during latency tests with users 100, 400, 700 and 1000.

Fixing the users to 100 and changing the payload from 1 B to 10 kB in tenfold steps results following latencies that can be seen in Figure 15. Both platforms retain low 2-4 ms latency up to 1000 B payload similar to latency tests. When payload is increased to 10 kB much higher latency can be observed where SQL seems to have lower latency in this scenario. There's still a lot of variation between the runs in 10 kB scenario as more data needs to be written to the disk and it will generate big latency spikes.

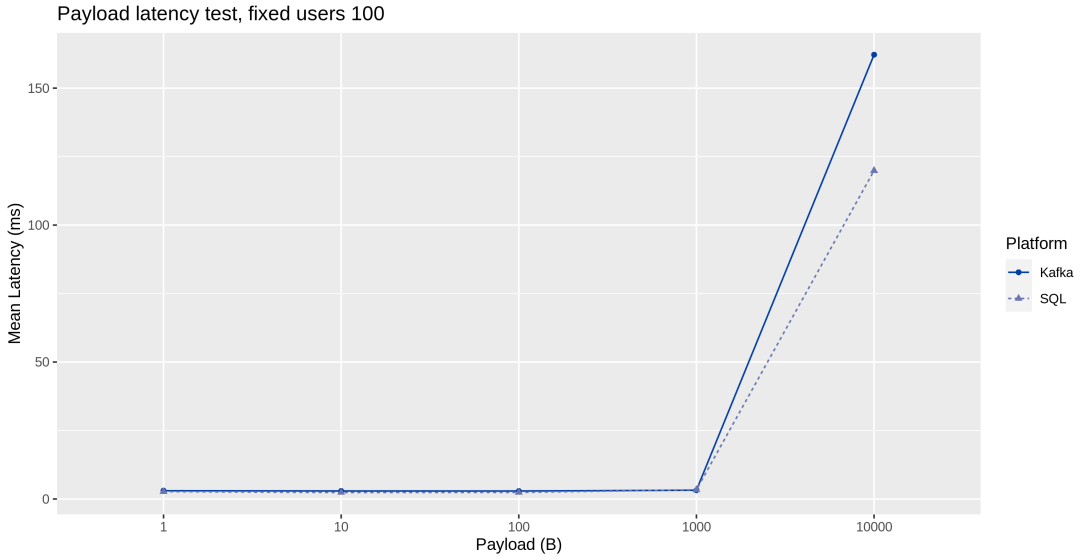

Figure 15. Average latency with 100 users and payload from 1 B to 10 kB in tenfold steps.

CPU usage in the payload test is shown in Figure 16. Interestingly CPU usage seems to be smaller with bigger payloads in Kafka. This may be because of less connections are made because actual data transmission takes longer resulting less hassle for the CPU. On the other hand, SQL has consistent CPU usage across payloads up to 1 kB but has slight increase in 10

kB scenario. Overal Kafka seems to use more than double the amount of CPU time than SQL platform.

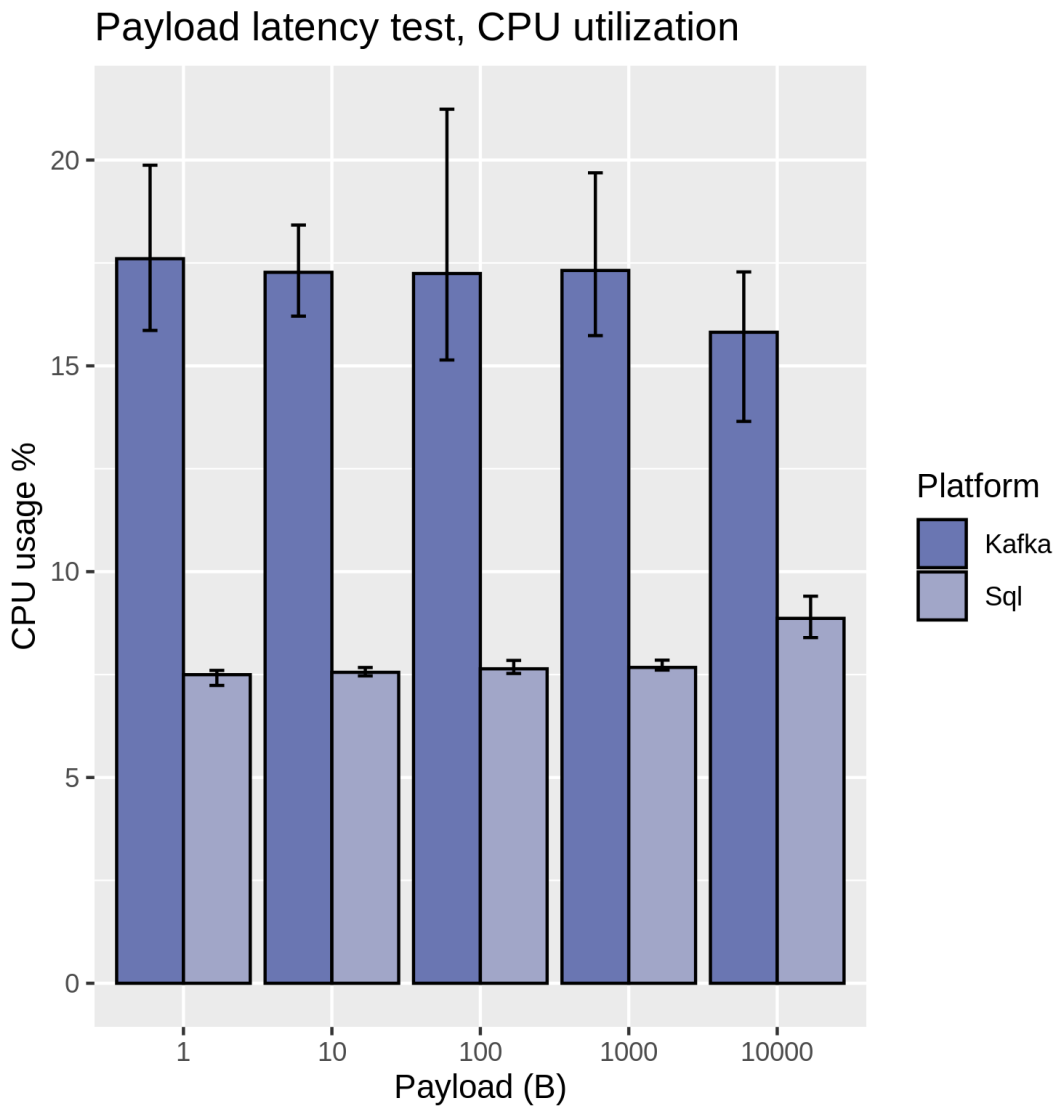

Figure 16. CPU usage during payload test.

4.2 Throughput

Throughput test is done same way as all the other tests but instead of measuring latency, the time used to send and receive a fixed sized data block is measured. Multiple queries in producer side are bundled together to raise the performance in both systems. In both platforms payload size is set to be 1 kB which is small to test maximum throughput of the system, but it shows the system performance when processing multiple requests in row. 100 MB of data is sent in each run using the 1 kB payload size. Two instances of these tools are used, one for the producer and other for the consumer. In one scenario, both clients are running in same time providing throughput from producer to consumer measured by the consumer. In another scenario producer is run alone to provide producer throughput. After producer finishes consumer starts running

consuming freshly created messages from the Kafka server. This results consumer throughput when data is already available on the server cache. Results of this measurement are shown in Figure 17. and the measurements are averaged over 10 different runs and the minimum and maximum are shown by the whiskers.

In these results consumer side seems very similar for both platforms with little bit less variation on SQL side. In both scenarios reading from the server is much faster than writing to the server. In order to store the data, it must be written to the disk which is much slower than reading from it. On the producer side Kafka seems to perform little better, at least more consistently. When both consumer and producer have been running same time Kafka performs still a bit better. While both have similar variation in both scenarios, Kafka manages to get higher avarage throughput.

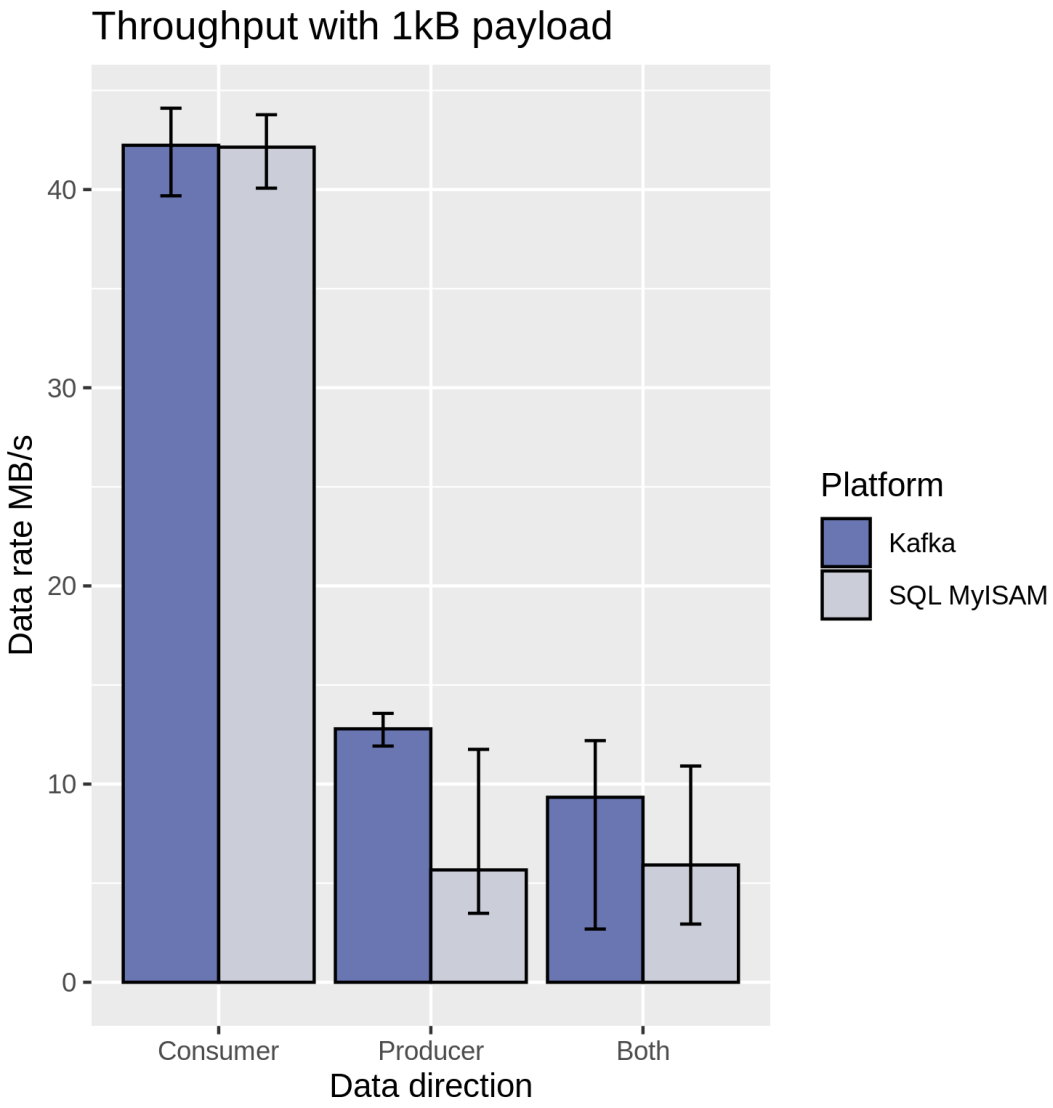

Figure 17. Data rates in throughput test.

Looking at the CPU utilization in Figure 18. we'll see that Kafka uses less CPU when only the consumer is running. Producer scenario and combination scenario are so close together as both have the producer factor raising the CPU utilization. Variances in these scenarios are big

but slight increase in CPU utilization can be observed when both are running compared to only producer. Kafka uses similar amount of CPU time between producer and both while SQL have increase in CPU utilization when both are running.

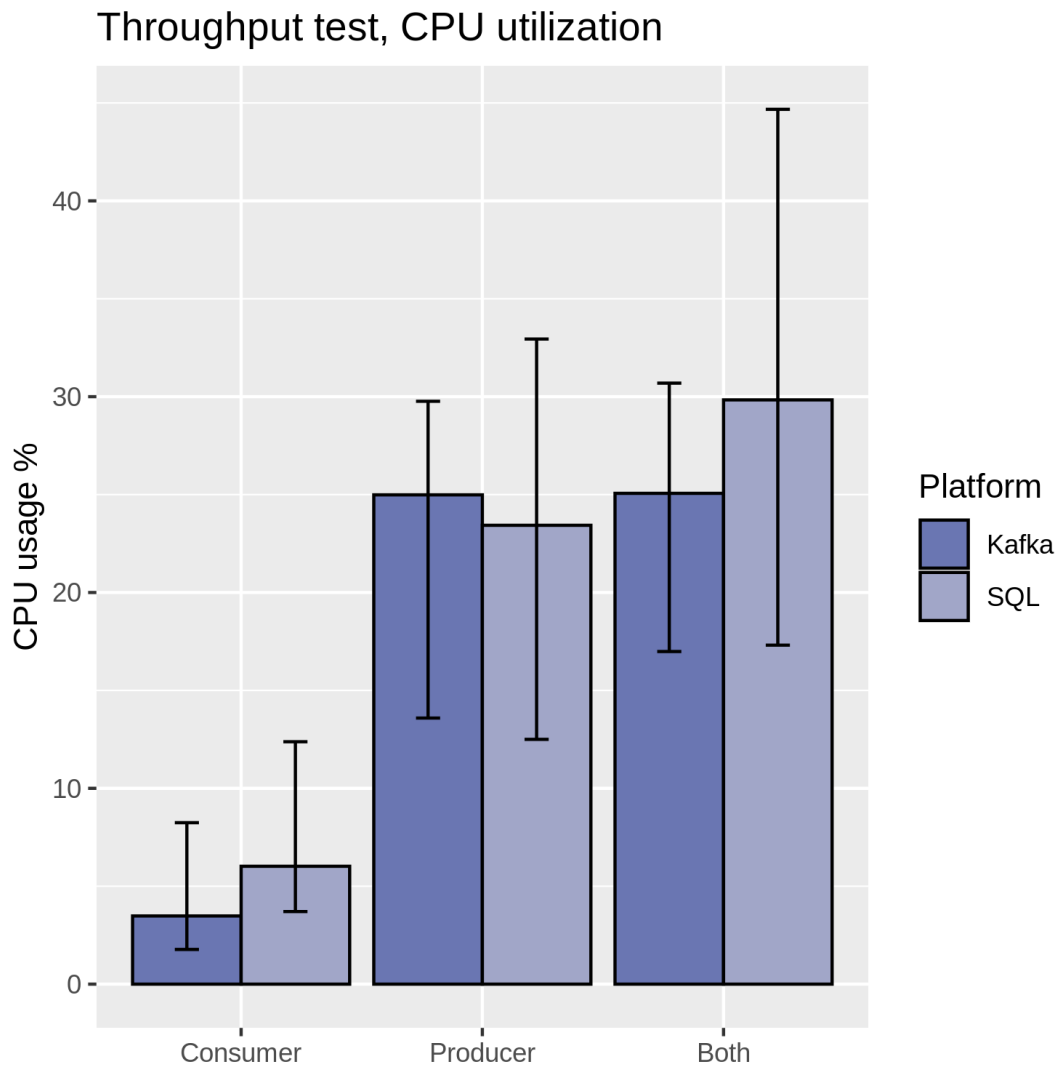

Figure 18. CPU usage during throughput test.

Network usage is shown in Figure 19. where A) shows received network traffic and B) sent network traffic both in the server side. Because producer is run first it does not differ much from consumer in the received data graph. In A) we can observe higher amount of data received by the server in Kafka scenarios compared to SQL. On the B) side of the figure, data sent from the server is shown. There producer has greatly lower data amount than consumer scenarios because of producers have only sent data to the server but server has not sent it to the consumers yet. Consumer and both running at the same time result similar network traffic from the server. This means that running both at the same time does not saturate the system and generate lower throughput. SQL seems to have least amount of data sent in each scenario.

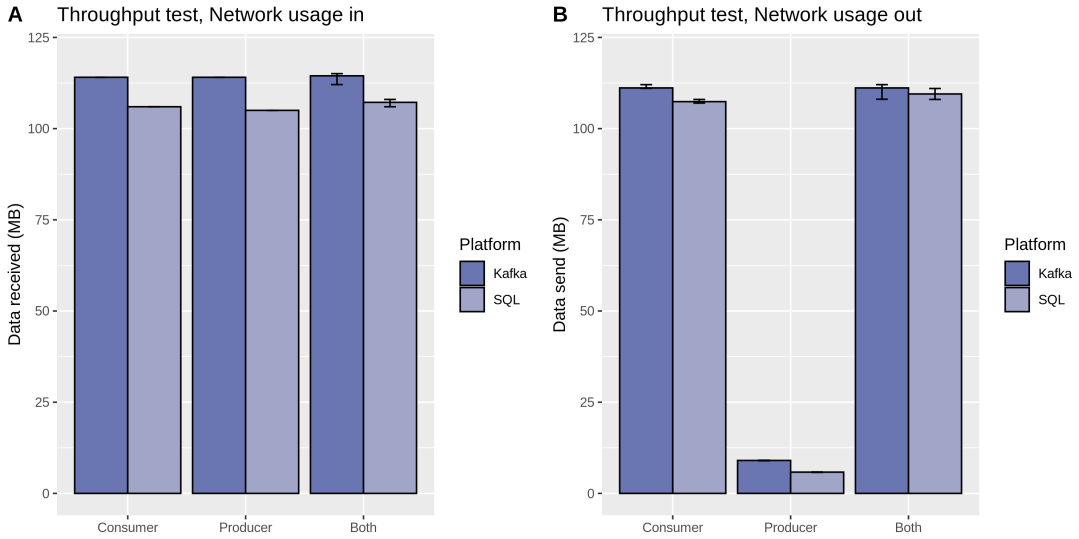

Figure 19. Network usage during throughput test.

Disk writes on the server are shown in Figure 20. Both platforms write reasonable amount of data to the disk, but SQL writes little more than Kafka. SQL also seems to write data less frequently to the disk as this can be observed by the variance in producer and both scenarios. In consumer scenario also SQL have written the data to the disk where in producer scenarios some of the data is not yet written.

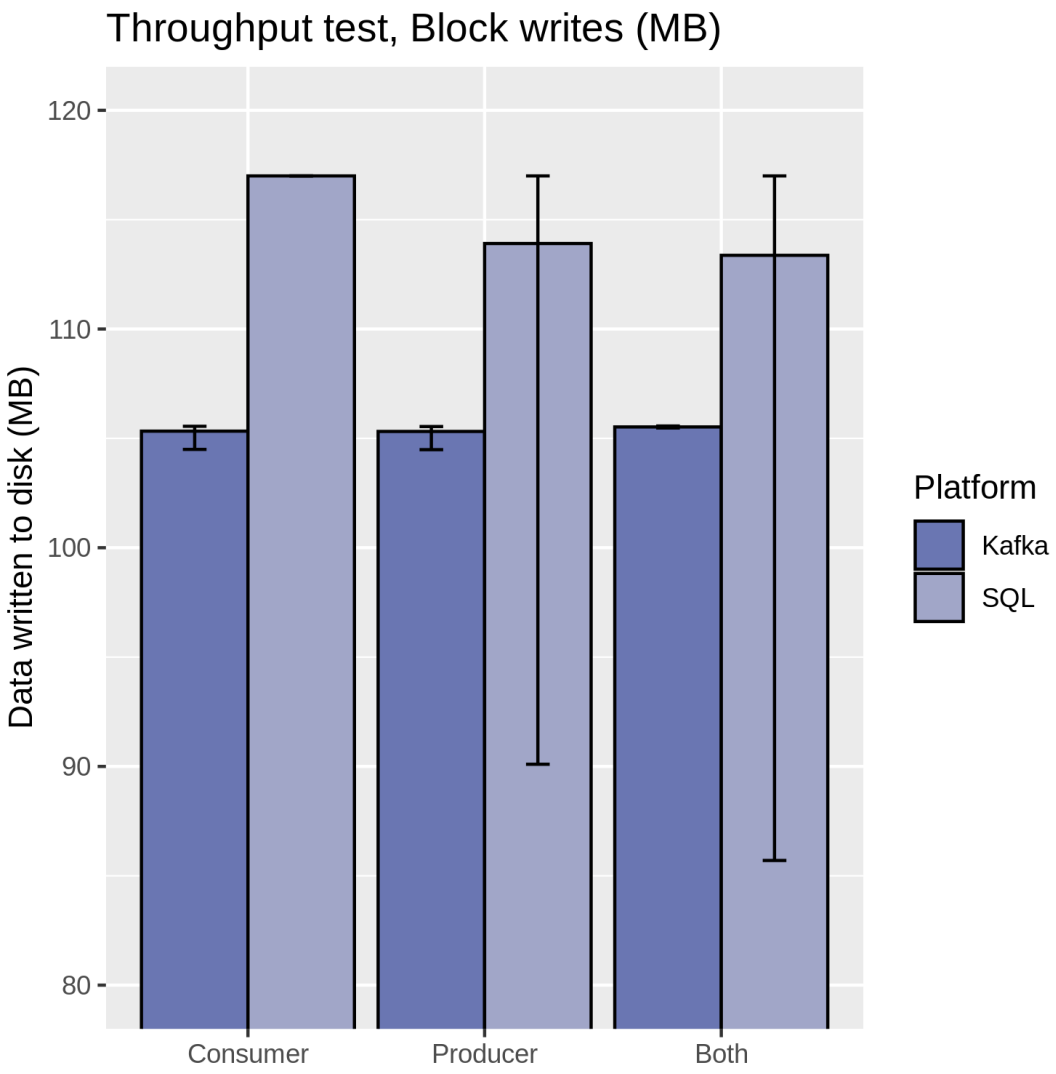

Figure 20. Block writes during throughput test.

5 DISCUSSION

Kafka and SQL performed both similarly showing possibility to one of the IoT devices to be hub for other devices around. Serving multiple users with relatively small payload both platforms resulted similar latencies serving the users.

Main performance slowdown for both platforms was disk write speed on the Raspberry side. As both platforms were configured to save the data to the disk, the data throughput exceeded the Secure Digital (SD) card write speed resulting high I/O wait. This resulted latency spikes when platform flushed data from memory to disk and made little variation to the results.

During the testing one major slowdown was detected in the testing chain. After around one hour the test client that simulated users started to show increased latencies. This was fixed by restarting test client container and repeating tests that were affected by the slowdown. Restarting test client container should have been done after each test to generate more consistent results. This behaviour may have introduced some extra variation between results, especially in the latency tests some extra jitter may have been caused by this.

Handling of containers in the server side (Raspberry) is implemented by connecting remotely to the server and invoking local restart of the container. This was implemented to speedup testing procedure as it was fast implementation and did everything that was expected. Idea was to change this procedure of using remote shell to some container orchestration utility and using its API to manage state of the containers. This was never implemented in this work but would be good addition for more practical way of managing containers.

Security options were not implemented in this testing setup. Something like data encryption would be essential feature for over the internet data transmission. Both platforms tested supported this kind of behaviour, but it was not implemented as it increased the complexity of the testing. This could have increased CPU and memory usages in all scenarios.

Authorization was implemented on the SQL side by using user and password combination to give users access to database. In Kafka implementation no access control list was used giving any client connecting to the server access to write and read data off the platform. This may have small overhead difference between the two implementations but as tests continue for at least a minute a small overhead in the beginning is negligible.

Kafka being append only database it does not allow edits on the data after data is submitted to the server. In SQL one can limit user permissions to edit the data but root admin still has capability to manipulate the data. This gives Kafka small advantage in offering immutable data in cost of general usability.

Based on the test results, Kafka seems to utilize CPU much more in tests that do not push the throughput to the limits. This means that simple scenario, where couple IoT devices are collecting data and sending it to the server, SQL would be better choice for the task. On the other hand, when pushing throughput limits Kafka manages to get similar or lower CPU usage than SQL. Video streaming could be one of those tasks that would be in that region of high throughput, where Kafka would be better choice over SQL.

5.1 Future Work

To further expand the Kafka possibilities in IoT usage more suitable comparison than SQL should be considered. Other event stream platforms would be more even comparison in terms of usage and performance. This would also show how well those possibilities work on IoT

platform. Also, because Kafka performed well here, maybe some lower end IoT device could be considered as resource utilizations were not high.

Scalability of the platform could be tested in same performance metrics as this project. Instead of scaling users one could scale number of servers in the cluster and try to measure performance from different nodes.

6 SUMMARY

Growing number of IoT devices with the expanding Internet has led to ever increasing data volumes. Higher data volumes have evolved IoT devices to manage higher data rates making those devices more capable of data management. Reducing huge data flow from cluster of IoT nodes to the cloud by processing the data closer than cloud. Doing processing closer to the client has become more lucrative choice to make. This has been done by processing data in the fog or in the edge moving processing closer and reducing big data flow over the Internet. Taking the process one step closer by processing data locally would reduce the data flow even further, leading to the idea of this thesis by processing data on the IoT device. Doing processing locally does not only reduce the data flow over the Internet, but also cuts down latency between source and processing. Minimizing latency is critical for many emergency situations like critical medical care.

In this thesis two services, Kafka and MySQL-based MariaDB are tested on a local IoT device and compared against each other to determine if Kafka can be used in this environment. Comparison to MySQL came from the idea of traditional way storing data in contrast to this more streaming centric, constant data flow architecture that Kafka provides.

Parameters to determine usability of the platform were latency, throughput and resource usage. Most cases both platforms performed similarly but differences can still be observed. Kafka can be run in IoT environment but in basic data management does not give great advantage compared to traditional solution. High throughput situations like video surveillance, Kafka could be used over traditional way of storing data.

7 REFERENCES

[1] StatCounter, Desktop vs Mobile vs Tablet Market Share Worldwide (read 24.2.2022). URL: https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/worldwide/2021.

[2] Hui S., Wang H., Xu D., Wu J., Li Y. & Jin D. (2022) Distinguishing between smartphones and iot devices via network traffic. IEEE Internet of Things Journal 9, p. 1182–1196.

[3] Vanneste S., de Hoog J., Huybrechts T., Bosmans S., Eyckerman R., Sharif M., Mercelis S. & Hellinckx P. (2019) Distributed uniform streaming framework: An elastic fog computing platform for event stream processing and platform transparency. Future Internet 11.

[4] Sisinni E., Saifullah A., Han S., Jennehag U. & Gidlund M. (2018) Industrial internet of things: Challenges, opportunities, and directions. IEEE Transactions on Industrial Informatics 14, p. 4724–4734.

[5] Ejaz M., Kumar T., Kovacevic I., Ylianttila M. & Harjula E. (2021) Health-blockedge: Blockchain-edge framework for reliable low-latency digital healthcare applications. Sensors 21.

[6] Vitabile S., Marks M., Stojanovic D., Pllana S., Molina J., Krzyszton M., Sikora A., Jarynowski A., Hosseinpour F., Jakobik A., Ilic A., Respicio A., Moldovan D., Pop C. & Salomie I. (2019) Medical data processing and analysis for remote health and activities monitoring. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 11400, p. 186–220.

[7] Al-Fuqaha A., Guizani M., Mohammadi M., Aledhari M. & Ayyash M. (2015) Internet of things: A survey on enabling technologies, protocols, and applications. IEEE Communications Surveys and Tutorials 17, p. 2347–2376.

[8] Abdelnapi N., Omran N., Ali A. & Omara F. (2018) A survey of internet of things technologies and projects for healthcare services. In: Proceedings of 2018 International Conference on Innovative Trends in Computer Engineering, ITCE 2018, vol. 2018-March, vol. 2018-March, p. 48–55.

[9] Alavi A., Jiao P., Buttlar W. & Lajnef N. (2018) Internet of things-enabled smart cities: State-of-the-art and future trends. Measurement: Journal of the International Measurement Confederation 129, p. 589–606.

[10] Smart Appliances: IoT and the Laundry (read 16.2.2021). URL: https://www.aeris.com/news/post/smart-appliances-iot-and-the-laundry/.

[11] Chen B., Wan J., Shu L., Li P., Mukherjee M. & Yin B. (2017) Smart factory of industry 4.0: Key technologies, application case, and challenges. IEEE Access 6, p. 6505–6519.

[12] Coulby G., Clear A., Jones O., Young F., Stuart S. & Godfrey A. (2020) Towards remote healthcare monitoring using accessible iot technology: state-of-the-art, insights and experimental design. BioMedical Engineering Online 19.

[13] Raspberry Pi Foundation, Teach, Learn, and Make with Raspberry Pi (read 9.1.2021). URL: https://www.raspberrypi.org/.

[14] Arduino Home (read 9.1.2021). URL: https://www.arduino.cc/.

[15] ESPRESSIF (read 9.1.2021), Development Boards | Espressif Systems. URL: https://www.espressif.com/en/products/devkits.

[16] Raspberry Pi Trading Ltd. (2021), Raspberry Pi computer model B (read 24.5.2021). URL: https://datasheets.raspberrypi.org/rpi4/raspberry-pi-4-product-brief.pdf.

[17] Introduction to Embedded Operating Systems (read 23.1.2021). URL: https://microcontrollerslab.com/embedded-operating-system/.

[18] Bare-metal and RTOS Based Embedded Systems (read 27.5.2022). URL: https://microcontrollerslab.com/difference-bare-metal-and-rtos-based-embedded-systems/.

[19] McCreary D. (2020), Netburner: TCP vs UDP: Battle of the Protocols (read 25.2.2022). URL: https://www.netburner.com/learn/tcp-vs-udp-battle-of-the-protocols/.

[20] Thubert P., Bormann C., Toutain L. & Cragie R. (2017), RFC 8138, IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Routing Header (read 19.1.2021). URL: https://tools.ietf.org/html/rfc8138.

[21] Assured systems, What is a ZigBee Network? A Concise Overview (read 20.1.2021). URL: https://www.assured-systems.com/uk/news/article/what-is-a-zigbee-network/.

[22] Core Specification Working Group, Bluetooth core specification v5.2 (read 21.1.2021). URL: https://www.bluetooth.com/specifications/specs/core-specification-5-2/.

[23] Mwakwata C., Malik H., Alam M., Moullec Y., Parand S. & Mumtaz S. (2019) Narrowband internet of things (nb-iot): From physical (phy) and media access control (mac) layers perspectives. Sensors (Switzerland) 19.

[24] Kuukka A., (read 22.1.2021) LoRaWAN technology. URL: https://www.digita.fi/en/services/iot/lorawan-technology/.

[25] Li G. (2022), MQTT, CoAP, or LwM2M? Which IoT protocol to choose? (read 26.5.2022). URL: https://www.emqx.com/en/blog/iot-protocols-mqtt-coap-lwm2m.

[26] Shelby Z., Hartke K. & Bormann C. (2014), RFC 7252, The Constrained Application Protocol (CoAP) (read 4.12.2020). URL: https://www.rfc-editor.org/info/rfc7252.

[27] Banks A., Briggs E., Borgendale K. & Gupta. R. (2019), MQTT Version 5.0, OASIS Standard (read 7.12.2020). URL: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[28] Yu F., Liu J., He Y., Si P. & Zhang Y. (2018) Virtualization for distributed ledger technology (vdlt). IEEE Access 6, p. 25019–25028.

[29] El Ioini N. & Pahl C. (2018) A review of distributed ledger technologies. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 11230 LNCS, p. 277–288.

[30] Bashir I. (2017) Mastering blockchain : distributed ledgers, decentralization and smart contracts explained. Packt, Birmingham, England ; Mumbai, India, 1st ed., 531 p.

[31] Seth S. (2021), Public, Private, Permissioned Blockchains Compared (read 13.1.2021). URL: https://www.investopedia.com/news/public-private-permissioned-blockchains-compared/.

[32] Buterin V., Ethereum Whitepaper (read 13.1.2021). URL: https://ethereum.org/en/whitepaper/.

[33] Hyperledger White Paper Working Group (2018), An Introduction to Hyperledger (read 11.12.2020). URL: https://www.hyperledger.org/wp-content/uploads/2018/08/HL_Whitepaper_IntroductiontoHyperledger.pdf.

[34] Oracle, What Is a Database? (read 18.05.2021). URL: https://www.oracle.com/database/what-is-database/.

[35] JavaTPoint, Types of Databases (read 24.05.2021). URL: https://www.javatpoint.com/types-of-databases.

[36] IBM Cloud Education, Relational Databases (read 19.05.2021). URL: https://www.ibm.com/cloud/learn/relational-databases.

[37] IBM: ACID properties of transaction (read 25.2.2022). URL: https://www.ibm.com/docs/en/cics-ts/5.4?topic=processing-acid-properties-transactions.

[38] Microsoft: Non-relational data and NoSQL (read 25.2.2022). URL: https://docs.microsoft.com/en-us/azure/architecture/data-guide/big-data/non-relational-data.

[39] Mongodb: What is NoSQL? (read 25.2.2022). URL: https://www.mongodb.com/nosql-explained.

[40] Cattell R. (2010) Scalable sql and nosql data stores. SIGMOD Record 39, p. 12–27.

[41] Hecht R. & Jablonski S. (2011) Nosql evaluation: A use case oriented survey. Proceedings - 2011 International Conference on Cloud and Service Computing, p. 336–341.

[42] VMware Tanzu, Event streaming: Build applications that keep up with business (read 25.4.2022). URL: https://tanzu.vmware.com/event-streaming.

[43] Kafka 2.7 Documentation (read 27.1.2021). URL: http://kafka.apache.org/documentation/.

[44] Waehner K. (2020), Apache Kafka and Blockchain: Friends, Enemies, or Frenemies? (read 27.1.2021). URL: https://dzone.com/articles/apache-kafka-and-blockchain-friends-enemies-frenem.

[45] Waehner K. (2020), Is Apache Kafka a Database? (read 18.05.2021). URL: https://kai-waehner.medium.com/is-apache-kafka-a-database-ddc310898f5c.

[46] Microsoft Azure, What is a virtual machine (VM)? (read 25.4.2022). URL: https://azure.microsoft.com/en-us/overview/what-is-a-virtual-machine/.

[47] Celesti A., Mulfari D., Fazio M., Villari M. & Puliafito A. (2016) Exploring container virtualization in iot clouds. 2016 IEEE International Conference on Smart Computing, SMARTCOMP 2016 .

[48] Kurek T. (2019) Unikernel network functions: A journey beyond the containers. IEEE Communications Magazine 57, p. 15–19.

[49] Vmware: What is desktop virtualization? (read 25.2.2022). URL: https://www.vmware.com/topics/glossary/content/desktop-virtualization.html.

[50] Citrix: What is desktop virtualization? (read 25.2.2022). URL: https://www.citrix.com/solutions/vdi-and-daas/what-is-desktop-virtualization.html.

[51] IaaS vs PaaS vs SaaS Enter the Ecommerce Vernacular: What You Need to Know, Examples and More (read 27.1.2021). URL: https://www.bigcommerce.com/blog/saas-vs-paas-vs-iaas/.

[52] hermin88, The key difference between on-premise, SaaS, PaaS and IaaS (read 25.2.2022). URL: https://forum.huawei.com/enterprise/en/the-key-differences-between-on-premise-saas-paas-iaas/thread/780671-893?page=1.

[53] Fayos-Jordan R., Felici-Castell S., Segura-Garcia J., Lopez-Ballester J. & Cobos M. (2020) Performance comparison of container orchestration platforms with low cost devices in the fog, assisting internet of things applications. Journal of Network and Computer Applications 169.

[54] IBM Cloud Education, Containers (read 26.5.2022). URL: https://www.ibm.com/cloud/learn/containers.

[55] Eldridge I. (2018), What Is Container Orchestration? (read 1.2.2021). URL: https://blog.newrelic.com/engineering/container-orchestration-explained/.

[56] solsson / dockerfiles (read 2.6.2022). URL: https://github.com/solsson/dockerfiles/blob/921f762a5c835f48f5b0fbbcedff03c0a940927e/kafka/Dockerfile.

[57] Kafka download (read 3.5.2021). URL: https://kafka.apache.org/downloads.

[58] Edenhill M., librdkafka - the Apache Kafka C/C++ client library (read 3.5.2021). URL: https://github.com/edenhill/librdkafka.

[59] mikilian / arm32v7-mariadb-alpine (read 2.6.2022). URL: https://github.com/mikilian/arm32v7-mariadb-alpine.

[60] Sidhpurwala H. (2019), Understanding random number generators, and their limitations, in Linux (read 26.5.2022). URL: https://www.redhat.com/en/blog/understanding-random-number-generators-and-their-limitations-linux.

[61] Epoch & Unix Timestamp Conversion Tools (read 26.5.2022). URL: https://unixtime.org/.