UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Martin Kisand

Serverless Data Pipelines for IoT Data in Edge and Cloud Environments using Microsoft Azure

Bachelor's Thesis (9 ECTS)

Supervisor(s): Shivananda. R. Poojara, MSc

Tartu 2023

Serverless Data Pipelines for IoT Data in Edge and Cloud Environments using Microsoft Azure

Abstract:
Advancements in wireless connectivity and smart device technologies have contributed to the fast growth of Internet of Things (IoT) devices. Growth of IoT networks is generating increasing amounts of heterogeneous raw data that has to be analysed in real time to enable effective data based decision making. Challenges with latency, bandwidth and cost of cloud centric IoT solutions have driven adoption of edge computing paradigm to bring computing closer to data source. To harness edge and cloud computing resources together for continuous IoT data, serverless data pipelines can be designed taking advantage of Function as a Service (FaaS) paradigm. In this paper serverless data pipeline is proposed for real time IoT use case using Microsoft Azure cloud computing platform. Proposed forced audio to text alignment pipeline was tested to evaluate its performance consistency, reliability and how performance was affected by computing resources in pipeline. Pipeline proved to be mostly reliable with few failures but end-to-end execution times were rather inconsistent. Adding CPU and working memory to serverless function performing forced alignment lowered function execution times with increase of working memory from 3.5GB to 7GB. Raising working memory further to 14GB did not achieve better results compared to 7GB with current test scenario. Lowest working memory setting proved to be most cost effective because pricing is based on available computing resources and doubling or quadrupling resources did not have that much of an impact on performance as it had on cost. With proposed serverless pipeline implementation Azure did not seem to offer cost effective FaaS options considering smaller scale IoT applications that need custom functionality because premium plan can not scale to zero instances and lowest working memory setting is 3.5GB. In case of smaller scale IoT applications where functionality offered by Azure function consumption plan is not sufficient other cloud service providers could offer more suitable solutions with lower costs.

Keywords:
Internet of Things, cloud computing, edge computing, serverless data pipelines, Function as a Service (FaaS)

CERCS: P170 Computer science, numerical analysis, systems, control

Serverivabad andemkonveierid esemevõrgu andmete töötlemiseks kasutades servtöötlust ja pilvtöötlust Microsoft Azure platvormil

Lühikokkuvõte:

Arengud traadita andmeside võimekuses ja nutikate seadmete tehnoloogias on panustanud asjade interneti seadmete arvukuse kiiresse kasvu. Asjade interneti kasv toodab järjest suuremas koguses erisuguseid töötlemata andmeid, mida on vaja reaalajas analüüsida, et nende abil saaks tõhusalt teha andmepõhiseid ostuseid. Pilvepõhised esemevõrgu andmetöötluslahendused on problemaatilised seoses kõrge latentsuse, hinna ja ribalaiuse kasutusega ja sellest lähtuvalt on rohkem hakatud kasutama servtöötluse põhist lähenemist, et tuua admetöötlus andmete tekkimise kohale lähemale. Selleks et saada kasu voona tekkivate esemevõrgu andmete töötlemiseks nii pilv- kui servtöötlusest samaaegselt, on võimalik luua serverivabu andmekonveiereid, mis kasutavad ära eeliseid, mida pakub funktsioon teenusena lähenemine. Selles töö käigus pakutakse välja serverivaba andmekonveieri lahendus esemvõrgu andmete reaalajas töötlemiseks kasutades Microsoft Azure pilvearvutusteenuste platvormi. Välja pakutud andmekonveierit heli- ja tekstifaili jõuga vastavusse viimiseks testiti katsete käigus, et hinnata selle järjepidevust ja usaldusväärsust ning kuidas kasutatava arvutusvõimsuse muutmine mõjutab jõudlust. Välja pakutud serverivaba andmekonveier oli küllaltki vastupidav ja esines vaid mõni ebaõnnestunud katse, kuid andmekonveieri töötluseks kuluv aeg ei olnud eriti järjepidev. Protsessori võimsuse ja töömälu lisamine serverivaba funktsiooni tarbeks, mis tegeles heli- ja tesktifaili vastavusse viimisega, vähendas ülesandeks kuluvat aega, kui töömälu suurendati 3.5 gigabaidilt 7 gigabaidini. Praeguse testimise käigus täiendav töömälu suurendamine 14 gigabaidini enam funktsiooni täitmiseks kuluvat aega ei vähendanud. Kõige madalam töömälu seadistus osutus kõige kulutõhusamaks, sest serverivabade funktsioonide hind tuleneb kättesaadavate arvutusressursside hulgast ja nende kahe- või neljakordistamise tagajärjel tõusevad ka kulud kaks või neli korda, kuid jõudlus ei paranenud enamasti samas suurusjärgus. Lähtudes selles töös välja pakutud serverivaba andmekonveieri lahendusest jäi mulje, et Azure ei paku väiksemate rakenduste jaoks kulutõhusaid võimalusi, kui need vajavad rohkem funktsionaalsust, kui suudab pakkuda tarbimispõhine plaan. Võimekam premium plaan suudab küll pakkuda vajaliku funktsionaalsust, aga seda pole võimalik vähendada alla ühe töötava instantsi ja väikseim seadistatav töömälu maht on 3.5GB, mistõttu tekib pidevalt arvestataval määral kulusid, isegi kui tegelikult andmeid ei töödelda. Väiksemate rakenduste puhul, mis vajavad rohkem funktsionaalsust, kui pakub Azure'i pakutav tarbimisplaan, tasuks kaaluda teisi teenusepakkujaid, kes pakuvad selliseks olukorraks sobivaid lahendusi madalama hinnaga.

Võtmesõnad:

esemevõrk, pilvtöötlus, servtöötlus, serverivaba andmekonveier, funktsioon teenusena

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Contents

1 Introduction

As a result of recent developments in the areas of wireless networking and smart device technologies there has been a fast growth of Internet of Things (IoT) devices. Many embedded devices like vehicles and manufacturing equipment that previously were part of isolated systems are now equipped with smart device capabilities and are able to connect to the internet. In 2019, IoT devices made up half of internet connected devices with 10 billion active connections and this number is predicted to rise to 29.7 billion by 2027 [36].

Increasing adoption of IoT technology enables to generate high amounts of raw data from different types of sensors [29]. This data can provide valuable insight about observable systems improving decision making and helping to automate tasks but in addition has to be processed and analysed to extract usable information [39]. Considering that processing high amounts of data needs sufficient computing power there is a need for local servers or cloud services. Cloud services are the more practical choice in many scenarios offering easier setup and lower costs as well as being scalable when necessary [15]. In order to use cloud computing for processing IoT data, this data has to be transferred to the cloud. Deployment of 5G wireless technology has improved connectivity of smart devices but struggles to keep up with huge amounts of data generated by IoT networks [28]. Given that IoT applications are often event driven, there is a requirement to process incoming data in real-time to react effectively. This poses challenges on how to achieve low latency and bandwidth while processing data fast and reliably [3].

Working with heterogeneous data stream from IoT systems can be simplified by designing serverless data pipelines [13]. This approach takes advantage of the Function-as-a-Service (FaaS) paradigm [4] where data pipeline is divided to separate containerized stateless functions that consume input data and produce an output. Leading cloud service providers like Amazon Web Services[1] and Microsoft Azure[2] are offering pay-per-use FaaS solutions that can be seamlessly integrated with IoT networks. Building serverless data pipelines minimizes development effort because service provider takes care of the infrastructure management ensuring on demand scalability while keeping costs predictable.

In order to reduce network load and increase security and reliability of IoT applications some data processing tasks can be completed prior to transferring data to the cloud [3]. With the edge computing model idle computing resources of smart devices can be utilized to process data close to the source. This enables to filter or compress data that will be sent to the cloud decreasing network load, encrypt data to enhance security and make mission critical applications more resilient in the case of networking problems.

[1] `https://aws.amazon.com/lambda/`

[2] `https://azure.microsoft.com/en-us/products/functions/`

Designing serverless data pipelines with products like AWS Greengrass[3] and Azure IoT Edge[4] makes running serverless functions easy on the edge devices.

The aim of this thesis is to propose a simple architecture for serverless data pipeline using Microsoft Azure, and measure its performance with tests. Proposed serverless data pipeline design takes into consideration the scope of the real time IoT use case. During testing performance, durability and effectiveness of the pipeline are measured in simulated scenarios where multiple edge devices would generate data. Conclusions are made about the results of testing with following suggestions for possible future improvements.

This paper consists of following chapters: Section 2 describes overall background of this work, covering internet of things use cases and architecture, motivation to use edge computing and using serverless pipelines for IoT data. In Section 3 most relevant previously done work is represented. Section 4 describes real time IoT use case for audio data and in Section 5 pipeline implementation is proposed for the use case. Section 6 gives overview how experiment was set up and conducted. In Section 7 results are presented and discussed. Finally Section 8 ends the paper with conclusions.

[3] https://aws.amazon.com/greengrass/

[4] https://azure.microsoft.com/en-us/products/iot-edge

2 Background

This section covers background that is relevant for this paper. Section 2.1 gives general overview of IoT and different situations that can benefit from using this paradigm. Main elements of IoT and architecture is described. Section 2.2 explains how edge computing can benefit IoT systems increasing their efficiency and reliability. In Section 2.3 covers using serverless data pipelines for event driven IoT data.

2.1 Internet of Things

The Internet of Things refers to the interconnectedness of physical objects like devices, vehicles, buildings, and other items embedded with electronics, software and sensors which enables these objects to collect and exchange data [23]. IoT device is an item that is part of the IoT network playing the role of generating and exchanging data. IoT devices are connected to the existing internet infrastructure allowing to gather information in real time and to control devices remotely. This means IoT devices can be a broad range of different items. It can be minimalistic RFID tag with sensor capabilities [41] to transfer information about it, something more complex like smart household thermostat that is connected and controlled through local network or smartphone, and even highly complex self-driving autonomous vehicle. Multi-purpose devices including smartphones and personal computers can act as IoT devices depending on how they are used.

Main characteristics of IoT could be seen as following [37]:

1. Each smart device must have unique identification. As the number of devices in a network will get very high there must be suitable solution for addressing such as IPv6.

2. Devices with sensors that can collect information from physical world.

3. Communication with devices to access collected data and change device state. This could be achieved with wired connections in some cases but most likely by using wireless communication technologies like RFID, Bluetooth, Wi-Fi etc.

4. Collected data must be stored and analysed to extract usable information. This means that network contains centralized infrastructure with storage space and analytical tools with capable algorithms for processing raw data that can support heterogeneous platforms.

5. Visualization to enable users to interact with analysed data through application solutions to get information about physical environment.

The IoT has the potential to revolutionize everyday life and industrial systems by connecting objects to the internet and allowing them to send and receive data enabling

to access information from remote sensors and control physical world [23]. On one hand, This enables to automate more processes that previously depended more on human intervention and decision-making, making these processes more efficient. On the other hand, there are still multiple challenges to be addressed before IoT could reach its true potential. The concept of IoT itself is not new [18] but its current relevance lies in accessibility and quickly growing usage and capabilities thanks to the technological improvements.

2.1.1 Use cases and challenges of IoT systems

There is wide area of situations where using IoT systems can offer considerable value. In general, increasing number of IoT devices and networks leads to increased amounts of real time data collection [29]. IoT data is automatically transferred using M2M protocols and can be processed near real time offering up to date insight of the state of systems and environments [19]. Having near real time awareness of physical properties measured by sensors on IoT devices enables efficient decision making and controlling of IoT systems. Developments in areas like machine learning, quantum computing and 5g/6g wireless technologies create new opportunities to get even more value from IoT data [28]. These new technologies help IoT data to be collected and processed more efficiently and reliably whilst also helping to keep up with increasing amounts of generated data. Although earlier predictions of future numbers of IoT devices have been somewhat reduced it's still clear that growth is going to continue and predicted to reach 29.7 billion IoT devices by 2027 [36].

There are multiple examples of IoT use cases. Considering the scope of factories where industrial machinery is used, predictive maintenance can be implemented to observe conditions of machinery and predict when some part of equipment could fail [10]. In this case machinery will be equipped with sensors that gather data about its operation for example vibrations. When signs of wearing out are seen, maintenance will be scheduled to avoid unscheduled off time and reduce further possible damage. IoT can be also used for supply-chain optimization by tracking inventory levels, monitoring shipments and providing real time insights for logistics and distribution [1]. Combining IoT with technologies like blockchain could further increase transparency and traceability of supply-chain processes. In agriculture, IoT sensors and analytics can be used to monitor soil conditions, weather patterns and crop health [17]. This enables farmers to make data-driven decisions optimizing resource usage, resulting in higher yields. IoT solutions in electrical grids enable to optimize electricity distribution and demand management to keep grids balanced and improve overall stability [33]. In healthcare smart wearables can track health metrics providing valuable data for individuals and healthcare professionals raising chances for preventive healthcare and giving patients suggestions to increase their well-being [9].

Currently there are still multiple challenges that are slowing the growth of IoT

systems. Ongoing chip shortage has been one of the most recent problems reducing available components to manufacture IoT devices and thus increasing cost of production [27]. Organizations planning to use IoT solutions also have to consider the increase of risks in data security and privacy and how to cope with it [20]. IoT devices can collect vast amounts of data about behaviour of individuals that could result in the decrease of anonymity and increase the ability to predict the habits and preferences of the individual. This information could be used for business interests resulting damage to the users and the breach of data privacy regulations. Mistakes in information-handling could cause data leaks whilst cyber attacks can result in data breaches [16]. Having more internet exposed IoT devices also increases attack surface. In addition to possible data related issues, IoT devices can increase risk of service disruption in case of cyber attacks causing downtime or the malfunctioning of IoT systems. This is especially relevant in context of vital services like smart electrical grids or mission critical devices like autonomous vehicles. the current state of IoT landscape is diverse and dynamic where different organizations and vendors are developing multiple technologies, communication protocols and applications [14]. There are multiple notable efforts to promote standardization for IoT systems like work done by Industry IoT Consortium[5]. Despite of this, there is still lots of diversity reducing the interoperability of IoT applications increasing effort of implementing safe and compatible IoT solutions. Researchers and organizations keep working to solve current and emerging challenges of IoT and with further maturing of the IoT field, industry standards will likely continue to evolve.

2.1.2 IoT Architecture

There is no officially adopted IoT reference architecture. Main reason behind this is that under IoT concept there are multiple different technologies and application domains which has resulted in having a great variety of different solutions [14]. IoT data processing solution presented in this paper is based on three tier reference architecture presented in 'Cloud Customer Architecture for IoT (CCAIoT)' [24]. This three tier architecture pattern consists firstly of the edge tier with the goal of collecting data from edge nodes using proximity network. Secondly, the platform tier which is hosting a service platform that covers data transformation, analytics and operations. In addition, it acts as a middle layer mediating upstream data flow and downstream control flow. Thirdly, the enterprise tier is controlling other tiers while implementing domain-specific applications, decision making systems and providing interfaces to end users.

Currently described architecture however has no focus on utilizing edge computing for data processing. To accommodate the use of edge and cloud computing together a more suitable architecture could be derived by splitting the edge tier into two separate tiers (device and edge tier) and a third one as cloud tier [31]. This as seen on Figure 1,

[5]https://www.iiconsortium.org/

allows some of the data to be already processed in the edge tier which was previously was only done in cloud tier.

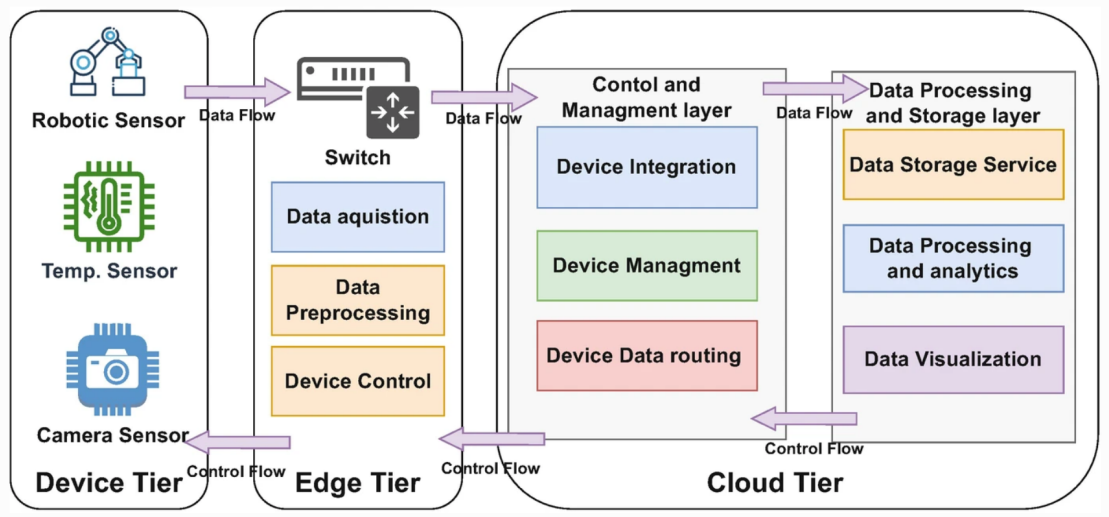

Figure 1. High level Three tier IoT architecture [31]

2.2 Edge computing

As numbers of IoT devices are growing quickly it raises challenges on how to access data generated by them. In many cases IoT networks can grow into vast systems that can generate high amounts of raw data. For example, in the setting of an industrial IoT environment like a manufacturing plant, there can be machinery, equipment, surveillance systems, vehicles, inventory and warehouse systems that are all embedded with sensors generating and sharing data [25]. One solution to process this raw data is to send it to cloud where there are enough computing resources available to handle this task. This however can be demanding on the network, especially because IoT devices are often connected with wireless technologies that are more limited in bandwidth and latency. There can be even larger IoT systems like smart cities [38] and electrical grids [33] that generate even more data. In other use cases like environmental monitoring systems, the number of devices can be lower in specific areas but they might be positioned in remote locations with poor available network connection and limits on power consumption [6]. As there is often requirement to analyse IoT data near real time [40] there should be good enough network capability to achieve this. New wireless technologies like currently deployed 5G and upcoming 6G [28] can help with this challenge but might require costly investments and thus they are only part of the solution. Amounts IoT data

is predicted to keep increasing in the future and it's more likely that improvements of wireless technology help to keep up with this increase rather than being able to upload all IoT raw data in all situations cost efficiently.

To reduce the amounts of IoT data needed to upload to the cloud some of it could be processed close to the source of creation. Distributed computing paradigm called edge computing can be used to utilize locally available computational resources to pre-process IoT data on the same hardware that is collecting it [22]. IoT devices can have some built-in processing power needed for their usual operations that has some idle time during which it could be used for local data processing. In context of video data [26], this locally available computing resource could be used to reduce the amount of uploaded frames, compress data before uploading or only starts sending data when movement is detected on footage. For this use case there are existing solutions like Apache NiFi for creating edge computing clusters that combine resources of available edge devices [12]. This approach could significantly reduce network load. Main constraints of edge computing are having limited computing resources related to small form factor, low power consumption and cost considerations of edge devices.

Besides edge computing, fog computing [5] could be used for similar purpose to bring computation closer to the source of data. Fog and edge computing are used interchangeably in some cases but fog layer is generally considered to be a local intermediate layer between edge devices and cloud. Fog layer could have dedicated fog servers for performing computations or use local network devices like gateways and routers that are close to edge devices and have idle processing power available. Fog devices have generally more processing power than individual edge devices. Fog computing and edge computing could be used together or as standalone solution depending on the available resources and requirements.

In addition to relieving stress on network by reducing bandwidth usage and keeping latency low, edge computing can also improve security of IoT systems by keeping sensitive data localized and pre-processing it before uploading [43]. This can reduce risks of possible situations where privacy or compliance requirements would not be fulfilled. In addition to keeping data safe, edge computing can considerably improve resilience of mission critical systems [35]. In situations where failure or downtime of edge device would impose safety risks or cause severe operational disruptions, usage of embedded processing capabilities can ensure that critical functionality is provided in situations of offline operation or poor internet connection. For example, IoT devices like autonomous vehicles should be able to maintain sufficient functionality while losing network connection to provide safety and avoid disrupting traffic and thus can greatly benefit from embedded edge computing capabilities.

2.3 Serverless Data Pipelines for IoT data

Data pipeline refers to the collection of data processing elements that are connected in series where the output from one element is used as an input for the next one. This type of automated data processing enables to feed raw data from multiple sources to the first element of pipeline and extract processed information from the data sink after the last processing element. After setup, data pipelines offer a quick and simple solution to process data without constantly focusing on all individual processing steps. As pipeline elements can be independent from each other they are able to run simultaneously, catering for continuous incoming data stream. Data pipelines can be part of the solution to simplify processing real time heterogeneous IoT data from multiple types of sources [32].

Data pipelines processing elements could be run using cloud services and local hardware or a combination of them. Because of the event driven nature of IoT data the need for processing power can fluctuate a lot [8]. Investing in local servers that can handle situations where pipelines need sufficient scaling would be costly and this would be inefficient in case where most of the time this resource would stay idle. A similar situation would apply to the use of cloud based virtual machines where billing is based on the number of machines not on their usage and event driven load would lead to cost inefficiency. There are also cloud centric IoT data collection platforms that offer simple out of the box solutions like Cayenne[6], Thingspeak[7]. These solutions mostly send data from IoT devices to the cloud where analytics and processing will take place as well as managing and controlling devices. Using cloud centric solutions can lead to latency and bandwidth problems when amounts of data increase and network struggles to keep up with it [31].

Integration of function as a service (FaaS) cloud computing paradigm into data processing pipelines results in the creation of serverless data pipelines. Serverless function is a stateless function that is initiated on demand [7]. It consumes input data, performs processing and returns output data. After producing output, serverless function is terminated and computing resources used by it are released. Because of such nature they are not suitable for data storage and additional storage solution should be added to retain data after function instance has been terminated. Serverless functions offer good scalability because new instances can be created at need and service provider allocates necessary resources for them to run. Serverless functions are generally priced for actually used recources and number of function runs. This makes them a good choice for real time IoT use cases where the need for computing recources can fluctuate considerably [32]. Serverless pipelines can make use of FaaS by including serverless functions as pipeline elements to build efficient and scalable pipelines for processing high amounts of IoT data in real time. With this approach, developers can write standalone functions

[6] `https://developers.mydevices.com/cayenne/features/`

[7] `https://thingspeak.com/`

that will be initiated when needed and don't need to consider infrastructure and scaling. Pipelines data processing functions can be designed to run both on cloud through service provider or on local hardware using solutions that emulate serverless environment like AWS Greengrass and Azure IoT Edge [11]. In edge computing use cases, this approach can be used to utilize available local computing resources together with cloud computing in order to increase pipeline efficiency. In such situations developers need to just write a function once and initiate it on the optimal environment in any given scenario.

3 Related Work

In this section, relevant research is covered about using serverless computing in data pipelines to cater for applications working with real time IoT data.

Poojara et al. [32] proposed three serverless data pipeline approaches looking at different video and audio processing scenarios like forced alignment, speech to text conversion and real time object identification. Proposed serverless data pipeline approaches were based on off-the-shelf data flow tool, object storage service and MQTT queue. Experiment results showed that MQTT delivered best results with forced alignment and data flow tool with video processing. Object storage service managed to produce decent results compared to other solutions.

With another work Poojara et al. [31] investigated differences between main cloud computing service providers AWS, Microsoft Azure and Google Cloud in context building serverless data pipelines for IoT data. In this work, authors proposed AWS and Azure based data pipeline solutions and implemented them for the use case of predictive maintenance for industrial machinery. While testing implementations, they found that there was considerable performance difference between edge tier solutions of these platforms caused by different architectural deployments. Results showed that Azure based pipelines used more memory and CPU power compared to AWS because AWS runs edge modules on native host environment whereas Azure uses Docker containers as extra layer. However, AWS pipeline implementations were slower in terms of processing time caused by extra messaging queues.

Study conducted by Pogiatzis and Samakovitis [30] developed and presented architecture for event based ETL using serverless technologies provided by AWS. They followed with evaluating their pipeline proposition with tests to assess its performance, consistence and reliability as well as cost effectiveness. They concluded that proposed solution is suitable for numerous IoT related fields using event based reactive systems while reducing infrastructural overhead. This model was especially suitable for sparse event processing needs.

Das et al. have [11] presented framework for optimizing performance using dynamic task placement in serverless edge-cloud platforms. They were able to develop models for predicting end-to-end cost and latency while running functions in edge or cloud environments. This approach allows to create systems that can dynamically choose when to run serverless functions on edge or cloud layer to maximize efficiency.

4 Real Time IoT Use Case

In this paper, focus lies on real time IoT use cases where IoT data is produced in continuous streams and needs to be processed near real time. Real time processing for IoT data is useful in multiple situations providing immediate insight and decision making that enables to take event based actions effectively when needed for example in context of medical systems, traffic control or safety and security systems. It also enables to build synergy in vast IoT systems like smart cities [38]. If data is processed in real time, IoT devices in different parts of network can exchange data and use joint decision making to better understand and influence processes happening in the physical world.

One of the data types that could be used in the scenario of near real time IoT use case is audio data. Audio data could be used to manage urban traffic, monitor public spaces, detect gunshots and other emergency sounds, monitor livestock in farms or to observe smart home environment [34]. Depending on the situation, different approaches can be used to analyse this audio data. Pattern recognition or sound classification algorithms [21] could be used to monitor environment of interest. Speech recognition and voice biometrics could be used to identify individuals and to extract information from their speech [42].

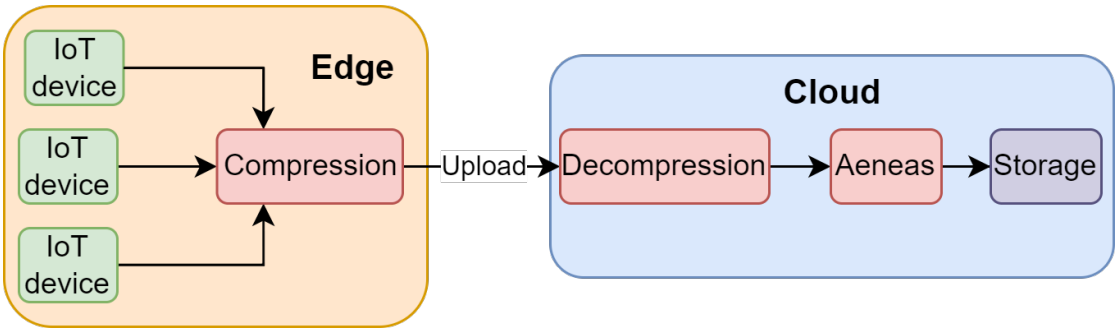

Figure 2. Flow of Aeneas data pipeline

Aeneas is a forced alignment tool that takes an audio file and a text file as input and creates a synchronization map based on the text fragments that are recognized[8]. This Python/C based library is suitable to investigate the processing of IoT audio data and will give general representation of different methods that could be used in this case. Audio data is on average considerably larger and more resource intensive to analyze compared to text based data. When multiple IoT devices are recording and uploading audio files it can become bandwidth intensive and put strain on the network. In addition, considerable amounts of computing power will be needed to process collected data.

[8] https://www.readbeyond.it/aeneas/docs/

Thus it's a challenging scenario for real time IoT use case that could benefit from using serverless pipelines for data processing.

Figure 2 describes flow of Aeneas based real time IoT application:

- IoT devices record audio data
- Audio data is compressed in edge tier to reduce network load
- Data is uploaded to cloud server
- Data is decompressed in cloud
- Aeneas creates synchronization map based on audio file and given text
- Synchronization map is stored in cloud storage

5 Proposed Serverless Data Pipeline for IoT Use Case

Considering the use case described in the previous section, serverless data pipeline is designed using commercially available services of Microsoft Azure[9] cloud computing. Azure is one of the main cloud computing service providers besides others like AWS and Google [2]. Azure was chosen because of it's wide use and that it was available to use under student licence. Tools provided by Azure can cover all needs for implementing serverless pipeline for current use case. Main focus is to keep this build simple and straightforward using most intuitive tools available in Azure suite while following serverless pipeline architecture for use case described in this paper.

Architecture of pipeline implementation can be seen on Figure 3. In edge tier are devices that are collecting audio data, compressing it and uploading compressed data to Azure blob storage container in cloud. When new file upload is detected, SF1 is triggered that decompresses audio data and forwards output to second storage container. From that container SF2 is triggered that uses Aeneas module to perform forced alignment and resulting synchronization map is stored in final third storage container where it can be accessed for further use. All modules and serverless functions in this project were developed using versions of Python 3. It's a suitable language in the current situation because its simple implementation and wide use in data analytics. However the programming language that is most supported by Azure platform is C# and some functionality is not supported in other languages. IoT edge runtime has a capability to run serverless functions on an edge device but at the current time this is only possible when functions are written using C# and are not available for Python. This reduces options when designing this pipeline. All code written for this paper is accessible in public Github repository[10].

5.1 Edge tier

In IoT systems there are usually numerous similar edge devices in the edge layer that simultaneously generate data. During this implementation for practical reasons a single RPi 4B with running ARM32 Raspbian OS was used to simulate edge layer. This was achieved using Azure IoT Edge runtime that enables to create edge device deployment on RPi. Edge runtime is installed on edge devices where it enables to deploy, run and manage containerized modules using Moby engine[11]. Once modules are developed with providing suitable dependencies, they can be run on different hardware using IoT Edge taking care of module management on local hardware. If Python based serverless

[9] https://azure.microsoft.com/en-us/

[10] https://github.com/mk-3rd/serverless_data_pipelines_for_IoT_data_in_edge_and_cloud_environments_using_microsoft_azure

[11] https://mobyproject.org/

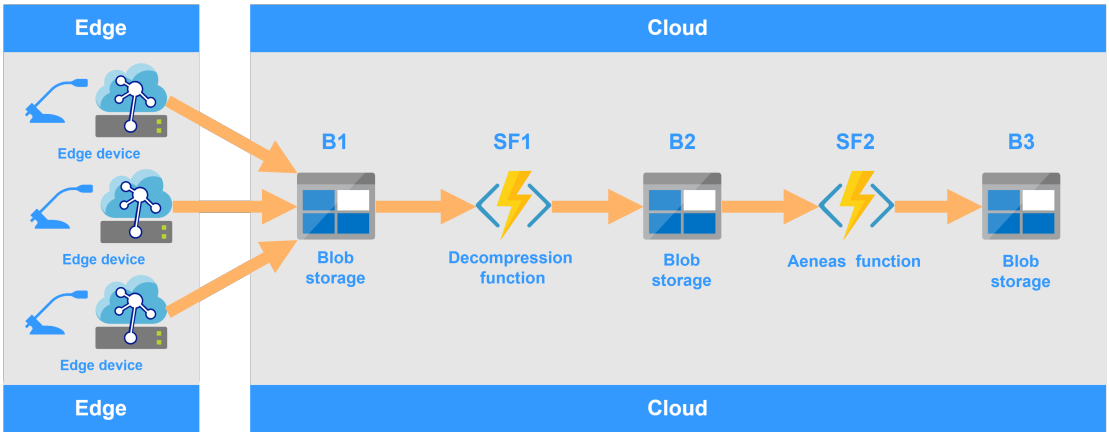

Figure 3. Architecture of serverless pipeline implementation with Azure

functions would be supported on IoT edge, they would be a more convenient choice for the preprocessing of data on edge devices and could be run on cloud as well.

In this design single Python 3.7 edge module is created for edge runtime to compress sample .mp3 data files with python zipfile and upload them to cloud storage using BlobServiceClient during each pipeline run. To connect edge device to cloud, IoT Hub service is used that is central part of Azure IoT suite. This service takes care of device management, bidirectional communication, security, monitoring and diagnostics simplifying working with numerous IoT devices. For commercial use IoT Hub has 3 paid tier levels. Suitable tier should be selected considering per day message and data limits. For current work free tier was suitable with message limit of 8000 messages per day.

5.2 Cloud tier

The cloud layer of the current pipeline implementation consists of two main elements fig2: blob storage containers and serverless functions. All three containers are based on the same storage account[12]. To host serverless functions on Azure cloud platform, there are different function plans to choose from based on available computing resources and functionality. Function plans can host multiple different functions and take care of scaling up when higher amount of function runs are detected.

From two main options, first is to use consumption plan[13]. This plan is the best available option to use with a serverless computing approach with pay per use pricing

[12] https://azure.microsoft.com/en-us/products/storage/blobs
[13] https://learn.microsoft.com/en-us/azure/azure-functions/consumption-plan

model. This plan will scale down to zero and up to 100 instances with linux based functions but there is no available information from Azure how much resources one instance would have exactly. Scaling down to zero will mean that there is higher impact of cold starts but it manages to reduce costs.

One relevant constraint with the function plan is that it does not support custom containers or Kudu[14] service to run commands in function container and thus installing Python modules is mainly constrained to using python package installer pip. Installing Aeneas module is very error prone because of dependencies failing to install and using pip for it can't be relied on. Custom container is needed to forward bash commands and run scripts to manually install Aeneas using specific github repository[15]. So Premium plan[16], which is the second function plan option covered in this paper should be used to run Aeneas function for this pipeline implementation. Besides proving more functionality compared to consumption plan, premium plan is priced differently. Premium plan has three tiers with different resources available for single instance. Plan 1 has single core and 3.5GB working memory, Plan 2 has two cores and 7GB working memory and Plan 3 has four cores and 14 GB working memory. For both premium and consumption plans costs are calculated based on usage time of vCPU cores and working memory, but for consumption plan there is a generous free grant[17]. Premium plans offer option to use pre-warmed instances that help reduce cold starts. Lowest setting for pre-warmed instances is one, meaning that the Premium function plan cannot scale down to zero instances and is thus generating costs even when there are no function executions. Especially with lower intensity applications this would have major impact on the overall cost of the pipeline.

When compressed audio files are uploaded to first to the Azure blob storage container it activates the event grid trigger to start the decompression function that is running on consumption plan. Event grid trigger adds complexity and cost to the pipeline compared to basic blob storage trigger but is needed to avoid delays with triggering[18]. According to Azure documentation, triggering functions running on consumption plan with a basic blob trigger can take up to 10 min which would not be suitable for real time IoT application. With premium plan there is no such problem. Decompression function uses Python 3.9 and decompresses .zip file to .mp3 file and forwards it to the next blob container. From there Aeneas function is triggered with default blob trigger. Aeneas function runs on premium function plan with custom Linux container. Aeneas module is also based on Python 3.9. Dependencies are installed with using Dockerfile and pip to ensure the proper working of the Aeneas module. Forced alignment is performed and the synchronization

[14] https://learn.microsoft.com/en-us/azure/app-service/resources-kudu

[15] https://github.com/qub-blesson/DeFog

[16] https://learn.microsoft.com/en-us/azure/azure-functions/functions-premium-plan?tabs=portal

[17] https://azure.microsoft.com/en-us/pricing/details/functions/

[18] https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-event-grid-trigger

map is stored in the third final blob container. This concludes a successful pipeline run. All code and related files of the edge module and serverless functions are stored in projects Github repository.

6 Experimental Design

Experiments were designed to investigate how the proposed pipeline performs when processing incoming audio data in a simulated scenario. Experimental design approach was derived from previous work done by Pogiatzis et al. [30] where AWS based serverless ETL pipeline was tested. The focus of the thesis lies on measuring performance, reliability and consistency of the pipeline. Performance was evaluated by measuring end-to-end data flow time and total edge function execution time (compression, decompression and forced alignment) under different loads and event frequencies. Also different premium function plans were tested for function running Aeneas to investigate how different amounts of computing resource affects function performance. Reliability was evaluated based on the amount of successful pipeline runs from a total set. Consistency was viewed to see if pipeline performance is consistent or not.

Single RPi 4B device was used to simulate edge tier that was connected to private network with advertised maximum upload speed of 200MB/s using Ethernet cable. Three different sized sample .mp3 audio files of 500kb, 5MB and 50MB were used to simulate different amounts of IoT devices generating data for the pipeline. 5 different event frequencies were used for file uploads in scenarios where sample file was uploaded after every – 1s, 3s, 10s, 30s and 60s. Each file was uploaded 10 times per file size and event frequency combination which results in 150 pipeline runs in total. This was repeated three times, once for each of the three premium function plans: Plan 1 with 1 vCPU core and 3.5GB memory, Plan 2 with 2 vCPU cores and 7GB memory, Plan 3 with 4 vCPU cores and 14GB memory. All premium plans were limited to a single instance to avoid scaling and to have a more accurate representation on how available resources affect the pipeline performance. Testing all three plans resulted in a total of 450 pipeline runs.

Experiments were manually triggered using device twins for every file, frequency and premium plan combinations. Device twin is a digital representation of the edge device in Azure cloud. It's kept in sync with IoT edge runtime that is running on the device. Updating module setting in digital twin results these settings being sent to the edge device while it's operating. This functionality was used to forward signal to start the experiment and the settings that should be applied.

Experiment data was collected from edge module logs in plain text format and for functions using Application Insights from where it was exported as .csv files. To combine these different log files into a single unified and cleaned .csv file Python 3.9 script was used. With this script also necessary result data was extracted and using Matplotlib[19] library charts were generated to get readable results.

[19] `https://matplotlib.org/stable/index.html`

7 Results and Discussions

The data results of the experiment runs about total function execution and end-to-end performance can be seen on Figure 4. Total function execution represents overall duration of three computing tasks in the pipeline (compression, decompression, forced alignment) during pipeline run, while end-to-end flow time represents the duration from the start of the pipeline until the results reach the data sink. One box of data on plot covers all experiment data for the respective file size and premium function plan. So for example when looking at a plot of total function execution with plan 1 then the box labeled 500KB represents 50 data points. These come from running the experiment with 5 different event frequencies 10 times for each. Experiments show that an increase in file size of 10 times from 500KB to 5MB results in about doubled function execution time and increasing file size 100 times to 50MB results in about 10 times longer processing time. This holds true for all 3 function plans. When looking at end to end flow times that in addition to processing includes upload and other file transfer and triggering delays in pipeline in general, it is at least double the time of processing or more. These results also have less consistency than processing times and that is mainly because of the file upload time. As files uploaded by edge device are quite large, especially with the 50MB file, and internet connection was not very stable with upload times taking in some cases almost around 100 seconds. Even with smaller files upload time variation was remarkable, like seen with 500KB files with plan 3.

Figure 5 shows more precisely how different function plans and file sizes affected the Aeneas function's performance. However, because of the experiment's design and constraints of the upload speed of the edge device all this data is not accurate. Experiment was set up so that the next upload could not start before the previous one was finished to avoid clogging uploading with multiple large files. If the upload was not finished before the next event interval deadline it was skipped and retried at the next one. So in the most extreme situation when the edge device should have uploaded ten 50MB files during 10 seconds, in reality it took a few minutes to achieve the goal of 10 uploads, doing it as quickly as possible at given moment. 500KB files were able to upload with 1 sec when connection was good but with 5MB files it was more realistic to achieve 10 second intervals and with 50MB files 30 sec intervals. This should be considered when looking at the data. When looking at performance differences on 5 it is visible that plan 2 performed considerably better than plan 1. This difference is higher with shorter file upload intervals that generate more strain on the Aeneas function. Between plan 2 and plan 3 there is no noticeable difference in performance. That indicates that current experiment was not able to put enough pressure on this function to make use of the extra computing resources provided by plan 3. When considering cost efficiency, it seems that working loads generated during the experiment were most efficient on plan 1. Plan 2 offered somewhat better performance but costs 2 times more and plan 3 proved to be an overkill for the current task with even higher costs but no noticeable performance

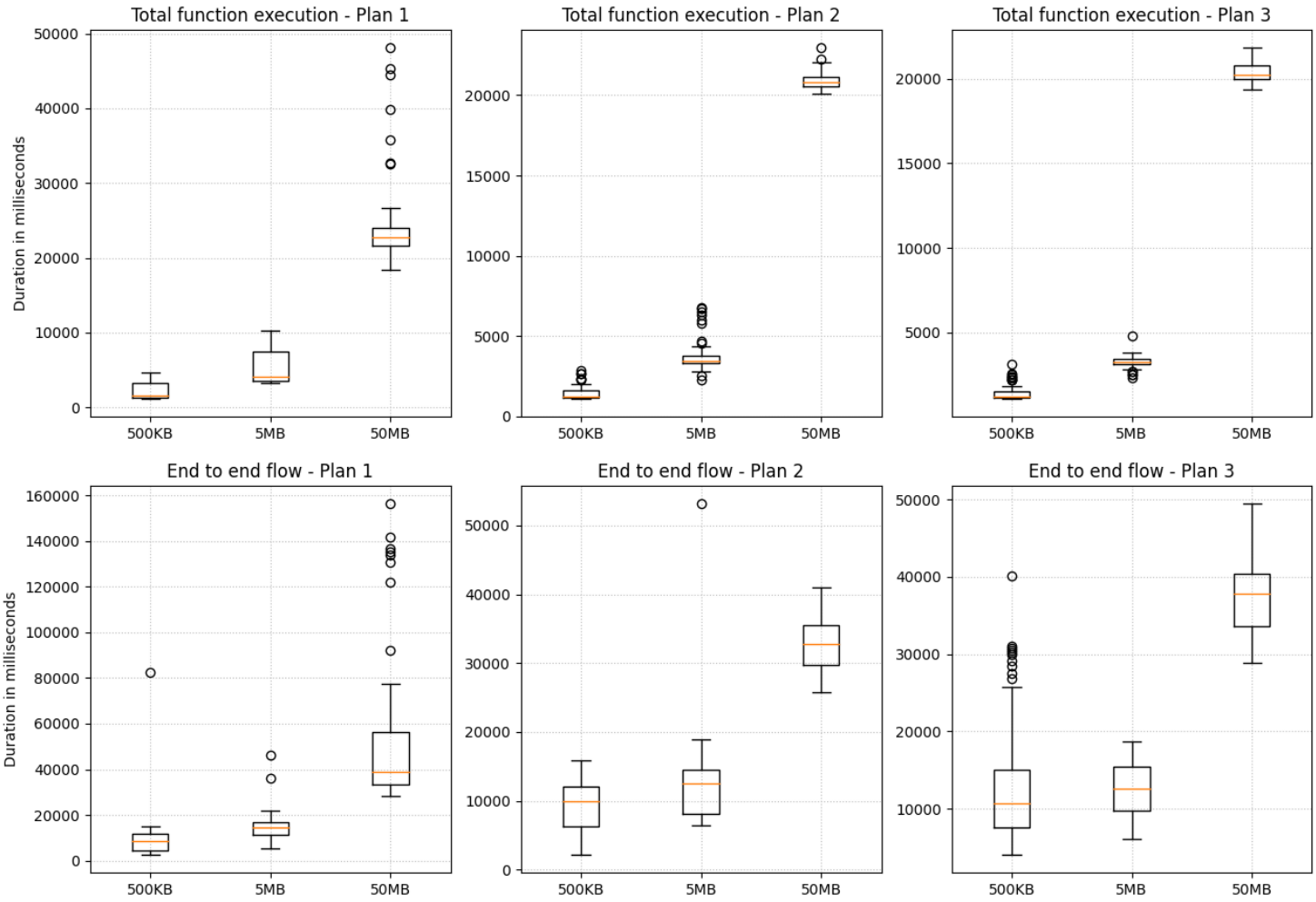

Figure 4. Total function execution and end to end flow times of pipeline over all test runs

improvements.

When looking at consistency of the pipeline's end-to-end performance and function runs it is quite inconsistent with considerable variance. End-to-end inconsistency was mainly caused by variance in upload time. Figure 5 shows that premium function's execution time variance was higher with more frequent events. That is somewhat explained with lower load when idle function receives first events and increasing strain when more events were arriving. In other cases, there were failures with function runs. That was the case with both the decompression function running on a function plan and the Aeneas function on a premium plan. By default, Azure functions have a policy to retry to run function up to five times on failure. So in cases of failures and retries there was a

holdup with failing event and extra computing resources were used. Because the next pipeline runs could already catch up with the one that was failing, even more computing resource was needed, resulting in longer execution times and more inconsistent runs.

In terms of reliability, out of 450 pipeline executions there were 2 that failed with end of life and zipfile related errors. These failures occurred because the decompression function was not able to decompress .zip file and did not send the audio file forward in pipeline. There were 24 failed invocations for the decompression function running on consumption plan and except for two times previously described, the retry policy was able to overcome failures. There were a few 503 server busy errors but many more were end of life errors from the Python module. To have a better insight of the underlying problem better error handling could have been used in the function code. Most of these failed function invocations took place in a short time period where 50MB files were uploaded in high event frequencies, so probably errors where related with higher data load. This did not reoccur when experiment was repeated for other function plans. In comparison Aeneas function running on premium plan also had 26 failed invocations but all were resolved with retry policy. There were 3 types of issues. One was that Aeneas module did not find text fragments from .xhtml file that was used for forced alignment. This file was held in a separate blob storage container and was loaded by the Aeneas function. There must have been a problem related to loading it properly in some instances. Second type of error was related to uploading the synchronization map to the final storage container. In that case, when synchronization map was created and saved on local function storage it either was not created properly or just failed to read for some other reasons. When Python tried to read this file to forward it to the storage container it resulted in a file not found error. Third type of issue was related to reading audio file when one of the Aeneas dependencies ffprobe gave error about not supporting audio file format for unknown reason. Nevertheless, these issues did not result in pipeline failures and just made the function executions slower because of raised use of computing resources.

In general pipeline proposed in this work was mostly reliable but if considering use cases when failures cannot be tolerated at all, there is need to implement extra measures to increase reliability and avoid failures. Code of functions used in this pipeline could also be improved to get better insights about errors. This could help with finding solutions to avoid failed invocations to increase the function's performance and consistency. Currently used method of compressing .mp3 files to .zip files made no meaningful difference in file size but was suitable for simulating data preproccessing. Different pipeline design options could also be explored to see how they affect overall performance.

In future works, the experiment design could be improved. Currently, there was an issue that the edge device was not able to upload files with the desired frequency. One solution could be to perform another set of experiments where events are triggered in cloud to just measure the performance of the serverless functions while removing the

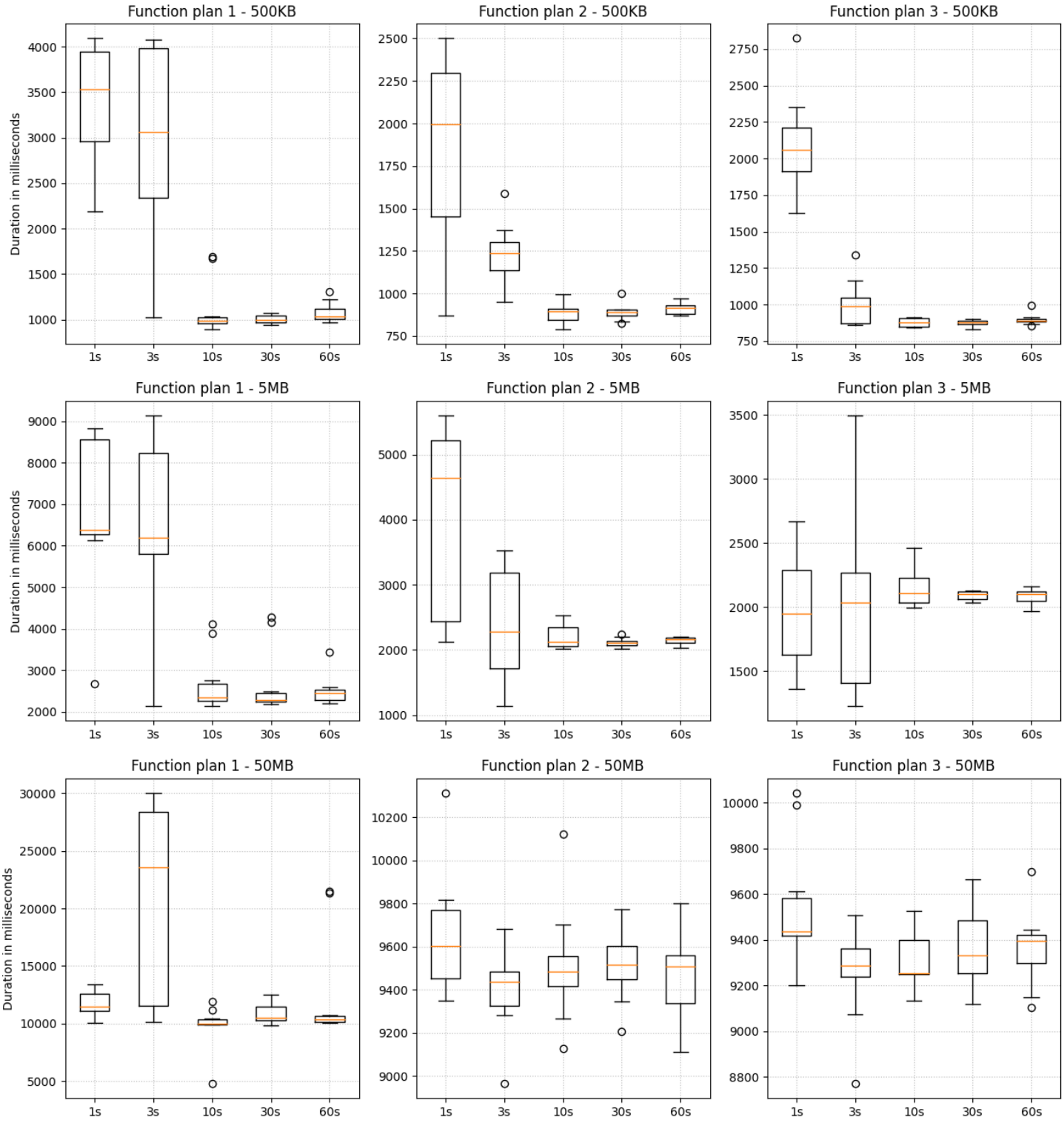

Figure 5. Aeneas function performance with different function plans and file sizes

edge device bottleneck. Also more edge devices with good internet connection could be added to distribute uploading file size and frequency.

A point could be made that uploading 50MB file is not very realistic in terms of real time IoT data use case. However, this approach was chosen to be able to apply high processing load to the pipeline and is representing a high number of smaller files. Experiment results show that, with caveat that desired frequency was actually not achieved, this still was not enough to show noticeable performance differences between function plan 2 and 3.

When considering smaller scale IoT applications, Azure has limitations with premium function plan because of not scaling down to zero instances and starting with rather powerful plan setting with 1 core and 3.5GB working memory. This means using premium function will quickly start ramping up costs. AWS lambda seems to offer better options from a perspective of smaller applications with working memory options starting from 128MB per function[20]. Using Azure consumption plan is the better option for smaller applications but it lacks some functionality that could be needed like using custom containers. Also if C# is not being used as the programming language for the functions, some useful functionality is not accessible like running serverless functions on edge environment.

[20] `https://aws.amazon.com/lambda/pricing/`

8 Conclusion

The contribution of this work is in proposing a simple serverless data pipeline solution for real time IoT use case. Microsoft Azure cloud computing service was used to implement the proposed approach. Created pipeline was tested for performance, reliability and consistency with a simulated scenario using RPi 4B device acting as IoT devices. Different test file sizes, event frequencies and function plans were used during the tests to find out how these variables affect proposed pipeline's performance.

Results showed that function plans with more computing resources delivered better performance but current experiment was not able to generate enough processing load to see noticeable performance increase using the most resourceful function plan. Using function with less computing power proved to be more cost efficient. Azure premium function offers less fine tuning with smaller function memory options compared to AWS Lambda and is not able to scale to zero. In context of smaller applications the consumption plan is the only cost efficient option, but has less functionality and might not be usable in some situations like when deploying custom Docker containers. Also C# should be used to develop Azure functions to make all features usable.

End-to-end pipeline performance was not very consistent having multiple times difference in some situations. With the current experiment design the pipeline's performance was greatly impacted by test file upload times. Other factors like failing function executions and gradually increasing load during experiment also contributed to the inconsistency. During experiments there were numerous situations where function instances failed. In most cases, the retry policy was able to solve this issue and avoid pipeline failures. There were still a few pipeline failures so in use cases where this would not be acceptable, extra measures or design changes should be implemented to increase pipeline's resilience.

In future work proposed serverless pipeline could be improved with better error handling to have more accurate understanding of reasons behind errors and executions. Different pipeline design options could be explored as well to see how it affects performance. Experiment could be made more realistic using smaller test file sizes and upload bottleneck should be avoided to put more processing load on the pipeline in order to get more meaningful data. One solution could be to test the cloud based part of pipeline separately.

References

[1] Mohamed Abdel-Basset, Gunasekaran Manogaran, and Mai Mohamed. Internet of things (iot) and its impact on supply chain: A framework for building smart, secure and efficient systems. *Future generation computer systems*, 86(9):614–628, 2018.

[2] Abdulelah Almishal and Ahmed E Youssef. Cloud service providers: A comparative study. *International journal of computer applications & information technology*, 5(II), 2014.

[3] Mohammad S Aslanpour, Adel N Toosi, Claudio Cicconetti, Bahman Javadi, Peter Sbarski, Davide Taibi, Marcos Assuncao, Sukhpal Singh Gill, Raj Gaire, and Schahram Dustdar. Serverless edge computing: vision and challenges. In *Proceedings of the 2021 Australasian Computer Science Week Multiconference*, pages 1–10, 2021.

[4] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, et al. Serverless computing: Current trends and open problems. *Research advances in cloud computing*, pages 1–20, 2017.

[5] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16, 2012.

[6] Gilles Callebaut, Guus Leenders, Jarne Van Mulders, Geoffrey Ottoy, Lieven De Strycker, and Liesbet Van der Perre. The art of designing remote iot devices—technologies and strategies for a long battery life. *Sensors*, 21(3):913, 2021.

[7] Paul Castro, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. Serverless programming (function as a service). In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2658–2659. IEEE, 2017.

[8] Paul Castro, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. The rise of serverless computing. *Communications of the ACM*, 62(12):44–54, 2019.

[9] Nidhi Chawla. Ai, iot and wearable technology for smart healthcare-a review. *International Journal of Recent Research Aspects*, 7(1), 2020.

[10] Michele Compare, Piero Baraldi, and Enrico Zio. Challenges to iot-enabled predictive maintenance for industry 4.0. *IEEE Internet of Things Journal*, 7(5):4585–4597, 2019.

[11] Anirban Das, Shigeru Imai, Stacy Patterson, and Mike P Wittie. Performance optimization for edge-cloud serverless platforms via dynamic task placement. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 41–50. IEEE, 2020.

[12] Rustem Dautov and Salvatore Distefano. Stream processing on clustered edge devices. *IEEE Transactions on Cloud Computing*, 10(2):885–898, 2020.

[13] Chinmaya Dehury, Pelle Jakovits, Satish Narayana Srirama, Vasilis Tountopoulos, and Giorgos Giotis. Data pipeline architecture for serverless platform. In *Software Architecture: 14th European Conference, ECSA 2020 Tracks and Workshops, L'Aquila, Italy, September 14–18, 2020, Proceedings 14*, pages 241–246. Springer, 2020.

[14] Beniamino Di Martino, Massimiliano Rak, Massimo Ficco, Antonio Esposito, Salvatore Augusto Maisto, and Stefania Nacchia. Internet of things reference architectures, security and interoperability: A survey. *Internet of Things*, 1:99–112, 2018.

[15] Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud computing: issues and challenges. In *2010 24th IEEE international conference on advanced information networking and applications*, pages 27–33. Ieee, 2010.

[16] Stacy-Ann Elvy. Data privacy and the internet of things, February 2022. `https://en.unesco.org/inclusivepolicylab/analytics/data-privacy-and-internet-things`. Accsessed 10. Aug 2023.

[17] Muhammad Shoaib Farooq, Shamyla Riaz, Adnan Abid, Tariq Umer, and Yousaf Bin Zikria. Role of iot technology in agriculture: A systematic literature review. *Electronics*, 9(2):319, 2020.

[18] Keith D. Foote. A brief history of the internet of things, January 2022. `https://www.dataversity.net/brief-history-internet-things/`. Accsessed 9. Aug 2023.

[19] Eliza Gomes, Felipe Costa, Carlos De Rolt, Patricia Plentz, and Mario Dantas. A survey from real-time to near real-time applications in fog computing environments. In *Telecom*, volume 2, pages 489–517. MDPI, 2021.

[20] Wan Haslina Hassan et al. Current research on internet of things (iot) security: A survey. *Computer networks*, 148:283–294, 2019.

[21] Shawn Hershey, Sourish Chaudhuri, Daniel PW Ellis, Jort F Gemmeke, Aren Jansen, R Channing Moore, Manoj Plakal, Devin Platt, Rif A Saurous, Bryan

Seybold, et al. Cnn architectures for large-scale audio classification. In *2017 ieee international conference on acoustics, speech and signal processing (icassp)*, pages 131–135. IEEE, 2017.

[22] Linghe Kong, Jinlin Tan, Junqin Huang, Guihai Chen, Shuaitian Wang, Xi Jin, Peng Zeng, Muhammad Khan, and Sajal K Das. Edge-computing-driven internet of things: A survey. *ACM Computing Surveys*, 55(8):1–41, 2022.

[23] Hermann Kopetz and Wilfried Steiner. Real-time communication. In *Real-time systems: Design principles for distributed embedded applications*, pages 177–200. Springer, 2022.

[24] Shi-Wan Lin. The industrial internet reference architecture, February 2022. `https://www.iiconsortium.org/wp-content/uploads/sites/2/2022/11/IIRA-v1.10.pdf`. Accsessed 10. Aug 2023.

[25] Yuehua Liu, Wenjin Yu, Tharam Dillon, Wenny Rahayu, and Ming Li. Empowering iot predictive maintenance solutions with ai: A distributed system for manufacturing plant-wide monitoring. *IEEE Transactions on Industrial Informatics*, 18(2):1345–1354, 2021.

[26] Pankaj Mendki. Docker container based analytics at iot edge video analytics usecase. In *2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU)*, pages 1–4. IEEE, 2018.

[27] IoT Business News. Supply chain day: the ongoing chip shortage and its impact on the electronics industry, April 2023. `https://iotbusinessnews.com/2023/04/18/88788-supply-chain-day-the-ongoing-chip-shortage-and-its-impact-on-the-electronics-industry/`. Accsessed 10. Aug 2023.

[28] Dinh C Nguyen, Ming Ding, Pubudu N Pathirana, Aruna Seneviratne, Jun Li, Dusit Niyato, Octavia Dobre, and H Vincent Poor. 6g internet of things: A comprehensive survey. *IEEE Internet of Things Journal*, 9(1):359–383, 2021.

[29] Robert Pepper and John Garrity. The internet of everything: How the network unleashes the benefits of big data. *Global Information Technology Report 2014*, 2014.

[30] Antreas Pogiatzis and Georgios Samakovitis. An event-driven serverless etl pipeline on aws. *Applied Sciences*, 11(1):191, 2020.

[31] Shivananda Poojara, Chinmaya Kumar Dehury, Pelle Jakovits, and Satish Narayana Srirama. Serverless data pipelines for iot data analytics: A cloud vendors perspective and solutions. In *Predictive Analytics in Cloud, Fog, and Edge Computing:*

Perspectives and Practices of Blockchain, IoT, and 5G, pages 107–132. Springer, 2022.

[32] Shivananda R Poojara, Chinmaya Kumar Dehury, Pelle Jakovits, and Satish Narayana Srirama. Serverless data pipeline approaches for iot data in fog and cloud computing. *Future Generation Computer Systems*, 130:91–105, 2022.

[33] Asmaa H Rabie, Ahmed I Saleh, and Hesham A Ali. Smart electrical grids based on cloud, iot, and big data technologies: state of the art. *Journal of Ambient Intelligence and Humanized Computing*, 12:9449–9480, 2021.

[34] Sayed Khushal Shah, Zeenat Tariq, and Yugyung Lee. Audio iot analytics for home automation safety. In *2018 IEEE international conference on big data (big data)*, pages 5181–5186. IEEE, 2018.

[35] Changyang She, Yifan Duan, Guodong Zhao, Tony QS Quek, Yonghui Li, and Branka Vucetic. Cross-layer design for mission-critical iot in mobile edge computing systems. *IEEE Internet of Things Journal*, 6(6):9360–9374, 2019.

[36] Satyajit Sinha. State of iot 2023: Number of connected iot devices growing to 16.7 billion globally, May 2023. `https://iot-analytics.com/number-connected-iot-devices/`. Accsessed 9. Aug 2023.

[37] Aditya Tiwary, Manish Mahato, Abhitesh Chidar, Mayank Kumar Chandrol, Mayank Shrivastava, and Mohit Tripathi. Internet of things (iot): Research, architectures and applications. *International Journal on Future Revolution in Computer Science & Communication Engineering*, 4(3):23–27, 2018.

[38] Ralf Tönjes, P Barnaghi, M Ali, A Mileo, M Hauswirth, F Ganz, S Ganea, B Kjærgaard, D Kuemper, Septimiu Nechifor, et al. Real time iot stream processing and large-scale data analytics for smart city applications. In *poster session, European Conference on Networks and Communications*, page 10. sn, 2014.

[39] Athina Tsanousa, Evangelos Bektsis, Constantine Kyriakopoulos, Ana Gómez González, Urko Leturiondo, Ilias Gialampoukidis, Anastasios Karakostas, Stefanos Vrochidis, and Ioannis Kompatsiaris. A review of multisensor data fusion solutions in smart manufacturing: Systems and trends. *Sensors*, 22(5):1734, 2022.

[40] Keiichi Yasumoto, Hirozumi Yamaguchi, and Hiroshi Shigeno. Survey of real-time processing technologies of iot data streams. *Journal of Information Processing*, 24(2):195–202, 2016.

[41] Junho Yeo, Jong-Ig Lee, and Younghwan Kwon. Humidity-sensing chipless rfid tag with enhanced sensitivity using an interdigital capacitor structure. *Sensors*, 21(19):6550, 2021.

[42] Dong Yu and Lin Deng. *Automatic speech recognition*, volume 1. Springer, 2016.

[43] Jiale Zhang, Bing Chen, Yanchao Zhao, Xiang Cheng, and Feng Hu. Data security and privacy-preserving in edge computing paradigm: Survey and open issues. *IEEE access*, 6:18209–18237, 2018.

Appendix

I. Glossary

- 5G/6G - 5th and 6th generation mobile networks
- ARM - Advanced Risc Machine
- AWS - Amazon Web Services
- ETL - Extract Transform Load
- FaaS - Function as a Service
- IoT - Interent of Things
- IPv6 - Internet Protocol Version 6
- M2M - Machine-to-machine
- MQTT - Message Queuing Telemetry Transport
- OS - Operating System
- RFID - Radio Frequency Identification
- RPi 4B - Raspberry Pi 4 Model B
- SF - Serverless Function
- vCPU - Virtual Central Processing Unit

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Martin Kisand**,
*(*author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Serverless Data Pipelines for IoT Data in Edge and Cloud Environments using Microsoft Azure,
   *(*title of thesis)

   supervised by Shivananda. R. Poojara.
   *(*supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Martin Kisand
11/08/2023