Empirical Study Of Smartphones As An Edge Device

by

SHAH VYOM HITESHKUMAR
202111012

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of

MASTER OF TECHNOLOGY
in
INFORMATION AND COMMUNICATION TECHNOLOGY
to

DHIRUBHAI AMBANI INSTITUTE OF INFORMATION AND COMMUNICATION TECHNOLOGY

July, 2023

Declaration

I hereby declare that

i) the thesis comprises of my original work towards the degree of Master of Technology in Information and Communication Technology at Dhirubhai Ambani Institute of Information and Communication Technology and has not been submitted elsewhere for a degree,

ii) due acknowledgment has been made in the text to all the reference material used.

Vyom

Vyom Shah

Certificate

This is to certify that the thesis work entitled Empirical Study Of Smartphones As An Edge Device has been carried out by VYOM SHAH for the degree of Master of Technology in Information and Communication Technology at *Dhirubhai Ambani Institute of Information and Communication Technology* under my/our supervision.

[illegible]

Prof. Kalyan Sasidhar
Thesis Supervisor

Acknowledgments

First and foremost, I would like to express my gratitude to my research supervisor and mentor, Prof PS. Kalyan Sasidhar, for his invaluable guidance and dedicated involvement in every aspect of my work. His detailed explanations and humble attitude profoundly impacted me, and his focused vision steered me in the right direction. I am thankful for his instruction on research methodology and for presenting research work on various platforms. Additionally, I am grateful for the workspace he provided me and for making me a part of the 'UbiSense Lab' in lab-003. I also appreciate the workstation he gave me to complete my research.

I deeply thank Prof Bhaskar Chaudhary for his unwavering support, encouragement, and insightful feedback. His constructive criticism has challenged me to think critically and creatively, enabling me to reach my full potential. His expertise, dedication, and commitment have shaped my intellectual growth and success.

I would also like to thank my lab colleague, Jaineel Trivedi, for assisting me throughout the process and providing me with theoretical knowledge about the project. I would be remiss not to mention PhD scholars Pramod Tripathi and Maitri Vaghela for providing me the guidance for the research.

I am thankful to my friends and the entire MTech-2021 class for helping me throughout the two-year road to finish my MTech program and Thesis, which required more than just academic assistance. I want to express my gratitude to my friends Vinay Sheth, Harsh Savaliya, Dhara Mehta, Kavan Hajare, Rohit Mishra, and many more for supporting me during the entire journey.

Lastly, I would like to thank my parents and family for their constant belief in me and their unwavering support throughout the process.

Contents

Abstract

Increased automation and intelligence in computer systems have revealed Cloud-based computing constraints such as unpredictable latency in safety-critical and performance-sensitive applications. Features of smartphones attract researchers more towards using smartphones as edge computing devices because of the presence of the sensors inbuilt and the computing powers of CPU cores. So a smartphone is a combination of IoT and Edge computing devices.

To overcome the usage of high-end computing devices at the edge layer, this article proposes the idea of using a smartphone as an edge device for processing data. Sensors or IoT devices generally send data to the edge device rather than directly sending it to the cloud for processing. So mainly, this article emphasizes solving the research question of whether smartphones can be used as edge devices. So in this, a distributed smartphone system following master-slave architecture is proposed, which helps to distribute the computation power among slaves. Word count, average of temperature data and indoor localization. Compared to desktop PCs computation, master-slave utilizes CPU 75% more than just on single-device computation and on an average 50% faster than on computing on a single device. This motivates us to design an architecture that can utilize the data from the cloud and perform the computation using the CPU cores of the smartphone.

List of Principal Symbols and Acronyms

α	temperature data as key
$\beta(\alpha)$	count of α found by thread
δ	A single window
ω	Total windows
ρ_i	final count of key α_i aggregated by master
ζ_i	count of α_i aggregated by slave
CWC	Computing
DTW	Dynamic Time Warping
HPC	High-Performance Computing
master_DTW	minimum DTW distance found at master
MR	Map Reduce
MS	Master-Slave
slave_DTW	minimum DTW distance found at slave
U	Total number of keys

List of Tables

List of Figures

CHAPTER 1
Introduction

1.1 Organization of Thesis

A flow diagram for the organization of the thesis is shown in Figure 1.1.

Chapter 1 (Introduction): This chapter briefly describes edge computation, the evolution of smartphones, and issues with the server as an edge device and then I have provided the motivation for this thesis and problem statement.

Chapter 2 (Rise of smartphones): This chapter discusses about the evolution of smartphones, that how both hardware and software of the device has been improved in last few decades.

Chapter 3 (Literature Survey): This chapter discusses the existing work where smartphones are used as an edge device. And also some of the issues and challenges regarding the edge and cloud computing paradigm, which can be resolved with the use of smartphones as edge devices for specific workload/tasks.

Chapter 4 (Proposed Architecture): This chapter talks about the propsed architecture based on master slave using smarphones and idea on how map-reduce can be implemented on mobiles using this architecture

Chapter 5 (Methodology): This chapter will talk about the actual system architecture of master-slave developed for our application and a detailed description of the working of two applications on this architecture.

Chapter 6 (Indoor Localization):Brief description of how to find the location of person based on magnetic field using master slave architecture is been discussed in this chapter.

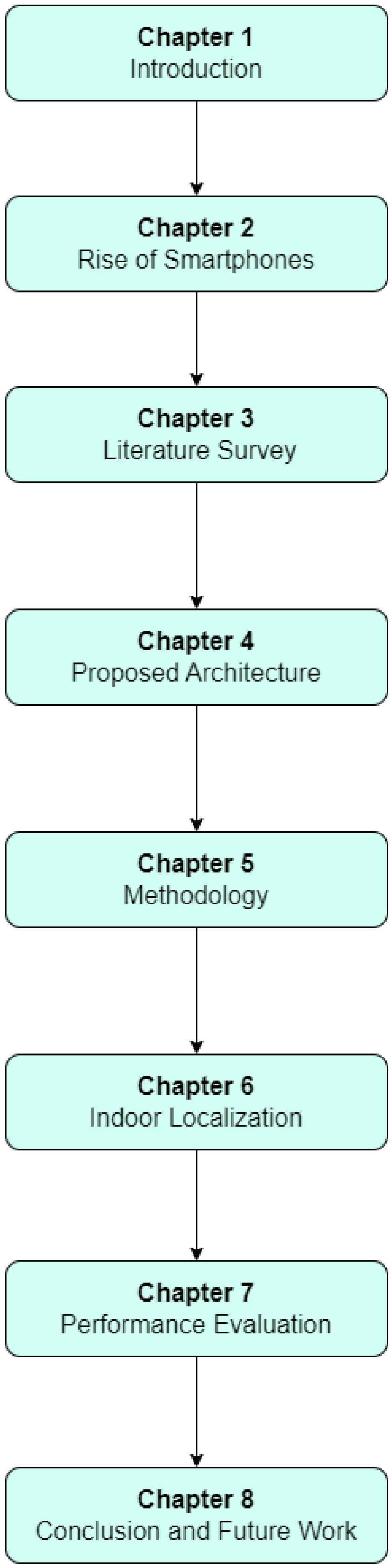


Figure 1.1: Organization of Thesis

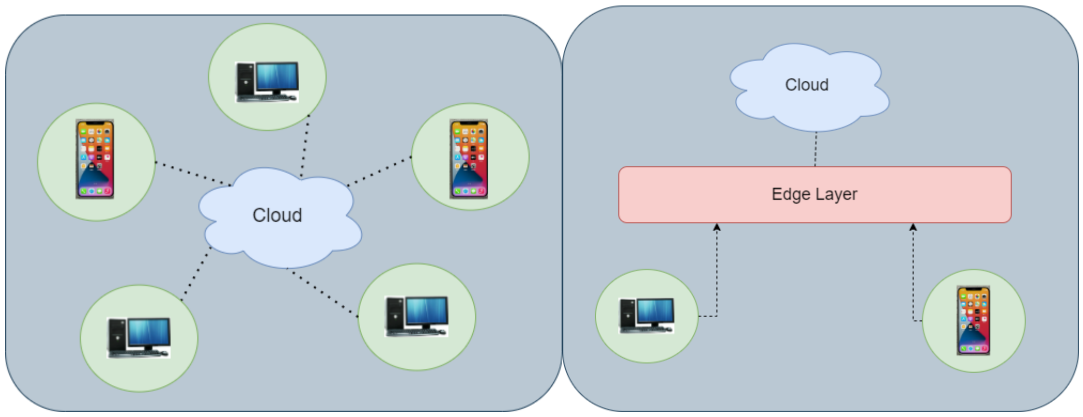
Chapter 7 (Performance Evaluation): This chapter will give an overview of the behavior of our algorithm based on the three metrics CPU utilization, time, and scalability.

Chapter 8 (Conclusion and Future Work): This chapter concludes my work and discusses the future possibilities of work that could be done to improve the system.

1.2 Overview of Edge Computing

With the rapid development of the Internet of Things (IoT), the number of intelligent devices connected to the Internet is increasing. Cisco claims that there will be 29.3 billion devices by 2023, resulting in large-scale data [20]. Traditional cloud computing models incur bandwidth load, slow response speed, poor security, and poor privacy. To reduce the latency, edge and fog computing solutions have arisen. Edge computing facilitates the execution of computations at the network's edge - proximity to the user and proximity to the source of the data. It is lightweight at the network's edge for local, small-scale data storage and processing.

Edge computing [18] extends computational, network connectivity, and storage capabilities from the cloud to the network's edge. It permits business logic between the upstream data of the Internet of Things and the downstream data of the cloud service (IoT). Edge computing offers further advantages of agility, real-time processing, and autonomy in the area of Industrial IoT to provide value for intelligent manufacturing. In most IoT applications, sensor data is gathered from diverse sensors on a server for data analysis. For instance, the analysis could be as simple as calculating a region's average temperature or air quality. However, the computation may require good computing power considering smart city applications with high volume/variety of data from wearable sensors, deployed sensors, cameras, etc.



(a) (b)

Figure 1.2: (a) Cloud (b) Edge Layer

Fig. 1.2(a) shows the simple architecture of Cloud Computing where the end devices are directly connected to the cloud. The raw data will be directly sent to the cloud. Fig. 1.2(b) shows the simple architecture of edge computing, where there is an extra edge layer between the end devices and the cloud, where the raw data will be stored locally, and some computations can be done. Then the refined data will be sent to the cloud.

In edge computing, an edge serves as a gateway between the cloud and the data-generating devices. Any edge device, from smartphones to industrial equipment, can produce data that needs analysis. As a mediator, the edge server gathers data from the devices and transmits it to the cloud for additional processing. This makes it possible to use less bandwidth and respond more quickly. The use of edge computing has the potential to change how we interact with the Internet completely. Edge computing enables programs to be more responsive and effective by moving computation and data storage closer to the point of use.

Edge computing can drastically reduce latency by moving computation and storage closer to the data source. This is especially crucial for real-time processing applications like self-driving cars and health applications. As most of the data can be processed at the edge, i.e., near the sensors, the bandwidth efficiency can also be improved. Another main advantage of using an edge server is that data will not be transferred to the cloud over the Internet. Instead, the raw data will be stored and processed locally near the sensors at an edge server. The use of edge computing has the potential to change how we interact with the internet completely. Edge computing can increase applications' responsiveness, efficiency, and security by moving computation and data storage closer to the point of use.

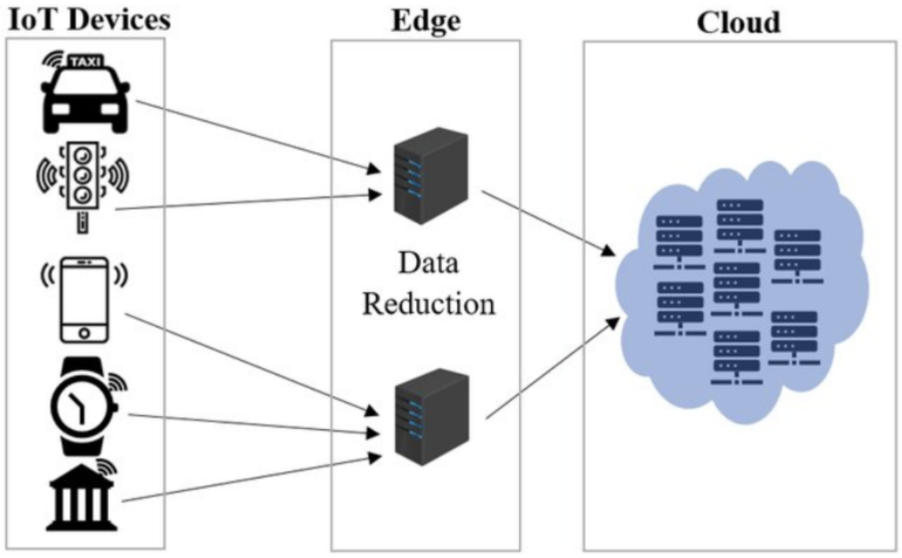


Figure 1.3: Edge Computing

Researchers are more focused on creating a local server environment where vital data can be stored or processed before sending it to the centralized data center or cloud storage repository. The below figure illustrates the paradigm of edge computing, creating a server or network of servers which can act as a middleman between IOT devices and the cloud.

1.3 Evolution of Smartphones

Nowadays, smartphones have quickly supplanted feature phones as the most common design for mobile devices. By the end of 2023, smartphones are expected to make up 90 percent of mobile devices used worldwide, according to [55], the rapid technological development of mobile handsets, from communication devices with fixed features to general-purpose devices with increased computational power and network connectivity, is reflected in the popularity of smartphones.

We will discuss about the rise of smartphones in the last decade in detail in the next chapter.

1.4 State of the art of Computational power (Server)

Advancements in the computational power of servers have improved in the last decades[43]. Servers comprise high computing power, where they can handle more than one computation request of the clients/users. These servers are used in client-server, i.e., the client will request the specific task, and the server will do the job and revert it back to the client. Using this architecture, even if the client has no high computational setup, it can also see the specific task's results or output. They are equipped with advanced networking capabilities, which give them the power to handle a significant amount of data traffic and give smooth communication between the client and server. Servers also play an essential role in tasks that needs high computational power. They are made to efficiently compute complex functions like machine learning, data analytics, etc. This architecture is efficient for cloud computing systems [45]. Traditionally, a conventional PC had been used to set up a small server that can facilitate the requests of the clients.

1.5 Issues and Challenges in Server

Although servers are undoubtedly the most powerful computers available today, there are a few drawbacks to be aware of. Firstly they can be costly to purchase and maintain and also require high technical expertise for smooth and efficient operations. A typical AWS server costs you around $52.57 for a month[51, 7], even if you are doing high computational tasks. As the server act as a central point of the architecture, failure in the server can disrupt the entire system; yes, there are solutions for this, like duplicating the data at different nodes/servers. Still, this may not be feasible or cost-effective for a small server setup. Delay in sending and receiving messages is also a significant disadvantage due to many reasons like network error, excessive traffic on the server, etc. To overcome these, we can use the edge layer architecture that instead of entirely relying on the cloud servers. Running a server also requires high energy input, i.e., it consumes more power. For example, let us take a PC acting as a server with an Intel Core i3, which will consume approximately 50*(1+2.5)*6 = 1050 WH(Watt-Hours) if we keep it running for 6 hours. Here 2.5 indicates that every Watt consumed requires 2.5 Watts for cooling and power distribution [1]. Also, due to the extensive use of servers which leads to more power consumption, it also emits more carbon dioxide to the environment, which leaves a carbon footprint[53]. The authors of this paper did an empirical study of using smartphones as a server/computing device for specific tasks and proposed an architecture for the same.

1.6 Phone as a Computing Platform (PaaP)

As discussed in the above chapters, the recent advancements in hardware and software of smartphones, they can also be used as a computational device. From the fig 2.1 we can see that the performance of the SoC or the processor clock speed has increased significantly compared with the conventional PCs.

Former studies show that we can use a smartphone as a computing platform [53, 12, 48], where the researchers have shown with different empirical studies that smartphones are capable of complex computations too. Edge computing applications like health care, smart city, and smart farming in these different applications we can use smartphones as local computational devices.

1.7 Why Smartphones are advantageous ?

Smartphones are the most used computing device in the world. According to [3], India will have around 1 billion smartphones by 2026. Also, as discussed in the above sections, the computing power of smartphones has increased significantly, and it is still improving. However, the irony is that such high computing power is under-utilized. Apart from gaming applications on phones, not many apps utilize CPU power. For instance, according to[38], we found that the most commonly used applications were Whatsapp, Instagram, and Snapchat, which utilized only a fraction of the CPU despite being used for an entire day. Gaming apps were observed to be used for a mere 2-3 hours/per week. We can use smartphones' computing power to utilize it for complex specific tasks efficiently.

1.8 Motivation

Many applications such as smart farming, data analysis, smart home, and smart city use IoT sensors and edge computing devices for computation as a dedicated local server. Although not too much computation is required, high-end computing devices are kept as edge devices. Nowadays, people tend to change mobile phones every 2-3 years even though older ones are in working condition. Basically, older ones will create e-waste if not kept in use. To bring the computation power from idle to some extent, we can aim to use the mobile phone as an edge device, replacing the conventional server at an edge layer. Multiple phones can provide even high computation power, so one of the motives is to propose an architecture that can effectively utilize the computation power. This can optimize resource utilization as well as can be cost-effective, which can eliminate the use of high-end computing devices.

1.9 Problem Statement

This thesis revolves around using the capabilities of multiple smartphones as edge devices to utilize their computing power. Computation is distributed using the master-slave paradigm, and along with multithreading tasks can be executed in parallel. The master, as a central device divide task among slaves, and these slaves perform computations using multithreading and then sends results back to the master. Inbuilt sensors of mobile phone can help person to locate its location in the indoor environment using the master-slave architecture of mobile phones.

1.10 Chapter Summary

In this chapter, we have discussed about edge computing, the capabilities of smartphones, can we use smartphones as a computing platform, what are the issues and challenges, and finally the motivation of the thesis followed by the problem statement.

CHAPTER 2
The Rise of Smartphones

Since their initial release, smartphones have undergone remarkable advancements in crucial hardware components, leading to rapid expansion in application capabilities. These improvements have also had significant implications for using a smartphone as a computational device, with the integration of Neural Processing Units (NPUs) in the latest smartphone models. These NPUs enable the implementation of machine learning techniques, such as deep learning, into standard phone functionalities. This integration has resulted in various enhancements, ranging from better computation power to efficient battery usage, which leads researchers to think that a smartphone can be used as a computing device. Along with this, smartphone manufacturers have also started leveraging dedicated deep-learning hardware to improve the overall utilization of smartphones. However, researchers have started working on using the capabilities of smartphones to revolutionize personalized health applications. In this area the smartphones are employed in many diverse diagnostic applications, including human activity recognition[46], human sleep analysis [38], stress analysis[58], and also heart attack risk prediction [42]. This chapter will discuss the main components that facilitate these technological advancements, which include smartphone processors/ System of Chip(SoC), Random-access Memory(RAM), and battery capacity.

2.1 Are smartphones nearing the processing speed capabilities of PCs?

In the past decade, there were seen improvements in the processing speed of smartphones, and think tanks were saying that smartphones may have a significant processing speed as compared to PCs. Comparing the processing power of both these computational devices was not simple.

To compare the processing speed of the smartphones with conventional Pcs i.e. cross platform, cross architecture, it is necessary to test it with some standard benchmark tests. And they suggests the performance gap between smartphones and PCs is decreasing fast and much closer than thinked before. Geekbank5 [24] is a mainstream benchmark framework which was created by Primate Labs and it provides the comparison testing of device and its's CPU performance. It is a cross platform and cross-architecture, which makes it possible to compare the performances of different types of devices. For an instance lets discuss about how it compare the performance, Geekbench 5 uses Intel Core i3 as the base of different devices comparison and its test performance scored as 1000. In September 2021, phone manufacturer Apple releases the latest version of its A-series Bionic SoCs which they called as "A-15 Bionic" [11]. The preliminary Geekbench 5 results suggest that the processing speed of A15 approaches even more than that of the Intel's latest mobile CPUs.

The processing speed of smartphones is indeed approaching that of the most recent generation of PCs, as seen by the latest smartphones[9, 33], whether they run the iOS or Android operating systems. This is advantageous for using the smartphone for on-device processing and analytics rather than doing it elsewhere, in addition to using it for data collection.

2.2 Improvements in the hardware of smartphones(SoC)

In the past decade, there were seen improvements in the main smartphone's hardware i.e., SoC of the phone. An SoC is the main heart of the smartphone, where different chips are integrated. SoC typically has 3 major components, CPU/processor cores for computation, RAM for the local temporary storage of the computational data as well as dedicated input/output(I/O) interfaces for data accessibility by the other components [19].

Application-specific integrated circuits (ASICs) for accelerated video-encoding (camera record) and decoding (video playback) applications are now included in more modern SoCs. Additionally, SoCs provide wireless communication through Bluetooth, Wi-Fi, and compatibility for older cellular networks, whereas future networks, such as fifth-generation "5G," are frequently initially provided on separate hardware[41]. Since the SoC in every smartphone device is the main component of the current application code, researchers are continuously improving and enhancing the three crucial aspects of SoC design that have brought performance levels up to par with those of laptop computers today.

2.3 Improvements in Processor Clock Speed

Clock speed, the number of processing cores, and the core microarchitecture are some of these aspects. [52]. The clock speed of a processor is the rate at which the CPU generates clock pulses to control all the functions including arithmetic, logic and data transfer for fetching and storing from RAM and external storage [50]. The advancements in the clock speed of the processors is showed in the fig 2.1, where the the processing speed has been increased significantly.

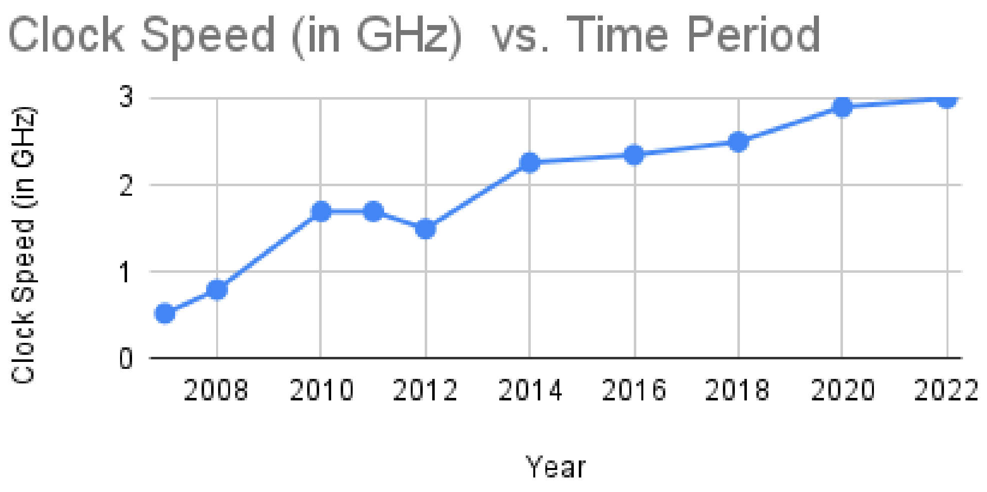


Figure 2.1: Evolution of Clock speed of mobiles

A CPU can execute software code more quickly and a mobile device appears to operate more quickly the higher the clock speed. In terms of smartphone SoCs, CPU clock speed is expressed in "Hertz".

2.3.1 Need of more number of cores

The more the number of cores the better computation power you can have. In the last decade the number of cores a processor of a smartphone has, has increased to 8, and research on multicore processors is still going on. A modern processor core consists of many millions of transistors.And electric signals are lept switching this transistors. The transistors' maximum switching frequency determines the maximum clock speed of a processor [5]. Hence, there is a limit to the clock speed of any processor.

To overcome this limitation of maximum clock speed, the solution was to increase the number of processor cores. A processor with more than one processor is enable to split a coded application to split the time consuming comples code into various threads. Each thread can be processed on a separate core, thus making the code parallely computing, will decrease the overall code execution time [57].

2.4 Improvements in Random-access Memory(RAM)

In smartphones, RAM is a high-speed volatile memory storage for temporarily storing the data related to the on-going processes or computations. More the RAM, larger and data intensive application can be executed on the phone.

The amount of RAM in smartphones has significantly increased over the last ten years. Devices quickly advanced from 256MB to 512MB of RAM, with limited memory capacity, reaching 1GB of RAM around 2010. AdvancLater, RAM capacities of 2GB and 3GB were the norm, enhancing multitasking efficiency and performance. 4GB of RAM became the industry standard by 2016–2017, greatly boosting smartphone performance. The RAM capacities of smartphones have increased in recent years, reaching 6GB, 8GB, or even 12GB [10]. The capacity to multitask and manage resource-intensive tasks has significantly increased thanks to these developments and software optimisations.

2.5 Chapter Summary

In this chapter, we have discussed the rise of smartphones in all the major aspects. The idea of this chapter was taken from [58], where the authors have explained the rise of smartphones.

CHAPTER 3

Literature Survey

Several works have already been done on using the power of smartphones efficiently. As we have discussed earlier about the capabilities of smartphones and how efficiently we can use the smartphone for different purposes.

Authors in [49] explore the potential of using smartphones as alternative computing engines for cloud-based applications. Traditionally, cloud computing relies on centralized data centers to process and store data, requiring significant infrastructure investments and energy consumption. However, the rapid advancements in smartphone capabilities, such as processing power, storage capacity, and network connectivity, have opened up new possibilities for leveraging these devices as distributed computing resources. The authors highlight several benefits of using smartphones as alternative cloud computing engines. First, smartphones are ubiquitous, with billions of devices in use worldwide. This widespread adoption allows for a vast network of distributed computing resources, potentially leading to enhanced scalability and improved performance for cloud-based applications. Moreover, smartphones are equipped with various sensors, such as cameras, accelerometers, and GPS, which can enrich the capabilities of cloud applications by providing contextual information and enabling new types of services.

Another advantage of utilizing smartphones for cloud computing is their mobility. Unlike traditional data centers, smartphones are mobile devices that can move closer to the end-users or specific locations where computing resources are needed. This proximity can reduce latency, improve response times, and enable edge computing capabilities, enhancing the overall user experience. In conclusion, the paper highlights the potential benefits of leveraging smartphones as alternative cloud computing engines, including scalability, performance improvements, mobility, and potential environmental advantages.

The concept of Smart farming [14] is related to real-time data gathering, processing, and analysis, as well as automating the technologies used in farming. Moreover, it can also help the farmers in decision-making regarding farming procedures. Here authors are using the smartphones just for collection of data and sending the data to a local or cloud server.A similar kind of study has been done by the authors in[31], where they discuss using a smartphone or tablet, a herd management system that enables cow ranchers to manage their dairy or beef herds.. MooMonitor+ [21] uses wearable collars to track the health and fertility of each individual cow, allowing farmers to keep an eye on their entire herd from their phones. Map your meal [2]is a smartphone app focused on agriculture that explores the world's food system with the goal of raising public awareness of global interconnectedness. Farmers and customers may use this smartphone app to scan products to determine how fair and environmentally friendly they are. Finally, FoodLoop[37] provides real-time deals on "nearly expired" products and connects the grocer inventory system and smart tagging to consumer-facing mobile applications to help the reduction of food waste. In this [35], the authors propose developing a system optimally watering agricultural crops based on a wireless sensor network. The system consists of three components: hardware, web application, and mobile application. Soil moisture sensors are used to monitor the field, connecting to the control box. The web-based application manipulates the details of crop data and field information. It applies data mining to analyze the data for predicting suitable temperature, humidity, and soil moisture for optimal future management of crop growth. The mobile application is mainly used to control crop watering through a smartphone.

Authors in [6, 15]contend that a number of crucial IoT services and applications should be run on the Cloud Computing platform. They contend that Fog Computing supports a new breed of apps and services rather than focusing on Cloud Computing. The article is divided into sections that outline and introduce the Fog Computing paradigm's properties. They then closely examine a few noteworthy apps and services that support their claim that the Fog is an essential part of the platform needed to support the Internet of Things. Another interesting research study and implementation of smart city[36], where the authors have discussed about conceputlizing a smart city where the devices are connnected. Authors in [23] have discussed the emerging field of fog computing and how it can be helpful by creating a whole layer of the computational device, where all the devices come of a particular city comes under an umbrella and can be used as an edge layer. The authors have estimated the computation power that can be generated by applying this architecture by making an edge layer of all computational devices of the city of Barcelona. And they have claimed that it can be 10 to 20 times more potent than an Amazon Web Service(AWS) cloud server.

In recent times researchers have been trying to use the power of smartphones which are remained idle or are not being utilized, i.e., instead of keeping the phone idle, efficiently utilizing the power of smartphones. Authors in [12] discuss the potential of using idle smartphones being charged overnight as distributed computing infrastructure for an enterprise that supplies its employees with smartphones and utilizes smartphones efficiently and as a cost-effective alternative to running specific tasks on traditional servers. They address the challenges of using smartphones for computing and develop CWC, a distributed computing infrastructure using smartphones. They implement and evaluate a prototype of CWC that employs a novel scheduling algorithm to minimize the makespan of a set of computing tasks. The evaluations show that CWC's scheduler yields a makespan that is 1.6x faster than other, more straightforward approaches. Junk-Yard Computing[54, 16] is a new paradigm, and here, authors are discussing utilizing smartphones by creating a cluster in a client-server model and making a smartphone cluster that can do specific tasks more efficiently than a traditional server. It introduces the novel concept of "junkyard computing," which repurposes discarded smartphones as a computational resource. They demonstrate that even decade-old smartphones can perform modern cloud microservices and explore how to combine them to execute complex tasks. They have proposed a cloudlet comprising reused Pixel 3A phones and assessed the carbon benefits of deploying large, end-to-end microservice-based applications on these smartphones. OpenRTiST[25] is also a similar work, where the authors have created a benchmark by running specific tasks across traditional servers, mobile-servers and doing the same task on smartphones. They also discuss how OpenRTiST can be used as a workload generator in controlled benchmarks to measure motion-to-photon latency, estimate scalability, and identify performance bottlenecks.

The past decade has seen an exponential rise in smartphone production and the advancement of hardware specifications. Smartphone CPUs, especially, are on par with desktop CPUs [44]. In the last five years, there have been smartphones with quad-core and octa-core processors. However, these multi-core processors may not be utilized to their full potential barring a few intensive mobile gaming or video streaming applications. For instance, in our recent work [56], we performed an experimental study through a custom-developed app [39] to quantify smartphone usage. We found that the average phone usage was 7.5 hours/per day. All phones were running with a quad-core CPU. The most commonly used applications were Whatsapp, Instagram, and Snapchat, which do not burden the CPU, even running for an entire day. Gaming apps were observed to be used for a mere 2-3 hours/per week. This study shows that the phones' CPU power is left underutilized. Garcia et al., [23] estimate the computation power of personal computers, smartphones, on-board car computers, WiFi routers, and other embedded computers across a city to be 19 $*10^{15}$ instructions per second. They propose the use of such computing power in place of high-end servers. The earliest attempts were made by authors in [17], who set up a smartphone cluster to execute an HPC application and found the performance nearly close to a high-end PC. Further, work in [12] proposed a distributed framework of smartphones and developed a scheduling algorithm to schedule and execute three compute-intensive tasks on smartphones. Recently, a similar study [47] evaluated the performance of Moto x4 and x5 in terms of their computation time and energy consumption while running matrix multiplication and other algorithms.

3.1 Chapter Summary

Summarizing this chapter, we learned about various methods of using the edge computing paradigm in an efficient way. We discussed the capabilities of smartphones and some real-world applications where smartphones can be used as an edge devices in a client-server model. I've discussed about some of the well-known works related to smartphone computing and edge computing. In the next chapter we will discuss about our proposed system architecture and how map-reduce can be applied on it.

CHAPTER 4
Proposed Architecture

4.1 Master Slave Paradigm

In [4] we discovered that mobile phone is capable to perform computation with the multithreading concept. We came up with an extension of the idea to work with the master-slave (MS) architecture, which includes the distributed computing concept and the multithreading concept related to parallelism. So for this architecture, multiple phones will be used, and we will study whether we can bring the CPU from idle to active mode. As we have seen in previous chapter 3, mobile phones are capable of computation power also.

4.1.1 Our System Architecture

Fig 4.1 shows the three-tier master-slave architecture.

- **First Layer:-** Master handles coordinating as well as dividing tasks among the slaves and also making decisions for the final answer. Master is the main point of contact where all data are sent, which is to be computed and also to get the final answer.

- **Second Layer:-** As a second layer slaves come into the picture as communicators for transferring data and handling the parallelism of the task given by the master to it. Slaves are like mediators and are responsible for ensuring that given task is performed parallel by creating threads.

- **Third Layer:-** Thread is the actual task performer which does the computation on the data, which is divided by slave, and then sends the results back to the slave. Each thread executes a specific portion of the assigned task, utilizing the available resources efficiently.

Our basic idea is to implement the map-reduce application on the master slave architecture shown in 4.1 using smartphones to utilize smartphone capabilities.

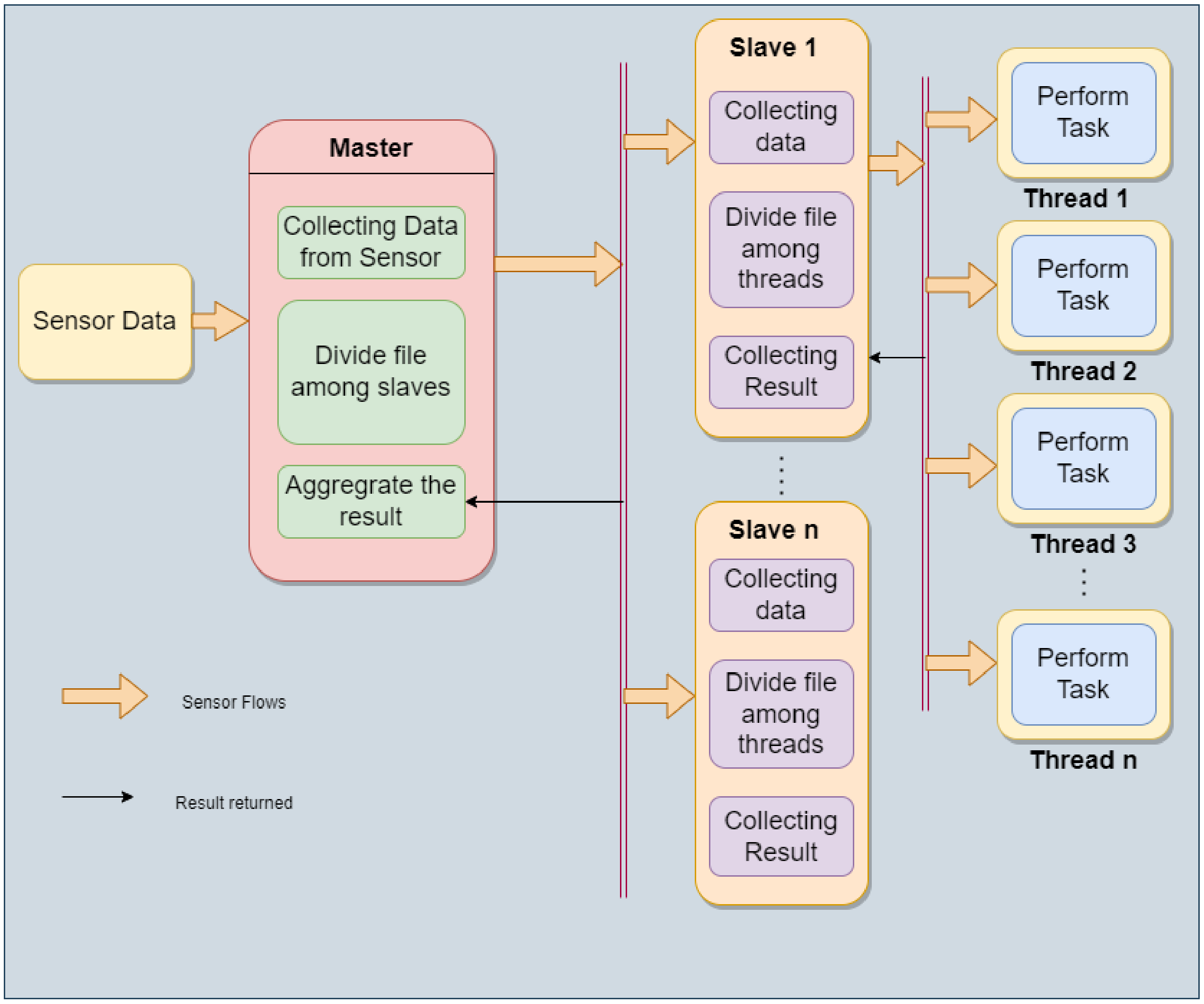


Figure 4.1: Master-Slave Architecture

We have created an android app that works on master-slave architecture and has been designed so that master and slave applications can be run individually on mobile phones. The application implements the program of finding the average of temperarture data program using a concept similar to the map-reduce.

- **Collecting Data File**

  Data collectively stored in a CSV file format will be sent to the master and expect the output from the master as per the experiment performed on the master slave architecture.

  For example, a CSV file size of 900MB of data passed to the master mobile phone.

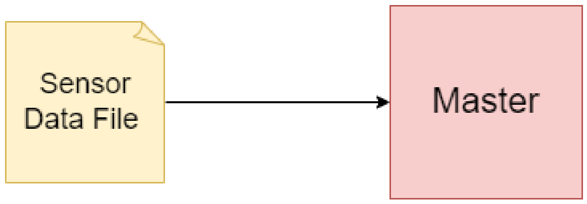


Figure 4.2: Sending File to master

- **Dividing File among Slaves**
  After receiving the file, the master will start the server so that slaves in the same network, whoever wants to connect to the master, will register themselves with the master by sending their number of cores. The main reason to divide the file according to cores is that the computation can be divided equally with the ratio of the cores due to which one with less number of cores will get file of smaller size and one with more cores will get file of large size. Once the master finds enough slaves connected, it will divide the file among the slaves according to the number of cores using the wireless network with the help of TCP protocol. The reason for using TCP protocol is to cover the reliability issue. If Slave1 and Slave2 are registered with the

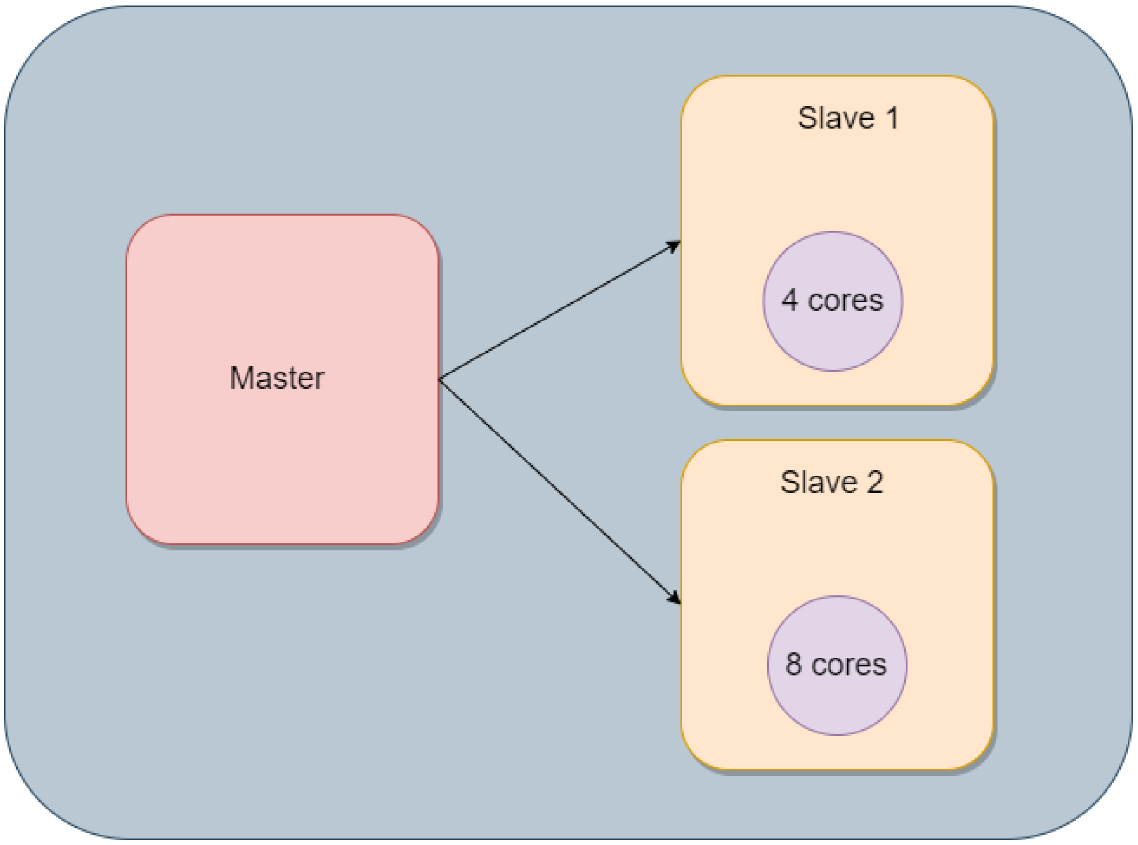


Figure 4.3: Dividng File among Slaves

master with 8 and 4 cores, respectively, then the master will divide the file in a ratio of 2:1. So Slave 1 will get a file with a size of 600MB, and the other one will get a file of size 300MB.

- **Splitting into file chunks**

  Once the slave receives the file, it will divide the received file according to its cores. The slave will now create the threads which will work on the file chunk size. Basically, the thread will linearly traverse over the file chunk and find the count of every unique temperature data. At the end thread will return key-value pair in which temperature data as the key and count of that data as the value.

Keeping Thread constant

In this case, we have kept the number of threads constant. The number of threads is kept equal to the number of cores because each core can easily handle each thread.

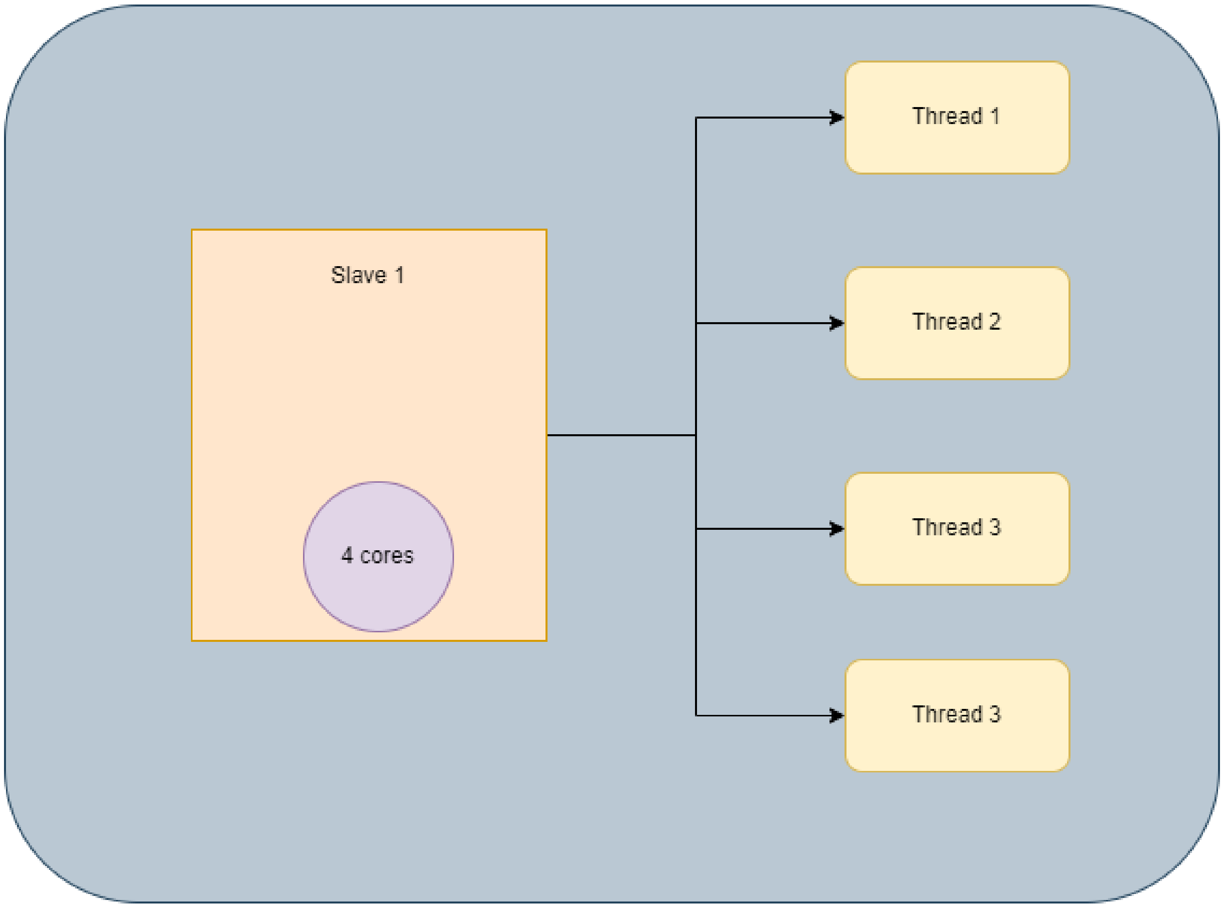


Figure 4.4: Thread Count Constant

For example, if a mobile phone is quad-core, then four threads will be created by the program itself. So here in this scenario, the file chunk size will also increase with the increase in the main file size.

Keeping File Chunk Size constant

In this case, we have kept the size of the file chunk constant.

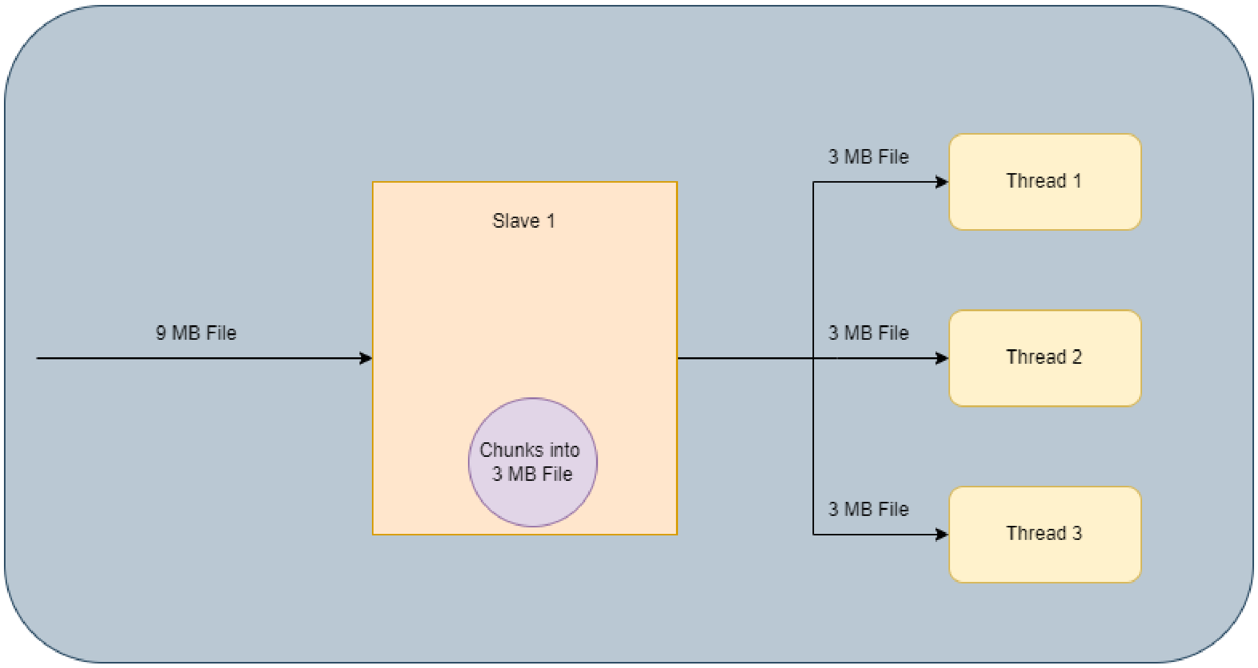


Figure 4.5: File Chunk Size Constant

As shown in Fig 4.5, a 9MB file is passed to the slave from the master, and the slave has set up a threshold of 3MB file only, so now the 9MB file will be split into three files chunk each of 3MB size. In this case, the number of threads will not remain constant because the number of threads will be the same as the number of chunks created after splitting the main file. File chunk size can be adjusted according to our requirements. So the number of threads should also increase with the increase in primary file size.

4.2 Map Reduce

Map Reduce is one of the application that can be executed based on the master slave manner. It is also the programming model which is used for processing of big datasets in a parallel manner and can be executed on the clusters with distributed algorithms. Map Reduce contains main two functions 1) Map and 2) Reduce.

- Map: The input data is split into smaller sections, and a map function is individually applied to each section. The map function takes the input data and generates intermediate key-value pairs by performing operations like transformation or extraction. This process produces intermediate results.
- Reduce: The reduce function receives the sorted intermediate key-value pairs from the map functions and performs an operation that aggregates or summarizes them. It combines the values related to the same key and generates a collection of final output key-value pairs.

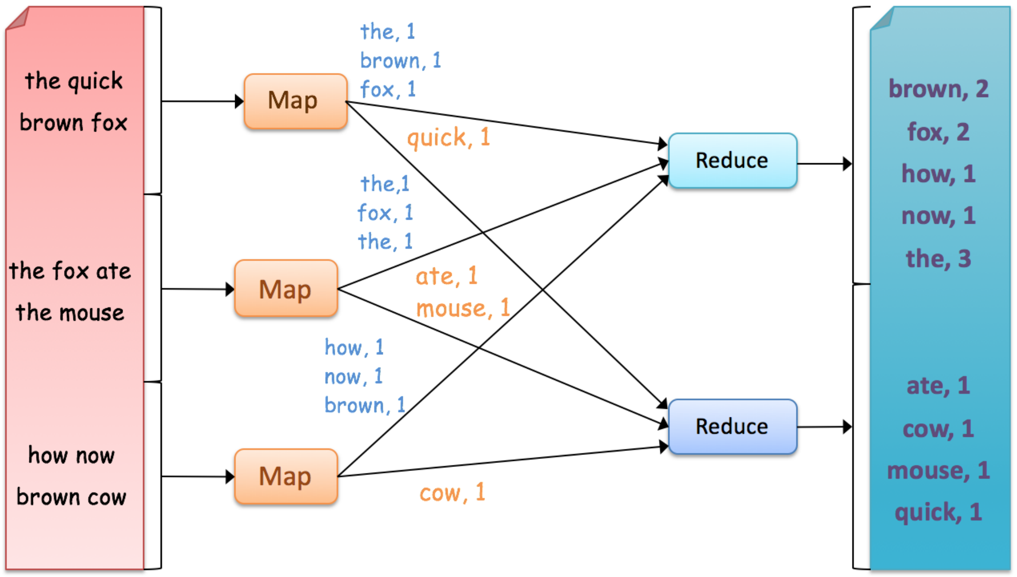


Figure 4.6: Word count using Map reduce [22]

Fig 4.6 shows how the word count application can be executed using word count. Simply the sentences are agiven as input mapper will tokenize the word and then found the key value pair as word and count respectively which is supplied to reducer to aggregrate the count of its word and atlast count will be given as output. There can be simultaneously many map functions running and as map functions completes its task , intermediate results will be passed to the reduce function to aggregrate the result and output will be generated. Also there can be multiple reduce functions running simultaneously.

4.2.1 Map-reduce application on our Master-Slave Architecture

We have implement map-reduce same kind of application on the master slave architecture with help of the data structure called HashMap. HashMap store data in the format of key value pair only. The thread will actually do similar task of map operation of extraction and generate HashMap which is intermediate result of key-value pairs which will be passed to the slave. Slave collect the result from all the threads created and do the aggregration on the intermediate result similar like reduce operation. After that again aggregrated intermediate result will be sent to the master, which will be collecting such results from all the slaves and aggregrating the results to get the final output. So based on our master slave architecture map operation is applies at the third layer which is thread which is actually perform task and two time reduce operation is applies at slave level and at master level to aggregrate the results and get the final output respectively.

4.3 Chapter Summary

This chapter has discussed bried about the our three tier master-slave architecture and implementation of map reduce application on the architecture. Next chapter will give the brief description about the setup of the architecture and implementation of the three different applications on the master slave architecture.

CHAPTER 5
Methodology

5.1 Setup

5.1.1 Hardware Setup

Initially, we collected unused but active mobile phones from our family and friends. After gathering the mobile phones of different specifications, we rooted the phone to get access to the terminal emulator of Android OS, which is the flavor of Linux only. However, we could not access the kernel level due to the system's restriction of access to the normal user.

Name of Mobiles	**Processor**	**Processor Speed(in GHz)**	**Number of cores**	**RAM (in GB)**	**Storage (in GB)**	**Read /Write Speed(in GHz)**	**Android Version**
Lenovo K5	ARM Cortex A53 415	1.3	4	2	12	0.677	5.1
Moto G7	Kryo 250 636	1.7	8	4	64	0.933	10
Redmi 5	Cortex A53	1.8	8	3	32	0.933	8.1
Redmi 5A	Cortex A53	1.4	4	3	32	0.667	8.1

Table 5.1: Mobile Specifications

We have used around four different smartphones with diverse specifications regarding the number of cores and CPU speed. Varying RAM storage and different I/O read-write speed can lead to experimenting with various accuracies.

Also, phones vary from the older Android version of 5 to the newer version of 10. Table 5.1 displays a detailed description of the smartphones used and their specifications.

A IoT testbed was developed to collect the data. For this testbed, temperature sensor DHT11 was used to collect the temperature of the surrounding environment. DHT11 sensor was connected with the microcontroller Aruino Uno which supplies the power to the sensor, collectively we called it a single module.

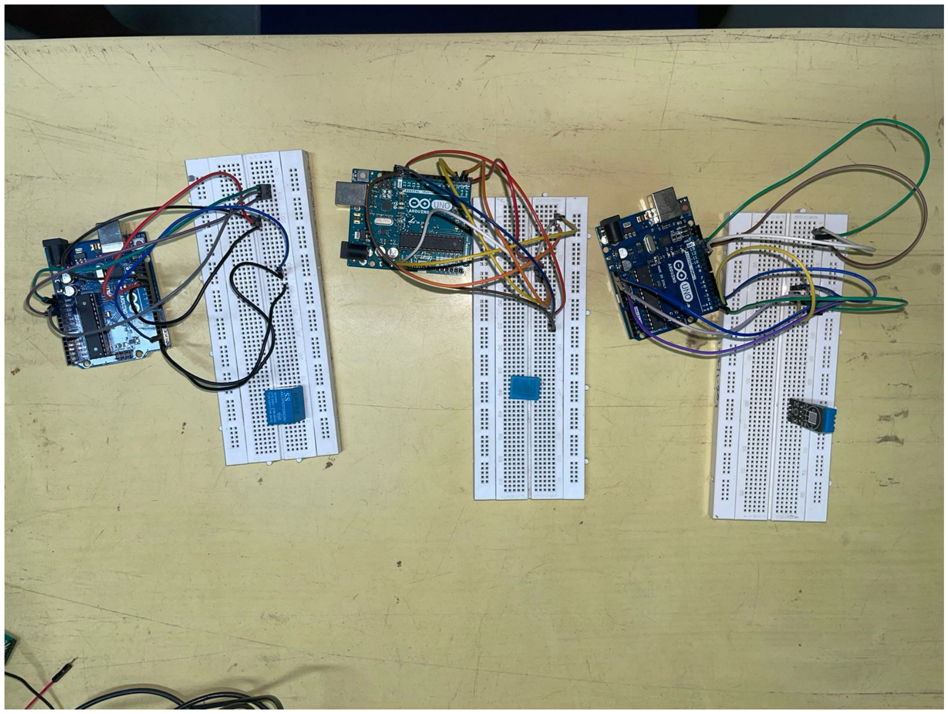


Figure 5.1: IoT testbed

IoT testbed shown in Fig 5.1 was developed by us which comprises of such three modules to collect the temperature data. A single module was made to collect data every 0.1 seconds. At the end of six months, around 32 crore samples were collected which is equivalent make to the 1GB of file size.

5.1.2 Software Setup

We developed an Android-based application based on master-slave using Android Studio, which is compatible with almost all mobile phones having any Android version. The benefit of using this software is that we can easily observe the CPU utilization and the memory used by the application we developed. Apart from this, Android Studio also shows the energy consumption by our application only for those mobile phones whose API version is greater than 26. There is the inbuilt interface of Android Studio called Profiler, which easily shows the CPU utilization of a particular application on mobile phones. Also, Arduino IDE software was used to simulate and program hardware to easily program any microcontroller available in the IDE list by connecting to a computer. Also, we can control the data transfer speed from the computer to the microcontroller and vice versa.

5.2 Experiments Performed

5.2.1 Word Count Application

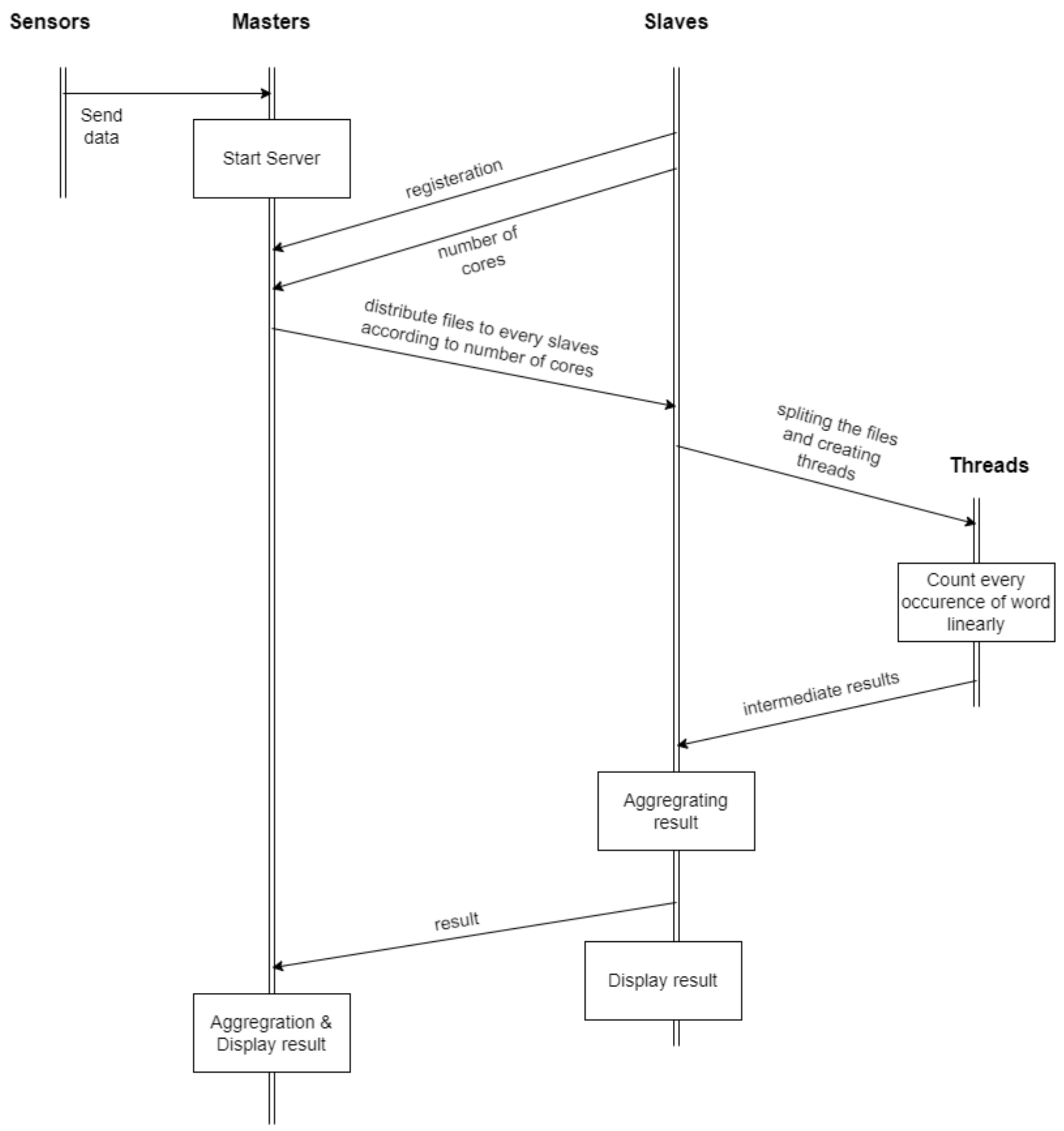

Sensors
Masters
Slaves
Send data
Start Server
registeration
number of cores
distribute files to every slaves according to number of cores
spliting the files and creating threads
Threads
Count every occurence of word linearly
intermediate results
Aggregrating result
result
Display result
Aggregration & Display result

Figure 5.2: Workflow of mapreduce application

Algorithm 1: Word Count master

```
Input   : Text File
Output: Count of every word
// Server class
class Server
    // Creating server
    Create Hash_Map with key and value Start ServerHelper Thread
    Click on Send Button()
    text_file= get from local_storage
    Divide text_file among clients in buckets according to cores
    // ServerHelper class
    class ServerHelper extends Thread
        Initialize the server socket and start with port number and IP
          address
        run()
        while true do
            socket ← Accept Client Connections
            Add new client into bucket
            ClientHandler handler ← new Client with socket;
            Start a handler for new client
            handler.start();
    // Handler class
    class ClientHandler extends Thread
        client= get new client
        outputstream=get outputstream from client
        inputstream=get inputstream from client
        cores= get cores of client
        run()
        client_hash_map= get from client client_count +=1
        if client_count == Bucket.length then
            for key in client_hash_map do
                value=Hash_Map(key)
                if value==null then
                    Hash_Map.put(key,value)
                else
                    Hash_Map.put(key,value+ client_hash_map(key))
            print(Hash_Map)
```

Algorithm 2: Word Count Slave

```
Input  : File chunk
Output: Occurence of every word
// Client class
class Client
    Create client_hash_map with key as word and value as count
    Click on Connect Button()
    Create client socket
    Connect client socket with server
    ClientHelper.start()
    // ClientHelper class
    class ClientHelper extends Thread
        client_word_file=Get chunk of file from server & store in file.
        splitting_file()
        Processing_thread[] of size equal to number of cores
        client_file_chunks[]=Divide client_word_file into equal chunks
        // where number of chunks equal to the number of cores
        for i=0 to core do
            Processing_thread[i].start(client_file_chunks[i])
        for j to cores do
            Processing_thread.join();
        outputstream.send(client_hash_map)
    // Processing class
    class Processing_thread extends Thread
        find_word_count()
        client_word_file_chunk=client_file_chunks
        while EOF do
            line=get line from client_word_file_chunk
            synchronized(lock)
            word[]=line.split(" ")
            for every w in word do
                count = client_hash_map(w);
                client_hash_map.put(w,count+1);
```

To check the feasibility of our system we have implemented Word Count problem which is one of the classic problems that can be solved using the map-reduce(MR) model. Fig 5.2 shows the workflow of the word count application on the master slave architecture. A text file is been given as input to the master which is splitted by the master into small parts and then passed to the slaves according to their number of cores. The reason for dividing file among slaves according to the number of cores is proper distribution of computation task among the slaves. Again slaves will split the files into chunks and will be given to the threads to perform the word count. The thread will traverse the file line by line in a linear fashion and split the line based on space as a delimiter. After tokenizing the line into word, thread will create a data structure called HashMap to store data in key-value pair, where key will be the word and value will be the count of that word. While traversing file if same word is encountered by thread, it will increase the count by 1. The intermediate result of key-value pair will be given to the slave and it will again aggregrate result into HashMap and same thing will be repeated by the master which collects the result and aggregates to find the final output as final count of each unique word.

5.2.2 Average of Temperature data

After checking the feasibility of word count application on master slave, we tried to implement our architecture to use as an edge architecture where statistical analysis can be done on the data that is collected from IoT testbed shown in fig 5.1.

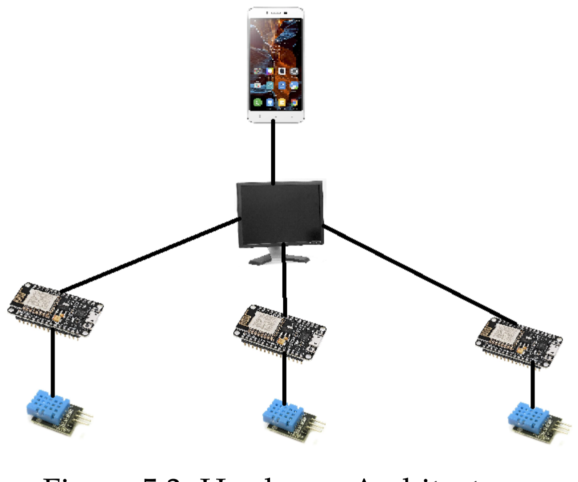


Figure 5.3: Hardware Architecture

Fig 5.3 show how the flow of data collected from IoT testbed will be collected at one device in CSV format and then it will be passed to our architecture for computation.

Algorithm 3: Temperature master [26]

```
Input  : Temperature File
Output: Average of temperature
// Server class
class Server
    // Creating server
    Create Hash_Map with key and value
    Create variable sum,count,average Start ServerHelper Thread
    Click on Send Button()
    temperature_file= get from local_storage
    Divide temperature_file among clients in buckets according to cores
    // ServerHelper class
    class ServerHelper extends Thread
        Initialize the server socket and start with port number and IP
          address
        run()
        while true do
            socket ← Accept Client Connections
            Add new client into bucket
            ClientHandler handler ← new Client with socket;
            Start a handler for new client
            handler.start();
    // Handler class
    class ClientHandler extends Thread
        client= get new client
        outputstream=get outputstream from client
        inputstream=get inputstream from client
        cores= get cores of client
        run()
        client_hash_map= get from client client_count +=1
        if client_count == Bucket.length then
            for key in client_hash_map do
                value=Hash_Map(key)
                if value==null then
                    Hash_Map.put(key,value)
                else
                    Hash_Map.put(key,value+ client_hash_map(key))
            for key in Hash_Map do
                sum+=(Hash_Map(key)*value)
                count+=Hash_Map(key)
            average=sum/count
            print(average)
```

To begin with experiment to find the average of the temperature data, first of all CSV file containing sensor data was sent to the master. Then master split the file among the slaves according to cores similar like the word count application as discussed in 5.2.1 and again slaves will split file among the threads to count ocuurences of the temperature data. The thread will traverse the file chunk line by line in a linear fashion and splits every data by considering the space as a delimiter. While traversing, it will try to map that data and increase the count of that data by 1. Once the thread traverses the whole file, it will have the frequency of every unique temperature data, so now it will return that count to the slave in the format of key-value pair where the key α will be the data and the value $\beta(\alpha)$ will be the count of that temperature data α, equivalent to the MR model's map operations.

Slaves will collect such key-value pairs from the threads and aggregate the count ζ_i corresponding to the key α_i, which is actually the data by traversing over all keys and all threads which can be given as,

$$\zeta_i = \sum_{n=1}^{N} \beta(\alpha_i) \tag{5.1}$$

where N is the total number of threads created by the slave, and i moves from 1 to the total number of unique keys (U).

And then, the master will aggregate every response from the slaves in the same way. So, we will finally get the count of every unique data from the file, equivalent to the MR model's reduced operation. So now the master will have the key α_i as the data and value ρ_i as the final count of that data by aggregating the value ζ_i over every key α_i given by thread which can be given as

$$\rho_i = \sum_{n=1}^{M} \zeta_i \tag{5.2}$$

where ρ is the final count of each key α, and M is the total number of slaves and i moves from 1 to total number of unique keys (U).

To calculate the average *avg* the total sum is found out by multiplying every key α which is the temperature data with its count ρ in the entire file and then dividing with the total number of samples which is equilatent to sum of final

Algorithm 4: Temperature Slave with Constant Threads[29]

```
Input   : Temperature File chunk
Output: Count of every temperature
// Client class
class Client
    Create client_hash_map with key as word and value as count Click on
      Connect Button()
    Create client socket
    Connect client socket with server
    ClientHelper.start()
    // ClientHelper class
    class ClientHelper extends Thread
        client_temperature_file=Get chunk of file from server & store in
          file.
        splitting_file()
        Processing_thread[] of size equal to number of cores
        client_file_chunks[]=Divide client_temperature_file into equal
          chunks
        // where number of chunks equal to the number of cores
        for i to cores do
            Processing_thread[i].start(client_file_chunks[i])
        for j=0 to cores do
            Processing_thread.join();
        outputstream.send(client_hash_map)
    // Processing class
    class Processing_thread extends Thread
        find_temperature_count()
        client_temperature_file_chunk=client_file_chunks
        while EOF do
            line=get line from client_temperature_file_chunk
            synchronized(lock)
            count = client_hash_map(line);
            client_hash_map.put(line,count+1);
```

count of every α which is given by

$$avg = \frac{\text{Sum of all samples}}{\text{Total number of samples}} \tag{5.3}$$

$$\frac{\sum_{i=1}^{U}(key_i \times \text{count of key}_i)}{\sum_{i=1}^{U} \text{count of key}_i} \tag{5.4}$$

$$= \frac{\sum_{i=1}^{U} \alpha_i \times \rho_i}{\sum_{i=1}^{U} \rho_i} \tag{5.5}$$

where U is the total number of keys.

At the end, master will find the average of the temperature data using the master slave application. The benefit of our application is that the connection is wireless, so once a task is given to the master, it can divide among the slaves who are in the range of its network as we were connected to the Wi-Fi router that allows a range of 45 meters. As each thread has to traverse over each file

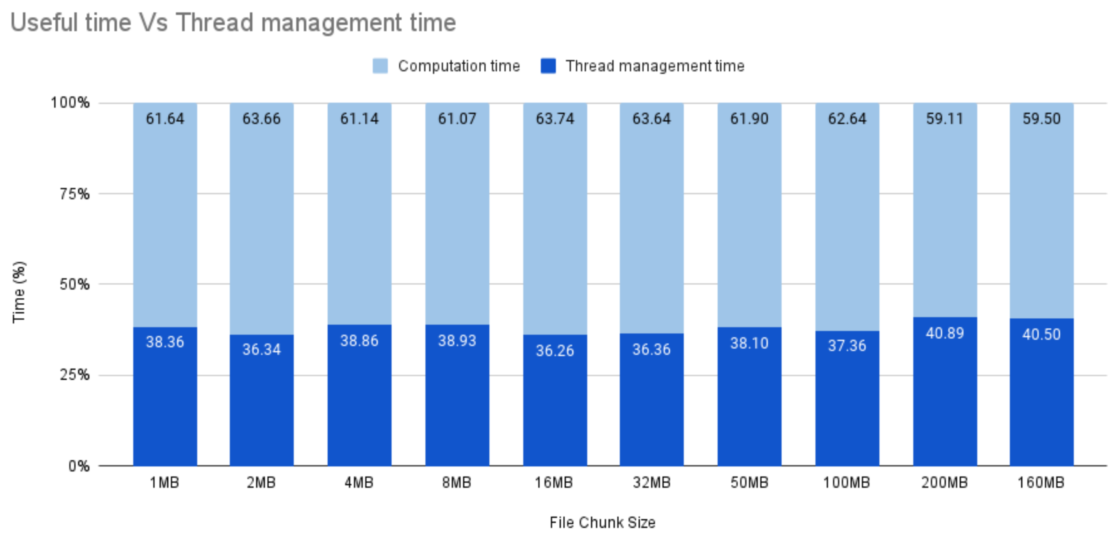


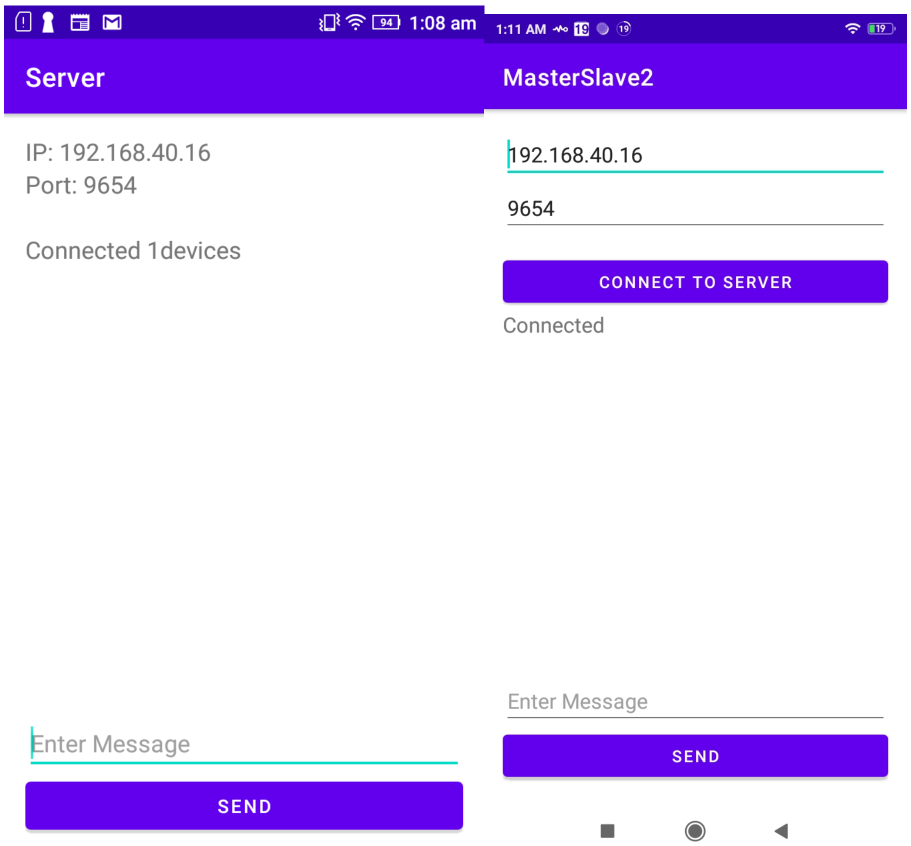
Figure 5.4: Computation time vs Thread Management time

For this experiment, we arbitrarily chose one mobile device as a master and the rest as slaves. Our main idea was to find out whether smartphones are able to perform computations. If yes, then what can be the optimal file size? Our hypothesis is whether we can use smartphones as computing devices or not. For this, we have experimented with different file sizes varying from 2.5MB to 1GB with different numbers of slaves to observe the behavior of the algorithm on our master-slave architecture. So, currently, manually, we are providing a file to the master, but it can be automated to collect data from the sensor directly. For scenario 1 discussed in 4.1.1, a single octa-core phone and a single quad-core phone were used as slaves responding to create 8 and 4 threads, respectively, because threads will be created equal to the number of cores. So as soon as the file is received from the master, the slave will store the data from the communication channel to a file. And then, that file is divided into chunks as per the cores and provided to the threads, so now the thread will have that chunk and will calculate the count of each temperature data from that chunk.

Algorithm 5: Temperature Slave File Constant[28]

```
Input  : Temperature File chunk
Output: Count of every temperature
// Client class
class Client
    Create client_hash_map with key as word and value as count
    Click on Connect Button()
    Create client socket
    Connect client socket with server
    ClientHelper.start()
    // ClientHelper class
    class ClientHelper extends Thread
        client_temperature_file=Get chunk of file from server & store in
          file.
        splitting_file()
        total_chunks=client_temperature_file/fixed file chunk size
          client_file_chunks[]=Divide client_temperature_file into equal
          fixed file size chunks
        Processing_thread[] of size equal to total_chunks
        for i=0 to total_chunks do
            Processing_thread[i].start(client_file_chunks[i])
        for j=0 to total_chunks do
            Processing_thread.join();
        outputstream.send(client_hash_map)
    // Processing class
    class Processing_thread extends Thread
        find_temperature_count()
        client_temperature_file_chunk=client_file_chunks
        while EOF do
            line=get line from client_temperature_file_chunk
            synchronized(lock)
            count = client_hash_map(line);
            client_hash_map.put(line,count+1);
```

For scenario 2, to fix the file chunk size, we ran a small experiment for a particular file size; we divided it into different file chunk sizes and measured the thread management time with the computation time. As shown in 5.4, we ran for 160MB main file size, which was further divided into smaller different file chunk sizes and we analyzed the percentage difference between time taken for thread management and time taken for computation out of total time. The optimal answer will be in which thread management time will be the lowest compared to all others. in our case for 160 MB file size we got the 16MB as ideal because thread management time is the lowest among all different file chunk sizes, which is 1/10 of 160MB file.



IP: 192.168.40.16
Port: 9654

Connected 1devices

192.168.40.16

9654

Connected

(a) master (b) slave

Figure 5.5: Master Slave Connection

Similarly, we ran the experiment for 320MB, for which we got 32MB as ideal, so from this, we can take file chunk size as 1/10 of the primary file size. In our experiment, we are going maximum up to 1GB file size for which we have taken 100MB file size as constant.

In above both experiment, if the algorithm is traversed sequentially then complexity comes out to be O(n) where n is the total samples collected in the file. Comparatively if same algorithm is executed on master slave architecture which has two slaves of same cores, then for single slave which is octacore complexity comes out to be O(n/8) because 8 threads will be created which will execute on the 8 file chunk size of equal size in the parallel manner.

And then, it was performed on the same two mobile phones to record the performance metrics based on the number of threads created based on file chunk size. After fixing the file chunk size, when the slave receives the file it will be divided into chunks of fixed size chunks and again thread will have each chunk to work upon it to find word count.

The UI part of the master application as well as for the slave is shown in images shown in Fig 5.5.

5.3 Indoor Localization

A real life application indoor localization was implemented on the master-slave architecture which actually will locate the location of a person based on the magnetic field. Person is told to walk about 1-2 meters and based on the magentic signal collected, it is matched with the magnetic signals that is collected in the database. Matching with the database signal is done with the Dynamic Time Warping(DTW) which is discussed later in next chatper.

5.4 Chapter Summary

This chapter builds the foundation of this project, describing the architecture and its application for using the master-slave kind of architecture. We saw the detailed working of the algorithm on this architecture.

CHAPTER 6

Indoor Localization

6.1 Data Collection

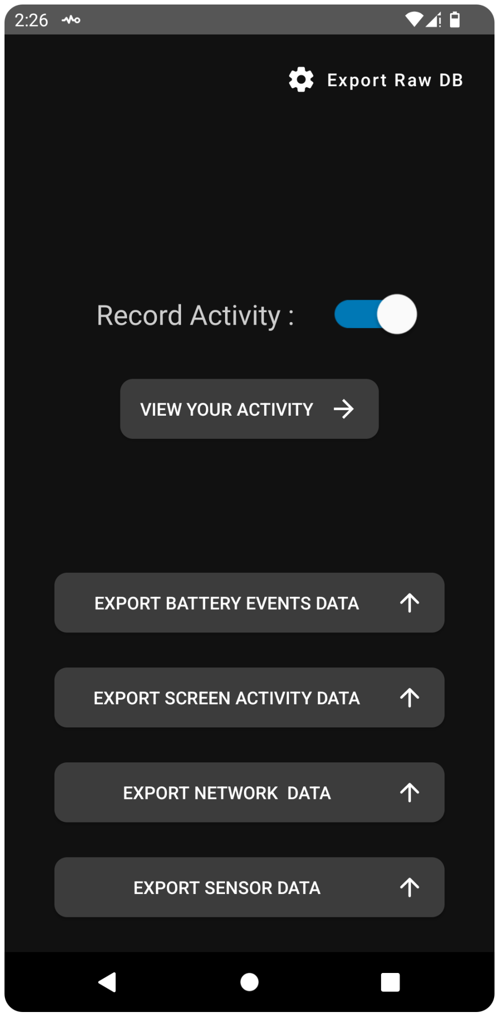


Figure 6.1: User Interface of Usage Tracker

To implement the experiment, we employed the above three mobile phones mentioned in Table 5.1 to collect the magnetic field data from the Android-based application, which was developed in-house to collect sensor data named as Usage Tracker App whose interface shown in Fig 6.1.

The magnetic field data were collected in the hallways of three floors of the Laboratory building. Around 200 samples were collected for each floor, out of which only the X-axis of the magnetic field was considered for our computation. Likewise, around 4 times, this task was repeated to collect the magnetic signal for each floor, So at the end, around 12 magnetic signals were collected, 4 signals per floor.

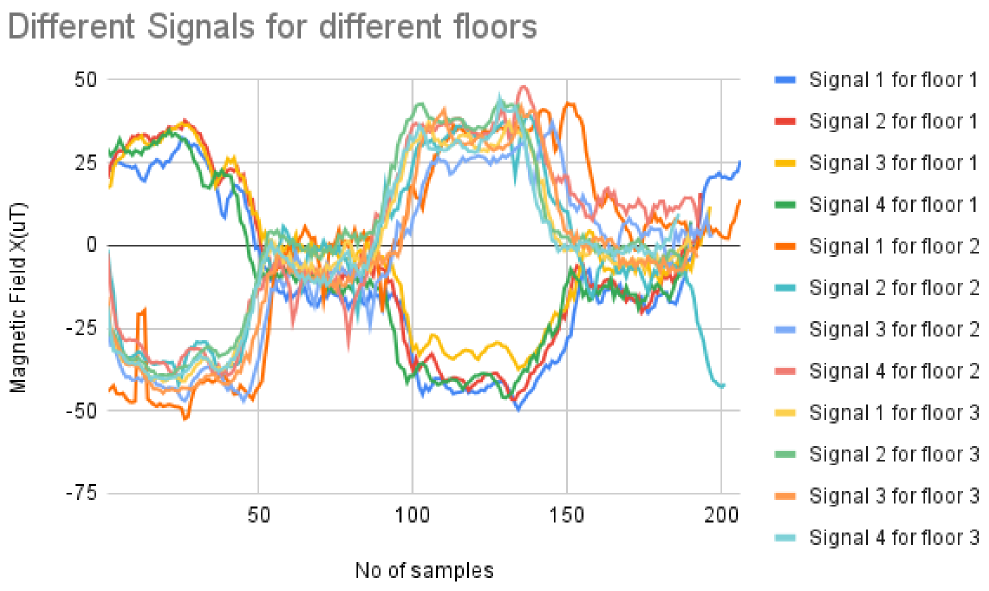


Figure 6.2: Different Signals

Fig 6.2 show the signal data collected each time or all hallways difference in every signal is due to walking speed which ranged from 0.98m/s to 1.5m/s. Also, the different orientation of mobile phones during walking affects the data.

6.2 Dynamic Time Warping

Dynamic Time Warping (DTW) is a method for comparing and aligning two temporal sequences that may differ in timing or speed. It is frequently used in the fields of data mining[34], speech recognition[40], and signal processing[13].DTW tries to achieve a better alignment among the two-time series by compressing or stretching either of the series. DTW aims to find an optimal alignment between two sequences, allowing for variations in the timing and speed of the underlying processes.

In general, consider two signals, $P = \{p_1, p_2, ..., p_A\}$ and $Q = \{q_1, q_2, ..., q_B\}$ of different lengths. The goal is to find the best match between the two signals by some alignment w, the optimal warping path. The warping path is given by $w = w(1), w(2), ..., w(n)$, where $w_n = [i(n), j(n)]$ is the set of matched samples, where i and j corresponding to the value of two signals respectively. The objective of the warping function is to minimize the overall cost function given by

$$Y = \sum_{n=1}^{K} cost(w(n)) \tag{6.1}$$

where $cost(w(n))$ is the absolute difference between the sample points given by

$$cost(w(n)) = abs(i(n) - j(n)) \tag{6.2}$$

To generate a warping path, a cost 2D matrix is constructed. The cost matrix represents the minimum cost required to reach a particular point (i, j) from $(1, 1)$. This minimization problem is usually solved using the dynamic programming approach, whereby a cumulative distance $\gamma(i, j)$ is computed as the sum of the distance obtained from the current set of points $(cost(w(n)))$ and the minimum cumulative distances of the adjacent elements or neighbors. This is given by

$$\gamma(i, j) = cost(w(n)) + min[\gamma(i-1, j), \gamma(i-1, j-1), \gamma(i, j-1)] \tag{6.3}$$

After performing time warping closest match between two signals is obtained by the lowest cumulative distance. The algorithm works by constructing a distance matrix where each element in the matrix represents the cost of aligning two specific elements from the sequences. The distance matrix is then populated by finding the minimum cost path from the starting point to the endpoint. This path represents the optimal alignment between the two signals.

6.2.1 Sliding Window

As mentioned above, DTW uses dynamic programming to find the cumulative distance difference between two signals. So for this, each and every point on the signal is traversed, and the minimum distance is found from each other. In general, consider two signals, $P = \{p_1, p_2, ..., p_A\}$ and $Q = \{q_1, q_2, ..., q_B\}$ where Q is reference signal from the database, and P is test signal which is to be used to find the distance from S and also $A < B$. So to find the DTW distance between signal P and signal Q, the best idea is to use the sliding window concept. So for that, we will take a window signal δ size of A from signal S and find the DTW distance.

$$\delta(i) = \{q_i, ..., q_{i+A}\} \tag{6.4}$$

, where i move from 1 to $B - A$. After every iteration, a window will slide by one element till $B - A$ is reached. So now, DTW distance will be given by

$$D = \min_{n=1}^{B-A} dtw(\delta(n), T) \tag{6.5}$$

Every window will be calculated with the test signal. Then distance will be returned, out of which minimum distance will be extracted, and it can be concluded that the test signal best fits that window of the reference signal.

6.2.2 Methodology

We have implemented DTW on master-slave with the concept of parallelism. Though DTW distance is dynamic programming in which tasks are dependent on each other it is not possible to implement the parallel task to be performed. One test signal of around 30 samples was collected, and the actual task was to match with one of the collected 12 magnetic signals Fig6.2.

First of all, the master will have a test signal as well as a database of all signals. So master will divide the database among the slaves according to the number of cores. So if one slave is octa-core and another is quadcore then the master will divide the database in 2:1 respectively. Also master will send the test signal, which has to be matched with one of the databases signals to every slave.

Now slave will try to divide the task and try to make it parallel by using the concept of thread. The slave will create threads equal to the number of database signals that it receives from the master. So each thread will be allocated one database signal.

The thread will do the actual task of finding the DTW distance between the database signal and the test signal. After calculating the DTW distance, the window will slide by one element, and again DTW distance is calculated. The total number of windows ω for the sliding window protocol can be defined as:

$$\omega = B - A + 1 \tag{6.6}$$

where B is the total samples of a single database signal and A is the total samples of the test signal. each window size is equal to the size of the test signal.

Algorithm 6: Algorithm for Server to find DTW distance between magnetic signals[30]

```
Input  : Database of Signals,Test Magnetic Signal
Output: Minimum DTW distance between Database signal and Test signal
// Server class
class Server
    // Creating server
    minimum_dtw_distance=MAX Start ServerHelper Thread
    Click on Send Button()
    test_signal= get from local_storage
    db_signals[]=get db signals from local storage
    Divide db_signals among clients in buckets according to cores
    // ServerHelper class
    class ServerHelper extends Thread
        Initialize the server socket and start with port number and IP
          address
        run()
        while true do
            socket ← Accept Client Connections
            Add new client into bucket
            ClientHandler handler ← new Client with socket;
            Start a handler for new client
            handler.start();
    // Handler class
    class ClientHandler extends Thread
        client= get new client
        outputstream=get outputstream from client
        inputstream=get inputstream from client
        cores= get cores of client
        run()
        dtw_distance= distance
        signal_number=db_signal number from client
        client_count +=1
        min_DTW_distance= min(min_DTW_distance,dtw_distance)
        if client_count == Bucket.length then
            print(min_DTW_distance)
```

Algorithm 7: Algorithm for Client to find DTW distance between magnetic signals[27]

```
Input  : Few Database Signals, Test Signal
Output: Minimum DTW Distance
// Client class
class Client
    Click on Connect Button()
    Create client socket
    Connect client socket with server
    ClientHelper.start()
    // ClientHelper class
    class ClientHelper extends Thread
        test_signal=Get test signal from the server & store in a file
        client_db_signals[]=Get each database signal from server & store in
          file.
        splitting_file()
        Processing_thread[]=processing_thread[client_db_signals.length]
        from i to client_db_signals.length
        Processing_thread[i].start(client_db_signals[i])
        from j to client_db_signals.length
        Processing_thread.join();
        outputstream.send(minimum)
    // class
    class Processing_thread extends Thread
        find_dtw()
        n=test_signal.length
        m=client_db_signal.length // db signal got from helper class
        dtw_matrix[n][n]
        for i=0 to m-n do
            Set dtw_matrix[][] as max value dtw_matrix[0][0]=0; for j=0 to n
              do
                for k=0 to n do
                    dtw_cost=absoulue.(client_db_signal[k+j-1]-
                      test_signal[i-1] )
                    dtw_matrix[i][j]=dtw_cost+ min(dtw_matrix[i-
                      1][j],dtw_matrix[i][j-1],dtw_matrix[i-1][j-1])
            synchronized(lock)
            if dtw_matrix[n][n]<minimum then
                minimum=dtw_matrix[n][n];
```

Test Signal Size (A)	Total Windows (ω)
10	191
20	181
30	171
40	161
50	151

Table 6.1: Number of windows created with different test signal sizes

Table 6.1 shows the total no of windows created with different no of samples in test signal where size of database signal (B) is 200 samples which is same.

Here, each thread will work on equation 6.5, where each thread will find the D. Therefore slave will find the minimum distance based on DTW distance received from the threads. Thus it can be given as follows.

$$slave\_DTW = \min\{D_1, D_2, ..., D_n\} \tag{6.7}$$

where n= number of threads created by the slave.

This is the minimum result returned by the slave to the master will again find the minimum DTW distance from the slaves which can given as

$$master\_DTW = \min\{slave\_DTW_1, slave\_DTW_2, ..., slave\_DTW_m\} \tag{6.8}$$

where m= number of slaves. And at the end, we will get the minimum DTW distance matching signal with one of the database signals. Based on this, we can decide the location of the person in the indoor environment.

6.3 Chapter Summary

This chapter will discuss about the brief description of implementation of indoor localization application on the master-slave architecture. The next chapter will see the specific factors' performance analysis and experimental results from the experiment performed.

CHAPTER 7

Performance Evaluation

This chapter will discuss the observation and analysis of the metrics based on the experiment performed.

7.1 Word Count and Average of Temperature Data

7.1.1 Scenario1-Thread count constant

We have performed this experiment on different mobiles with different cores as we have kept the thread count = number of cores.

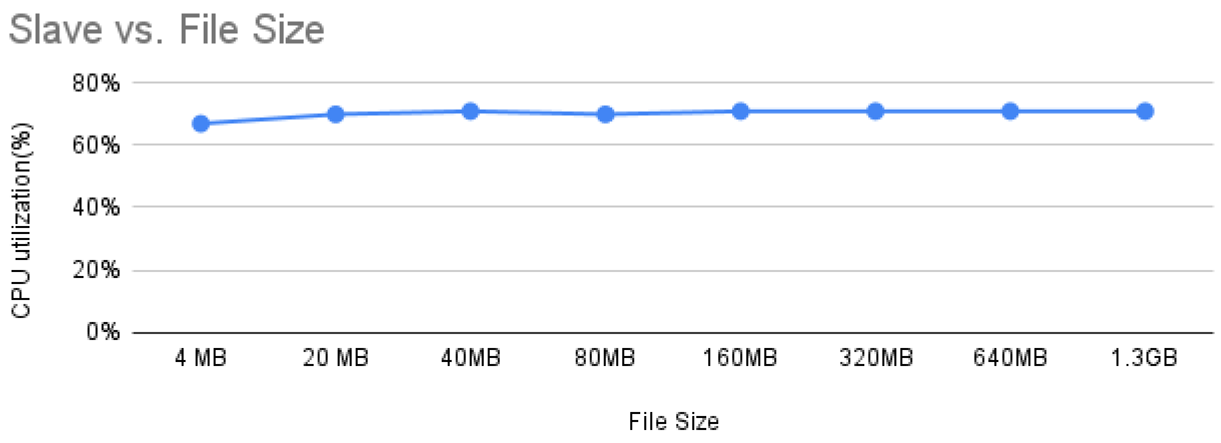


Figure 7.1: Case 1: CPU utilization with increasing file sizes

From the 7.1, we can easily observe that CPU utilization remains almost similar though there is an increase in the size of the file. This is because every time, the same number of threads will only be created, whatever will be the file size.

7.1.2 Scenario2- File chunk size constant

We observed that the number of threads increases with the primary file size increase as the file chunk size remains constant at 100MB so that when the file size increases and it is divided in to file chunk size of 100MB, which increases the number of threads that will work on the file chunk size created. Fig 7.2 shows the number of threads with increased file size for octa-core and quadcore. Similarly,

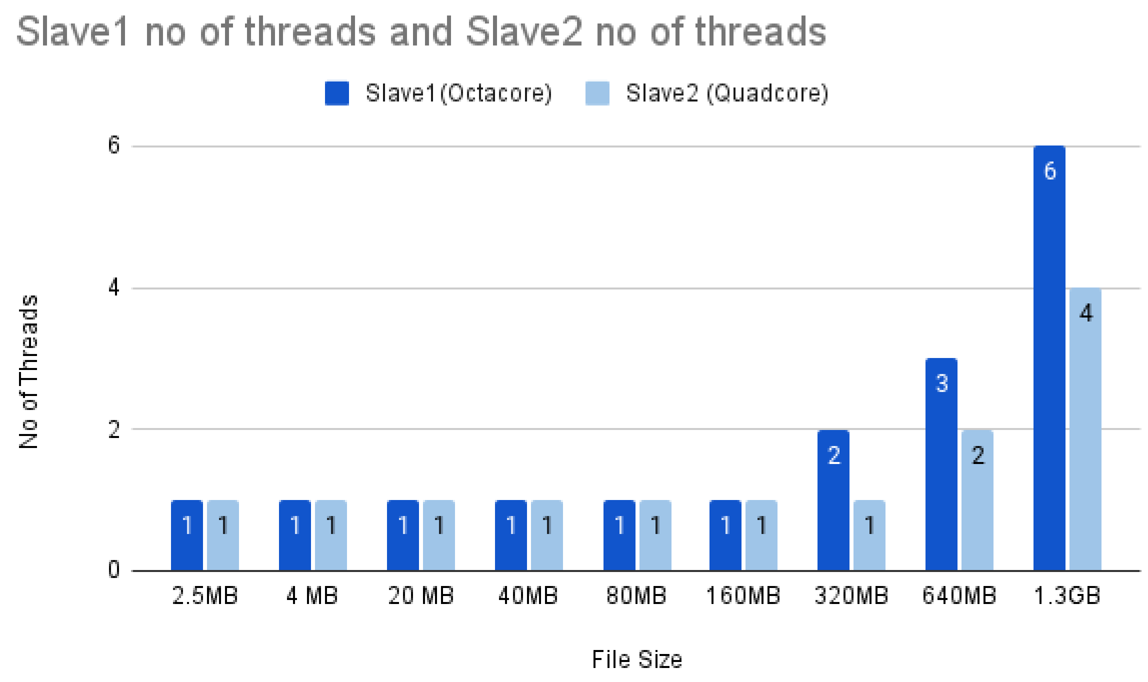


Figure 7.2: No of threads with increasing file sizes

Fig 7.3 shows the CPU utilization of mobile phones the application uses for octa-core and quad-core. Comparing both results in Fig, we can easily see that CPU utilization also increases as the number of threads increases. For quad-core, when a single thread is created, CPU utilization comes out to be 1/4, which is equivalent to 25%. Similarly, for octa-core single thread gives the CPU utilization = 1/8, equivalent to 12.5%. Although only 4 threads are created for the file size of 1.3 GB, it is higher CPU utilization because, for quad-core, four threads are the threshold, while even six threads are created for octa-core, which is less than the threshold of eight threads.

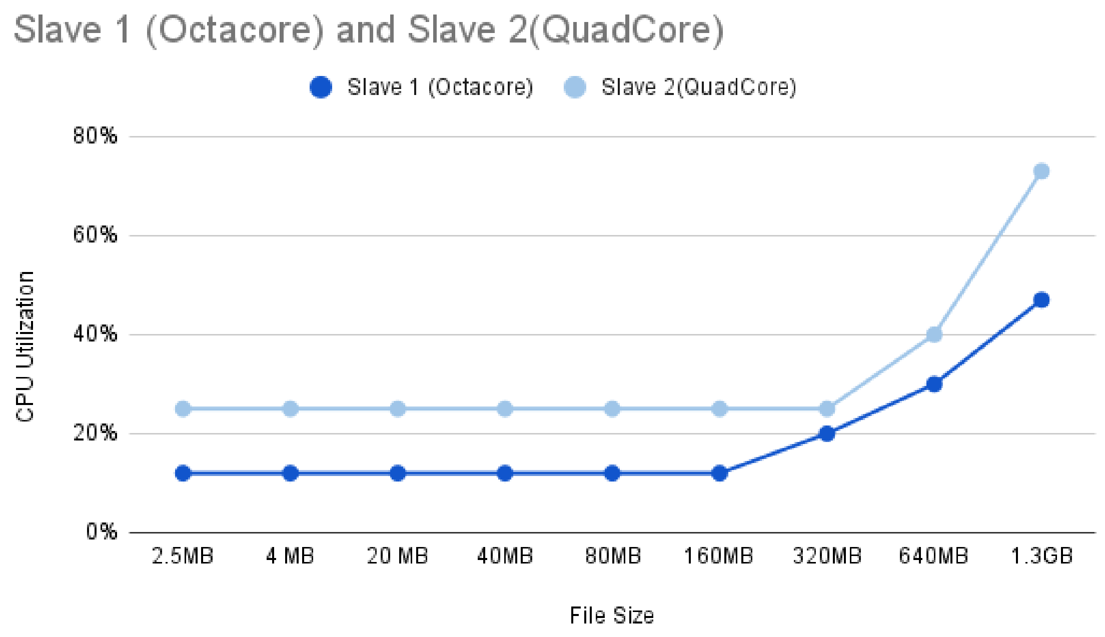


Figure 7.3: Case 2: CPU utilization with increasing file sizes

7.1.3 Comparison with Desktop

Apart from this experiment, we also performed the word count program on the Hadoop cluster using map-reduce, which was already set up on a desktop with a single master and two slaves. The table shows the specification of the desktop Table 7.1.

	Desktop	Mobile	
CPU speed	3.2 GHz	1.6 GHz	2X
RAM Read- write speed	19.2 Gb/s	3.2 Gb/s	6X
Bandwidth	10 MBps	1 MBps	10X
OS	Linux	Android	
Disk I/O	165.77 Mb/s	6.5 Mb/s	25X

Table 7.1: Comparison of Desktop with mobile

Fig 7.4 shows the computation time comparison between the Hadoop and Android clusters. The time for the Android cluster is almost square times the time taken by the Hadoop cluster. The reason behind such results can easily be observed from the table, which tells that desktop is much faster than mobile phones. Again everything comes with pros and cons within itself. Many such factors can

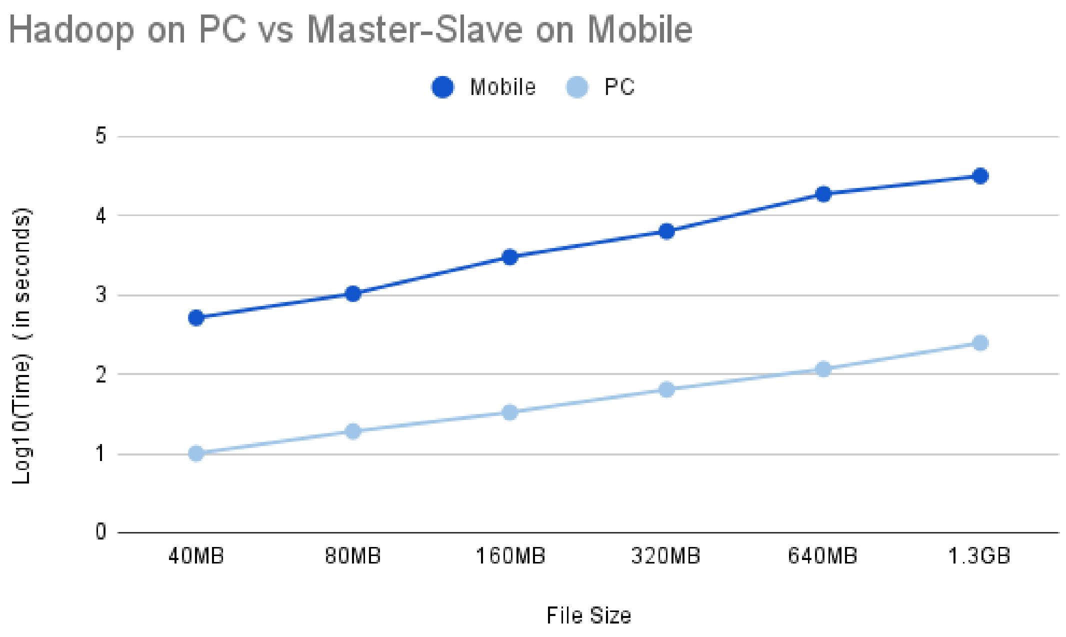


Figure 7.4: Time comparison of mobile with desktop

be taken into consideration, which are mentioned below.

- Cost:- The cost for the desktop/server is very much higher than the mobile phones. This assertion is based on the understanding that desktop computers generally have larger storage capacity and more advanced hardware components than mobile devices, which makes them more expensive to manufacture and purchase.
- Carbon Emission:- According to a study by [8], the total carbon footprint of a mobile device is typically lower than that of a desktop computer. The study found that the total carbon footprint of a mobile device (including manufacturing, usage, and end-of-life disposal) is 80% less than the carbon footprint of the desktop.
- Battery Power:- Desktop computers tend to consume more energy than mobile devices due to their larger size, more advanced hardware, and tendency to stay plugged in for longer. According to a study [32], the average power consumption of a desktop computer is almost 50% more than the average power consumption than mobile device.

7.2 Dynamic Time Warping

DTW is a computation-intensive program with a dynamic programming concept that traverses and puts values by calculating DTW distance in the 2D matrix. In addition to this, a sliding window concept is used so only a small window size equal to a smaller signal is taken, and DTW distance is found out, and every time a window will slide by one element after calculating for a particular window.

7.2.1 Time Comparison with single device

We have implemented a DTW algorithm on ms architecture as well as on a single device for comparing the algorithm's behavior. In single-device computation, for every single signal, total windows are calculated. And for every window, the 2D matrix is calculated based on the equation6.3. So complexity is O($k\omega n^2$) where k is no of total database signals, ω is total windows, and n^2 is the matrix size.

We have experimented with different numbers of database signals and analyzed the behavior from Fig 7.5, 7.6, 7.7, 7.8, 7.9 that almost DTW on master-slave is almost 50% faster than the single device when the no of samples increases. But there is not much difference in computation time for fewer samples. But when the number of samples increases, master-slave increases linearly while single device computation increases nearly equal to exponential. To work with more number of signals we replicated the same signals twice to get 24 signals, similarly thrice to get 36 signals. The reason for increasing number of signals to observe the behavior as well as CPU utilization of DTW algorithm respectively.

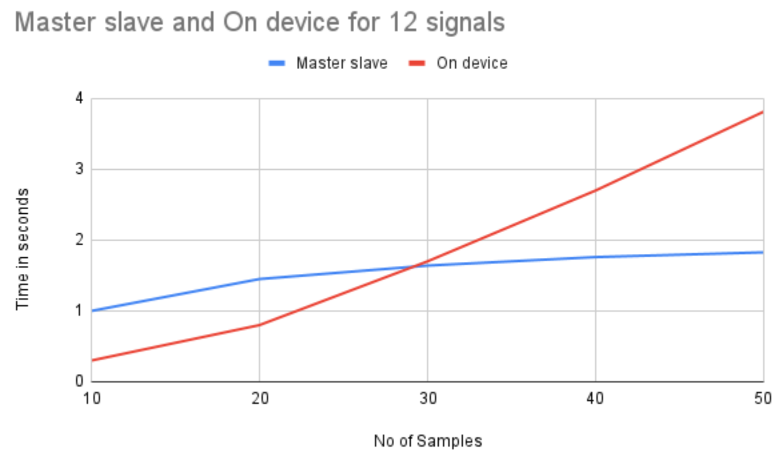


Figure 7.5: Time comparison with a single device for 12 database signals

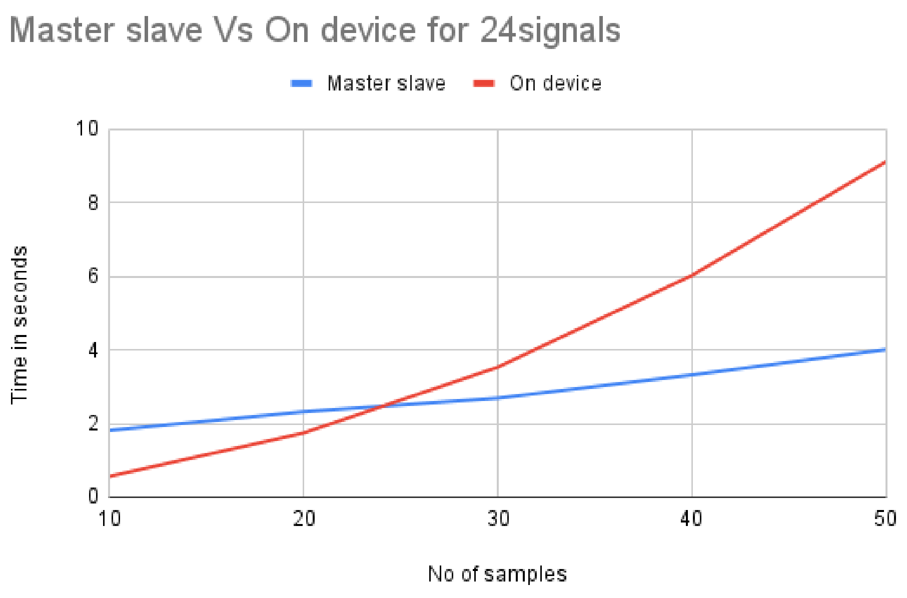


Figure 7.6: Time comparison with a single device for 24 database signals

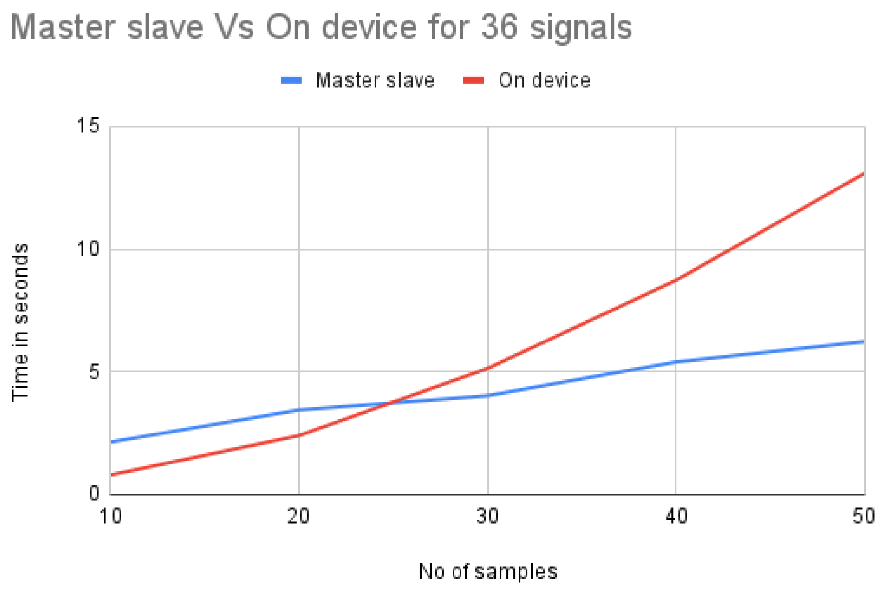


Figure 7.7: Time comparison with a single device for 36 database signals

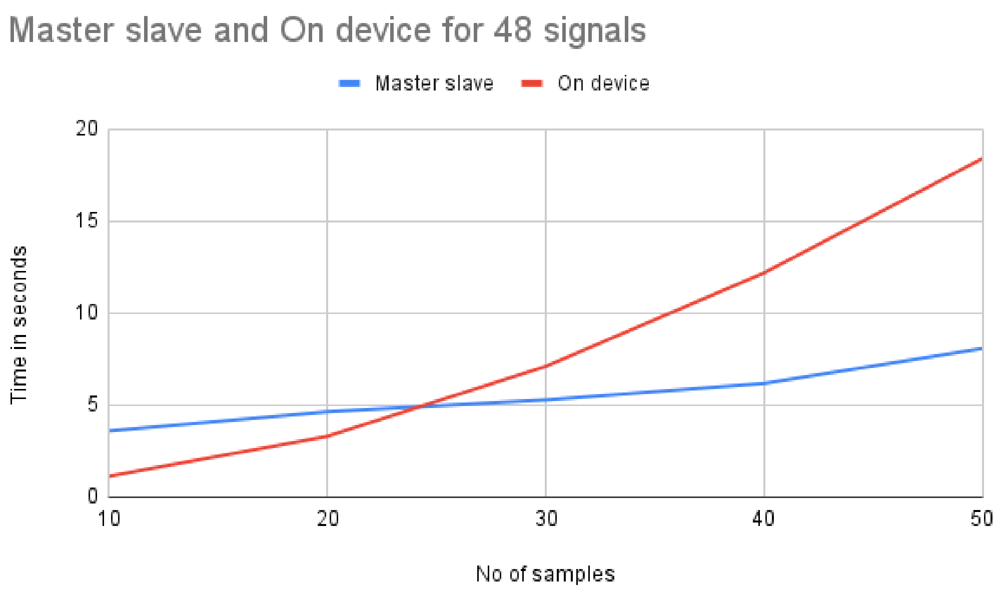


Figure 7.8: Time comparison with a single device for 48 database signals

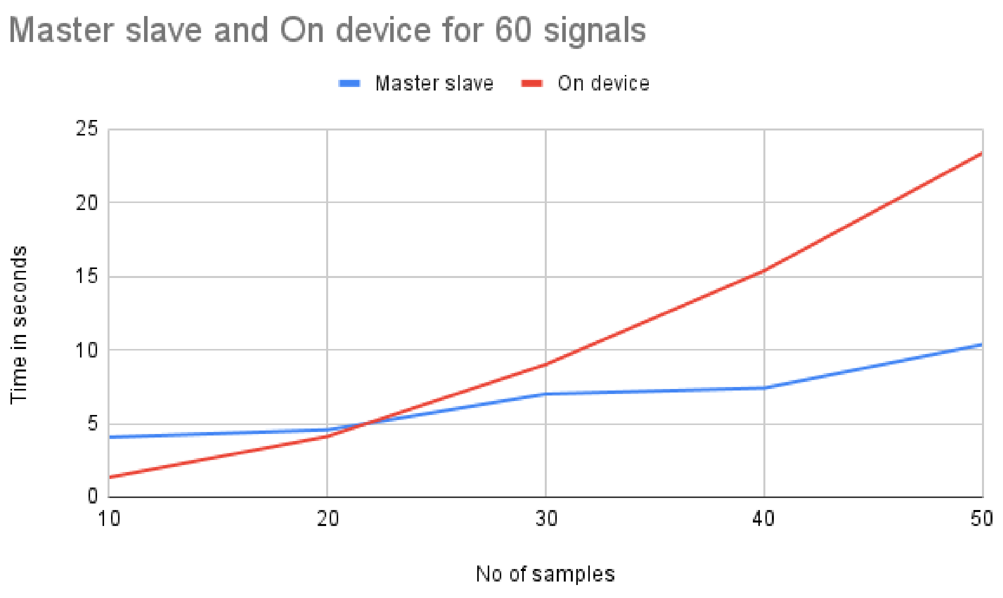


Figure 7.9: Time comparison with a single device for 60 database signals

7.2.2 CPU utilization

Our aim was to use the mobile phone's cores which were kept idle, so to measure this metric, we can have the CPU utilization, which says how much CPU is used for running the algorithm and that can be measured from the Android Studio

Profiler only. From fig 7.10, we can easily observe that CPU is almost utilized between 80% to 95%.

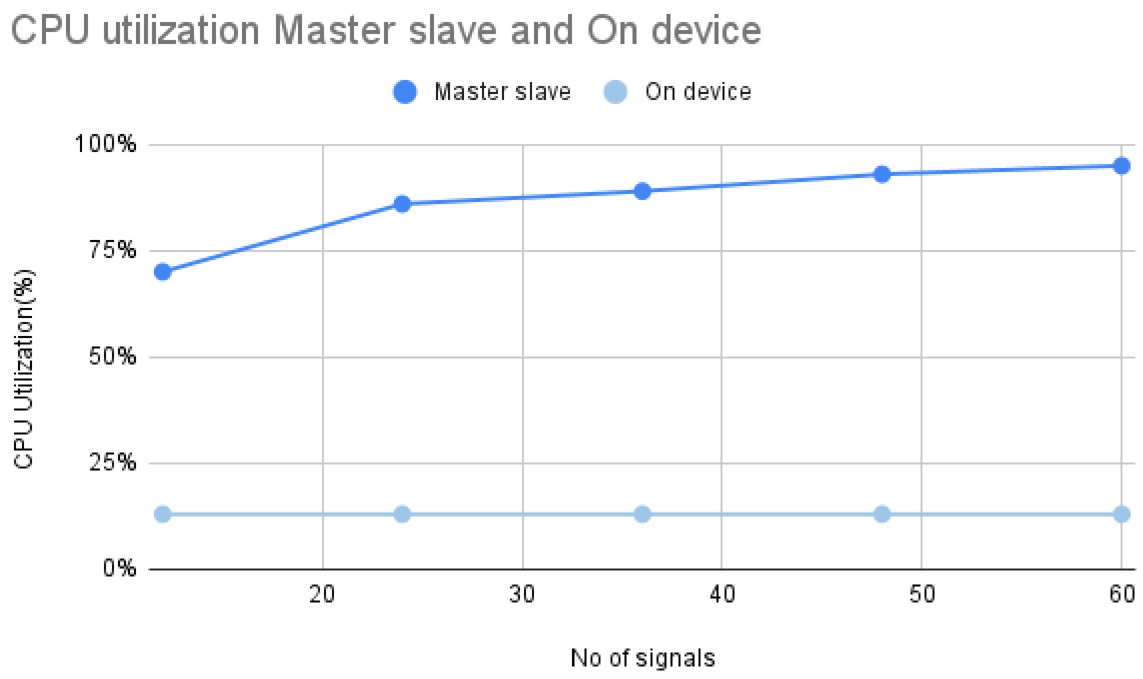


Figure 7.10: CPU utilization for DTW

At the same time, single device/on-device computation remains almost the same every time, with an increase in the number of samples due to only one thread being created for computation. In contrast, for master-slave, the slave will create threads equal to the number of database signals received from the master.

7.3 Chapter Summary

This chapter had discussed different metrics based on the experiment performed for different scenarios and applications. Apart from this also, we have compared the map-reduce task of wordcount on mobile and PC and DTW for MS and single device. The next chapter discusses the conclusions and future scope of the research work that can be carried out.

CHAPTER 8

Conclusion and Future Work

8.1 Conclusion

This thesis presents a three-tier architecture based on master-slave and the concept of parallelism using thread. Using master-slave architecture, computation is distributed over the devices while using threads; processing can be done in parallel, which saves computation time. Two applications:- word count and DTW, were executed on this MS architecture. A detailed description of how both applications are implemented on the MS architecture. Then a comparison of algorithms with metrics like CPU utilization, the computation time for octa cores and quad cores devices, and single device computation. Although both applications are different, word count is more I/O based and less computation. At the same time, DTW is a computation-intensive application in which a matrix is traversed to find the optimal DTW distance between two signals. For DTW, MS application is 50% faster than the single device and almost utilizes 7.5 times more CPU than single device.

8.2 Future Scope

To extend the work, master-slave architecture can be used for different applications. As this project can be improved by adding some functionalities. We can develop better code designs that reduce data transfer latency from master to slave. Especially for DTW, we can add the classifying the window to locate exact place for indoor localization. Also, it be useful as multiple users can access it at a time. Also, develop the user interface that shows the navigation for further destinations on the phone.

References

[1] Green grid data center power efficiency metrics: Pue and dcie [online]. 2008.

[2] Map your meal europe aid funded project. [online]. 2016.

[3] Delloite,https://www.business-standard.com/article/current-affairs/india, 2017.

[4] D. B. Abdullah and M. M. Al-Hafidh. Developing parallel application on multi-core mobile phone. *International Journal of Advanced Computer Science and Applications*, 4(11), 2013.

[5] V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. Burger. Clock rate versus ipc: The end of the road for conventional microarchitectures. 28(2):248–259, may 2000.

[6] B. Alessio, W. De Donato, V. Persico, and A. Pescapé. On the integration of cloud computing and internet of things. *Proc. Future internet of things and cloud (FiCloud)*, pages 23–30, 2014.

[7] O. Alzakholi, H. Shukur, R. Zebari, S. Abas, M. Sadeeq, et al. Comparison among cloud technologies and cloud performance. *Journal of Applied Science and Technology Trends*, 1(2):40–47, 2020.

[8] A. Andrae and O. Andersen. Life cycle assessment of consumer electronics - are they consistent? *The International Journal of Life Cycle Assessment*, 15:827–836, 07 2010.

[9] Apple. Apple website retrieved from https://nanoreview.net/en/soc/apple-a15-bionic.

[10] Apple. Mike Elgan. 2017. With smartphones like these, why do we need laptops? Retrieved from https://www.computerworld.com/article/3241233/with-smartphones-like-these-why-do-we-need-laptops.html. .

[11] Apple. Samuel Axon. 2021. iPhone 13 and 13 Pro review: If you could have three wishes. Retrieved from https://arstechnica.com/gadgets/2021/09/iphone-13-and-13-pro-review-if-you-could-have-three-wishes/.

[12] M. Y. Arslan, I. Singh, S. Singh, H. V. Madhyastha, K. Sundaresan, and S. V. Krishnamurthy. Cwc: A distributed computing infrastructure using smartphones. *IEEE Transactions on Mobile Computing*, 14(8):1587–1600, 2015.

[13] V. K. Bhardwaj and S. Maini. Measurement of micro-harmonic vibration frequency from the modulated self-mixed interferometric signal using dynamic time warping method. *Mechanical Systems and Signal Processing*, 168:108712, 2022.

[14] R. Bongiovanni and J. Lowenberg-DeBoer. *Precision agriculture*, 5:359–387, 2004.

[15] F. Bonomi, R. Milito, J. Zhu, and S. 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.

[16] F. Büsching, S. Schildt, and L. Wolf. Droidcluster: Towards smartphone cluster computing–the streets are paved with potential computer clusters. In *2012 32nd International Conference on Distributed Computing Systems Workshops*, pages 114–117. IEEE, 2012.

[17] F. Büsching, S. Schildt, and L. Wolf. Droidcluster: Towards smartphone cluster computing - the streets are paved with potential computer clusters. 06 2012.

[18] K. Cao, Y. Liu, G. Meng, and Q. Sun. An overview on edge computing research. *IEEE access*, 8:85714–85728, 2020.

[19] W. O. Cesário, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A. A. Jerraya, L. Gauthier, and M. Diaz-Nava. Multiprocessor soc platforms: a component-based design approach. *IEEE Design & Test of Computers*, 19(6):52–63, 2002.

[20] CISCO. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html. , 2018.

[21] M. O. DairyMaster. http://www.dairymaster.com/heat-detection/. 2016.

[22] N. ECSU/IU. Admi workshop,http://admicloud.github.io/www/wordcount.html, year = 2016.

[23] J. Garcia Almiñana, E. Simo, X. Masip, E. Marin-Tordera, and S. Sanchez-Lopez. Do we really need cloud? estimating the fog computing capacities in the city of barcelona. pages 290–295, 12 2018.

[24] Geekbench_11. John Poole. 2021. Introducing Geekbench 5. Retrieved from https://www.geekbench.com/ .

[25] S. George, T. Eiszler, R. Iyengar, H. Turki, Z. Feng, J. Wang, P. Pillai, and M. Satyanarayanan. Openrtist: end-to-end benchmarking for edge computing. *IEEE Pervasive Computing*, 19(4):10–18, 2020.

[26] GitHub. https://github.com/vyom18/server_for_temperature.

[27] GitHub. https://github.com/vyom18/slave_dtw.

[28] GitHub. https://github.com/vyom18/temperature_slave_fileconstant.

[29] GitHub. https://github.com/vyom18/temperature_slave_threadconstant.

[30] GitHubLink. https://github.com/vyom18/master_for_dtw.

[31] Herdwatch.[Online]. 2016.

[32] Z. hu and J. Ruutu. Comparison of energy consumption between a mobile device and a collection of dedicated devices. pages 1–6, 05 2011.

[33] Intel. https://ark.intel.com/content/www/us/en/ark/products/series/88393/6th-generation-intel-core-i5-processors.html. .

[34] E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping for datamining applications. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, page 285–289, New York, NY, USA, 2000. Association for Computing Machinery.

[35] J. Muangprathub, N. Boonnam, S. Kajornkasirat, N. Lekbangpong, A. Wanichsombat, and P. Nillaor. *Computers and electronics in agriculture*, 156:467–474, 2019.

[36] T. Nam and T. A. Pardo. Conceptualizing smart city with dimensions of technology, people, and institutions. In *Proceedings of the 12th annual international digital government research conference: digital government innovation in challenging times*, pages 282–291, 2011.

[37] F. G. . F. [Online]. 2016.

[38] S. Patel, K. Sasidhar, M. Vaghela, P. Katrodia, and R. Wagani. An m-health app that goes beyond screen time applications. In *2022 IEEE 10th Region 10 Humanitarian Technology Conference (R10-HTC)*, pages 94–99, 2022.

[39] S. Patel, K. Sasidhar, M. Vaghela, P. Katrodia, and R. Wagani. An m-health app that goes beyond screen time applications. In *2022 IEEE 10th Region 10 Humanitarian Technology Conference (R10-HTC)*, pages 94–99, 2022.

[40] Y. Permanasari, E. H. Harahap, and E. P. Ali. Speech recognition using dynamic time warping (dtw). *Journal of Physics: Conference Series*, 1366(1):012091, nov 2019.

[41] QUALCOMM. https://arstechnica.com/gadgets/2019/02/qualcomm-is-already-announcing-next-years-5g-chips. , 2019.

[42] M. Raihan, S. Mondal, A. More, M. O. F. Sagor, G. Sikder, M. A. Majumder, M. A. Al Manjur, and K. Ghosh. Smartphone based ischemic heart disease (heart attack) risk prediction using clinical data and data mining approaches, a prototype design. In *2016 19th International Conference on Computer and Information Technology (ICCIT)*, pages 299–303, 2016.

[43] A. P. Rajan et al. Evolution of cloud storage as cloud computing infrastructure service. *arXiv preprint arXiv:1308.1303*, 2013.

[44] N. Rajovic, P. M. Carpenter, I. Gelado, N. Puzovic, A. Ramirez, and M. Valero. Supercomputing with commodity cpus: Are mobile socs ready for hpc? In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, New York, NY, USA, 2013. Association for Computing Machinery.

[45] B. P. Rimal, E. Choi, and I. Lumb. A taxonomy and survey of cloud computing systems. In *2009 Fifth international joint conference on INC, IMS and IDC*, pages 44–51. Ieee, 2009.

[46] C. A. Ronao and S.-B. Cho. Human activity recognition with smartphone sensors using deep learning neural networks. 59(C):235–244, oct 2016.

[47] B. Schaffner, J. Sawin, and J. Myre. Smartphones as alternative cloud computing engines: Benefits and trade-offs. pages 244–250, 08 2018.

[48] B. Schaffner, J. Sawin, and J. M. Myre. Smartphones as alternative cloud computing engines: Benefits and trade-offs. In *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 244–250, 2018.

[49] B. Schaffner, J. Sawin, and J. M. Myre. Smartphones as alternative cloud computing engines: Benefits and trade-offs. In *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 244–250, 2018.

[50] C. L. Seitz et al. System timing. *Introduction to VLSI systems*, pages 218–262, 1980.

[51] A. W. services website. https://aws.amazon.com/getting-started/projects/host-net-web-app/services-costs/. .

[52] M. P. Singh and M. K. Jain. Evolution of processor architecture in mobile phones. *International Journal of Computer Applications*, 90(4):34–39, 2014.

[53] J. Switzer, G. Marcano, R. Kastner, and P. Pannuto. Junkyard computing: Repurposing discarded smartphones to minimize carbon. New York, NY, USA, 2023. Association for Computing Machinery.

[54] J. Switzer, G. Marcano, R. Kastner, and P. Pannuto. Junkyard computing: Repurposing discarded smartphones to minimize carbon. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 400–412, 2023.

[55] E. topics blog. https://explodingtopics.com/blog/smartphone-stats. , 2019.

[56] M. Vaghela, K. Sasidhar, A. Parikh, and R. Wagani. Assessing mobile usage, physical activity and sleep through smartphone sensing: A digital phenotype study. *SN Computer Science*, 3, 06 2022.

[57] M. D. Ventra and Y. V. Pershin. The parallel approach. *Nature Physics*, 9:200 – 202, 2013.

[58] D. Yates and M. Islam. Readiness of smartphones for data collection and data mining with an example application in mental health. 10 2019.