A Kernel Level Implementation of Hadoop

Progress Report

In fulfillment of the requirements for the

NU 302 R&D Project

At NIIT University

Submitted by

U101115FCS055:	Antra Tripathi
U101115FCS243:	Debanshu Majumdar
U101115FCS090:	Godavarthi Siddharth
U101115FCS098:	Hitesh Yadav
U101115FCS117:	Mukul Gupta

Computer Science Engineering

NIIT University

Neemrana

Rajasthan

CERTIFICATE

*This is to certify that the present research work entitled "***A Kernel Level Implementation of Hadoop***" being submitted to NIIT University, Neemrana, Rajasthan, in the fulfillment of the requirements for the course at NIIT University, Neemrana, embodies authentic and faithful record of original research carried out by Antra Tripathi, Debanshu Majumdar, Godavarthi Siddharth, Hitesh Yadav, Mukul Gupta, student/s of B Tech Computer Science at NIIT University, Neemrana. She /He has worked under our supervision and that the matter embodied in this project work has not been submitted, in part or full, as a project report for any course of NIIT University, Neemrana or any other university.*

Name and Title of the Mentor

Mr. Vikas Malviya

Assistant Professor

Computer Science Engineering

LIST OF FIGURES

LIST OF TABLES

CONTENTS

Rational of Work

Hadoop has gained its popularity due to its ability of storing, analyzing and accessing large amount of data, quickly and cost effectively through clusters of commodity hardware. But, No one uses kernel alone. “Hadoop” is taken to be a combination of HDFS and MapReduce. To complement the Hadoop modules there are also a variety of other projects that provide specialized services and are broadly used to make Hadoop laymen accessible and more usable, collectively known as Hadoop Ecosystem. All the components of the Hadoop ecosystem, as explicit entities are evident to address particular needs. Recent Hadoop ecosystem consists of different level layers, each layer performing different kind of tasks like storing your data, processing stored data, resource allocating and supporting different programming languages to develop various applications in Hadoop ecosystem.

With a rapid pace in evolution of Big Data, its processing frameworks also seem to be evolving in a full swing mode. The huge data giants on the web has adopted Apache Hadoop had to depend on the partnership of Hadoop HDFS with the resource management environment and MapReduce programming. Hadoop ecosystem has introduced a new processing model that lends itself to common big data use cases including interactive SQL over big data, machine learning at scale, and the ability to analyses big data scale graphs. Apache Hadoop is not actually single product but instead a collection of several components. When all these components are merged, it makes the Hadoop very user friendly. The Hadoop ecosystem and its commercial distributions continue to evolve, with new or improved technologies and tools emerging all the time.

Hadoop reduces time requirement, provides reliability, scalability and protection from node failure which are handled within the system itself simply utilizing commodity hardware. This project is intended to optimize the current Hadoop implementation on low energy consuming android devices. Since new android devices are becoming powerful with its dual, quad or octa core processors with 1 to 2 GB ram becoming common practice in mobile devices. They utilized this mobile power for industrial purposes so that they can save cost on even commodity hardware. Big data is a big thing today. Analyzing and processing this big data to produce future predictions in order to support automated decision-making system is a main task Hadoop can do with this application they intend to bring that productivity on android platform. With this application it becomes possible to carry the Big Data on android phones.

The main need of this project is to process this Big Data on Android platform. Android Hadoop will provide a mobile Hadoop Environment with reduction in hardware cost. So, it is energy efficient i.e. Quality of hardware remains same, hardware cost is reduced. Increasing mobility power of different devices encourages to think about the processing power. This lead, to thinking of providing mobility to existing workforce environment. Hadoop is chosen to Optimized Map reduce Implementation on Android Platform mobility target for transferring desktop workforce into a mobile workforce. Problem definition defines to provide an application which will allow porting of Hadoop to android platform. They have ported that capability to an android device.

With the use of Android Hadoop, they have provided a limited installation of Linux, just to support Hadoop installation. With its distributed processing, to understand this "Hadoop", they needed an efficient system which can teach and train the working of hadoop to students and learners without investing in formation of large-scale clusters. Small scale industries also cannot afford to have built large scale clusters. Today mobile and mobile based applications have become a part of our day to day life. With the revolution in mobile computing many great features and technologies were added to the field and mobiles got smaller, faster and better as the decade passed. It gave rise to the introduction of new mobile based operating systems where the programmers where presented with open source operating system named Android.

Android and Hadoop are two technologies which we decided to select to go further with our research and development of our project. These individual technologies solve many real time problems. So how about combining them? This was the question which interest us during this research work and also if we look in future development of technology. Could it be possible to run Hadoop on a cluster of mobile phones? This can be considered as the future scope of Hadoop. Recently a team at Google has pushed Hadoop to the limits by creating a cluster whose size is on the order of 1,00,000 nodes, running on the recently released Nexus One mobile phone hardware, powered by Android. By pushing computation out to these devices, the Nexus.

So, with this thinking of porting Hadoop on Android makes a lot of sense if one looks a step for the future technology. Mobile devices are getting more and more powerful and advance with octa core devices just around the corner. Also, modern phones or tablets have lots of GBs of RAM., preferably, 3-4GB with GPU & ARM powered devices consume less power than comparable Intel based devices. This topic “Kernel Level Implementation of Hadoop” is mostly theoretical and not much knowledge can be gained about it. Kernel Level Implementation of Hadoop is almost next to impossible till now. During the course of this project we will be finding possibility of implementing Hadoop on Kernel and the difficulties faced in doing so

Review of Literature

A literature review is a text of a scholarly paper, which includes the current knowledge including substantive findings, as well as theoretical and methodological contributions to a particular topic. Our R&D topic i.e. “Kernel Level Implementation of Hadoop” is very current and ongoing topic so not many researches can be found related to this topic. Research Paper and scholarly articles which we have reviewed for this project are:

- **Hadoop Ecosystem: An Introduction (**International Journal of Science and Research (IJSR))
- **XConveryer: Guarantee Hadoop Throughput via Lightweight OS-level Virtualization (**2009 Eighth International Conference on Grid and Cooperative Computing)
- **Hadoop MapReduce for Mobile Clouds. (**IEEE transactions on cloud computing, vol. 3, no. 1, January 2014)
- **Accelerating Machine Learning Kernel in Hadoop Using FPGAs. (Department** of Electrical and Computer Engineering, George Mason University)
- **Migration of Hadoop To Android Platform Using ‘Chroot’. (IJIRCT1201105** International Journal of Innovative Research and Creative Technology.)
- **Virtual Hadoop: The Study and Implementation of Hadoop in Virtual Environment using Cloud Stack KVM (International** Journal of Engineering Development and Research)

1. Hadoop Ecosystem: An Introduction

International Journal of Science and Research (IJSR)
Sneha Mehta1, Viral Mehta2
International Institute of Information Technology, Department of Information Technology

Hadoop has gained its popularity due to its ability of storing, analyzing and accessing large amount of data, quickly and cost effectively through clusters of commodity hardware. But, No one uses kernel alone. “Hadoop” is taken to be a combination of HDFS and MapReduce. To complement the Hadoop modules there are also a variety of other projects that provide specialized services and are broadly used to make Hadoop laymen accessible and more usable, collectively known as Hadoop Ecosystem. All the components of the Hadoop ecosystem, as explicit entities are evident to address particular needs. Recent Hadoop ecosystem consists of different level layers, each layer performing different kind of tasks like storing your data, processing stored data, resource allocating and supporting different programming languages to develop various applications in Hadoop ecosystem.

The holistic view of Hadoop architecture gives prominence to Hadoop common, Hadoop YARN, Hadoop Distributed File Systems (HDFS) and Hadoop MapReduce of the Hadoop Ecosystem. Hadoop common provides all Java libraries, utilities, OS level abstraction, necessary Java files and script to run Hadoop, while Hadoop YARN is a framework for job scheduling and cluster resource management. HDFS in Hadoop architecture provides high throughput access to application data and Hadoop MapReduce provides YARN based parallel processing of large data sets.

Data Storage is the layer where the data is stored in a distributed file system; consist of HDFS and HBase Column DB Storage. HBase is scalable, distributed database that supports structured data storage for large tables. When data is pushed to HDFS, it automatically splits up into multiple blocks and stores/replicates the data thus ensuring high availability and fault tolerance. HDFS comprises of 2 important components (Fig. 2) – Name Node and Data Node. HDFS operates on a Master- Slave architecture model where the Name Node acts as the master node for keeping a track of the storage cluster and the Data Node acts as a slave node summing up to the various systems within a Hadoop cluster. MapReduce [8] is a software framework for distributed processing of large data sets that serves as the compute layer of Hadoop which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. The compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System are running on the same set of nodes. This configuration allows the framework to effectively schedule task on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster.

With a rapid pace in evolution of Big Data, its processing frameworks also seem to be evolving in a full swing mode. The huge data giants on the web has adopted Apache Hadoop had to depend on the partnership of Hadoop HDFS with the resource management environment and MapReduce programming. Hadoop ecosystem has introduced a new processing model that lends itself to common big data use cases including interactive SQL over big data, machine learning at scale, and the ability to analyse big data scale graphs. Apache Hadoop is not actually single product but instead a collection of several components. When all these components are merged, it makes the Hadoop very user friendly. The Hadoop ecosystem and its commercial distributions continue to evolve, with new or improved technologies and tools emerging all the time.

2. XConveryer: Guarantee Hadoop Throughput via Lightweight OS-level Virtualization

2009 Eighth International Conference on Grid and Cooperative Computing
An Qin, Dandan Tu , Chengchun Shu, Chang Gao,
Institute of Computing Technology, Chinese Academy of Sciences, Beijing China

---

Large-scale data parallel applications such as web indexing, data mining demand plenty of computing and storage resources. As a widely adopted solution, hadoop partitions and distributes the large datasets into chunks across multiple nodes in clusters to process in parallel. For a single cluster node, typically there are several concurrently running applications, sharing and competing system CPU, memory, disk and network bandwidth. The competition will cause unfairness for some jobs and extend their response time. Particularly, some jobs have to be cancelled and rescheduled because of resource starvation. In this paper, we present a solution to reduce resource challenge. The XConveryer, as one of key component, is designed based on kernel-level virtualization facilities, to isolate Hadoop jobs and to guarantee their resource share. Built on XConveryer, Hadoop can enforce scheduling polities down to Operating System, that more fine-granular control can be achieved. The evaluation shows the XConveryer can encapsulate Hadoop jobs executed in isolated containers and guarantee strict control their system resources usages.

XConveryer, a resource manager for I/O bandwidth enforcement, to reduce job resource challenge in Hadoop based on kernel cgroups [26] and namespace mechanisms [27]. The cgroups mechanism in kernel is exploited to manage resource allocation and scheduling after job has been run as processes in Operating System. Furthermore, kernel namespace mechanism is also adopted to isolate process space from malignant challenge. Based on the support from kernel, the Hadoop is patched to guide job running and scheduling under the control of kernel process group to reduce the challenge on resources. The rest of the paper is organized as follows: we first describes the background of several technologies related to our methods including OS-level process isolation and enforcement mechanisms in section II. Hadoop's job scheduling is also presented in section II. Then, we detail our techniques for confining job running by grouping hadoop tasks, which also guarantee scheduling policies of hadoop down to kernel to efficiently utilize system resources.

On XConveryer, Hadoop can be on the way to guarantee scalable performance regardless the number of submitted jobs. Our method features with OS kernel resource control and policy enforcement in a lightweight way. The performance evaluation shows the method can put the tasks executed in isolation and strict control their system resources usages. The evaluation of total implementation is weak because patching and enhancing Hadoop is not accomplished. To achieve efficient job execution in a scalable manner, we still need to exploit scheduling feature in Hadoop to further enhance resource utilization and to shorten job response time.

3. Hadoop MapReduce for Mobile Clouds

IEEE transactions on cloud computing, vol. 3, no. 1, January 2014
Johnu George, Chien-An Chen, Radu Stoleru, Member, IEEE, Geoffrey G. Xie Member, IEEE

Building Hadoop on a mobile network enables the devices to run data intensive computing applications without direct knowledge of underlying distributed systems complexities. Applications have severe energy and reliability constraints (e.g., caused by unexpected device failures or topology changes in a dynamic network). As mobile devices are more susceptible to unauthorized access, when compared to traditional servers, security is also a concern for sensitive data. Addresses issues for big data processing in mobile clouds. The implementation addresses all constraints in processing large amounts of data in mobile clouds. Thus, the system provided is a viable solution to meet the growing demands of data processing in a mobile environment. Several research studies that attempted to bring MapReduce framework to the heterogeneous cluster of devices, due to its simplicity and powerful abstractions.

Marinelli introduced the Hadoop based platform Hyrax for cloud computing on smartphones. Hadoop Task Tracker and Data Node processes were ported on Android mobile phones, while a single instance of Name Node and Job Tracker were run in a traditional server. P2P-MapReduce describes a prototype implementation of a MapReduce framework which uses a peer-to-peer model for parallel data processing in dynamic cloud topologies. Server-client model-based MapReduce system was proposed over a cluster of mobile devices where the mobile client implements MapReduce logic to retrieve work and produce results from the master node. MapReduce is a scalable parallel processing framework that runs on HDFS.

It basically refers to the two tasks performed – Map task and Reduce task. Map task takes the input data set and produces a set of intermediate key value pairs which are sorted and partitioned by reducer. Then its output is passed to the Reducers to produce the final output. There are two modules in MapReduce – Job Tracker and Task Tracker. Job Tracker is the MapReduce master daemon that accepts the user jobs and splits them into multiple tasks. Assigns slave nodes in thee cluster called the task trackers. Task Trackers are the processing nodes in the cluster that runs the tasks.

HDFS is designed to store very large datasets. In HDFS each file is splits into blocks, each block is replicated to several devices across the cluster. The two modules in HDFS layer are Name Node and Data Node. Name Node holds the metadata information about the stored files, Data Node are the system slave nodes which are the storage nodes in the cluster. In traditional MDFS architecture the files were stored at the same level in the file system without the use of directories. The user invokes the object to interact with the file system. In order to switch from HDFS to MDFS. MDFS framework consists of 18,365 lines of Java code, which might be a reason why it cannot be implemented in the kernel.

4. Accelerating Machine Learning Kernel in Hadoop Using FPGAs

Katayoun Neshatpour, Maria Malik, and Houman Homayoun
Department of Electrical and Computer Engineering, George Mason University

This paper analyses how offloading computational intensive kernels of machine learning algorithms to a heterogeneous CPU+FPGA platform enhances the performance. They used the latest Xilinx Zynq boards for implementation and result analysis. Furthermore, they performed a comprehensive analysis of communication and computation overheads such as data I/O movements and calling several standard libraries that cannot be offloaded to the accelerator to understand how the speedup of each application will contribute to its overall execution in an end-to-end Hadoop MapReduce environment. Emerging big data analytics applications require a significant amount of server computational power. Big data analytics applications heavily rely on big-data-specific deep machine learning and data mining algorithms and are running complex database software stack. This set of characteristics is necessitating a change in the direction of server-class microarchitecture to improve their computational and memory efficiency.

The Hardware acceleration through specialization has received renewed interest in recent years, mainly due to the dark silicon challenge. Heterogeneous architectures comprise different types of cores, domain-specific accelerators, as well as programmable fabrics (FPGA) to synthesize a custom accelerator. Tight integration between the general-purpose processor and the programmable logic can provide enormous opportunities to add any type of custom-designed accelerator. An example of such architecture is ZYNQ (ARM+FPGA). To address the computing requirements of big data, and based on the benchmarking and characterization results, they envisioned a heterogeneous architecture for next big data server platforms that leverage the power of FPGA to build custom accelerators. Our baseline architecture consists of a standard Hadoop platform which is extended with FPGA hardware for acceleration. To evaluate how hardware acceleration can address performance demands of big data analytics applications, we study various data mining and machine learning algorithms to find the CPU-intensive and time-consuming kernels to offload to FPGA. They assumed the applications are fully known, therefore they could find the best possible application-to-core+FPGA match. They calculated the acceleration gained through the hardware-software co-design process and evaluate how it will speed up the Hadoop end-to-end system.

To significantly improve performance of processing big data analytics applications, a heterogeneous architecture that integrates general-purpose CPUs with dedicated FPGA accelerators was studied. Full

Hadoop MapReduce profiling was used to find the hot regions of several widely used machine learning and data mining applications. The hardware equivalents of hot regions were developed using high level synthesis tools. With hardware-software co-design, we offloaded the hot regions to the hardware to find the design with the highest speed-up. A comprehensive analysis of communication and computation overheads was used to understand how the speedup of each application will contribute to its overall execution in an end-to-end Hadoop MapReduce implementation. Sensitivity analysis was performed on the size of data input splits to understand the speedup sensitivity to size of data. The results showed that a kernel speedup of upto ×94 with hardware-software co-design can be achieved. This results in ×2.69 speedup in an end-to-end Hadoop MapReduce environment considering various data transfer and communication overheads.

Future work was focused on detailed implementation of the kernels including the hardware-software co-design of individual map and reduce functions for each application. Comprehensive and experimental exploration of other design aspects, including number of mapper slots, type of master core and hotspot analysis of the Hadoop environment for a wide range of input split sizes will allow the development of a more generalized and accurate model to better understand the performance gains of using FPGA to accelerate big data analytics applications.

5. Migration of Hadoop To Android Platform Using 'Chroot'

IJIRCT1201105 *International Journal of Innovative Research and Creative Technology*
Namrata B Bothe, Snehal S Karale,, Anagha N Mate , Nayan D Kumbhar
Information Technology Rmd Sinhgad School of Engineering Pune, India

---

Google`s Map Reduce is one of the largest popular algorithms for big data processing. Moreover, big data can be termed as large number of data to be process within short period of time. Its implementation - Hadoop reduces time requirement, provides reliability, scalability and protection from node failure which are handled within the system itself simply utilizing commodity hardware. This project is intended to optimize the current Hadoop implementation on low energy consuming android devices. Since new android devices are becoming powerful with its dual, quad or octa core processors with 1 to 2 GB ram becoming common practice in mobile devices. They utilized this mobile power for industrial purposes so that they can save cost on even commodity hardware. Big data is a big thing today. Analyzing and processing this big data to produce future predictions in order to support automated decision-making system is a main task Hadoop can do with this application they intend to bring that productivity on android platform. With this application it becomes possible to carry the Big Data on android phones.

The main need of this project is to process this Big Data on Android platform. Android Hadoop will provide a mobile Hadoop Environment with reduction in hardware cost. So, it is energy efficient i.e. Quality of hardware remains same, hardware cost is reduced. Increasing mobility power of different devices encourages to think about the processing power. This lead, to thinking of providing mobility to existing workforce environment. Hadoop is chosen to Optimized Map reduce Implementation on Android Platform mobility target for transferring desktop workforce into a mobile workforce. Problem definition defines to provide an application which will allow porting of Hadoop to android platform. They have ported that capability to an android device. With the use of Android Hadoop, they have provided a limited installation of Linux, just to support Hadoop installation. With its distributed processing, to understand this "Hadoop", they needed an efficient system which can teach and train the working of hadoop to students and learners without investing in formation of large-scale clusters. Small scale industries also cannot afford to have built large scale clusters. Today mobile and mobile based applications have become a part of our day to day life. With the revolution in mobile computing many great features and technologies were added to the field and mobiles got smaller, faster and better as the decade passed. It gave rise to the introduction of new mobile based operating systems where the programmers where presented with open source operating system named Android.

HDFS cluster is a combination of a single Name Node which manages the cluster metadata and Data Node that stores the data. Files and directories are represented on the Name Node. The file contents are split into large blocks. Hadoop implements a Master/Slave architecture. The Hadoop Distributed File System (HDFS) is a distributed file system designed to keep running on product equipment. It has numerous similitudes with existing distributed file systems. However, the differences from other disseminated document frameworks are huge. HDFS is highly fault-tolerant and is designed to be deployed on minimal effort equipment. HDFS gives high throughput access to application data and is suitable for applications that have large data sets. 'Chroot' changes root environment. It mounts a specific. Registry as root and make a sub-process tree for specific system. Processes running under this environment, typically can`t access the registries outside the 'chroot'ed catalog. This behavior of the newly created root directory is known as ‘chroot jail’. By and large chroot is utilized as testing and improvement environment, to test dependency control, to test the capability of software for particular device, recover system etc. Andro-Hadoop will provide a mobile Hadoop Environment with reduction in hardware cost. The system will be energy efficient. With the help of this system it becomes possible to work with Hadoop on Android platform. This application helps to bring the productivity platform on android devices to access big data using MapReduce algorithm. They have successfully implemented Hadoop on android using ‘chroot’ method.

6. Virtual Hadoop: The Study and Implementation of Hadoop in Virtual Environment using CloudStack KVM

International Journal of Engineering Development and Research
Arun S Devadiga, Shalini P.R, Aditya Kumar Sinha
Computer Science and Engineering, NMAM Institute of Technology

The paper center around utilizing Hadoop apparatus in virtual condition utilizing Cloud Stack KVM for taking care of enormous information related issues. Hadoop is an apache apparatus which is utilized to process a gigantic measure of information simultaneously. Since, Hadoop is an open source application; it has been utilized all through the business. Utilizing Hadoop in virtual condition gives an approach to parallel registering, and aides in sending and administration of uses for conveyed figuring. MapReduce segment of Hadoop is utilized here for huge scale parallel applications and by means of virtualization we can enhance the current figuring assets, which is basic in distributed computing field. By sending virtual machine administration of Hadoop we can have successful administration of asset for substantial number of hub regarding setup, organization and asset usage.

At present, there are numerous open source answers for building cloud condition. One among them is Cloud Stack, which is an open source cloud stage that permits fabricating all sort of cloud condition including private, open and crossover cloud. KVM virtual machine gives the virtual condition. Consequently, this article clarifies the work engaged with coordinating the Hadoop, Cloud Stack and KVM. This joining will bring about virtual Hadoop which will enable client to process gigantic measure of information simultaneously in virtual condition, with effective utilization of assets.

The proposed demonstrate primarily centers around conveying Hadoop in Cloud Stack KVM, which brings about virtual Hadoop. Afterward, various MapReduce projects and datasets are given as the contribution to the virtual Hadoop made. The execution of the Hadoop in virtual condition is contrasted and the Hadoop in Physical condition. It demonstrates that Hadoop in virtual condition delivers preferred execution over the Hadoop in physical condition.

To convey Hadoop in Cloud Stack KVM following advances should be taken after:

1. Before introducing Hadoop Java JDK should be introduced, since Hadoop utilizes Java as the programming model. So, the most recent rendition of java can be downloaded from `http://www.oracle.com/, selected jdk-7u4-linux-i586.tar.gz.`

2. enter the Java document and set all nature factors.

3. Download Hadoop from the http://archive.apache.org/and unload and introduce Hadoop

```
$ sudo apt-get install openssh-server.
```

4. Arrange Hadoop by altering the setup document hdfs-site.xml, mapred-site.xml and center site.xml. Likewise introduce Hadoop overshadow module and change the obscuration java viewpoint to MapReduce condition.

The last conclusion is that virtual Hadoop has marginally higher yet relatively comparable execution time to execute the MapReduce program than the Physical Hadoop. In any case, the points of interest are that the administration is simpler, completely using the figuring assets, make Hadoop more dependable and spare power. Consequently, this favorable position demonstrates that virtual Hadoop utilizing Cloud Stack as higher proficiency contrasted with the Physical Hadoop.

Objective

Hadoop is an Apache open source framework written in Java that allows distributed processing of large datasets across clusters of computers using simple programming models in a distributed computing environment. The kernel is the essential center of a computer operating system. It acts as an interface between the user applications and the hardware. The sole aim of the kernel is to manage the communication between the software (user level applications) and the hardware (CPU, disk memory etc.). Hadoop works on application layer so the motive of this research is to find:

- Why it cannot be implemented at kernel level
- What are the difficulties faced?
- And try to implement it on android platform.
- Finding solution to the encounter problems if possible.

Methodology

The Research and Development of this project will be done as Applied Research as this project is directed toward gaining knowledge or understanding necessary for determining the means by which recognized and specific needs can be met. This topic “Kernel Level Implementation of Hadoop” is mostly theoretical and not much knowledge can be gained about it. Kernel Level Implementation of Hadoop is almost next to impossible till now. During the course of this project we will be finding possibility of implementing Hadoop on Kernel and the difficulties faced in doing so.

After researching for a while and reading some research papers and scholarly articles on possibilities of “Kernel Level Implementation of Hadoop” we came across some possible proposed methods which can be tested to check the possibility of implementing hadoop on kernel but none of them till now are successfully implemented. Listed below are some of the possible ways proposed by researchers:

- Virtual Hadoop: Implementation of Hadoop in Virtual Environment using Cloud Stack KVM
- Offloading computational intensive kernels of machine learning algorithms to a heterogeneous CPU+FPGA platform enhances the performance.
- XConveryer: Guarantee Hadoop Throughput via Lightweight OS-level Virtualization
- Hadoop MapReduce for Mobile Clouds
- Migration of Hadoop To Android Platform Using ‘Chroot’

After considering the feasibility of the above proposed hypothesis Implementation of Hadoop on Android kernel was the most feasible way and we can easily attempt to experiment with it, were as the reasons to reject others was that either their cost of implementation was very high or the required resources were not available. Android and Hadoop are two technologies which we decided to select to go further with our research and development of our project. These individual technologies solve many real time problems. So how about combining them? This was the question which interest us during this research work and also if we look in future development of technology. Could it be possible to run Hadoop on a cluster of mobile phones? This can be considered as the future scope of Hadoop. Recently a team at Google has pushed Hadoop to the limits by creating a cluster whose size is on the order of

1,00,000 nodes, running on the recently released Nexus One mobile phone hardware, powered by Android. By pushing computation out to these devices, the Nexus.

So, with this thinking of porting Hadoop on Android makes a lot of sense if one looks a step for the future technology. Mobile devices are getting more and more powerful and advance with octa core devices just around the corner. Also, modern phones or tablets have lots of GBs of RAM., preferably, 3-4GB with GPU & ARM powered devices consume less power than comparable Intel based devices.

With Hadoop`s power we can bring great productivity on android platforms and in attempt of so we had followed various methods listed below:

- Running Hadoop framework on Android platform through different methods
    i. - ***"Chroot method"***
    ii. -Virtual Workspace using ***"Samsung knox"***
    iii. -Using **"*Android Studio*"** and its "***Android Emulator"***
- Implementing Hadoop on Raspberry Pi 2
- THREADS – An alternative to HADOOP on MDFS

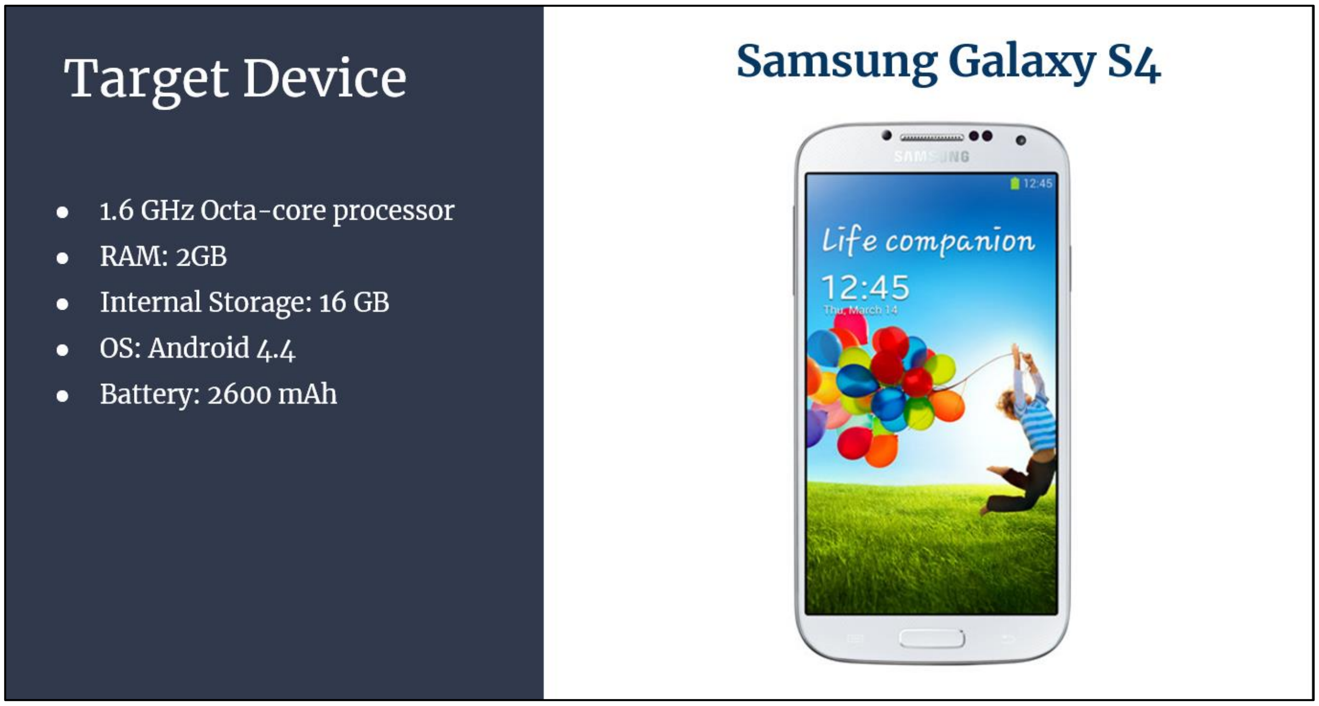
Implementation and Results

As mentioned before about the different approaches which we have followed, this section contains the complete description of the methods adopted during research and development of our project work.

1. Hadoop on Android Devices

Map reduce Implementation on Android Platform mobility target for transferring desktop workforce into a mobile workforce. Problem definition defines to provide an application which will allow porting of Hadoop to android platform. Considering this, leads to solving following problems: Build an interface to install Linux. Build or use existing VNC app to interact with GUI. Install and configure hadoop for running pseudo-distributed mode. Run sample test program Hadoop. Today mobile and mobile based applications have become a part of our day to day life. With the revolution in mobile computing many great features and technologies were added to the field and mobiles got smaller, faster and better as the decade passed. It gave rise to the introduction of new mobile based operating systems where the programmers where presented with open source operating system named Android.

Our Targeted Device:



Target Device

- 1.6 GHz Octa-core processor
- RAM: 2GB
- Internal Storage: 16 GB
- OS: Android 4.4
- Battery: 2600 mAh

Samsung Galaxy S4

SAMSUNG
12:45
Life companion
12:45
Thu, March 14

Figure 1

i. *- “Chroot method”*

What is 'chroot'?

‘Chroot’ changes root environment. It mounts a specific. Registry as root and make a sub-process tree for specific system. Processes running under this environment, typically can`t access the registries outside the 'chroot'ed catalogue. This behaviour of the newly created root directory is known as ‘chroot jail’. We can see such implementation with Debian Runner also.

Following are the requirements needs to be fulfilled, in order to implement this method.

Requirements:

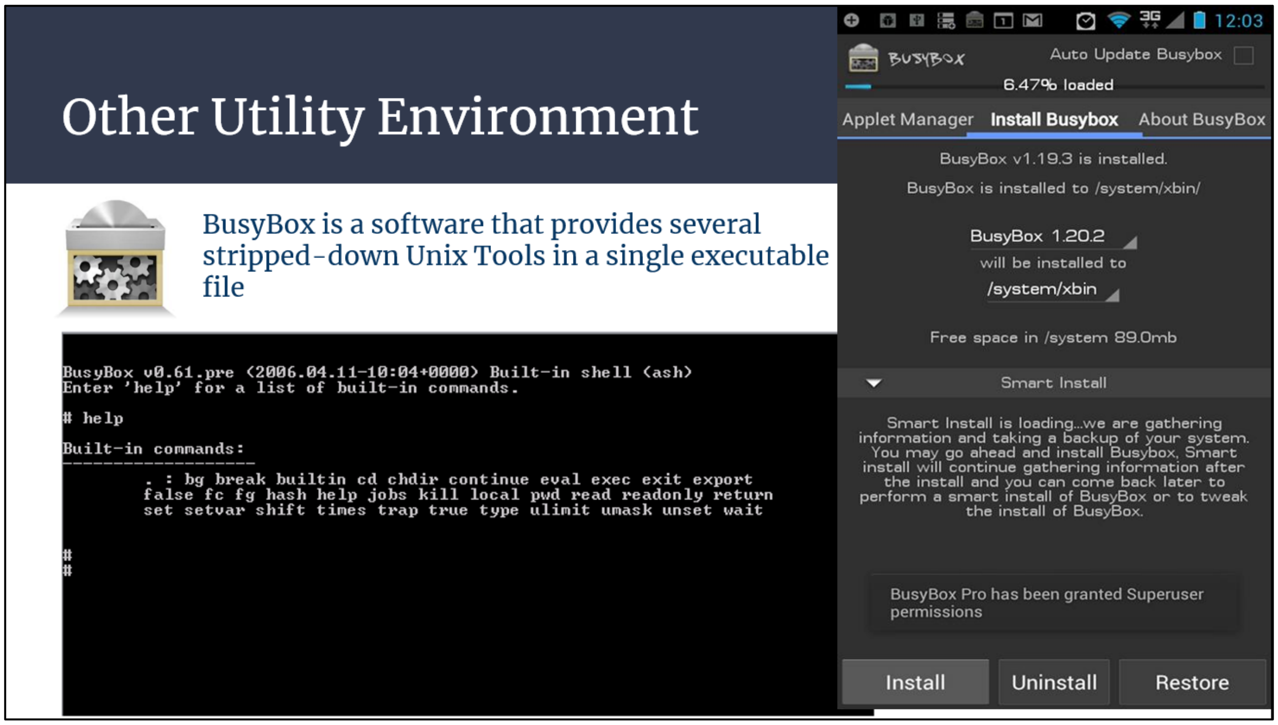
- Root Privileges.
- ARM supported Linux distribution
- Kernel modules loaded.
- Swap enabled and Looping enabled device
- Internet connection forwarding.
- High Speed Internet connection to download and install additional components.

By and large chroot is utilized as testing and improvement environment, to test dependency control, to test the capability of software for particular device, recover system etc.

Steps to actualize chroot:

There are two approaches to execute on android device.

Using terminal emulator application and busy box from Google play store and performing normal chroot on android. One can perform it on Linux by mounting the new Linux framework index or picture.



Other Utility Environment

BusyBox is a software that provides several stripped-down Unix Tools in a single executable file

```
BusyBox v0.61.pre (2006.04.11-10:04+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# help

Built-in commands:
-------------------
        . : bg break builtin cd chdir continue eval exec exit export
        false fc fg hash help jobs kill local pwd read readonly return
        set setvar shift times trap true type ulimit umask unset wait

#
#
```

BusyBox v1.19.3 is installed.
BusyBox is installed to /system/xbin/
BusyBox 1.20.2 will be installed to /system/xbin
Free space in /system 89.0mb

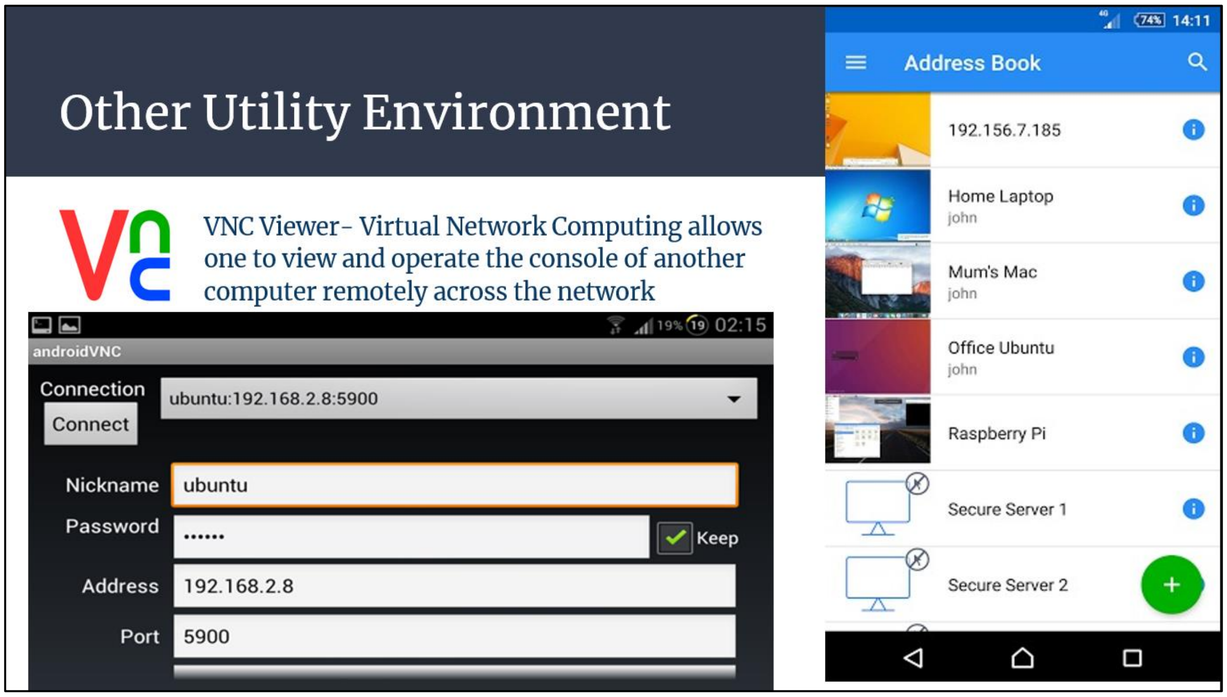
Smart Install

Smart Install is loading...we are gathering information and taking a backup of your system. You may go ahead and install Busybox, Smart install will continue gathering information after the install and you can come back later to perform a smart install of BusyBox or to tweak the install of BusyBox.

BusyBox Pro has been granted Superuser permissions

Figure 2

After installing perfect Linux for your device(x86/armv5/armv71), introduce VNC viewer app for remote login interface to same Linux. VNC viewer typically uses 5900port number.



Other Utility Environment

VNC Viewer- Virtual Network Computing allows one to view and operate the console of another computer remotely across the network

Connection	ubuntu:192.168.2.8:5900
Nickname	ubuntu
Password	•••••• ✓ Keep
Address	192.168.2.8
Port	5900

Figure3

Implementation:

All present execution of Linux is accessible for different platforms. However, Hadoops performance on various platform varies with processor architecture [13], [14]. Comparing with different architectures, and Hadoop`s basic need of commodity hardware, low power expending arm gadgets ought to be perfect for Hadoop.

Setting up environment:

To install Hadoop in such 'chroot'-ed environment we have to perform the accompanying steps.

- Update Linux conveyance.
- Install java for your gadget design/abi.
- Download most recent Hadoop from apache storehouse.
- Navigate to home folder of user and add environmental variables.

After this setup:

We tried easy implementation of Hadoop by running simple map reduce program on this device.

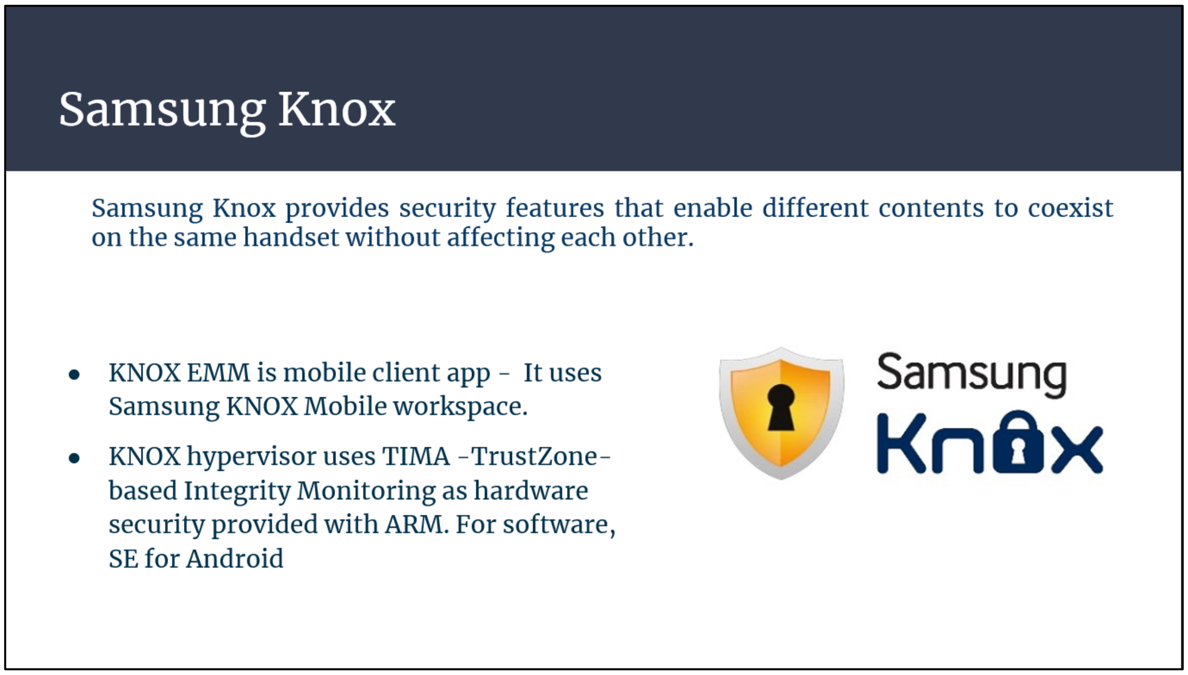
Result:

- Normal Linux command were executed. (Successful Attempt)
- 100% job execution could not be achieved, and the program terminated in between.

ii. -Virtual Workspace using *"Samsung knox"*

"According to the strict computing definition of virtualization means "does not exist physically but rather simulated by software."

This is the technique which reduces hardware cost to run different operating systems with same hardware. It also provides security in terms of access limitations outside the defined folder. For android, there are multiple virtualization methods: We are using one of the most successful method i.e. Samsung Knox (for our targeted device).



Samsung Knox

Samsung Knox provides security features that enable different contents to coexist on the same handset without affecting each other.

- KNOX EMM is mobile client app - It uses Samsung KNOX Mobile workspace.
- KNOX hypervisor uses TIMA -TrustZone-based Integrity Monitoring as hardware security provided with ARM. For software, SE for Android

Samsung
Knox

Figure 4

Steps:

- Creating virtual workspace in Samsung Knox.
- We tried running a simple android application (.apk file) of wordcount problem.

Result:

- Unexpected device failure i.e. device is not able to switch ON.
- It is trapped in “boot loop”. (Malfunctioning of the device)

iii. -Using “*Android Studio*” and its “*Android Emulator”*

Android Studio is the official integrated development environment for Google's Android operating system, used for Android development. Android emulator is an Android Virtual Device (AVD) that represents a specific Android device. We can use Android emulator as a target platform to run and test our Android applications on your PC.

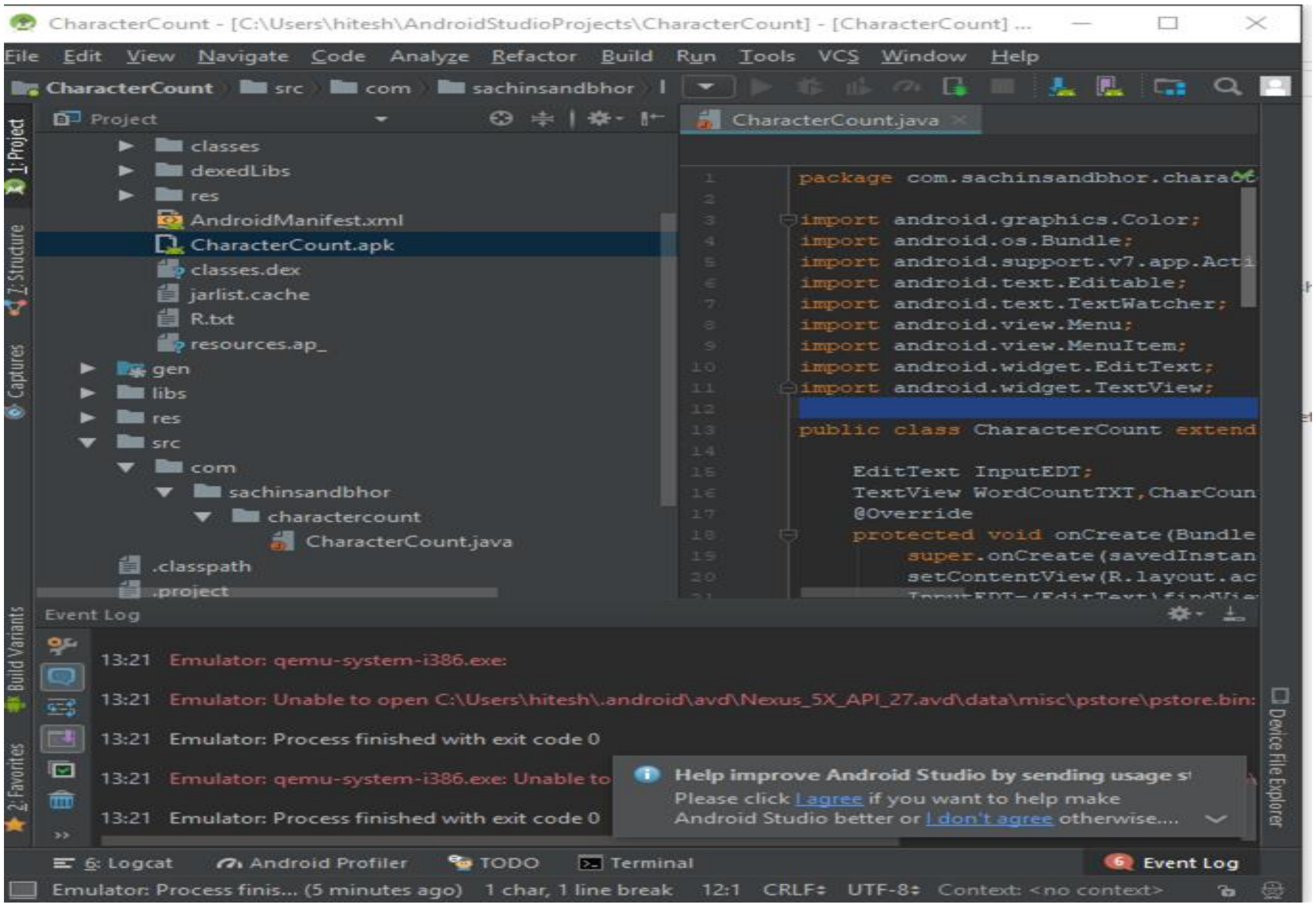
Steps:

- Emulator: Google Nexus 5X API 27.
- Create an android project.

- Add specific libraries as per requirement
- Create a java file and code it for a simple word count program
- Open .xml file and write the specific code for it
- Executing the code on Android Studio using above specified emulator

Results:

- Successful creation of an Android apk. Apk is successfully running
- Able to calculate correct number of words entered in the area
- Able to calculate correct number of characters entered in the area
- Certain files are missing. Suppression of some exceptions
- We are not able to include Apache HDFS dependency.
- Hence the code cannot be executed completely



```
package com.sachinsandbhor.charact

import android.graphics.Color;
import android.os.Bundle;
import android.support.v7.app.Acti
import android.text.Editable;
import android.text.TextWatcher;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.EditText;
import android.widget.TextView;

public class CharacterCount extend

    EditText InputEDT;
    TextView WordCountTXT,CharCoun
    @Override
    protected void onCreate(Bundle
        super.onCreate(savedInstan
        setContentView(R.layout.ac
```

```
13:21  Emulator: qemu-system-i386.exe:
13:21  Emulator: Unable to open C:\Users\hitesh\.android\avd\Nexus_5X_API_27.avd\data\misc\pstore\pstore.bin:
13:21  Emulator: Process finished with exit code 0
13:21  Emulator: qemu-system-i386.exe: Unable to
13:21  Emulator: Process finished with exit code 0
```

Help improve Android Studio by sending usage st
Please click I agree if you want to help make Android Studio better or I don't agree otherwise....

Emulator: Process finis... (5 minutes ago) 1 char, 1 line break 12:1 CRLF: UTF-8: Context: <no context>

Figure 5

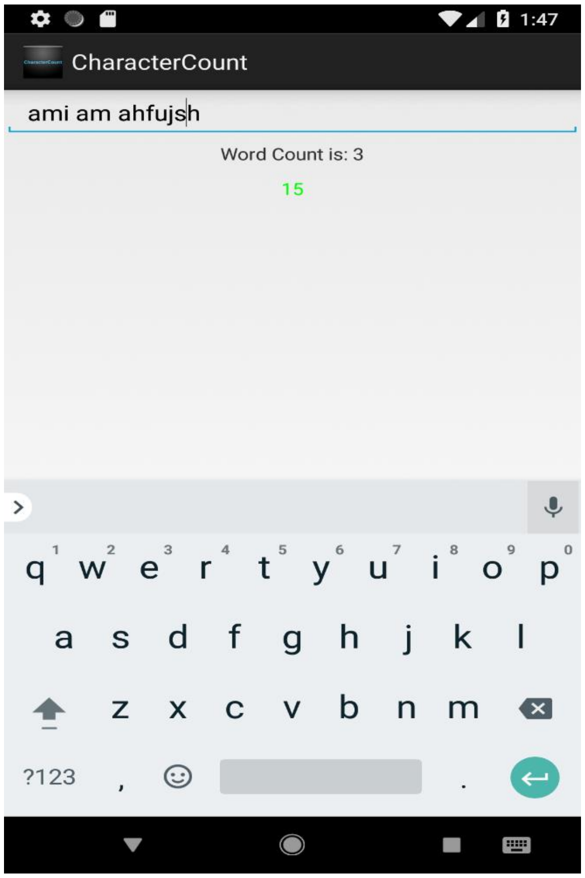


Figure 6

From the screenshot provided we can see that:

1. A text field is taking input from the user.

2. And number of words and characters are directly displayed calculated from Map Reduce code

Problem Encountered while Trying to Implement Hadoop on Android

1. Hadoop's value is greatest for databases that are 10 to 1,000 times larger than the typical databases used by mobile applications.
2. A Hadoop job might take several minutes, even if the amount of data correlated is modest
3. **Computational Power:** Hadoop requires high network speed and RAM. (with respect to targeted device)
4. **Power Consumption:** The battery life for phones is a big concern!!!
5. If possible, we are trying to increase the computational power of our targeted device by switching it to Raspberry pi.

2. Implementing Hadoop on Raspberry Pi2

The Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote the teaching of basic computer science. feature a Broadcom system on a chip (SoC) with an integrated ARM compatible central processing unit (CPU) and on-chip graphics processing unit (GPU). Processor speed ranges from 700 MHz to 1.4 GHz for the Pi 3 Model B+; on-board memory ranges from 256 MB to 1 GB RAM. Secure Digital (SD) cards are used to store the operating system and program memory in either SDHC or MicroSD sizes. The boards have one to four USB ports. For video output, HDMI and composite video are supported, with a standard 3.5 mm phono jack for audio output. Lower-level output is provided by a number of GPIO pins which support common protocols like I^2C. The B-models have an 8P8C Ethernet port and the Pi 3 and Pi Zero W have on-board Wi-Fi 802.11n and Bluetooth. A Debian-based Linux distribution known as Raspbian is tailored for duty as an on-board operating system for Raspberry Pi.

Advantages of Using Raspberry Pi:

- Microcontroller are simple computer that are application specific and which means one program can be run at a time. Whereas Raspberry Pi is a mini-computer or a mini-CPU, which works with Linux operating system and can handle multiple programs running simultaneously. Pi is capable of doing multiple tasks at a time like a computer.

- You can get internet connectivity in microcontroller. It’s doable but not that easy. Whereas it is fairly easy to connect your raspberry pi to internet.

- You can work with many different programming languages on Raspberry Pi such as C, C++, Java, Python, Perl etc. Whereas mostly C is used for many microcontrollers.

- Operating system can be easily switched on the single Raspberry Pi board. Pi uses SD card as flash memory to install the OS, just by swapping the memory card you can switch the operating system easily.

- Raspberry Pi provides you to work on GUI mode very easily as it has HDMI port. Recent release of Pi is compatible to work on WINDOWS 10. Not completely windows 10 mode but it can be used for various IOT based projects.

- For server-based application Raspberry Pi is best suited. It can be connected via SSH and files can be transfer over FTP.

- Raspberry pi has hardware support for SPI and I2C, to enable interfacing with various other devices

Hadoop, one of the open source frameworks for processing big data, uses distributed computational model designed to be able to run on commodity hardware. The aim of this attempt was to analyze Hadoop functioning on Raspberry Pi as a commodity hardware and to determine the possibilities of kernel level implementation of Hadoop. Since we are targeting ARM devices for our research work we decided to check it for Raspberry Pi also in hope of getting any positive results. Below is the complete setup description of the Raspberry Pi with Hadoop and configuration of Data Node and Name Node.

Single node Hadoop on Raspberry Pi 2 Model B

Preliminaries: -

To get going with your single node Hadoop setup, you will need the following Raspberry Pi 2 Model B bits and pieces:

- One Raspberry Pi 2 Model B, i.e. the latest Raspberry Pi model featuring a quad core CPU with 1 GB RAM.

- 8GB microSD card with NOOBS (“New Out-Of-the-Box Software”) installer/boot loader pre-installed

- Wireless LAN USB card.

- Mini USB power supply, heat sinks and HDMI display cable.

- Optional, but recommended: A case to hold the Raspberry circuit board.

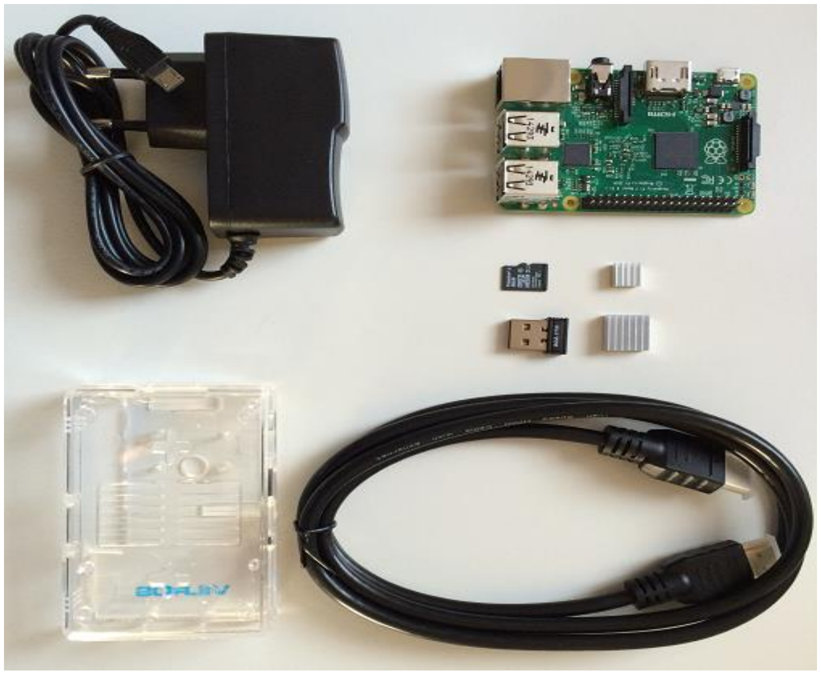


Figure 7

We intend to install the latest stable Apache Hadoop and Hive releases available from any of the Apache Software Foundation download mirror sites.

- Hadoop 2.7.2
- Hive 1.1.0
- SAP Lumira 1.23 desktop edition

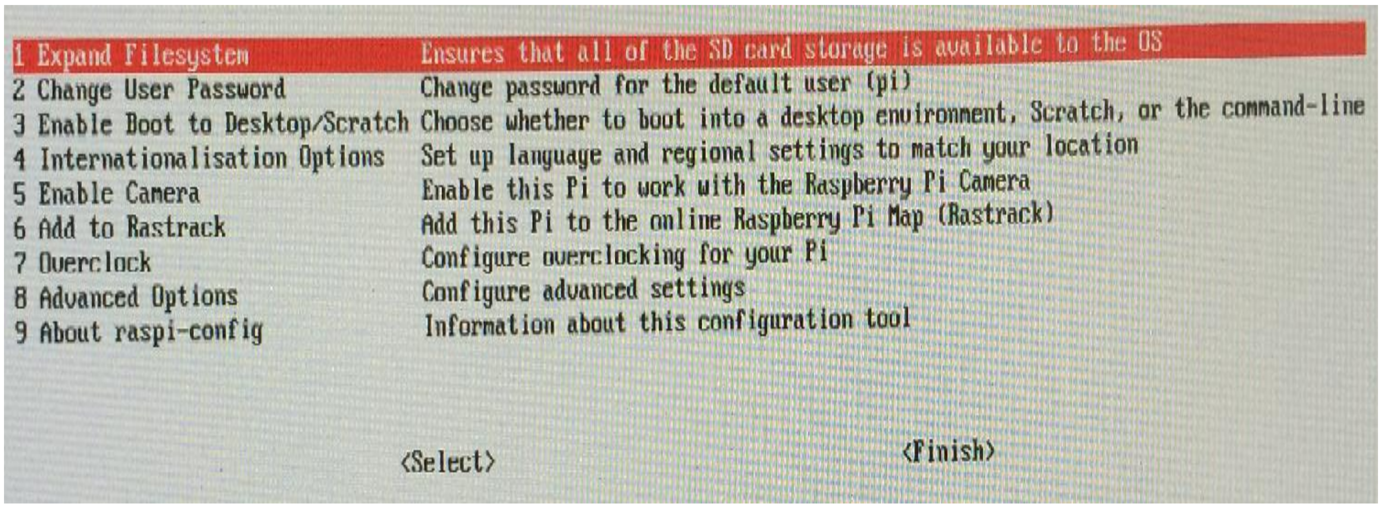
We will follow his approach for the basic Raspbian setup in this part but updated to reflect Raspberry Pi 2 Model B-specific aspects and providing some more detail on various Raspberry Pi operating system configuration steps. To keep things nice and easy, we are assuming that you will be operating the environment within a dedicated local wireless network thereby avoiding any firewall and port setting (and the Hadoop node & rack network topology) discussion. The subsequent document parts will be based on this basic setup.

Raspberry Pi setup:

Powering on your Raspberry Pi will automatically launch the pre-installed NOOBS installer on the SD card. Select "Raspbian", a Debian 7 Wheezy-based Linux distribution for ARM CPUs, from the installation options and wait for its subsequent installation procedure to complete. Once the Raspbian operating system has been installed successfully, your Raspberry Pi will reboot automatically and you will be asked to provide some basic configuration settings using raspi-config. Note that since we are assuming that you are using NOOBS, you will not need to expand your SD card storage (menu Option Expand Filesystem). NOOBS will already have done so for you. By the way, if you want or need to run NOOBS again at some point, press & hold the shift key on boot and you will be presented with the NOOBS screen.

Basic configuration:

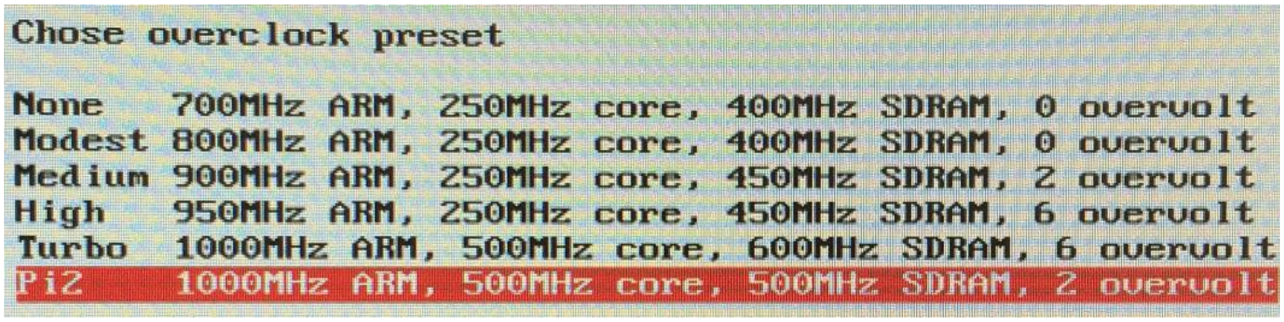
What we might want to do though is to set a new password for the default user "pi" via configuration optionChange User Password. Similarly, set our internationalisation options, as required, via optionInternationalisation Options.



```
1 Expand Filesystem              Ensures that all of the SD card storage is available to the OS
2 Change User Password           Change password for the default user (pi)
3 Enable Boot to Desktop/Scratch Choose whether to boot into a desktop environment, Scratch, or the command-line
4 Internationalisation Options   Set up language and regional settings to match your location
5 Enable Camera                  Enable this Pi to work with the Raspberry Pi Camera
6 Add to Rastrack                Add this Pi to the online Raspberry Pi Map (Rastrack)
7 Overclock                      Configure overclocking for your Pi
8 Advanced Options               Configure advanced settings
9 About raspi-config             Information about this configuration tool
```

Figure 8

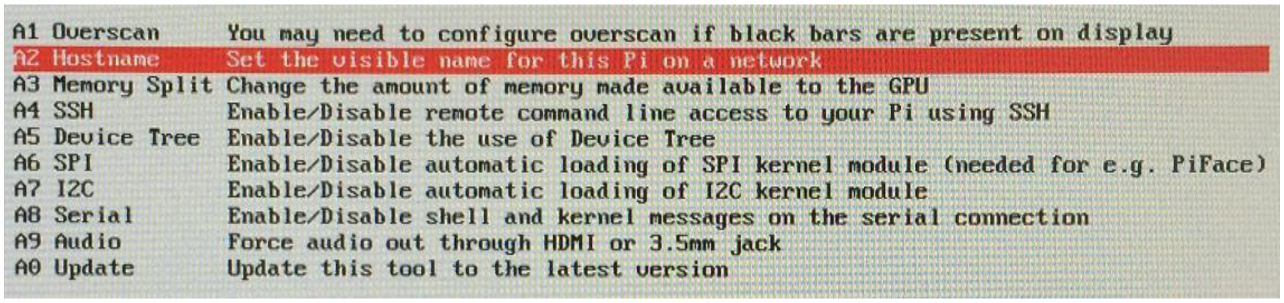
More interestingly in our context, go for menu item *Overclock* and set a CPU speed to your liking taking into account any potential implications for the power supply/consumption ("voltmodding") and the life-time of your Raspberry hardware. If we are somewhat optimistic about these things, go for the "Pi2" setting featuring 1GHz CPU and 500 MHz RAM speeds to make the single node Raspberry Pi Hadoop experience a little more enjoyable.



```
Chose overclock preset

None    700MHz ARM, 250MHz core, 400MHz SDRAM, 0 overvolt
Modest  800MHz ARM, 250MHz core, 400MHz SDRAM, 0 overvolt
Medium  900MHz ARM, 250MHz core, 450MHz SDRAM, 2 overvolt
High    950MHz ARM, 250MHz core, 450MHz SDRAM, 6 overvolt
Turbo   1000MHz ARM, 500MHz core, 600MHz SDRAM, 6 overvolt
Pi2     1000MHz ARM, 500MHz core, 500MHz SDRAM, 2 overvolt
```

Figure 9

Under Advanced Options, followed by submenu item Hostname, set the hostname of your device to "node1". Selecting Advanced Options again, followed by Memory Split, set the GPU memory to 32 MB.



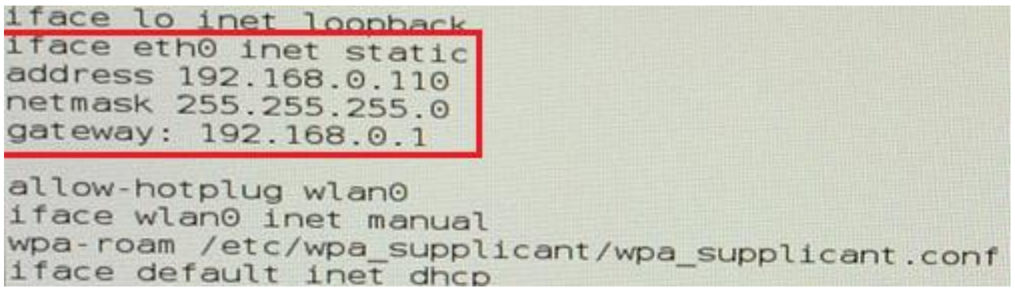
```
A1 Overscan      You may need to configure overscan if black bars are present on display
A2 Hostname      Set the visible name for this Pi on a network
A3 Memory Split  Change the amount of memory made available to the GPU
A4 SSH           Enable/Disable remote command line access to your Pi using SSH
A5 Device Tree   Enable/Disable the use of Device Tree
A6 SPI           Enable/Disable automatic loading of SPI kernel module (needed for e.g. PiFace)
A7 I2C           Enable/Disable automatic loading of I2C kernel module
A8 Serial        Enable/Disable shell and kernel messages on the serial connection
A9 Audio         Force audio out through HDMI or 3.5mm jack
A0 Update        Update this tool to the latest version
```

Figure 10

Finally, under Advanced Options, followed by SSH, enable the SSH server and reboot the Raspberry Pi by selecting <Finish> in the configuration menu. We will need the SSH server to allow for Hadoop cluster-wide operations. Once rebooted and with the "pi" user logged in again, the basic configuration setup of the Raspberry device has been successfully completed and we are ready for the next set of preparation steps.

Network configuration:

To make life a little easier, we launch the Raspbian GUI environment by entering startx in the Raspbian command line. (Alternatively, you can use, for example, the vi editor, of course.) Use the GUI text editor, “Leafpad”, to edit the `/etc/network/interfaces` text file as shown to change the local ethernet settings for eth0 from DHCP to the static IP address 192.168.0.110. Also add the netmask and gateway entries shown. This is the preparation for our multi-node Hadoop cluster.



```
iface lo inet loopback
iface eth0 inet static
address 192.168.0.110
netmask 255.255.255.0
gateway: 192.168.0.1

allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

Figure 11

Check whether the nameserver entry in file `/etc/resolv.conf` is given and looks ok. Restart your device afterwards.

Java environment:

Hadoop is Java coded so we require Java 6 or later to operate. Check whether the pre-installed Java environment is in place by executing:

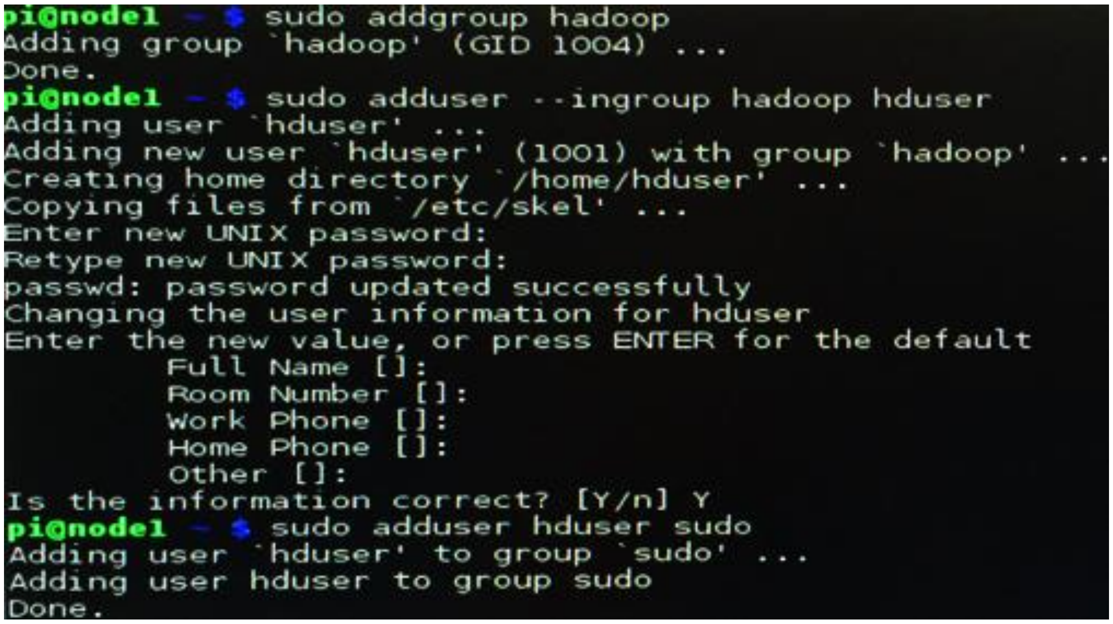
```
java –version
```

You should be prompted with a Java 1.8, i.e. Java 8, response.

Hadoop user & group accounts:

Set up dedicated user and group accounts for the Hadoop environment to separate the Hadoop installation from other services. The account IDs can be chosen freely, of course. i.e. group account ID “hadoop” and user account ID “hduser” within this and the sudo user groups.

```
sudo addgroup hadoop
sudo adduser --ingroup hadoop hduser
sudo adduser to group sudo
```



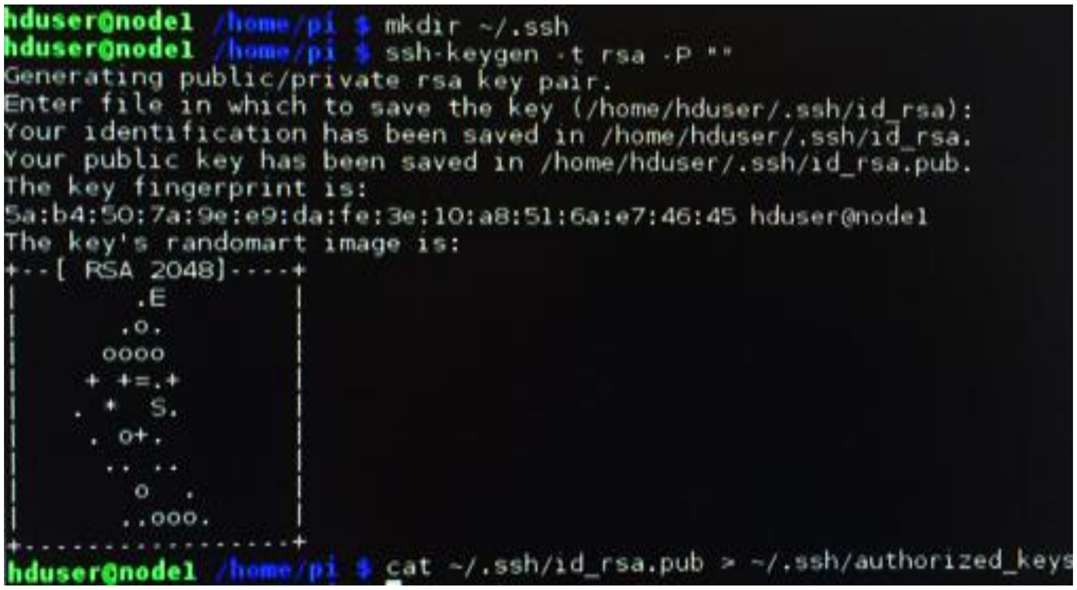
```
pi@node1 ~ $ sudo addgroup hadoop
Adding group `hadoop' (GID 1004) ...
Done.
pi@node1 ~ $ sudo adduser --ingroup hadoop hduser
Adding user `hduser' ...
Adding new user `hduser' (1001) with group `hadoop' ...
Creating home directory `/home/hduser' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for hduser
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] Y
pi@node1 ~ $ sudo adduser hduser sudo
Adding user `hduser' to group `sudo' ...
Adding user hduser to group sudo
Done.
```

Figure 12

SSH server configuration:

Generate an RSA key pair to allow the "hduser" to access slave machines seamlessly with empty passphrase. The public key will be stored in a file with the default Name "id_rsa.pub" and then appended to the list of SSH authorised keys in the file "authorized_keys". Note that this public key file will need to be shared by all Raspberry Pi's in a Hadoop cluster.

```
su hduser
mkdir ~/.ssh
ssh-keygen -t rsa -P “”
cat ~/.ssh/id_rsa.pub > ~/.ssh/authorized_keys
```



```
hduser@node1 /home/pi $ mkdir ~/.ssh
hduser@node1 /home/pi $ ssh-keygen -t rsa -P ""
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hduser/.ssh/id_rsa):
Your identification has been saved in /home/hduser/.ssh/id_rsa.
Your public key has been saved in /home/hduser/.ssh/id_rsa.pub.
The key fingerprint is:
5a:b4:50:7a:9e:e9:da:fe:3e:10:a8:51:6a:e7:46:45 hduser@node1
The key's randomart image is:
+--[ RSA 2048]----+
|       .E        |
|      .o.        |
|     oooo        |
|    + +=.+       |
|   . +  S.       |
|    . o+.        |
|     .. ..       |
|       o  .      |
|      ..ooo.     |
+-----------------+
hduser@node1 /home/pi $ cat ~/.ssh/id_rsa.pub > ~/.ssh/authorized_keys
```

Figure 13

Verify your SSH server access via: ssh localhost. This completes the Raspberry Pi preparations and we are all set for downloading and installing the Hadoop environment.

Hadoop installation & configuration:

Similar to the Raspbian installation & configuration description above, we have to install Hadoop first then followed by the various environment variable and configuration settings.

Basic setup

We need to get the latest stable Hadoop version (here: version 2.6.0) so we initiated the download from any of the various Apache mirror sites and download the latest jar file version according to your system configuration "*apache.cs.utah.edu/hadoop/common/stable2/hadoop-2.7.2.tar.gz".* Once the download has been completed, we unpack the archive to a sensible location, e.g., /opt represents a typical choice.

```
sudo mkdir /opt

sudo tar –xvzf hadoop-2.7.2.tar.gz -C /opt/
```

Following extraction, rename the newly created hadoop-2.7.2 folder into something a little more convenient such as “hadoop”.

```
cd /opt
sudo mv hadoop-2.7.2 hadoop
```

Running, for example,ls –al, we will notice that the “pi” user is the owner of the “hadoop” directory, as expected. To allow for the dedicated Hadoop user “hduser” to operate within the Hadoop environment, change the ownership of the Hadoop directory to “hduser”.

```
sudo chown -R hduser:hadoop hadoop
```

This completes the basic Hadoop installation and we can proceed with its configuration.

Environment settings:

Switch to the “hduser” and add the export statements listed below to the end of the shell startup file ~/.bashrc. Instead of using the standard vi editor we made use of the Leafpad text editor within the GUI environment.

```
su hduser
vi ~/.bashrc
```

Export statements to be added to ~/.bashrc:

```
export JAVA_HOME=$(readlink -f /usr/bin/java | sed “s:bin/java::”)
export HADOOP_INSTALL=/opt/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin:$HADOOP_INSTALL/sbin
```

This way both the Java and the Hadoop installation as well as the Hadoop binary paths become known to the user environment. Note that we may add the JAVA_HOME setting to the hadoop-env.sh script instead, as shown below.

Apart from these environment variables, modify the *`/opt/hadoop/etc/hadoop/hadoop-env.sh`* script as follows. If you are using an older version of Hadoop, this file can be found in: `/opt/hadoop/conf/`. Note that in case if decided to relocate this configuration directory, we will have to pass on the directory location when starting any of the Hadoop daemons (see daemon table below) using the–config option.

```
vi /opt/hadoop/etc/hadoop/hadoop-env.sh
```

Hadoop assigns 1 GB of memory to each daemon so this default value needs to be reduced via parameter HADOOP_HEAPSIZE to allow for Raspberry Pi conditions. The JAVA_HOME setting for the location of the Java implementation may be omitted if already set in your shell environment, as shown above. Finally, set the data node's Java virtual machine to client mode. (Note that with the Raspberry Pi 2 Model B's ARMv7 processor, this ARMv6-specific setting is not strictly necessary anymore.)

# The java implementation to use. Required, if not set in the home shell

```
export JAVA_HOME=$(readlink -f /usr/bin/java | sed "s:bin/java::")
```

# The maximum amount of heap to use, in MB. Default is 1000.

```
export HADOOP_HEAPSIZE=250
```

# Command specific options appended to HADOOP_OPTS when specified

```
export HADOOP_DATANODE_OPTS="-Dcom.sun.management.jmxremote
$HADOOP_DATANODE_OPTSi -client"
```

Hadoop daemon properties:

With the environment settings completed, we are ready for the more advanced Hadoop daemon configurations. Note that the configuration files are not held globally, i.e. each node in a Hadoop cluster holds its own set of configuration files which need to be kept in sync by the administrator using, for example, rsync. Modify the following files, as shown below, to configure the Hadoop system for operation in pseudo distributed mode. We can find these files in directory `/opt/hadoop/etc/hadoop`. In the case of older Hadoop versions, look for the files in: `/opt/hadoop/conf`.

core-site.xml	Common configuration settings for Hadoop Core
hdfs-site.xml	Configuration settings for HDFS daemons: The namenode, the secondary namenode and the datanodes.
mapred-site.xml	General configuration settings for MapReduce daemons. Since we are running MapReduce using YARN, the MapReduce jobtracker and tasktrackers are replaced with a single resource manager running on the namenode

Table 1

File: core-site.xml

```
<configuration>
     <property>
            <name>hadoop.tmp.dir</name>
            <value>/hdfs/tmp</value>
     </property>
     <property>
            <name>fs.default.name</name>
            <value>hdfs://localhost:54310</value>
     </property>
```

```
</configuration>
```

File: hdfs-site.xml

```
<configuration>
     <property>
          <name>dfs.replication</name>
          <value>1</value>
     </property>
</configuration>
```

File: mapred-site.xml.template (“mapred-site.xml”, if dealing with older Hadoop versions)

```
<configuration>
     <property>
          <name>mapred.job.tracker</name>
          <value>localhost:54311</value>
     </property>
</configuration>
```

Hadoop Data File System (HDFS) creation:

HDFS has been automatically installed as part of the Hadoop installation. Create a tmp folder within HDFS to store temporary test data and change the directory ownership to your Hadoop user of choice. A new HDFS installation needs to be formatted prior to use. This is achieved via -format.

```
sudo mkdir -p /hdfs/tmp
sudo chown hduser:hadoop /hdfs/tmp
sudo chmod 750 /hdfs/tmp
hadoop namenode -Format
```

Launch HDFS and YARN daemons:

Hadoop comes with a set of scripts for starting and stopping the various daemons. They can be found in the /bin directory. If you need to stop these daemons use,

```
/opt/hadoop/sbin/start-dfs.sh
/opt/hadoop/sbin/start-yarn.sh
```

Execution of Simple Word Count Program:

We will run some word count statistics on the standard Apache Hadoop license file to give Hadoop core setup a simple test run. The word count executable represents a standard element of your Hadoop jar file. To get going, we need to upload the Apache Hadoop license file into your HDFS home directory.

```
hadoop fs -copyFromLocal /opt/hadoop/LICENSE.txt /license.txt
```

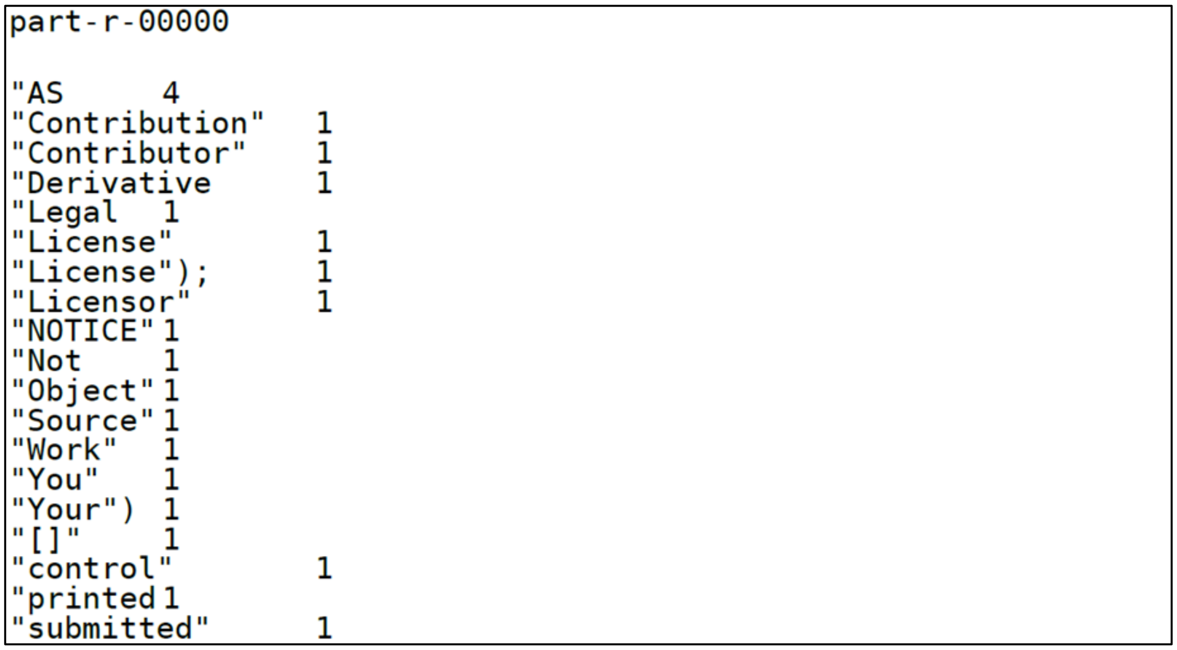
Run word count against the license file and write the result into license-out.txt.

```
hadoop jar /opt/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-
examples-2.7.2.jar wordcount /license.txt /license-out.txt
```

We can get hold of the HDFS output file via:

```
hadoop fs -copyToLocal /license-out.txt ~/
```

Have a look at ~/license-out.txt/part-r-00000 with the preferred text editor to see the word count results



```
part-r-00000

"AS	4
"Contribution"	1
"Contributor"	1
"Derivative	1
"Legal	1
"License"	1
"License");	1
"Licensor"	1
"NOTICE"	1
"Not	1
"Object"	1
"Source"	1
"Work"	1
"You"	1
"Your")	1
"[]"	1
"control"	1
"printed	1
"submitted"	1
```

Figure 14

3. THREADS – An alternative to HADOOP on MDFS

Hadoop vs Multithreading

Hadoop and MapReduce have been some kind of a "Black Magic" to me for quite some time now. It was a black box for me as I didn't know the internal details of how it works but I was coming across all kinds of stuffs about its magical power of managing Big data and solving all the application and data center problems that the present-day business is facing or is going to face. It at first appears that Hadoop is some kind of multithreading technology and meant to process big data over the internet, which is far from truth. MapReduce is not a conventional multithreading programming model.

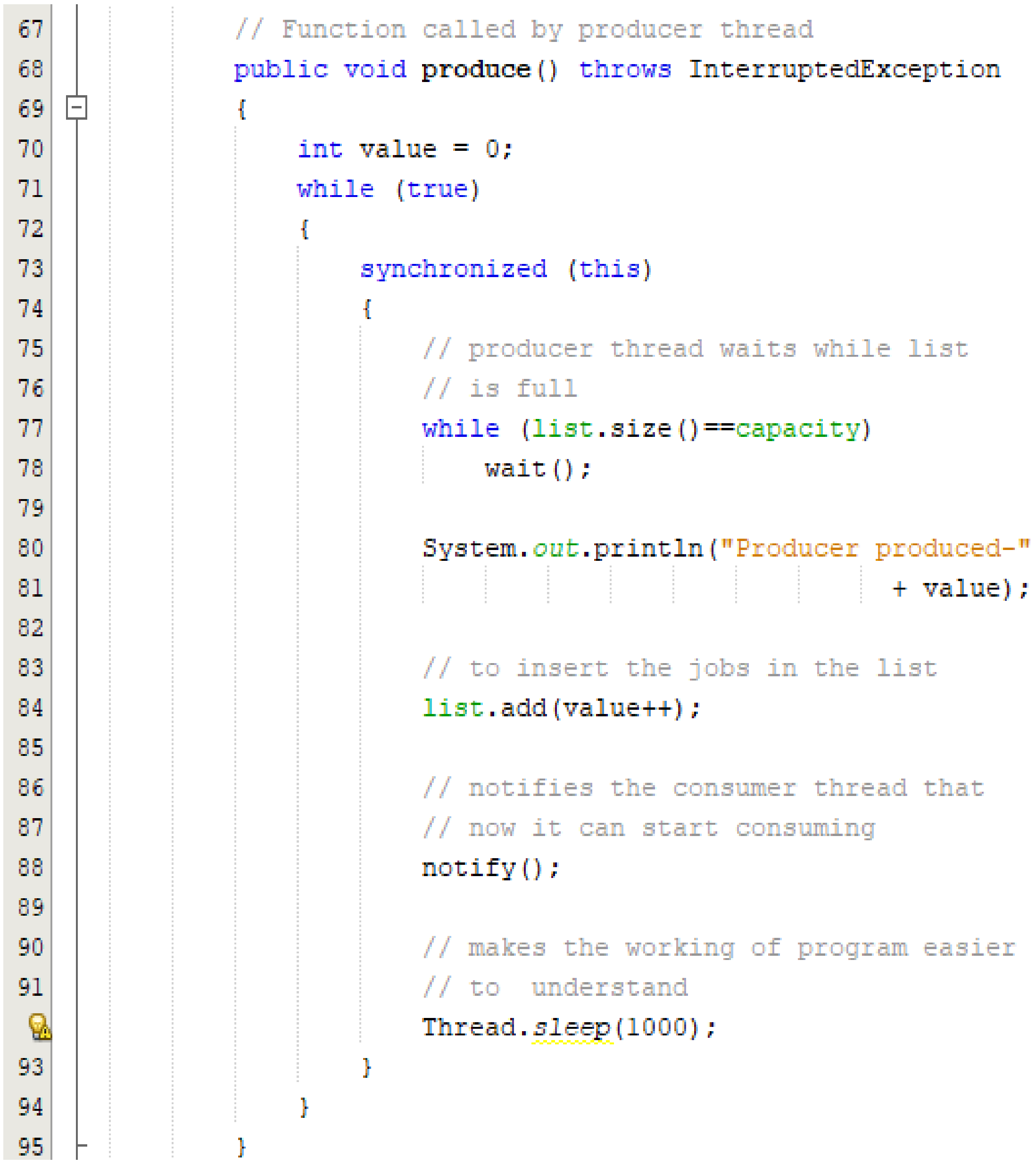
Of course, MapReduce derives its power from concurrent processing of a very large data set. However, Multithreading programming model and MapReduce Programming model are based on fundamentally different principles and both are meant to solve different kinds of data storage and processing problems. In case multithreading subsets of the processing logic is parallelized. The dataset upon which each of the thread operates is same and shared by all the concurrent processes by mechanisms such as Locks, Semaphores etc. In a multithreading environment the real programming challenges comes in terms of Deadlocks, Racing Condition, Circular Wait etc.

On the contrary, MapReduce way of programming arrange the processing logic into typically four phases (Mapping, Partitioning, Shuffling and Reducing) which operate on subsets of the data set (called data splits). The phases of processing are strictly sequential in nature. That means The Partitioning phase cannot start before all the maps are executed completely, Shuffling phase cannot start before partitioning is done completely and so on. For a Map Reduce algorithm to be useful in processing large data set we need to store the data in a distributed file system (i.e. data chunks are stored logically in one place but physically in several nodes of a distributed system.) and the data processing is done in each of the node concurrently on the respective chunk of data.

So, we have tried to develop a Thread programming code, which basically does the same job as Hadoop or you can say similar job. The explanation is given below along with code algorithms and result snippets:

- There are two actors, PRODUCER and CONSUMER.
- The PRODUCER breaks a large set of instruction into multiple thread execution.
- The CONSUMER consumes them one by one, making a large execution happen in a smaller time.
- This methodology is based on breaking of code, in HADOOP data is broken, here the Threads (Instructions) are broken.

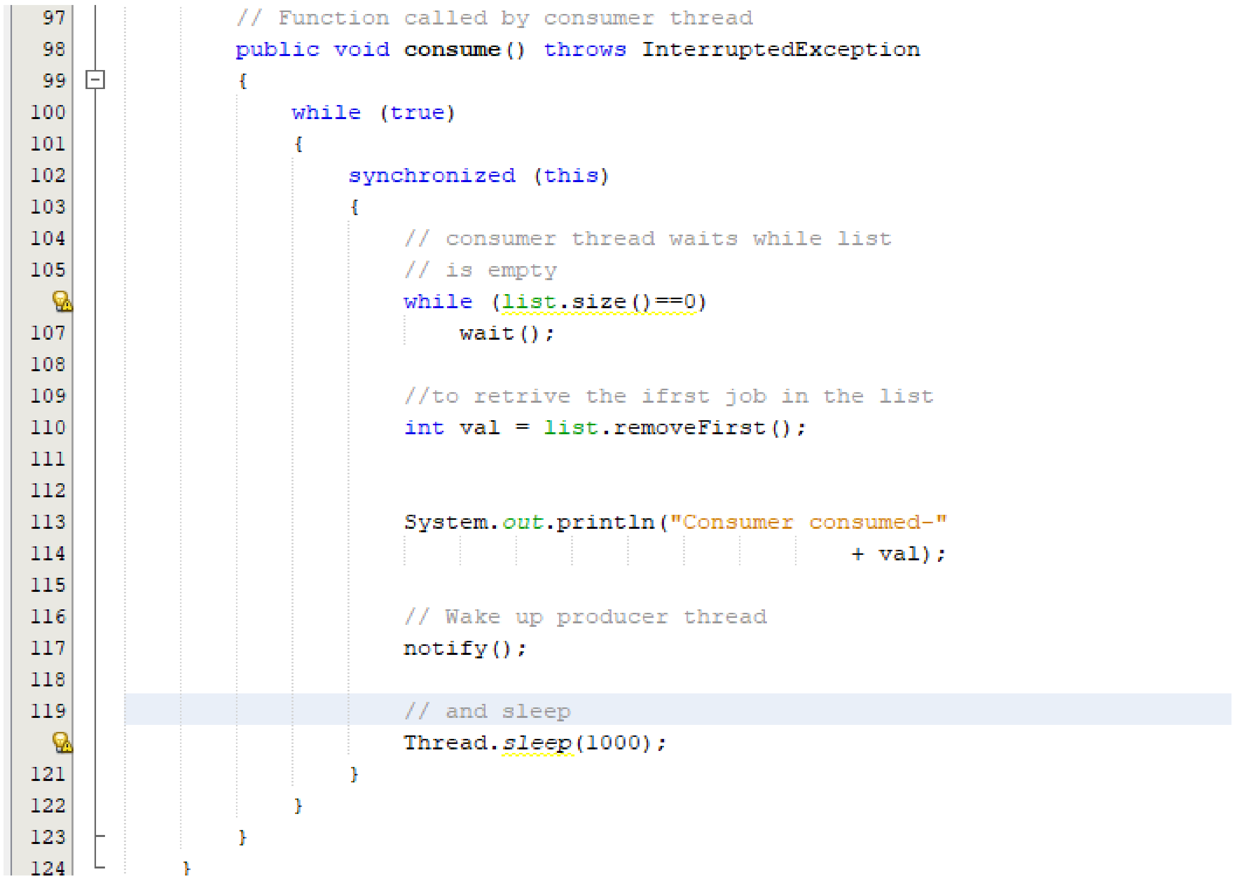
PRODUCER : (Java Code Snippet)



```java
        // Function called by producer thread
        public void produce() throws InterruptedException
        {
            int value = 0;
            while (true)
            {
                synchronized (this)
                {
                    // producer thread waits while list
                    // is full
                    while (list.size()==capacity)
                        wait();

                    System.out.println("Producer produced-"
                                                  + value);

                    // to insert the jobs in the list
                    list.add(value++);

                    // notifies the consumer thread that
                    // now it can start consuming
                    notify();

                    // makes the working of program easier
                    // to  understand
                    Thread.sleep(1000);
                }
            }
        }
```

Figure 15

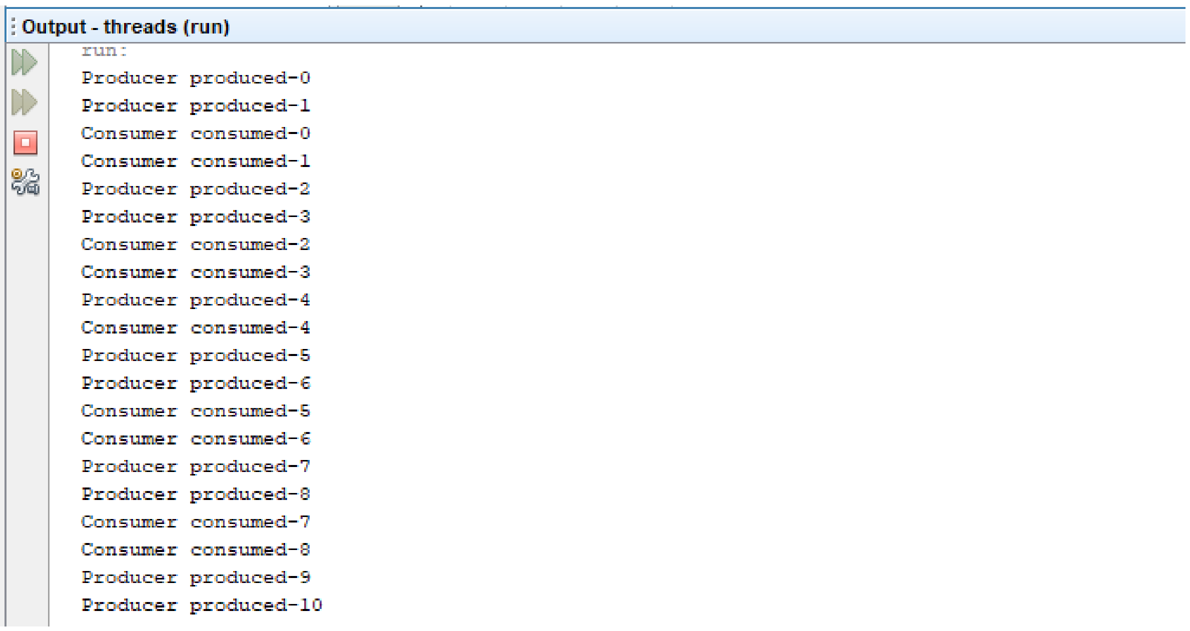
CONSUMER: (Java Code Snippet)



```
        // Function called by consumer thread
        public void consume() throws InterruptedException
        {
            while (true)
            {
                synchronized (this)
                {
                    // consumer thread waits while list
                    // is empty
                    while (list.size()==0)
                        wait();

                    //to retrive the ifrst job in the list
                    int val = list.removeFirst();


                    System.out.println("Consumer consumed-"
                                                    + val);

                    // Wake up producer thread
                    notify();

                    // and sleep
                    Thread.sleep(1000);
                }
            }
        }
    }
```

Figure 16

RESULT:



Output - threads (run)

```
run:
Producer produced-0
Producer produced-1
Consumer consumed-0
Consumer consumed-1
Producer produced-2
Producer produced-3
Consumer consumed-2
Consumer consumed-3
Producer produced-4
Consumer consumed-4
Producer produced-5
Producer produced-6
Consumer consumed-5
Consumer consumed-6
Producer produced-7
Producer produced-8
Consumer consumed-7
Consumer consumed-8
Producer produced-9
Producer produced-10
```

Figure 17

CONLUSION:

Multithreading is based on Parallelization of processing steps whereas Hadoop takes power by parallelization of Data. In a multithreading environment, as the data is shared by all the concurrent processes, the main challenges lie in co-coordinating the data access by all concurrent processes. In MapReduce the challenges of Data Access are taken care of by Hadoop framework and distributed file system such as HDFS.

However, the main programming challenge in Hadoop is a matter of breaking down the algorithm in a number of sequential jobs which are capable of exploiting data parallelism within each job. Neither all data processing algorithms can fit into the MapReduce model nor Hadoop is the solution to all your data processing challenges. But that's another discussion for another day.

- As we know running HADOOP on Mobile system, is quite difficult and inefficient.
- Threads can maybe not provide the same function but can surely aid the Map-Reduce methodology as HADOOP
- The biggest disadvantage is that coding becomes a hard part, and it is difficult to find such instruction codes which can be broken into threads.
- Multi-threads models lower the overhead but suffers from the huge cost of fault tolerance. If an executor in Spark fails, all the tasks running inside the executor have to re-run but only single task needs to re-run for MapReduce. Also, Spark suffers from huge memory pressure because all the tasks inside executor needs to cache data in terms of RDD. But MapReduce task only process one block at a time.
- The biggest advantage would be that it is very easy to operate on mobile devices and can reduce time of execution up to some extent.

Summary

Hadoop is an Apache open source framework written in Java that allows distributed processing of large datasets across clusters of computers using simple programming models. The Hadoop framework application works in an environment that provides distributed storage and computation across clusters of computers. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage. The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on commodity hardware. The kernel is the essential center of a computer operating system. It acts as an interface between the user applications and the hardware. The sole aim of the kernel is to manage the communication between the software (user level applications) and the hardware (CPU, disk memory etc.). Hadoop works on application layer so through this research we were trying to discover possibilities of implementing Hadoop on kernel level to optimize the kernel process and increases the efficiency of the output. Until now there is no successful research which shows the kernel level implementation of Hadoop around the globe. So, the motive of this research was basically to find why it is not implemented at kernel level, what are the difficulties faced and finding various methods which can be considered as possible ways to implement it.

After researching for a while we came across some possible proposed methods which can be tested to check the possibility of implementing hadoop on kernel but none of them till now are successfully implemented. Considering the feasibility all the methods, Implementation of Hadoop on Android kernel was the most feasible way and we can easily attempt to experiment with it. Android and Hadoop are two technologies which we decided to select to go further with our research and development of our project. These individual technologies solve many real time problems. So, with an aim of porting Hadoop on Android makes a lot of sense if one looks a step for the future technology. With Hadoop`s power we can bring great productivity on android platforms and in attempt of so we had followed various methods listed below:

- Running Hadoop framework on Android platform through different methods such as *"Chroot method",* Virtual Workspace using *"Samsung knox",* Using "*Android Studio*" and its "*Android Emulator".*

- Implementing Hadoop on Raspberry Pi 2. The aim of this attempt was to analyze Hadoop functioning on Raspberry Pi as a commodity hardware and to determine the possibilities of kernel level implementation of Hadoop. Since we are targeting ARM devices for our research work we decided to check it for Raspberry Pi also in hope of getting any positive results.

- THREADS – An alternative to HADOOP on MDFS. We have tried to develop a Thread programming code, which basically does the same job as Hadoop or you can say similar job.

Although we tried many possible ways to completely implement Hadoop on Android kernel but we could not integrate MDFS in the android application code. There were many to problems which we encountered such as several exceptions were encountered due to missing dependencies while writing android, Computational Power i.e. Hadoop requires high network speed and RAM. (with respect to targeted device), Battery life was a major concern. etc. Moreover, HDFS is written in Java, and runs as a user space application. There are potential performance improvements to be gained from writing a distributed filesystem in kernels pace, but IMO they are outweighed by the ease of development and debugging that come from running in user space. All the problems indicated that we need to change the hardware and software structure of Android devices to increase the possibilities of Implementation on Hadoop on Android Kernel.

Future Work

Keeping in mind the future work on this project we can clearly say that the process of integrating Hadoop and mobile applications can be broken down into three segments:

- Recognizing Hadoop's inherent limitations in mobile environment.
- Creating realistic Hadoop application frameworks
- Supporting and troubleshooting Hadoop in mobile applications

In context of upgrading hardware and software configuration of Android devices as mentioned before following can be done:

- Hadoop requires local disk memory, network speed and RAM. These can be built in mobile phones.
- Hadoop is completely written in java. Why should not we think about Hadoop in java micro edition?
- As battery life for phones can be a big concern, we have to find an alternative for it.

References

[1] Sneha Mehta1, Viral Mehta2, Hadoop Ecosystem: An Introduction, *International Journal of Science and Research (IJSR)*

[2] An Qin, Dandan Tu , Chengchun Shu, Chang Gao, XConveryer: Guarantee Hadoop Throughput via Lightweight OS-level Virtualization, *2009 Eighth International Conference on Grid and Cooperative Computing*

[3] Johnu George, Chien-An Chen, Radu Stoleru, Member, IEEE, Geoffrey G. Xie ,Hadoop MapReduce for Mobile Clouds, *IEEE transactions on cloud computing, vol. 3, no. 1, January 2014*

[4] Katayoun Neshatpour, Maria Malik, and Houman Homayoun, Accelerating Machine Learning Kernel in Hadoop Using FPGAs, *Department of Electrical and Computer Engineering,*

[5] Namrata B Bothe, Snehal S Karale,, Anagha N Mate , Nayan D Kumbhar, Migration of Hadoop To Android Platform Using 'Chroot', *IJIRCT1201105 International Journal of Innovative Research and Creative Technology*

[6] Arun S Devadiga, Shalini P.R, Aditya Kumar Sinha, Virtual Hadoop: The Study and Implementation of Hadoop in Virtual Environment using CloudStack KVM, *International Journal of Engineering Development and Research*

[7] Repository: git-wip-us.apache.org/repos/asf/hadoop.git (https://git-wip-us.apache.org/repos/asf/hadoop.git)

[8] Website: hadoop.apache.org (https://hadoop.apache.org)

[9] Anderson M., (2014). Using Chroot to Bring Linux.Applications to Android, Embedded Linux Conference +.Android Builders Summit April 29th – May 1st 2014, The Linux Foundation.

[10] Samsung, (2014). Meet evolving enterprise mobility Challenges with Samsung KNOX, White Paper, Samsung Inc

[11] Gupta, A., Preston, K., & Rodriguez, A. (2010). DebianRunner: Running Desktop Applications on Android Smartphones.

[12] Raspberry Pi Cluster setup: http://www.widriksson.com/raspberry-pi-hadoop-cluster