

On the Use of Embedded Devices for Heterogeneous Cloud Infrastructure

By

Pardeep Kumar

A thesis submitted in partial fulfillment of the requirements for the degree

of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

UNIVERSITY OF PUERTO RICO

MAYAGÜEZ CAMPUS

2016

Approved by:

Emmanuel Arzuaga, Ph. D.
President, Graduate Committee

Date

Pedro I. Rivera Vega, Ph. D.
Member, Graduate Committee

Date

Manuel Rodriguez Martinez, Ph.D.
Member, Graduate Committee

Date

José Cruz, Ph.D.
Graduate Studies Representative

Date

José G. Colom, Ph.D.
Department Chairperson

Date

Abstract of Thesis Presented to the Graduate School
of the University of Puerto Rico in Partial Fulfillment of the
Requirements for the Degree of Master of Science

On the Use of Embedded Devices for Heterogeneous Cloud Infrastructure

by

Pardeep Kumar

December 2016

Chair: Dr. Emmanuel Arzuaga

Department: Electrical and Computer Engineering Department

Regardless of the type of user, be it a computer professional or smart-phone user, working in a small bank or at a high tech IT company, everyone is (knowingly or not) using cloud-computing. In fact, the cloud-computing model provides various advantages over traditional computing in terms of service availability, scalability, processing and administration. Further, the cloud-computing framework relies on the principle of sharing resources and underlying hardware architectures through the virtualization of various components using software abstractions.

Increasing popularity of ARM-based boards with multi-core processors, along with commodity hardware components with cost-effective power consumption, yet with smaller and compact design, has exposed a wide range of opportunities to positively impact computing infrastructure design such as redefining building blocks for multiple computing paradigms such as parallel computing, virtual computing, cloud computing, high performance computing, and real-time computing. With widespread availability of such hardware it is possible for the common user to now build a specialized hardware solution to meet requirements on application performance, cost or power consumption.

This thesis is undertaken to study and evaluate the use of virtualization to enable the integration of ARM-based embedded boards into cloud computing infrastructure. In

particular, this work will focus on the understanding of how effective is virtualization on embedded boards and how this idea could be further nurtured to develop a whole cloud based service that provides for example virtual machines (VMs) on demand. We will discuss various techniques for setting up VMs using open source and publicly available software and tools on ARM-based boards. Our goal is to provide a system that is capable of running common desktop operating systems in current standard architectures, such as MS Windows and Ubuntu Linux on x86 as well as to have the capability of supporting native ARM-based operating systems taking advantage of the full virtualization support using KVM/QEMU on this platform.

Resumen de tesis presentado a la Escuela Graduada
de la Universidad de Puerto Rico como requisito parcial de los
requerimientos para el grado de Maestría en Ciencias

Usando Sistemas Embebidos como Infraestructura Heterogénea de Computación en la Nube

by

Pardeep Kumar

Diciembre 2016

Consejero: Dr. Emmanuel Arzuaga

Departamento: Electrical and Computer Engineering Department

Independientemente del tipo de usuario, ya sea un profesional de la computadora o un usuario de teléfonos inteligentes, trabajando en un banco o en una empresa de alta tecnología, todo el mundo está usando (a sabiendas o no) tecnología de computación en la nube (cloud computing en inglés). De hecho, el modelo de computación en la nube ofrece varias ventajas sobre la informática tradicional en términos de disponibilidad de servicios, escalabilidad, procesamiento y administración de recursos. Además, el marco de computación en la nube se basa en el principio de compartir recursos y arquitecturas de hardware subyacentes a través de la virtualización de varios componentes utilizando abstracciones de software.

La creciente popularidad del hardware basado en ARM con procesadores multi-núcleo (multi-core en inglés), junto con los componentes de hardware de los productos básicos con un consumo de energía rentable, pero con un diseño más pequeño y compacto, ha expuesto una amplia gama de oportunidades para impactar positivamente el diseño de infraestructura computacional para múltiples paradigmas de computación como computación paralela, computación virtual, computación en la nube, computación de alto rendimiento y

computación en tiempo real. Con la disponibilidad generalizada de este tipo de hardware es posible para el usuario común construir soluciones de hardware especializado que cumplan con los requisitos de rendimiento de la aplicación, el costo o el consumo de energía.

Esta tesis se lleva a cabo para estudiar y evaluar el uso de la virtualización para permitir la integración de hardware basado en la arquitectura ARM en la infraestructura de computación en la nube. En particular, este trabajo se centrará en la comprensión de la eficacia de la virtualización en sistemas basados en ARM y cómo esta idea podría fomentarse para desarrollar un servicio basado en la nube que proporciona por ejemplo máquinas virtuales (VM) a petición. Discutiremos varias técnicas para configurar VMs utilizando software libre así como software y herramientas disponibles públicamente en sistemas basados en ARM. Nuestro objetivo es proporcionar una plataforma que sea capaz de ejecutar sistemas operativos comunes en arquitecturas actuales, como MS Windows y Ubuntu Linux en x86, así como tener la capacidad de ejecutar sistemas operativos basados en ARM nativos aprovechando en pleno la disponibilidad de soporte a nivel del kernel como KVM/QEMU en esta plataforma.

Copyright ©2016

by

Pardeep Kumar

Contents

Title Page	i
Abstract	ii
Resumen	iv
List of Figures	ix
List of Tables	xi
1 INTRODUCTION	1
1.1 Motivation	1
1.2 Thesis contributions	2
1.3 Outline	3
2 Theoretical Background	5
2.1 Introduction	5
2.2 Related Work	5
3 Methodology	9
3.1 Introduction	9
3.2 ODROID XU4 board and its virtualization capability.	11
3.3 Performance evaluation using Dhrystone & Whetstone	15

3.4	Performance evaluation using TPC-C & TPC-H.	15
3.5	Migration of virtual machines between hosts	16
3.6	Setup of a heterogeneous cloud environment	19
3.7	Other Models	21
3.7.1	Apache Spark	21
3.7.2	Apache Spark benchmarking	22
4	Building Heterogeneous Cloud	24
4.1	Introduction	24
4.2	Openstack	25
4.2.1	Openstack Nova	27
5	Experiments & Results	36
5.1	Introduction	36
5.2	Dhrystone and Whetstone	36
5.3	Query Processing	39
5.3.1	MySQL	39
5.3.2	TPC-C & TPC-H Benchmark	39
6	Conclusion & Future Work	45
6.1	Introduction	45
6.2	Conclusion	45
6.3	Future Work	46
	Bibliography	47

List of Figures

3.1	ODROID-XU4 Embedded board	10
3.2	ODROID-XU4 running Window XP	12
3.3	ODROID-XU4 running x86 Ubuntu	13
3.4	ODROID-XU4 running ARM based Ubuntu	14
3.5	Traditional Cloud Environment	19
3.6	Proposed Heterogeneous Cloud Environment	20
3.7	Experimental setup for Apache Spark and Apache Hadoop	22
3.8	Job Execution Time	23
3.9	Average Throughput	23
4.1	Openstack Architecture	26
4.2	Openstack Nova Architecture	27
4.3	Openstack Nova execution flow.	28
4.4	Controller Node running on x86 Ubuntu 16.04 LTS	31
4.5	Compute Node running on ARM based Ubuntu 16.04 LTS	31
4.6	Compute Node running on x86 Ubuntu 16.04 LTS	32
4.7	Available Compute Nodes, Flavor-list and Glance-Images on Controller . .	32
4.8	Boot command to issue VM to a specific host	33
4.9	'nova list' command to list all the VMs and their status	33
4.10	Output of 'nova-show' command to show the details of VM.	34

4.11	Initiating live-migration of VM from x86 based host to ARM based host . . .	34
4.12	Successfully migrated VM	35
5.1	General use cases for our performance analysis.	36
5.2	Overview of experimental setup for benchmarking.	37
5.3	Dhrystone benchmark results.	38
5.4	Whetstone benchmark results.	38
5.5	Use cases for TPC-C benchmark.	40
5.6	TPC-C benchmark results.	41
5.7	Use cases for TPC-H benchmark.	42
5.8	TPC-H benchmark results.	42
5.9	TPC-H benchmark results.	43

List of Tables

3.1	To start our defined VM we can issue: <code>virsh start ubuntu14.04-arm</code>	18
4.1	Openstack Mitaka Configuration (for minimal setup)	30
5.1	TPC-C & TPC-H dataset	43

List of Abbreviations

ARM Advanced RISC Machines

eMMC embedded Multi-Media Controller

GPIO General-purpose input/output

GPU Graphical Processing Unit

KVM Kernel-based Virtual Machine

LTS Long Term Support

OLAP Online Analytical Processing

OLTP Online Transactional Processing

QEMU Quick Emulator

TPC-C An on-line transaction processing benchmark from TPC

TPC-H An on-line analytical processing benchmark from TPC

Chapter 1

INTRODUCTION

1.1 Motivation

Increasing capacity and the reduced size of various hardware components such as processors, RAM, GPU, etc. has contributed heavily in the development of high-end smartphones and other ARM-based embedded devices [1, 4]. These small devices tend to be cheaper and yield lower power consumption than traditional x86 servers. At the same time, the computational power of these devices has increased dramatically to the point where they are currently capable of running full versions of complex operating systems such as Linux [5]. Therefore, there is a potential for setting up clusters, for cloud infrastructure, parallel computing or even for running virtual machines on these ARM-based embedded boards. In order to use such highly efficient hardware to its full extent and getting the maximum out of it, one needs understand the capabilities of the hardware platform and align it with software requirements. Instead of using multiple processors on the same board, the increasing demands for CPU-cores are now being completed by using single multi-core processors with on-board RAM, Ethernet and GPU. ARM processors are playing a key role in providing low power, yet powerful CPU architecture designs. This further leads to the development of embedded boards, which are small

in size with comparative performance as of high-end machines for some applications [9]. For example the ODROID-XU ARM-based board [4] has a hardware configuration of a multi-core CPU with all the peripherals required to convert it to a minicomputer running Linux/Ubuntu(14.04 LTS).

Virtualization is a fundamental building block in cloud computing infrastructure, it is an effective way of consolidating hardware resources and adding flexibility to the overall system resource usage. Currently, there are many commercial as well as freely available software and tools to set up and manage VM, such as Oracle VirtualBox, VMware, XEN, KVM [8] and they provide the capability of Desktop Virtualization. A considerable amount of research in virtualization has focused on enabling VM execution on x86 machines, but the same is not true for ARM-based embedded devices. This presents challenges and opportunities to learn from the available solutions for x86 machines to enable VM execution and thus the creation of a Cloud Computing System based on ARM-based boards.

In this work, we will use ARM-based boards to build cloud infrastructure. We have made preliminary performance tests and have found that they can yield comparative performance to traditional x86 hardware with a small form factor and low power consumption. Combining virtualization with this commodity hardware, we have an excellent platform to build a prototype of our proposed design of low-powered cloud computing infrastructure.

1.2 Thesis contributions

In this thesis we make the following contributions:

- Investigating current potentials of ARM-based ODROID XU4 board and its virtualization capability.

– Pardeep Kumar and Emmanuel Arzuaga, *Integrating ARM Devices in Cloud*

Infrastructure, CAHSI Summit at HENAAC Conference 2016 (received the **Best Graduate Research Poster Award**).

- Performance evaluation of the ARM-based system prototype by running specialized commercial workloads based on the TPC-C and TPC-H benchmark, to study and understand the behaviour of the system for OLTP and OLAP transactions under different scenarios.
 - Performance Evaluation of Commercial Workloads on ARM-Based Virtualized Systems, Submitted to ACM VEE 2017
- Present a novel heterogeneous cloud computing infrastructure paradigm using embedded ARM hardware and x86 hosts.
 - Migration of virtual machines between heterogeneous hosts (ARM-based hosts and x86 hosts) and facilitating VMs to work seamlessly around the systems.
 - Provide a prototype of such system
- Setup of a heterogeneous cloud environment using embedded ARM hardware and x86 hosts.
- Evaluate ARM based cluster configurations that handle other Cloud Computer Paradigms
 - Apache Spark

1.3 Outline

Chapter 2 provides background of what has been done in this area and how our work differentiates or adds to it. The step by step process of the completion of the objectives

is explained in Chapter 3. Chapter 4 illustrate our approach of building proposed Heterogeneous Cloud. A study of the results for all benchmarks are given in Chapter 5. Finally, Chapter 6 gives the conclusions and the direction for further development.

Chapter 2

Theoretical Background

2.1 Introduction

Advancement in development of powerful commodity hardware has traditionally driven costs down along with potentially reducing their form factor, has motivated many developers and researchers around the globe and provided them the opportunity to develop low cost platforms for education and experimentation. The ARM-based embedded boards currently dominate the phone and tablet market, the same way Intel dominates the low to mid range server sector, be it in power consumption or FLOPS (floating point operations per second) per cost ratio. These boards are based on 32-bit ARM processors whereas the 64-bit version of these boards are still in development/launch queue or early adoption hardware. Most of the developers are waiting and others trying new ways of setting up small/medium cloud computing environments on 32-bit versions of these powerful machines.

2.2 Related Work

The statistics and analysis provided by the author in [1], for ten ARM-based embedded boards and their comparison with the Intel Atom (an x86 implementation targeting

the embedded market) and two x-86 based high end servers, could be used as underlying principles for deciding the appropriate hardware for conducting experiments. They have provided comparison tables for power consumption, FLOPS , FLOPS per watt and FLOPS per cost for each mentioned board (refer to Table I). The author has used various benchmarks such as HPL (High performance Linpack), which is commonly used to measure performance of supercomputers worldwide, and STREAM, which tests a machines memory performance by performing various operations like copying bytes in memory, then adding or scaling its values.

The work done by the author [3] for getting processor virtualization in embedded boards using KVM/ARM on ARM based TI OMAP5432 uEVM board (Texas Instruments) has also provided performance analysis of different virtualization mechanisms. The author has preferred KVM over Xen as a hypervisor and successfully implemented a running prototype of a Virtual Machine on top of QEMU [6] and KVM/ARM. In our work, we have relied on QEMU to work as an emulator and virtualizer to run a virtual machine on top of it. In, [2], the author has stated different virtualization solutions for embedded boards in order to get platform level virtualization. The paper provided an overview of all the solutions and issues related to full virtualization in contrast with paravirtualization. In this paper we have provided a QEMU based solution which comes under Type-2 virtualization solution along with KVM. In all the solutions provided in the paper KVM seems to be the best option for full platform virtualization, but it relies on hardware virtualization extensions that are currently limited on embedded boards. Thus using QEMU as a hardware emulator and virtualizer is one of the best ways on ODROID-XU to get Type-2 full virtualization.

The comparative study done by author [9] has shown how ARM-based nodes are much more energy efficient and are able to replace traditional x86 servers at the cost of slightly lower throughput. ARM based ODROID-XU was the target machine for their

experimentation and comparison to Intels Xeon servers. They have used many benchmarks for testing the performance and power consumption of ARM boxes to run under the hadoop cluster. They have stressed all the system components and characterized the CPU, memory and I/O using different workloads given as part of Hadoop examples and query processing benchmarks such as TPC-C and TPC-H. The paper, [9], discussed the impact of big data and how ARM based boards can help in providing required computation at low power consumption and at slightly low throughput. In this work, we will be more focused on characterising the ARM-based ODROID XU4 box which is an advanced version of the board used by author in [9]. We will be running query processing benchmarks, such as TPC-C and TPC-H, natively on ARM boxes, as well as on virtual machine instances (armhf and x86) running on these ARM boxes.

Researchers [10] from University of Southern California have proposed a heterogeneous cloud by providing an ability to user to specify more specific option for a target architecture on demand. As heterogeneity can also be achieved among same processor architectures but by adding new features to processor on user demand. They have provided the GPU based accelerator support for virtual machines via Libvirt driver and support for Tilera Linux which lacked KVM or virtualization extensions on their cloud. They build and demonstrated the use of bare metal or Type-1 hypervisor on Tilera architecture by developing a proxy compute node to manage and map requests from OpenStack. In this work we proposed a heterogeneous cloud using Type-2 hypervisor on our compute nodes running on 32 bit ARM based embedded board and x86 machines for running ARM or x86 based virtual machines irrespective of the host's native capabilities. We have used Libvirt and QEMU in order to provide full platform virtualization for guest OS and to facilitate virtual machine creation, management and migration between ARM and x86 based hosts.

Authors in paper [14] provides overview and benefits of mobile virtualization on ARM

based devices. They have provided advantages and disadvantages of different virtualization technologies such as OS level virtualization, Microkernel, Hardware-Assisted and Para-virtualization. They used Android Containers under OS level virtualization solution, CodeZeroOKL4 for Microkernel based virtualization solution, KVM-on-ARM for Hardware-Assisted full virtualization and EmbeddedXEN (Xen on ARM) for Para-virtualization solution. They used Urbetter S5PV210 board with Exynos 4412 ARM Quad-core CPU, 2GB RAM and 8GB ROM and provided performance analysis of Context Switching, Micro and Macro benchmarks, and full scalability analysis under three use cases by running them on Microkernel, Para-virtualization and Hardware-Assisted based virtualization solution. Under these use cases, they measured context switching time between guest OSes and host OS. Further, LMBench micro-benchmark suite is used to measure latency and bandwidth, Qtopia & WMV stream encoder/decoder macro-benchmark for UI loading and codec benchmarking. For scalability analysis they increased the number of VMs running under each use case progressively and then captured and analysed the system's CPU, Memory and Storage usage. In our case, we used KVM/QEMU as a hypervisor on ARM-based Ubuntu Host OS and collected data for CPU intensive and query processing benchmarks on native OS as well as in virtual machines running ARM & x86 based OS.

Chapter 3

Methodology

3.1 Introduction

In this chapter, we take proposed objectives one by one and provide steps taken for their completion. The purpose of this work is to examine and demonstrate the ability of ARM based embedded boards against different workload over physical or virtual environments. The purpose of this chapter is to (1) investigate potential and virtualization capability of ARM boards, (2) performance evaluation under CPU-intensive workloads, (3) performance evaluation under Query processing workloads, (4) migration of virtual machine in real time, and (5) setting up a heterogeneous cloud.

ODROID-XU has been shown to provide the best set of hardware components to support general purpose computing environments [1]. Furthermore, the ODROID-XU4 is a more advanced version of the ODROID-XU family, thus we have selected it as a basis to conduct our experiments.

The ODROID-XU4 box [4] comes with an Exynos 5422 Octa Cortex, which consists of two quad-core heterogeneous processors, i.e. one Cortex A15 and one Cortex A7, arranged in what is commonly known as ARM Big.Little Architecture. It also has a Mali GPU, 2GB of Dual Channel Low Power DDR3 RAM, three USB ports: two USB 3.0 and one

USB 2.0, eMMC 5.0 flash storage of 32GB and one Gigabit Ethernet along with a standard HDMI port. The availability of such a hardware platform leads to rapid experimentation and performance analysis of different technologies, e.g., running application servers or VMs on these devices and comparing them with execution in traditional x86 machines.



Fig 3.1: ODROID-XU4 Embedded board

The idea is to couple a low power consuming and slower Cortex A7 processor with relatively more powerful and power consuming Cortex A15. The key is to swap workloads around them on the fly based on the application needs and requirements. For example, if an application requires high processing only while booting up but has less demands in rest of its life cycle then we can boot it up using the bigger cores and later switch it to smaller cores. The intention behind big.Little is to have a such a system which can adjust better to dynamic computing needs and save as much power as possible.

For this research we used three ODROID-XU4 boxes and two x86 machine. The idea here is to setup Ubuntu 14.04/16.04 (LTS) on each node. ODROID-XU4 has support for running full-fledged latest Ubuntu operating system specially released for ARM architec-

ture by hardkernel.com (we will refer to it as Ubuntu-armhf and for x86 as Ubuntu-x86).

3.2 ODROID XU4 board and its virtualization capability.

In order to get best out of this commodity hardware we need to set goals and examine every single feature provided to us in such small device. As we have already discussed that Virtualization is the key for Cloud Computing hence the first step is to examine what this board can offer us. Lets talk about the things we can expect out of the box:

- ODROID XU4 comes with preloaded Android OS, but one can easily boot it with Ubuntu LTS 14.04 or 16.04 armhf (hf means hard float) with full support for onboard components.
- We have onboard gigabyte Ethernet but one can easily buy WiFi module from official website of hardkernel.
- ODROID XU4 also provides on board GPIO/I2C/I2S, serial console and UART port.
- Other commodity hardware such as high quality video cam, pluggable display screen, bluetooth, etc. are fully supported and works out of the box.

In order to find out virtualization capability of such hardware we have to first look at what are the basic requirements for virtualization. Firstly, we have to decide what architecture we are going to support for running VMs (Virtual Machine). Secondly, if we are planning to support Type I or Type II virtualization. Thirdly, to find out if we can run Vms with KVM extensions to get near native performance. To these ends we have decided to support x86 and ARM based VM using Type II, full virtualization, with KVM support for ARM Vms running on Cortex A15 (it supports KVM).

ODROID XU4 box doesn't have or support many components required by standard operating systems such as PCI bus or VGA support. We used QEMU, Quick Emulator,

to emulate x86 and ARM hardware for running unmodified x86 and ARM based guests. Further, in order to support KVM we recompiled the Linux kernel to add support for available Cortex A15 cores.

Next we started configuring, building and running our VMs on ODROID XU4 using QEMU command line [6]. We experimented for following different configurations and scenarios using QEMU:

- Running custom ARM based linux kernel with small initial ram disk attached to it with total size of 3.7 MB.
- Running ARM based Ubuntu server image without KVM support.
- Running ARM based Ubuntu server images with full KVM support.
- Running standard x86 based Ubuntu server and Windows XP.

QEMU Commands used:

- For Windows:

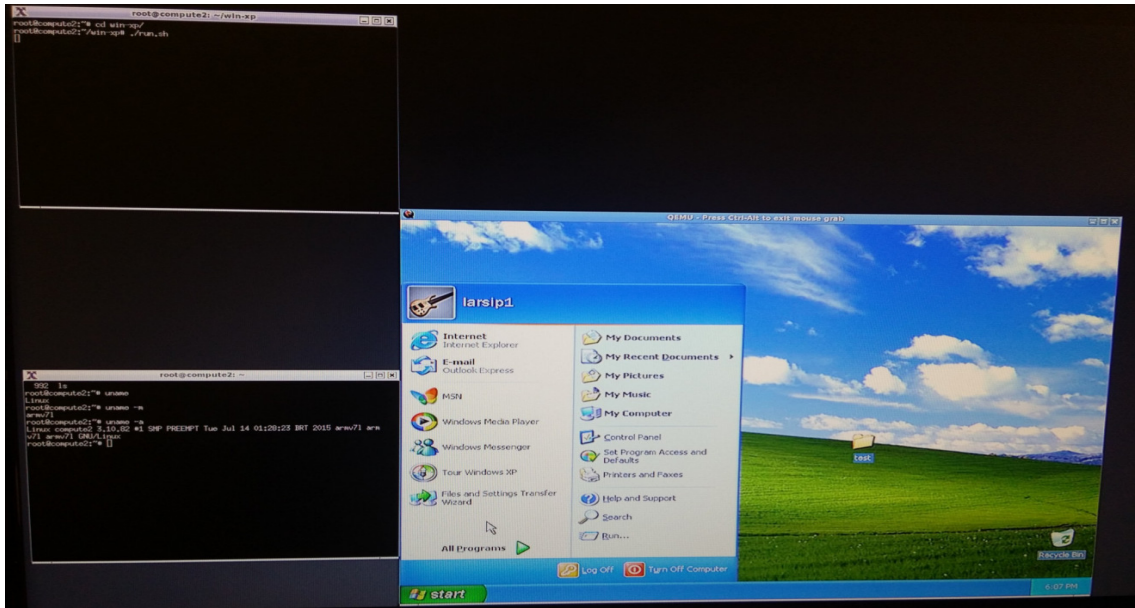
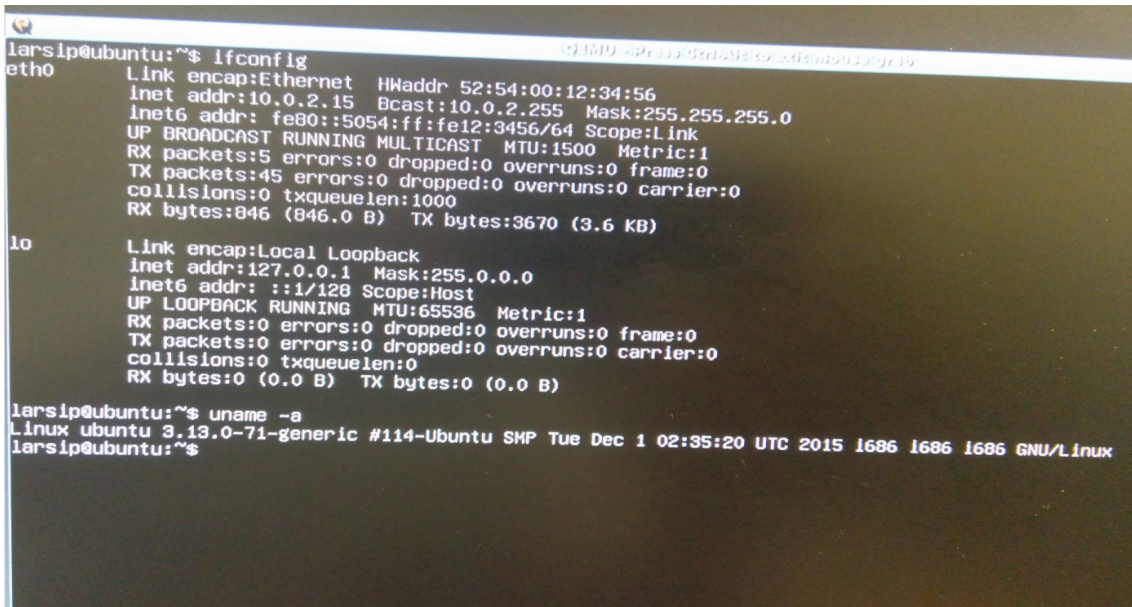


Fig 3.2: ODROID-XU4 running Window XP

qemu-system-i386 -hda winxp.img -m 1536

- For Ubuntu x86:



```

larsip@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 52:54:00:12:34:56
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::5054:ff:fe12:3456/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:5  errors:0  dropped:0  overruns:0  frame:0
          TX packets:45  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0  txqueuelen:1000
          RX bytes:846 (846.0 B)  TX bytes:3670 (3.6 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128  Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0  errors:0  dropped:0  overruns:0  frame:0
          TX packets:0  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0  txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

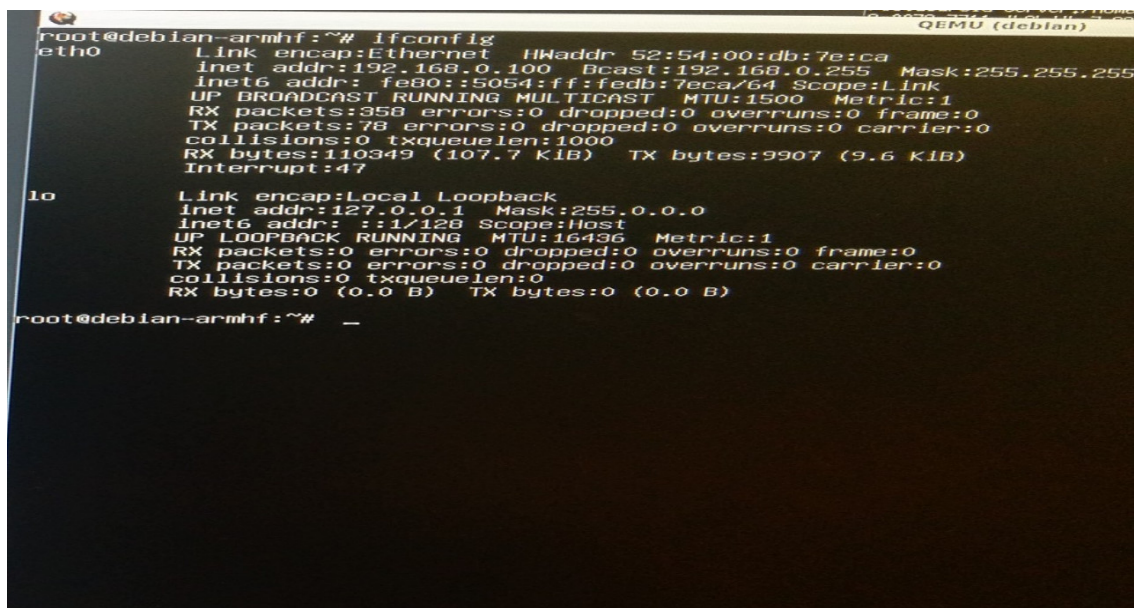
larsip@ubuntu:~$ uname -a
Linux ubuntu 3.13.0-71-generic #114-Ubuntu SMP Tue Dec 1 02:35:20 UTC 2015 i686 i686 i686 GNU/Linux
larsip@ubuntu:~$

```

Fig 3.3: ODR0ID-XU4 running x86 Ubuntu

qemu-system-i386 -hda ubuntu-image.img -m 1536

- For Ubuntu ARM:



```

root@debian-armhf:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 52:54:00:db:7e:ca
          inet addr:192.168.0.100  Bcast:192.168.0.255  Mask:255.255.255
          inet6 addr: fe80::5054:ff:fedb:7eca/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:358 errors:0 dropped:0 overruns:0 frame:0
          TX packets:78 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:110349 (107.7 KiB)  TX bytes:9907 (9.6 KiB)
          Interrupt:47

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@debian-armhf:~#

```

Fig 3.4: ODROID-XU4 running ARM based Ubuntu

```

qemu-system-arm -M vexpress-a15 -m 1536 -cpu cortex-a15 -serial stdio -kernel
kernel/zImage -dtb kernel/rtsm-ve-cortex-a15x1.dtb -smp 2, sockets=2, cores=1,
threads=1 -drive file=ubuntu-xu4.img,id=virtio-blk,if=none -device virtio-blk-device,
drive=virtio-blk -device virtio-net-device, netdev=net0, mac="52:54:00:12:34:55"
-netdev type=tap, id=net0, script=no, downscript=no, ifname="tap0" -append
"earlyprintk=ttyAMA0 console=ttyAMA0 mem=1536M root=/dev/vda2 rw ip=dhcp
"

```

- For Ubuntu ARM (with KVM):

```

qemu-system-arm -enable-kvm -M vexpress-a15 -m 1536 -cpu cortex-a15 -serial
stdio -kernel kernel/zImage -dtb kernel/rtsm-ve-cortex-a15x1.dtb -smp 2, sockets=2,
cores=1, threads=1 -drive file=ubuntu-xu4.img,id=virtio-blk,if=none -device virtio-
blk-device, drive=virtio-blk -device virtio-net-device, netdev=net0, mac="52:54:00:12:34:55"
-netdev type=tap, id=net0, script=no, downscript=no, ifname="tap0" -append
"earlyprintk=ttyAMA0 console=ttyAMA0 mem=1536M root=/dev/vda2 rw ip=dhcp
"

```

” With all setup and running we are good to tackle our next objective.

3.3 Performance evaluation using Dhrystone & Whetstone

We have made preliminary performance tests by running Dhrystone [18] and Whetstone [19] benchmarks on these boards. Further, these benchmarks have provided us with a vision as what we can expect in terms of running virtualized solutions on these small boards. We further divided our work based on the placement of benchmark under different scenarios i.e. running:

- Benchmarks natively and pinned to one Cortex-A15 core.
- Benchmarks natively and pinned to one Cortex-A7 core.
- Benchmarks in a Ubuntu-armhf VM running on top of host OS using one CPU with KVM.
- Benchmarks in a Ubuntu-armhf VM running on top of host OS using one CPU but without KVM extensions.
- Benchmarks in a Ubuntu-x86 VM running on top of host OS using one CPU.

We collected the Dhrystone and Whetstone benchmarks results for above mentioned cases and detailed analysis is done and explained in chapter 5

3.4 Performance evaluation using TPC-C & TPC-H.

Now that we have overall understanding of virtualization capabilities of our embedded board, and we have our VMs up and running on demand using QEMU command line, provides basis for performance evaluation of such VMs under specialized benchmarks such as TPC-C [11] and TPC-H [12].

TPC-C simulates an OLTP (Online Transactional Processing) type order entering environment where users executes transactions against database. It executes different transactions on the database such as delivery, new order, order status, payment and monitoring stock level at warehouses. The performance is measured in new-order transactions per minute (tpm).

Whereas, TPC-H simulates an OLAP (Online Analytical Processing) type decision support system against the database. It executes business oriented adhoc queries as well issues concurrent data modification queries.

We conducted our benchmark experiments under five different use cases such as:

- Running benchmarks directly on native host system with full access to hardware resources.
- Running benchmarks directly on native host system with only one CPU assigned.
- Running benchmarks inside a ARM based Ubuntu virtual machine using one cpu, 1.5 gigabyte of RAM without KVM support.
- Running benchmarks inside a ARM based Ubuntu virtual machine using one cpu, 1.5 gigabyte of RAM with KVM support.
- Running benchmarks inside a x86 based Ubuntu virtual machine using one cpu and 1.5 gigabyte of RAM.

We collected the TPC-C and TPC-H output for above mentioned cases, a detailed analysis is done and explained in chapter 5.

3.5 Migration of virtual machines between hosts

Now that we have ODROID-XU4 setup and evaluated among various benchmarks running across different scenarios, as discussed in previous section, we are ready to manage and migrate those VMs around the system.

Up to now we have virtual machines running using QEMU from the command line. In order to manage all such virtual machine we need to run them in a unified way. Libvirt [16] is an open source library for managing platform virtualization. The idea here is to use Libvirt as it supports QEMU and KVM out of the box and takes an XML based approach to define VM configuration. Moreover, it also provides set of commands to manage life cycle of VM as well as to live migrate VM across different hosts. Libvirt is used by many virtualization programs and it also has a graphical interface, VMM (Virtual Machine Manager), and ,virsh, a command line interface. The graphical interface [17] has limited functionality based on the version you have installed .

Based on our need and requirements of using commands which are hard to set via graphical console and to be consistent in our scripts to interact with Libvirt we opted to not to use its graphical interface. For our experiments we installed Libvirt from Ubuntu repository using apt-get command. Once installed Libvirt and its commands can be accessed via terminal by typing virsh and the command to run. There are specific steps one needs to follow to get virtual machines running through libvirt such as:

- Define your virtual machine or domain in XML using Libvirt XML tags or auto-generate domain(virtual machine) XML using arguments we used to run VM using QEMU. The command to do this is virsh domxml-from-native, it takes qemu arguments and generates XML.
- Once domain is defined, run domain using: virsh start domain-name
- Check if domain started or not by listing it out using: virsh list. For example, to define ARM based ubuntu server using virsh we can use XML given in Table 3.1

Additional options can be provided in XML which can further be conveyed to QEMU by Libvirt such as setting VGA to NONE or adding vnc server to domain. Each mentioned option in domain XML gets mapped to QEMU command line option by Libvirt on the

```

<domain type='qemu' id='1'>
<name>ubuntu14.04-arm</name>
<uuid>932bd1dd-2f43-4c7c-81d0-a3b1b5748a6c</uuid>
<memory unit='KiB'>524288</memory>
<currentMemory unit='KiB'>524288</currentMemory>
<vcpu placement='static'>1</vcpu>
<resource>ubuntu14.04-arm <partition> /machine </partition>
</resource>
<os>
<type arch='armv7l' machine='vexpress-a15'> hvm </type>
<kernel> zImage</kernel>
<cmdline> root=/dev/mmcblk0p2 rw </cmdline>
<dtb> rtsm_ve-cortex_a15x1.dtb </dtb>
<bootmenu enable='yes' />
</os>
<clock offset='utc' /> </>
<on_poweroff>destroy </on_poweroff>
<on_reboot>restart </on_reboot>
<on_crash> restart </on_crash>
<devices> <emulator> qemu-system-arm </emulator>
<disk type='file' device='disk'>
<driver name='qemu' type='raw' cache='none' />
<source file='ubuntu14.img' />
<backingStore />
<target dev='sda' bus='sd' />
<alias name='sd-disk0' />
</disk>
</devices>
</domain >

```

Table 3.1: To start our defined VM we can issue: `virsh start ubuntu14.04-arm`.

fly. In particular the architecture information is used to find the right qemu system binary to emulate the appropriate machine. For example, for `arch=armv7l`, Libvirt will pick `qemu-system-arm` binary then parse other options mentioned in XML and pass it to `qemu-system-arm`. Libvirt also takes advantage of various optimizations available for target architecture by default while parsing XML and automatically adds accelerator type, clock type etc., if not mentioned in explicitly in XML.

To find out the detailed capacity of the host on which Libvirt is running one can issue command, `virsh capabilities` and check all the supported architecture, machine types and other details as required.

3.6 Setup of a heterogeneous cloud environment

Heterogeneous cloud provide various benefits in terms of efficient power consumption, scalability and optimized performance. Cloud computing provides isolated and administrative access to virtualized computing resources on demand. With availability of numerous problems where the underlying architecture being used can make huge difference, in terms of speed of execution, performance or cost, cloud computing platforms which support heterogeneity are still not available for commercial use. We have our ODROID-XU4 box with a capability of managing and deploying virtual machines using Libvirt on a single host. Next step is to start configuring our heterogeneous cloud environment on ODROID-XU4 nodes running Libvirt and using QEMU as hypervisor. There are many big players in the market which allows anyone to configure or use their cloud services for public or private use such as Amazon AWS, Microsoft Azure, OpenStack etc. In our case Openstack is the best suited one as its free, open source, integrates easily with Linux based hosts and has a huge community behind it.

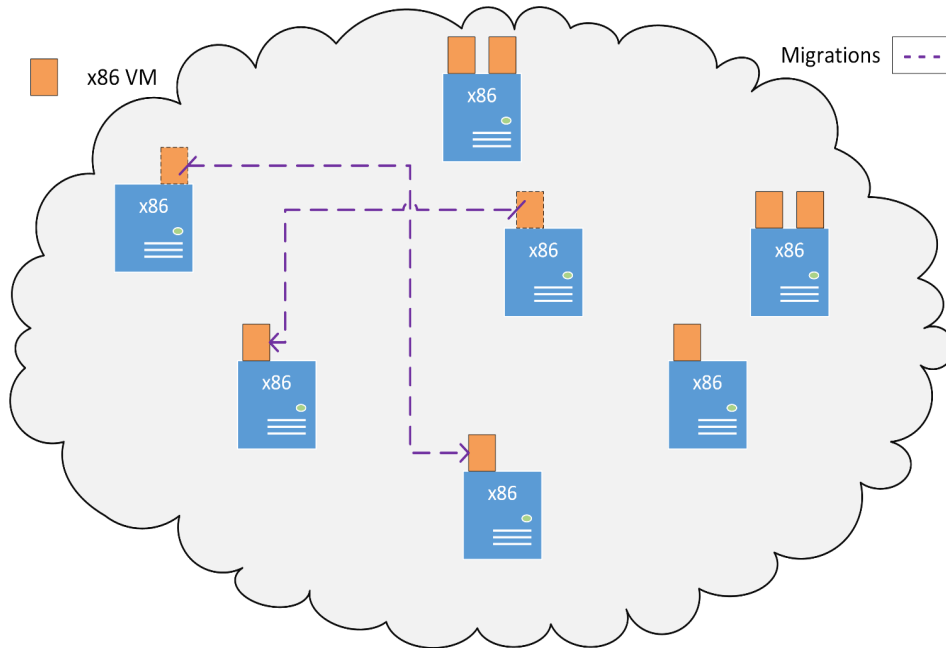


Fig 3.5: Traditional Cloud Environment

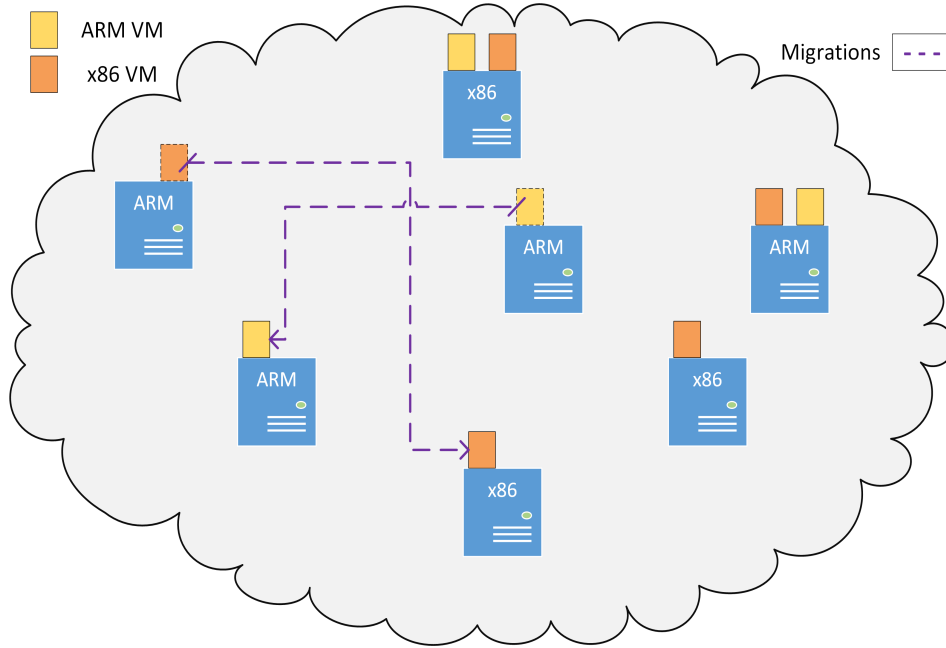


Fig 3.6: Proposed Heterogeneous Cloud Environment

Openstack works out of the box with Libvirt and QEMU on most of the hardware platforms but do not support heterogeneity. Openstack do not have support for embedded hardware yet but it does support some ARM based hosts with standard hardware such as VGA or PCI bus. Whereas most of embedded hardware, such as ODROID-XU4, doesn't have any PCI or VGA support from hardware and Openstack doesn't work out of the box on such boards. Additionally, the available options for configuring a virtual machine are very limited or in other words are not yet supported due to standardization based on the host capabilities. However, it can easily be modified and integrated according to the needs. This further lead us to understand the internal of how Openstack works and what changes or modifications are required in order to deploy a heterogeneous cloud on ODROID-XU4 nodes.

After mapping and understanding the responsibilities of each components of Open-Stack and how it delegate requests around the system we came up with a solution of writing custom wrappers to make it adjust to our needs and requirements. Once we

have wrapper in place we have our heterogeneous cloud running on ODROID-XU nodes and virtual machines can easily be managed via Openstack command line on demand. We have successfully configured and deployed a heterogeneous cloud using OpenStack components to manage virtual machines and system resources. Our heterogeneous cloud is configured in such a way that it takes full advantage of high end features of OpenStack cloud computing system such as live migration of virtual machines across nodes, web-based dashboard also known as Horizon [29] and others.

3.7 Other Models

3.7.1 Apache Spark

It is an open source in-memory cluster computing framework.

Apache Spark has following features.

- **Speed** Spark helps to run an application in Hadoop cluster, up to 100 times faster in memory, and 10 times faster when running on disk. This is possible by reducing number of read/write operations to disk. It stores the intermediate processing data in memory.
- **Supports multiple languages** Spark provides built-in APIs in Java, Scala, or Python. Therefore, you can write applications in different languages. Spark comes up with 80 high-level operators for interactive querying.
- **Advanced Analytics** Spark not only supports Map and reduce. It also supports SQL queries, Streaming data, Machine learning (ML), and Graph algorithms.

Figure 3.7 shows our experimental setup for running Apache Spark using ARM nodes with Apache hadoop.

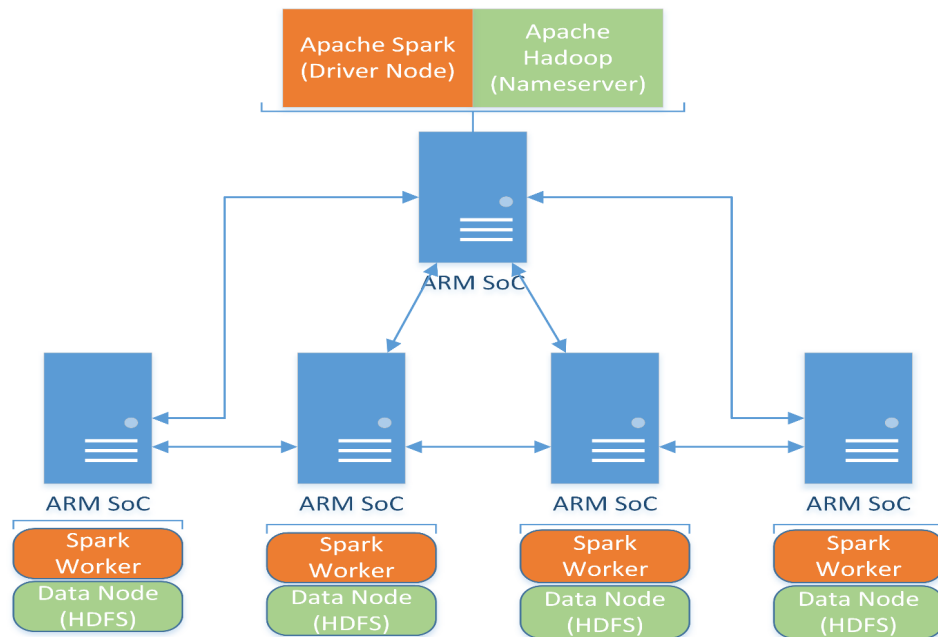


Fig 3.7: Experimental setup for Apache Spark and Apache Hadoop

3.7.2 Apache Spark benchmarking

We ran three benchmarks on our Apache Spark setup given below and collected their Job Execution Time and throughput.

- Hive-SQL
- RDD Relation SQL
- KMeans

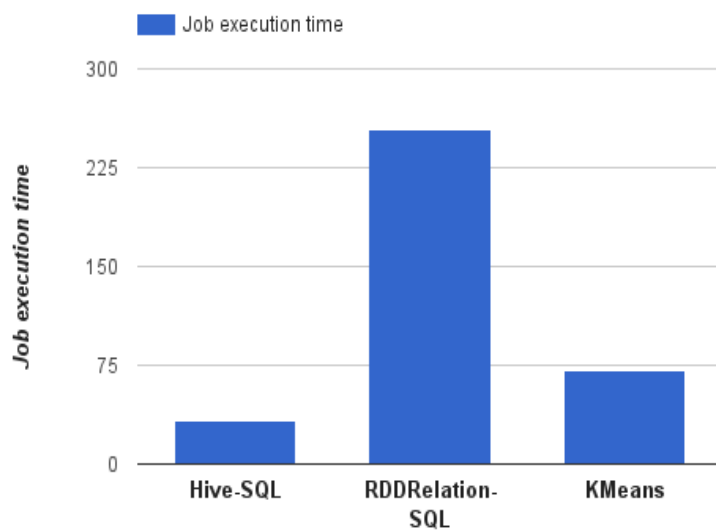


Fig 3.8: Job Execution Time

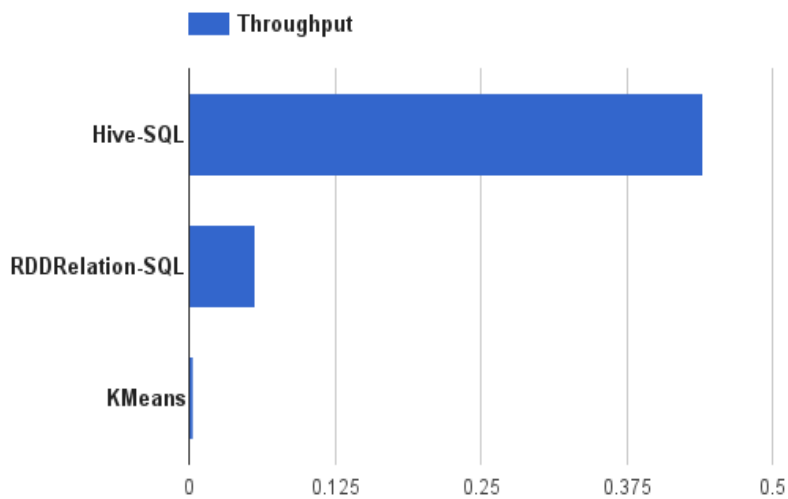


Fig 3.9: Average Throughput

Chapter 4

Building Heterogeneous Cloud

4.1 Introduction

This chapter provides the overview of what is required to build a cloud environment and how the idea can be extended in order to support heterogeneity in cloud. We are interested in IaaS , Infrastructure as a Service, cloud as one can easily scale from this to other cloud models such as PaaS, SaaS or DaaS. There are many vendors in the market providing cloud infrastructure and cloud services such as AWS, Azure, Google Cloud, Openstack etc.

The idea is to :

- To provide a unified cloud platform for VMs to share and take advantage of different architecture.
- To provide on demand live migration of VMs to low power consuming ARM hosts.
- Achieve efficient power and resource consumption by migrating x86 VMs to DC powered ARM hosts when idle or on demand and vice-versa.

As we already have our ODROID-XU4 running ARM and x86 based VMs using Libvirt with QEMU as a hypervisor. The next step is to install Libvirt and QEMU on our x86

hosts running Ubuntu and then spawn instances on demand. The idea here is to provide a unified platform for VMs to migrate between hosts irrespective of their architecture. For example, ability to run a x86 VM on x86 host and then migrate it to ARM based host without interrupting the VM execution. This will result in efficient power consumption by migrating x86 VMs to DC powered ARM hosts when idle or on demand. Additionally, supporting heterogeneity by sharing VMs across two different set of CPU architectures and hardware constraints.

4.2 Openstack

To carry out our proposed work, we chose to extend the OpenStack. Openstack is implemented as a set of python scripts and services, and has many components which interact with each other via message queue and database.

- It is a free and open source platform for cloud computing mostly deployed as IaaS.
- Fully distributed and modular architecture.
- Scale horizontally using commodity hardware.
- Role based access control.
- It's a 'Linux of Cloud Computing'.
- Large community and Industry support.

Lets take a look at Openstack Cloud Architecture and its components:

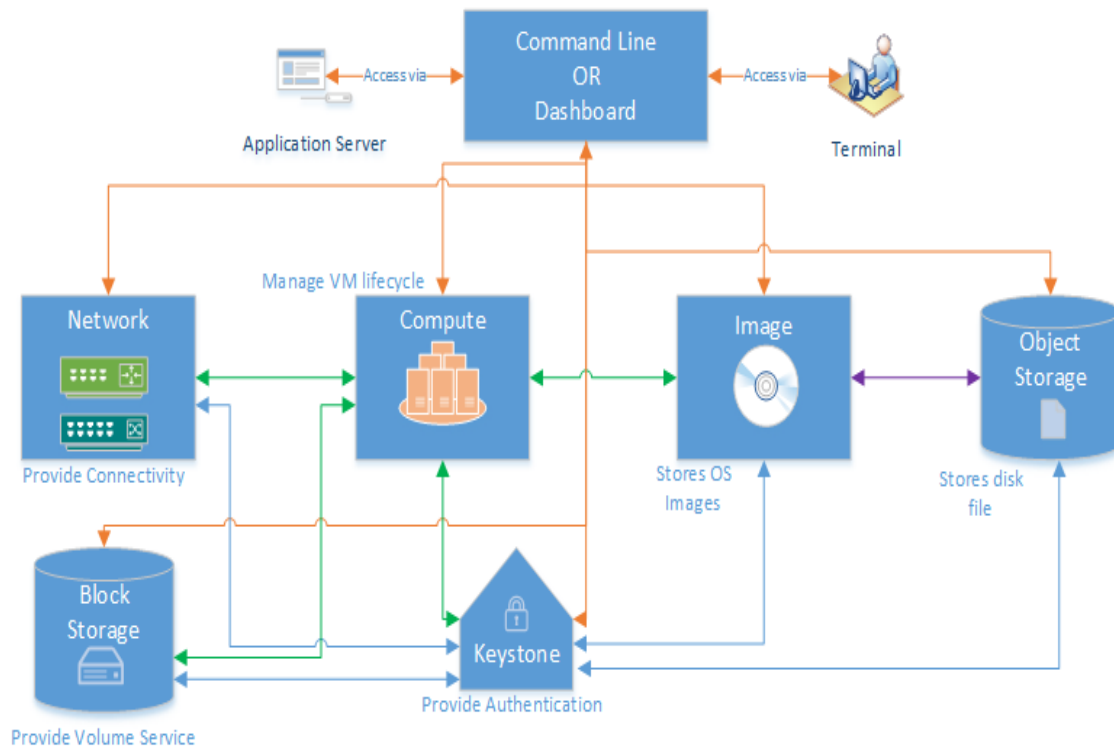


Fig 4.1: Openstack Architecture

Dashboard provides a UI , User Interface, for managing resources and handling requests. Identity service provides authentication for users and services. Network manages internal and external network interfaces. Image provides service for uploading disk images. Compute is the heart of Openstack which manages life cycle of VMs. Object Storage provides services to Image service to store and retrieve OS images whereas Block Storage provides volume for VMs.

In our case, all services works out of box and can be used as it is except Compute, also known as Openstack Nova . Nova consists Nova-API, Nova-Scheduler, Nova-Conductor, Nova-Compute and Nova-Network.

4.2.1 Openstack Nova

In order to understand how Openstack Compute works let's look at the figure below:

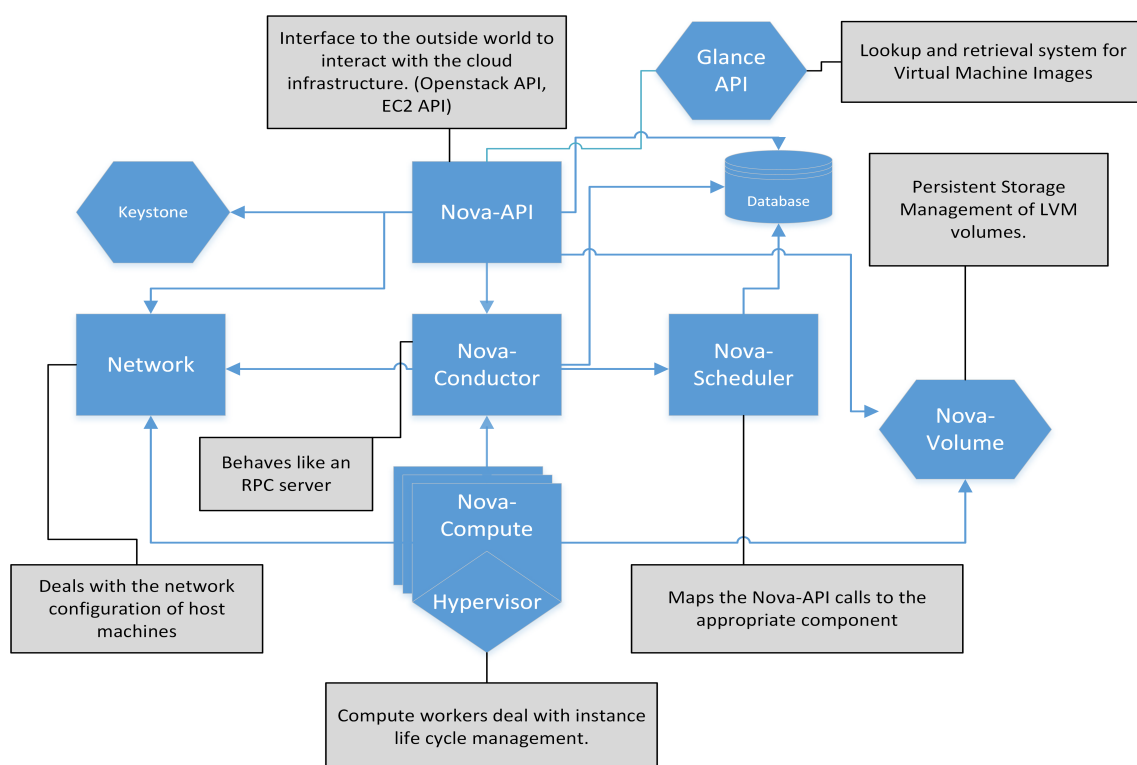


Fig 4.2: Openstack Nova Architecture

Nova-API: This is responsible for interfacing with the outside world and getting requests from users.

Nova-Network: Nova network service provide IP addresses and VLANs for the requested VM instances.

Nova-Scheduler: This is responsible for scheduling compute resources and finding valid host to deploy VM for incoming request.

Nova-Compute: This is responsible for managing VM life cycle on compute nodes.

Queue Server: It is a message queue and works as glue between different components and provides communication mechanism among different components.

Nova Execution Flow

Nova-API is the interfacing module of Openstack Nova component, shown as Compute in Openstack architecture above. It is responsible for fielding user request and delegating it to other nova components after proper authentication using Keystone services. Once authenticated it further delegates the task to Nova-Conductor which then ask Nova-Scheduler to find a valid host to accommodate the incoming request. Nova-Scheduler then find the best host based on the user requirement and provides the information back to Nova-Conductor. Nova-Conductor then passes VM metadata to that specific host and tells Nova-Compute running on that host to boot the VM. Nova-Compute then asks for IP address from Nova-Network or Neutron and collects the OS image from Glance service and VM storage device from Cinder or other services and boots up the VM. The communication between all components are carried out using message queue or by reading and writing to database. Following diagram provides the overall interaction of Openstack Nova-Compute:

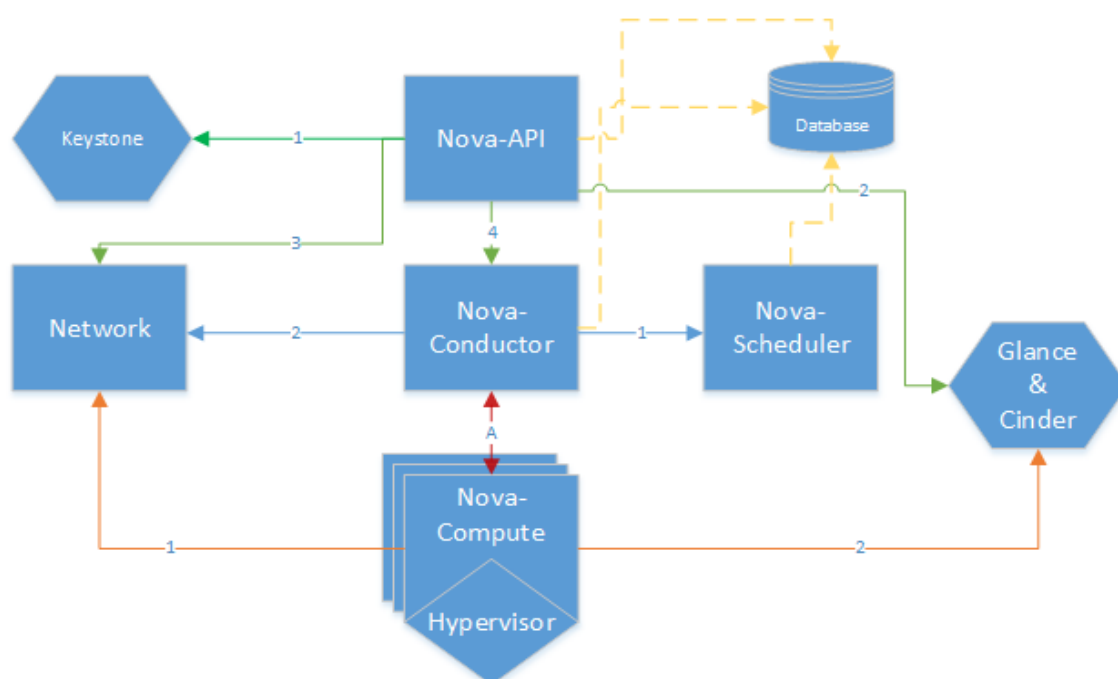


Fig 4.3: Openstack Nova execution flow.

The execution flow works as follow:

- Nova-API: Receives the user request either from Dashboard or Command line.
 1. Authenticates the user making the request.
 2. Validate if user is allowed to access the requested Image & Storage files.
 3. Validate if user is allowed to access the requested network.
 4. If all is good, forward the request to Nova-conductor.
- Nova-Conductor: It is just a RPC server. Receives the request from Nova-API.
 1. Request scheduler to find a valid and best compute node to serve the request.
 2. Setup the requested network.
- Nova-Compute: Receives the request to setup the VM.
 1. Request the IP address before booting the VM.
 2. Request the required Image & Storage from Glance or Cinder services to boot VM.
 3. Forwards it to hypervisor, boot up the VM and manage its life cycle.

Openstack does not support heterogeneity i.e. one can only run host-architecture based virtual machines. For example: Nova-Compute running on x86 host will only facilitate x86 based VM. Openstack do support ARM based VMs but requires hosts to must have PCI and VGA buses available and exposed to their hypervisor. In our embedded board, ODROID-XU4, we lack such hardware as it is a very small commodity hardware and cannot support standard hardware such as PCI bus and etc. Further, Openstack does not support migration of VMs between two host with different ISA or if any of the hardware mentioned or used in VM on source machine is not available in target machine.

Table 4.1: Openstack Mitaka Configuration (for minimal setup)

Service Type	Controller Node	Compute Node
Identity Service	Keystone	NA
Image Service	Glance	NA
Compute Service	Nova Management	Nova-Compute (hypervisor)
Network Service	Neutron	Nova-Network or Neutron
Database	MySQL or MariaDB	NA
Message Broker	RabbitMQ	NA

However, In our case we are emulating almost everything using QEMU and managing our VMs through Libvirt and successfully migrating them by providing more details to Libvirt and QEMU. For example: QEMU can be configured to emulate VGA or to just ignore it when requested by the user if VGA support is not available for the target board type.

After reading Openstack Nova source code intensively we mapped our requirements and came up with an idea of modifying Nova-Compute at Hypervisor level to support heterogeneity we proposed and proved earlier using Libvirt and QEMU only. This lead us to develop fully operational Openstack cloud which can issue x86 instances on ARM host as well facilitates live migration from ARM host to x86 host and vice versa. We used Openstack Mitaka release [13] to build our proposed cloud, and it require one controller and one or more compute nodes. Table 4.1 provides services required on each node type. Openstack Mitaka :

- Controller Node :

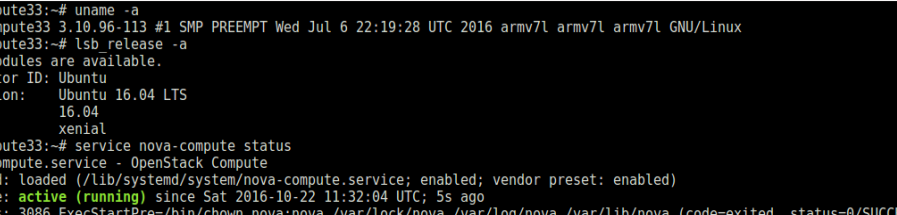
```

root@controller:~# uname -a
Linux controller 4.4.0-38-generic #57-Ubuntu SMP Tue Sep 6 15:42:33 UTC 2016 x86_64 x86_64 x86_64 GNU/Linux
root@controller:~# lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 16.04.1 LTS
Release:        16.04
Codename:       xenial
root@controller:~# nova-manage --version
Option "logdir" from group "DEFAULT" is deprecated. Use option "log-dir" from group "DEFAULT".
Option "verbose" from group "DEFAULT" is deprecated for removal. Its value may be silently ignored in the future.
13.1.0
root@controller:~# openstack service list
+-----+-----+-----+
| ID | Name | Type |
+-----+-----+-----+
| 51cb5920e3864713b0ce382b41877f6c | neutron | network |
| 5512a80bb45a9bb62de4f7a6d72e | identity | identity |
| 95e0f434095e4f588091dca2f84d7ab8 | nova | compute |
| f9183a0da0f144eea2789e8adb56e749 | glance | image |
+-----+-----+-----+
root@controller:~# nova service-list
+-----+-----+-----+-----+-----+-----+-----+-----+
| Id | Binary | Host | Zone | Status | State | Updated_at | Disabled Reason |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 4 | nova-consoleauth | controller | internal | enabled | up | 2016-10-22T11:18:46.000000 | - |
| 5 | nova-scheduler | controller | internal | enabled | up | 2016-10-22T11:18:49.000000 | - |
| 6 | nova-conductor | controller | internal | enabled | up | 2016-10-22T11:18:49.000000 | - |
| 7 | nova-compute | compute33 | nova | enabled | up | 2016-10-22T11:18:53.000000 | - |
| 8 | nova-compute | compute22 | nova | enabled | up | 2016-10-22T11:18:56.000000 | - |
| 9 | nova-network | compute22 | internal | enabled | down | 2016-09-01T04:14:36.000000 | - |
| 10 | nova-compute | compute44 | nova | enabled | down | 2016-09-01T12:13:39.000000 | - |
| 11 | nova-compute | computex44 | nova | enabled | up | 2016-10-22T11:18:47.000000 | - |
+-----+-----+-----+-----+-----+-----+-----+-----+
root@controller:~# nova hypervisor-list
+-----+-----+-----+-----+
| ID | Hypervisor | hostname | State | Status |
+-----+-----+-----+-----+
| 1 | compute33 | | up | enabled |
| 2 | compute22 | | up | enabled |
| 3 | compute44 | | down | enabled |
| 5 | computex44 | | up | enabled |
+-----+-----+-----+-----+
root@controller:~#

```

Fig 4.4: Controller Node running on x86 Ubuntu 16.04 LTS

- Compute Node (ODROID-XU4) :



The screenshot shows a terminal window with three tabs: 'root@compute33: ~', 'root@compute33: ~', and 'root@compute33: ~'. The active tab is the first one. The terminal output is as follows:

```

root@compute33:~# uname -a
Linux compute33 3.10.96-113 #1 SMP PREEMPT Wed Jul 6 22:19:28 UTC 2016 armv7l armv7l armv7l GNU/Linux
root@compute33:~# lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 16.04 LTS
Release:       16.04
Codename:      xenial
root@compute33:~# service nova-compute status
● nova-compute.service - OpenStack Compute
   Loaded: loaded (/lib/systemd/system/nova-compute.service; enabled; vendor preset: enabled)
   Active: active (running) since Sat 2016-10-22 11:32:04 UTC; 5s ago
     Process: 3086 ExecStartPre=/bin/chown nova:nova /var/lock/nova /var/log/nova /var/lib/nova (code=exited, status=0/SUCCESS)
     Process: 3083 ExecStartPre=/bin/mkdir -p /var/lock/nova /var/log/nova /var/lib/nova (code=exited, status=0/SUCCESS)
    Main PID: 3089 (nova-compute)
      CGroup: /system.slice/nova-compute.service
              └─3089 /usr/bin/python /usr/bin/nova-compute --config-file=/etc/nova/nova.conf --log-file=/var/log/nova/nova-compute.log

Oct 22 11:32:04 compute33 systemd[1]: Starting OpenStack Compute...
Oct 22 11:32:04 compute33 systemd[1]: Started OpenStack Compute.
root@compute33:~# hostname
compute33
root@compute33:~#

```

Fig 4.5: Compute Node running on ARM based Ubuntu 16.04 LTS

- Compute Node (x86) :

```

root@computex44:~# uname -a
Linux computex44 4.4.0-38-generic #57-Ubuntu SMP Tue Sep 6 15:42:33 UTC 2016 x86_64 x86_64 x86_64 GNU/Linux
root@computex44:~# lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 16.04.1 LTS
Release:        16.04
Codename:       xenial
root@computex44:~# service nova-compute status
nova-compute start/running, process 2083
root@computex44:~#

```

Fig 4.6: Compute Node running on x86 Ubuntu 16.04 LTS

To issue an instance we need to pick OS Image, Flavor and Network.

```

root@controller:~# nova hypervisor-list
+-----+
| ID | Hypervisor hostname | State | Status |
+-----+
| 1  | compute33           | up    | enabled |
| 2  | compute22           | up    | enabled |
| 3  | compute44           | down  | enabled |
| 5  | computex44          | up    | enabled |
+-----+

root@controller:~# nova flavor-list
+-----+
| ID | Name | Memory_MB | Disk | Ephemeral | Swap | VCPUs | RXTX_Factor | Is_Public |
+-----+
| 0  | custom | 512 | 5 | 0 | | 1 | 1.0 | True |
| 1  | m1.tiny | 512 | 1 | 0 | | 1 | 1.0 | True |
| 2  | m1.small | 2048 | 20 | 0 | | 1 | 1.0 | True |
| 3  | m1.medium | 4096 | 40 | 0 | | 2 | 1.0 | True |
| 4  | m1.large | 8192 | 80 | 0 | | 4 | 1.0 | True |
| 5  | m1.xlarge | 16384 | 160 | 0 | | 8 | 1.0 | True |
| 7fdf355a-1f8a-49f6-8537-3f954ab97558 | x86-custom-81 | 512 | 8 | 0 | | 1 | 1.0 | True |
| d6c04f86-00de-4efe-95a1-a8894bc5bc42 | x86-custom-82 | 512 | 8 | 0 | | 2 | 1.0 | True |
| dc8a855d-95fa-4c4d-a014-1d7d2bb680ae | x86-custom | 1024 | 20 | 0 | | 1 | 1.0 | True |
+-----+

root@controller:~# nova image-list
+-----+
| ID | Name | Status | Server |
+-----+
| 233c3206-4d52-4dd5-bff7-9ffda9f77f19 | ubu-arm-img | ACTIVE | |
| 0dfe13a9-be60-4938-af55-8e5a55c97ac4 | ubu-arm-kernel | ACTIVE | |
| b35723c5-b793-4a81-9a59-94a47805bd7e | ubux86 | ACTIVE | |
+-----+

```

Fig 4.7: Available Compute Nodes, Flavor-list and Glance-Images on Controller

Once decided, we can run an instance on a specific compute Node, as shown in Figure 4.8, by using option '–availability-zone' with nova.

```

root@controller:~# nova hypervisor-list
+-----+-----+-----+-----+
| ID | Hypervisor hostname | State | Status |
+-----+-----+-----+-----+
| 1 | compute33           | up    | enabled |
| 2 | compute22           | up    | enabled |
| 3 | compute44           | down  | enabled |
| 5 | computex44          | up    | enabled |
+-----+-----+-----+-----+
root@controller:~# nova boot --flavor x86-custom-81 --image b35723c5-b793-4a81-9a59-94a47805bd7e --key-name mykey --availability-zone ZON
E:computex44 x86-001

```

Fig 4.8: Boot command to issue VM to a specific host

To check the status of VM, one can use 'nova list' command as shown in Figure 4.9.

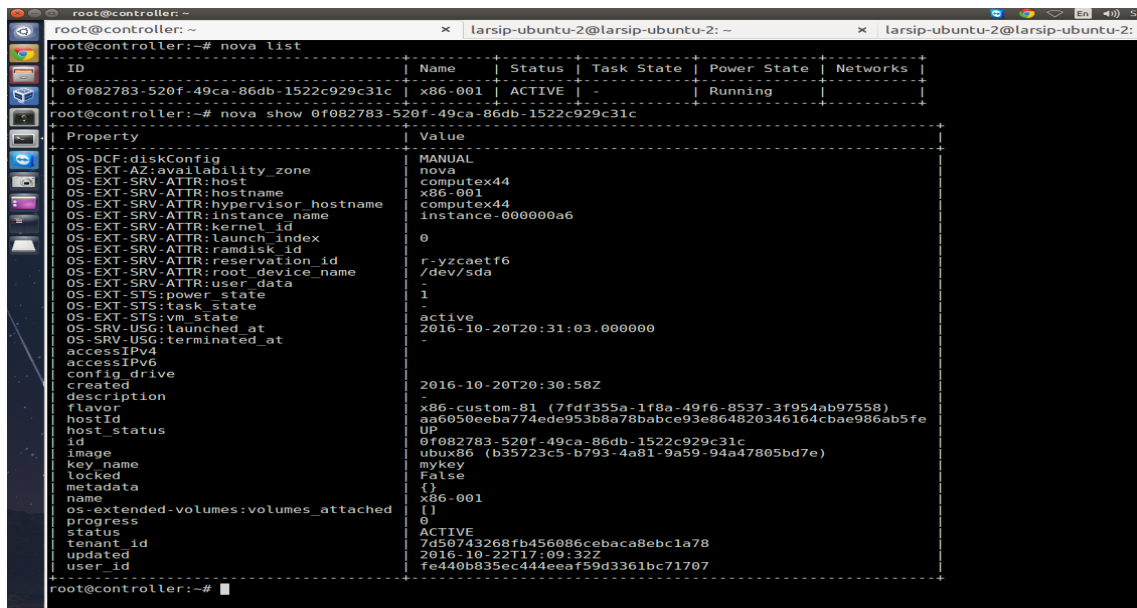
```

root@controller:~# nova list
+-----+-----+-----+-----+-----+-----+
| ID | Name | Status | Task State | Power State | Networks |
+-----+-----+-----+-----+-----+-----+
| 0f082783-520f-49ca-86db-1522c929c31c | x86-001 | ACTIVE | - | Running | |
+-----+-----+-----+-----+-----+-----+
root@controller:~#

```

Fig 4.9: 'nova list' command to list all the VMs and their status

'nova-show' command can be used to see more details, as shown in figure 4.10, about the instance such as its status, host its running on, image used etc.



```

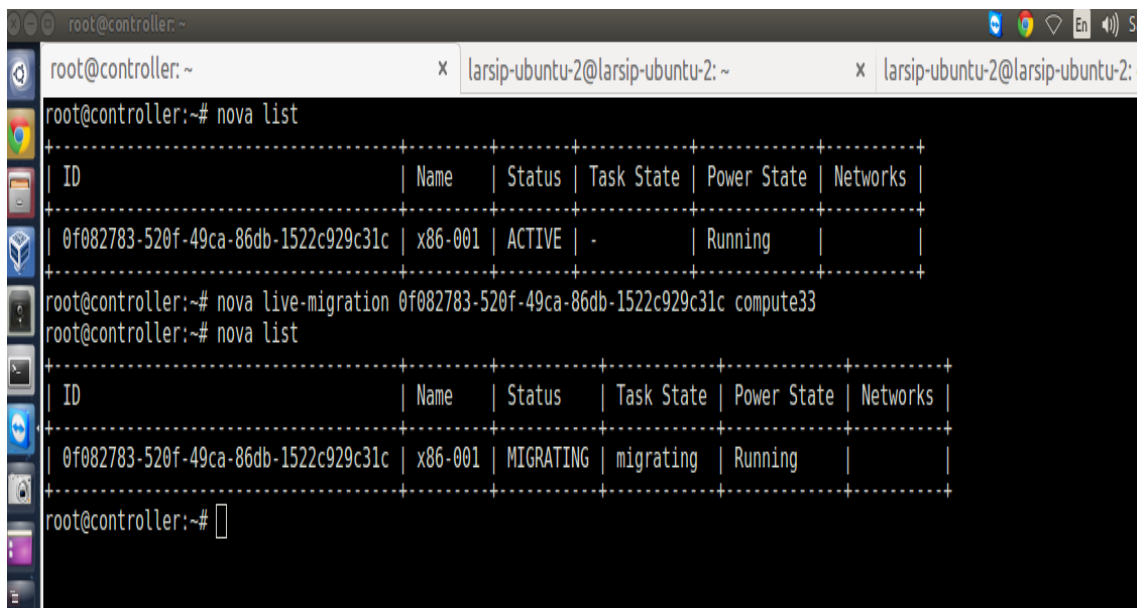
root@controller:~# nova list
+-----+-----+-----+-----+-----+-----+
| ID | Name | Status | Task State | Power State | Networks |
+-----+-----+-----+-----+-----+-----+
| 0f082783-520f-49ca-86db-1522c929c31c | x86-001 | ACTIVE | - | Running | - |
+-----+-----+-----+-----+-----+-----+

root@controller:~# nova show 0f082783-520f-49ca-86db-1522c929c31c
+-----+-----+
| Property | Value |
+-----+-----+
| OS-DCF:diskConfig | MANUAL |
| OS-EXT-SRV-ATTR:host | nova |
| OS-EXT-SRV-ATTR:instance_name | compute44 |
| OS-EXT-SRV-ATTR:kernel_id | x86-001 |
| OS-EXT-SRV-ATTR:launch_index | 0 |
| OS-EXT-SRV-ATTR:ramdisk_id | r-yzcaetf6 |
| OS-EXT-SRV-ATTR:reservation_id | /dev/sda |
| OS-EXT-SRV-ATTR:user_data | - |
| OS-EXT-SIS:power_state | 1 |
| OS-EXT-SIS:task_state | - |
| OS-EXT-SIS:vm_state | active |
| OS-SRV-USG:launched_at | 2016-10-20T20:31:03.000000 |
| OS-SRV-USG:terminated_at | - |
| accessIPv4 | - |
| accessIPv6 | - |
| config_drive | 2016-10-20T20:30:58Z |
| created | - |
| description | - |
| flavor | x86-custom-81 (7fdf355a-1f8a-49f6-8537-3f954ab97558) |
| hostId | aa6050eeba774ede953b5a78babce93e864820346164cbae986ab5fe |
| host status | UP |
| id | 0f082783-520f-49ca-86db-1522c929c31c |
| image | ubux86 (b35723c5-b793-4a81-9a59-94a47805bd7e) |
| key name | mykey |
| locked | False |
| metadata | {} |
| name | x86-001 |
| os-extended-volumes:volumes_attached | [] |
| progress | 0 |
| status | ACTIVE |
| tenant_id | 7d50743268fb456086cebac8ebc1a78 |
| updated | 2016-10-22T17:09:32Z |
| user_id | fe440b835ec444eeaf59d3361bc71707 |
+-----+-----+

```

Fig 4.10: Output of 'nova-show' command to show the details of VM.

In order to live migrate the shown VM to ARM based ODROID-XU4 node, we will use command 'nova live-migration instance_id host_name'. As we showed in last image the VM is running on Computex44, which is x86 based host, will now be migrated to ARM based host i.e. compute33.



```

root@controller:~# nova live-migration 0f082783-520f-49ca-86db-1522c929c31c compute33
root@controller:~# nova list
+-----+-----+-----+-----+-----+-----+
| ID | Name | Status | Task State | Power State | Networks |
+-----+-----+-----+-----+-----+-----+
| 0f082783-520f-49ca-86db-1522c929c31c | x86-001 | ACTIVE | - | Running | - |
+-----+-----+-----+-----+-----+-----+

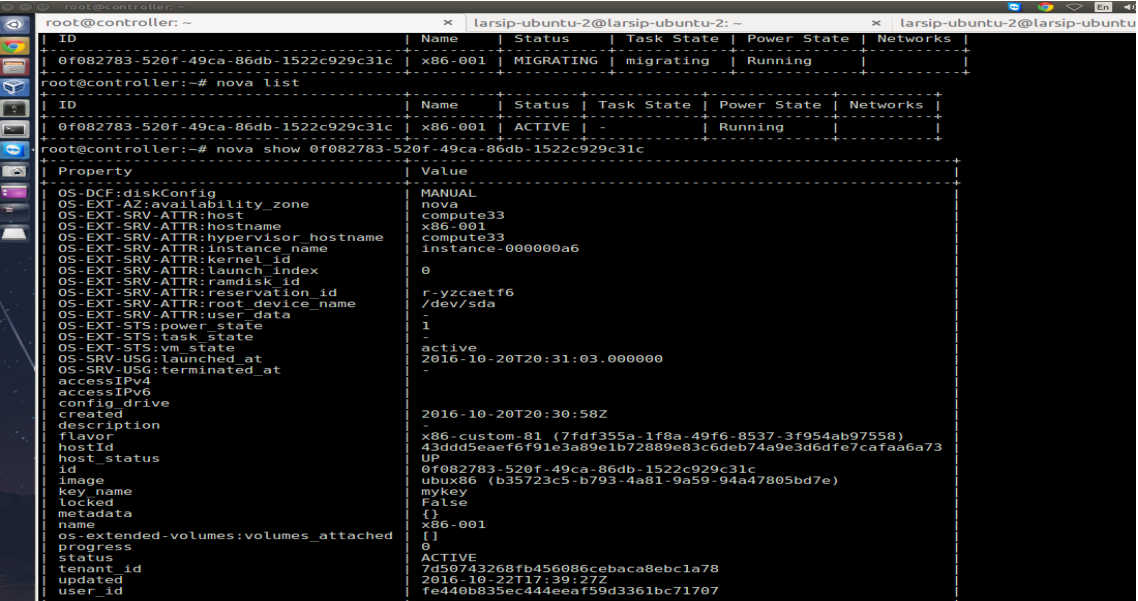
root@controller:~# nova live-migration 0f082783-520f-49ca-86db-1522c929c31c compute33
root@controller:~# nova list
+-----+-----+-----+-----+-----+-----+
| ID | Name | Status | Task State | Power State | Networks |
+-----+-----+-----+-----+-----+-----+
| 0f082783-520f-49ca-86db-1522c929c31c | x86-001 | MIGRATING | migrating | Running | - |
+-----+-----+-----+-----+-----+-----+

root@controller:~#

```

Fig 4.11: Initiating live-migration of VM from x86 based host to ARM based host

Next figure shows the success of live migration as the host name changes to compute33.



```

root@controller: ~
x larsip-ubuntu-2@larsip-ubuntu-2: ~
x larsip-ubuntu-2@larsip-ubuntu-2: ~

ID | Name | Status | Task State | Power State | Networks |
---|---|---|---|---|---|
0f082783-520f-49ca-86db-1522c929c31c | x86-001 | MIGRATING | migrating | Running | |

root@controller:~# nova list
+-----+-----+-----+-----+-----+-----+
ID | Name | Status | Task State | Power State | Networks |
+-----+-----+-----+-----+-----+-----+
0f082783-520f-49ca-86db-1522c929c31c | x86-001 | ACTIVE | - | Running | |
+-----+-----+-----+-----+-----+-----+

root@controller:~# nova show 0f082783-520f-49ca-86db-1522c929c31c
+-----+-----+
Property | Value
+-----+-----+
OS-DCF:diskConfig | MANUAL
OS-EXT-AZ:availability_zone | nova
OS-EXT-SRV-ATTR:host | compute33
OS-EXT-SRV-ATTR:hostname | x86-001
OS-EXT-SRV-ATTR:hypervisor_hostname | compute33
OS-EXT-SRV-ATTR:instance_name | instance-000000a6
OS-EXT-SRV-ATTR:kernel_id | 0
OS-EXT-SRV-ATTR:launch_index | 0
OS-EXT-SRV-ATTR:ramdisk_id | r-yzcaetf6
OS-EXT-SRV-ATTR:reservation_id | /dev/sda
OS-EXT-SRV-ATTR:user_data | 1
OS-EXT-STS:power_state | active
OS-EXT-STS:task_state | -
OS-EXT-STS:vm_state | 2016-10-20T20:31:03.000000
OS-SRV-USG:launched_at | -
OS-SRV-USG:terminated_at | -
accessIPv4 | -
config_drive | 2016-10-20T20:30:58Z
created | -
description | -
flavor | x86-custom-81 (7fdf355a-1f8a-49f6-8537-3f954ab97558)
hostId | 43ddd5eae76f91e3a89e1b72889e83c6deb74a9e3d6dfe7cafaa6a73
host status | UP
id | 0f082783-520f-49ca-86db-1522c929c31c
image | ubuX86 (b35723c5-b793-4a81-9a59-94a47805bd7e)
key name | mykey
locked | False
metadata | {}
name | x86-001
os-extended-volumes:volumes_attached | []
progress | 0
status | ACTIVE
tenant_id | 7d50743268fb456086cebaca8ebc1a78
updated | 2016-10-22T17:39:27Z
user_id | fe4408835ec444eeaf59d3361bc71707

```

Fig 4.12: Successfully migrated VM

Chapter 5

Experiments & Results

5.1 Introduction

In this chapter we present a measurement driven assessment of the proposed experiments. In section 5.2 we evaluate the performance of CPU intensive workloads. Section 5.3 provides results for Query Processing benchmarks such as TPC-C & TPC-H

5.2 Dhrystone and Whetstone

We have made preliminary performance tests by running Dhrystone and Whetstone benchmarks on these boards. Further, these benchmarks have provided us with a vision as to

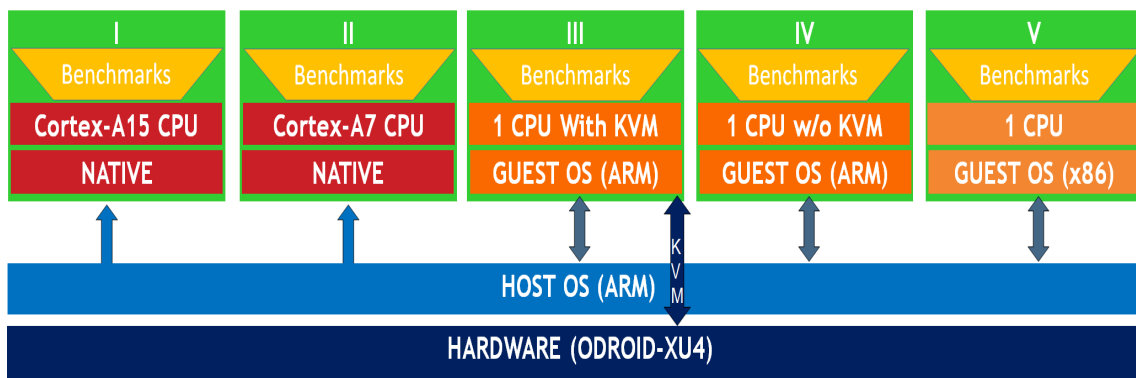


Fig 5.1: General use cases for our performance analysis.

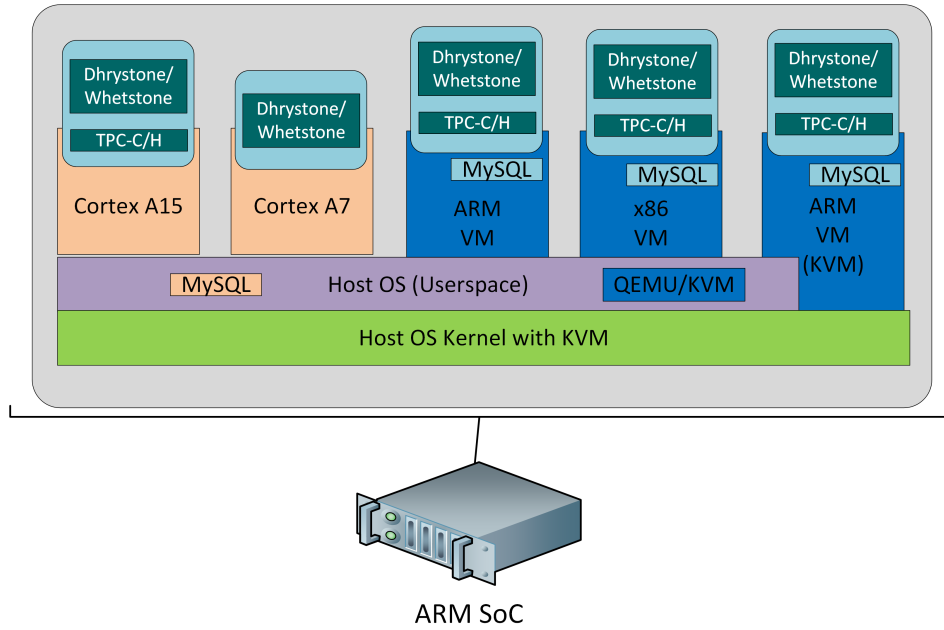


Fig 5.2: Overview of experimental setup for benchmarking.

what we can expect in terms of running virtualized solutions on these small boards. We further divided our work based on the placement of benchmark under different scenarios as shown in Figure 5.1 Such as running:

- Benchmarks natively and pinned to one Cortex-A15 core.
- Benchmarks natively and pinned to one Cortex-A7 core.
- Benchmarks in a Ubuntu-armhf VM running on top of host OS using one CPU with KVM.
- Benchmarks in a Ubuntu-armhf VM running on top of host OS using one CPU but without KVM extensions.
- Benchmarks in a Ubuntu-x86 VM running on top of host OS using one CPU.

Our experimental setup is showed in Figure 5.2. As Dhrystone's small size allows it to easily fit inside processor caches but after adding few thousand loops the score remains

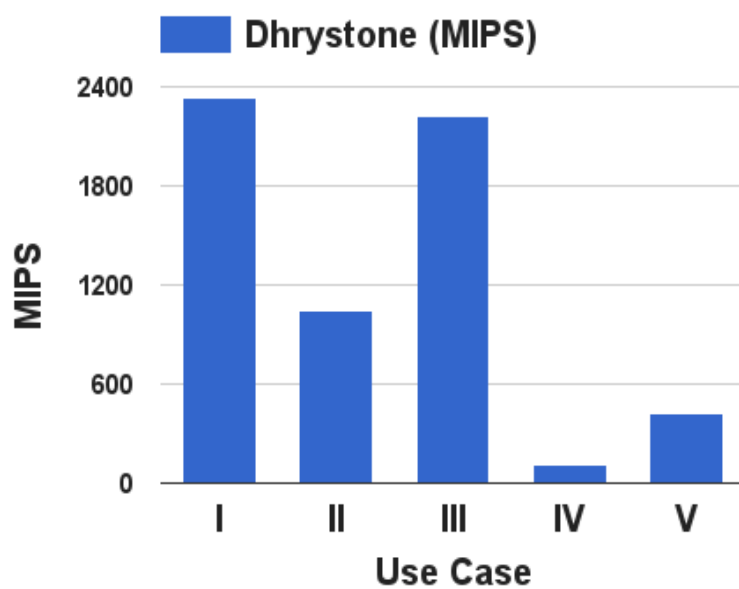


Fig 5.3: Dhrystone benchmark results.

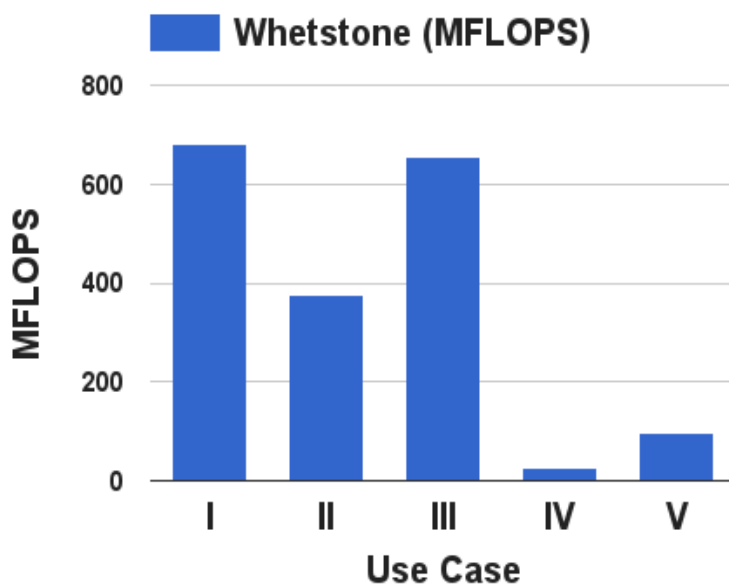


Fig 5.4: Whetstone benchmark results.

constant and scales linearly for clock speed. We collected Dhrystone and Whetstone output by running it multiple time and progressively increasing the number of loops and loop count respectively. Results are shown for Dhrystone in Figure 5.3 and for Whetstone in Figure 5.4. These data suggests that the near native performance can be achieved in virtualized environment on such a specialized embedded hardware using KVM whereas standard x86 operating systems can also take advantage of performance to power trade-off and performance to cost trade-off for CPU-intensive applications.

5.3 Query Processing

Now that we have overall understanding of virtualization capabilities of our embedded board, and we have our VMs up and running on demand using QEMU command line, now its time to evaluate the performance of such VMs under specialized query processing benchmarks such as TPC-C and TPC-H.

5.3.1 MySQL

To show the performance of our query processing benchmarks we run TPC-C (OLTP) and TPC-H (OLAP) benchmarks on ODROID-XU4 on MySQL (ver 5.0.23) with mysql++ (ver 2.0.7). We compiled and installed MySQL on native Host OS as well as in our virtual machines (ARM & x86 based) from source using default configuration. Table 1 provides more information about input datasets used.

5.3.2 TPC-C & TPC-H Benchmark

TPC-C simulates an OLTP (Online Transactional Processing) type order entering environment where users executes transactions against database. It executes different transactions on the database such as delivery, new order, order status, payment and monitoring stock level at warehouses. The performance is measured in new-order transactions per

minute (tpm). TPC-C tends to measure maximum sustained system performance. For example, a system with 40 tpm is generating 40 New-Order transaction per minute while fulfilling the rest of TPC-C transaction mix workload.

TPC-H simulates an OLAP (Online Analytical Processing) type decision support system against the database. It executes business oriented ad hoc queries as well issues concurrent data modification queries. It generates 8 database tables all in 3rd Normal Form. It can be run with pre-determined database sizes also referred to as "scale factors". Scale factor of 1 represent 1 Gigabyte of raw data size of data warehouse. 22 queries are available under TPC-H benchmark which can be further divided into scan-based, random-access and long running queries. For scan-based queries we used Q1, Q6, Q12, Q14, Q15, Q21. For random-access queries we used Q3, Q5, Q7, Q8, Q11, Q16, Q19, Q22. We conducted our query processing benchmark experiments for TPC-C and TPC-H (using scan-based and random-access queries) under five different use cases such as:

For TPC-C :

- I Running benchmarks directly on native host system with full access to hardware resources.
- II Running benchmarks directly on native host system with only one Cortex-A15 core assigned.

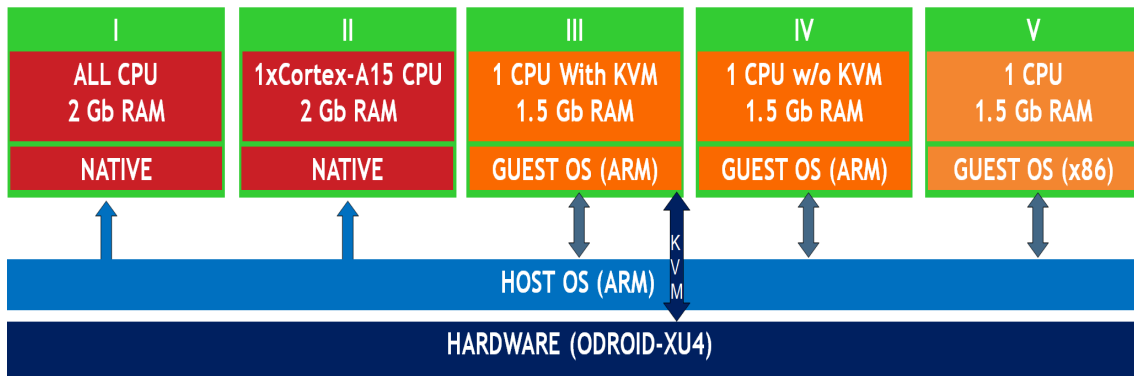


Fig 5.5: Use cases for TPC-C benchmark.

III Running benchmarks in a ARM based virtual machine running as guest OS using one CPU, 1.5GB RAM on top of host OS and KVM.

IV Running benchmarks in a ARM based virtual machine running as guest OS using one CPU, 1.5GB RAM on top of host OS but without KVM.

V Running benchmarks in a x86 based virtual machine running as guest OS using one CPU and 1.5GB RAM on top of host OS.

For TPC-H scan-based queries:

I Running benchmarks directly on native host system with full access to hardware resources.

II Running benchmarks directly on native host system with only one Cortex-A15 core assigned.

III Running benchmarks in a ARM based virtual machine running as guest OS using one CPU, 1GB RAM on top of host OS and KVM.

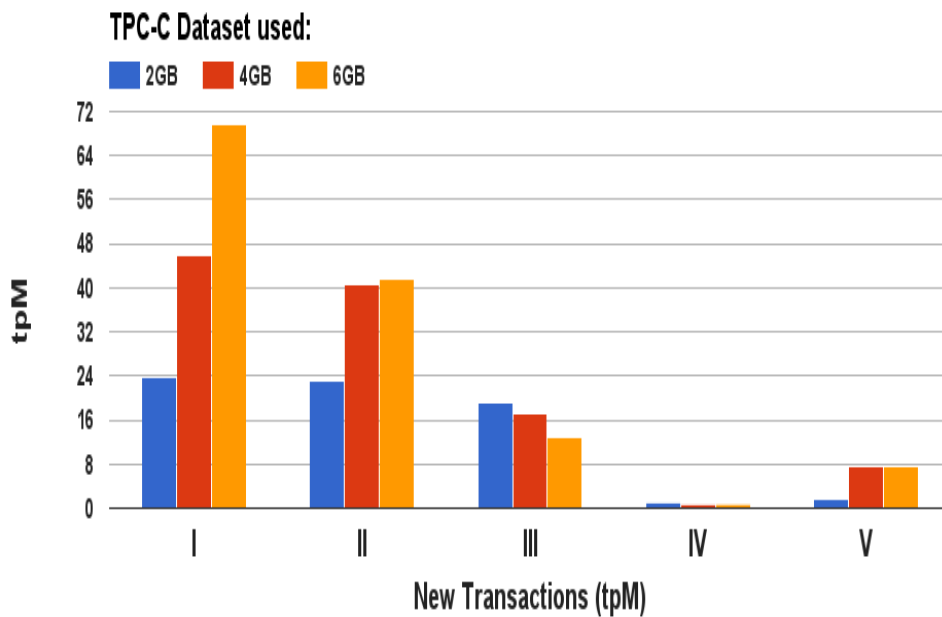


Fig 5.6: TPC-C benchmark results.

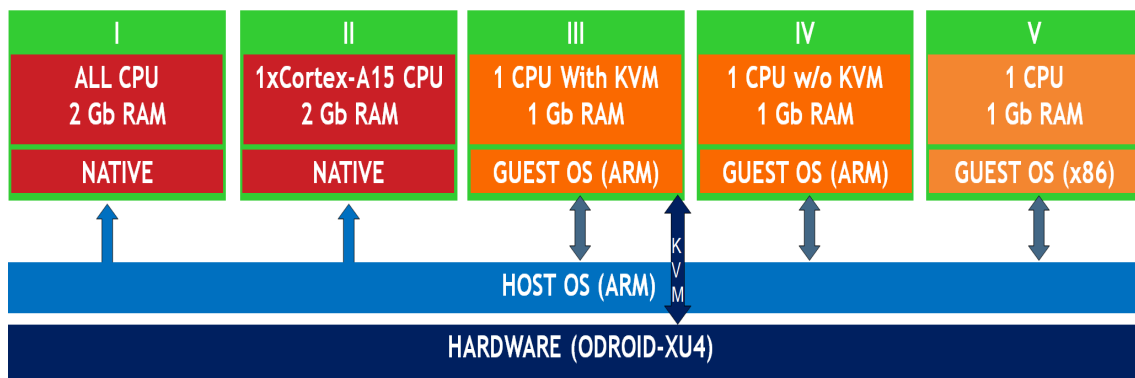


Fig 5.7: Use cases for TPC-H benchmark.

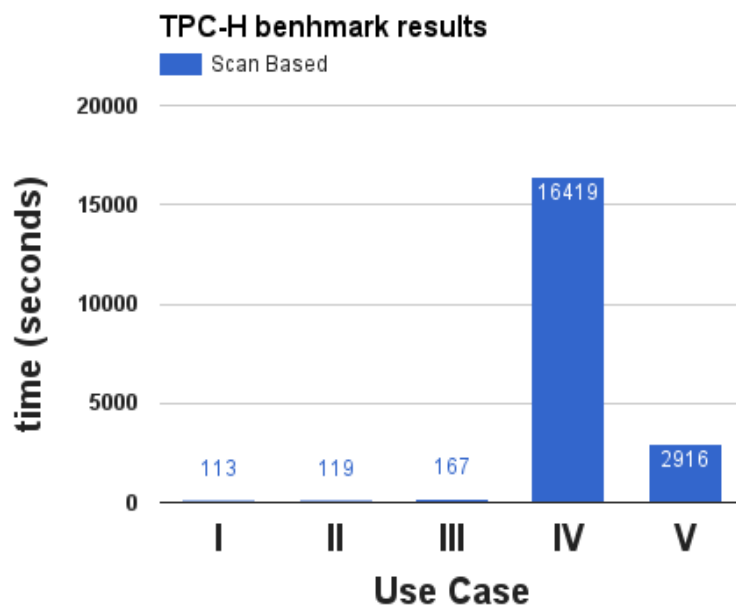


Fig 5.8: TPC-H benchmark results.

IV Running benchmarks in a ARM based virtual machine running as guest OS using one CPU, 1GB RAM on top of host OS but without KVM.

V Running benchmarks in a x86 based virtual machine running as guest OS using one CPU and 1GB RAM on top of host OS.

For TPC-H random-access based queries:

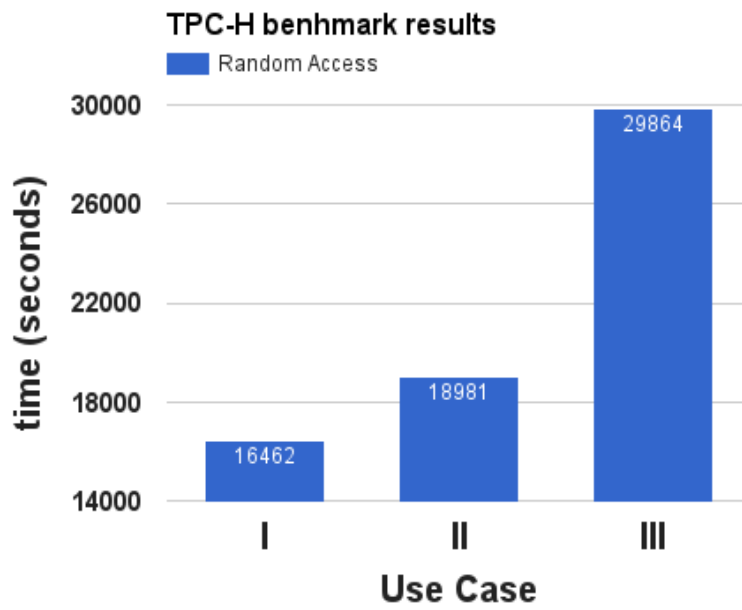


Fig 5.9: TPC-H benchmark results.

I Running benchmarks directly on native host system with full access to hardware resources.

Table 5.1: TPC-C & TPC-H dataset

Use Case	TPC-C(GB)	TPC-H Scan-based (scale factor)	TPC-H Random-access (scale factor)
I	2,4,6	0.5	0.5
II	2,4,6	0.5	0.5
III	2,4,6	0.5	0.5
IV	2,4,6	0.5	-
V	2,4,6	0.5	-

II Running benchmarks directly on native host system with only one Cortex-A15 core assigned.

III Running benchmarks in a ARM based virtual machine running as guest OS using one CPU, 1GB RAM on top of host OS and KVM.

We collected the TPC-C and TPC-H output for above mentioned cases. TPC-C results are shown in Figure 5.6. We ran scan based and random access queries in the mentioned use cases. Results for TPC-H workload, scan based queries is shown in Figure 5.8 and for random access based queries in Figure 5.9.

Chapter 6

Conclusion & Future Work

6.1 Introduction

The purpose of this chapter is to summarize the thesis research and provide research recommendations for further analysis. The first section of this chapter presents the conclusion. The second section will discuss some of the future scope of our work.

6.2 Conclusion

In this work, we have presented

- A novel heterogeneous cloud computing infrastructure paradigm using embedded ARM hardware and x86 hosts.
- A successful integration of our heterogeneous cloud with latest version of OpenStack Mitaka and run virtual machines via OpenStack command line. This further provides foundation to researchers to do more experimentation.
- A performance evaluation using MIPS, MFLOPS and tpM of well known Dhryhstone, Whetstone, TPC-C and TPC-H benchmarks .

- A comparison of the behavior of the system under different use cases. The results show that near native performance can be achieved for CPU intensive applications running in virtual environment.
- For Query processing TPC-C benchmark we achieved competitive results with KVM for 4 & 6 GB datasets and near native tpM with 2GB dataset.

Further, the data provides a strong ground to use these boards as a foundation for building more complex ARM based virtual infrastructure.

6.3 Future Work

This work has proved the feasibility of running virtual machine on the ARM based embedded boards using QEMU, thus incorporating high computation power at low cost and low power consumption rate.

- Our heterogeneous cloud could serve as a platform for setting up a cloud infrastructure for running on-demand virtual machine instances, application servers, or web-servers.
- It will open a whole new area to experiment and implement various freely and publicly available open source software such as OpenStack, Apache Hadoop or Spark and serve these ODROID-XU4 clusters as huge file servers or application servers.
- Our low power-consuming yet powerful hardware will be highly beneficial for communities/small to mid-sized organizations/institutions/research centers that do not have enough funds to deploy high grade cloud services at their own place.
- Our heterogeneous cloud provide benefit of on demand power and resource management which could easily be leveraged and used as a building block for fog computing i.e. providing on demand computing on the edge of network or in modern IoT architectures.

Bibliography

- [1] Michael F. Cloutier, Chad Paradis and Vincent M. Weaver, Design and Analysis of a 32-bit Embedded High-Performance Cluster Optimized for Energy and Performance 2014 Hardware-Software Co-Design for High Performance Computing,doi:10.1109/Co-HPC.2014.7.
- [2] Zonghua Gu, Qingling Zhao, A State-of-the-Art Survey on Real-Time Issues in Embedded Systems Virtualization, Journal of Software Engineering and Applications, 2012, 5, 277-290, <http://dx.doi.org/10.4236/jsea.2012.54033> Published Online April 2012.
- [3] Geoffrey Papaux, Daniel Gachet, and Wolfram Luithardt, Processor Virtualization on Embedded Linux Systems,IEEE doi: 10.1109/EDERC.2014.6924360.
- [4] Online Technical/Hardware reference for ODROID-XU Board, Retrieved Dec 12, 2016 from http://www.hardkernel.com/main/products/prdt_info.php?g_code=G137510300620
- [5] Ubuntu fork, ubuntu-14.04lts, for ODROID - XU, Retrieved Dec 12, 2016 from http://odroid.com/dokuwiki/doku.php?id=en:xu3_release_linux_ubuntu
- [6] QEMU download link, Retrieved Jan 18, 2016 from <https://en.wikibooks.org/wiki/QEMU>
- [7] QEMU user documentation and commands, Retrieved Jan 18, 2016 from <http://wiki.qemu.org/Download>.

- [8] Yasunori Goto, "KVM-Kernel-based Virtual Machine Technology".
- [9] Zhang, Yong, Beng, Dumitrel, Bogdan, A Performance Study of Big Data on Small Nodes, Department of CS, National University of Singapore.
- [10] Steve Crago, Kyle Dunn, Patrick Eads, Lorin Hochstein, Dong-In Kang, Mikyung Kang, Devendra Modium, Karandeep Singh, Jinwoo Suh, John Paul Walters, "Heterogeneous cloud computing", University of Southern California / Information Sciences Institute
- [11] T. P. P. Council, TPC-C benchmark specification, <http://www.tpc.org/tpcc/>
- [12] T. P. P. Council, TPC-H benchmark specification, <http://www.tpc.org/tpch/>
- [13] Openstack Mitaka installation, Retrieved Sep 03, 2016 from <http://docs.openstack.org/mitaka/install-guide-ubuntu/>
- [14] Lei Xu, Zonghui Wang, and Wenzhi Chen, "The Study and Evaluation of ARM-Based Mobile Virtualization", College of Computer Science and Technology, Zhejiang University, Hangzhou 310000, China, International Journal of Distributed Sensor Networks
- [15] Libvirt, Retrieved Dec 12, 2016 from <https://en.wikipedia.org/wiki/Libvirt>
- [16] Libvirt documentation, Retrieved Dec 12, 2016 from <https://libvirt.org/docs.html>
- [17] Virt-Manager: Source and Documentation, Retrieved Dec 12, 2016 from <https://virt-manager.org/>
- [18] Dhrystone Benchmark, Retrieved Oct 18, 2016 from <https://en.wikipedia.org/wiki/Dhrystone>
- [19] Whetstone Benchmark, Retrieved Oct 18, 2016 from https://en.wikipedia.org/wiki/Whetstone_benchmark

- [20] Sami Yanguiy, Pradeep Ravindrany, Ons Bibanizx, Roch H. Glithoy, Nejib Ben Hadj-Alouanex, Monique J. Morrow, Paul A. Polakos, "A Platform as-a-Service for Hybrid Cloud/Fog Environments"
- [21] Loai A. Tawalbeh, Waseem Bakhader "A Mobile Cloud System for Different Useful Applications"
- [22] Thinh Le Vinh¹, Reddy Pallavali², Fatiha Houacine¹ & Samia Bouzefrane¹
¹CEDRIC Lab, CNAM, 292 rue Saint Martin, Paris, France ²Dept. of computer science, Andhra Pradesh, India "Energy efficiency in Mobile Cloud Computing Architectures"
- [23] OpenStack, Retrieved Sep 16 2016 from <http://www.openstack.org>
- [24] Xen hypervisor, Retrieved Sep 16, 2016 from <http://xen.org/>
- [25] Openstack Nova Developer documentation, Retrieved Dec 12, 2016 from <http://docs.openstack.org/developer/nova/>
- [26] Openstack Nova Architecture documentation, Retrieved Dec 12, 2016 from <http://docs.openstack.org/developer/nova/architecture.html>
- [27] Openstack api, Retrieved Sep 16, 2016 from <http://developer.openstack.org/api-guide/quick-start/>
- [28] Libvirt migration, Retrieved Dec 12, 2016 from <https://libvirt.org/migration.html>
- [29] OpenStack Dashboard: Horizon, Retrieved Sep 16, 2016 from <http://docs.openstack.org/mitaka/install-guide-ubuntu/horizon.html>