CLOUD FOR DISTANCE EDUCATION

Combined Report

submitted in partial fulfilment of the requirements for the degree of

Mater of Technology

by
Shubhanga N.
and
Harshad Bokil

with the guidance of
Prof. Vikram M. Gadre

Department of Electrical Engineering
Indian Institute of Technology, Bombay
Powai, Mumbai - 400 076.

2014-2015

ACKNOWLEDGEMENTS

We take this opportunity to express our gratitude and regards to Prof. V. M. Gadre for providing us this wonderful opportunity to work under his supervision and for his guidance throughout the project. His guidance and continuous encouragement has made this work much more exciting.

We would also like to thank all TI-DSP lab mates for their help and support throughout our stay at IIT Bombay.

Finally we thank TI-DSP lab, Department of Electrical Engineering, CDEEP IIT Bombay for providing us all the resources needed to carry out this work.

Preface

This report is a result of two individual Masters dissertations by Shubhanga and Harshad, focused on creating a cloud environment for distance education. The first dissertation is focused on building a miniaturised low cost cloud cluster using Raspberry Pi computers. This cluster intends to provide "Infrastructure as a Service" (*IaaS*), by servicing the users' request to access the cloud for viewing stored video lectures. The other dissertation focuses on using multimedia cluster for encoding and transcoding videos. This multimedia cluster is built using commercial Personal Computers for real-time processing of videos. Unlike the cluster using Raspberry Pi computers, this cluster is specifically built for multimedia processing with fixed computing resources.

To effectively communicate the work that has been done, this report has been organised as following.

- **Chapter 1** gives the motivation for the work that has been done.
- **Chapter 2** discusses state of the art experimental cloud clusters that have been set up by academic institutions around the world.
- **Chapter 3** discusses the basic concepts required to understand the work that has been done.
- **Chapter 4** gives a detailed implementation of the Raspberry Pi cluster.
- **Chapter 5** discusses the methodology used for setting up a test-bed for distributed encoding and transcoding in detail.
- **Chapter 6** gives detailed results and their analysis for the experiments carried out on the TI-DSP lab cluster.
- **Chapter 7** concludes this report with the pointers leading to the future scope of this expedition.

Glossary

Video encoding It is the process of compressing raw video files in order to meet formats specified by different standards.

Video bit-stream It is a time series of bits coming out of video encoder representing encoded video data.

Video transcoding Video transcoding is a process of converting compressed video from one format to another.

Distributed system It is collection of multiple *independent* computing resources which are connected to each other and can collectively perform useful operations.

Cloud computing It is type of computing where computing resources are available on demand. They are abstract to the user. User need not to have local servers or any other personal devises to run desired application.

Cluster It is a set of network connected computers which can collaboratively complete a big computing task using "divide and conquer" strategy.

Shared memory multi-core system A single chip which contains multiple processors which have common level 2 or level 3 cache is known as shared memory multi-core system.

Heterogeneous multi-core system A multi-core system in which all the computing nodes do not have equal compute capability.

Contents

List of Figures

Chapter 1

Introduction

1.1 Distance Education

Distance Education is a mode of delivering education to students who are not physically present in a classroom. It provides for a means of learning to students who are separated from instructors either by time or distance. With the increased development of Internet and the availability of Personal Computers, distance education has become a reality and has gained a lot of attention in the recent past. A lot of universities and various websites on the internet (for example, Coursera) provide means to learn without having to be physically present in a traditional setting such as a classroom. Indian Institute of Technology Bombay (IIT Bombay) itself has a department, Centre for Distance Engineering Education Programme (CDEEP), which has innumerous video lectures by professors of various departments of the institute, which caters to learners all over the world. With the increasing number of video courses every year that need to be managed by such departments, a setup needs to be in place to store all these videos, retrieve it when required and display it in the required format to users over the internet. A suitable solution also needs to be developed to cater to the users requests, for example, in handling the bandwidth requirements for streaming video data for all the users over the internet.

1.2 Cloud Computing

Cloud Computing can be defined as the sharing of resources such as servers (to serve users), databases (to store user data), and storage devices (to store volumes of data), etc., so as to form an abstract entity. A Cloud is usually outsourced to organizations or institutions for use, such as for storage or computation purposes. The basic functionality of a cloud is to maximize the effectiveness of these shared resources. A cloud is considered to consist of sufficient resources so as to easily cater to the maximum number of users i.e. so as to not attain saturation of the resources and not deny any service. The structure of a Cloud can be as shown in Figure 1.1.

Cloud Computing is emerging as a new processing paradigm based on outsourcing infrastructure on a pay-as-you-go basis, accommodating services ranging from private data processing to public website hosting. These services are supported by establishing

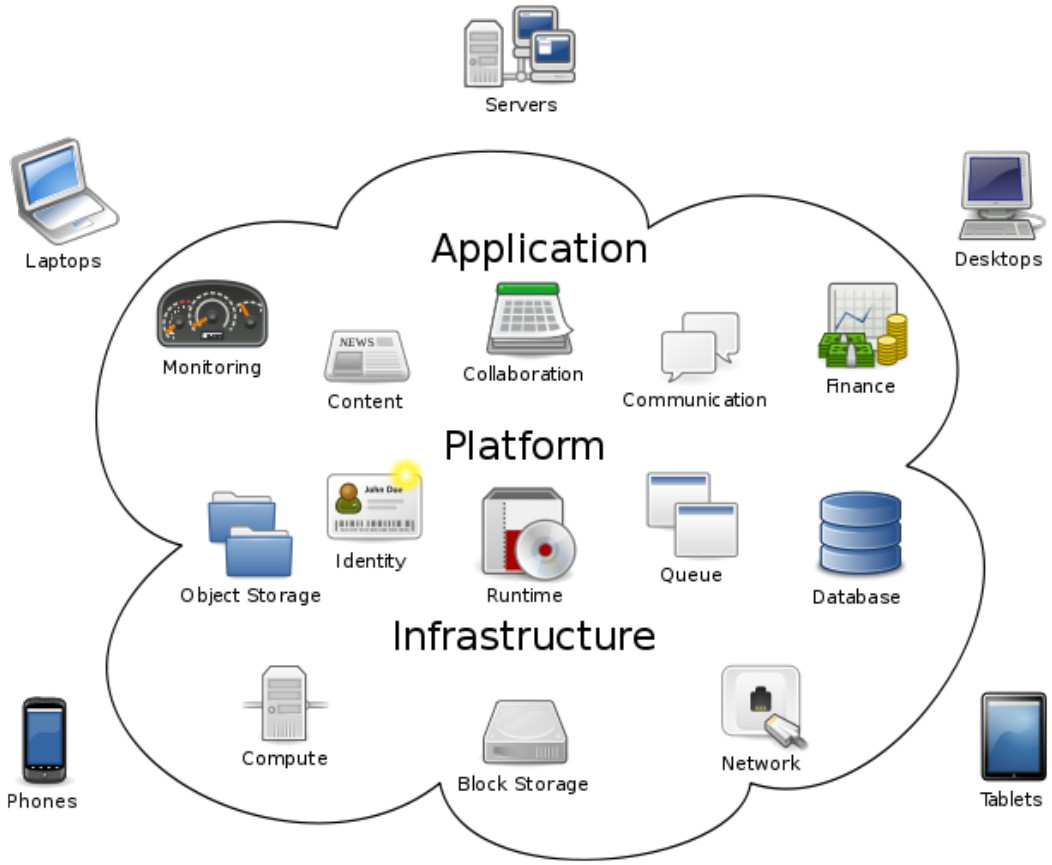

Figure 1.1: Structure of a Cloud [3]

Data Centers (DC), which consist of tens of thousands of networked machines under a single roof.

1.3 Cloud Services for Distance Education

Cloud computing seems to be the best solution for distance education in terms of servicing all the learners over the world based on requirement, to handle huge volumes of video content, manage user data such as their username, password, personal information, their subject preferences etc. and also to transcode a video to a required format for displaying on the user's device. Instead of having a huge DC with all machines running with maximum capacity, it is better to use these resources on a need basis to minimize power requirements and to increase the efficiency of the DC. Thus, the cloud solution is better than the traditional client - server model used by most websites to cater to user's needs, as this can be used on a scalable basis i.e. using the resources of the DC, based on the number of users and their requirements.

1.4 Problems associated with setting up a Cloud

The main obstacle for educational institutions to set up a Cloud for distance education purposes is the capital required to set up Data Centers with resources to handle the large number of videos and users. Building a cloud infrastructure can cost millions of dollars [1]. Currently such a Cloud can be established only by huge organizations such as Amazon, Cisco, etc., and they outsource it to other organizations/institutes. Even though a Cloud service can be hired by educational institutions, a model needs to be developed so as to maximize the usage of the Cloud and minimize the cost. Research activities pertaining to cloud is also in an infant stage due to the inaccessibility of Data Center infrastructures for cost-effective large-scale experimentation [1]. Thus a miniaturised, cost - effective version of a Cloud is necessary for educational institutions in order to study its performance and capabilities, so as to effectively implement a cloud for distance education.

1.5 Cloud for multimedia processing

Multimedia is gaining lot of importance in the area of distance education in the recent time. Majority of multimedia bandwidth is consumed by high definition videos. Apart from educational video content over the internet, according to a survey by CISCO [28], every second, nearly million minutes of video content will cross over Internet by the year 2018 and it is also estimated that, data traffic by mobile devices will increase 11-fold from 2013 to 2018 and majority of which will be high definition video streaming

Coming to the Indian context, our country has over 151.5 million active Internet users as per survey conducted in 2012 (source: Wikipedia), which is third largest in the world. However on the darker side, India has an average connection speed of 1.4 Mbps [29] which is the lowest in Asia-Pacific region. Hence it is very important to choose video encoding techniques which can deliver very high compression efficiency. New generation video codecs such as H.264 and High Efficiency Video Encoder (HEVC) shows much better performance as compared to their predecessors in terms of compression efficiency. However improvement in compression efficiency comes at the cost of more and more compute power.

According to Correa et al. [30], High Efficiency Video Codec, which is state of the art, is 200% to 500% more CPU intensive as compared to its predecessor H.264 video codec. Where as H.264 has computational complexity increase of the order of $10x$ over its predecessors [32]. On the contrary, silicon industry is deviating from Moor's law. Now we have reached the state where further increment in processor operating frequency to achieve superior performance has become impossible. Hence many computer vendors are coming up multi-core processors. Since multi-core processors also have an upper limit on their performance, it is important to use multiple multi-core processors together to complete more complex task of efficiently encoding videos.

There is need for improving encoding time for batch processing of videos and real time video encoding. This problem can be tackled by implementing thread level parallelism in next generation video encoders to efficiently use computing resources. There are various approaches (please refer chapter 3.4), for example group of picture (*GOP*) level, frame level, slice level and macro-block level parallelization, out of which frame, slice and macro-block level implementations have problem of data dependency. Here, We have implemented GOP level parallelism. GOPs are assigned to different processor threads and each thread processes sequence of frames. In GOP data access pattern, there is dependency between frames within GOPs. However no data dependency between two GOPs, thus each thread can process a GOP independently, without referencing frame outside GOP.

Chapter 2

Literature Survey

This chapter intends to introduce the state of the art Raspberry Pi cloud clusters around the world. Before going ahead with it, we need to review the capabilities of a Raspberry Pi. Section 2.1 gives an introduction to the Raspberry Pi. Section 2.2 and 2.3 introduces to two Raspberry Pi cloud clusters that have achieved fame in the academic world.

2.1 Raspberry Pi

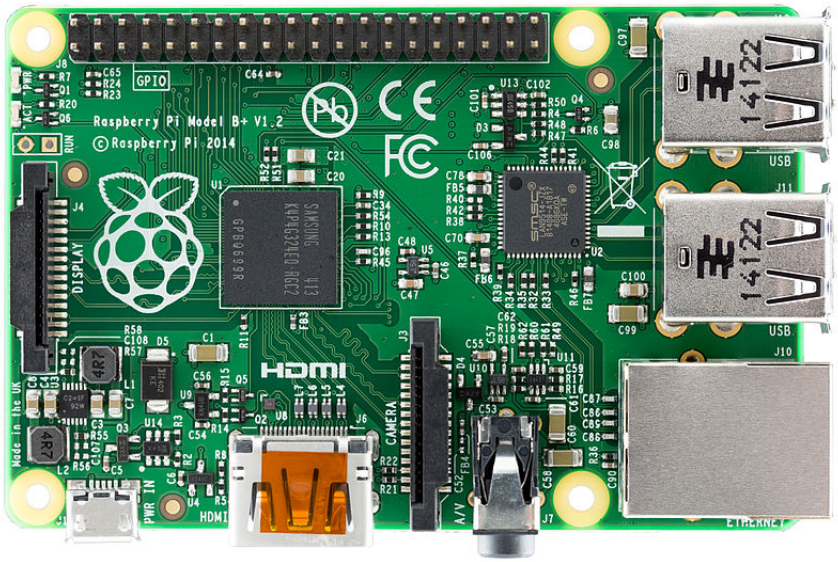

Figure 2.1: Raspberry Pi [2]

The Raspberry Pi is a credit card-sized single-board computer developed in the UK by the Raspberry Pi Foundation with the intention of promoting the teaching of basic computer science in schools. The Raspberry Pi has a Broadcom BCM2835 system on a chip (SoC), which includes an ARM1176JZF-S 700 MHz processor, VideoCore IV GPU, and originally had 256 megabytes of RAM, which was later upgraded to 512 MB. The 256 MB RAM Raspberry Pi is named Model - A, while the 512 MB RAM version is named

Model - B and Model - B+. It does not include a built-in hard disk or solid-state drive, but it uses an SD card for booting and persistent storage. It has an RJ45 ethernet port (in Model - B and Model - B+) to connect to a network, a camera interface, a HDMI port to connect to a monitor or TV, USB ports (number varying on the version of Raspberry Pi) for connecting a keyboard and a mouse and a 3.5 mm jack for audio output. A picture of Raspberry Pi is as shown in Figure 2.1.

The basic version of Raspberry Pi costs as little as $25 which makes it economically suitable for implementing a miniaturised Cloud service. The Raspberry Pi is primarily designed for multimedia-capable embedded devices and shares many of the properties of Cloud-based servers, such as limited storage and peripheral capability, however at a smaller scale [1]. Hence, it is ideal for distance education purposes which deals with videos. The Cloud setup can also be used to transcode a video to the required format before transmission, depending on the requirement of the user. With all these advantages, Raspberry Pi seems ideal to create a miniaturised Cloud, to ascertain its performance and other characteristics, when used for distance education.

This chapter outlines the existing state - of - the - art on setting up a small - scale cloud using Raspberry Pis.

Since the release of Raspberry Pi in 2012, research enthusiasts around the world have shown keen interest in creating a system with many Raspberry Pis to form a cluster that emulates a Cloud [2]. Since a single Raspberry Pi cannot perform computationally intensive operations, a lot of literature can be found on the internet pertaining to forming clusters of Raspberry Pis in order to implement a Cloud, which can perform computationally challenging operations (though not comparable to the x86 servers that are in use in DC). Such a cluster can be used to observe the performance characteristics in a small - scale. Two papers of significant interest in this field that motivated this report were:

- The Glasgow Raspberry Pi Cloud: A Scale Model for Cloud Computing Infrastructures
- Affordable and Energy - Efficient Cloud Computing Clusters: The Bolzano Raspberry Pi Cloud Cluster Experiment

2.2 The Glasgow Raspberry Pi Cloud: A Scale Model for Cloud Computing Infrastructures

This paper was accepted in the 33rd IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW) in July 2013. The Glasgow Raspberry Pi Cloud implements a prototype scale model of a Cloud Data Centre cluster using 56 Model - B version Raspberry Pis. These Raspberry Pis are housed in 4 racks constructed using Lego bricks, with each rack containing 14 Raspberry Pis. The Raspberry Pis are connected through a canonical multi - root tree topology i.e. the 14 Raspberry Pis of one rack are connected to a Top of Rack (ToR) switch, which are in turn connected to the rest

of the devices using an OpenFlow - enabled aggregation switch. OpenFlow switch is a Software Define Networking (SDN) switch, which enables the entire topology to be fully programmable. All the Raspberry Pis are in turn connected to the internet through the school's university gateway.

The system architecture mentioned above is as shown in Figure 2.2.

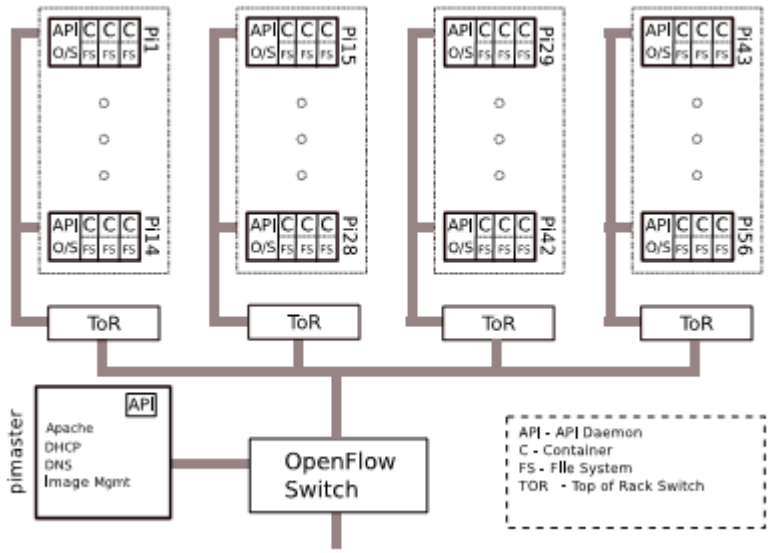

Figure 2.2: System architecture of the Glasgow Raspberry Pi Cloud [1]

The 56 Raspberry Pis are run on Raspbian Linux, which is the optimised operating system for Raspberry Pi [2] [5] and is easily available on the Raspberry Pi web page. On top of the Raspbian OS, Linux Containers (LXC) are used to provide a type of virtualisation. There is an API daemon on each Raspberry Pi providing a RESTful management interface for facilitating virtual host management and interacting with a head node (the pimaster). A system administrator can implement customised IP and naming policies through DHCP and DNS services running on the pimaster. On top of the management layer are the virtual hosts. Each Raspberry Pi can comfortably host 3 containers concurrently and each container can execute any user application. An outward - facing webserver on pimaster provides a web - based control panel to users and administrators.

The software stack for an individual Raspberry Pi is as shown in Figure **??**.

2.3 Affordable and Energy - Efficient Cloud Computing Clusters: The Bolzano Raspberry Pi Cloud Cluster Experiment

This paper was accepted in the 5th IEEE International Conference on Cloud Computing Technology and Science in December 2013. This paper also focuses on setting up

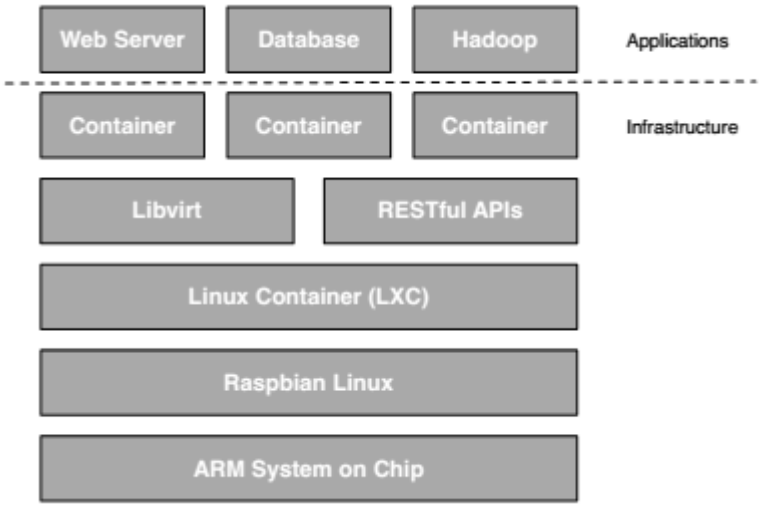

Figure 2.3: Software stack of an individual Raspberry Pi in the Glasgow Raspberry Pi Cloud [1]

a cluster of Raspberry Pis to create a system that can be further developed to create any prototype system such as the Pi Cloud mentioned in the previous section. The main focus of this paper is to evaluate the performance of a cluster of Raspberry Pis as compared to a single Raspberry Pi. This is the biggest cluster of Raspberry Pis as of this writing with 300 Raspberry Pis connected together [2].

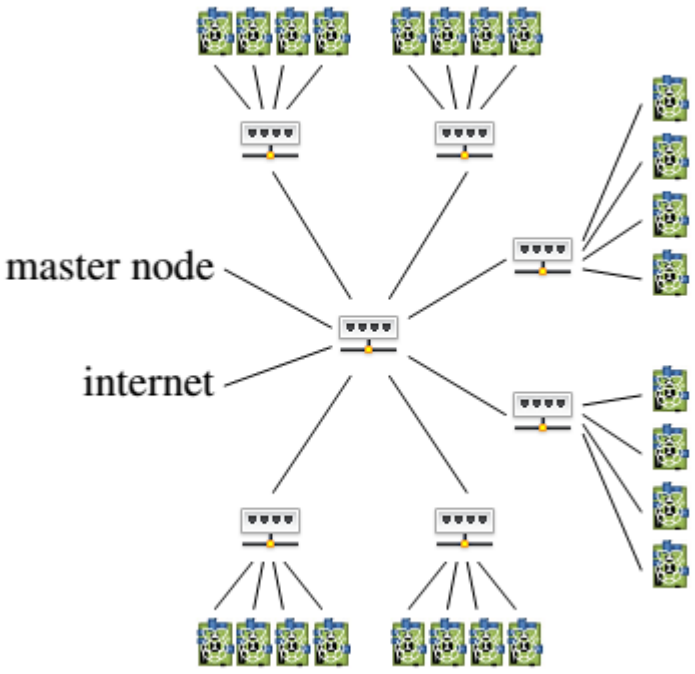

Figure 2.4: Network Topology of Bolanzo Raspberry Pi Cloud cluster [2]

In this paper, 300 Raspberry Pis were connected in a Star topology as shown in Figure 2.4, with one switch acting as the core of the star network, and the other switches connecting this core switch to the 300 Raspberry Pis. The core switch is also connected to a master node, to configure the entire topology and a router which is in - turn connected to the internet. This router is beneath the university firewall. The Raspberry Pis are housed using a holder and a container. A holder is a transparent plastic surface used to connect a single Raspberry Pi. There are 24 such holders in a single container, which forms a rack of Raspberry Pis as shown in the Figure 2.5.

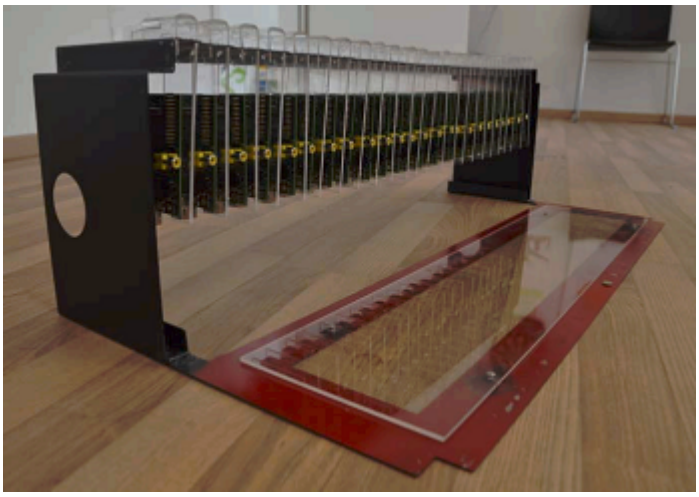

Figure 2.5: A container containing 24 holders, each holder with a Raspberry Pi [2]

The huge number of Raspberry Pis pose a problem of supplying power to each device which was solved using Power Supply Units (PSU) of unused computers. Each PSU can deliver a current of 25 - 30 A, while a single Raspberry Pi needs a current of 0.5 - 1 A for its normal working. This power is distributed to the Raspberry Pis using USB cables.

The Bolanzo Raspberry Pi uses a custom made OS based on Debian 7, featuring a minimal configuration sufficient for integrating an RPi into the cluster. It does not have any virtual hosts, as was the case with Glasgow Raspberry Pi, because of the large number of Raspberry Pis used. Each Raspberry Pi acts as a single server in a DC. Thus, the Bolanzo Cloud cluster claims improvement in the Raspberry Pi performance wen compared to a single Raspberry Pi. They also suggest future work to extend this to a Cloud setup by adding the software stack for the Raspberry Pis and monitoring the data flow in / out of them.

Chapter 3

Fundamental Concepts

Before going to the implementation of the system and the work done in this report, some concepts have to be explained to understand the working of servers in a Data Center. The servers of a Data Center thrive on a concept called as Virtual Machine Hypervisors. A similar paradigm called as LXC (Linux Containers) is of significant interest with reference to this report, owing to the fact that the networked servers are emulated out of computationally conservative Raspberry Pis. Hence, the need of the hour is to understand LXC, the requirement of Containers, what they are and how they work. We also need to know as to how Containers are better than Hypervisors and also how it is best suited for the Raspberry Pi.

In a cloud environment, it is very important to utilise every bit of power for the useful computation. Section 3.3 discusses different techniques of efficiently utilising available resources in computing cluster while exploiting GOP level parallelism in video encoding and transcoding. Section 3.4 gives overview of different encoder parallelization techniques explored by researchers across the world. Finally section 3.5 presents different ways to predict performance of a multimedia cluster.

3.1 Hypervisors and Containers

The backbone of most servers in a Data Center in the real world are Virtual Machine (VM) Hypervisors. A Hypervisor is a software program that runs on a host machine (running a host OS) and is used to manage multiple operating systems (VMs) to share the same processor, memory and other resources. These multiple operating systems (Guest OS) can either be same as that running on the host machine or different. The hypervisor decides which OS gets how much memory, processor time and other resources of the host machine i.e. each Guest OS appears to have direct access to the processor and memory, but the Hypervisor is actually controlling the host processor and the resources, allocating what is required by each guest OS. The Hypervisor also has to make sure that there is isolation between the VMs, so that a process on one VM cannot disrupt the functioning of another process in another VM [8].

VMware vSphere, Microsoft Windows Server 2012 Hyper-V, Citrix XenServer and Kernel - Based Virtual Machine (KVM) are some of the famous Hypervisors available as of this writing [9].

Containers are isolated partitions that are incorporated into the kernel of a host machine. In these isolated partitions, multiple instances of applications can be run without the need for a guest OS [10]. This means that the operating system is virtualized in the sense that the containers that run in them have processes that have their own identity and are thus isolated from another process in another container.

LXC in Debian (Raspbian) and Docker for Arch Linux are the examples of containers available as of this writing.

3.2 Containers Vs. Hypervisors

Hypervisors are very heavy on the host OS in terms of the host memory for its operation, even though the VMs are idle. They provide complete isolation between the VMs, since every guest OS assumes it is right above the host hardware, while Hypervisors emulate the host hardware to the VMs [11]. Containers are incorporated into the kernel of the host OS thus making it as an essential tool already available with the host machine. Containers are very lightweight i.e. they do not require a powerful machine to be run in and can be run even on a Raspberry Pi. However since there is no guest OS in a container, a process is truly not independent even though it is isolated.

Containers are gaining a lot of importance these days, with Google investing on it a lot from nearly a decade [11]. The one thing that hypervisors can do that containers cannot is to use different operating systems or kernels [11]. For example, a Hypervisor say VMware vSphere can be used to run instances of Linux and Windows at the same time. With LXC, all containers must use the same operating system and kernel. In short, containers cannot be mixed and matched the way VMs can be.

3.3 Load Balancing

Load Balancing is very important issue in any type of distributed computing application. It is implemented in order to extract higher speed-up from a distributed system. In simple words, proper load balancing makes sure that all the resources in distributed computing system are being utilized to 100% of their capacity. In the cloud computing context, different computing nodes can have different compute capability. Hence load balancing becomes even more important. Conceptually any load balancing technique can be classified into following two basic types [45]

- Static Load Balancing
- Dynamic Load Balancing

3.3.1 Static Load Balancing

In this type of technique, scheduler (the one which distributes the job to all the worker nodes) knows compute capacity of all worker nodes a priory. Then the scheduler or task manager distributes jobs to all worker nodes such that the worker nodes which have lesser compute capability get less amount of job to do and the powerful once get more amount of job. This strategy makes sure that all the nodes are being utilized to the best of their capacity.

However in the context of video encoding, job is highly data dependent. For example, let us suppose a video sequence under test doesn't have much spatio-temporal variation (i.e. absence of motion and rich texture in video sequence) and we divide it into several constant sized *GOP*s (Group Of Pictures). Let us also consider that all the nodes which we are using for distributed video encoding have same compute power. Then two different *GOP*s will take almost the same amount of time to encode on two different nodes. However if video sequence under test has too much of motion or scene changes, two different *GOP*s will take different time on similar worker nodes. Hence Static Load Balancing algorithm will fail to estimate job complexity and will overload low compute capacity worker node in such cases.

For comparison purpose, simple static load balancing algorithm was used for video encoding/transcoding job scheduling on media cluster that was implemented in TI-DSP lab (for further details about implemented media cluster, please refer chapter 5). Figure 3.1 shows flowchart for static load balancing algorithm which was implemented and used for comparative study with dynamic load balancing.

3.3.2 Dynamic Load Balancing

As name suggests, in this type of load balancing algorithms, amount of job to be assigned to a particular node is decided on the fly. Scheduler (or a job manager) need not to know about relative compute capabilities of worker nodes a priory. There are various types to realize dynamic load balancing. Few of them are as following

- **Control Theory and Machine Learning Based**:
  Some metric is chosen in the beginning. Then based upon correctness of current prediction, scheduler generates next prediction for job distribution while trying to minimise errors. Different machine learning and control theory based algorithms are used for correcting prediction in consecutive iterations. Dhinesh Babu et al [46] presented honey bee inspired dynamic load balancing. However video encoding/-transcoding job is too complex and it is very challenging to predict job completion time since it is data dependent.

- **Acknowledgement Based Approach**:
  Figure 3.2 shows flowchart for acknowledgement based dynamic load balancing algorithm. In this approach, a new job is assigned to node as soon as it finishes

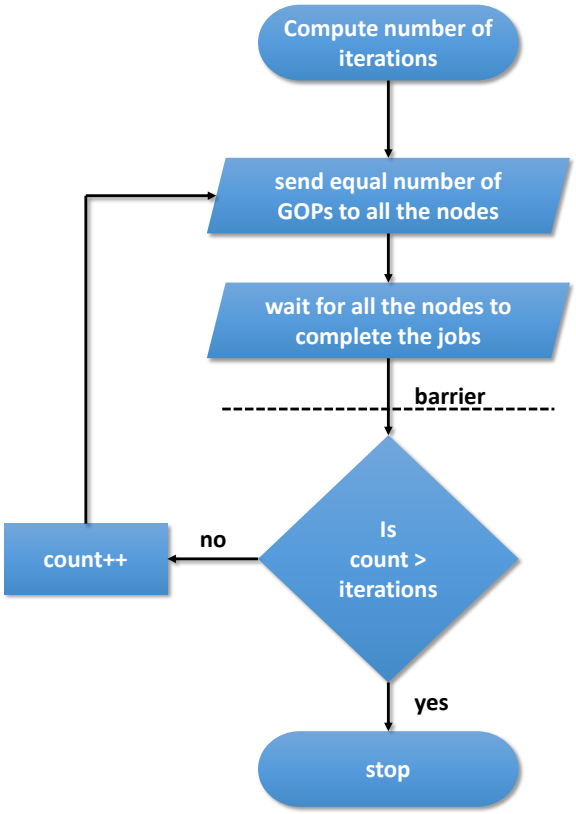

Figure 3.1: Static load balancing algorithm

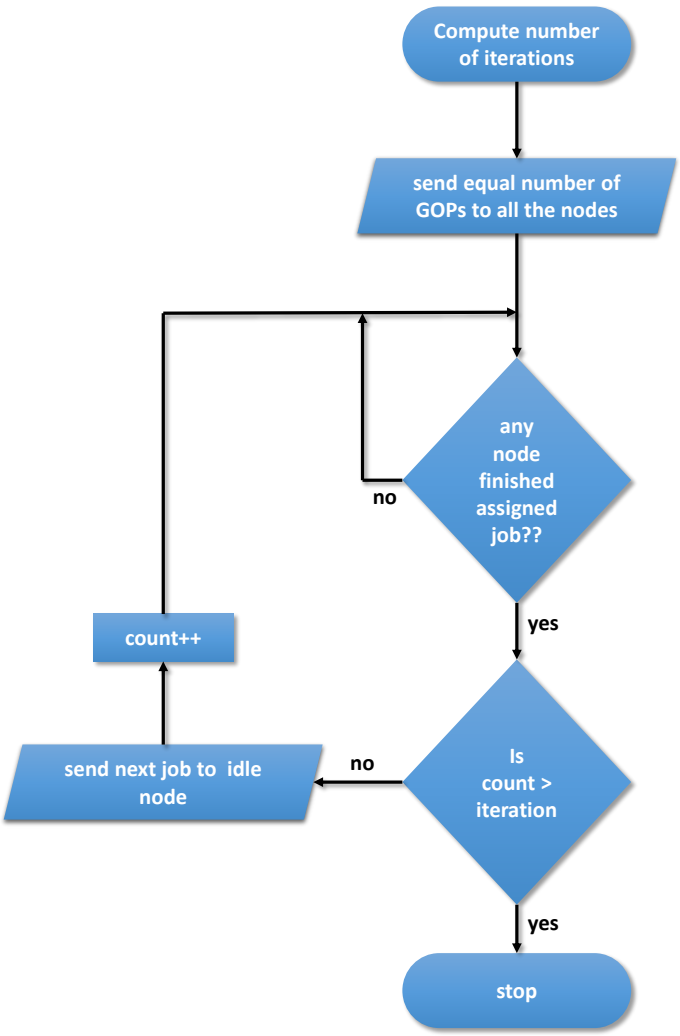

Figure 3.2: Acknowledgement based dynamic load balancing algorithm

previous job. Scheduler need not to spend time in predicting and optimising computing time for a particular node. However there should be a mechanism which makes scheduler aware of job completion status from all the worker nodes. Implementation of this approach is relatively easy as compared to any other dynamic load balancing strategy. Comparative study of static and dynamic load balancing algorithms is presented in chapter 6.

3.4 Encoder Parallelization Techniques

As compression efficiency of video codecs is increasing, the computing power required for video encoding is also touching the skies. For example, H.264 has over 50% more compression efficiency as compared to its predecessors [31]. However, computational

complexity is of the order of $10x$ [32]. Researchers have been trying to exploit inherent parallelism available in video codecs. A detailed study of various video codec parallelization techniques was carried out as a part of this dissertation. [33] Gives an overview of different approaches undertaken by various research groups for the parallelization of H.264/AVC encoder.

Researchers have explored mainly two types of techniques for video encoder parallelization which can be categorised as following.

- Task Level Parallelism *(TLP)*
- Data Level Parallelism *(DLP)*
  - Fine Grain Data Level Parallelism
  - Courser Grain Data Level Parallelism

We will have a look at them one-by-one in following sections.

3.4.1 Task Level Parallelism

In this approach, video encoder is divided into several 'Functional Blocks' *(FB)* and depending upon whether on not these *FB* can be run in parallel, a group of *FB* is mapped to a core either statically or dynamically in multi-core architecture. Many implementations use *TLP* to extract parallelism. For example [35], [36] use TLP for improving encoder performance of H.264 video codec where as [37] use TLP for HEVC encoder.

However, it is clear from most of the studies that we may not be able to achieve proper load balancing in the multi-core processor if we use TLP since time required for each function is different, so the one with larger latency will become a bottle-neck point. For example, after profiling H.264/AVC encoder, it is found that motion estimation and motion compensation (*MEMC*) takes about 60% of the total encoding time [35]. So *MEMC* will certainly be the bottleneck functional block if we use TLP. Hence this approach may be suitable for heterogeneous multi-core architecture and certainly unfit candidate for a platform independent distributed memory systems.

3.4.2 Data Level Parallelism

Instead of spawning different thread for individual tasks (or assigning different task to an individual processor in case of multi-core architecture), data to be decoded can be divided in chunks and then parallely processed. Before discussing further details of *DLP*, we will have brief overview of MPEG encoded video stream's datastructure.

Video bit-stream Structure for MPEG encoded Videos

Figure 3.3 illustrates generic data structure used in almost all the MPEG standard. Everything which is required to playback MPEG encoded bit-stream is embedded inside bit-stream itself. Hence no additional header is required for decoding it [38]

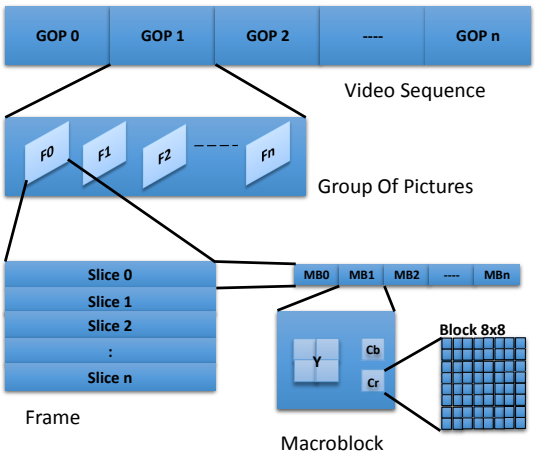

Figure 3.3: MPEG data structure

A picture is divided into 16x16 *Luminance* and 8x8 *Crominance* samples in raw 4 : 2 : 0 YUV video. Each such section of picture frame is known as **Macroblock**. Where as a single **Slice** contains an integer number of Macroblocks. There can be multiple slices in a given picture frame. In some cases whole frame is considered as a single slice. However it increases complexity of motion compensation. There are 3 supported types of frames depending upon degree of freedom of encoding.

- **I-frame:** It is intra-coded frame using transform coding.
- **P-frame:** Both intra and inter picture coding is supported. One motion vector per Macroblock is transmitted for this type of frame.
- **B-frame:** Bidirectional inter-picture coding is supported for this type of frame. 2 prediction signals per Macroblock with weighted average are supported.

And finally, combination of multiple frames is called **Group Of Pictures** (*GOP*). Every *GOP* starts with an *I-frame* followed by *P* and *B* frames. Size of *GOP* can be varied by encoder. However most of the times it is a fixed value. By rule of thumb, larger the *GOP* size, more is the compression efficiency. However increasing *GOP* size also increases memory requirement for encoding and also affects encoding time badly. We will have more insights on effects of *GOP* size on encoding in chapter 6.

Types of Data Level Parallelism

Based on what depth we want dig in to the encoded video datastructure, there are mainly two types of *DLP* as mentioned in the beginning of this chapter.

Fine Grain *DLP* In this approach, multiple *slices* and/or *macroblocks* are encoded in parallel. H.264 implementations presented in [39] and [40] exploit fine grain *DLP*. However, there are lot many data dependencies present in fine grain *DLP*, resulting in too much data communication and control dependencies in between independent nodes/threads. Hence they are more suitable to be implemented on shared memory platforms. For example, in [39] researches have realised parallel encoder on a Graphics Processing Unit (*GPU*) where many cores share large cache memory.

Course Grain *DLP* As the name suggests, this approach works with one level higher in video datastructure depicted in figure 3.3. As seen in sub-section 3.4.2, encoding of one *GOP* is independent of another *GOP*, we can spawn multiple encoder threads and give one *GOP* to each thread which will encode respective *GOP*s in parallel. However this approach has major shortcomings of having relatively higher latencies and higher memory requirement as compared with Fine Grain *DLP*. Still it is more suitable for distributed memory architectures.

[41] and [42] used Course Grain *DLP* for H.264 encoding on distributed and shared memory based multi-core systems respectively. Course Grain *DLP* is relatively easy to implement. We cold extract out more than $15x$ speed-up over serial encoders. Detailed results will be discussed in chapter 6. As already stated in introductory part of this dissertation, our main aim is to use Ethernet connected cluster of computers for encoding High Definition videos using H.264 and HEVC codecs, *GOP* level parallelism is the most suitable way to go ahead.

3.5 Performance Prediction

It is very important to estimate how much performance gain we are going to get if we add certain number of computing units in our multimedia cluster. Sometimes it may happen that adding more number of compute units does not improve performance as expected rather performance might also degrade or reach to saturation if we blindly keep on adding computing resources. Hence it is very important to have a rough estimation of optimized number of resources that one needs to deploy.

In 1967 Amdahl proposed a model which gives a theoretical estimate of the upper bound on the performance attainable with multi-core architectures [47]. It says that the maximum performance attainable is bounded by the serial portion of a program. According to Amdahl, *speedup* for N node computing system relative to a serial execution is defined as shown in equation 3.1

$$speedup_N^{Amdahl} = \frac{1}{(1-f)+\frac{f}{N}} \tag{3.1}$$

Where,

$$\begin{aligned} f &\implies \text{fraction of parallelism for a given application} \\ N &\implies \text{number of compute nodes available} \end{aligned}$$

Amdahl's model holds true if total job size (i.e. total number of frames to be encoded) remains same as we keep on increasing number of compute units. If in a simulation, we increase job size (number of frames to be encoded) as we increase number of compute nodes then Gustafson's [48] model needs to be used which is given by equation 3.2

$$speedup_N^{Gustafson} = (1-f) + f * W_N \tag{3.2}$$

Where,

$$\begin{aligned} f &\implies \text{fraction of parallelism for a given application} \\ N &\implies \text{number of compute nodes available} \\ W_N &= N * W \\ W &\implies \text{the quantum of total job (in our case size of one } GOP\text{)} \end{aligned}$$

Results of predicted performance using above mentioned models and actual extracted *speedup* values are plotted in chapter 6.

Chapter 4

Implementation of Raspberry Pi Cluster

The main objective of this section is to create a working model of a distance education scenario using Raspberry Pis. As a start, only passive learning i.e. distance education through transmission of stored videos in a server to a client is being implemented. The live transmission of a classroom can be thought of as an extension to this project once the passive model is implemented.

4.1 The Big Picture

The Glasgow Raspberry Pi Cloud model was found to perfectly suit the requirements for storing and transmitting video to users in the distance education scenario. Hence, the main aim of this project is to replicate the Glasgow Raspberry Pi Cloud. The basic idea in replicating the system is to achieve the software stack for the individual Raspberry Pi, as explained in section 2.2, and to extend it to all the Raspberry Pis. Once this has been achieved, this server can be used to host many Virtual Machines in the form of Containers i.e. a Webserver and a database can be setup in containers of the Raspberry Pi. This will serve all the clients that log on to the Raspberry Pi Cloud.

Apart from implementing the Glasgow Raspberry Pi Cloud, a Network Attached Storage (NAS) has to be implemented in the Raspberry Pi in order to store videos that need to be transmitted. The list of all the stored videos can then be shown as links in a website that will be hosted in the Webserver in one of the Containers. When a user clicks on a link to a video, it has to be retrieved from the NAS and then transmitted to the user.

As a futuristic thought, the cluster formed using the Raspberry Pis can also be used for transcoding a video from one format to another. This can be thought of as "Cloud as a Computing platform", for parallel execution of a given program among the various Raspberry Pis used in the cluster. Once this is achieved, the transcoding process can be "outsourced" to the Cloud cluster. Since transcoding takes a lot of computation power to convert a video from one format to another, the transcoding achieved using a parallel execution amongst the Raspberry Pis cannot be done in real - time.

Also, a parallel model to the Glasgow Raspberry Pi Cloud software stack was found through the installation of Docker, which will be explained in detail in the sub - sections to follow.

4.2 Operating System for the Raspberry Pi

There is much debate among the Raspberry Pi community [5] [6] [7] as to which is the best Operating System (OS) for the Raspberry Pi, among:

- Raspbian (Debian Linux)
- Risc OS (developed by ARM)
- Pidora (based on Fedora)
- Android (the OS that is present in most smart phones)
- Arch Linux (Bare bone Linux)

After much research by going through the Raspberry Pi community blogs [5] [6], the comparison between the different OS mentioned above can be as explained.

- **Raspbian** is the most sought after OS for Raspberry Pis for its perfect balance between simplicity, nice GUI and performance. This is the same OS that is used by the creators of Glasgow Raspberry Pi Cloud which we are trying to replicate. Raspbian has support for LXC which help in creating Containers which makes it the best OS to start with.
- **Risc OS** and **Pidora** do not have much documentation on the internet and hence does not have much support for creating different systems using the Raspberry Pi.
- The **Android** OS, used in most smart phones takes up too much of the limited RAM available, thus making the Raspberry Pi too slow. This also hampers the performance of the system and is hence out of the question.
- **Arch Linux** is the bare bone flavour of Linux and allows the users to customize the Raspberry Pi to their needs. However, the main drawback of Arch Linux is the huge learning curve involved in creating any system using such a bare OS. However, Arch Linux has support for Docker that is an equivalent to LXC in Raspbian. Hence, this OS will also be used in one of the Raspberry Pis.

4.3 The Glasgow Raspberry Pi Cloud

The implementation of the Glasgow Raspberry Pi Cloud lies in implementing the software stack as mentioned in figure 2.3. The software stack can be implemented by installing Linux Containers (LXC) in Raspbian.

4.3.1 Installation of LXC

The installation of LXC on Raspbian turned out to be a humongous task despite innumerous resources available on the internet, thus forcing this section to be written in the report. People all over the world have documented their installation and have provided resources, but it was found to vary from version to version of both the LXC and Raspbian. The difficulty in installing LXC is due to the fact that the kernel of Raspbian had to be prepared before its installation, which involved compiling the mighty Linux kernel on the resource - limited Raspberry Pi. This took hours of time without any guarantee that it will work. Finally, LXC was installed and in this section, every step in the installation of LXC on the Raspbian has been documented. However, the container fails to start after its creation due to an unknown error, as of this writing.

To start with, LXC was tried to be installed on one of the latest versions (and was tried on the latest version released on 05-05-2015) of Raspbian available on The Raspberry Pi Downloads page [12]. However, Containers could not even be created, owing to some support related issues with the latest Raspbian version. Hence, LXC was installed on the Raspbian version `2013-02-09-wheezy-raspbian` since this was the version used in the Glasgow Pi Cloud [13]. This version though stable is very old and ran only on Raspberry Pi Model B (not the model B+).

Building an LXC - Friendly Kernel on the Raspberry Pi

This sub - section deals with compiling a Linux kernel that is compatible to install LXC on the Raspbian. This section is adapted from [13]. It is assumed that the Raspbian version ”2013-02-09-wheezy-raspbian” is already installed in the Raspberry Pi and the preliminary settings are already done before using the Pi.

It is preferred to use the super user mode since normal privileges are not sufficient to execute some of the commands. Notes have jotted below as necessary when something failed and a work around was found.

Update Raspbian.

```
# apt-get update
# apt-get dist-upgrade
```

Install git. (This step is not necessary as the cloning will be done in a different way mentioned in the next step)

```
# sudo apt-get install git-core
```

Update Firmware of Raspbian.

```
# cd /opt
# git clone git://github.com/raspberrypi/firmware.git
```

Cloning from Github to the Pi (using either the git or https protocol) cannot be done if the Pi is behind a Proxy, as is the case with IITB. A workaround would be to use the `curl`

command through `internet.iitb.ac.in` to get internet access and executing the above `git clone` command. However, cloning directly to a Pi takes a lot of time (about 10 - 15 hours) and can rather be done from the website `http://github.com/raspberrypi/firmware.git` on to a desktop. If Windows is used, the clone can be transferred to the Pi using WinSCP. If Linux is used then use the `scp` command to transfer the clone to the Pi. This method can be employed another three times below, whenever a clone from Github needs to be made.

```
# cd firmware/boot
# cp * /boot
# cd ../modules
# cp -r * /lib/modules
# reboot
```

Increase the Swap File Size

```
# pico /etc/dphys-swapfile
```

Change 100 to 500 (MB). Save and exit.

```
# sudo dphys-swapfile setup
# reboot
```

Prepare to Build Kernel

```
# cd /opt
# mkdir raspberrypi
# cd raspberrypi
```

Before executing the next command, please read this very carefully. This clone will be used to compile the Linux kernel. A lot of issues were faced during this process with no solution available on the internet. The **ONLY** solution to this was to clone from the site, `https://github.com/raspberrypi/linux.git`, to a **WINDOWS** computer and to transfer the files after extraction using `WinSCP`. However, Windows is case - insensitive to file names because of which the following files do not get copied. Copy the following files **individually** from the zipped Linux folder to their corresponding locations mentioned below.

```
include/uapi/linux/netfilter_ipv4/ipt_ECN.h
include/uapi/linux/netfilter_ipv4/ipt_TTL.h
include/uapi/linux/netfilter/xt_DSCP.h
include/uapi/linux/netfilter_ipv6/ip6t_HL.h
include/uapi/linux/netfilter/xt_DSCP.h
include/uapi/linux/netfilter/xt_RATEEST.h
include/uapi/linux/netfilter/xt_TCPMSS.h
net/netfilter/xt_DSCP.c
net/netfilter/xt_HL.c
net/netfilter/xt_RATEEST.c
net/netfilter/xt_TCPMSS.c
```

Kindly go through (http://raspberrypi.stackexchange.com/questions/10126/problems-compiling-kernel-on-raspberry-pi) for more queries.

Do not execute the below `git clone command` if you have followed the above important procedure.

```
# git clone git://github.com/raspberrypi/linux.git
# cd linux
# zcat /proc/config.gz > .config
```

Decrease the Swap Space File

```
# pico /etc/dphys-swapfile
```

Change 500 to 100 (MB). Save and exit.

```
# dphys-swapfile setup
# reboot
```

Install Packages for Kernel Compilation. Set the date and time for proper kernel compilation. The format for setting the date is shown in the command below.

```
# sudo date -s "Fri Apr 24 21:31:26 IST 2015"
# sudo apt-get install bc
# apt-get install ncurses-dev
# cd /opt/raspberrypi/linx
# make menuconfig
```

Enable the options in the tree as mentioned below. Press `Y` to select `N` to deselect, `Esc - Esc` to go back to the previous menu or Exit. Save the file before exiting.

```
* General -> Control Group Support ->
           Memory Resource Controller for Control Groups
                (*and its three child options*)

* General -> Control Group Support -> cpuset support

* Device Drivers -> Character Devices ->
                  Support multiple instances of devpts

* Device Drivers -> Network Device Support ->
                  Virtual ethernet pair device
```

Build Kernel. This command takes about 15 hours to compile the kernel.

```
# make
# make modules_install
# cd /opt/raspberrypi
# git clone git://github.com/raspberrypi/tools.git
# cd tools/mkimage
# python ./imagetool-uncompressed.py /opt/raspberrypi/
                                  linux/arch/arm/boot/Image
# cp /boot/kernel.img /boot/kernel-old.img
# cp kernel.img /boot/
# reboot
```

Download Latest LXC

```
# mkdir /opt/lxc
# cd /opt/lxc
# git clone https://github.com/lxc/lxc.git
# apt-get install automake libcap-dev
# cd lxc
# ./autogen.sh && ./configure && make && make install
```

Testing the Installation and verifying if everything works fine.

```
# lxc-checkconfig
```

User Namespace should be missing and CGroup Namespace should be required.

Creating an LXC Container on the Raspberry Pi

This sub - section deals with the installation of LXC (after building the required kernel) on the Raspbian. This section is adapted from [14].

```
# echo "lxc /sys/fs/cgroup cgroup defaults 0 0 >> /etc/fstab
# mount -a
```

Create a Directory to Store Hosts

```
# mkdir -p /var/lxc/guests
```

Create a File System for the Container with the name `test`. This takes about an hour.

```
# apt-get install debootstrap
# mkdir -p /var/lxc/guests/test
# debootstrap wheezy /var/lxc/guests/test/fs/
                http://archive.raspbian.org/raspbian
```

Modify the Containers File System. I do not know how this is signifant in the functioning of LXC.

```
# chroot /var/lxc/guests/test/fs/
# passwd
# pico /etc/hostname
# exit
```

Create the following Configuration file in `/var/lxc/guests/test/config`

```
lxc.utsname = test
lxc.tty = 2
lxc.rootfs = /var/lxc/guests/test/fs
```

Create the Container

```
lxc-create -f /var/lxc/guests/test/config -n test -t none
```

Test the Container. The following commands are not executed successfully with various errors seen at different times. A solution to this problem is yet to be found as of this writing.

```
# lxc-start -n test -d
# lxc-console -n test -t 1
```

4.3.2 Current Status

After much effort, LXC is successfully installed in the Raspbian version ”2013-02-09-wheezy-raspbian”. A Container was successfully created, but is in the stopped state. This can be seen using the `lxc-info` and `lxc-ls` commands. Any attempt to start the containers results in various errors that need to be debugged yet. Sometimes the Wheezy file system that is created for the container gets deleted, the reason behind which is still unknown and yet to be debugged.

Once a Container can be started, a Webserver can be run in it which can host a website with links to the videos, that are stored in a Network Attached Storage (NAS).

4.4 Docker

An alternative to using LXC to create containers in Raspberry Pi is to use a Docker. A Docker is a layer of abstraction same as the LXC but the Docker engine is an advancement to the LXC released in June 2014. The only drawback of Docker is that it is compatible only with Arch Linux, and Arch Linux does not have good documentation on the internet for developing systems. However, according to [15] as of 19th April, 2015, The Docker is made available for Raspbian as well, though unstable. The Docker is claimed to be better than LXC [16] since it is built on top of LXC along with the `libcontainer` library as shown in Figure 4.1

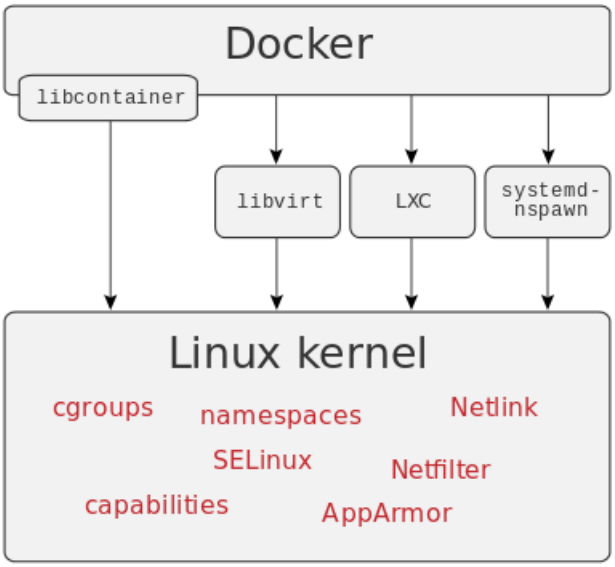

Figure 4.1: Docker [17]

The working of Docker and its advantage over VM Hypervisor can be understood by the Figure 4.2.

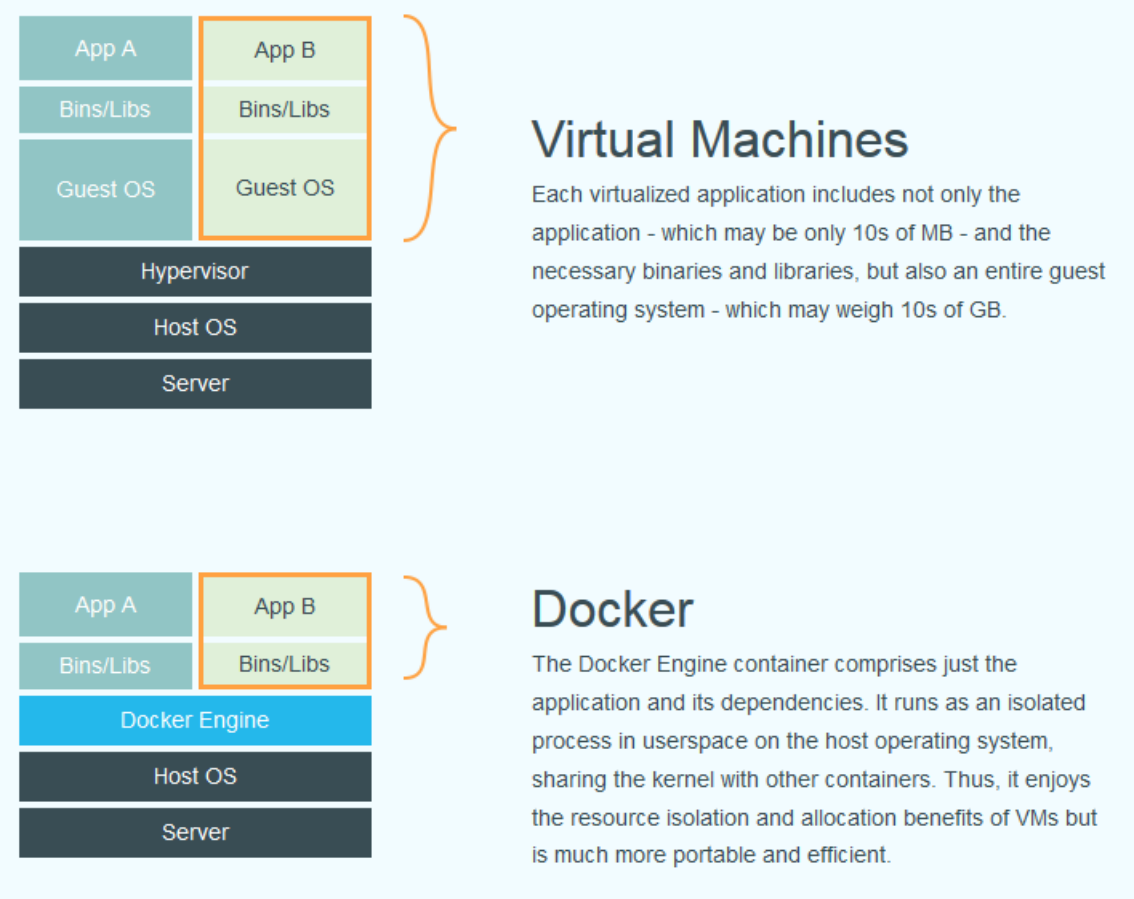

Figure 4.2: Comparison between Docker and Hypervisor [18]

The installation of Docker was found out to be pretty straight forward [19]. Surprisingly, Docker has a lot of documentation on the internet [20] [21] [22] unlike LXC, which facilitated a very fast learning.

4.4.1 Working with Docker

Using Docker requires the keyword `docker` to be prefixed at the beginning of every command. This invokes the Docker libraries. Before using Docker, the Docker daemon has to be launched and this can be done using the command,

```
# docker -d &
```

Every command in Docker is of the form

```
docker [OPTIONS] COMMAND [ARGUMENTS]
```

The list of ”commands” and ”arguments” can be viewed by entering the command `docker` in the prompt.

Once a docker daemon is up and running, a container can be started using the command by using the command

```
docker run [OPTIONS] IMAGE [COMMAND] [ARGUMENTS]
```

OPTIONS define the state of the container upon starting. For example, a docker can be started in the foreground or as a daemon (in the background). Also a port in a container can be exposed to the host so that any incoming requests to the host can be forwarded to the required container.

IMAGE defines the container. Recall that Hypervisors require a guest OS for its virtualization. Docker gets around this burden by creating a bare image of the guest OS and the packages installed above it as layers. This is called as an image and has to be specified while starting the container.

COMMAND specifies the command to be run in the container. For example, `docker run -d shubhanga/raspbian /bin/echo "Hello World!"` starts a container, executes the command `/bin/echo "Hello World!"` in the raspbian running inside the container. This would print "Hello World!" in the terminal. Note that `-d` requests the container to be run as a daemon. Also note that, the image is specified as `shubhanga/raspbian"`. It has two parameters of the form "username/image". This username refers to login of the user who created the image. If the mentioned image is not available locally, Docker automatically searches for the image online on the public docker repository.

It has to be noted here that, a container stops running once the process running in it stops. This has to be taken care of during scenarios that demand containers to be available all the time. More Docker commands and their syntax can be found in [23]

4.4.2 Setting up the Raspberry Pi Cloud

A Docker base image, for example Raspbian Wheezy is enhanced by installing a Webserver into it and saving it. Now to host a Raspberry Pi cloud on to the internet, one simply has to start that container and request it to keep running as a daemon. Since the Webserver is running inside of a container, a port is opened that sends all the incoming requests coming from the clients to the container. The Raspberry Pis are connected to a switch which is connected to the university's gateway. Thus the Raspberry Pi is available from anywhere in the university.

A database server also needs to be setup in another container. This can be used to handle the users, by giving them login credentials and warding off other people.

Another container needs to be started with NAS installed in it. This NAS can be used as a file handling server to store videos that needs to be retrieved and displayed to the users, on request. As of now, NAS has been set up on a separate Raspberry Pi as explained in the next section.

4.5 Network Attached Storage using a Raspberry Pi

The Distance Education system that is being designed requires a lot of storage space, to store videos that can be retrieved on demand and transmitted to a user. A Network Attached Storage (NAS) is best suited for this purpose, since the storage is connected to the network of Raspberry Pis that are used to serve the clients requesting for the video lectures.

A NAS can be defined as a file - system that is connected to a network (of computers) that is used to service a diverse set of clients (as shown in the Figure 4.3). Setting up a NAS requires a Hard Disk to be interfaced to a Raspberry Pi. The main advantage of a NAS is that the system managing the hard disk need not be computationally powerful. The Raspberry Pi also best suits to be used for managing a NAS. Various resources are available on the internet as to how to set up a NAS using a Raspberry Pi.

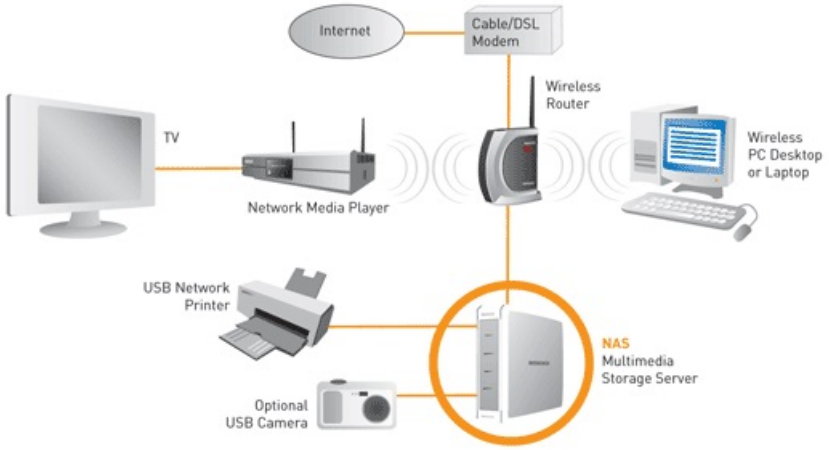

Figure 4.3: Picture depicting the usage of NAS [24]

The procedure to create a NAS using Raspberry Pis is taken from [25].

A 2 TB hard disk was interfaced to another Raspberry Pi on the same network. A folder called as `shared` is created inside which the video lectures can be stored. As of now if anybody tries to access this Raspberry Pi from a browser, the video lectures are showed and can be played. A method now has to be found out to retrieve these videos on the click of a link, from the website that is hosted in one of the containers of the other Raspberry Pi.

Chapter 5

Implementation of Multimedia Cluster

In the previous chapters, we had an overview of cloud computing, different algorithms for load balancing etc. To study some of those algorithms for video encoding and transcoding, we need a testbed with machines having sufficient computing power to encode videos in *real-time*. Hence we set up a small cluster of 11 computers in TI-DSP lab. Section 5.1 mainly focuses on methodology used for setting up TI-DSP lab cluster. Section 5.2 gives overview of *Message Passing Interface* library which was used for managing inter-node communication. Section 5.3 discusses about 'ffmpeg' software which was used for encoding and transcoding videos.

5.1 TI-DSP lab Multimedia Cluster

Figure 5.1 shows block diagram of multimedia cluster which has been set up in TI-DSP lab. All the 11 computers are connected through GBPS Ethernet switch. They are running 32 bit Ubuntu 12.04 operating system with variants of Intel Core 2 Duo processor. Although all the nodes have similar microprocessors, their compute capabilities are different since there is a variation in motherboard model. In short, it is a heterogeneous cluster platform.

Figure 5.2 plots relative compute capabilities of all the nodes in the TI-DSP lab multimedia cluster. To measure relative compute power, 'ffmpeg' H.264 encoder was run on each node with the same video sequence and the time required to complete the task was noted down. Based upon the time taken to finish the job, relative compute power

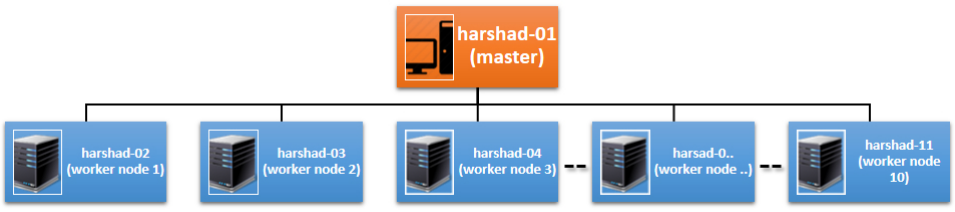

Figure 5.1: Block diagram of computing cluster set in TI-DSP lab

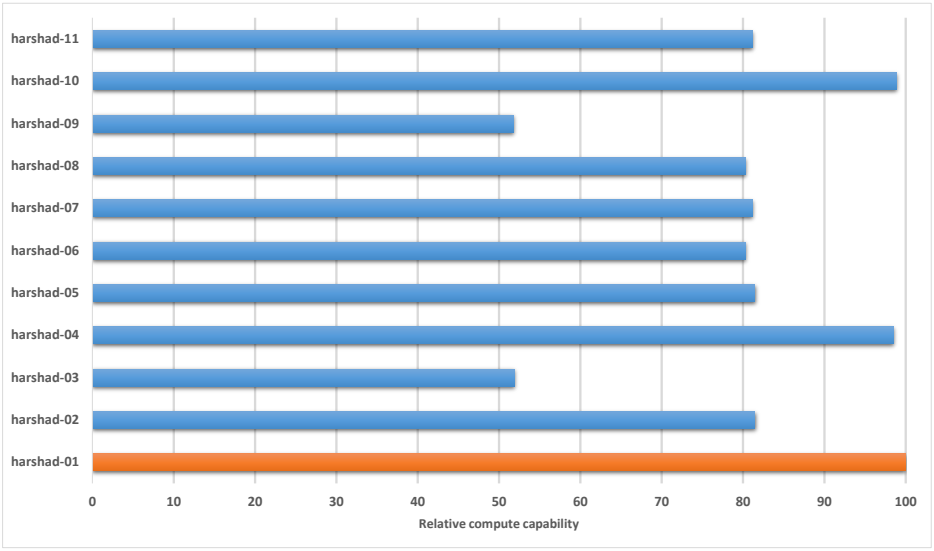

Figure 5.2: relative computing power of all the nodes in DSP lab media cluster

was measured. As we can observe from figure 5.2, *harshad-01* (the MASTER node), *harshad-04* and *harshad-10* have almost same compute capabilities where as *harshad-03* and *harshad-09* have as much as 50% compute capability relative to the master node. If we take average of all the values, it comes out to be equal to 80.61%

The term '*speedup*' is relative. In general, *speedup* is defined as,

$$speedup = \frac{\text{time taken by a single processor}}{\text{time taken by cluster to finish the same job}}$$

However, DSP lab cluster is heterogeneous in nature, we need to normalize time taken by a single processor. Since all the serial jobs are run on the master node and it also happens to have greatest compute power out of all worker nodes, we need to multiply total time taken by serial work by a normalization factor which given by equation 5.1.

$$\begin{aligned} \text{normalization factor } (N_f) &= \frac{1}{\frac{\sum_1^N \text{relative compute power}}{N}} \qquad (5.1) \\ N &\implies \text{total number of nodes in a cluster} \\ \text{after substituting values,} & \\ N_f &= 1.24057 \end{aligned}$$

5.2 Message Passing Interface Library

Distributed computing involves lot of data communication in between computing nodes. There are numerous ways to communicate data in distributed and shared memory architectures, however Message Passing Interface (*MPI*) library is more efficient and portable to use [49]. It is available for all the major programming languages like *C, C++, Fortran, Java* etc. There are various implementations of *MPI* some of them are available in public domain too. We chose **Open-MPI** library. Open-MPI is being used by many super-computers including *K-Computer*, one of the fastest super-computers in the world.

5.2.1 MPI Communication Modes

Inter node communication is the core of *MPI* library. According to *MPI* documentation [49] [50] there are certain 'modes' of communicating data under the *MPI* standard. Some of them are listed below [51]

- Synchronous Mode: This is the safest way to send a data over network. Sending process (or a node) will wait for a ready signal from destination process (or a node). Whenever destination process sends a ready signal, data transfer will be initiated. However involves lot of communication overhead. So this mode has not been used in TI-DSP lab multimedia cluster.

- Buffered Mode: In this mode, data to be sent to the destination is first buffered to a local memory. Programmer holds responsibility of allocating required buffer size before initiating buffered mode communication. However, while using multimedia cluster for video encoding or transcoding, data size to be sent is higher so use of buffered mode communication will require more system memory.

- Ready Mode: This is simplified version of synchronous communication mode. Source, from where data needs to be transmitted, assumes that receiver is ready to receive message and directly sends data. Although ready mode reduces synchronization overhead, this is very risky way to communicate! one may encounter unexpected errors. Hence use of this type of communication mode is discarted.

- Standard Mode: This is the most optimized way to communicate. Protocol for standard mode communication changes according to data size to be transmitted. It is more complex to understand how it works however this is the most popular mode for data transmission. For more detailed information about this mode, interested readers can go through [51].

Standard communication mode have been extensively used while working with TI-DSP lab multimedia cluster due its optimal performance. In the next section we will briefly discuss various *open-MPI* library application program interfaces for *C++* programming language.

5.2.2 MPI Application Program Interface

There are over 370 plus Application Program Interfaces (*APIs*) available with *open-MPI* v1.8.4 [52]. We will go through some of the most used and important *APIs*.

- *MPI::COMM_WORLD.Get_size()*: This function returns number of processes associated with communicator (COMM_WORLD). This *API* is very useful in knowing how many processes have be spawned by an application at the run time

- *MPI::COMM_WORLD.Get_rank()*: *MPI* assigns unique rank to all the processes. This function returns rank of the calling process.

- *MPI::COMM_WORLD.Bcast()*: This *API* is used for broadcasting message to all the nodes. Its syntax is as shown:

```
MPI::COMM_WORLD.Bcast(void* buffer, int count, const
    MPI::Datatype& datatype, int root);
```

  where *buffer* is a pointer to the memory location which is to be broadcasted, *count* is number of elements to be transmitted and *root* is the rank of broadcasting process.

- *MPI::COMM_WORLD.Send()*: This *API* is used for sending data from one process to another using *Standard* mode of communication. This is *blocking send* i.e. program will halt until all the data is sent to the destination. Its syntax is as following:

```
MPI::COMM_WORLD.Send(void* buffer, int count, const
    MPI::Datatype& datatype, int dest, int tag);
```

  where *dest* is the rank of the receiving process and *tag* is a message tag, used for error free reception of the data.

- *MPI::COMM_WORLD.Recv()*: It works exactly opposite of *Send()* API. Every *Send()* call must have corresponding *Recv()* in receiving process. Syntax for this *API* is as following

```
MPI::COMM_WORLD.Recv(void* buffer, int count, const
    MPI::Datatype& datatype, int source, int tag);
```

  where *source* is the rank of the source process.

- *MPI::COMM_WORLD.Isend()*: This *API* is variant of *blocking send()*. Only difference being it is a *non-blocking send* i.e. program will not wait till all the data is sent to the destination. Hence this can be used to parallely send data to multiple worker processes. Its syntax is as following:

```
Request MPI::COMM_WORLD.Isend(void* buffer, int count, const
    MPI::Datatype& datatype, int dest, int tag);
```

  where *Request* is an object which can be used to check whether requested non-blocking send has been completed or not in the later stages of the program. Similarly we also have *Irecv()* API in *MPI* library.

Having looked at the basic *open-MPI* APIs, we will see a basic *C++* program which uses MPI library. Skeleton *C++* code is shown in the following code-snippet:

```
#include <iostream>
#include <stdio.h>
#include "mpi.h"
//mpi.h library is included along withother standard libraries

using namespace std;

int main(int argc, char* argv[]){
   //initializing mpi library
   MPI::Init();
   int numtasks = MPI::COMM_WORLD.Get_size();
   //numtasks stores number of processes spawned by mpiexec

   int rank = MPI::COMM_WORLD.Get_rank();
   //calculating process rank

   //check whether rank is zero??
   if(rank == 0){
      cout<<"I am the MASTER!!"<<endl;
      /*
      *
      *
      *
      code to be executed by the MASTER preocess
      *
      *
      *
      */
   }
   else{
   //if rank is not 0 then process is NOT the MASTER
      cout<<"worker node #"<<rank<<endl; //show rank
      /*
      *
      *
      *
      code to be executed by worker processes
      *
      *
      *
      */
   }
   MPI::Finalize(); // terminates MPI environment
   return 0;
}
```

5.2.3 Setting-up MPI Environment

In section 5.2.2, we have seen APIs and a basic MPI program. In this section we will go through process of setting up MPI environment and how to actually run the MPI code step-by-step.

- *open-MPI installation:*
  Run following command in terminal (for Ubuntu and Debian Linux only)

  ```
  :~$ sudo apt-get install g++ openmpi-bin openmpi-dev
     openmpi-common libopenmpi1.3 libopenmpi-dbg libopenmpi-dev
  ```

- *Code Compilation*
  mpicc and *mpic++* compilers are used for compiling *C* and *C++* codes instead of *gcc* and *g++* compilers.

  ```
  :~$ mpic++ mpi_program.cpp -o mpi_program.exe
  ```

- *Executing MPI code*
  syntax for executing MPI code is as following

  ```
  :~$ mpirun -n xx mpi_program.exe
  ```

  mpirun is used for executing serial/parallel jobs in open MPI. Above code will run xx copies of 'mpi_program.exe' on a single machine.

- *Executing MPI code on Distributed System*
  syntax for executing MPI code is as following

  ```
  :~$ mpirun -machinefile machinefile -n xx mpi_program.exe
  ```

  Above code will run xx copies of 'mpi_program.exe' on a distributed system. Here 'machinefile' contains IP addresses of all the MPI configured nodes. This file has to be provided by programmer. Example 'machinefile' is as shown

  ```
  machinefile

  10.107.74.110 slots=1
  10.107.74.111 slots=1
  10.107.74.112 slots=1
  10.107.74.113 slots=1
  10.107.74.114 slots=1
  10.107.74.115 slots=1
  10.107.74.116 slots=1
  10.107.74.117 slots=1
  10.107.74.118 slots=1
  10.107.74.119 slots=1
  10.107.74.120 slots=1
  ```

  slots=1 indicates number of processes to be spawned on the particular node.

- *Output!*
  After example program, mentioned in section 5.2.2, gets executed, output will look similar to the following:

```
worker node #1
worker node #2
worker node #3
worker node #5
I am the MASTER!!
worker node #4
worker node #6
worker node #8
worker node #10
worker node #9
worker node #7
```

  Point to be noted here is, since all the processes spawned are independent of each other and they are being executed in parallel, their execution order in totally random.

5.3 FFmpeg

FFmpeg [53] is a command line program which is used for video and audio encoding and transcoding purposes. It supports more than 100 codecs. It has several components using which can be used for playing different video/audio types, probing video streams, streaming videos over the internet etc. FFmpeg also supports various data transmission protocols for remote accesses [54]. It is the "Swiss army knife" of multimedia processing. FFmpeg also includes H.264 and HEVC encoders which are optimized for shared memory multiprocessor systems. Hence FFmpeg was used to achieve real-time encoding of HD videos on DSP-lab media cluster.

In this section we will briefly go through usage of FFmpeg for video encoding and transcoding. Detailed instructions on how to install ffmpeg on Ubuntu operating system are provided in [55]. Basic format of using FFmpeg in terminal is as following:

```
:~$ ffmpeg -i <input> <options> <ouput>
```

Where input can be a raw '.yuv' video or video in any other format. We need to specify options for which encoder to be used for output video/audio stream. Following command can be used for encoding raw video with H.264 encoder.

```
:~$ ffmpeg -s <resolution> -r <framerate> --pix_fmt yuv420p -i
    input.yuv -c:v libx264 -x264opts keyint=<GOP_size> -crf <Qp>
    ouput.h264
```

Since we are giving '.yuv' file as an input, we need to specify its resolution, frame rate and what type of YUV is it. *–pix_fmt yuv420p* sets input YUV file's format to $4:2:0$. Then *-c:v libx264* sets output files encoder to the H.264 encoder. *-x264opts keyint=<GOP_size>* further sets H264 encoder options, 'keyint' decides period of *I-frame* which is indirectly a GOP size. Finally *-crf <Qp>* sets encoder to constant quality mode and tries to keep quality of each frame to be equal to Q_P.

```
:~$ ffmpeg -s <resolution> -r <framerate> --pix_fmt yuv420p -i
   input.yuv -c:v libx265 -x265-params keyint=<GOP_size> -b:v
   <bitrate> ouput.hevc
```

Here FFmpeg will encode raw video into HEVC format. Option *-b:v <bitrate>* will set HEVC encoder to constant bitrate mode and will try to keep average bitrate to specified value in *kbps*.

```
:~$ ffmpeg -i <input> output.yuv
```

Decoding with FFmpeg is as simple as this! We just need to specify that output file is a raw video, it will detect input files encoder on its own.

Chapter 6

Experiments and Results

On TI-DSP lab multimedia cluster, several experiments were carried out. These experiments can be categorised into two types.

- Raw Video Encoding into H.264 format
- Video Transcoding from MPEG2 to H.264 and HEVC format

We have seen in section 3.5 that, *speedup* is bounded by amount of serial fraction of a program (f). In distributed video processing application, data communication is the serial component. Hence based upon data transfer speed and compute capability of nodes, we can decide whether it is feasible to run a particular application on distributed platform or not. If compute time is much greater as compared to data communication time, then *speedup* will be almost linear as we keep on adding more computing nodes. However if computing time is comparable to data communication time, *speedup* will be sub-linear and even may reach value less that unity after some point of time.

Results of all the experiments those were carried out on TI-DSP lab multimedia cluster are presented in this chapter. When JM-18.6 [56] H.264 encoder was used for parallel raw-video encoding, we observed almost linear *speedup*. Section 6.1 presents detailed results for parallel video encoding using JM-18.6 H.264 encoder. Section 6.2 presents results obtained on shared memory multi-core system using *x264* software [57]. Finally section 6.3 and 6.4 presents results for MPEG2 to H264 and HEVC transcoding on TI-DSP lab multimedia cluster respectively.

6.1 Parallel Implementation of JM

JM-18.6 is a reference H.264 encoder software provided by Fraunhofer Heinrich Hertz Institute. It is not optimized for any particular architecture hence it takes longer time to encode even CIF resolution videos. However it gives flexibility to change almost all the parameters for encoding. Table 6.1 lists different parameters set while carrying out this experiment. Here *chunk size* means number of frames to be sent at a time to a node.

Table 6.2 enlists results when parallelized version of JM-18.6 reference software was run on DSP-lab multimedia cluster. Column 4 of table 6.2 shows actual measured

Table 6.1: Experiment (using JM-18.6) Parameters

encoding parameters	video resolution	320x240
	video type	nptel lecture
	GOP size	8
	Motion estimation search size	16
	B frames	ON
	Quality parameter	30
media cluster parameters	chunk size	32 frames
	chunk size (in MB)	3.686MB
	serial time (for 32 frames)	48.231sec
	normalization factor	1.2407
	effective serial time	59.839sec
	Ethernet speed	12MBps
	chunk transfer time	0.3072sec

speedup. For *speedup* calculation, time taken by single processor to encode 32 frames was noted down and total time was multiplied by a "Normalization Factor" as discussed in section 5.1. Then using MPI environment encoder was run on multiple nodes and total frames were kept *number of nodes times* $*$ 32. Column 5 of table 6.2 shows estimated *speedup* calculated using equation 3.2 as discussed in section 3.5. Figure 6.1 plots number of nodes Vs *speedup* for this experiment which is quite linear and closer to the estimated values.

Table 6.2: Experiment (using JM-18.6) results

Number of node	Total frames to be encoded	time taken by multimedia cluster	relative speedup	speedup according to Gustafson's model
2	64	60.57	1.975894019	1.994892445
3	96	60.57	2.963841029	2.989784891
4	128	63.88	3.747022565	3.984677336
5	160	61.19	4.889683802	4.979569782
6	192	64.59	5.558750615	5.974462227
7	224	61.12	6.853397457	6.969354672
8	256	61.09	7.836300589	7.964247118
10	320	61.97	9.656277291	9.954032009
13	416	61.88	12.57141814	12.93870934
16	512	61.66	15.52771985	15.92338668

6.2 Parallel Implementation of x264 on Shared Memory

x264 is highly optimized open source H.264 encoder [57]. It uses multi-threading for video encoding. However to study what happens when we increase number of threads

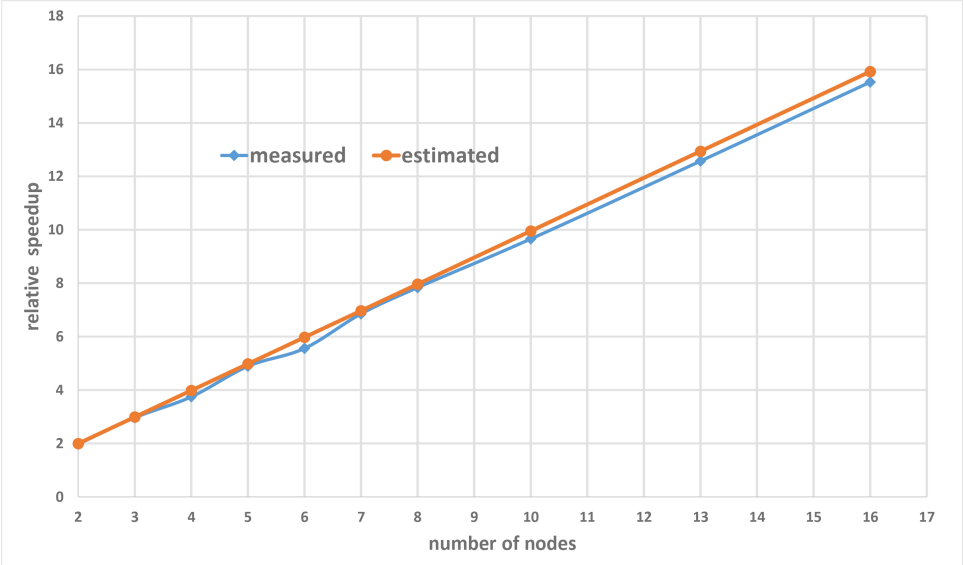

Figure 6.1: number of node Vs speed-up when 320x240 NPTEL video lecture was encoded using parallelized JM-18.6 encoder on DSP-Lab medial cluster

one-by-one, x264 was configured to use only single thread at a time. This time, Ultra High Definition (*UHD*) video with lot of spatio-temporal data was used for encoding purpose. Since we used UHD video, it was unfruitful to encode such a large raw video on distributed memory (since time required for data communication will be much more as compared to compute time), this experiment was carried out on a shared memory multi-core machine (details of encoder parameter and multi-core machine are given in table 6.3).

Table 6.3: Experiment (using x264) Parameters

encoding parameters	video resolution	2048x858
	video type	gravity movie
	GOP size	8
	B frames	ON
	Qp	30
shared memory system specifications	processor	Intel Core i7-4770
	frequency	3.4GHz
	Memory	8GB
	number of cores	8

Figure 6.2 shows best case *speeup* Vs number of threads for static and dynamic load balancing algorithm as discussed in chapter **??** section 3.3. We can see that performance is saturating after number of threads goes beyond 3. This happens because of data communication itself is taking too much time as compared to useful computation. To study effect of variation in *chunk size* on *speedup* keeping other variables constant, plot shown in figure 6.3 was drawn. We can observe that there exists an optimized number of frames to be transmitted at a time (*chunk size*) which is equal to 16.

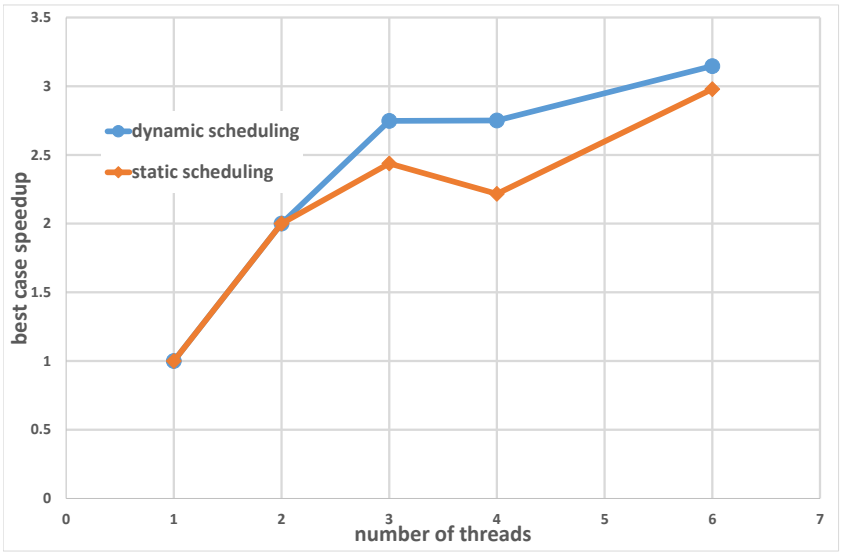

Figure 6.2: Number of node Vs speed-up when 2048x858 'gravity' video was encoded using x264 encoder on shared memory architecture

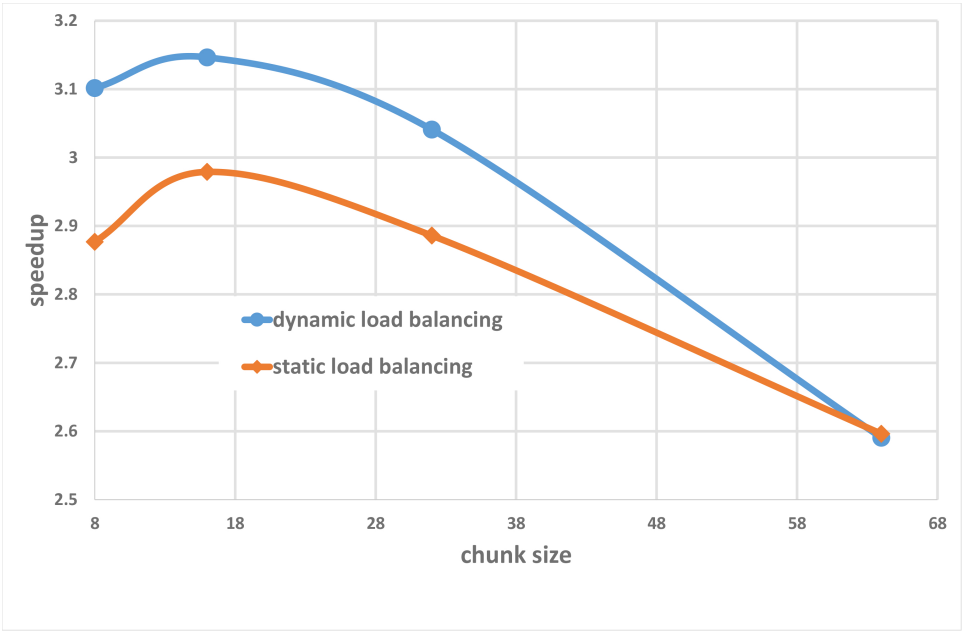

Figure 6.3: *Chunk Size* Vs *speedup* on shared memory system with number of threads kept constant to 6 for "gravity 2048x858" video

6.3 Parallel Transcoding using ffmpeg and libx264

FFmpeg is a command-line program to encode and decode video and audio streams. As mentioned in previous section, if we want to encode HD or UHD video on distributed system, we can not send raw-video data chunks over Ethernet port to worker nodes since data communication will take much more time as compared to actual encoding time. Hence instead of sending raw-video data to nodes, we decided to use lower CPU intense video codec to losslessly compress raw video and then send desired number of GOPs to nodes. This saved lot of data communication time. After extensive survey and practicality issues, MPEG2 encoder was chosen for compressing raw-video data for transmitting to the worker nodes.

Table 6.4: FFmpeg transcode summery

input video resolution	1920x1080
video type	edX video lecture
MPEG2 encoding FPS on the Master node	44 FPS
compression ratio after MPEG2 encoding	$\sim$136
H.264 encoding FPS on the Master with FFmpeg	4 FPS

From table 6.4, we can conclude that MPEG2 provides decent compression ration without much degradation in quality. Since master node is able to encode raw video using MPEG2 encoder with 44 FPS (*Frames Per Second*), we can assume that MPEG2 encoded stream is already available to us in real time so that we can focus more on transcoding of the same video. Master nodes can encode H264 video with maximum 4 FPS. Now we can use our TI-DSP lab multimedia cluster for speeding-up H264 encoding.

Figure 6.4 shows how *speedup* varies with respect to increase in number of nodes for both dynamic and static load balancing based algorithms. As expected, static load balancing based algorithm performs poor as compared to dynamic load balancing based approach for higher number of nodes. Figure 6.5 shows how efficiency per node varies for two different algorithms as we increase number of nodes. Efficiency per node is defined as

$$\text{Efficiency per node } (E_p) = \frac{\text{speedup}}{N}$$

where,

$$N \implies \text{Number of worker nodes}$$

Table 6.5 enlists different measurements made by varying GOP size and chunk size in combination. This gives an idea of how much FPS can be extracted out at the loss of increased bit-rate for different size of GOP. This information may be very useful for optimizing amount of computing resources, speed of video processing and output bitrate.

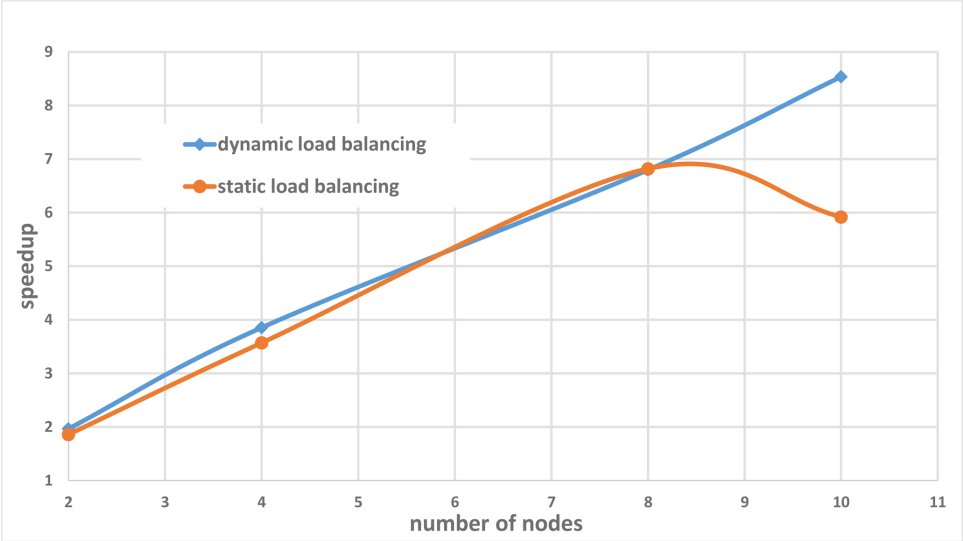

Figure 6.4: number of nodes Vs *speedup* plot when 1920x1080 'edX' lecture video was transcoded using ffmpeg on TI-DSP lab cluster

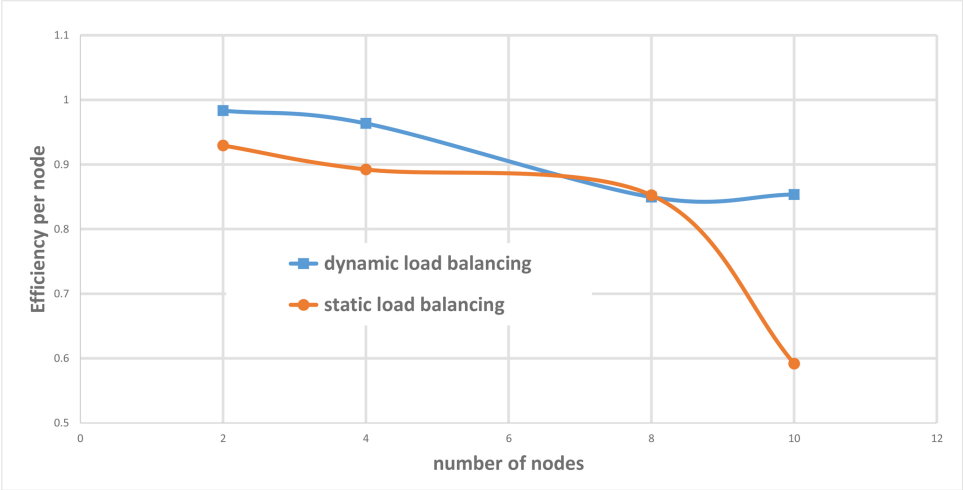

Figure 6.5: efficiency per node when 1920x1080 'Edx' video lecture was transcoded using ffmpeg on TI-DSP Lab cluster

Table 6.5: FFmpeg encoder in constant quality factor mode with *CRF* = 20 and *Chunk Size* = 32 frames

#frames	#nodes	GOP size	avg. bitrate (MBps) (size*30)/#frames	total encoding time		FPS (#frames/time)	
				static load balancing	dynamic load balancing	static	dynamic
1024	4	4	0.876	78.38	72.63	13.06456	14.09886
		8	0.662	90.55	83.86	11.30867	12.21083
		16	0.577	109.91	102.02	9.316714	10.03725
		32	0.524414063	123.39	115.37	8.29889	8.875791
1024	8	4	0.864257813	41.25	41.58	24.82424	24.62722
		8	0.662109375	47.39	47.55	21.60793	21.53523
		16	0.577148438	58.47	58.02	17.51325	17.64909
		32	0.524414063	65.94	65.31	15.52927	15.67907
1600	10	4	0.91875	80.16	57.13	19.96008	28.0063
		8	0.67875	91.89	63.72	17.41212	25.10986
		16	0.615	109.39	73.34	14.62657	21.8162
		32	0.57375	118.33	93.93	13.52151	17.03396

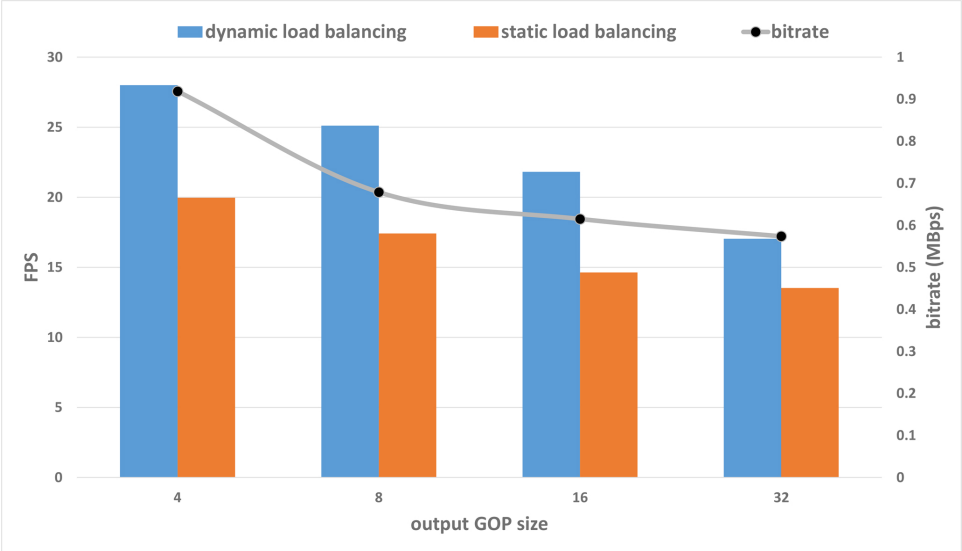

Figure 6.6: GOP size Vs FPS Vs bit-rate tradeoff for 1920x1080 'edX' video lecture transcoded using ffmpeg on TI-DSP lab multimedia cluster

Figure 6.6 gives a very good insight of bit-rate Vs transcoding speed Vs number of worker nodes trade-off. As we can clearly see that if we want to achieve real-time video transcoding (i.e. $FPS > 25$) with limited computing capability of TI-DSP lab multimedia cluster, we need to keep GOP size to be equal to 4, which hampers compression efficiency H.264 encoder.

6.4 Parallel Transcoding using ffmpeg and libx265

High Efficiency Video Codec (HEVC) [58] also known as H.265 is the generation next video codec standard claims to have up to 50% bit-rate reduction as compared to its predecessor H.264 video codec while maintaining the same visual quality, as shown in figure 6.7 (adapted from [59]). Some experiments were carried out with HEVC encoder using 'libx265' library of FFmpeg. This very high compression ratio of HEVC comes at the cost of increases CPU intensive computations.

Figure 6.8 shows how FPS changed as we increased number of nodes in TI-DSP lab multimedia cluster. Maximum FPS that we could get out of limited available resources was 4.65 FPS. We need to add more high performance computing resources to our DSP-lab multimedia cluster if we want real-time encoding with HEVC encoder.

6.5 CPU Utilization

CPU utilization indicates amount of "useful" work done by a CPU over the period of time. If CPU utilization is less for a particular application, it clearly indicates that particular application in not the optimized one and CPU is wasting its time in waiting for i/o.

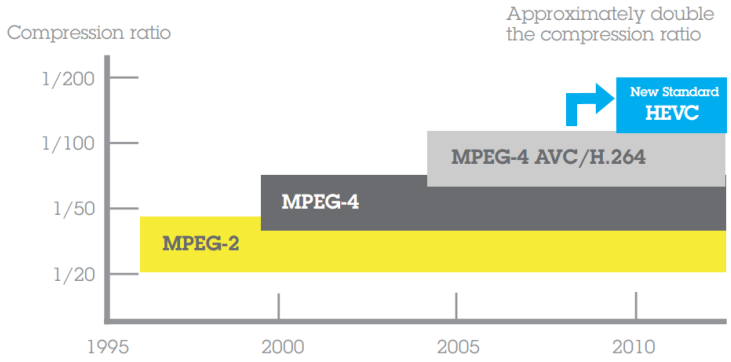

Figure 6.7: Compression ratio comparison

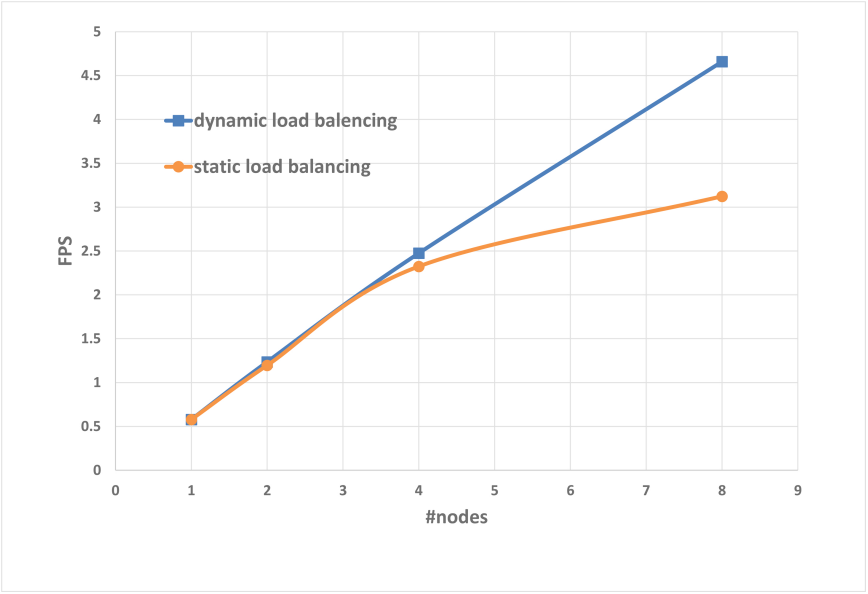

Figure 6.8: FPS Vs number of nodes for libx265 library

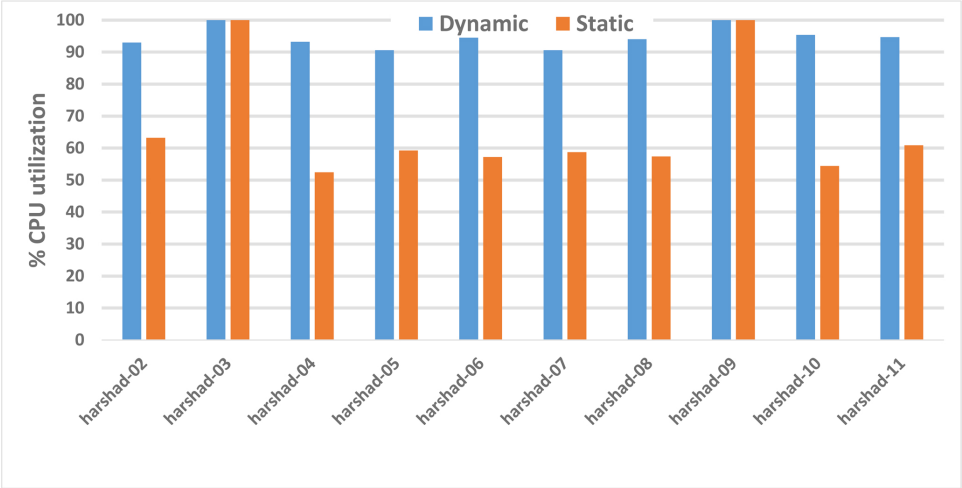

Figure 6.9: CPU utilization for each worker node

We have already seen in previous sections that dynamic load balancing algorithm works better as compared to static one almost in all the cases. However measuring CPU utilization gives more quantified results about how one algorithm performs better in comparison to the other one. In figure 6.9, we have plotted CPU utilization of all the worker nodes for a test run of video transcoding job. Following command was used for the real-time profiling of the worker nodes.

```
:~$ sar 1 > statistics_nonblock.txt &
//this command logs %CPU utilization data into
    "statistics_nonblock.txt" file on each worker node
```

As we can see from figure 6.9, CPU utilization for dynamic load balancing algorithm is always above 90% where as it is on an average 60% for static load balancing algorithm.

Figure 6.10 shows number of frames processed by individual worker nodes. As we have seen in section 5.1, TI-DSP lab multimedia cluster is a heterogeneous in nature (i.e. all worker nodes do not have the same computing power). Figure 5.2 indicates relative computing power for each node. If we compare both the above mentioned figures, the node with more computing power gets more number of frames for processing and vice-versa in the case of dynamic load balancing algorithm. Where as number of frames processed per node remains the same for static load balancing algorithm.

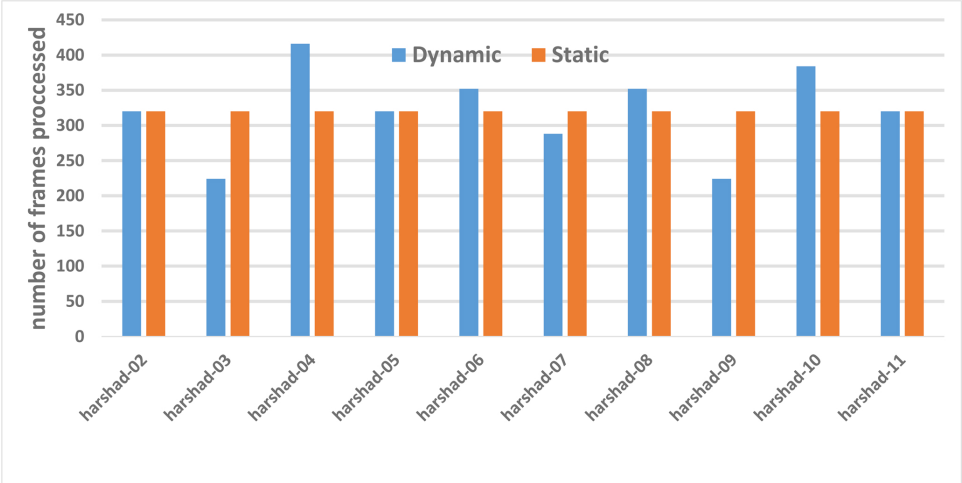

Figure 6.10: number of frames processed by each worker node

Chapter 7

Conclusions and Future Work

7.1 Conclusion

Raspberry Pi cluster

The usage of Cloud Platform for Distance Education demanded multiple ideas to be used together in the cluster of Raspberry Pis. For example, some sort of virtualization of the Raspberry Pis was required along with the usage of a server to manage file systems to store large volumes of video lectures, which can be easily retrieved and transmitted upon request within the network, etc. Hence, multiple ideas were executed parallelly with some ideas successfully completed, while others stuck at a crucial step waiting to be debugged. The current scenario of the multiple ideas are explained in detail in this section.

The report started with the replication of the Glasgow Raspberry Pi Cloud Cluster which involved the usage of LXC for the creation of isolated Containers, rather than the usage of Hypervisors which is the case with the servers in a Data Center. The replication involved the installation of LXC on Raspbian after compiling the Linux Kernel with suitable kernel patches that would support LXC. This process took a lot of time than expected and finally LXC was installed on an older version of Raspbian (2013-02-09-wheezy-raspbian) due to issues with LXC. A container with a minimal configuration was successfully created. However, the container fails to start as of this writing, the debugging of which could not be completed due to lack of time. If the container had started, the virtualization of Raspberry Pi would be successful and processes such as Webserver and Databases could have been started in the container. This would finish the task of serving the clients who wish to view the video lectures by setting up a website with links.

Containers in a Raspberry Pi were instead started through Docker, the only drawback being, Docker as of now is only compatible with Arch Linux. Arch Linux required a very steep learning curve with little documentation available on the internet. However due to the good documentation available on the internet, Docker was successfully installed in one of the Raspberry Pis and a container was started. This container had a Raspbain Jessie image with Apache2 webserver installed. A simple website is also developed for demonstration purposes. When a client logs on to this Raspberry Pi, the website is shown.

One of the Raspberry Pi was also set up as a server for Network Attached Storage i.e. a folder in a 2 TB hard disk was interfaced to a Raspberry Pi. This hard disk can now be used to store video lectures that the Webserver in one of the containers would request for (on request from the client). Every video lecture in the hard disk needs to be given a link in the website that the Webserver would display to the client. The setting up of NAS was successfully completed on another networked Raspberry Pi.

The Raspberry Pis were also used as a cluster for parallel execution of a program through MPI. Any C or Python program can be made available to all the Raspberry Pis (in the same path) and can be executed parallelly. If a transcoding program were to be executed, a video can be converted from one format to another parallelly, by utilizing all the Raspberry Pis.

Multimedia cluster for video encoding and transcoding

In the present work, different techniques of using distributed computing system for video encoding and transcoding have been explored. An extensive study of numerous encoder parallelization techniques and feasibility of their implementation on distributed computing system was done as a part of this dissertation work. Further a brief survey of different load balancing techniques to extract out maximum performance out of distributed computing system was also carried out.

A testbed for an experimentation of different algorithms for parallel encoding and transcoding videos on distributed computing system was implemented inside TI-DSP lab. State of the art message passing system, such as *open-MPI*, as well as optimized video encoder libraries, such as *libx264* and *libx265*, were used for the efficient implementation of the multimedia cluster. In the end various experiments were carried out to study effects of different load balancing algorithms and data chunk size on the multimedia cluster performance.

7.2 Future Work

Raspberry Pi cluster

As mentioned, once LXC was installed on Raspbian, a container could just be created but not started. This problem has to be solved first. Once solved, a Webserver can be started on one of the containers. A simple website can be developed with a link to a video lecture (that is stored in the 2 TB hard disk). The Webserver code also needs to be developed using PHP so as to retrieve the corresponding video from the hard disk.

The website developed and launched in one of the containers of the Raspberry Pi (using Docker) can be further developed by providing links to the video lectures stored in the hard disk. Also, the NAS can be moved to another container in the same Raspberry Pi. A

method has to be found out to link the 2 containers (one with the Webserver and the other with NAS) so that video can be transmitted to the client on request.

Further, the Webserver can be made secure by providing a login check facility by starting a database. The Distance Education Website can be made realistic by hosting many video lectures and checking the capacity of the Raspberry Pi Cloud cluster thus created by increasing the number of clients.

Multimedia cluster for video encoding and transcoding

A foundation stone to build a cloud based real-time video encoding and transcoding system was put in this dissertation work. This can further be extended to a real-time Ultra HD video conferencing. Some of the suggested changes to the current implementation are as following

- Effective implementation of GOP level parallelism by incorporating better load balancing algorithms. Size of data chunk sent at a time to a computing node can also be varied adaptively for better results.
- Migrating TI-DSP lab multimedia cluster to public cloud servers so that we can get sufficient number of computing nodes in order to real-time transcode UHD videos in HEVC format without investing a lot of money into the current infrastructure.
- Extending implemented platform for scalable video codecs which are more suitable for heterogeneous networks.
- Using Graphic Processing Units (GPU) for video encoding and transcoding purpose as suggested by Tse Kai Heng et al [60] to further extract fine grain data level parallelism.

Bibliography

[1] Tso, F. P., White, D. R., Jouet, S., Singer, J., & Pezaros, D. P. (2013, July). The Glasgow Raspberry Pi Cloud: A Scale Model for Cloud Computing Infrastructures. In *Distributed Computing Systems Workshops (ICDCSW), 2013 IEEE 33rd International Conference on* (pp. 108-112). IEEE.

[2] Abrahamsson, P., Helmer, S., Phaphoom, N., Nicolodi, L., Preda, N., Miori, L., ... & Bugoloni, S. (2013, December). Affordable and Energy-Efficient Cloud Computing Clusters: The Bolzano Raspberry Pi Cloud Cluster Experiment. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on* (Vol. 2, pp. 170-175). IEEE.

[3] People Sector Infocomm Resource Centre (PSIRC). (2014). Seminar on Collaboration as a service Cloud Computing. Retrieved from http://www.psirc.sg/events/seminar-on-collaboration-as-a-service-cloud-computing

[4] Adrian Bridgwater. (8 April, 2011). Migrating legacy applications: if it aint broke, dont move it. Retrieved from http://www.itpro.co.uk/632629/migrating-legacy-applications-if-it-aint-broke-dont-move-it

[5] Raspberry Pi Forums. (n.d.). Operating System Distributions. Retrieved from http://www.raspberrypi.org/forums/viewforum.php?f=18

[6] David Hayward. (May 9th 2013). Raspberry Pi operating systems: 5 reviewed and rated. Retrieved from http://www.techradar.com/news/software/operating-systems/raspberry-pi-operating-systems-5-reviewed-and-rated-1147941

[7] J.A. Watson. (January 2, 2014). Raspberry Pi: Hands On with Arch Linux and Pidora. Retrieved from http://www.zdnet.com/raspberry-pi-hands-on-with-arch-linux-and-pidora-7000024688/

[8] Gordon Haff. (September 19, 2013). What are containers and how did they come about? Retrieved from http://bitmason.blogspot.in/2013/09/what-are-containers-anyway.html

[9] David Davis. (May 12, 2013). The Top 5 Enterprise Type 1 Hypervisors You Must Know. Retrieved from http://www.virtualizationsoftware.com/top-5-enterprise-type-1-hypervisors/

[10] Mike Wheatley. (Aug 22, 2014). Deadly Docker: Why containers are a threat to cloud virtualization. Retrieved from http://siliconangle.com/blog/2014/08/22/deadly-docker-why-containers-are-a-threat-to-cloud-virtualization/

[11] Steven J Vaughan-Nichols. (April 8, 2014). Why Containers Instead of Hypervisors? Retrieved from http://blog.smartbear.com/web-monitoring/why-containers-instead-of-hypervisors/

[12] Raspberry Pi Downloads. (n.d.). Downloads. Retrieved from https://www.raspberrypi.org/downloads/

[13] David R. White. (March 12, 2013). Building an LXC-friendly Kernel for the Raspberry Pi. Retrieved from https://raspberrypicloud.wordpress.com/2013/03/12/building-an-lxc-friendly-kernel-for-the-raspberry-pi/

[14] David R. White. (March 12, 2013). Creating an LXC Container on the Raspberry Pi. Retrieved from https://raspberrypicloud.wordpress.com/2013/03/12/creating-an-lxc-container-on-the-raspberry-pi/

[15] Raspberry Pi Beta. (Jun 11, 2014). Docker on Raspbian? Retrieved from http://raspberrypi.stackexchange.com/questions/15389/docker-on-raspbian

[16] Steven J. Vaughan-Nichols. (August 4, 2014). What is Docker and why is it so darn popular? Retrieved from http://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/

[17] Wikipedia. (n.d.). Docker (Software). Retrieved from http://en.wikipedia.org/wiki/Docker

[18] Docker. (n.d.). What is Docker? Retrieved from https://www.docker.com/whatisdocker/

[19] Resin. (Nov 22, 2013). Docker on Raspberry Pi in 4 Simple Steps. Retrieved from http://resin.io/blog/docker-on-raspberry-pi-in-4-simple-steps/

[20] Docker. (n.d.). Welcome to the Docker User Guide. Retrieved from https://docs.docker.com/userguide/

[21] Docker. (n.d.). The best way to understand Docker is to try it! Retrieved from https://www.docker.com/tryit/

[22] CoreOS. (n.d.). Getting Started with Docker. Retrieved from https://coreos.com/docs/launching-containers/building/getting-started-with-docker/

[23] Docker. (n.d.). Working with Containers. Retrieved from https://docs.docker.com/userguide/usingdocker/

[24] Dabs (n.d.). Network Attached Storage (NAS) Explained. Retrieved from http://www.dabs.com/learnmore/components-and-storage/network-attached-storage-%28nas%29-explained/

[25] Tinkernut. (n.d.). How to make a Raspberry Pi NAS (Network Attached Storage). Retrieved from http://www.tinkernut.com/portfolio/make-raspberry-pi-nas-network-attached-storage/

[26] Tinkernut. (April 27, 2014). Make your own Cluster Computer. Retrieved from https://www.tinkernut.com/2014/04/make-cluster-computer/

[27] Tinkernut. (April 27, 2014). Make your own Cluster Computer (Part 2). Retrieved from https://www.tinkernut.com/2014/05/make-cluster-computer-part-2/

[28] "Visual Networking Index Forecast 2013−2018," Cisco., *http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf*

[29] "The State of the Internet," *Akamai*, vol.6, num.3, 3^{rd} Quarter, 2013. *http://www.akamai.com/dl/akamai/akamai-soti-q313.pdf*

[30] Correa, G.; Assuncao, P.; Agostini, L.; da Silva Cruz, L.A., "Performance and Computational Complexity Assessment of High-Efficiency Video Encoders," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.22, no.12, pp.1899,1909, Dec. 2012

[31] Kamaci, N.; Altunbasak, Y., "Performance comparison of the emerging H.264 video coding standard with the existing standards," *International Conference on Multimedia and Expo, 2003. ICME '03. Proceedings*. 2003, vol.1, no., pp.I,345-8 vol.1, 6-9 July 2003

[32] S. Saponara; K. Denolf; G. Lafruit; C. Blanch; J. Bormans, "Performance and Complexity Co-evaluation of the Advanced Video Coding Standard for Cost-Effective multimedia communication", *EURAPIS Journal on Applied Signal Processing*, pp. 220-235, 2004:2.

[33] Mohammed Faiz Aboalmaaly; Adel Nadhem Naeem; Hala A. Albaroodi; Sureswaran Ramadass, "Parallel H.264/AVC Encoder: a Survey" *IJACT: International Journal of Advancements in Computing Technology*, Vol. 5, No. 9, pp. 334 - 341, 2013

[34] S Sankaraiah; HS Lam; C Eswaran; J Abdullah, "GOP level parallelism on H. 264 video encoder for multicore architecture", *Int. Conf. on Circuits, System and Simulation (IPCSIT)*, 2011

[35] Seongmin Jo; Song Hyun Jo; Yong Ho Song, "Exploring parallelization techniques based on OpenMP in H.264/AVC encoder for embedded multi-core processor", *Journal of Systems Architecture: the EUROMICRO Journal*, Volume 58 Issue 9, Pages 339-353. October 2012

[36] Yen-Kuang Chen; Xinmin Tian; S. Ge, M. Girkar, "Towards efficient multi-level threading of H.264 encoder on Intel Hyper-Threading architectures", *International Parallel and Distributed Processing, Symposium*, 2004.

[37] Bingjie Han; Ronggang Wang; Zhenyu Wang; Shengfu Dong; Wenmin Wang; Wen Gao, "HEVC decoder acceleration on multi-core X86 platform," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol., no., pp.7353,7357, 4-9 May 2014

[38] http://www.fileformat.info/format/mpeg/egff.htm

[39] Youngsub Ko; Youngmin Yi; Soonhoi Ha, "An efficient parallel motion estimation algorithm and X264 parallelization in CUDA," *Conference on Design and Architectures for Signal and Image Processing (DASIP)*, vol., no., pp.1,8, 2-4 Nov. 2011

[40] Mei Wen; Ju Ren; Nan Wu; Huayou Su; ChangQing Xun; Chunyuan Zhang, "Data Parallelism Exploiting for H.264 Encoder," *International Conference on Multimedia and Signal Processing (CMSP)*, vol.1, no., pp.188,192, 14-15 May 2011

[41] Rodriguez, A.; Gonzalez, A.; Malumbres, M.P., "Hierarchical Parallelization of an H.264/AVC Video Encoder," *International Symposium on Parallel Computing in Electrical Engineering*, vol., no., pp.363,368, 13-17 Sept. 2006

[42] S. Sankaraiah; Lam Hai Shuan; C. Eswaran; Junaidi Abdullah, "Performance Optimization of Video Coding Process on Multi-Core Platform Using Gop Level Parallelism", *International Journal of Parallel Programming*, vol. 42, issue 6, pp 931-947, December 2014

[43] Mell P.; Grance T., "The NIST definition of cloud computing", *Special publication (Draft)*, pp. 800145. January 2011

[44] http://www.ibm.com/cloud-computing/in/en/what-is-cloud-computing.html

[45] B. Yagoubi; Y. Slimani, "Task load balancing strategy for grid computing", *Journal of Computer Science*, vol. 3 (3), pp. 186194, 2007

[46] Dhinesh Babu L.D; P. Venkata Krishna, "Honey bee behavior inspired load balancing of tasks in cloud computing environments", *Applied Soft Computing*, vol. 13, issue 5, pp. 2292-2303, May 2013

[47] G.M. Amdahl, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities" *Proc. Am. Federation of Information Processing Soc. Spring Joint Computer Conf.* (AFIPS 07), AFIPS Press, pp. 483-485, 1967

[48] John L. Gustafson. "Reevaluating Amdahl's law".*Communications of the ACM*, vol. 31, issue 5, pp 532-533, May 1988

[49] MPI Forum. MPI: A Message-Passing Interface Standard. Version 2.1, *www.mpi-forum.org*, Sept. 2008

[50] The MPI standard: A specification for MPI implementations. *http://www.mcs.anl.gov/research/projects/mpi/*, as accessed on Jan. 2015

[51] Cornell Virtual Workshop, *https://www.cac.cornell.edu/VW/mpip2p/SyncSend.aspx*, as accessed on Feb. 2015

[52] Open MPI v1.8.4 documentation, *https://www.open-mpi.org/doc/v1.8/*, as accessed on Feb. 2015

[53] FFmpeg official webpage, *https://www.ffmpeg.org/*

[54] FFmpeg Protocols Documentation, *https://www.ffmpeg.org/ffmpeg-protocols.html*

[55] FFmpeg installation guide for Ubuntu, Debian or Mint OS, *https://trac.ffmpeg.org/wiki/CompilationGuide/Ubuntu*

[56] JM Reference Software 18.6, *http://iphome.hhi.de/suehring/tml/download/old_jm/jm18.6.zip.*, as accessed on Jan. 2015

[57] x264 software, *http://www.videolan.org/developers/x264.html*

[58] G.J. Sullivan, "Overview of the High Efficiency Video Coding (HEVC) Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649-1668, Dec., 2012

[59] "Accelerating HEVC adoption with DivXs end-to-end solution," 2014, available at *http://www.divx.com/files/hevc-markets-whitepaper-feb2014.pdf*

[60] Tse Kai Heng; Asano, W.; Itoh, T.; Tanizawa, A.; Yamaguchi, J.; Matsuo, T.; Kodama, T., "A highly parallelized H.265/HEVC real-time UHD software encoder," *IEEE International Conference on Image Processing (ICIP)*, vol., no., pp.1213,1217, 27-30 Oct. 2014