A Research Project Report

Design and Implementation of Distributed Supercomputing platform for ASTU to support scientific research with aggregated Gigaflops

Project Team:

1. Mr. Dereje Regassa(PI)
2. Mr. Mohd. Wazih Ahmed
3. Mr. Zelalem Mihret
4. Mr. Dileep Kumar G.

አዳማ ሳይንስ እና ቴክኖሎጂ ዩኒቨርሲቲ
Adama Science & Technology University

School of Electrical Engineering & Computing
Computer Science & Engineering Program

1. Dr. Tilahun Malak

   Reviewer I

   _____

   Signature

2. Mr. Desta Zerihun

   Reviewer II

   _____

   Signature

3. Mr. Girma Debele

   Assoc. Dean for Research
   & Technology Transfer, SoEEC

   _____

   Signature

Abstract

Today, distributed supercomputing is a buzzword for the universities and organizations and most of the parallel computing researchers are focused on harnessing the power of commodity processors and even internet computers to aggregate their computation powers to solve the computationally complex problems. The advantage of using flexible commodity cluster for supercomputing workloads over a dedicated supercomputer as well as expensive HPC infrastructure. The cost effective and scalable nature can better utilize the available resources of an organization. It can benefit researchers who wish to carry out huge amounts of repetitive calculations on large amounts of data and wish to obtain valid results in a reasonable time. This research is a pilot research project which has been commenced with an objective to implement HPC based supercomputing facility in ASTU campus. In this research, the research team has provided two separate implementations for cluster based supercomputing- Hadoop and Spark based HPC cluster primarily for data intensive jobs, while torque based cluster for MIMD workloads. The performance of these clusters was measured with an extensive experimentation. However with implementation of MPI the performance of Spark and Torque clusters shall be compared in full scale implantation of this research. The team of researchers have concluded that the specific application or job can be chosen to run based on computation parameters on implemented clusters.

Keywords: HPC, shared memory, optimization, commodity hardware.

Contents

List of Figures

List of Tables

Revision History

Version	Date	Remarks
1.0	10-APRIL-17	Initial Draft
1.1	21-APRIL-17	Revision
2.0	24-APRIL-17	Final Doc.
3.0	30-MAYL-17	Revised Doc.

Glossary

Abbreviation	Explanation
HDFS	Hadoop Distributed File System
MR	Map Reduce
RAM	Random Access Memory
RDD	Resilient Distributed Dataset
VM	Virtual Machine
YARN	Yet Another Resource Negotiator
HPDA	High Performance Data Analytics

Chapter 1

Introduction

1.1 Background and Motivation

Today distributed supercomputing is a buzzword for the universities and organizations and most of the parallel computing researchers are focused on harnessing the power of commodity processors and even internet computers to aggregate their computation powers to solve the computationally complex problems. The floating point operations (FLOPs) [1, 2] are used in scientific computing community to measure the processing power of individual computers, different types of HPC, Grid and supercomputing facilities. Today we can find a few hundred GigaFlops on a multi-core processor, while an HPC cluster can generate the few teraflops of operations, Cray XC series supercomputers can generate Petaflops of operations, and there is world's number one supercomputer of this year, Tihane-II, Chinese supercomputer, which can flex its muscles at the level of ExaFlops of operations per second. Indian space and research organization has invested with an aim of hundreds of ExaFlops of computing power in the nearest future. The Chinese supercomputer is developed by national defense university [3] and involves distributed clustered architecture. Similarly KAUST, a leading university of Saudi Arabia [4] has installed Shaheen-II supercomputer from Cray XC series.

However, purchase of a supercomputer for academic and collaborative research may cost an investment of 30 Million USD to 150 Million USD (KAUST Shaheen- II Budget), and therefore most of the universities avoid directly purchasing the supercomputers [3,4]. But in order to provide a platform for the students and researchers to work on the most challenging scientific problems related to different fields such as computational fluid dynamics, medical, imaging, graphics, higher dimensional visualization, big data, parallel machine learning and data mining, multidisciplinary optimization etc. the various universities have implemented low cost supercomputing using the clusters of Intel processors.

The normal computer's fails to process the complex jobs with very large data and computations, as well as the stand alone software's also start complaining about size of solution set, memory requirements and millions of times complex function evaluation. For example a ray tracing experiment in computer graphics of simple tea-pot may take 8 hours to render on a single multi-core processor.

In order to solve these problems, scientists have invented cost effective solutions to aggregate individual gigaflops of individual computers by be-wolf Linux clustering, and in production environment special HPC Racks are installed along with high speed Infiniband Network to facilitate the high speed I/O and communication. Most of the universities hosts the bewolf clusters and provide a platform to perform the large scale experiments.

Applications of low cost supercomputing:
While previously most of the large scale applications demanded a supercomputer in physical form, but the advent of the distributed, clustered and grid type of environments have leveraged the researchers to aggregated the computational power of existing commodity processors. Following list shows the applications of distributed low cost supercomputing:

- Earthquake simulations
- Folding proteins
- Weather forecasting
- Predicting climate changes
- Fluid dynamics
- Multidisciplinary Optimization
- Aerodynamic research
- Drug design
- Understanding viruses and diseases
- Population growth simulation
- Combinatorial optimization
- Big data analysis
- Distributed machine learning
- Large scale data mining
- Astronomic simulations
- Economics

In addition to this small list of problems there are 20 problems which are still a challenge to be solved by the scientists in real time; these problems include the approximations to various NP complete problems.

1.2 Problem Statement

Industry estimates that we are creating more than 2.5 Quintillion bytes of data every year [5]. It is difficult to imagine this scale of data generation. While this pace of data generation is very exciting, it has created entirely new set of challenges and has forced us to find new ways to handle Big data effectively. Challenges associated with big data and computation intensive can be classified in following categories [6]:

- **Challenges in data capturing**: Capturing huge data could be a tough task because of large volume and high velocity. There are millions of sources emanating data at high speed. To deal with this challenge, we have created devices which can capture the data effectively and efficiently. For example, sensors which not only sense data like temperature of a room, steps count, weather parameters in real time, but send this information directly over to cloud for storage.

- **Challenges with data storage**: Given the increase in data generation, we need more efficient ways to store data. This challenge is typically dealt by combination of various methods including increasing disk sizes, compressing the data and using multiple machines, which are connected to each other and can share data efficiently.
- **Challenges with Querying and Analysing data**: This is probably the most difficult task at hand. The task is to not only to retrieve the past data, but also coming out with insights in real time (or as little time as possible). To handle this challenge, we can look at several options. One options is to increase the processing speed. However, this normally comes with increase in cost and can not scale as much. Alternately, we can build a network of machines or nodes known as 'Cluster'. In this scenario, we first break a task to sub-tasks and distribute them to different nodes. At the end, we aggregate the output of each node to have final output.

Hence, there is need of high performance distributed supercomputing platforms which can process Big Data efficiently and fastly with low cost commodity hardware. In a university environment, HPC is expected to perform CPU bound operations with heavy asymmetric load on compute nodes, therefore the design of HPC cluster in a research environment, computing power of the nodes as well as message passing infrastructure between the processes plays an important role in the performance of application programs.

Most of the current research in ASTU is performed on isolated computers, but various departments and individuals engaging in large scale research projects and scientific simulations need HPC in the campus. This pilot project explores the various prototypes of cluster management platforms and provides an opportunity for full implementation of the HPC system after the pilot study is completed.

1.3 Objectives of the Study

The main objective of the proposed study is to develop cost effective, distributed supercomputing platform which can be used by various researchers at Adama Science and Technology (ASTU)

The specific objectives of the proposed study are:

- To implement a cluster of 30 computers approximately to attain an aggregated Giga flops processing power.
- To configure a Cluster Management Software with a suitable Operating System.
- To perform scientific experiments on newly implemented cluster in relation with distributed optimization, machine learning etc.
- To compare the practical performance of the proposed system under different configurations, considering various algorithms and load.
- To research the methods to optimize the performance of considered algorithms in proposed environment by maximization of system utilization.

1.4 Research Questions

The following research questions are formulated for this work:

1. What is the role of commodity hardware in HPC and How does performance of HPC cluster varies as the number of such compute nodes are increases?

2. What are the optimum host and network configuration to implement basic functionality of HPC cluster?

3. How does various parameter of Installed HPC cluster like reliability, latency profile and delay varies under various type of workloads.

1.5 Organization of Document

The remainder of the document is organized as follows.

- Chapter 2 presents a comprehensive description of related literature. Recent work with distributed computing when applying the Hadoop framework for scalability and availability is also referenced.
- In Chapter 3, there is a discussion on research methodology.
- Chapter 4 demonstrates the design and implementation of the proposed framework.
- Chapter 5 illustrates the experimentation conducted.
- Chapter 6 gives the results obtained and tests performed with discussions.
- Finally, Chapter 7 concludes with a summary of contributions and discuss future work.

Chapter 2

Literature Review

This chapter provides a literature review of the high performance computing in general and the history behind application of HPC cluster in scientific computing. Some of the representative papers of the current trends in supercomputing are reviewed.

2.1 High Performance Computing

High performance computing is defined in terms of distributed, parallel computing infrastructure with high speed interconnection networks and high speed network interfaces, including switches and routers which are specially designed in order to provide aggregate performance of many-core and multicore systems, computing clusters, cloud or in the form of a grid. While grid spans over a large area and utilize WAN protocols, clusters are suitable for performing scientific computations in a small area. Clusters can be categorized into many types based on the purpose and objectives of their implementation, including high availability cluster, load balancing clusters, failover clusters, and high performance clusters.

The objective of installation of HPC cluster is to provide a single computer view for seamless integration and aggregate computational resources, while hiding the task of resource management, scheduling and coordination on cluster management software. Basically a cluster management software work as a platform for providing basic operating services for distributed applications which need Teraflop level computing power. However the installation of HPC cluster with dedicated HPC rack from manufacturer like HP,IBM etc. with Marynet network card and Mallnox switches with 40-100Gbps supporting the infiniband wiring is recently used in many scientific projects [7–9]. A dedicated infiniband backbone is a costly installation and often used in production level environment and real time research and development organizations especially in meteorology, simulation, information retrieval and data science etc. therefore in the recent past various universities have started to implement the HPC cluster using one Gbps Ethernet backbone and cost effective commodity computers have been successfully used as compute nodes and client nodes.

Applications of HPC in scientific research have greatly increased in recent past as an alternative to the dedicated supercomputers [8, 10], as they are difficult to program, costly and they require sophisticated expertise. Supercomputers are generally standalone and scalability is a problem in supercomputer based data centres [11]. While HPC cluster based supercomputing platforms have been successfully implemented in various universities [8, 12] also there is a list of Open-Source cluster management software's [13]as well as some of the commercially available schedulers are also available in various domains.

2.2 Related Work

In 1991, US program HPCC was initiated with an objective to extend the leadership in the field of high performance computing, by providing key enabling technologies and dissemination of progress in the field. Later in 1996 after the US bill was passed named CIC , 12 federal agencies joined this effort including DARPA, NSF, NASA, NIH, NSA,NIST,NOAAEPA, AHPCR under the hood of National Coordination Centre (NCO) for high performance computing. Most of the development and progress work in HPC was made available at HPCC website as annual reports and this initiative led to significant improvement in HPC research. As a result of this coordinated dissemination of high quality research, various academic institutions, private companies and research groups have started to collaborate and contributed the literature of HPC. The major applications of HPC are seen in Data Storage and Analysis, Data Mining, Simulation and Modeling, Scientific Calculations, Bio-informatics, Big Data Challenges, Complex Visualizations etc. In this research most of the reviewed research is taken from application of HPC in scientific computing since the objective of this research is to provide a HPC based supercomputing infrastructure for various scientific computing applications.

The use of HPC cluster in solving a density estimation problem is described in [7]. This research is based on shared memory parallel execution of kernel density estimation algorithm on a GPU enabled system , the OpenMP library [14] is used in the implementation and various experiments are designed to compute the density of simulated Gaussian density function. The results of this study shows significant improvement in the time taken by HPC cluster in estimating the density function. While GPU based computing provide easy access to higher teraflops with smaller number of machines, but this setup is only suitable for embarrassingly parallel jobs, at the same time a fine grain parallelism, synchronization of threads and programming GPU enabled system is complex task. Therefore a number of studied are performed on Intel Xeon processor based HPC cluster. These clusters have advantages of being easy to program, support SIMD, MIMD models as well as provide low cost alternative to few teraflops with ease of scalability.

In addition to advances in hardware, communication infrastructure, HPC community have witnessed a growing set of cluster management solutions including Torque, Apache Hadoop, Open-Mosiac, Rocks, OSCAR, OpenPBS, Alchemy and HTcondor etc. Also there are a number of opensource libraries like OpenCV, CUDA, OenMP, OpenMPI, MPICH and a number of implementation of MPI programming paradigm on almost all the cluster platforms reviewed here. The libraries for individual research domain, corresponding benchmarks and the quantitative performance evaluation of benchmark problems are also available for most of the HPC implementations.

In this race of cost effective HPC implementation, various universities have developed their dedicated HPC cluster data centres and provide researchers access to the HPC resources for cutting edge research. Our research team have surveyed the HPC implementation of representative academic importance. The University of Columbia has implemented a 167 node cluster with 2672 cores on Dual Intel E5-2650v2 Processors (2.6 GHz) with Torque/Moab job scheduler in year 2009 which was upgraded to support research projects in various application areas [15], similarly Yale university provide HPC based computing environment with excellent publication record, Stanford HPC Centre provides million core compute nodes [16], FSU HPC cluster centre with more than 10000 cores and 201449 Gflops, with 3 million job capacity [17].

The current research has identified certain gaps in ASTU campus as unavailability of HPC platform works as a bottleneck for researchers in natural science, mechanical and material engineering as well as biological, genetic and structure simulation research. In addition to these departments, core computer science and electrical engineering research also need specialized HPC centre with commodity computers available in the SIG research laboratories for performing research on new domains like big data, distributed machine learning, internet of things and cloud

computing. Based on the literature review we have found the gaps in terms of various HPC advancements and specifically in context ASTU and on-going research projects.

2.2.1 Performance Comparison of OpenMP, MPI, MapReduce and Spark

OpenMP, MPI, and MapReduce are the most widely recognized parallel or distributed programming frameworks. The performance study of three parallel programming frameworks was done [18]. The comparative studies have been conducted for two problem sets: the all-pairs-shortest-path problem and a join problem for large data sets. OpenMP [19] is the de facto standard model from shared memory systems, MPI [20] is the de facto standard for distributed memory systems, and MapReduce [21] is recognized as the de facto standard framework intended for big data processing. For each problem, the parallel programs have been developed in terms of the three models, and their performance has been observed. The experiment results concluded that If a problem is small enough to be accommodated and the computing resources such as cores and memory are sufficient, OpenMP is a good choice. When data size is moderate and the problem is computation-intensive, MPI can be considered the framework. When data size is large and the tasks do not require iterative processing, MapReduce can be an excellent framework. OpenMP is the easiest to use because there is no special attention needed to be paid because it just need to place some directives in the sequential code. MapReduce is relatively easy to use once we can abstract an application into Map and Reduce steps. The programmers do not have to consider workload partitioning and synchronization. MapReduce programs, however, take considerable time for the problems requiring much iteration, like all-pairs shortest-path problem. MPI allows more flexible control structures than MapReduce; hence MPI is a good choice when a program is needed to be executed in parallel and distributed manner with complicated coordination among processes.

Message Passing Interface (MPI) is a language-independent communications protocol for parallel computing where point-to-point and collective communication are supported [21]. However, the standard does not currently support fault tolerance [20] since it mainly addresses High-Performance Computing (HPC) problems. Another MPI drawback is that it is not suitable for small grain level of parallelism, for example, to exploit the parallelism of multi-core platforms for shared memory multiprocessing. OpenMP, on the other hand, is an Application Programming Interface (API) that supports multi-platform shared memory multiprocessing programming on most processor architectures and operating systems [19]. OpenMP is becoming the standard for shared memory parallel programming for its high performance, but unfortunately, it is not suitable for distributed memory systems. The idea of extending this API to cope with this issue is now a growing field of research [22]. OpenMP's user-friendly interface allows to easily parallelize complex algorithms. The same thing cannot be said about MPI since the code must be heavily re-engineered in order to obtain relevant performance improvements.

Spark is a state-of-the-art framework for high performance parallel computing designed to efficiently deal with iterative computational procedures that recursively perform operations over the same data [23], such as supervised machine learning algorithms. It is based on the concept of maintaining data in memory rather than in disk as it is done by other well-known approaches such as Apache Mahout that require data reloading and incur considerable latencies. Experiments have shown Spark outperforms by up to two orders of magnitude conventional MapReduce jobs in terms of speed [24, 25]. The core data units in Spark are called Resilient Distributed Datasets (RDDs). They are a distributed, immutable and fault-tolerant memory abstraction that collects a set of elements in which a set of operations can be applied to either produce other RDDs (transformations) or return values (actions). RDDs can reside in memory, disk or in combination. However, they are only computed on actions following a Lazy Evaluation (LE) strategy, in order to perform minimal computation and prevent unnecessary memory usage. RDDs are not cached

in memory by default, therefore, when data are reused, a persist method is needed in order to avoid recomputation.

Various cluster management options are available for running Spark: they range from the simple Spark's integrated Standalone Scheduler to other widespread cluster managers such as Apache Mesos and Hadoop YARN [26]. The study [27] chose to deploy Spark in a Hadoop cluster. Apache Hadoop is an open-source software platform for distributed big data processing over commodity cluster architectures [28]. It has three main elements: a) a MapReduce programming model that separates data processing in mapping for performing data operations locally, shuffling for data redistribution over the network and reduction for data summarization; b) a distributed file system (HDFS) with high-throughput data access; and c) a cluster manager (YARN) in charge of handling the available computing resources and job scheduling. Nevertheless, Spark on Hadoop can be preferred [27] because it also:

- offers a distributed file system with failure and data replication management.
- allows the addition of new nodes at runtime.
- provides a set of tools for data analysis and management that is easy to use, deploy and maintain.

Chapter 3

Methodology

This research work was started with specific design goals and objectives restricted to the installation of a pilot HPC cluster for future research projects. We followed the standard methodology for IT/ICT projects during implementation of DeepStack and HPDA clusters. This research used a hybrid methodology involving qualitative, quantitative and empirical aspects of research at various stages. The two stages of this research are identified as (i) the design and implementation of HPC cluster as a platform (ii) empirical performance measure of various parameters of newly implemented platform. First objective relies on qualitative aspects of research and aim at design of the platform, while Second stage focuses on quantitative and empirical measure of performance under careful design of relevant experiments.

In our design oriented approach for installation of HPC cluster platform, initially we have collected necessary data from various sources and experts. This data include standards, benchmarks, heuristics, network, topology, workload patterns, optimization methods, node configuration profiles and detailed description of protocol selection procedures. This data was analysed and the output of analysis was used for planning, topology design, network installation, and actual design and implementation phase for our HPC cluster platform. After platform was successfully set up, the empirical methodology was used and design of experiments, performance analysis and data analysis tasks were completed.

Following steps are taken during implementation of this research project:

3.1 Data Collection

Following sources of data were used to collect necessary information regarding hardware and software details, design issues and availability of cluster management systems.

- **Documents**: The data was collected from various websites related to university HPC cluster systems from Stanford University, FSU, MIT and Yale.
- **Interviews**: Expert interviews were performed with various experts and professionals in the field of HPC and specific requirements were collected for the design and implementation of the proposed platform.
- **Questionnaires**: different questionnaires were prepared to collect experiential and heuristic knowledge of different stakeholders and administrators of existing HPC facilities in AAU, meteorology etc.

3.2 Data Pre-processing

Once the data was collected and encoded in standard format, pre-processing of data was performed including checks for consistency, validity, missing values, errors and updates.

3.3 Planning and Design of HPC platforms

Based on the input of various experts and available HPC installations in AAU and Meteorology department of Addis Ababa, these design parameters for proposed system were set. This included selection of appropriate network topology, protocols, hardware and software components to enable HPC server and compute nodes each. Once the rough design and layout was finalized, it was the time of actual implementation of proposed platforms. The task of implementation was divided into two teams, each from separate SIG.

3.4 Implementation

Actual implementation of three separate types of HPC clusters was initiated within separate initiatives of data science and intelligent systems research groups. The implementation procedure was divided into various phases as per the lifecycle of an ICT/IT project. On each HPC server and compute node internetworking, protocol configuration, installation and configuration of server and client software was completed in a sequential manner. This step also included the installation and configuration of necessary libraries, dependencies, XML based configuration files and authorization of various users and groups in the HPC cluster. This section also included various scripts for libraries, path profiles, dependency resolutions and job submission.

3.5 Verification and validation

This step included the verification of the recent installation against requirements and objectives of proposed research. Also the correctness of installation and availability of HPC cluster was checked. Testing Installation: A thorough study of behaviour of HPC platform was performed by various test scripts, including programs to test and debug functionality of individual components and testing the availability of system level services of newly installed HPC system. This phase include testing failover, reliability, responsiveness and crash reports in various occurrence of events. Due to large number of possible test cases for HPC environment, we have restricted the test domain for this HPC cluster system.

3.6 Design of experiments

As the platform was implemented on available hardware the experimental methodology is used to an experimental research in which we have designed structured workloads as various experiments and tested the performance of underlying platform under different settings. Since it is an experimental research, therefore we have designed two groups of experiments - control and experiment group. In control group experiment, we have used a standard algorithm like FIFO scheduling algorithm with default parameters for the installed platform, in the experiment group we have an experimental set up with varying platform parameters like number of cores, memory, virtual memory, default wall times, network topology and allocated bandwidth. These experiments were performed on introductory level workloads and the optimum performance profile of underlying platform was deducted. More details of design of experiments and their execution are given in a separate chapter.

3.7 Data Analysis

After the execution of planned stages and experiments on real time data, data analysis was performed to understand the performance of system, interpretation and implications of various parameters changes in newly installed HPC clusters. Details of data analysis are given in the separate chapter.

Chapter 4

Design and Implementation

4.1 DeepStack Cluster in Intelligent Systems SIG

Torque [13] is based on Open PBS a portable batch scheduling system, and it provides highly optimized platform for HPC and cluster computing [2]. Torque provides an abstraction of multiple distributed memory machines as a single resource and these resources can be allocated to various jobs submitted to the torque server. The detailed architecture diagram, hardware specification and implementation process for torque and its components are given in reference to our implementation in the next section. In order to save the readers from confusions regarding general torque specifications and our specific implementation we have given a unique name for our implementation of torque cluster as DeepStack HPC cluster. This cluster in experimental phase includes three 5 node clusters with approximately 15 TB storage and 120 GB RAM. But in the expansion plan we have projections of 30 Core permanently connected compute nodes under DeepStack and 200 Ad-hoc compute nodes available from various laboratories detected on run time but not guaranteed. Also we have planned to extend the capabilities of DeepStack as the source of highest computing power in the country and full-fledged implementation of scientific libraries for various scientific jobs.

Torque enabled us in achieving our objectives in following ways:

- Torque along with MPICH2 and OpenMPI can be used to run large scale, distributed memory jobs over a number of available processors from configured compute nodes.
- Exposer to shared memory programming as well as distributed memory programming.
- Excellent support to various scientific libraries:

4.1.1 Hardware Specification

Three clusters of one torque server and five compute nodes on each sub-cluster are created for the purpose of pilot experiment. Interconnectivity among various computers participating in the cluster is done on the Ethernet cable with 100 Mbps bandwidth while D-Link 8 port is used to create the backbone network with DHCP/static IP. Each compute node and server is maintained on identical Intel core i-7 desktop computers with individual RAM of 8 GB and hard disk capacity of 1 Terabyte each.

However the expected hardware requirements of HPC cluster for Torque server are more advanced, for example torque is not made for desktop environment where operating system level changes are regularly occurring events. It is observed during experiments that if the configuration, libraries,

path variable and other environment variables of underlying operating system are changed, most of the time torque crashed and was difficult to restore and perform recovery task on torque server. Perhaps it is because of multilevel dependency of torque on underlying Linux core and development packages. At the same time the server and compute node are designed with focus on specialized hardware for cluster computing which should be dedicated, high bandwidth and less prone to changes at platform level.

4.1.2 Cluster Specifications

ASTU DeepStack HPC cluster has the following hardware and software specifications:

Master Machine	CPU 2.25 GHz RAM - 8 GB HDD 1 TB OS – Cent OS 7
Compute Nodes	CPU 2.25 Ghz RAM - 8 GB HDD 1 TB OS – Cent OS 7
Software Packages	CentOS Enterprise server 7(full install Platform) Torque 6.1.1, OpenMPI MPICH
Libraries	netcdf/intel/3.6.3, netcdf/intel/4.1.1, stata/11, R/intel/ 2.9.2, MATLAB/2017/b
Programming Languages	C, C++, Java, Fortran

Also we have a plan to implement various scripts for integration of various scientific libraries including various compilers, tools and libraries. The benchmark is taken from NYC HPC cluster computing center.

4.1.3 Cluster Setup and Installation

The following section describe the customizations that are needed to setup and run Spark.

- Multi Node Torque Installation & Configuration : Appendix C

4.1.4 DeepStack Cluster Architecture

In this research, each torque sub cluster is implemented from scratch and involved installation and configuration of various server side dependencies and libraries at the first. The list of all dependencies and libraries used in our implementation are given in appendix and also detailed procedure is described in a separate installation manual. The installation manual shall be supplemented with necessary references and links for various libraries that are needed to be installed on the cluster. Here we explain architecture of DeepStack-2 HPC Cluster:

4.1.4.1 Torque Resource Manager Components

Each torque resource manager designated as server computer contain three components:

1. Torque Server module (Pbs_server) In our architecture each torque server is given same hostname and server-name as per the recommendation of Adaptive Computing Pvt Ltd Company who developed the torque as open source cluster management software. Each torque server is an instance of Pbs_Server. Each torque server comes with default queue called batch. There are certain configuration steps which should be done in a sequence so that we can make server and install with its dependencies.

2. Scheduler module (Pbs_Sched) Torque provide a very basic built in scheduler which is capable of scheduling jobs in FIFO order in job-exclusive and shared mode on compute nodes. If this cluster is to be used for scientific simulations, it must be configured with advanced schedulers such as MOAB and MUAI, which are of-course not free like default scheduler. In fact the default scheduler is enough for the start up the cluster computing. Each server instance is configured with default scheduler.

3. Authentication server module (TRQATHD)

   Torque provides an authentication server whose job is to allow only preconfigured clients to submit the jobs. An instance of Trqauthd is must at the Pbs_server along with pbs_sched demon. While server computer is by default allowed submitting the jobs on server itself, but the purpose of Trqauthd is to register the clients with server so that the jobs can be submitted by external Ip addressed registered with trqauthd.

4.1.4.2 Torque Compute Node Components

Torque compute node contains the Pbs_Mom modules which participate in the execution of jobs in processor exclusive and processor shared mode. If a compute node is enabled to submit the jobs for execution, the trqauthd client side module must be configured with compute node.

1. Multi Operation Machine module (Pbs_Mom)

   Pbs_mom is the module which communicate with pbs_server and participate in actual execution of jobs. It provides various resources for the execution of the jobs like processor, memory and wall-time. Each mom receives its jobs into a queue and executes one job at a time in basic settings. Each job acquires a lock over MOM node and proceeds for its completion before the processor is freed for other jobs at the server in waiting queue. During implantation various kind of communication patterns have been noticed between the MoM node and Pbs_server which will be presented as case studies for helping researchers to troubleshoot the various issues in the job execution, debugging and performance enhancement.

2. Authentication Module (TRQAUTHD)

   Each MoM node which is allowed to submit the jobs at a server must have authentication module installed and configured. For MOM nodes it is an optional package.

4.1.4.3 Torque Client Nodes

Torque Client modules are developed at the time of installation of torque server and they can be copied to the clients along with the necessary scripts for their registry in the system services. Torque clients are designated nodes which are allowed to submit the jobs to pbs_srever. Each client can be registered with the server by its hostname and IP address. A change in IP and hostname of client will crash the job submission system and the whole process of registering and service invocation will be required to be repeated to bring system in working mode.

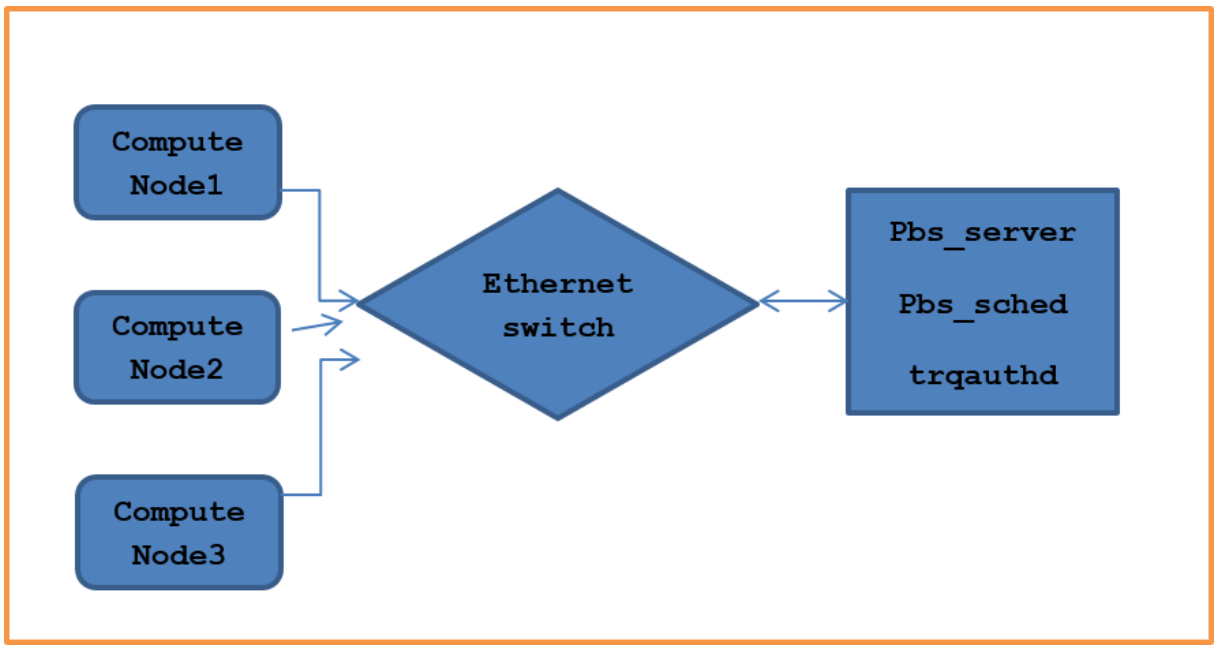


Figure 4.1.1: Torque Cluster Diagram

4.1.5 Cluster Diagram

Each set of sub-clusters contains one server, an instance of scheduler, one authentication demon at server. For the purpose of failover capability, the identical servers are installed in there similar clusters. Currently five Compute nodes are attached to each server and one client is also configured to submit the jobs. The complete architecture of the implemented cluster is given in figure 4.1.1

4.1.6 Technical Challenges with Torque

- Setting up server is difficult and tedious job. Less documentation available.
- Server crashes after small changes in path profile of the CentOS.
- Communication with clients if affected on installation of new libraries.
- Torque is not designed for highly dynamic environment instead HPC server must be configured completely once and run for long time with one configuration.

4.2 HPDA Cluster in Data Science SIG.

The low-cost, distributed and data-intensive cluster, known as HPDA(High Performance Data Analytics) Cluster has been setup in Data Science SIG. laboratory, located at B509 R11.

In simple terms, distributed computing is just a distributed system, where multiple machines are doing certain work at the same time. While doing the work, machines will communicate with each other by passing messages between them. Distributed computing is useful, when there is requirement of fast processing (computation) on Big Data. Apache Hadoop [28] and Apache Spark [23] are well-known examples of Big data processing systems.

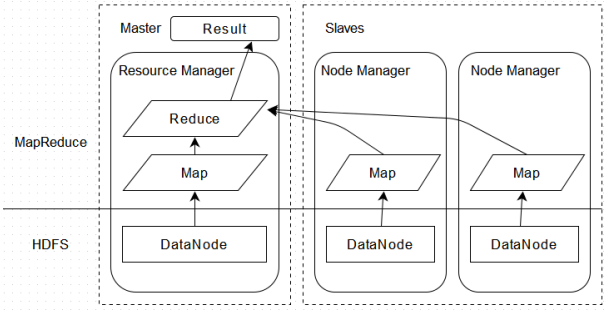


Figure 4.2.1: Architecture of Hadoop

4.2.1 Apache Hadoop

The Apache Hadoop [28] software is an open-source framework built for reliable, scalable distributed computing tasks with huge data sets over a cluster of multiple computers. This framework is written in Java programming language and it is under Apache License.

The important features of Apache Hadoop are:

- allows distributed processing of large data sets across cluster of computers using simple programming model
- has become the de facto standard for storing, processing, and analyzing hundreds of terabytes and petabytes of data
- is cheaper to use in comparison to other traditional proprietary technologies such as Oracle, IBM etc. It can run on low cost commodity hardware.
- can handle all types of data from disparate systems such as server logs, emails, sensor data, pictures, videos etc.

4.2.1.1 Hadoop Architecture

Generally, a Hadoop system is composed of a computer acting as the master node and multiple computers acting as the slave nodes, as shown in figure 4.2.1. Hadoop has two modules in total, including HDFS (Hadoop Distributed File System) and MapReduce Framework. HDFS usually only has one NameNode, which is used to manage the directory tree and the metadata of related files of HDFS. It could also own a Secondary NameNode, which can be employed to backup mirror files and to combine logs and mirror files periodically and send back to NameNode. In general, NameNode and Secondary NameNode are deployed on the master node. In addition, DataNode of HDFS is responsible for store data and sending processed data back to NameNode, and it is usually deployed on the slave node.

After version 0.23, Hadoop starts to adopt new MapReduce framework, called Yarn, consisting of Resource Manager and Node Manager. Resource Manager takes responsibility for managing and dispatching resources in the Hadoop cluster, and receiving data from Node Manager of each slave node. Node Manager is used to send information about usage of resources and task progress to Resource Manager. MapReduce task is executed in a distributed manner. Each DataNode passes original data to a map function. After being processed by map function, original data becomes key-value pairs and is sent back to a reduce function. After completing the step of reduce function, the data will be saved as result or for further processing.

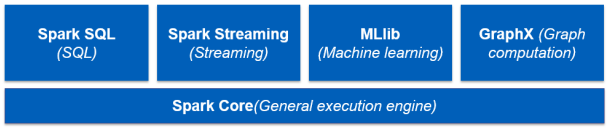


Figure 4.2.2: Spark Architecture Overview

4.2.2 Apache Spark

Apache Spark is installed on top of Hadoop. Spark [23] is a fault-tolerant and distributed data analytics tool capable of implementing large-scale data intensive applications on commodity hardware. Hadoop and other technologies have already popularized acyclic data flow techniques for building data intensive applications on commodity clusters, but these are not suitable for applications that reuse a working dataset for multiple parallel operations. Some of these applications are iterative machine learning algorithms and interactive data analysis tools. Spark addresses these problems, and is also scalable and fault-tolerant. In order to accommodate these goals, Spark introduces a data storage and processing abstraction called Resilient Distributed Datasets (RDDs).

Resilient Distributed Dataset is a collection that has been distributed all over the Spark cluster [29]. RDDs' main purpose is to support higher-level, parallel operations on data in a straightforward manner. There are currently two types of RDDs: parallelized collections, which take an existing Scala collection and run operations on it in parallel, and Hadoop datasets, which run functions on each record of a file in HDFS (or any other storage system supported by Hadoop).

Spark can run tasks up to 100 times faster, when it utilizes the in-memory computations and 10 times faster when it uses disk than traditional map-reduce tasks. Spark performs well in these cases, where Hadoop users have reported deficiency with MapReduce:

- **Iterative jobs**: Gradient-Descent is an excellent example of an algorithm that is repeatedly applied to the same dataset to optimize a parameter. While it is easy to represent each iteration as a MapReduce job, the data from each iteration has to be loaded from the disk, incurring a significant performance penalty.
- **Interactive analytics**: Interfaces like Pig [29] and Hive [30] are commonly used running SQL queries on large datasets using Hadoop. Ideally this dataset is loaded into memory and queried repeatedly, but with Hadoop every query is executed a MapReduce job that incurs significant latency from disk read.

4.2.2.1 Spark Architecture

The following are the components of Apache Spark, shown in figure 4.2.2.

- **Spark Core**: Spark Core is the underlying general execution engine for spark platform that all other functionality is built upon. It provides In-Memory computing and referencing datasets in external storage systems.
- **Spark SQL**: Spark SQL is a component on top of Spark Core that introduces a new data abstraction called SchemaRDD, which provides support for structured and semi-structured data.
- **Spark Streaming**: Spark Streaming leverages Spark Core's fast scheduling capability to perform streaming analytics. It ingests data in mini-batches and performs RDD (Resilient Distributed Datasets) transformations on those mini-batches of data.

- **MLlib (Machine Learning Library)**: MLlib is a distributed machine learning framework above Spark because of the distributed memory-based Spark architecture. Spark MLlib is 9x as fast as the Hadoop disk-based version of Apache Mahout.
- **GraphX**: GraphX is a distributed graph-processing framework on top of Spark. It provides an API for expressing graph computation that can model the user-defined graphs by using Pregel abstraction API. It also provides an optimized runtime for this abstraction.

4.2.3 Cluster Specifications

Hadoop and Spark are designed to run on commodity hardware. That means that you are not tied to expensive, proprietary offerings from a single vendor; rather, you can choose standardized, commonly available hardware from any of a large range of vendors to build your cluster.

"Commodity" does not mean "low-end". Low-end machines often have cheap components, which have higher failure rates than more expensive (but still commodity class) machines. When you are operating tens, hundreds, or thousands of machines, cheap components turn out to be a false economy, as the higher failure rate incurs a greater maintenance cost. On the other hand, large database class machines are not recommended either, since they don't score well on the price/performance curve. And even though you would need fewer of them to build a cluster of comparable performance to one built of mid-range commodity hardware, when one did fail it would have a bigger impact on the cluster, since a larger proportion of the cluster hardware would be unavailable.

Hardware specifications rapidly become obsolete, but for the sake of experimentation, our choice of machine for running a Hadoop DataNode and TaskTracker, Workers have the specifications as shown illustrated in Table 4.2.1. While the hardware specification for cluster will assuredly be different, Hadoop and Spark are designed to use multiple cores and disks, so it will be able to take full advantage of more powerful hardware.

Table 4.2.1: HPDA Cluster Specifications

Parameter	Value
The number of Nodes: Cores in Use: Availed Memory: Availed HDFS:	10 (1 Master, 9 Workers/Slaves) 72 24.9 GB 6.4 TB
Master Machine	CPU 2.25 Ghz RAM - 4 GB HDD 1 TB OS - Ubuntu 16.04 LTS
Slave Machine	CPU 2.25 Ghz RAM - 4 GB HDD 1 TB OS - Ubuntu 16.04 LTS
Ethernet Switch	16 Port 10/100 Mbp
Software Tools	Hadoop 2.7, Spark 2.1.0, Scala 2.12.1, OpenJDK 1.8 Hive 2.0.0, HBase 1.1.4 Flume 1.6.0, Anaconda3 4.3.0 R 3.3.2, RStudio 1.0.136
Language Support	Scala, Java, Python, R,& MapReduce

The bulk of Hadoop is written in Java, and can therefore run on any platform with a JVM, although there are enough parts that harbor Linux assumptions (the control scripts, for example) to make it unwise to run on a non-Linux platform in production.

The bulk of Spark is written in Scala language and it has some code written in Java, Python and R. It can therefore run on any platform with a JVM. Apache Spark provides several APIs for programmers which include Java, Scala, R and Python.

4.2.4 Cluster Setup and Installation

There are various ways to install and configure Hadoop. This section describes how to do it from scratch using the Apache Hadoop distribution, and will give the background to cover the things you need to think about when setting up Hadoop. Alternatively, if you would like to use RPMs or Debian packages for managing your Hadoop installation, then you can start with Cloudera's Distribution.

To ease the burden of installing and maintaining the same software on each node, it is normal to use an automated installation method like Red Hat Linux's Kickstart or Debian's Fully Automatic Installation. These tools allow you to automate the operating system installation by recording the answers to questions that are asked during the installation process (such as the disk partition layout), as well as which packages to install. Crucially, they also provide hooks to run scripts at the end of the process, which are invaluable for doing final system tweaks and customization that is not covered by the standard installer.

The following section describe the customizations that are needed to install Hadoop and Spark on Ubuntu 16.04 LTS.

- Multi Node Hadoop Installation & Configuration : Appendix A
- Multi Node Spark Installation & Configuration : Appendix B

4.2.5 Hadoop Cluster Diagram

The Apache Hadoop cluster diagram of 10 nodes is shown in figure 4.2.3.

4.2.6 Spark Cluster Diagram

The Apache Spark cluster diagram of 10 nodes is shown in figure 4.2.4.

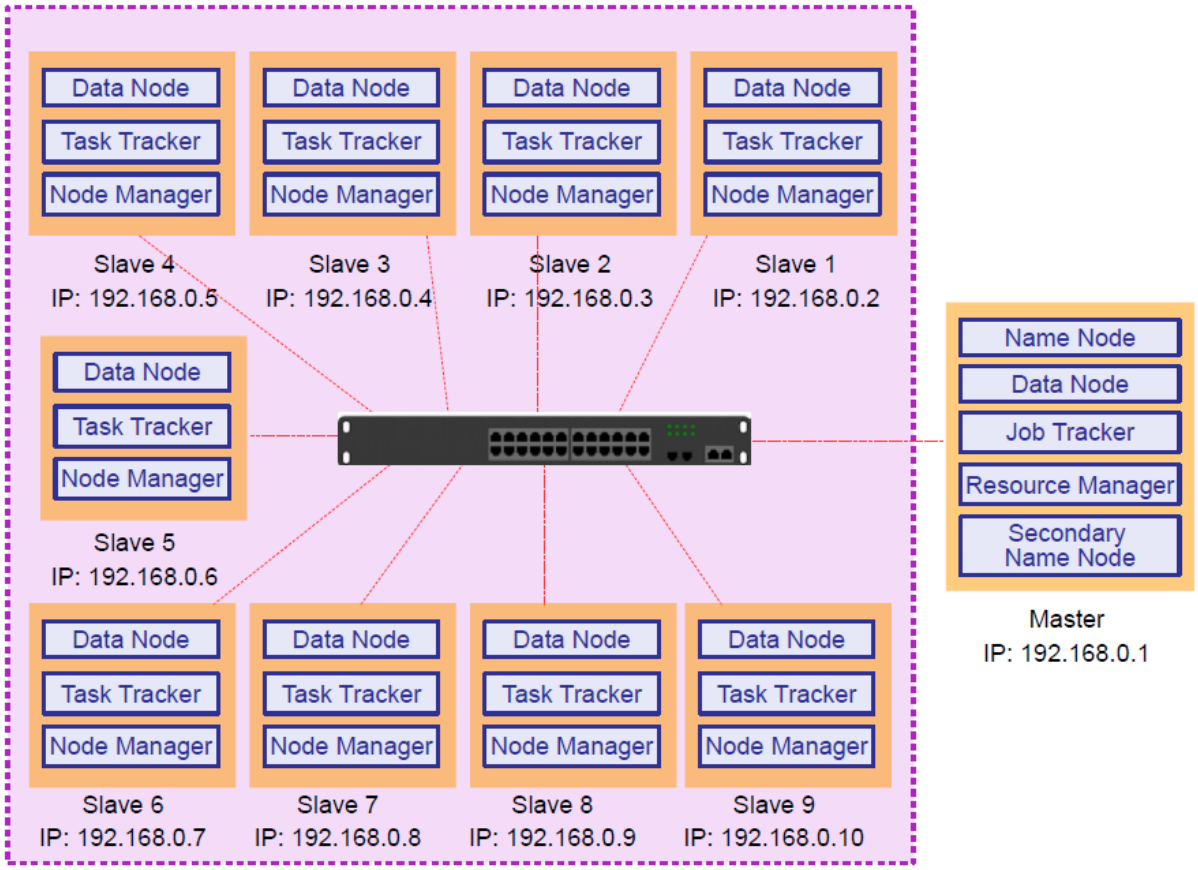


Figure 4.2.3: 10 Nodes Hadoop Cluster

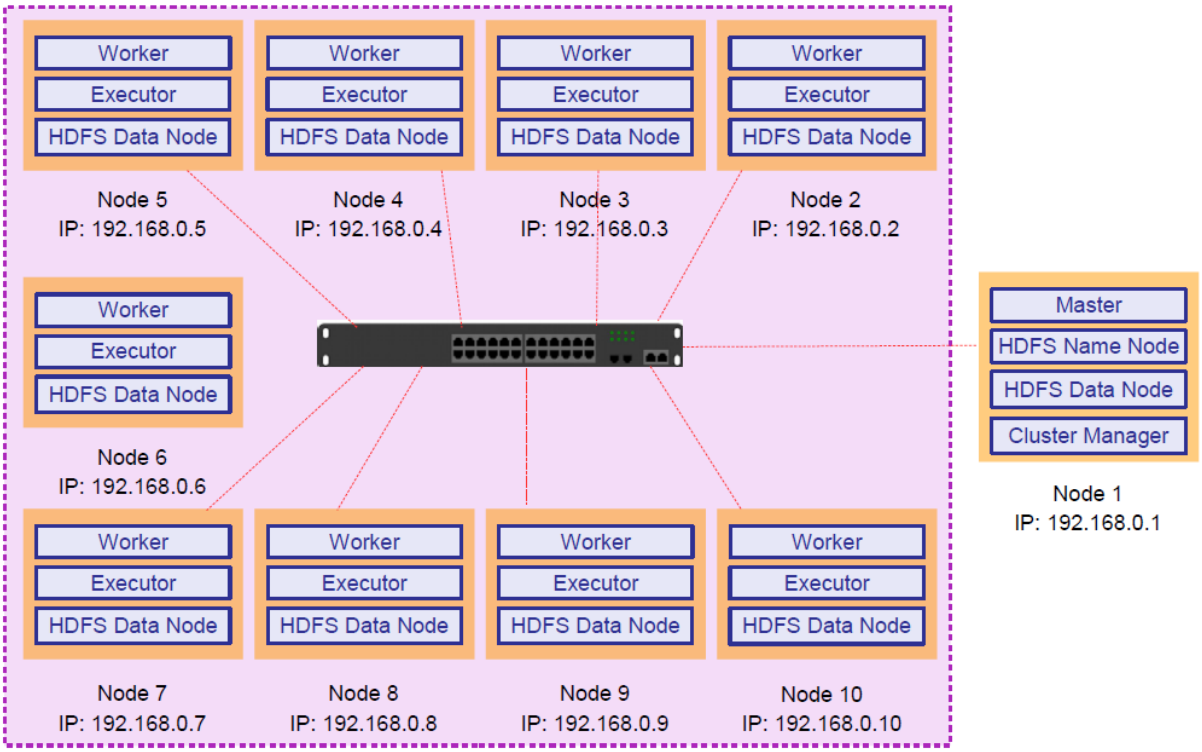


Figure 4.2.4: 10 Nodes Spark Cluster

Chapter 5

Experimentation

5.1 Experiments on DeepStack Cluster

The benchmark experiments on this HPC cluster can be categorized into three types:

1. **Test and Debugging level programs**:
   This level of programs is designed to test the availability of cluster server and compute nodes in order to debug and test the cluster system itself. We have used following scripts for DeepStack to be used as test programs:

```
Echo sleep 10 | qsub #press enter 10 times
Echo 'hello world on DeepStack-2 cluster' | qsub
  qsub random.pbs # script to get renadom numbers
  qsub sort.pbs # script for sorting
```

2. **Programs of Intermediate complexity**:
   These programs include graduate level research and simulation projects including markov chains, monte-carlo simulation, distributed pageRank algorithm, HITS algorithm, Distributed Gradient descent etc.

3. **Benchmark Programs**:
   Benchmark programs include the problems based on highly optimized libraries for core scientific research with competitive baseline figures in the form of linear algebra, genetics, and simulation studies etc. using LPACK, LINPACK, BLAS etc. Implementation of these libraries is proposed to be in second round of this project.

5.1.1 Theoretical Basis

Speedup :There are three types of speedup in the latency and throughput which can be achieved by using our Torque cluster architecture:

1. Linear Speedup(fully parallel case experimented)

2. Sub-linear Speedup(sequential parallel mixed case)

3. Superliner Speedup(cached case - future work)

(1) and (2) can be explained by Ahmdal's of diminishing returns; while (3) is explained by principle of locality of data i.e. distributed caching architecture used.

Latency and Throughput: Latency (L) : Latency of an architecture is the time taken for per unit workload and it is given by following formula:

$$L = W/T \tag{5.1.1}$$

Where, T = total time taken by workload on this architecture, and
W = total workload in no. of instructions/jobs.

Throughput: Defined on the execution density ρ, a the number of processors. Also Q is inversely proportional to the latency of the architecture:

$$Q = \rho.AW/T = \rho.A/L \tag{5.1.2}$$

where A= number of processors used;
W = total workload executed ; and
T = total time taken.

Speedup in latency : by making architecture parallel we are intended to speed up in latency of the system. Following formula is used to compute the speed up in latency between architecture1 and architecture2:

$$S_{latency} = L_1/L_2 = T_1W_2/T_2W_1 \tag{5.1.3}$$

where S = speedup in latency from architecture1 to architecture2
$L_1 = Latency on architecture1$
$L_2 = Latency on architecture2$

Speedup in Throughput : defined by the following formula:

$$S_{throughput} = \frac{Q_2}{Q_1} = \frac{\rho_2 A_2 T_1 W_2}{\rho_1 A_1 T_2 W_1} = \frac{\rho_2 A_2}{\rho_1 A_1} S_{latency} \tag{5.1.4}$$

where $S_{throughput}$ = speedup in throughput of the architecture2 with respect to architecture1
Q_1 = throughput of architecture1
Q_2 = throughput of architecture2

Amdahl's law: If there is some part of the program which can not be parallelized because of some kind of dependencies, then actual speedup cannot be scaled in proportion to the number of newly added processors linearly; also the speedup is inversely proportional to the amount of sequentiality in the program with following formula:

$$S_{latency}(S) = \frac{1}{(1-p) + (p/s)} \tag{5.1.5}$$

where $S_{latency}$ is the theoretical speedup of the execution of the whole task;
s is the speedup of the part of the task that benefits from improved system resources;
p is the proportion of execution time that the part benefiting from improved resources originally occupied.

Linear Speedup Explained: As per Amdahl's law on a fixed parallizability level in a program, if we increase number of processors, Torque shows approximately linear increase in speedup as we add new processors.

Sub-linear Speedup Explained: If program has $s\%(< 100\%)$ sequential part, it will show sublinear speedup on Torque. The exact amount of penalty can be computed by Amdahl's equation and implementation of program.

5.1.2 Job Scheduling

Scheduling performance is critical for all high level applications. Therefore, we have performed experiments on FCFS workload (other scheduling algorithms in torque are can be simulated in same way including SJF and fair share scheduling). Here, we report the behavior of FCFS scheduling algorithm on a single processor and the gain in speedup with increasing number of processors as per Amdahl's law.

Table 5.1.1: Experiment Setup

Jobs:	Arrival Time, CPU burst time is given
Granularity level:	Process
Type:	fully parallel
Expected speedup:	Linear
Scheduling algorithm:	FCFS
Compute Nodes:	Exclusive
Process level Parameters:	Latency individual process, memory used, and virtual memory used
Overall performance parameters:	Speedup in latency
Number of queues:	One

Performance Metric:
Architecture: Single processor(benchmark) vs. Multiple compute nodes(empirical estimated incrementally up to 5 nodes)

Parameters used in evaluation:

- Avg. Waiting time i.e. $WT = \sum TAT - BT$ over all jobs
- Avg. Turn Around time i.e. $TAT = \sum CT - AT$
- Speedup in latency

The sample workload taken for experimentation is:

Table 5.1.2: Sample Workload on DeepStack Cluster

S.No.	Job-ID	Arrival Time	CPU burst time
1	P_1	4	200
2	P_2	10	500
3	P_3	6	400
4	P_4	2	300
5	P_5	5	200
6	P_6	3	100
7	P_7	12	250
8	P_8	14	320
9	P_9	20	250
10	P_{10}	22	150

The description of the results produced after executing the above workload can be found in section 6.1 of chapter 6. For further details of experiments, readers are directed to go through the section, Appendex C.

5.2 Experiments on HPDA Cluster

The Apache Hadoop Cluster is developed to scale up from single servers to thousands of machines, each offering local computation and storage. The Apache Spark provides In-Memory computing. As part of this research study, the experimentation was conducted on the cluster, running of WordCount program which counts the number of words specified in a given text file using Eclipse IDE.

The following are benchmark applications on HPDA cluster:

- Iterative Jobs
- Interactive Analytics
- Distributed Machine Learning
- Streaming Analytics
- Distributed Graphs Processing

The process of conducting experimentation(WordCount) is as given below:

1. Start Hadoop

```
hduser@master:~$ cd $HADOOP_HOME
hduser@master:/usr/local/hadoop-2.7.3$ cd sbin
hduser@master:/usr/local/hadoop-2.7.3/sbin$ ./start-all.sh
```

2. Verify running of Hadoop components on Master machine

```
hduser@master:/usr/local/hadoop-2.7.3/sbin$ jps
3587 Jps
2772 ResourceManager
2356 DataNode
2180 NameNode
2552 SecondaryNameNode
2922 NodeManager
```

3. Verify running of Spark components on Master machine

```
hduser@master:/usr/local/spark-2.1.0-bin-hadoop2.7/sbin$ jps
4243 Master
2772 ResourceManager
2356 DataNode
2180 NameNode
4342 Jps
2552 SecondaryNameNode
2922 NodeManager
```

4. Copy the file from local file system to Hadoop Distributed File System(HDFS)

```
hduser@master:/usr/local/hadoop-2.7.3$ bin/hdfs dfs -put /home/hduser/Downloads/
input.txt /input
```

5. Develop a WordCount program in scala language using Spark IDE [26].

```
import org.apache.spark.SparkContext
 import org.apache.spark.SparkConf

        object SparkWordCount {
             def main(args: Array[String]) {

          //Create conf object
          val conf = new SparkConf()
          .setAppName("WordCount")

          //create spark context object
            val sc = new SparkContext(conf)

          //Check whether sufficient params are supplied
          if (args.length < 2) {
              println("Usage: ScalaWordCount <input> <output>")
              System.exit(1)
          }
        //Read file and create RDD
        val rawData = sc.textFile(args(0))

        //convert the lines into words using flatMap operation
          val words = rawData.flatMap(line => line.split(" "))

        //count the individual words using map and reduceByKey operation
```

```
        val wordCount = words.map(word => (word, 1)).reduceByKey(_ + _)

      //Save the result
        wordCount.saveAsTextFile(args(1))

      //stop the spark context
        sc.stop
      }
}
```

6. Create a jar file of the project
7. Run the jar file, specifying the file name and output folder.

```
hduser@master:/usr/local/spark-2.1.0-bin-hadoop2.7$ bin/spark-submit --class Spa
rkWordCount --master spark://master:7077 $HOME/WordCount.jar hdfs://master:9000/
input/input.txt hdfs://master:9000/output3
```

8. Successful Running of command generates output in the given output folder.

For further details of experiments, readers are directed to go through the section, Appendex B. The results of this experiment can be found in section 6.2 of of Chapter 6.

Chapter 6

Results and Discussion

6.1 Results on DeepStack Cluster

Following section include some screenshots of our DeepStack HPC Cluster.

1. **Active Pbs_server, TRQAUTHD and scheduler**:
   After successful installation we can query status of each service on torque server as well as on compute node. Figure 6.1.1 that on ASTU DeepStack HPC cluster pbs_server, pbs_sched and trqauthd are active and running as well as on all compute node pbs_mom is active and running.

2. **Status of ComputeNodes**:
   By pbsnodes a, we can query about the state of nodes which can be any one of online, free, shared, job-exclusive etc. as shown in figure 6.1.2.

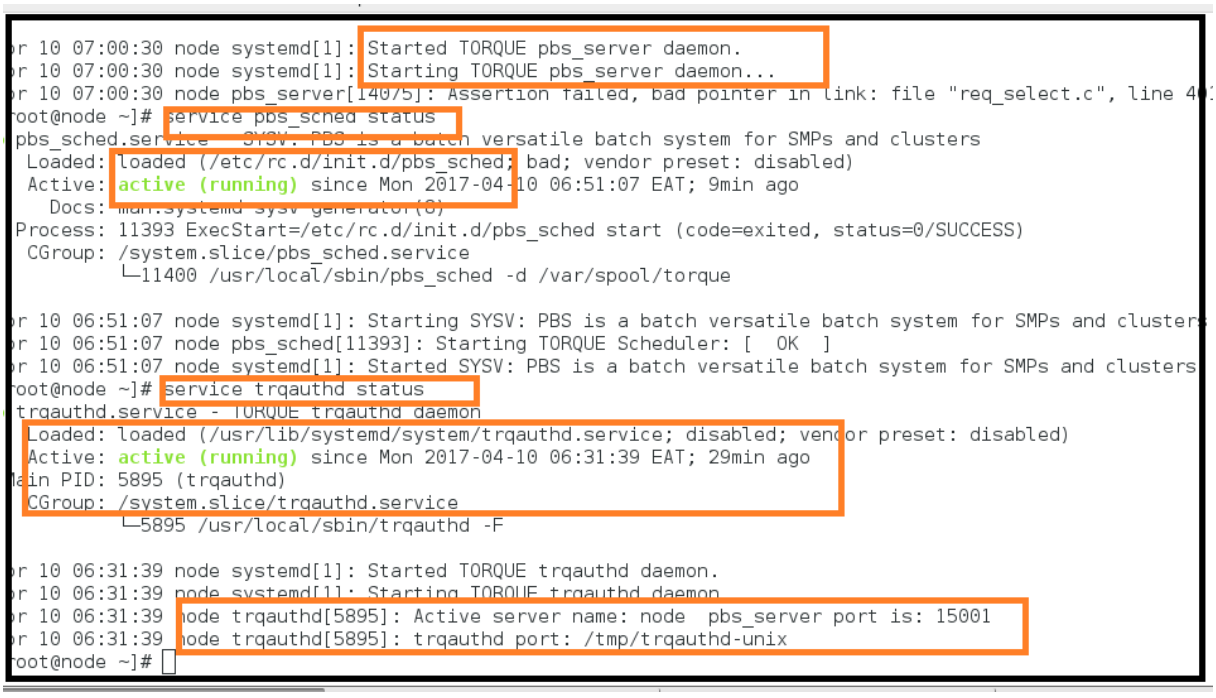


Figure 6.1.1: Active Pbs_server, TRQAUTHD and scheduler

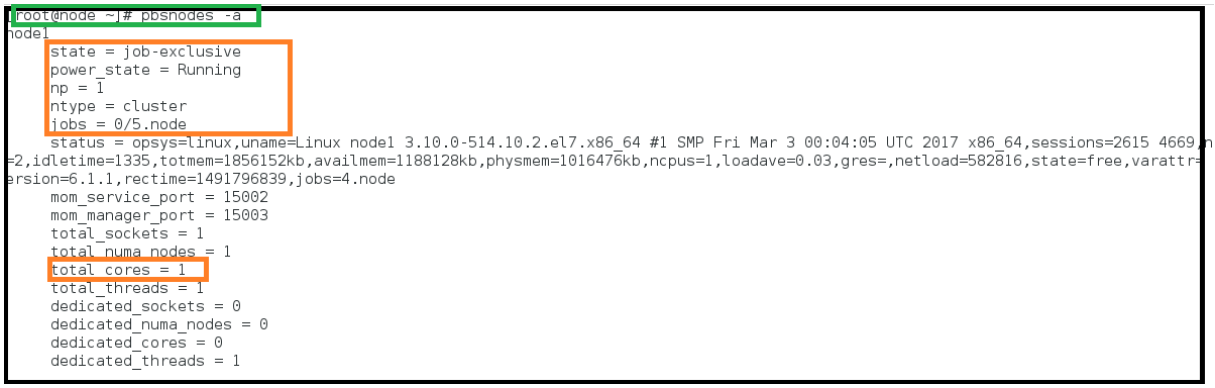


Figure 6.1.2: Status of Compute Nodes

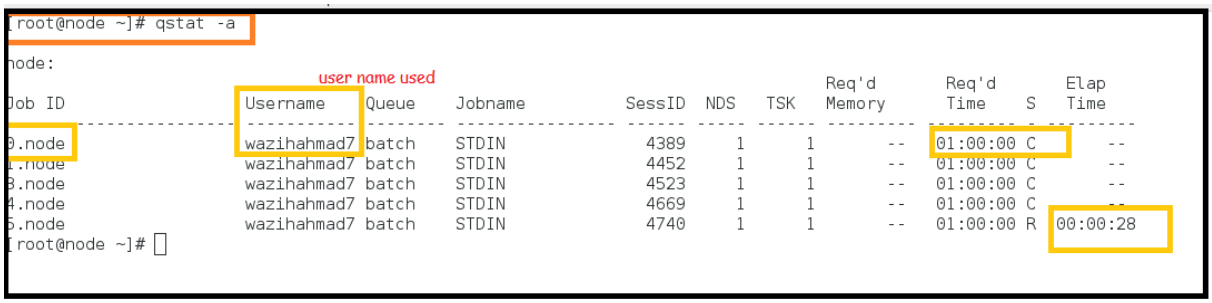


Figure 6.1.3: Job Statistics

3. **Job Statistics**:
   We can report the statistics of each job by using qstat a, it include job Progress, Queued, Execution status, Completed as shown in figure 6.1.3:

4. **Job execution details**:
   Job execution details, as shown in figure 6.1.4 contain the run time allocation of resources such as cores, memory virtual memory etc. the command to know these details is qstat -f

5. **Job Completion**:
   The completed jobs are statistically analysed for time taken, cpu resources used latencies involved, virtual memory used. We can trace states of a job, as shown in figure 6.1.5 till its completion using tracejobs jobid command while for debugging of struck jobs we may run pbs_mom or pbs_server under GDB debugger and in run mode we can trace the progress of each job.

For the purpose of performance analysis we have designed the experiments on the basis of distributed scheduling algorithms like FCFS, SJF etc. and we have found that a job with 10 second CPU burst Time was completed in 28 seconds in FCFS scheme. Even This turnaround time is reported when there was enough number of cores for each process to immediately lock the processor in exclusive mode. But if additional jobs were fired, they were waiting in the queue at pbs_server and standard rules of analysis for FCFS were applicable incurring scheduling level delays.

Due to less bandwidth of Ethernet cables, involved delays and TCP based communication, the current set up is very poorly performing in terms of communication. In future we have a goal to implement the GbpS Ethernet and design experiments for actual performance assessment in terms of network latency and the delay incurred. We will assess performance on various parameters such as distance, number of compute nodes, allowed virtual memory and wall times.

```
.node                   wazihahmad7 batch    STDIN                4523   1      1
.node                   wazihahmad7 batch    STDIN                4669   1      1
.node                   wazihahmad7 batch    STDIN                4740   1      1
root@node ~]# qstat -f
ob Id: 0.node
    Job_Name = STDIN
    Job_Owner = wazihahmad786@node
    resources_used.cput = 00:00:00
    resources_used.energy_used = 0
    resources_used.mem = 2228kb
    resources_used.vmem = 338392kb
    resources_used.walltime = 00:01:40
    job_state = C
    queue = batch
    server = node
    Checkpoint = u
    ctime = Mon Apr 10 06:42:05 2017
    Error_Path = node:/root/STDIN.e0
    exec_host = node1/0
    Hold_Types = n
    Join_Path = n
    Keep_Files = n
    Mail_Points = a
    mtime = Mon Apr 10 06:59:52 2017
    Output_Path = node:/root/STDIN.o0
    Priority = 0
    qtime = Mon Apr 10 06:42:05 2017
    Rerunable = True
```

Figure 6.1.4: Job Execution

```
ob: 0.node

04/10/2017 06:42:05.036 S    enqueuing into batch, state 1 hop 1
04/10/2017 06:42:05  A    queue=batch
04/10/2017 06:44:56.748 S    enqueuing into batch, state 1 hop 1
04/10/2017 06:44:56.748 S    Requeueing job, substate: 10 Requeued in queue: batch
04/10/2017 06:51:54.667 S    enqueuing into batch, state 1 hop 1
04/10/2017 06:51:54.667 S    Requeueing job, substate: 10 Requeued in queue: batch
04/10/2017 06:51:54.720 S    Job Modified at request of root@node
04/10/2017 06:51:54.720 L    Not enough of the right type of nodes available
04/10/2017 06:53:43.631 S    Job Modified at request of root@node
04/10/2017 06:53:43.640 L    Job Run
04/10/2017 06:53:43.632 S    Job Run at request of root@node
04/10/2017 06:53:43.640 S    Not sending email: User does not want mail of this type.
04/10/2017 06:53:43  A    user=wazihahmad786 group=wazihahmad786 jobname=STDIN queue=batch ctime=1491795725 qtime=1491795725 etime=1491795725
                          start=1491796423 owner=wazihahmad786@node exec_host=node1/0 Resource_List.nodect=1 Resource_List.neednodes=1
                          Resource_List.walltime=01:00:00 Resource_List.nodes=1
04/10/2017 06:55:31.916 S    Not sending email: User does not want mail of this type.
04/10/2017 06:55:31.917 S    Exit_status=0 resources_used.cput=0 resources_used.energy_used=0 resources_used.mem=2228kb resources_used.vmem=338392kb
                          resources_used.walltime=00:01:40
04/10/2017 06:55:31  A    user=wazihahmad786 group=wazihahmad786 jobname=STDIN queue=batch ctime=1491795725 qtime=1491795725 etime=1491795725
                          start=1491796423 owner=wazihahmad786@node exec_host=node1/0 Resource_List.nodect=1 Resource_List.neednodes=1
                          Resource_List.walltime=01:00:00 Resource_List.nodes=1 session=4389 total_execution_slots=1 unique_node_count=1
                          end=1491796531 Exit_status=0 resources_used.cput=0 resources_used.energy_used=0 resources_used.mem=2228kb
                          resources_used.vmem=338392kb resources_used.walltime=00:01:40
04/10/2017 07:00:30.512 S    enqueuing into batch, state 6 hop 1
04/10/2017 07:00:30.512 S    Requeueing job, substate: 59 Requeued in queue: batch
04/10/2017 07:00:30.535 S    on_job_exit valid pjob: 0.node (substate=59)
root@node ~]#
```

Figure 6.1.5: Job Completion

Also we have plan to assess the cluster performance on scheduling load with basic scheduler in shared processor as compared to now which is done in exclusive processor lock mode.

With the full scale implementation of DeepStack we hope that the research team will be able to get the standard latency figures for Gbps backbone. The completion of this research shall open a new platform for MPI based distributed programmers using C,C++, JAVA and Fortran bindings to design and run their programs on Gbps speed and taking the benefit of multiple processors, memory and storage. Application level performance analysis is not included in the scope of current research, but it mainly depends on the programming style and best use of available cluster resources under (future) published performance table for DeepStack.

6.1.1 Job Execution Order on Single Node

Baseline: Single Compute Node, FCFS scheduling performance is used as a benchmark for this experiment, which has execution order as given below.

P_4	P_6	P_1	P_5	P_3	P_2	P_7	P_8	P_9	P_{10}

Then, its performance is:

Process	AT	BT	Start	CT	TAT	WT
P4	2	300	2	302	300	0
P6	3	100	302	402	399	299
P1	4	200	402	602	598	398
P5	5	200	602	802	797	597
P3	6	400	802	1202	1196	796
P2	10	500	1202	1702	1692	1192
P7	12	250	1702	1952	1940	1690
P8	14	320	1952	2272	2258	1938
P9	20	250	2272	2522	2502	2252
P10	22	150	2522	2672	2650	2500
Avg.(BASELINE)				**1443**	**1433.2**	**1166.2**

6.1.2 Job Execution on DeepStack Cluster

6.1.2.1 Two Nodes Cluster

The execution order: Node 1, Queue 1

P_4	P_5	P_2	P_9	P_{10}

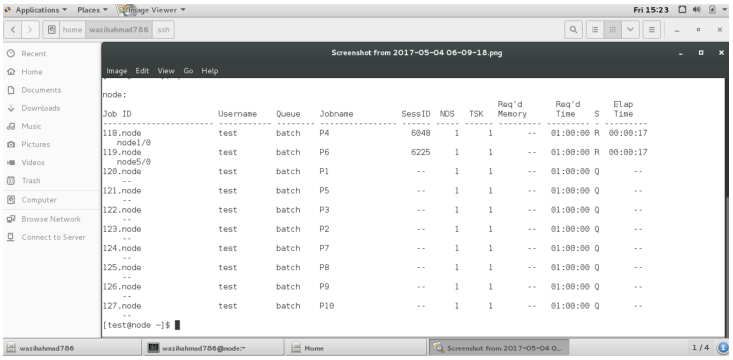
The execution order: Node 5, Queue 1

P_6	P_1	P_3	P_7	P_8

Then, the performance on these 2 nodes:

Job_Id	Job_Name	walltime	session_id	start_time	comp_time	total_runtime	host
128.node	P4	00:05:00	6841	1493895728	1493896036	308.279957	node1
129.node	P6	00:01:40	7034	1493895728	1493895836	108.25653	node5
130.node	P1	00:03:20	7080	1493895836	1493896044	208.285914	node5
131.node	P5	00:03:20	6929	1493896036	1493896244	208.303239	node1
132.node	P3	00:06:40	7151	1493896044	1493896453	408.229317	node5
133.node	P2	00:08:20	6999	1493896244	1493896753	508.269305	node1
134.node	P7	00:04:10	7264	1493896453	1493896711	258.282265	node5
135.node	P8	00:05:20	7340	1493896711	1493897039	328.246584	node5
136.node	P9	00:04:10	7128	1493896753	1493897011	258.251533	node1
137.node	P10	00:02:30	7212	1493897011	1493897169	158.267727	node1

Job scheduling on 2 nodes:



Node 1 Queue performance:

Process	AT	BT	ST	CT	TAT	WT
P4	2	300	2	310.28	308.28	8.28
P5	5	200	310.28	518.58	513.58	313.58
P2	10	500	518.58	1026.84	1016.84	526.84
P9	20	250	1026.84	1285.12	1238.12	988.12
P10	22	150	1285.12	1443.38	1421.38	1271.38
Avg.				916.84	899.64	621.64

Node 5 Queue performance:

Process	AT	BT	ST	CT	TAT	WT
P6	3	100	3	111.28	108.28	8.28
P1	4	200	111.28	319.56	315.56	115.56
P3	6	400	319.56	727.78	721.78	321.78
P7	12	250	727.78	986.06	974.06	724.06
P8	14	320	986.06	1314.3	1300.3	980.3
Avg.				691.796	683.996	429.996

Cluster level Average Performance on DeepStack:2-Node cluster

DeepStack		Speedup	Expected speedup
Avg.CT	**804.315**	**1443/804.315=1.79**	**2**
Avg.TAT	**791.818**	**1.8**	**2**
Avg.WT	**525.81**	**2.12**	**NA**
Avg latency	**8.28**	**NA**	**NA**

As per Amdahl's law in this case the speedup in CT and TAT should be twice as of the single processor benchmark. But due to Ethernet latencies it is decreased by 10%.

6.1.2.2 Three Nodes Cluster

The execution order: Node 1, Queue 1

P_4	P_2	P_1

The execution order: Node 5, Queue 5

P_6	P_5	P_7	P_8

The execution order: Node 4, Queue 4

P_1	P_3	P_9

Architecture 3: Three Node Torque Cluster performance

Job_Id	Job_Name	mem	walltime	session_id	start_time	comp_time	total_runtime	host
98.node	P4	692kb	00:05:00	5061	1493889158	1493889466	308.263185	node1
99.node	P6	680kb	00:01:40	5169	1493889158	1493889266	108.245294	node5
100.node	P1	584kb	00:03:20	5082	1493889158	1493889366	208.272266	node4
101.node	P5	572kb	00:03:20	5223	1493889266	1493889475	208.284777	node5
102.node	P3	572kb	00:06:40	5155	1493889366	1493889775	408.277225	node4
103.node	P2	688kb	00:08:20	5146	1493889466	1493889975	508.293653	node1
104.node	P7	684kb	00:04:10	5285	1493889475	1493889733	258.231792	node5
105.node	P8	692kb	00:05:20	5374	1493889733	1493890061	328.265583	node5
106.node	P9	572kb	00:04:10	5260	1493889775	1493890033	258.281908	node4
107.node	P10	700kb	00:02:30	5284	1493889975	1493890133	158.282741	node1

```
test@node ~]$ qstat -a -n

ode:
ob ID                  Username    Queue    Jobname         SessID  NDS   TSK   Req'd Memory   Req'd Time    S  Elap Time
8.node                 test        batch    P4              5061    1     1     --             01:00:00 R  00:00:37
   node1/0
9.node                 test        batch    P6              5169    1     1     --             01:00:00 R  00:00:37
   node5/0
00.node                test        batch    P1              5082    1     1     --             01:00:00 R  00:00:37
   node4/0
01.node                test        batch    P5              --      1     1     --             01:00:00 Q  --
02.node                test        batch    P3              --      1     1     --             01:00:00 Q  --
   --
03.node                test        batch    P2              --      1     1     --             01:00:00 Q  --
   --
04.node                test        batch    P7              --      1     1     --             01:00:00 Q  --
   --
05.node                test        batch    P8              --      1     1     --             01:00:00 Q  --
   --
06.node                test        batch    P9              --      1     1     --             01:00:00 Q  --
   --
07.node                test        batch    P10             --      1     1     --             01:00:00 Q  --
```


```
test@node ~]$ qstat -a -n

ode:
ob ID                  Username    Queue    Jobname         SessID  NDS   TSK   Req'd Memory   Req'd Time    S  Elap Time
8.node                 test        batch    P4              5061    1     1     --             01:00:00 R  00:00:37
   node1/0
9.node                 test        batch    P6              5169    1     1     --             01:00:00 R  00:00:37
   node5/0
00.node                test        batch    P1              5082    1     1     --             01:00:00 R  00:00:37
   node4/0
01.node                test        batch    P5              --      1     1     --             01:00:00 Q  --
02.node                test        batch    P3              --      1     1     --             01:00:00 Q  --
   --
03.node                test        batch    P2              --      1     1     --             01:00:00 Q  --
   --
04.node                test        batch    P7              --      1     1     --             01:00:00 Q  --
   --
05.node                test        batch    P8              --      1     1     --             01:00:00 Q  --
   --
06.node                test        batch    P9              --      1     1     --             01:00:00 Q  --
   --
07.node                test        batch    P10             --      1     1     --             01:00:00 Q  --
```

```
[test@node ~]$ qstat -a -n

node:

                                                                  Req'd    Req'd       Elap
Job ID          Username Queue    Jobname    SessID  NDS  TSK   Memory   Time    S   Time
--------------- -------- -------- ---------- ------ ----- ----- ------ --------- - --------
98.node         test     batch    P4           5061     1     1    --   01:00:00 C       --
   node1/0
99.node         test     batch    P6           5169     1     1    --   01:00:00 C       --
   node5/0
100.node        test     batch    P1           5082     1     1    --   01:00:00 C       --
   node4/0
101.node        test     batch    P5           5223     1     1    --   01:00:00 C       --
   node5/0
102.node        test     batch    P3           5155     1     1    --   01:00:00 C       --
   node4/0
103.node        test     batch    P2           5146     1     1    --   01:00:00 C       --
   node1/0
104.node        test     batch    P7           5285     1     1    --   01:00:00 C       --
   node5/0
105.node        test     batch    P8           5374     1     1    --   01:00:00 C       --
   node5/0
106.node        test     batch    P9           5260     1     1    --   01:00:00 C       --
   node4/0
107.node        test     batch    P10          5284     1     1    --   01:00:00 R  00:01:32
   node1/0
```

Node 1 Queue performance on queue 1:

Process	AT	BT	Start	CT	N/w Latency	TAT	WT
P4	2	300	2	310.26	8.24	308.25	10.25
P2	10	500	310.26	818.55	8.24	808.55	308.55
P10	22	150	818.55	976.8	8.24	954.8	804.8
			Avg.	**701.87**	**8.24**	**690.53**	**374.53**

Node 5 Queue performance on Queue 5:

Process	AT	BT	Start	CT	N/w Latency	TAT	WT
P6	3	100	3	111.45	8.245	108.45	8.245
P5	5	200	111.45	319.73	8.27	314.73	114.73
P7	12	250	319.73	577.96	8.23	565.96	315.96
P8	14	320	577.96	906.22	8.23	892.22	572.22
Avg				**478.84**	**8.24375**	**470.34**	**252.7888**

Node 4 Queue performance on Queue 4:

Process	AT	BT	Start	CT	N/w Latency	TAT	WT
P1	4	200	4	212.27	8.27	208.27	8.27
P3	6	400	212.27	620.54	8.27	614.54	214.5
P9	20	250	620.57	878.81	8.27	858.81	608.81
Avg				**570.54**	**8.27**	**560.54**	**277.1933**

Cluster level Average Performance on DeepStack:3-Node cluster

DeepStack		Speedup
Avg.CT	**583.75**	**1443/583.75=2.47**
Avg.TAT	**573.8**	**2.49**
Avg.WT	**301.5**	**3.87**
Avg latency	**8.25**	**NA**

As per Amdahl's law in this case the speedup in CT and TAT should be twice as of the single processor benchmark. But due to Ethernet latencies it is decreased by 15%.

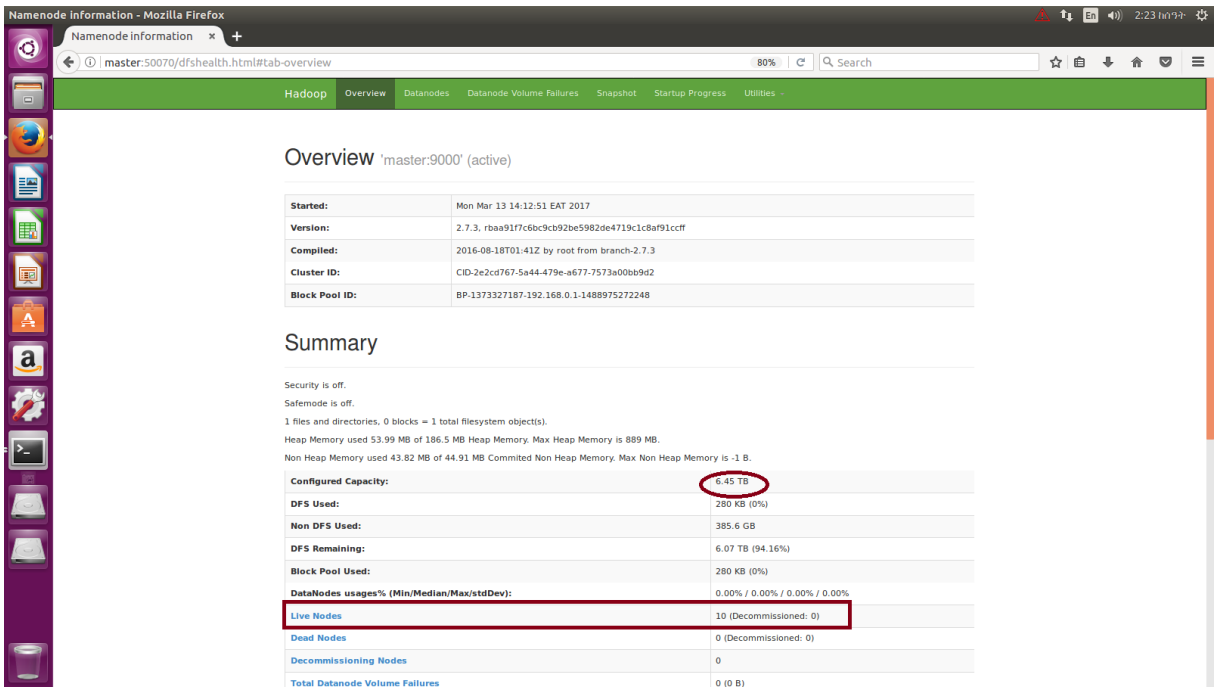


Figure 6.2.1: Apache Hadoop Running

6.1.3 Performance Comparison of DeepStack Cluster with Single Node cluster

Metric	Single Processor Timings	Deepstack-2Node Timings	Speedup Deepstack-2Node	Deepstack-3Node Timings	Speedup Deepstack-3Node
Avg.CT	1443	804.315	✓ 1.79	583.75	✓ 2.47
Avg.TAT	1433.2	791.818	✓ 1.8	573.8	✓ 2.49
Avg.WT	1166.2	525.81	✓ 2.12	301.5	✓ 3.87
Avg latency	0	8.28	✓ NA	8.25	✓ NA

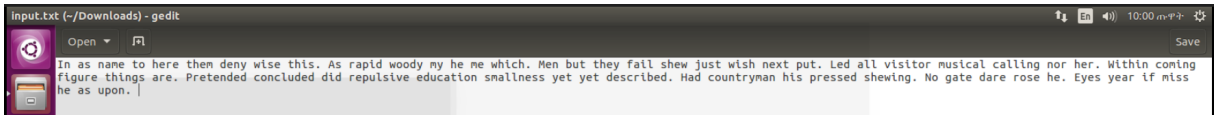
Similarly we have increased the number of compute nodes in the experiment and computed the performance parameters in each set up. Since workload and other conditions are fixed we can use speedup as a measure of performance of our HPC cluster.

6.2 Results on HPDA Cluster

We have developed a Hadoop cluster of 10 nodes with available commodity computers in the lab. The figure 6.2.1 is showing that 10 live nodes are running and able to avail distributed file system 6.45 terabytes(TB). The figure 6.2.2 is showing the list of data nodes that are currently running.

The figure 6.2.3 illustrating that there is 1 master and 9 live workers are running. The cluster able to compile the memory of 24.9GB, which is enough to run medium size data intensive applications or jobs on this cluster.

The procedure stated in section 5.2 of chapter 5 was followed to run WordCount application. The input file had the content as shown below.



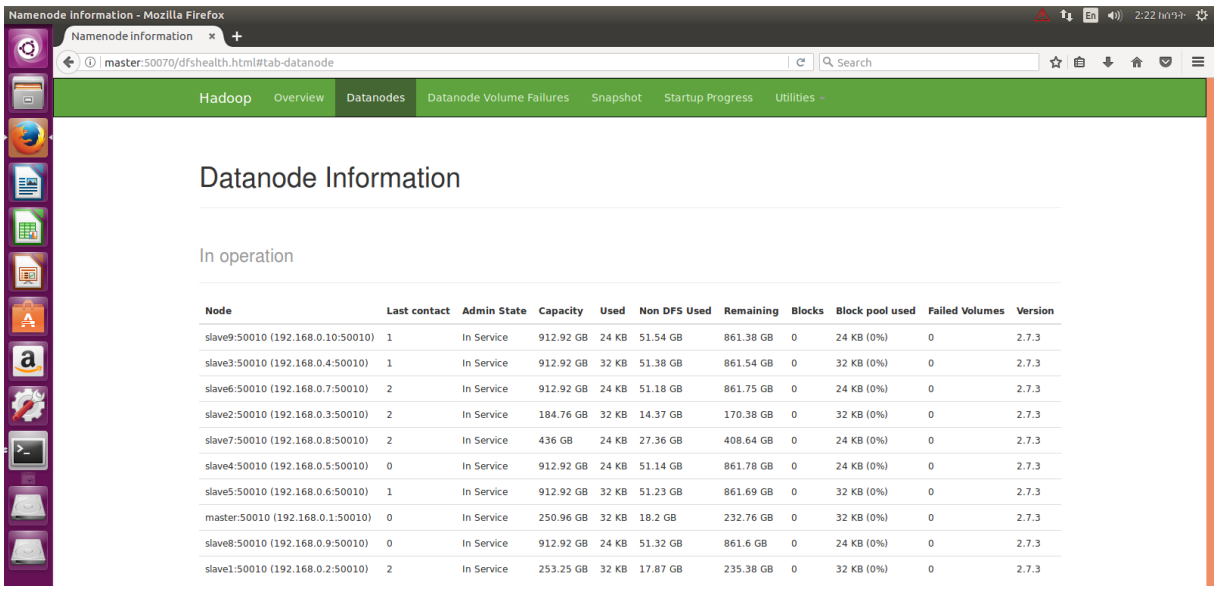


Figure 6.2.2: Hadoop Nodes Running

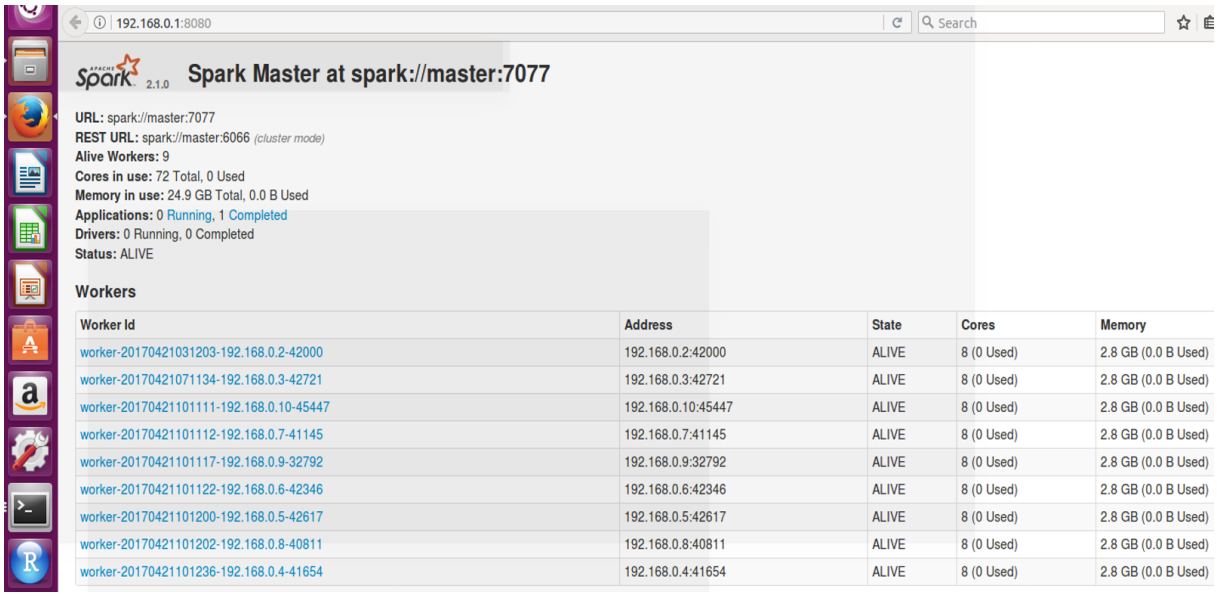


Figure 6.2.3: Apache Spark Cluster Running

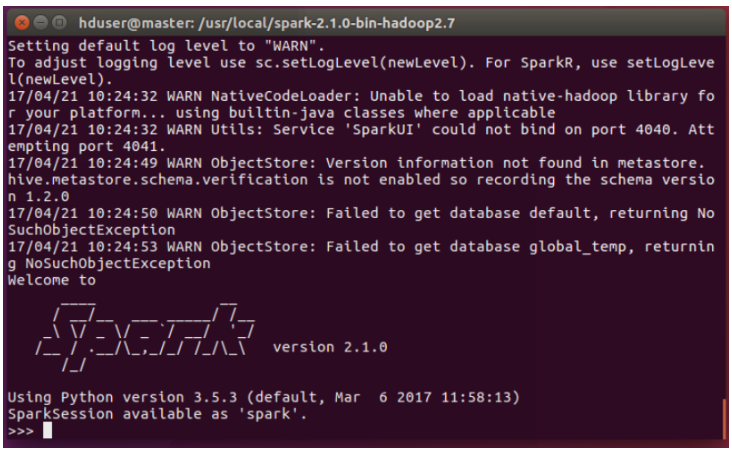


Figure 6.2.4: Running PySpark

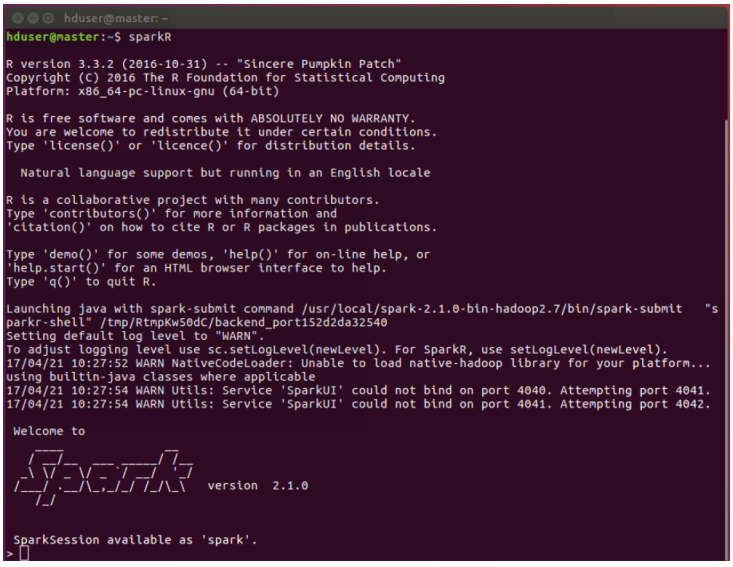
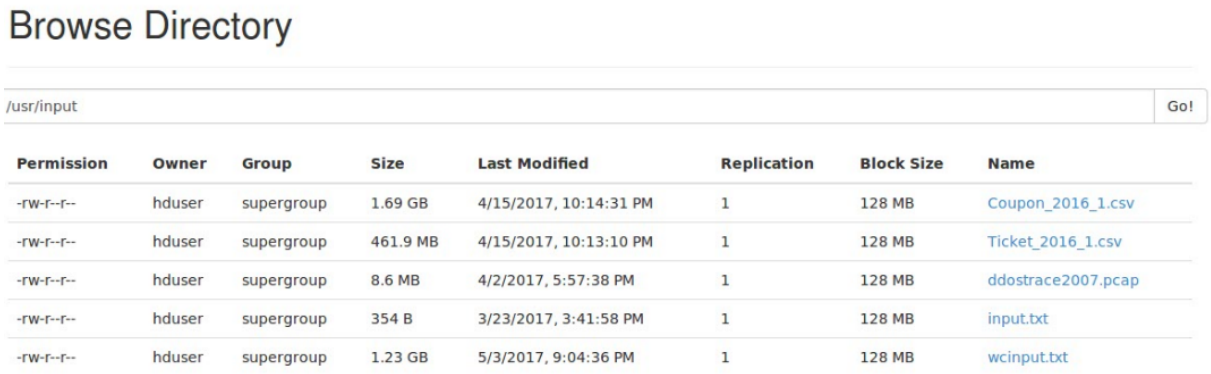


Figure 6.2.5: Running SparkR

After copying the input file(wcinput.txt) of size 1.23 GB onto HDFS, we can see it in the input directory, as shown below.

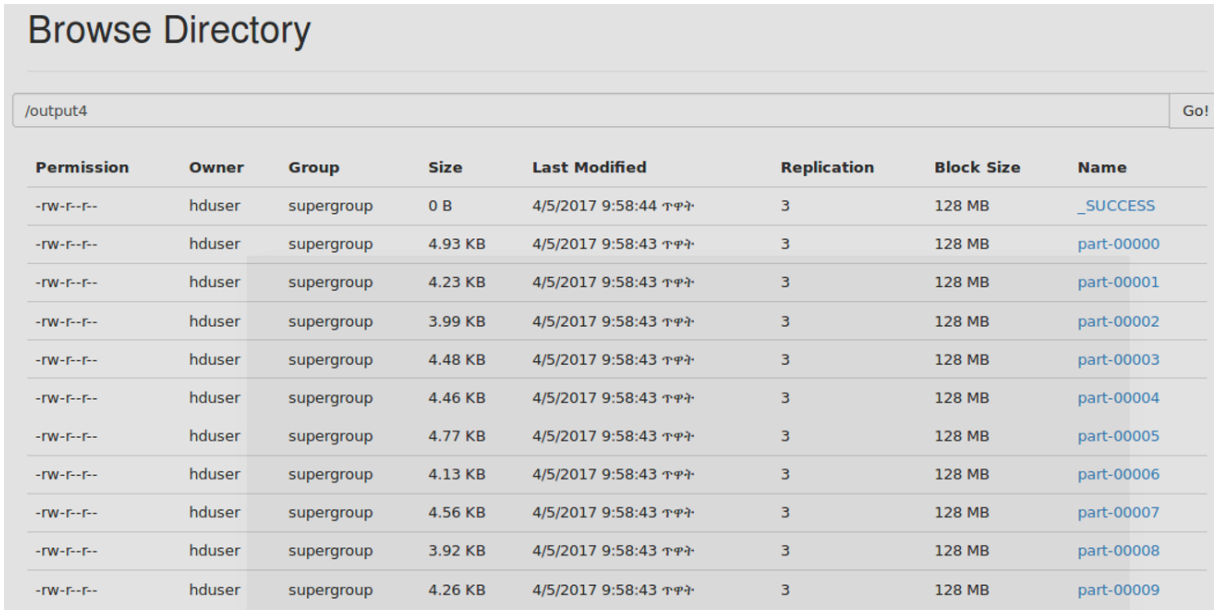


Browse Directory

/usr/input Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hduser	supergroup	1.69 GB	4/15/2017, 10:14:31 PM	1	128 MB	Coupon_2016_1.csv
-rw-r--r--	hduser	supergroup	461.9 MB	4/15/2017, 10:13:10 PM	1	128 MB	Ticket_2016_1.csv
-rw-r--r--	hduser	supergroup	8.6 MB	4/2/2017, 5:57:38 PM	1	128 MB	ddostrace2007.pcap
-rw-r--r--	hduser	supergroup	354 B	3/23/2017, 3:41:58 PM	1	128 MB	input.txt
-rw-r--r--	hduser	supergroup	1.23 GB	5/3/2017, 9:04:36 PM	1	128 MB	wcinput.txt

After running application, as shown section 5.2 of chapter 5, we can notice the output in specified output folder as shown below.



Browse Directory

/output4 Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hduser	supergroup	0 B	4/5/2017 9:58:44 ጥዋት	3	128 MB	_SUCCESS
-rw-r--r--	hduser	supergroup	4.93 KB	4/5/2017 9:58:43 ጥዋት	3	128 MB	part-00000
-rw-r--r--	hduser	supergroup	4.23 KB	4/5/2017 9:58:43 ጥዋት	3	128 MB	part-00001
-rw-r--r--	hduser	supergroup	3.99 KB	4/5/2017 9:58:43 ጥዋት	3	128 MB	part-00002
-rw-r--r--	hduser	supergroup	4.48 KB	4/5/2017 9:58:43 ጥዋት	3	128 MB	part-00003
-rw-r--r--	hduser	supergroup	4.46 KB	4/5/2017 9:58:43 ጥዋት	3	128 MB	part-00004
-rw-r--r--	hduser	supergroup	4.77 KB	4/5/2017 9:58:43 ጥዋት	3	128 MB	part-00005
-rw-r--r--	hduser	supergroup	4.13 KB	4/5/2017 9:58:43 ጥዋት	3	128 MB	part-00006
-rw-r--r--	hduser	supergroup	4.56 KB	4/5/2017 9:58:43 ጥዋት	3	128 MB	part-00007
-rw-r--r--	hduser	supergroup	3.92 KB	4/5/2017 9:58:43 ጥዋት	3	128 MB	part-00008
-rw-r--r--	hduser	supergroup	4.26 KB	4/5/2017 9:58:43 ጥዋት	3	128 MB	part-00009

It shows that the output is available in 9 partial files depending on the size of the input file.

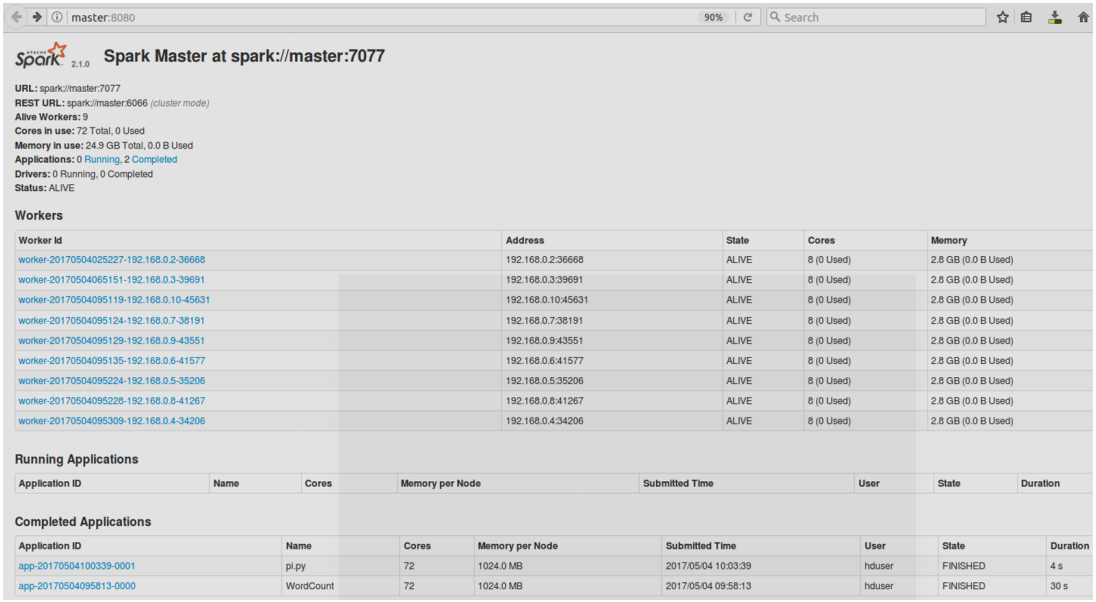
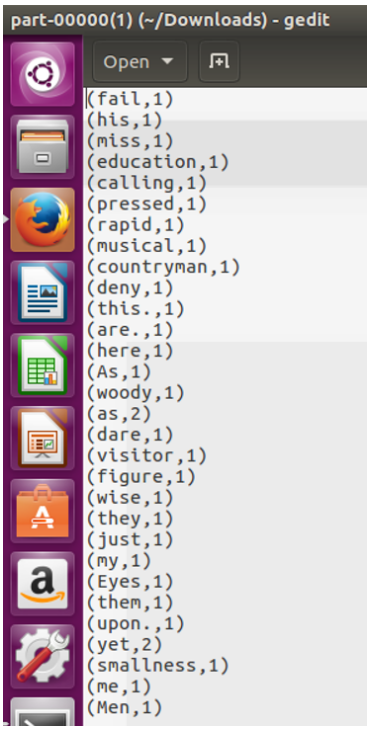
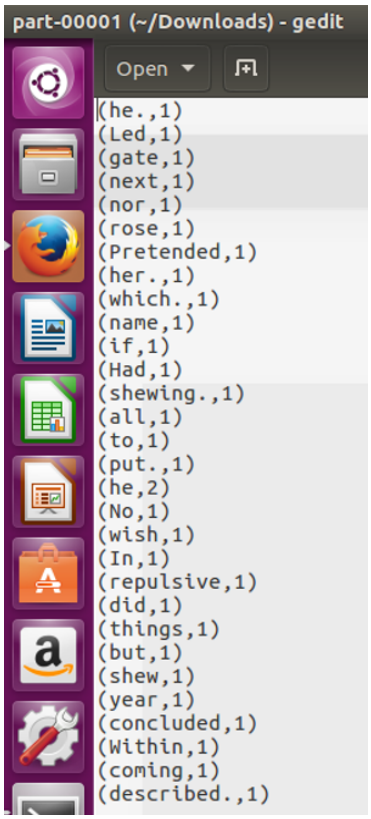


Figure 6.2.6: Pi App & WordCount Apps Execution on Multi Node Cluster

The content i.e. word counts of these two files are as shown below.





Similarly, the researchers were also able to run finding pi application on HPDA cluster and result that the submitted WordCount application used 72 cores and successfully processed in 4 seconds only.

We have also run the above mentioned applications i.e. WordCount and pi on a single machine which has 4 cores and 4 GB RAM. Then, the WordCount application successfully processed in

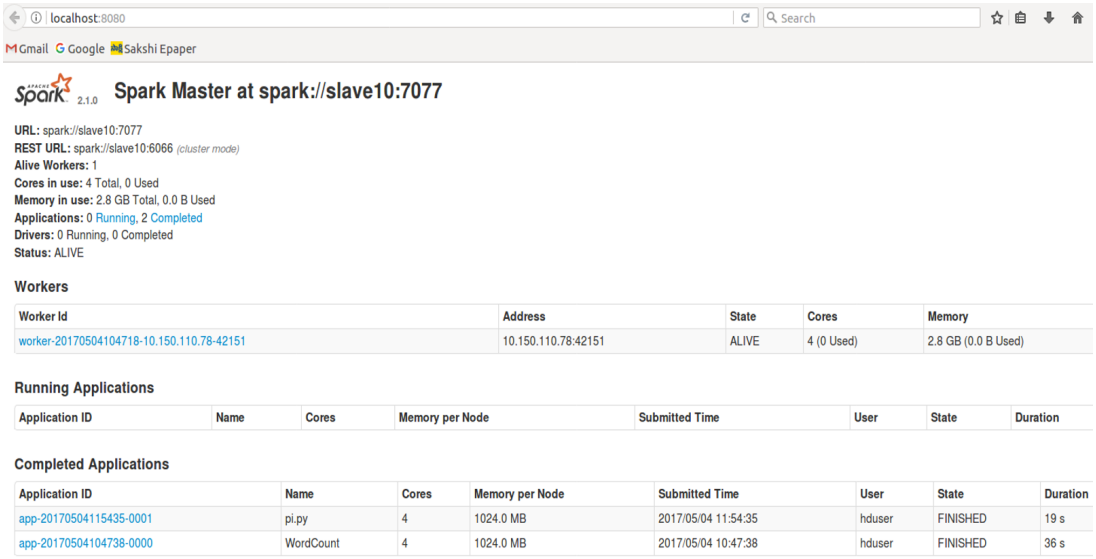


Figure 6.2.7: Pi App & WordCount Apps Execution on Single Node Cluster

36 seconds and pi application processed in 19 seconds which is been shown in below figure.

6.2.1 Performance Comparison of Single and Multi Node Clusters

The below table summarizes the processing time taken on single node and multi node clusters.

Table 6.2.1: Performance Comparison of HPDA Cluster with Single Node Cluster

Cluster Type/App	WordCount App	Pi App
Single Node Cluster (4 GB RAM, 1 TB HDD)	36 s	19 s
Multi Node Cluster (24.9 GB RAM, 6.4 TB HDD)	30 s	5 s

It has been observed that the execution time depends on the network communication(I/O), the number of search operations of words, size of the input file. The whole process of MapReduce processing and building up a cost function that explicitly models the relationship between the amount of input data, the available system resources (Map and Reduce slots), and the complexity of the Reduce function for the target MapReduce job.

6.3 Discussion

As per the research question 1, we have analysed the role of commodity hardware in cluster establishment. We have identified that at minimum an HPC cluster can be configured with Ethernet switch and cabling interconnectivity among available desktop computers. With the help of a suitable cluster management software. But during experiments our research team has found that the rate of failure of commodity computers is more as compared to the specially designed computers, also there is less than 50Mbps bit rate due to various types of delays between compute node and server nodes, which is quite less than the rate at which connected hard disk can supply data to a processor using SCSI.

Also under research question 2, we have studied the optimum configuration of HPC cluster system for various types of workloads. The choice of software and libraries is highly influenced by

type of parallelism available in application level programs. For shared memory programming we have identified that Torque was a good choice, for distributed memory scatter-gather pattern it was Hadoop based HPC cluster which shown better performance and for distributed in memory computing it was spark cluster. Also with the help of MPI library Torque supports distributed memory programming paradigm.

Under research question 3, we have analyzed the performance of various clusters under variable load conditions, but as per the scope of this research we have designed rather simple experiments which are limited to compute the latency profile of various clusters implemented. More sophisticated experiments involving the performance gain at compiler level optimizations, use of libraries , programming paradigms and different types of workloads shall be our focus area in nearest future.

Since this research was a pilot project, therefore the contribution of this project was to study and implement the various cluster management platforms which can be finally implemented in full scale. The full scale implementation needs a specialized hardware and a dedicated HPC data centre. We compared our platform with the AAU IGSSA platform and found that the current hardware and networking devices do not match with the expected HPC infrastructure in a full implementation.

Chapter 7

Conclusion and Future Scope

The research study has been carried with the available commodity hardware and open source tools and the researchers developed 2 types of clusters. Depending on the type and complexity of computation, individual cluster can be chosen to carry out jobs. The DeepStack cluster supports second generation or low-level programming languages such as C, C++ and Fortran. The HPDA cluster supports latest programming or high-level programming languages such as Java, Scala, Python and R.

The research concluded that if a problem is small enough to be accommodated and the computing resources such as cores and memory are sufficient, and data size is moderate then DeepStack cluster is a good choice. When data size is large and the tasks require iterative processing with high speed; then HPDA cluster is good choice.

However cluster monitoring, provisioning, scalability, upgrade and maintenance is a continuous activity but for the nearest future, our research team has following goals which also match with the guidelines of this research title:

1. Full scale implementation of DeepStack and HPDA clusters on dedicated HPC server with Gbps Ethernet.
2. Developing a Web Interface for user friendly job submission and monitoring.
3. Implementation of scripts, user accounts and job submission IP addresses.
4. Putting pbs_server at static IP. Possibly with a failover replication.
5. Enhancing clusters for extensive scientific library support for various intermediate programs.
6. Easy to use manual
7. Seminar and workshop for cluster usage for researchers.
8. Final publication of documentation, manual, research paper.

Bibliography

[1] Junehawk Lee; Hyojin Kang; Seokjong Yu; Chul Kim; Sang-Jun Yea, "Whole cancer genome analysis using an I/O aware job scheduler on high performance computing resource", *IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pp. 10–14, 2014.

[2] Wolfgang Karl Hermann HellWagner and Markus Leberecht, "Enabling a PC Cluster for High-Performance Computing", *The SPEEDUP Journal*, vol. 11, pp. 10–14, 1997.

[3] Andrew Chien, "High Performance Virtual Machines (HPVM):Clusters with Supercomputing APIs and Performance", 1997.

[4] Wikipedia, ", www.wikipedia.org.

[5] Cisco, "Cisco's VNI Global Traffic Forecast, 2015-2020", HPC Advisory Council, 2016.

[6] Sneha Gupta and Manoj S Chaudary, "Big Data Issues and Challenges", *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 3, no. 2, pp. 62–67, 2014.

[7] G. Hager and G. Wellein, *Introduction to High Performance Computing for Scientists and Engineers*, CRC Press - Taylor Francis Group, first edition edition, 2011.

[8] "A Review of High Performance Computing Foundations for Scientists", 2016.

[9] J Makino and M. Taiji, *Scientific Simulations with Special Purpose Computers: The GRAPE Systems*, Wiley, 1998.

[10] "On-demand data analytics in HPC environments at leadership computing facilities: Challenges and experiences", *IEEE International Conference,Big Data*, 2016.

[11] Allan R. Hoffman, "Supercomputers: directions in technology and applications", *National Academies*, pp. 35–47, 1990.

[12] "RADIOSS 13.0 Performance Benchmark and Profiling", HPC Advisory Council, http://hpcadvisorycouncil.com.

[13] "TORQUE resource manager", *Supercomputing Conference*, 2006.

[14] P. Mehrotra R. Biswas H. Lei J. Haoqiang, D. Jespersen and B. Chapman, "High performance computing using MPI and OpenMP on multi-core parallel systems", *Parallel Computing*, vol. 37, pp. 562–575, 2011.

[15] "HPC at Uinversity of Columbia", https://www.xsede.org/.

[16] "HPC at Standford University", https://hpcc.stanford.edu/.

[17] "FSU HPC cluster", https://rcc.fsu.edu/services/hpc.

[18] Sang Yeon Lee Sol Ji Kang and KeonMyung Lee, "Performance Comparison of OpenMP, MPI, and MapReduce in Practical Problems", *Hindawi Publishing Corporation Advances in Multimedia*, 2015.

[19] "OpenMP Application Program Interface", OpenMP Architecture Review Board, 2008, http://www.openmp.org/mp-documents/spec30.pdf.

[20] A. Lumsdaine et al W. Gropp, S. Huss-Lederman, *MPI: The Complete Reference*, vol. 2, The MIT Press, 1998.

[21] J.Dean and S.Ghemawat, "MapReduce: Simplified data processing on large clusters", *Hindawi Publishing Corporation Advances in Multimedia*, vol. 51, no. 1, pp. 107–113, 2008.

[22] S. J. Min A. Basumallik and R. Eigenmann, "Programming distributed memory systems using OpenMP", *IEEE International Parallel and Distributed Processing Symposium*, , no. 6, pp. 18, 2007.

[23] Michael J Franklin Scott Shenker Matei Zaharia, Mosharaf Chowdhury and Ion Stoica, "Spark: cluster computing with working sets", *In USENIX conference on Hot topics in cloud computing*, pp. 10–16, 2010.

[24] Matei Zaharia Michael J Franklin Scott Shenker Reynold S Xin, Josh Rosen and Ion Stoica, "Spark: Sql and rich analytics at scale", *In ACM SIGMOD International Conference on Management of data*, pp. 13–24, 2013.

[25] Tathagata Das Ankur Dave Justin Ma Murphy McCauley Michael J Franklin Scott Shenker Matei Zaharia, Mosharaf Chowdhury and Ion Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing", *In USENIX Conference on Networked Systems Design and Implementation*, 2012.

[26] Patrick Wendell Holden Karau, Andy Konwinski and Matei Zaharia, *Learning Spark*, O'Reilly Media, 2015.

[27] Luca Oneto Jorge L. Reyes-Ortiz and Davide Anguita, "Big Data Analytics in the Cloud: Spark on Hadoop vs MPI/OpenMP on Beowulf", *INNS Conference on Big Data*, pp. 121–130, 2015.

[28] Tom White, *Hadoop: The definitive guide*, O'Reilly Media, 2012.

[29] U. Srivastava R. Kumar C. Olston, B. Reed and A. Tomkins, "Pig latin: a not-so-foreign language for data processing", *In SIGMOD 08*, 2008.

[30] "Apache Hive", http://hadoop.apache.org/hive.

Appendix A

Multi Node Apache Hadoop Installation

```
Step 1. Installing Java

       Java is the main prerequisite for Hadoop. Install latest java.
       \$sudo apt-get update
       \$sudo apt-get install default-jdk
       Then, you should verify the existence of java using the command 'java - version'.
       \$java -version

Step 2. Note the IP address of master machine and slave machine
        \$ifconfig

Step 3. In the /etc/hostname file of master add the name of the name-node system.
        \$sudo gedit /etc/hostname
          master
        In the /etc/hostname file of slave add the name of the data-node system.
        slave1

Step 4. In the /etc/hosts file add the name-node(ip-address, name) and data-nodes(ip-address, name).
        Name-node is the master and data-node is the slave.
        \$sudo gedit /etc/hosts
          master-IP  master
          slave-IP slave1

          Restart the system

Step 5 Create a dedicated user account for hadoop
       \$sudo addgroup hadoop
       \$sudo adduser --ingroup hadoop hduser
       \$sudo usermod -a -G sudo hduser
       \$su hduser
        Restart the system & Login to hduser

Step 5. Configure ssh
        Hadoop requires SSH access to manage its nodes, therefore we need to install ssh on
both master and slave systems.

\$ sudo apt-get install openssh-server
          Now, we have to generate an SSH key on master machine. When it asks you to enter
a file name to save the key, do not give any name, just press enter.

        \$ ssh-keygen -t rsa -P " "
```

```
Second, you have to enable SSH access to your master machine with this newly created key.
        \$ cat \$HOME/.ssh/id_rsa.pub >> \$HOME/.ssh/authorized_keys
        Now test the SSH setup by connecting to your master.

        \$ ssh master
           Now run the below command to send the public key generated on master to slave.

        \$ ssh-copy-id -i \$HOME/.ssh/id_rsa.pub hduser@slave1
           Now that both master and slave have the pubic key, you can connect master to master
and master to slave as well.
        \$ ssh master
        \$ ssh slave1
        \$ exit
         On Master, Edit the masters file as below.
        \$ sudo gedit hadoop-2.7.3/etc/hadoop/masters
           master

         Edit the slaves file as below.
        \$ sudo gedit hadoop-2.7.3/etc/hadoop/slaves
           slave1

        On Slave, Edit the masters file as below.
        \$ sudo gedit hadoop-2.7.3/etc/hadoop/masters
           master

Step 6. Disable IPV6 by including the following lines in /etc/sysctl.conf file
        \$sudo gedit /etc/sysctl.conf

          net.ipv6.conf.all.disable_ipv6 = 1
          net.ipv6.conf.default.disable_ipv6 = 1
          net.ipv6.conf.lo.disable_ipv6 = 1

         Reboot the machine to make the changes and logon to hduser

Step 7. To find the java path
        \$ sudo update-alternatives --config javac

Step 8. Install Hadoop
        \$cd /usr/local

        \$sudo tar xvzf \$HOME/Downloads/hadoop-2.7.3.tar.gz
        \$sudo chmod 777 hadoop-2.7.3

Step 9. Set the hadoop environment variables.
        \$sudo gedit \$HOME/.bashrc
        Include the following lines in the \$HOME/.bashrc file

       # Set Hadoop-related environment variables export HADOOP_HOME=/usr/local/hadoop-2.7.3
       # Set JAVA home directory
         export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
       #Set aliases and functions for running Hadoop-related commands
        unalias fs &> /dev/null
        alias fs="hadoop fs"
        unaliash ls&> /dev/null
        alias hls="fs -ls"

       #Add Hadoop bin/ directory to PATH
```

```
        export PATH=\$PATH:\$HADOOP_HOME/bin

Step 10. Set hadoop environment variables.
         \$sudo gedit /etc/profile
         Include the following lines /etc/profile file
         # Insert JAVA_HOME JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
         # Insert HADOOP_PREFIX HADOOP_PREFIX=/usr/local/hadoop-2.7.3
         #--in PATH variable just append at the end of the line
 PATH=\$PATH:\$JAVA_HOME/bin:\$HADOOP_PREFIX/bin
         #--Append HADOOP_PREFIX at end of the export statement
         export PATH JAVA_HOME HADOOP_PREFIX

Step 11. Run the .bashrc& profile files from the \$ prompt for updating the changes
         \$ source \$HOME/.bashrc
         \$ source /etc/profile

Step 12. Check java & hadoop installation using
         \$ java -version
         \$ echo \$HADOOP_PREFIX
         \$ cd \$HADOOP_PREFIX
         \$ bin/hadoop version

Step 13. Configuration of the Hadoop files:
         hadoop-env.sh, core-site.xml, mapred-site.xml, hdfs-site.xml and yarn-site.xml
         \$cd etc/hadoop
          verify the path : /usr/local/hadoop-2.7.3/etc/hadoop
         13.1. Configuration of the hadoop-env.sh file
               \$sudo gedit hadoop-env.sh
                 export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
                 export HADOOP_PREFIX=/usr/local/hadoop-2.7.3

              Now we will create NameNode and DataNode directories.
              \$cd
              \$mkdir -p \$HADOOP_HOME/data/hdfs/namenode
              \$mkdir -p \$HADOOP_HOME/data/hdfs/datanode

         13.2. Configuration of the core-site.xml file
               \$sudo gedit core-site.xml
                 Include the following lines:
                 <property>
                   <name>fs.defaultFS</name>
                   <value>hdfs://master:9000</value>
                 </property>

         13.3 Configuration of the mapred-site.xml
              \$sudo cp mapred-site.xml.template mapred-site.xml
              \$sudo gedit mapred-site.xml
                Include the following lines in mapred-site.xml file:
                <property>
                   <name>mapreduce.framework.name</name>
                   <value>yarn</value>
                </property>

13.4 Configuration of the hdfs-site.xml
     \$sudo gedit hdfs-site.xml
       Include the following lines in hdfs-site.xml file:
```

```
    <property>
       <name>dfs.replication</name>
       <value>2</value>
    </property>
    <property>
       <name>dfs.permissions</name>
       <value>false</value>
    </property>
    <property>
       <name>dfs.namenode.name.dir</name>
       <value>/usr/local/hadoop-2.7.3/data/hdfs/namenode</value>
    </property>
    <property>
       <name>dfs.datanode.data.dir</name>
       <value>/usr/local/hadoop-2.7.3/data/hdfs/datanode</value>
    </property>

13.5 Configuration of the yarn-site.xml
     \$sudo gedit yarn-site.xml
     Include the following lines in yarn-site.xml file:
     <property>
         <name>yarn.nodemanager.aux-services</name>
         <value>mapreduce_shuffle</value>
     </property>
     <property>
         <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
         <value>org.apache.hadoop.mapred.ShuffleHandler</value>
     </property>

Step 14. Format the Hadoop File system implemented on top of the local file system using
         \$cd ..
         \$cd ..
         \$cd bin
           Verify the path :/usr/local/hadoop-2.7.3/bin
         \$hadoop namenode -format

Step 15. Start Hadoop using
         \$cd ..
         \$cd sbin
         Verify the path : /usr/local/hadoop-2.7.3/sbin
         \$./start-all.sh
         \$ jps

Step 15. Open Hadoop GUI in browser
         https://master:50070/
```

Appendix B

Multi Node Apache Spark Installation

```
Step 1: Install Java
        Java is the main prerequisite for Apache Spark. Install latest java.
        \$ sudo apt-get update
        \$ sudo apt-get install default-jdk
        Then, verify the existence of java in your system using the command 'java - version'.
        \$ java -version

Step 2. Set the Host Name as 'master' in /etc/hostname file

Step 3. Set the Known Hosts in /etc/hosts file
        \$ sudo gedit /etc/hosts

       192.168.0.1 master

        Restart the system \& Login

Step 4 Create a dedicated user account for hadoop
       \$ sudo addgroup hadoop
       \$ sudo adduser --ingroup hadoop hduser
       \$ sudo usermod -a -G sudo hduser
       \$ su hduser

        Restart the system \& Login to hduser

Step 5. Configure ssh

     5.1. Generate private and public key pair at terminal using

         \$ sudo apt-get install ssh
         \$ ssh-keygen

     5.2. To enable ssh to the local machine

         \$ cat \$HOME/.ssh/id_rsa.pub >>\$HOME/.ssh/authorized_keys
         \$ ssh master

Step 6. Disable IPV6 by including the following lines in /etc/sysctl.conf file
        \$ sudo gedit /etc/sysctl.conf

         net.ipv6.conf.all.disable_ipv6 = 1
         net.ipv6.conf.default.disable_ipv6 = 1
         net.ipv6.conf.lo.disable_ipv6 = 1
```

```
       Reboot the machine to make the changes and logon to hduser

Step 7. To find the java path
        \$sudo update-alternatives --config javac

Step 8. Install Scala
        \$sudo apt-get install scala
        or
        Download Scala and extract
        \$cd /usr/local
        \$sudo tar xvzf \$HOME/Downloads/scala-2.12.1.tar.gz
        \$sudo chmod 777 scala-2.12.1

Step 9. Download and Install Spark

        http://spark.apache.org/downloads.html

        Install Spark:
          \$cd /usr/local
          \$sudo tar xvzf \$HOME/Downloads/spark-2.1.0-bin-hadoop2.7.tar
          \$sudo chmod 777 spark-2.1.0-bin-hadoop2.7

Step 10. Set the environment variables.
         \$sudo gedit \$HOME/.bashrc

         Include the following lines in the \$HOME/.bashrc file

         # Set JAVA home directory
         export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64

         # Set Scala-related environment variables
         export SCALA_HOME=/usr/local/scala-2.12.1

         # Set Spark-related environment variables
         export SPARK_HOME=/usr/local/spark-2.1.0-bin-hadoop2.7

         #Add Spark bin/ directory to PATH
         export PATH=\$PATH:\$SCALA_HOME/bin:\$SPARK_HOME/bin

Step 11. Set environment variables.
         \$sudo gedit /etc/profile


         Include the following lines /etc/profile file

         # Insert JAVA_HOME
           JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64

         #--in PATH variable just append at the end of the line
 PATH=\$PATH:\$JAVA_HOME/bin:\$SCALA_HOME/bin:\$SPARK_HOME/bin

         #--Append SPARK_HOME at end of the export statement
         export PATH JAVA_HOME SCALA_HOME SPARK_HOME

Step 12. Run the .bashrc& profile files from the \$ prompt for updating the changes
         \$source \$HOME/.bashrc
```

```
        \$source /etc/profile

Step 13. Check Spark Installation
        \$ echo \$SPARK_HOME
        \$ bin/spark -version

Step 14. Configure Spark

        Edit configuration file spark-env.sh (in \$SPARK_HOME/conf/) and
 set following parameters: (Create a copy of template of spark-env.sh and rename it)

        \$ sudo cd \$SPARK_HOME
        \$ cd conf
        \$ sudo cp spark-env.sh.template spark-env.sh
        \$ sudo gedit spark-env.sh

         export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
         export SPARK_WORKER_CORES=8

Step 15. Add Slaves
        Create configuration file slaves (in \$SPARK_HOME/conf/) and add following entries:
       \$ sudo gedit slaves
          slave1
          slave2

Step 16: Install Spark on all slaves

        16.1 Setup Pre-requisites on all the slaves:

             Run following steps on all the slaves (or worker nodes):

             16.1.1 Add Entries in hosts file
             16.1.2 Install Java 7
             16.1.3 Install Scala
             16.1.4 Setup environment variables
             16.2 Copy setups from master to all the slaves
                16.2.1 Create tar-ball of configured setup:
                       \$ tar czf spark.tar.gz spark-2.1.0-bin-hadoop2.7
                          NOTE: Run this command on Master
                16.2.2 Copy the configured tar-ball on all the slaves
                       \$ scp spark.tar.gz slave01:~
                          NOTE: Run this command on Master
             16.3 Un-tar configured spark setup on all the slaves
                 \$ cd /usr/local
                 \$ sudo tar xzf /home/hduser/spark.tar.gz
                   NOTE: Run this command on all the slaves

Step 17. Start Spark Cluster
        17.1 Start Spark Services
             \$ sbin/start-all.sh
             Note: Run this command on Master
        17.2 Check whether services have been started
             17.2.1 Check daemons on Master
             \$ jps
                Master
             17.2.2 Check daemons on Slaves
                \$ jps
```

```
                    Worker

Step 18. Spark Web UI
         18.1 Spark Master UI
               Browse the Spark UI for information about the cluster resources (like CPU, memory),
details of worker nodes, running application, segments, etc.

http://MASTER-IP:8080/

        18.2 Spark application UI
             http://MASTER-IP:4040/

Step 19. Stop the Cluster

        Once all the applications have finished, you can stop the spark services
(master and slaves daemons) running on the cluster

        \$ sbin/stop-all.sh
           Note: Run this command on Master
```

Appendix C

Multi Node Torque Installation

```
1. Installation of the development tools:
    Necessary development tools are installed including various gCC compiler, assemblers
and interpreters for various languages.

2. Install Development Tools:
  yum groupinstall 'Development Tools'

3. Installation of Dependencies:

    TCL:
        yum install tcl

    LIBSSL:
     yum install libssl-devel

  LIBXML:
     yum install libxml2-devel

  OpenSSL:
     yum install libtool openssl-devel libxml2-devel boost-devel gcc gcc-c++

4. Intall and configure cpuset:
    yum install cpuset
    yum install hwloc-devel
    mkdir /dev/cpuset
    mount -t cpuset none /dev/cpuset
    ./configure --enable-cpuset
    sudo yum install libcgroup
    sudo service cgconfig start

5. Install Torque Server and configure on HPC server host
    [root]# tar -xzvf torque-6.1.1.tar.gz
    [root]# cd torque-6.1.1/
    [root]# ./configure --enable-cgroups --with-hwloc-path=/usr/local
    [root]# make
    [root]# make install

6. Install and Configure Compute Nodes
    ./torque-package-mom-linux-i686.sh --install

7. Install and configure client nodes
   ./torque-package-clients-linux-i686.sh --install
```

```
8. Test the cluster: Give a test job from a valid client who is registered
    at server in /etc/hosts.equiv directory

Echo sleep 10 | qsub #press enter 10 times

Each time qsub command is typed one job is submitted, in job01 it only sleep the compute node
for given amount of time and returns. After getting successful return EXIT CODE = 0
for a particular tarcejob command, we can assure that server and compute node are
cooperating for this job.
```