Projects in Big Data Software and Applications

Spring 2017

Bloomington, Indiana

Editor:
Gregor von Laszewski
Department of Intelligent Systems
Engeneering
Indiana University
laszewski@gmail.com

Contents

S17-IO-3004/report/report.pdf not submitted

S17-IO-3009/report/report.pdf not submitted

Remotely Deploying, Visualizing and Controlling a Robot Swarm with ROS

MATTHEW LAWSON[1] AND GREGOR VON LASZEWSKI[1,*]

[1] *School of Informatics and Computing, Bloomington, IN 47408, U.S.A.*
[*] *Corresponding authors: laszewski@gmail.com*

project-000, May 4, 2017

Our project demonstrates the feasability of harnessing remotely-located distributed computing environments, i.e., "clouds", to simulate large-scale robot swarms. Our proof-of-concept program creates a two-robot swarm on a cluster of remotely-located computers. It then pushes a visual simulation of the robots to the remote user. Finally, it sends a single command to the robots in order to demonstrate the feasibility of networked communication with the robots. The project utilizes two software packages from the Open Source Robotics Foundation (OSRF). Namely, it uses the *Robot Operating System* to define, create and control the virtual robots. The OSRF's *Gazebo* simulation software provides visualization of the simulation. We use *cloudmesh*, *Ansible* and a *nix shell script to deploy the software to a distributed computing environment.

Keywords: Cloud, I524, ROS, Gazebo, Robot, Swarm

https://github.com/cloudmesh/sp17-i524/tree/master/project/S17-IO-3010/report/report.pdf

1. INTRODUCTION

Simulating a single robot's actions and responses to its environment prior to real-world deployment mitigates risk and improves results at a relatively low cost. Therefore, it seems reasonable to conclude that simulating the actions and responses of a group of robots, e.g., a swarm, will also improve results at a low cost. However, deployment of an interconnected swarm of virtual robots on a remotely-located cluster of computers imposes additional requirements versus a locally-hosted single- or multi-robot deployment. For instance, accessing and configuring multiple computers presents a time and resource challenge in contrast to a single-host setup. In addition, network security measures, such as ssh keys and port access, impede ROS' intra-cluster communication capabilities. In order to address the unique requirements of a networked, remotely-located swarm, we create a multi-platform system to automate the creation and deployment of the virtual swarm.

2. VIRTUAL ROBOT SWARM COMPONENTS

2.1. Robot Operating System (ROS)

The Open Source Robotics Foundation's middleware product *Robot Operating System*, or ROS, provides a framework for writing operating systems for robots. ROS offers "a collection of tools, libraries, and conventions [meant to] simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms" [2]. The Open Source Robotics Foundation, hereinafter OSRF or the Foundation, attempts to meet the aforementioned objective by implementing ROS as a modular system. That is, ROS offers a core set of features, such as inter-process communication, that work with or without pre-existing, self-contained components for other tasks.

Figure 1 illustrates the ROS universe in three parts: a) the plumbing, ROS' communications infrastructure; b) the tools, such as ROS' visualization capabilities or its hardware drivers; and c) ROS' ecosystem, which represents ROS' core developers and maintainers, its contributors and its user base.

The modules or packages, which are analogous to packages in Linux repositories or libraries in other software distributions such as *R*, provide solutions for numerous robot-related challenges. General categories include a) drivers, such as sensor and actuator interfaces; b) platforms, for steering and image processing, etc.; c) algorithms, for task planning and obstacle avoidance; and, d) user interfaces, such as tele-operation and sensor data display. [3]

2.2. Gazebo

The Foundation also supports *Gazebo*, ROS' 3D virtual simulation software. "Gazebo...simulate[s] populations of robots in complex indoor and outdoor environments. [It] offers physics simulation at a much higher degree of fidelity [than gaming engines], a suite of sensors, and interfaces for both users and programs [4]." Gazebo's usefulness center on three main features: a) physics engines compatibility; b) its graphics engine;

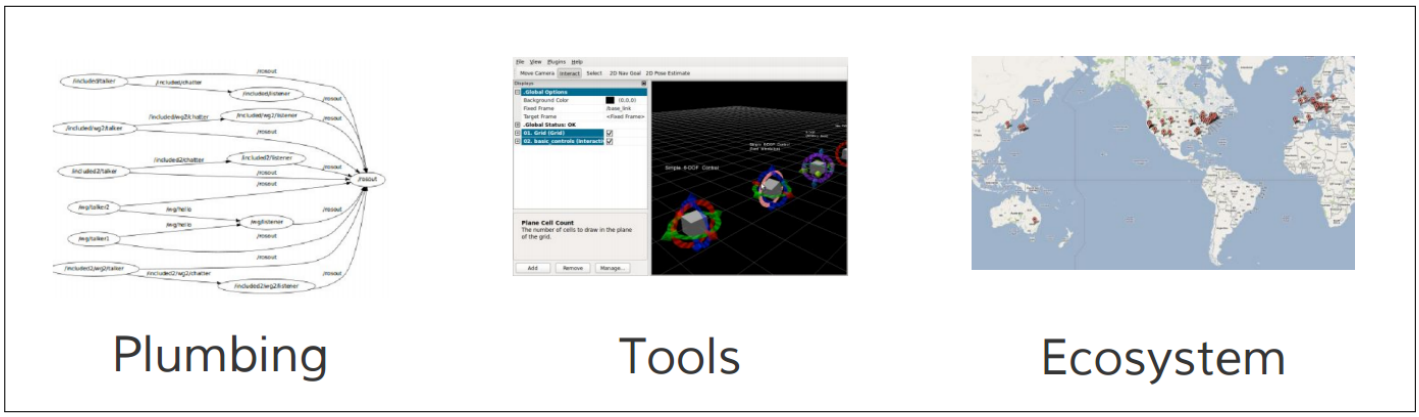


Fig. 1. A Conceptualization of What ROS, the *R*obot *O*perating *S*ystem, Offers to Roboticists [1]

and c) its sensor-data generators. with respect to physics engine compatibility Gazebo interfaces well with *Open Dynamics Engine* [5] (ODE), the default; b) *Bullet* [6]; *SimBody* [7]; and, *DART* [8]. Roboticists also benefit from its 3D graphics engine, *Object-oriented Graphics Rendering Engine* [9] (OGRE), which provides a C++ class library to "[abstract away] the details of using the underlying [graphics] system libraries like Direct3D and OpenGL [10]." Finally, Gazebo can supply *sensor* data to the virtual robot. Virtual sensor support ranges from 2D cameras to Kinect-style sensors. The system can also generate *noisy* data to better simulate real-world results.

Gazebo exists as a stand-alone project, suitable for use by programs other than ROS. However, it integrates tightly with ROS given its common ownership. In fact, the version supplied with a ROS installation automatically establishes communications between Gazebo and ROS for the end-user [11].

2.3. Ansible

Red Hat, Inc's [12] *Ansible* software purports to simplify numerous information technology tasks. It claims to do so by a) relying upon a human-readable script syntax, YAML; and b) by automating definable and repeatable IT tasks, such as configuration management and application deployment. Ansible's developers adopted a theater metaphor to describe the program's core functions. Thus, a computer's main duty within an IT infrastructure corresponds to the *role* an actor or actress might play in a theatrical production. Ansible calls the script a *playbook*, while the lines and directions within the script are referred to as *tasks*. Other aspects diverge from the metaphor, such as group vars and the config file (ansible.cfg). However, the *inventory* file hews to the metaphor - it represents the cast billing, the delineation of who plays what role. When used with an Ansible playbook, the inventory file specifies which servers belong to which logical group(s), i.e., which role(s).

As a result the software's applicability extends well beyond simplistic tasks even though Ansible's designers strive for simplification. In fact, an Ansible user can exercise fine-grained control over nearly every aspect of his or her IT infrastructure with a well-designed playbook.

Ansible also attempts to ease the burden of the IT administrator by eschewing SSL signing servers, daemons or client software. It simply pushes small programs to the target computers through an SSH connection to execute the desired tasks. When the task completes, Ansible removes the programs.

2.4. cloudmesh client toolkit

The *cloudmesh client toolkit* (cm) attempts to abstract away the complexities of establishing and utilizing different remotely-accessed computers and computer clusters [13]. Users can create, access and destroy a virtual machine or cluster of machines by issuing a single line of commands from a terminal emulator. cm supports access to clouds based on various back end-software stacks, including SLURM, SSH, Openstack and Heat. It provides an API, a command line client and a shell client.

2.5. Testing Environment

The Chameleon project's cluster of 650+ multi-core computer nodes, a joint venture between the University of Chicago and the Texas Advanced Computing Center, provides the infrastructre for the project. It is colloquially referred to as Chameleon Cloud or CC. A 100Gbps connection runs between the two centers. Project development primarily occurred on three-node clusters created as needed using Chameleon Cloud's *m1.medium flavor* of Ubuntu 16.04. The m1.medium flavor's resources consist of two virtual CPUs, 40GB of hard drive space and 4GB of RAM.

3. VIRTUAL ROBOT SWARM PROJECT IMPLEMENTATION

3.1. VR Swarm task

Although the swarm accomplishes the seemingly-trivial task of driving in circles, the real accomplishment rests in proving the feasability of automating the deployment of multiple controllable virtual robots on a remote cluster of computers and then visualizing them on a local computer.

3.2. Deployment

Achieving the aforementioned task requires coordinating a modestly-complex mix of shell commands, cloudmesh commands, Ansible commands and ROS / Gazebo commands. A shell script provides the automation for the shell, cloudmesh and ROS commands, while Ansible's *playbook* functionality handles the Ansible-focused portion. Only the initial step, retrieving and sourcing the shell script that begins the deployment process, requires user intervention. However, if the user wants to listen to the talker robot (*talker.py and listener.py*), s/he needs to follow the steps described in the README file.

***wget...begin* - Retrieve the Initalization Bash Script** Deployment begins when the user retrieves the initalization file, named *begin*,

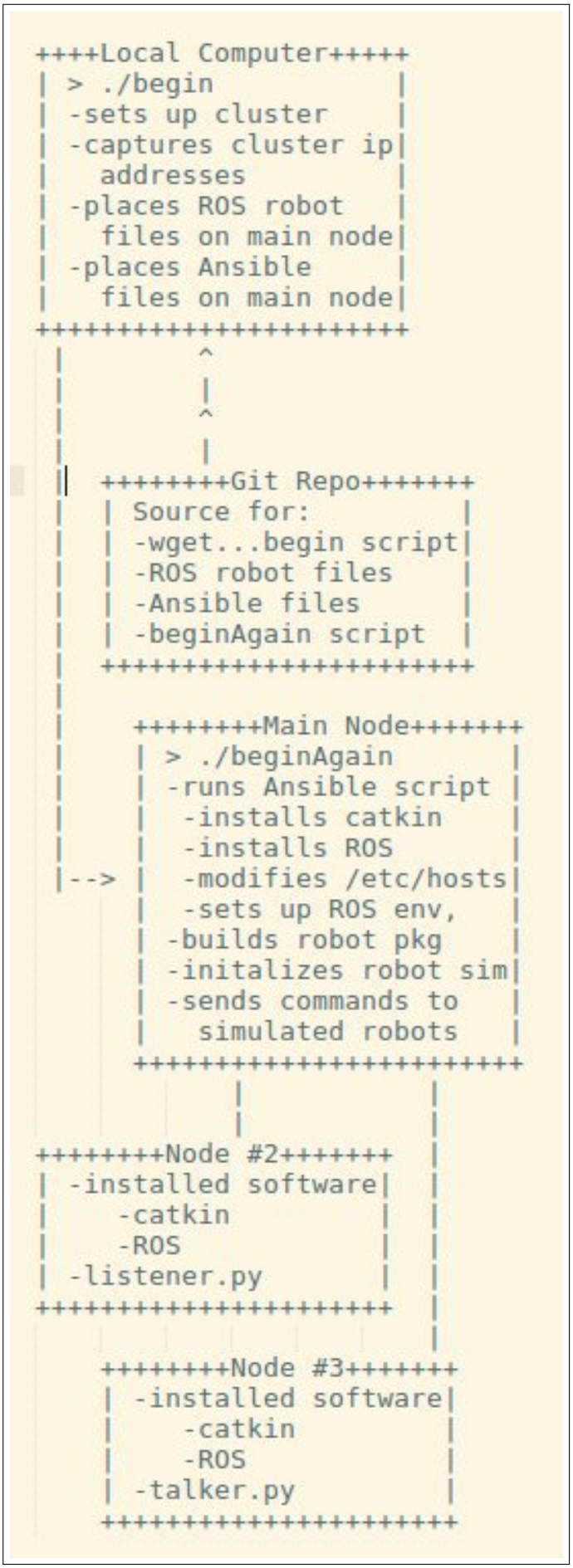


Fig. 2. Deployment Workflow for the Virtual Robot Swarm Project

from the project's Github repository. The user then starts the execution sequence by sourcing the file with the command

```
> ./begin
```

from the directory where the *begin* bash script resides.

***begin* bash script** The bash script calls three cloudmesh_client commands, `> cm cluster define −n rosA1 −c 3` , `> cm cluster allocate` and `> cm cluster cross_ssh` in order to create and prepare the three-node cluster named *rosA1*. The script then uses two additional Cloudmesh commands, `> cm cluster nodes` and `> cm vm ip show` to capture the public and private ip addresses of the cluster as well as the host names.

The private ip addresses, termed static ips by Chameleon, must be used because ROS nodes communicate by binding to any available port. That is, ROS does not specify the port beforehand, and it does not consider whether or not the firewall blocks the port for security purposes. As a result, intra-cluster communication requires access to every port, i.e., the firewall cannot close any port on any computer running ROS when that computer needs to communicate with another computer in the cluster. Obviously, this presents a major security concern, especially on shared resources.

In order to maintain security and enable intra-cluster communication, two of the three computer nodes must have their public ip addresses removed, i.e., disassociated, using Chameleon's browser-based graphical user interface (GUI), Horizon. Then, and only then, the ports can be opened by applying the *ros* security group to the two nodes in question (the private nodes). Since the third node, the main node, still has a public-facing ip address, the other two nodes can be accessed via an ssh connection (*ssh'ing*) to the main node and then ssh'ing to one of the private nodes.

The ip addresses and names are written to separate files for distinct purposes. The host names and private ip addresses are used to create a file to append to each cluster node's *known_hosts* file. The private ip addresses also end up in the inventory file Ansible references.

As its penultimate step, the *begin* script places all the necessary files for the demonstration on the correct cluster nodes. The ROS robot files and the Ansible files go onto the main node, while the script places a copy of the known_hosts addendum onto each cluster node. In addition, the script uses the *wget* command to copy the bash script for the next step in the process onto the main node.

The final lines of the file initiate the next step in the process and perform a few administrative duties.

***beginAgain* bash script** This script handles the software installation and initialization of the robots. It first establishes the veracity of the main node and the other nodes by connecting to each one in turn via ssh connection. If these steps do not occur before Ansible runs, the software installations never occur because Ansible hangs mid-process. Ansible installs all the necessary software on each cluster node. It installs Linux's *tree* package because the author prefers to use it to view directory structures; the ROS package; the rosbash package; the rosinstall package; and, the catkin_tools package. Ansible also creates a new known_hosts file for each computer node.

Upon completing the installation, the file's instructions initialize the robot workspace. The *catkin_tools* package accomplishes this on behalf of ROS. catkin_tools represents an evolution of ROS' built-in *catkin* family of commands. Initializing the robot workspace essentially involves the addition of numerous hidden helper files, which assist in building the robot package. Creating

the robot package, referred to as *building* the package, involves a series of automated *CMake* commands.

With the robot package built, the script copies another script from github. This small script starts ROS' *talker* robot, a simple robot included as part of ROS' tutorial packages. It relies upon Linux's *byobu* program to set up usable working environment if the user chooses to start the listener robot on the third cluster node.

Finally, execution of the last few lines of the script occurs. These lines start ROS, start the two robots, create a simulated world for the robots and then issue a command for the robots to drive in circles. Gazebo's simulation of the robots and their world come back through the ssh connection to the end user so s/he can see the simulation in real time.

before building the robot package that will be run in short order.

3.3. Modifications, Pitfalls

TBD; discuss any obstacles encountered with deployment due to dependency problems, connecting ROS and Gazebo, etc.

3.4. Initializing the Swarm

TBD; starting ROS and Gazebo to create the virtual environment; testing swarm interconnectivity; designating master node, etc.

3.5. Begin Task and Monitor Swarm's Progress

TBD; discuss the steps to initiate task completion and monitor the swarm's progress;

3.6. Information Acquired

TBD; discuss the iformation obtained from the swarm wrt the task at hand as well as each node's vital signs, e.g., battery level;

4. VR SWARM PROJECT CONCLUSIONS

TBD; present the data collected in some visualization format; discuss why this project advances robotics forward by utilizing distributed computing;

5. SUPPLEMENTAL MATERIAL

REFERENCES

[1] H. Boyer, "Open Source Robotics Foundation And The Robotics Fast Track," web page, nov 2015, accessed 19-mar-2017. [Online]. Available: https://www.osrfoundation.org/wordpress2/wp-content/uploads/2015/11/rft-boyer.pdf

[2] Open Source Robotics Foundation, "About ROS," Web page, mar 2017, accessed 16-mar-2017. [Online]. Available: http://www.ros.org/about-ros/

[3] National Instruments, "A Layered Approach to Designing Robot Software," Web page, mar 2017, accessed 18-mar-2017. [Online]. Available: http://www.ni.com/white-paper/13929/en/

[4] Open Source Robotics Foundation, "Beginner: Overview: What is Gazebo?" Web page, apr 2017, accessed 30-apr-2017. [Online]. Available: http://gazebosim.org/tutorials?cat=guided_b&tut=guided_b1

[5] Open Dynamics Engine, "Open Dynamics Engine Wiki," Web page, feb 2017, accessed 30-apr-2017. [Online]. Available: https://www.ode-wiki.org/wiki/index.php?title=Main_Page

[6] Real-Time Physics Simulation, "Bullet Physics Library: Real-Time Physic Simulation," Web page, apr 2017, accessed 30-apr-2017. [Online]. Available: http://bulletphysics.org/wordpress/

[7] P. E. Michael Sherman, "Simbody: Multibody Physics API," Web page, apr 2017, accessed 30-apr-2017. [Online]. Available: https://simtk.org/projects/simbody/

[8] Georgia Tech and Carnegie Mellon University, "DART: Dynamic Animation and Robotics Toolkit," Web page, mar 2017, accessed 30-apr-2017. [Online]. Available: http://dartsim.github.io

[9] Torus Knot Software Ltd, "OGRE3D," Web page, apr 2017, accessed 30-apr-2017. [Online]. Available: http://www.ogre3d.org

[10] Torus Knot Software Ltd, "OGRE: About," Web page, apr 2017, accessed 30-apr-2017. [Online]. Available: http://www.ogre3d.org/about

[11] Open Source Robotics Foundation, "gazebo-ros-pkgs: Package Summary," Web page, aug 2016, accessed 30-apr-2017. [Online]. Available: http://wiki.ros.org/gazebo-ros-pkgs

[12] Red Hat, Inc., "redhat," Web page, apr 2017, accessed 30-04-2017. [Online]. Available: https://www.redhat.com/en

[13] G. Laszewski, von, "Cloudmesh Client Toolkit," Web page, apr 2017, accessed 30-apr-2017. [Online]. Available: http://cloudmesh.github.io/client/

AUTHOR BIOGRAPHIES

Matthew Lawson received his BSBA, Finance in 1999 from the University of Tennessee, Knoxville. His research interests include data analysis, visualization and behavioral finance.

A. WORK BREAKDOWN

The work on this project was distributed as follows between the authors:

Matthew Lawson. Designed the project in collaboration w/ Gregor von Laszewski, researched the material and implemented the project. Slept far too little.

Gregor von Laszewski. Provided invaluable insights at key points during the process.

Charge Detection Mass Spectrometry

SCOTT MCCLARY[1,*]

[1] *School of Informatics and Computing, Bloomington, IN 47408, U.S.A.*
[*] *Corresponding authors: scmcclar@indiana.edu*

project-001, May 4, 2017

A Charge Detection Mass Spectrometry research application, developed at Indiana University by the Martin F. Jarrold research group, is used to indicate the performance and simplicity benefits of using Cloudmesh and Ansible Galaxy to deploy and run big data software on one or more virtual machines in the cloud. This proprietary research application was initially installed and run by hand on local servers and remote Supercomputers. The research application performed well on these powerful systems; however, the manual process of deploying and running the application turned out to be inefficient and too cumbersome for the domain scientists. Therefore, Cloudmesh and Ansible Galaxy were leveraged in order to automate the deployment of virtual clusters and execution of this research application in the cloud. This modification abstracted away the need for explicit human interaction while maintaining an efficient, reproducible and scalable Charge Detection Mass Spectrometry research workflow.

Keywords: Chemistry, Cloud, Hadoop Streaming, HPC, I524, Parallel Computing

Report: https://github.com/cloudmesh/sp17-i524/blob/master/project/S17-IO-3011/report/report.pdf
Code: https://github.com/cloudmesh/cloudmesh.cdms

1. INTRODUCTION

1.1. Research Background

The Martin F. Jarrold research group at Indiana University studies Charge Detection Mass Spectrometry (CDMS) [1]. Their general day-to-day workflow consists of conducting many scientific experiments using a Mass Spectrometer. This expensive scientific instrument creates raw frequency data at a rate of four (4) MB/s throughout the duration of each experiment. The research group has developed a Fast Fourier based application written in Fortran to processes this raw frequency data. The Fortran application generates human interpretable output, which assists the domain scientists in understanding the substances analyzed in the aforementioned experiments. The outputted results contain detailed mass information of the many ions discovered, which is used to solve important research topics such as the measurement and classification of the Hepatitis B virus. The mass and the abundance of the ions discovered by the application can be plotted to determine *Intermediates* that exist between definitive peaks in the plot, shown in Figure 1. This mass information can also be used to generate two and three-dimensional graphical representations of the ions, which help the domains scientists visualize the underlying structure of the Hepatitis B virus, shown in Figure 1.

1.2. General Problem

The Martin F. Jarrold research group has the ability to generate a lot of raw data, all of which needs to be processed by their Fortran application, as shown in Figure 2. A typical day conducting research consists of eight (8) to ten (10) one (1) hour experiments with each experiment generating raw frequency data at a rate of four (4) MB/s. Therefore, a single day of experiments has the ability to generate up to one hundred and forty four (144) GB of data. The research group must be able to process this data in a similar amount of time as the time required to generate the raw data. If their collection of compute resources is not powerful enough, they will quickly become inundated with piles and piles of raw data. This day-to-day research workflow typically strains the research group's local compute resources. Further-

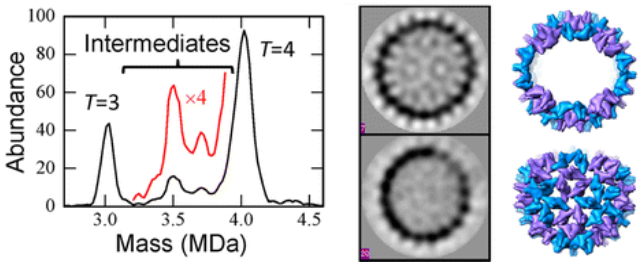


Fig. 1. The chart to the left displays an accurate measurement of the Hepatitis B virus (HBV) created by the research group [2]. This detailed mass information is used to create the images shown in the middle and to the right, which show 2-D and 3-D models of ions within the HBV. [2]

Fig. 2. The Martin F. Jarrold research group's pipeline is shown above. This pipeline includes a one (1) hour experiment, which creates approximately seven thousand (7,000) two (2) MB raw frequency files. Each of these files need to be transferred to the remote compute resource(s) and processed with a Fortran application to generate four (4) human interpretable output files.

more, the research group frequently makes algorithmic changes to the CDMS research application. When a significant change occurs, the research group must conduct a bulk reprocessing of months or even years worth of raw data. When a bulk reprocess is required, the limited compute resources available to the group become a significant limitation to the efficiency of their research. Additionally, when the application is run on remote systems, the raw input data must be transferred to the remote systems and the resulting output must be aggregated and then plotted in order to visualize and interpret the results. The process of moving data around by hand is time consuming and the process to aggregating results is tedious.

1.3. General Solution

The research group is composed of domain scientists who do not necessarily have backgrounds in Computer Science [3]. Therefore, a simple (i.e. automated) and reproducible solution must be developed in order to satisfy their day-to-day research workflow and their bulk reprocessing requirements.

1.3.1. Cloud Computing

Leveraging virtual clusters in the cloud to conduct their CDMS analysis increases their available compute power while simultaneously removing the need to explicitly manage a collection of compute resources. Furthermore, the ability to dynamically scale up or down the number of virtual machines aligns well with the evolving compute needs of the research group. The software tools Cloudmesh Client and Ansible Galaxy are at the foundation of this cloud computing solution [4, 5]. These two software tools collectively provide the ability to abstract away the technological details of the deployment and installation of virtual clusters in the cloud as well as automate the execution of the CDMS research application. These general modifications to their research workflow will ensure scalability, simplicity and reproducibility. These improvements allow the domain scientists in the Martin F. Jarrold research group to spend the majority of their time, effort and money on their research and not on the technological challenges of running the CDMS application.

2. ARCHITECTURE

The underlying architecture of this cloud computing research workflow is explicitly designed to facilitate automation. Cloudmesh and Ansible Galaxy are software tools that enable the creation of a virtual cluster, facilitate the deployment of software/data and automate the execution of the CDMS research application.

2.1. Software

2.1.1. Cloudmesh Client Toolkit

The Cloudmesh Client Toolkit provides an application programming interface (API), which allows users to simply manage a set of cloud resources (i.e. virtual machines, virtual clusters and etc.) [5]. The Cloudmesh Client Toolkit abstracts away the technological details of managing cloud computing resources.

2.1.2. Ansible Galaxy

Ansible is an information technology automation service designed for software deployment and execution [6]. Ansible Galaxy is an Ansible community, which "provides pre-packaged units of work known to Ansible as roles" [4]. Ansible Galaxy's pre-packaged units of work are essentially shared solutions to common automation tasks. This is a representation of the open source style of the Ansible Galaxy community. Ansible Galaxy promotes fast development since the wheel does not need to be reinvented for the automation of common tasks.

2.1.3. CDMS Application

The Martin F. Jarrold Group has written a Fast Fourier based application written in Fortran in order to conduct their CDMS research. This application is composed of approximately fifteen thousand (15,000) lines of Fortran code. Depending on the input, about 60% to 70% of the total compute time is spent within the external Intel Math Kernel Library (MKL) conducting the required Fast Fourier Transformations (FFT) [7].

2.1.4. Intel

The CDMS source code is compiled with the Intel compiler [8]. The CDMS application relies on the Math Kernel Library (MKL) to leverage efficient Fast Fourier Computations [7]. The application also leverages the Intel OpenMP parallel framework in order to divide the work amongst available CPU's [9]. Therefore, the Intel software is a fundamental piece of the architecture, which provides the compiler, MKL, and OpenMP functionality.

2.1.5. Hadoop

Apache Hadoop "is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models" [10]. Therefore, Hadoop must be installed at the foundation of each virtual machine in the cloud in order to leverage multiple virtual machines during the CDMS processing phase of the workflow shown in Figure 2.

2.2. Data

The CDMS application requires a set of raw two (2) MB files as input. In order to develop and test the efficiency of the deployment, a small dataset was used to generate all of the performance results. This small test dataset, composed of two hundred (200) files has a total size of four hundred (400) MB and is a representative sample. A typical dataset for the research group is approximately fourteen (14) GB in size. In a single day, up to ten (10) datasets are created and need to be processed.

3. LICENSING

3.1. CDMS Deployment Scripts

The source code (i.e. Bash, Ansible, Python) presented here is licensed under the Apache License, Version 2.0 [11].

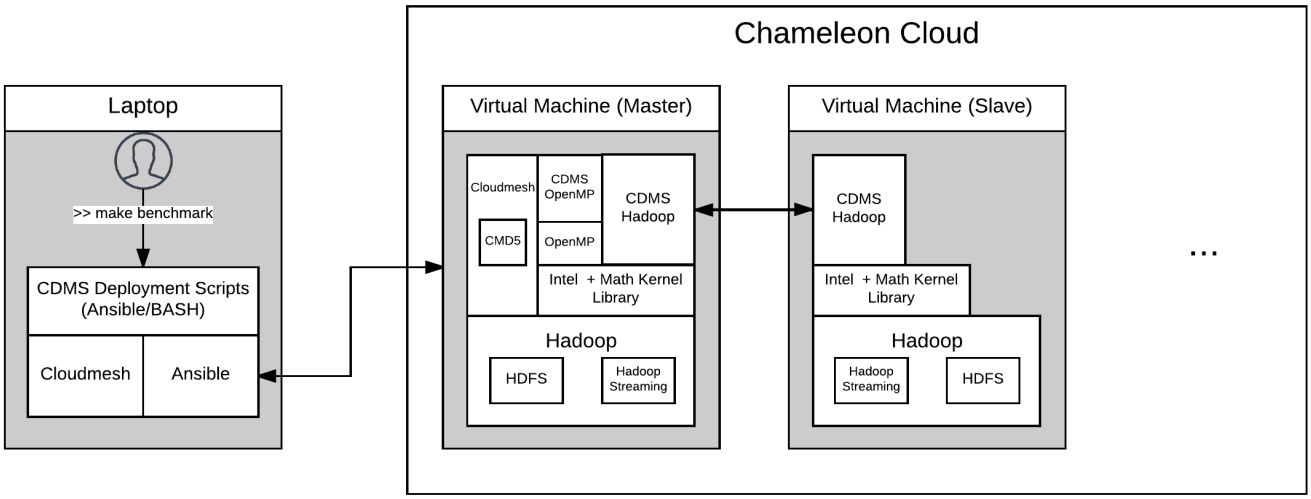


Fig. 3. The figure above provides a visual representation of the underlying architecture required for the Charge Detection Mass Spectrometry Cloud Computing workflow. The necessary software (i.e. Cloudmesh, Ansible, Hadoop, Intel, and etc.) and compute resources (i.e. Laptop and Virtual Machines) are explicitly shown above.

3.2. CDMS Application

The Martin F. Jarrold Group research group owns all of the rights to the Fortran Source code and data [1]. All distribution of the application and data must be consented by the research group.

3.3. Intel

The Intel software requires a license in order to complete the installation [12]. A student license is obtainable for free with an *EDU* email address; however, the leveraging the Indiana University Intel license server would provide a more complete and reproducible solution. In order to use the Indiana University Intel license server, the Virtual Machines must reside in the Indiana University IP address space. This can be achieved by connecting each virtual machine to Indiana University's Virtual Private Network (i.e. VPN) [13]. In order to connect to the VPN, one must connect via DUO Authentication (i.e. use a phone or token to validate) [14]. Given the complexity and reliability concerns with connecting to Indiana University's VPN simultaneously on multiple virtual machines, the Intel software is activated with the free Intel student license in order to promote simplicity and reproducibility.

3.3.1. Student License Limitations

The CDMS Deployment Scripts that were developed for this project leverage a free Intel student license to compile and link the CDMS application. While anyone can use this student license, it is registered to the author of this paper. This student license is *System Locked* and therefore can be installed on at most five (5) virtual machines. Once this threshold has been passed, the Intel software (i.e. compiler, MKL and OpenMP) can no longer be activated. This limitation somewhat inhibits the reproducibility and scalability of the research workflow. If a license registration error occurs during the Intel build phase of the deployment of the software, please contact the author of this paper. The author has the ability to uninstall the license from the currently registered hosts using Intel's Registration Center [15].

4. PARALLELIZATION

The Charge Detection Mass Spectrometry input data is split into many two (2) MB files. Conveniently, the data within each file is entirely independent to the data in the other input files. Therefore, the input data files can be processed simultaneously (i.e. in parallel). Parallel processing may not seem important when working on our sample dataset composed of two hundred (200) files; however, when a large collection of data requires reprocessing, parallel processing becomes critical to the efficiency of the Martin F. Jarrold research group.

4.1. OpenMP

OpenMP is a shared memory parallelization framework that is specified with simple compiler directives [9]. The shared memory parallelization structure limits the scalability of the application to a single node or virtual machine. This is in contrast to distributed memory parallelization, such as Message Passing Interface (MPI) or Hadoop, which enables multi-node parallelization [10, 16]. The original developers of the CDMS application decided to leverage OpenMP parallelization in order to exploit the natural data independency and improve overall performance of the application. However, this design choice limited the parallelization (i.e. scalability) to a single node or virtual machine.

4.2. Hadoop

In order to improve the overall performance and scalability, a distributed processing framework such as Hadoop must be integrated into the foundation of the CDMS application. Such an enhancement would allow for parallelization across multiple virtual machines. As discussed in Section 2.1.3, the source code is composed of fifteen thousand (15,000) lines of legacy Fortran code that interfaces with the Intel Math Kernel Library. In order to leverage the Hadoop MapReduce framework, the source code would need to be rewritten in a compatible programming language such as Java or Python.

4.2.1. Hadoop Streaming

Hadoop Streaming provides an alternative to rewriting the application in a compatible programming language. Hadoop Streaming allows one to "create and run Map/Reduce jobs with any executable or script as the mapper and/or the reducer" [17]. In Hadoop Streaming, "the mapper and the reducer are executables that read the input from stdin (line by line) and emit the output to stdout" [17]. The CDMS application is designed to read and write data to local files on disk; therefore, source code modifications were required in order to ensure the application read from stdin and wrote to stdout. The overall structure of the application and its data, which is naturally split into many relatively small files, allowed for a straightforward transformation from OpenMP parallelization Hadoop Streaming parallelization, as shown in Figure 4. Altering the way in which data was inputted to and outputted from the CDMS application was the only modification that was required in order to integrate Hadoop Streaming parallelization.

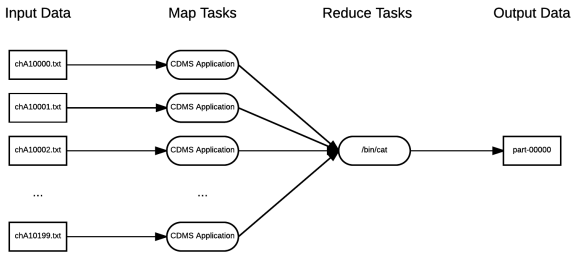


Fig. 4. The diagram shown above indicates the MapReduce style analysis of the CDMS Hadoop Streaming version of the application. Each two (2) MB input file is processed independently by the CDMS application and the results are aggregated with a single *cat* reduce task.

5. GETTING STARTED

The CDMS Deployment Scripts were specifically designed to promote simplicity and reproducibility. The following subsections describe how to use the CDMS Deployment Scripts to install and run the CDMS application in the cloud with as little as one simple command.

5.1. Requirements

In order to execute the CDMS Deployment Scripts, one must have the Cloudmesh Client installed and configured on their local system. This includes having a valid ∼/.cloudmesh/cloudmesh.yaml configuration file, registering Chameleon as the default cloud, registering a profile, uploading a ssh key and uploading a secgroup.

5.2. Fetch Code

The CDMS Deployment Scripts are hosted using GitHub [18]. A single repository contains the required Ansible and Bash scripts used to launch the CDMS research workflow [19]. See the following Bash commands:

```
>> git clone [REPOSITORY]
>> cd sp17-i524/project/S17-IO-3011/code
```

5.3. Benchmark

A single command will deploy the Hadoop virtual cluster, install the required software, run the three versions (i.e. Serial, OpenMP and Hadoop Streaming) of the CDMS application, aggregate the results, create plots of the output and delete the Hadoop virtual cluster. Timing information for each of these stages is printed to the screen once the benchmark has completed. The performance of this benchmark is plotted and explained in Section 7. See the following Bash command:

```
>> make benchmark
```

By default the benchmark will be run on a virtual cluster containing a single virtual machine. You can modify the maximum number of virtual machines to be used in the benchmark by passing in an optional argument to the *benchmark* Makefile option. The example shown below will run the entire benchmark with one, two and three virtual machines. See the following Bash command:

```
>> make benchmark num_nodes=3
```

5.4. Additional Commands

In case one would like to break up the aforementioned benchmark into individual pieces, there are separate Bash commands available. See the following Bash commands:

```
>> make deploy [num_nodes=n]
>> make install
>> make run
>> make view
>> make delete
>> make clean
```

5.4.1. Deploy

The *deploy* Makefile option leverages Cloudmesh Client to deploy a Hadoop virtual cluster in the Chameleon Cloud [20]. By default one (1) virtual machines will be created with the *deploy* option. The specific number of virtual machines deployed can be configured by passing in num_nodes=n, where n is the number of virtual machines requested to be deployed in the virtual cluster.

5.4.2. Install

The *install* Makefile option installs necessary software (i.e. Intel Compiler, Intel MKL, Python, Pip, Cloudmesh, Git, Charge Detection Mass Spectrometry application, and etc.) on the master and slave virtual machines of the active virtual cluster.

5.4.3. Run

The *run* Makefile option runs the serial, OpenMP, and Hadoop Streaming versions of the CDMS application on the active virtual cluster using the small test dataset containing two hundred (200) input files.

5.4.4. View

The *view* Makefile option aggregates the output data from the virtual machines in the active cluster, plots the results using Python's matplotlib and transfers a subset of the plots to the local system in order to visually validate the accuracy of the application [21].

5.4.5. Delete

The *delete* Makefile option deletes all of the virtual machines associated with active virtual cluster.

5.4.6. Clean

The *clean* Makefile option removes all of the local output files, if any exist.

6. COMPUTE RESOURCES

The CDMS OpenMP parallel version application was tested on multiple compute resources, as explained in Section 7. Each resource has unique architectural qualities that impact the performance, scalability and degree of parallelism. While the degree of parallelism (i.e. number of CPUs) may not be indicative of application performance, it certainly provides a baseline understanding of some architectural differences amongst the four (4) available systems.

6.1. Windows HPC Server

The Martin F. Jarrold's local Windows HPC Server has eight (8) CPUs; therefore, the Charge Detection Mass Spectrometry OpenMP version application can process up to eight (8) input files in parallel.

6.2. Karst

Indiana University's Linux Supercomputer, named Karst, has sixteen (16) CPUs per node; therefore, the Charge Detection Mass Spectrometry application can process up to sixteen (16) input files in parallel [22].

6.3. Big Red II

Indiana University's Linux Supercomputer, named Big Red II, has thirty-two (32) CPUs per node; therefore, the Charge Detection Mass Spectrometry OpenMP application can process up to thirty-two (32) input files in parallel [23].

6.4. Chameleon Cloud

The Cloudmesh Client allows one to specify different flavors of virtual machines to be deployed in the Chameleon Cloud [5, 20]. These flavors come in various sizes (i.e. Memory, vCPUs, and etc.). As shown in Table 1, these flavors can be used strategically to specify the number of virtual CPUs allocated to each virtual machine. As an example, the Chameleon Cloud m1.xlarge flavor provides eight (8) vCPUs. This allows the Charge Detection Mass Spectrometry OpenMP application to process up to eight (8) input files in parallel. Additionally, the virtual machines deployed in the Chameleon Cloud are running version 14.04 of the Ubuntu operating system.

Chameleon Cloud Virtual Machine Flavors

# Flavor	# of vCPUs
m1.medium	2
m1.large	4
m1.xlarge	8

Table 1. The table above indicates the number of virtual CPUs allocated to the various virtual machine flavors in the Chameleon Cloud [20]. The number of vCPUs indicates the maximum degree of parallelism for the CDMS application.

7. PERFORMANCE RESULTS

The following subsections describe the performance of the OpenMP and Hadoop Streaming versions of the application. These performance results only include the time-to-solution of the application processing the two hundred (200) input files. Perforamnce results including the entire deployment, installation and execution will be explained in Section 7.3.

7.1. OpenMP Scalability

As discussed in Section 4.1, the application was initially parallelized using OpenMP. This version of the application attempts to utilize the computational power available on a single node or virtual machine. Figure 5 compares the time-to-solution performance of the OpenMP version of the application on the available compute resources (i.e. local servers, Supercomputers and clouds) introduced in Section 6. As expected the time-to-solution (i.e. execution time) of the CDMS OpenMP version of the application decreases as amount of compute resources increase, as shown in Figure 5. For instance, on Karst the application running with sixteen (16) OpenMP threads completes in 9.4% of the time required for the application running with one (1) OpenMP thread.

The application performs most efficiently on Karst when using sixteen (16) CPUs (i.e. OpenMP threads). However, when the application is run using one (1), two (2), four (4) or eight (8) CPUs, the best performance exists on the Chameleon Cloud, as shown in Figure 5. Specifically, the application performs 18% faster on a single Chameleon Cloud virtual machine when compared to running the application on eight (8) CPUs of Karst.

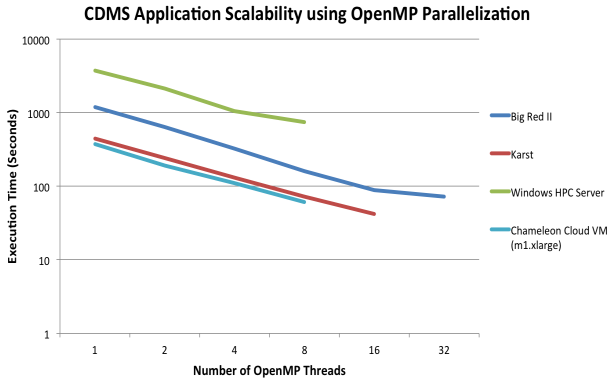


Fig. 5. The figure above shows the scalability (i.e. reduction in time-to-solution) as the number of OpenMP threads increase on local servers, Supercomputers and Clouds.

7.2. Hadoop Streaming Scalability

Unfortunately, the performance results for the Hadoop Streaming version of the CDMS application are not as promising as the performance results for the OpenMP version of the application. The Hadoop Streaming application does not exhibit the desired scalability. Since the application is essentially a map only Hadoop application, the performance (i.e. total runtime) of the application should decrease linearly as the number of virtual machines increase. However, the performance results shown in Figure 6 indicate that the execution time remains relatively consistent when one (1), two (2) or three (3) virtual machines are

used to process the two hundred (200) raw input files. Interestingly, the performance of the application significantly increases as the flavor of the virtual machine changes from smaller (i.e. less vCPUs) to larger (i.e. more vCPUs). Figure 6 shows that the Hadoop Streaming version of the application run on a m1.xlarge flavor requires only 43% of the execution time as the same application run on a m1.medium flavor.

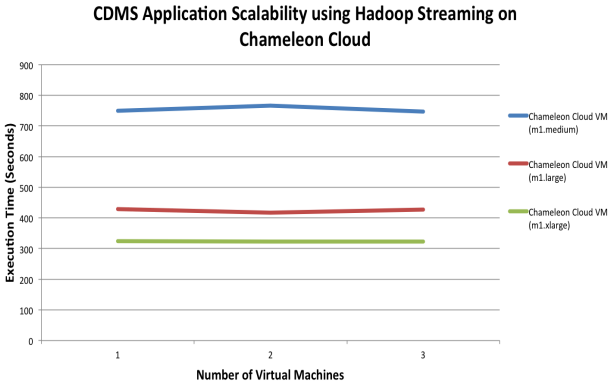


Fig. 6. The image above shows the scalability (i.e. reduction in time-to-solution) of the Charge Detection Mass Spectrometry Hadoop Streaming version of the application in a Chameleon Cloud virtual cluster. The performance information includes timing results for three (3) different virtual machine flavors.

7.3. Benchmark Scalability

The benchmark including the deployment of the virtual cluster, installation of the required software, execution the serial/OpenMP/Hadoop Streaming versions of the CDMS application and the aggregation of the results was tested using one (1), two (2) and three (3) virtual machines in the Chameleon Cloud. Interestingly, this benchmark required increasing time as the number of virtual machines increased, as shown in Figure 7. This performance information indicates that the deployment overhead outweighs the potential benefits of leveraging multiple virtual machines. The lack of scalability shown in the Hadoop Streaming version of the application and the small dataset are major factors, which contribute to the overall performance results. Specifically, if the Hadoop Streaming version of the application exhibited linear scalability and the dataset was significantly larger, the overhead incurred would not be as impactful as shown in Figure 7.

8. FUTURE WORK

Future work includes analyzing the performance of the CDMS Hadoop Streaming application to understand the poor scalability when running on a virtual cluster containing multiple virtual machines. Once the scalability issue has been fixed, larger flavors and more virtual machines will be utilized in order to increase the performance of the Hadoop Streaming application.

Future work includes figuring out how to leverage Indiana University's Intel license server. This will increase reproducibility by allowing the Intel Compiler and Intel MKL to be installed on an unlimited number of virtual machines.

Future work includes dispersing the raw input data across multiple virtual machines and running an instance of the CDMS OpenMP version of the application on each virtual machine and

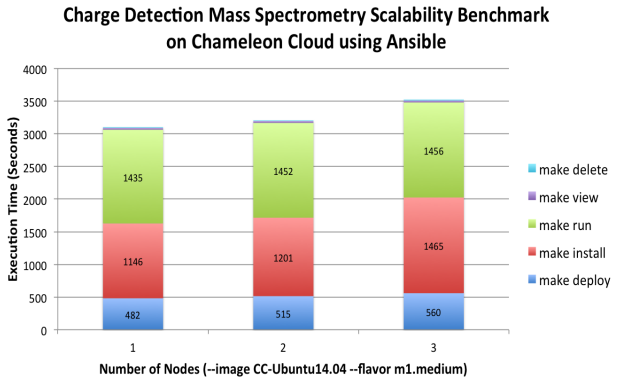


Fig. 7. The figure above indicates the time-to-solution for a full benchmark of the deployment of one (1), two (2) and three (3) virtual machines in the Chameleon Cloud. This benchmark includes depoying the virtual machines, installing all of the software, running the serial, OpenMP and Hadoop Streaming versions of the application and aggregating/plotting the results.

then aggregating the results. This modification will increase scalability for the shared memory parallelization.

Future work includes integrating Message Passing Interface (MPI) as the parallelization structure of the application rather than OpenMP or Hadoop Streaming [16]. This will allow for a dramatic increase in the scalability of the application.

9. CONCLUSION

The use of the Cloudmesh Client and Ansible Galaxy software to automate the execution of the Charge Detection Mass Spectrometry research application in the cloud improved the simplicity, efficiency and reproducibility. The automation allows the Martin F. Jarrold research group to focus on the details of their specific research rather than on the details of managing the software subsystems, executing the application and managing the input/output data. This automated cloud computing solution benefits the Martin F. Jarrold research group with respect to both simplicity and performance of the application. Streamlining the research workflow will inevitably result in an increase in productivity for the research group. An increase in research productivity may also result in an increase in grant funding and/or an increase in publications for the Indiana University research group.

The performance of the CDMS OpenMP version of the application performed favorably on a single Chameleon Cloud virtual machine when compared with a single node of the Indiana University High Performance Computing clusters (e.g. Karst and Big Red II). This unexpected result has sparked future work in optimizing the OpenMP version of the application for the Chameleon Cloud. However, the overhead included with deploying the virtual cluster and installing the necessary software causes the overall time-to-solution to increase dramatically.

The Hadoop Streaming version of the CDMS application did not exhibit optimal performance on a Chameleon Cloud virtual cluster. If the execution time reduced when more virtual machines were used, then this version of the application would become a viable solution for the research group's need to bulk reprocess raw input data. The Hadoop streaming version of the CDMS application is a step in the right direction; however, additional work must to be done to ensure scalability across

multiple virtual machines.

10. EXECUTION PLAN

The following subsections act as a timeline regarding how the project was divided up in order to complete all of the work by the desired deadline. The project execution plan is simply a guide and was followed diligently; however, some items were pushed slightly forwards or backwards as technological challenges were faced.

10.1. March 6, 2017 - March 12, 2017

This week I installed Cloudmesh on my local machine, created my first virtual machine on the Chameleon Cloud and tested Ansible Galaxy on remote systems such as one or more Chameleon Cloud virtual machine. I also wrote the project proposal, which eventually became this project report.

10.2. March 13, 2017 - March 19, 2017

This week I tested the deployment of the Intel Compiler on one or more Chameleon Cloud virtual machine using Cloudmesh and Ansible Galaxy. Given that I was out of town for Spring Break, I did not expect significant progress to be made during this week.

10.3. March 19, 2017 - March 26, 2017

This week I attempted to configure the Intel Compiler and Math Kernel Library to use the Indiana University Intel license server. Using this license server required connecting to Indiana University's Virtual Private Network (VPN) and using Two-Step Login (Duo) from the command line.

10.4. March 27, 2017 - April 2, 2017

This week I deployed the Charge Detection Mass Spectrometry research application along with the required input data on one or more Chameleon Cloud virtual machines using Cloudmesh and Ansible Galaxy.

10.5. April 3, 2017 - April 9, 2017

This week I modified the source code of the OpenMP parallel Charge Detection Mass Spectrometry research application to leverage Hadoop Streaming.

10.6. April 10, 2017 - April 16, 2017

This week I benchmarked the Charge Detection Mass Spectrometry research workflow on the Chameleon Cloud. This included varying the number and size of the virtual machines. I also wrote Python scripts to aggregate and plot the CDMS application's output from one or more virtual machines and locally visualize the results.

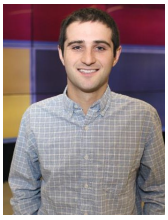
10.7. April 17, 2017 - April 23, 2017

This week I ensured the reproducibility of my source code as well as wrote and revised the final version of this report.

ACKNOWLEDGEMENTS

The authors would like to thank the School of Informatics and Computing for providing the Big Data Software and Projects (INFO-I524) course [24]. This project would not have been possible without the technical support & edification from Gregor von Laszewski and his distinguished colleagues.

AUTHOR BIOGRAPHIES



Scott McClary received his BSc (Computer Science) and Minor (Mathematics) in May 2016 from Indiana University and will receive his MSc (Computer Science) in May 2017 from Indiana University. His research interests are within scientific application performance analysis on large-scale HPC systems. He will begin working as a Software Engineer with General Electric Digital in San Ramon, CA in July 2017.

WORK BREAKDOWN

The work on this project was distributed as follows between the authors:

Scott McClary. He completed all of the work for this project including researching, deploying, testing and benchmarking the Charge Detection Mass Spectrometry research application as well as composing this paper.

REFERENCES

[1] Benjamin Draper, "MFJ Research Group," Web Page, Indiana University, 2017. [Online]. Available: http://www.indiana.edu/~nano/

[2] E. E. Pierson, D. Z. Keifer, L. Selzer, L. S. Le, N. C. Contino, J. C.-Y. Wang, A. Zlotnick, and M. F. Jarrold, "Detection of Late Intermediates in Virus Capsid Assembly by Charge Detection Mass Spectrometry," in *J. Am. Chem. Soc.* Department of Chemistry and Department of Molecular and Cellular Biochemistry, Indiana University, Bloomington, Indiana 47405,United States: ACS, 2014, pp. 3536—-3541. [Online]. Available: http://pubs.acs.org/doi/pdf/10.1021/ja411460w

[3] Benjamin Draper, "MFJ Research Group," Web Page, Indiana University, 2017. [Online]. Available: http://www.indiana.edu/~nano/group/

[4] Red Hat, Inc., "Ansible Galaxy | Find, reuse, and share the best Ansible content," Web Page, 2016. [Online]. Available: https://galaxy.ansible.com

[5] Gregor von Laszewski, "Cloudmesh Client Toolkit," Web Page, Indiana University, 2015. [Online]. Available: http://cloudmesh-client.readthedocs.io/en/latest/

[6] Red Hat, Inc., "Ansible is Simple IT Automation," Web Page, 2017. [Online]. Available: https://www.ansible.com

[7] md-rezaur-rahman (Intel) and Gennady F. (Intel), "The Intel Math Kernel Library and its Fast Fourier Transform Routines," Web Page, Jun. 2011. [Online]. Available: https://software.intel.com/en-us/articles/the-intel-math-kernel-library-and-its-fast-fourier-transform-routines

[8] Intel Corporation, "Intel Fortran Compiler," Web Page, 2017. [Online]. Available: https://software.intel.com/en-us/fortran-compilers

[9] OpenMP, "Home - OpenMP," Web Page, 2016. [Online]. Available: http://www.openmp.org

[10] The Apache Software Foundation, "Welcome to Apache Hadoop!" Web Page, Mar. 2017. [Online]. Available: https://cloudmesh.github.io/classes/

[11] The Apache Software Foundation, "Apache License," Web Page, 2017. [Online]. Available: https://www.apache.org/licenses/LICENSE-2.0.html

[12] Intel Corporation, "Product Licensing FAQ," Web Page, 2017. [Online]. Available: https://software.intel.com/en-us/faq/licensing

[13] Indiana University, "About the IU VPN," Web Page, Indiana University, Mar. 2017. [Online]. Available: https://kb.iu.edu/d/ajrq

[14] Indiana University, "What is Two-Step Login (Duo) and how does it work?" Web Page, Indiana University, Mar. 2017. [Online]. Available: https://kb.iu.edu/d/beum

[15] Intel Corporation, "Manage License," Web Page, 2017. [Online]. Available: https://registrationcenter.intel.com/en/products/license/2hws-f758wjvr/

[16] Software in the Public Interest, “Open MPI: Open Source High Performance Computing,” Web Page, Houston, TX, USA, Mar. 2017. [Online]. Available: https://www.open-mpi.org
[17] The Apache Software Foundation, “Hadoop Streaming,” Web Page, Aug. 2013. [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/streaming.html
[18] GitHub, Inc., “The world’s leading software development platform · GitHub,” Web Page, 2017. [Online]. Available: https://github.com
[19] GitHub, Inc., “Class submissions for Spring 2017 i524,” Web Page, 2017. [Online]. Available: https://github.com/cloudmesh/sp17-i524
[20] National Science Foundation, “Home | Chameleon Cloud,” Web Page. [Online]. Available: https://www.chameleoncloud.org
[21] The MathWorks, Inc., “Matplotlib: Python plotting - Matplotlib 2.0.0 documentation,” Web Page, Feb. 2017. [Online]. Available: https://matplotlib.org
[22] Indiana University, “Karst at Indiana University,” Web Page, Indiana University, Mar. 2017. [Online]. Available: https://kb.iu.edu/d/bezu
[23] Indiana University, “Big Red II at Indiana University,” Web Page, Indiana University, Mar. 2017. [Online]. Available: https://kb.iu.edu/d/bcqt
[24] Gregor von Laszewski and Badi Abdul-Wahid, “Big Data Classes,” Web Page, Indiana University, Jan. 2017. [Online]. Available: https://cloudmesh.github.io/classes/

Automated Sharded MongoDB Deployment and Benchmarking for Big Data Analysis

MARK MCCOMBE[1,*]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*] Corresponding author: mmccombe@iu.edu

S17-IO-3012, May 4, 2017

Using Python, Ansible, Bash Shell, and Cloudmesh Client a fully automated process is created for deploying a configurable MongoDB sharded cluster on Chameleon, FutureSystems, and Jetstream cloud computing environments. A user runs a single Python program which configures and deploys the environment based on parameters specified for numbers of Config Server Replicas, Mongos Instances, Shards, and Shard Replication. The process installs either MongoDB version 3.4 or 3.2 as requested by the user. Additionally, functionality exists to run benchmarking tests for each deployment, capturing statistics in a file as input for python visualization programs, the results of which are displayed in this report. These reports depict the impact of MongoDB version and degrees of sharding and replication on performance. Key performance findings regarding version, sharding, and replication are abstracted from this analysis. As background, technologies and concepts key to the deployment and benchmarking, such as MongoDB, Python, Ansible, Cloudmesh Client, and Openstack are examined.

Keywords: MongoDB, Cloud Computing, Ansible, Python, Cloudmesh Client, Openstack, I524

https://github.com/cloudmesh/sp17-i524/tree/master/project/S17-IO-3012/report/report.pdf

1. INTRODUCTION

As the final project for I524, Big Data Software and Projects, Spring 2017, a Python program invoking Bash shell scripts, Ansible playbooks, and Cloudmesh Client commands has been created to fully automate a configurable deployment of a MongoDB sharded cluster on various clouds. Chameleon Cloud, FutureSystems, and Jetstream are the currently supported cloud environments. The scripts have been developed and tested on an Ubuntu 16.04 LTS (Xenial Xerus) Virtual Machine running in Virtual Box. Using the Python cmd command line interface, the program project.py accepts parameters for deployment cloud, MongoDB version, Config Server server replication, number of Mongos instances, number of Data Shards, and Shard replication.

Also via project.py, automated benchmarking tests can be run. Tests were performed with various sharding and replication configurations to assess their impact on performance. Additionally, tests were run against MongoDB versions 3.4 and 3.2 to uncover any performance differences between these version. Performance results are captured and graphed using Python's matplotlib, the results of which are displayed and analyzed in this report.

2. INFRASTRUCTURE

Three clouds were selected for deployment: Chameleon Cloud, Futuresystems (also referred to as Kilo in some sections of this document), and Jetstream. In our automated deployment and benchmarking process, the cloud name is passed as a parameter to the deploy function of the main project.py script and a customized version of MongoDB is deployed to the selected cloud.

2.1. OpenStack

Chameleon Cloud, FutureSystems and Jetstream all utilize OpenStack. OpenStack is a free, open source cloud computing platform, primarily deployed as IaaS [1]. Openstack was created in 2010 as joint project between NASA and Rackspace that is currently managed by the OpenStack Foundation [1]. Open Stack is open source software released under the Apache 2.0 license [2].

Open Stack has various components, also known by code names [1]. Examples of Openstack components (and code names) are Compute (Nova), Networking (Neutron), Block Storage (Cinder), Identity (Keystone), Image (Glance), Object Storage (Swift), Dashboard (Horizon), Orchestration (Heat), Workflow (Mistral), Telemetry (Ceilometer), OpenStack Telemetry (Ceilometer), Database (Trove), Elastic Map Reduce (Sahara), Bare Metal (Ironic), Messaging (Zaqar), Shared File System

(Manila), DNS (Designate), Search (Searchlight), and Key Manager (Barbican) [1].

2.2. Chameleon Cloud

Chameleon is funded by the National Science Foundation and provides computing resources to the open research community. The Chameleon testbed is hosted at the Texas Advanced Computing Center and the University of Chicago. Chameleon provides resources to facilitate research and development in areas such as Infrastructure as a Service, Platform as a Service, and Software as a Service. Chameleon provides both an OpenStack Cloud and Bare Metal High Performance Computing Resouces [3].

2.3. FutureSystems

FutureSystems is a computing environment run by Indiana University that supports educational and research activities [4]. FutureSystems is directed by Geoffrey C. Fox and Gregor von Laszewski, both of Indiana University [4]. For our deployment, we utilize the OpenStack Kilo Cloud, running on the India machine. Because the environment is by default referred to as Kilo in the Cloudmesh documentation and setup file, it is referred to as both FutureSystems and Kilo in subsequent sections of this document and the accompanying diagrams.

2.4. Jetstream

Jetstream is a cloud computing environment implemented by many academic and industry partners including the University of Texas at Austin's Texas Advanced Computing Center (TACC), the Computation Institute at the University of Chicago, the University of Arizona, the University of Texas San Antonio, Johns Hopkins University, Penn State University, Cornell University, the University of Arkansas at Pine Bluff, the National Snow and Ice Data Center (NSIDC), the Odum Institute at the University of North Carolina, the University of Hawaii, and Dell [5]. At Indiana University, leadership is provided by the Pervasive Technology Institute with involvement from several members of the School of Informatics and Computing including Beth Plale, Katy Borner, and Volker Brendel [6].

2.5. Cloud Hardware Comparison

Table 1 shows a comparison of key computing resources on Chameleon, FutureSystems, and Jetstream cloud environments.

Table 1. Cloud Hardware Specification Comparison [7] [8] [9]

	FutureSystems	Chameleon	Jetstream
CPU	Xeon E5-2670	Xeon X5550	Haswell E-2680
cores	1024	1008	7680
speed	2.66GHz	2.3GHz	2.5GHz
RAM	3072GB	5376GB	40TBr
storage	335TB	1.5PB	2 TB

3. PYTHON/CMD

Python is utilized in two portions of the automated process. First, the main script, project.py, is a Python program that utilizes the cmd module to provide a simple command line interface [10] accepting parameters for deployment configuration. project.py also provides other functionality such as cluster deletion, benchmarking, benchmarking summarization and reporting, and data distribution reporting. Second, several visualization programs for benchmarking analysis are written in Python, utilizing the matplotlib and pandas modules.

4. ANSIBLE

Ansible is open source software typically use to automate software provisioning and configuration management. Ansible uses Playbooks specified in YAML file format to accomplish this goal. Ansible runs on Linux/Unix and requires Python [11].

In our deployment, virtual machines are created using Cloudmesh Client cluster commands. Once they are created, all direct cloud interaction for the MongoDB software installation and environment customization and setup is performance via Ansible playbooks.

5. CLOUDMESH CLIENT

The Cloudmesh Client toolkit is an open source client interface that standardizes access to various clouds, clusters, and workstations [12]. Cloudmesh Client is a python based application developed by Gregor von Laszewski and others collaborators primarily at Indiana University.

In the deployment, Cloudmesh Client is used to handle most interaction with the Virtual Machines in the clouds. Cloudmesh Client provides functionality in three main areas: Key Management, OpenStack Security, and virtual machine management. For key management, Cloudmesh's key add and upload commands simplify secure interaction with the cloud environments. For Openstack security, Cloudmesh's secgroup commands allow new security rules to be added and uploaded to the cloud. Virtual machine management is performed with Cloudmesh's cluster functionality, which allows easy creation and deletion of virtual machines and communication between them.

Cloudmesh Client simplifies and standardized interaction with the cloud for these tasks. This allows us to more easily port the deployment to additional clouds that are supported by Cloudmesh. Furthermore, by encapsulating the logic necessary to perform these tasks we are shielded from changes in interfaces made by individual clouds.

6. MONGODB

MongoDB is a popular open source, document oriented noSQL database. It stores documents in JSON-like schema-less formats called collections [13]. DBEngines ranks MongoDB as the most popular noSQL store and as the fifth most popular Database Management System overall [14].

6.1. Architecture

A sharded cluster in MongoDB has three main components, all of which will be implemented in our deployment:

- Config Servers - hold configurations setting and metadata
- Mongos - a query routing interface between applications and the cluster
- Shards - subsets of data

Figure 1 depicts a sharded MongoDB environment with two Mongos instances and two data Shards. The replica sets shown for both Config Servers and Shards may have any number of replicas within the set.

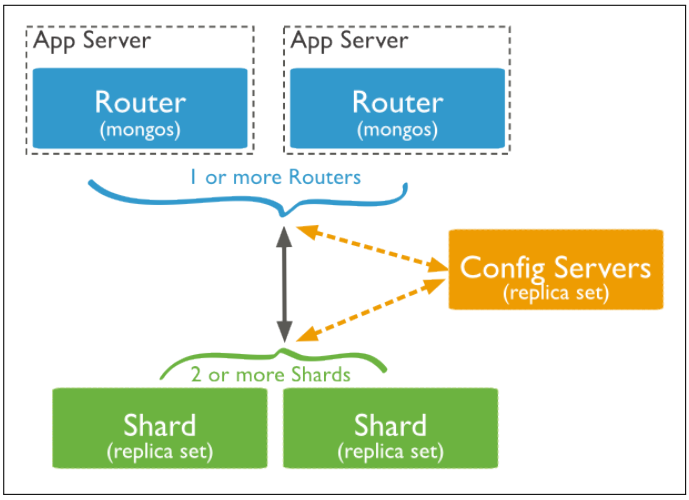


Fig. 1. Sharded MongoDB Architecture [15]

6.2. Config Servers

Config Servers stored metadata for sharded MongoDB clusters. This metadata includes information about the state and structure of the data and components of the sharded cluster [16].

Config Servers also contain authentication information. For example, information about the keyfiles used for internal authentication between the nodes is stored in the Config Servers [16].

In production deployments, it is recommended for Config Servers to be deployed in 3 member replica sets [15]. The rationale behind a 3 member set is discussed in more detail in the Replication subsection that follows.

In our deployment and benchmarking automation, the degree of replication in the Config Server Replica Set is by the third parameter to the main project.py script. For example, specifying 1 will create a Replica Set with one Config Servers (not replication), specifying 3 will create a Replica Set with three Config Server, and so on.

6.3. Mongos Routers

Mongos is a query routing service used in sharded MongoDB configurations. Queries from applications go through Mongos, which locates the data in the sharded cluster. The Mongos instances accomplish this by reading and caching data from the Config Servers [17].

For applications with high performance or availability requirements multiple Mongos instances may be ideal. In a high volume application, spreading the routing load over multiple Mongos instances can benefit performance. Additionally, multiple Mongos instances may increase availability in the case where a Mongos instance fails [16].

In our deployment and benchmarking automation, the number of Mongos instances is controlled by the fourth parameter to the main project.py script. For example, specifying 1 will create one Mongos instance, specifying 3 will create three Mongos instances, and so on.

6.4. Shards

Sharding, or distributing data across machines, is used by MongoDB to support large data sets and provide high throughput [18]. Our deployment and benchmarking will test the performance of various numbers of shards, measuring the performance improvements associated with sharding in MongoDB.

Documents are distributed among the shards using a shard key. A sharded collection must have one, and only one, shard key which must have a supporting index [18]. Since the shard key is critical to performance and efficiency, particular care must be given to shard key selection [19]. In our performance testing a key was chosen that would distribute data relatively evenly across the shards, but was not used in retrieving the data as the more costly retrieval of not using an index provided a better test case.

In our deployment and benchmarking automation, Sharding is controlled by the fifth parameter to the main project.py script. For example, specifying 1 cause only 1 shard to be created, specifying 3 will cause three shards to be created, and so on.

6.5. Replication

In databases, replication provides data redundancy leading to greater availability and fault tolerance. Replication in MongoDB is achieved via Replica Sets [20]. Replica Sets were implemented in our deployment for both Config Servers and Shards.

Two key benefits provided by replication are redundancy and fault tolerance. Each Replica in a Replica Set provides another copy of data. Higher redundancy means more nodes can be lost without data being lost. Higher numbers of Replicas in a set also increase fault tolerance, which leads to increased availability. As a general rule, a set will be able to tolerate faults while the majority of its nodes are still available

Table 2. Fault Tolerance by Replica Set Size [21]

Replica Members	Majority Needed	Fault Tolerance
2	1	0
3	2	1
4	3	1
5	3	2
6	4	2
7	4	3
8	5	3

As shown in Table 2, odd numbers of members in a replica set are a better choice for fault tolerance. For example, both a 3 and 4 replace set can only tolerate one member failing while maintaining availability. This is because a majority of the members must be available to maintain availability. In a 3 replica set the majority is 2, so it can tolerate 1 member failing. In a 4 replica set, the majority is 3, so it can still only tolerate 1 member failing. Increases in fault tolerance only occur when the next odd numbered member of a replica set is added [21].

For production systems, a standard deployment is a 3 Replica Set [21]. A 3 replica set provides 3 copies of the data for redundancy and fault tolerance if 1 member of the set were to fail. In a situation where availability was of higher concern, a 5 replica set would provide 4 copies of the data for redundancy and fault tolerance if 2 members of the set were to fail.

In our automated deployment and benchmarking process, the degree of replication for Shards is controlled by the sixth parameter to the main project.py script. For example, specifying

1 for will create a replica set per shard with only one copy of data (essentially no replication, although technically we create a one member replica set), specifying 3 will cause a replica set of with three copies to be created, and so on.

6.6. MongoDB Versions

The most current version of MongoDB is version 3.4, which was released on November 29, 2016. Based on an input parameter, our deployment will install either version 3.4 or version 3.2, the prior version of MongoDB. Many enhancements were made for version 3.4 impacting Sharded Clusters, Replica Sets, Aggregation, Indexes, Views, Security, Tools, and Platform Support. The complete list of 3.4 features and enhancements can be found in the Release Notes [22].

In our automated deployment and benchmarking process, the version of MongoDB installed is controlled by the second parameter to the main project.py script. Specifying 34 will install version 3.4. Specifying 32 will install 3.2. These versions were selected as they are the two most recent major versions of MongoDB and because they are the only two compatible with Ubuntu 16.04 LTS (Xenial Xerus).

6.7. Security

There are two levels of security to consider in a sharded MongoDB deployment: internal and external authentication.

In our deployment the various MongoDB components (config servers, mongos instances, shards, and replicas) all reside on separate Virtual Machines. These machines must be able to communicate with each other. Two steps were necessary to enable this internal authentication. First, the ports (27017, 27018, 27019, 28017) used by MongoDB needed to be opened for communication. This was accomplished by adding appropriate security group rules to the clouds through Cloudmesh client. Second, MongoDB requires the internal authentication to be done by either keyfiles or x.509 certificates [23]. In our deployment, authentication is done by keyfiles. A new keyfile is automatically created for each deployment and distributed to all of the virtual machines on the selected cloud.

For external authentication, the three users are created. The user *admin* is created with the role of *userAdminAnyDatabase*. *admin* performs administrative functions such as creating the other users. The user *cluster_admin_user* is created with the role *clusterAdmin*. *cluster_admin_user* user performs sharding functions such as sharding the collection and checking its data distribution. *user1* is a standard user with readWrite permissions. *user1* performs the benchmarking tests and other functions not requiring administrative privileges.

7. DEPLOYMENT

The automated process fully deploys a sharded MongoDB environment with the Cloud, MongoDB version, number of Config Servers, Mongos Instances, Shards, and degree of Replication specified as input parameters.

7.1. Computing Resources

In all cases, virtual machines are deployed with the Ubuntu 16.04 LTS (Xenial Xerus) operating system. On Openstack the flavor or the machine determines the amount of computing resources (CPU, memory, storage) allocated to it. In our testing, m1.small was used as the flavor for Chameleon Cloud and FutureSystems, while m1.tiny was used on Jetstream. Jetstream has more resources allocated to each flavor than Chameleon and FutureSystems, which are similar. In order to perform similar tests on each cloud, flavors with identical CPU and memory were selected. Table 3 shows the comparative resources of the flavors used in our testing. While storage is lower on Jetstream, it is sufficient for out tests and should not significantly impact performance.

Table 3. Computing Resources

Cloud	Flavor	VCPU	RAM	Size
Chameleon	m1.small	1	2	20
FutureSystems	m1.small	1	2	20
Jetstream	m1.tiny	1	2	8

7.2. Deployment Process

Several programs are involved in the deployment. A high level overview of each is provided.

7.2.1. project.py

The deployment process is invoked by running the deploy function of project.py, passing six required parameters: cloud (chameleon, jetstream, or kilo), version (32 or 34 for version 3.2 or 3.4), size of the config server replica set (a number, 1 or greater), number of mongos routers (a number, 1 or greater), number of data shards (a number, 1 or greater), and size of data shard replica sets.

Project.py calls a bash script, deploy.sh, which runs two bash shell scripts to accomplish the deployment: cluster.sh and ansible.sh.

7.2.2. cluster.sh

Cluster.sh does the work of creating the cluster in the specified cloud environment. First, it creates a keyfile needed for secure access between the nodes, and uses Cloudmesh secgroup commands to builds and uploads a new security group with the ports necessary for MongoDB (27017, 27018, 27019, and 28017) accessible. Next, it uses Cloudmesh client cluster commands to launch the appropriate number of virtual machines in the desired cloud. Then, it builds a file, inventory.txt, with sections for each MongoDB componenent (Config Servers, Mongos Instances, and Shard Replica Sets), allocating the correct number of IP addresses to each. Finally, cluster.sh builds a few complex commands that will need to be run later in the process by ansible.

7.2.3. ansible.sh

After the virtual machines have been created by cluster.sh, ansible.sh used the inventory.txt file to execute Ansible playbooks on the appropriate virtual machines.

1. install-python.yaml - Installs Python, if not installed. This script was necessary because the Ubuntu Zenial image on FutureSystems does not have Python installed. Python is required for Ansible.

2. mongo-install32.yaml - Using apt_key and apt_repository, installs the packages for version 3.2 of MongoDB on all virtual machines.

3. mongo-install34.yaml - Using apt_key and apt_repository, installs the packages for version 3.4 of MongoDB on all virtual machines.

4. add-mongo-key.yaml - Uploads the key file created in cluster.sh to all virtual machines.

5. mongo-config.yaml - On Config Servers only, stops the mongod service and uses a template file to start the mongod process for a Config Server.

6. mongo-config2.yaml - On only one Config Server, uses a template file to initiate the primary Config Server.

7. mongo-mongos.yaml - On Mongos Instances only, stops the mongod service and uses a template file to start the mongos process for a Mongos instance.

8. mongo-users.yaml - On only one Mongos Instance, create several users needed in later steps.

9. mongo-shard.yaml - On Shards only, stops the mongod service and uses a template file to start the mongod process for a Shard.

10. mongo-shard2.yaml - On the primary Shard in each Replica Set, uses a file built in cluster.sh to initiate the Shards.

11. add-shards.yaml - On one Mongos instance, uses a file built in cluster.sh to add all of the Shards.

12. create-sharded-collection.yaml - Uploads several files to one Mongos instance that will be need for benchmarking and shard the collection (benchmarking setup, not included in deployment times).

13. getdata.yaml - Downloads and unarchivse the pitches data from an AWS S3 directory. Also, create a smaller version for testing (benchmarking setup, not included in deployment times).

The kill function in project.py will delete and deallocate the last existing cluster on the cloud to clean up after the test is complete.

7.3. Deployment Timing

The configuration parameters and cluster and Ansible deployment times are captured in a file for each deployment (benchmarking timings are later captured as well). Total run time for a few interesting configurations are shown in Table 4.

Deployment A shows a simple deployment with only one of each component being created. This deployment may only be suitable for a development or test environment. Deployment A completed in 330 seconds.

Deployment B shows a more complex deployment with production like replication factors for Config Servers and Shards and an additional Mongos instance. This deployment may be suitable for a production environment as it has greater fault tolerance and redundancy. Deployment B took 1059 seconds to deploy.

Deployment C shows a deployment focused on high performance. It has a high number of shards, nine, but no fault tolerance or redundancy. The deployment may be suitable where performance needs are high and availability is less critical. Deployment C finished in 719 seconds.

The total number of virtual machines is highly correlated with deployment time as booting the machines and installing the software, tasks that occur for all nodes, take the most time. The additional steps to configure Config Servers, Mongos Instances, Replicas, and Shards run in relatively similar times, so the specific type of component created has little impact on the deployment time. For example, holding all other deployment variables at 1, a deployment with five Config Servers took 534 seconds, one with five Mongos Instances took 556 seconds, one with five Shards took 607 seconds, and one with a five Shard Replica set took 524 seconds. There is small extra overhead to starting additional data shards, but a strong correlation exists for total nodes to runtime for all configurations. Deployment times for version 3.4 were very similar to version 3.2.

Table 4. Deployment Times on Chameleon Cloud in Seconds

	Config	Mongos	Shards	Replicas	Seconds
A	1	1	1	1	*330*
B	3	2	3	3	*1059*
C	1	1	9	1	*719*

Table 5 shows this empirically, as it takes a very similar time to launch configurations with the same total number of nodes, but extremely different mixed of Config Servers, Mongos Instances, Replicas, and Shards.

The total number of nodes in a deployment can be calculated by the following equation involving the parameters to the deployment script.

c + m + (s * r) = total nodes

Table 5. Deployment Times on Chameleon Cloud in Seconds

Config Servers -c	Mongos -m	Shards -s	Replicas -r	Time in Seconds
5	1	1	1	*534*
1	*5*	1	1	*556*
1	1	*5*	1	*607*
1	1	1	*5*	*524*

Due to Chameleon Cloud having the most reliable and consistent performance of the three clouds, performance numbers are presented only for selected runs on Chameleon. While Chameleon has the best performance of the three clouds, these numbers are proportionately representative of deployment timings on Jetstream and FutureSystems.

8. BENCHMARKING

After the sharded MongoDB instance has been fully deployed, a benchmarking process is run to assess performance of the configuration. This process has also been fully automated. It is invoked by running the benchmark function of project.py and passing either the parameter large (for a full benchmark test) or small for a small test.

8.1. Data Set

The data set used in the benchmarking testing and analysis was Major League Baseball PITCHf/x data obtained by using the program Baseball on a Stick (BBOS) [24]. BBOS is a python program created by *willkoky* on github which extracts data from mlb.com and loads it into a MySQL database. While it would

be possible to convert this program to populate the MongoDB database directly, collecting all of the data is a time consuming process. Therefore, the data was captured locally to the default MySQL database and then extracted to a CSV file. This file contains 5,508,014 rows and 61 columns. It is 1,588,996,075 bytes in size uncompressed.

8.2. Methodology

There are several goals of the benchmarking process. The primary benchmarking goal of the project is to assess the impact of sharding on performance in MongoDB. Since replication was also built into the deployment process, a secondary goal was to assess the impact of replica sets on performance. A third goal is to assess performance of MongoDB version 3.4 versus version 3.2, specifically for various shard configuration. A final objective is to assess the relative performance of the Chameleon, FutureSystems, and Jetstream cloud computing environments.

The benchmarking tests are design to assess performance in three situations: Reads, Writes, and MapReduce operations.

8.2.1. Impact on Reads

To access the impact of different configurations on writes, we use MongoDB's mongoimport command. Mongoimport is a command line tool capable of loading JSON, CSV, or TSV files [25]. In this case, we load a CSV file to the pitches collections in the mlb database.

8.2.2. Impact on Writes

To assess the impact of different configurations on reads, we use MongoDB's find command. We read the data previously loaded by the mongoimport command to the pitches collection. The find command retrieves documents that meet a specified criteria. In this case, we search for pitches with a speed over 100 mph, a relatively rare event in baseball. To limit the information sent back over the network, we only return a count of these events. 3,632 is the count returned of 5,508,014 total documents. The column we search on does not have an index, as the goal is to test the impact of sharding on a long running query.

8.2.3. Impact on MapReduce

To assess the performance of MongoDB version, sharding, and replication on reads, a simple MapReduce operation was written against the pitches table to get the average speed of pitches that were stikes versus those that were not strikes [26] [27].

8.3. Benchmarking Process

The benchmarking process is invoked by running the benchmark function of the script project.py with the large parameter. Results for each test are automatically captured in file benchmark_datetime.csv. This file included the configuration the test was run under (cloud, MongoDB version, config server replication factor, mongos instances, number of shards, and shard replication factor) along with the run times of the find, mongoimport, and MapReduce commands. After all tests were run, a shell script, combine_benchdeploy.sh combines all files into one file, benchmark_combined.csv.

The graphical depictions of the test results shown in the next section were created by running python programs to average the run times across the shard, replication, and version configurations shown. For consistency, config server replication and mongos instances were both kept at one for all benchmarking tests. Additionally, replication was kept at one for sharding and version tests and sharding at one for replication tests. This methodologies allows us to isolate the variable we are assessing.

To setup for the test, a compressed version of file has been stored in an Amazon Web Services S3 directory. This file is prestaged on a Mongos instance during the deployment (but excluded from the run time) and is loaded it to a collection named *pitches* in MongoDB using mongoimport before running the find and MapReduce commands.

Before the benchmarking process can be run, a sharded collection must be created and sharded. This was also done via Ansible during the deployment in preparation for benchmarking. For benchmarking rerunability, the benchmarking process also deletes any data from the pitches collection that may have been loaded prior to running Mongoimport.

The shard key for the pitches table is set to pitchID. PitchID is a unique key to each pitch document. Selecting pitchID as the shard key should cause the data to be reasonably evenly distributed amound the shards. Data distribution will be analyzed in a subsequent section.

8.4. Data Distribution

To explore how data was allocated among the shards, a function called distribution was built into project.py. This function runs the getShardDistribution() command, which reports on how data and documents are distributed among shards [28]. Tables 6 and 7 show the results of tests with one, three, and five shards in version 3.2 and 3.4 of MongoDB. The results clearly show the data is well distributed, although interestingly, in all cases there is some minor skew toward the first shard having the most data. These results clearly show that data distribution is similar in both versions of MongoDB.

Table 6. Data Distribution among Shards - Version 3.2

	1	2	3	4	5
1	100				
3	35.84	32.18	31.96		
5	23.04	19.27	19.40	19.38	18.89

Table 7. Data Distribution among Shards - Version 3.4

	1	2	3	4	5
1	100				
3	36.23	31.82	35.75		
5	22.26	19.67	19.42	19.37	19.26

8.5. Benchmarking Analysis

8.5.1. Cloud Analysis

Chameleon Cloud was significantly more stable and reliable than FutureSystems and Jetstream Clouds for our testing. Chameleon yields (some functions on Jetstream also perform well) the fastest and most consistent results with very few errors. FutureSystem performance was the poorest with respect to run time. Environmental errors were frequent, but tests could still

be completed with moderate numbers of virtual machines. JetStream performance was good, but the environment was very unstable. Due to resource limitations and frequent errors, it was difficult to run high volume tests. For these reason, higher levels of sharding and replication were tested on Chameleon Cloud and FutureSystems, as detailed in the subsequent sections. Additionally, Chameleon was chosen as the enviroment to test MongoDB version 3.4 versus 3.2, due to its stability.

8.5.2. Impact of Sharding on Reads

Figure 2 depicts the impact on performance of various numbers of shards on a find command in Chameleon, FutureSystems, and Jetstream Clouds. All three clouds show a strong overall decline in run time as the number of shards increases, which shows the positive impact of sharding on performance. For example, while the run time for one shard on Chameleon and FutureSystems is over 25 seconds (well over for FutureSystems), the time drops to around five seconds for seven shards for both. This is a significant gain in performance.

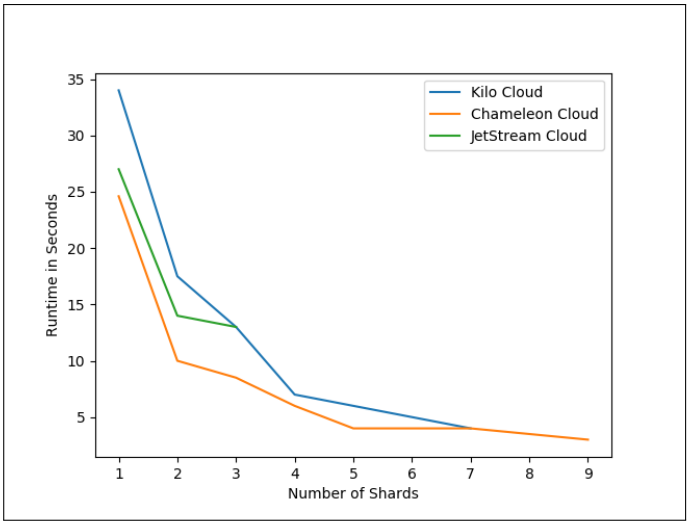


Fig. 2. Find Command - Sharding Test

For small numbers of shards, performance gains are almost exact proportion to the number of shards added. Two shards yields approximately half the run time of one shard. Three shards yields around one third the run time of one shard. From three to six chards, we still see still see significant improvement, but incrementally less than for the first three shards. After six shards, we see only slight performance gains.

From the closeness of the Chameleon and Kilo lines we can see that performance in the two clouds is very similar for this find test. This is an interesting observation as for both deployment and mongoimport, performance was much better on Chameleon Cloud than Kilo. One difference from the mongoimport test is that much less data is being sent over the network. Network speeds could be a factor in this discrepancy. Jetstream performance is better than or equal to FutureSystems, but worse than Chameleon for all shard counts tested.

Figure 2 can be recreated by running the program benchmark_shards_find.py passing the file benchmark_combined.csv as a parameter. It plots the average run time for each configuration as shown using matplotlib. This report is run automatically by the report function of project.py.

8.5.3. Impact of Sharding on Writes

Figure 3 depicts the impact on performance of various numbers of shards on a mongoimport command in the three clouds. For all clouds, run time of the mongoimport command in our tests does not appear to be impacted by the number of shards. Since the same amount of data is written with more computing resources available when there are more shards, we might expect to see a performance gain. However, there are possible explanations for performance not improving. First, the mongoimport command may not write data in parallel. This is not indicated in the documentation, but it seems likely that it reads the file serially. Second, resources on the server the data is written to may not be the bottleneck in the write process. Other resources like the network time seem more likely to be the bottleneck. Since we are always going over the network from the mongos instance to a data shard, regardless of the number of shards, a bottleneck in the network would impact all shard configurations equally.

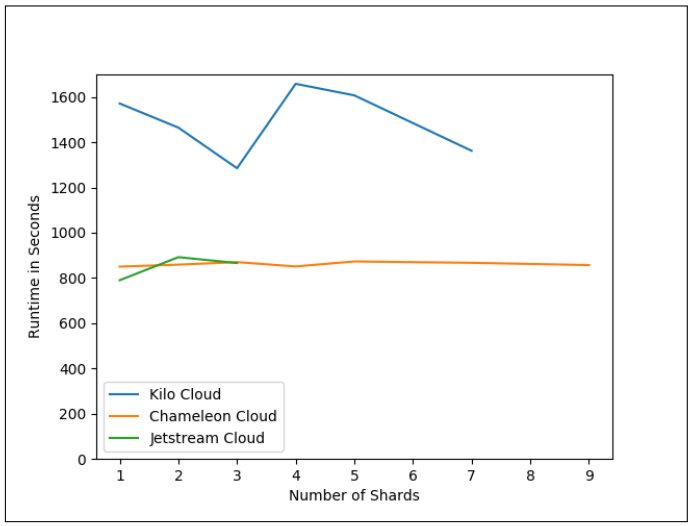


Fig. 3. Mongoimport Command - Sharding Test

While sharding did not benefit a single threaded mongoimport command, it is likely it would benefit other heavy write operations, particularly coming through multiple mongos instances. In a non-sharded environment, this would lead to a heavy load on the single data shard. In a sharded environment, the load on each shard would drop as the number of shards increased.

While performance on Chameleon and FutureSystems was very similar for the find command, performance of the mongoimport command was significantly better on Chameleon than on Kilo. We see approximately 50% better performance on both Chameleon and Jetstream Clouds compared to FutureSystems. Jetstream and Chameleon have nearly identical performance for the import test.

Figure 3 can be recreated by running the program benchmark_shards_import.py passing the file benchmark_combined.csv as a parameter. It plots the average run time for each configuration as shown using matplotlib. This report is run automatically by the report function of project.py.

8.5.4. Impact of Sharding on MapReduce

Figure 4 shows the performance of MapReduce across varous sharding configurations on our three clouds. These results are relatively similar to the find results. While results are incon-

sistent, likely due to environmental issues, all clouds show an overall decrease in processing time with addition of shards. Relative to Mongoimport performance, performance is more similar across the three clouds for MapReduce.

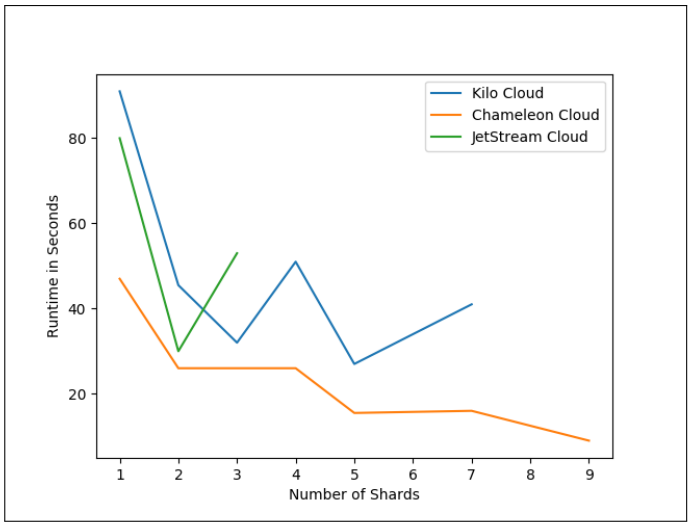


Fig. 4. MapReduce - Sharding Test

Figure 4 can be recreated by running the program benchmark_shards_mapreduce.py passing the file benchmark_combined.csv as a parameter. It plots the average run time for each configuration as shown using matplotlib. This report is run automatically by the report function of project.py.

8.5.5. Impact of Replication on Reads

Figure 5 depicts the impact on performance of various numbers of replicas on a find command in Chameleon, FutureSystems, and Jetstream Clouds. While it is clear that replication does not have the same performance impact on the find command that sharding does, it appears that there may be a performance penalty to high degrees of replication, particularly on Chameleon Cloud. Replica sets up to three did not show this penalty, but four through seven replica sets caused a significant impact to run times on Chameleon Cloud. Performance on FutureSystem showed no penalty up to five replicas, but Jetstream performance worsened with replication even at low levels. Without a larger sample size it cannot be determined if this is a real effect or random variation. We would not expect replication to have a significant performance degradation on the the find command since it only needs to read one copy of the data, but the increased communication necessary in a replica set may cause a small performance penalty in some cases. Another possibility is that this is a timing issue and that there was more work being done on the clouds when the higher replication tests were run.

Similarly to other tests, performance on Chameleon was best for the majority of the test runs in the find replication test.

Figure 5 can be recreated by running the program benchmark_replicas_find.py passing the file benchmark_combined.csv as a parameter. It plots the average run time for each configuration as shown using matplotlib. This report is run automatically by the report function of project.py.

8.5.6. Impact of Replication on Writes

Figure 6 depicts the impact on performance of various numbers of replicas on a mongoimport command on our three Clouds.

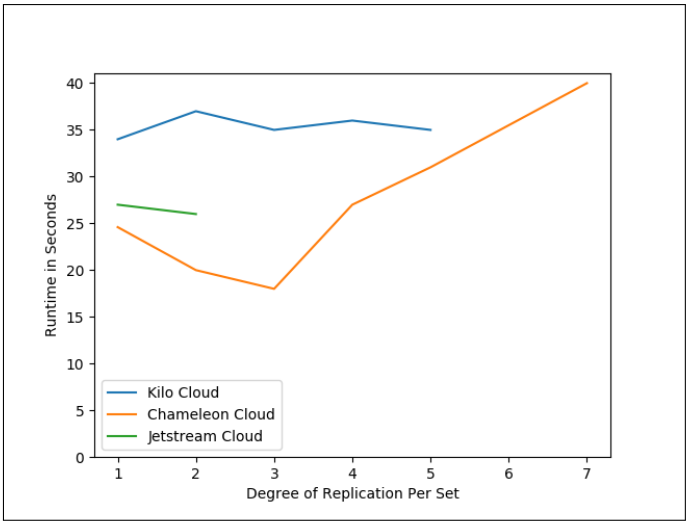


Fig. 5. Find Command - Replication Test

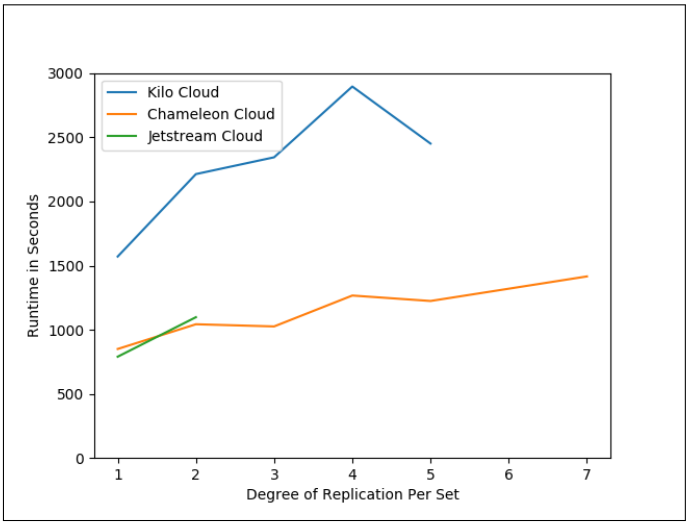


Fig. 6. Mongoimport Command - Replication Test

The test results show negative impact on the mongoimport command for increase levels of replication. On Chameleon, a replication factor of two leads to approximately a 20% performance penalty and a replication factor of seven leads to around 40% worse performance. The results scales similarly for the lower replication levels tested on FutureSystems and Jetstream. Given that an extra copy of data is written with each increase in the replication factor, this performance hit is expected.

Performance on Jetstream and Chameleon was nearly identical for the tests that were successfully run. FutureSystems import performance was by far the worst of the three clouds for this test.

Figure 6 can be recreated by running the program benchmark_shards_import.py passing the file benchmark_combined.csv as a parameter. It plots the average run time for each configuration as shown using matplotlib. This report is run automatically by the report function of project.py.

8.5.7. Impact of Replication on MapReduce

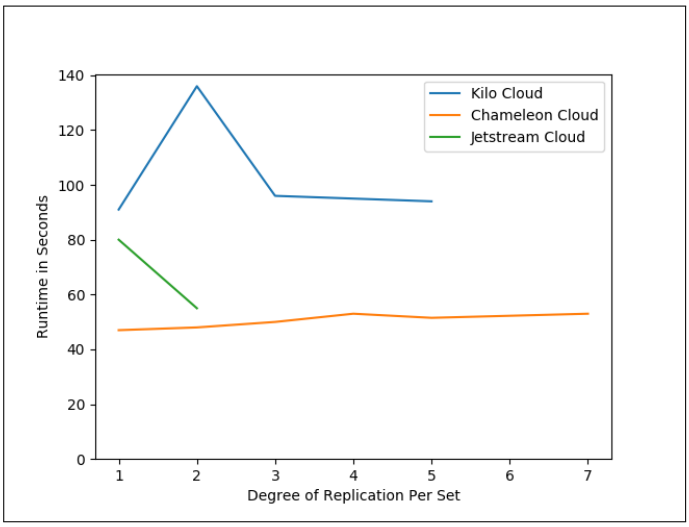


Fig. 7. MapReduce - Replication Test

As shown in Figure 7, replication appears to have no impact on MapReduce operations. While there are variations in FutureSystems and Jetstream performance for different numbers of replicas, they do not follow a consistent pattern and appear to be caused by environmental issues. This is an interesting result as increased levels of replication came with a performance penalty for the find commmand, which also reads data.

As with several other tests, Chameleon MapReduce performance was the best, followed by Jetstream, with FutureSystems again being the worst.

Figure 7 can be recreated by running the program benchmark_shards_import.py passing the file benchmark_combined.csv as a parameter. It plots the average run time for each configuration as shown using matplotlib. This report is run automatically by the report function of project.py.

8.5.8. Impact of Version and Sharding on Reads

Figure 8 shows the MongoDB version 3.4 and 3.2 find performance on Chameleon Cloud. Results are mixed, with version 3.2 having the best performance for one shard, version 3.4 having significantly better performance between two and eight shards, with performance equal at nine. Despite the mixed results at high and low levels, version 3.4 is the clear winner in this test.

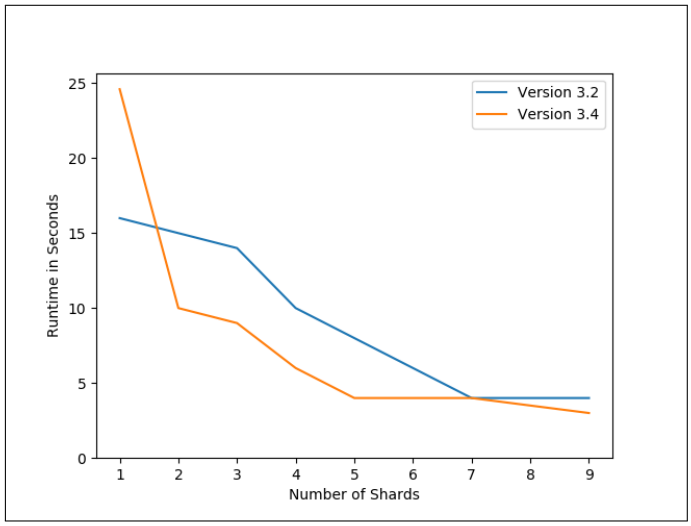


Fig. 8. Find Command - Version 3.2 vs 3.4

Figure 8 can be recreated by running the program benchmark_verson_find.py passing the file benchmark_combined.csv as a parameter. It plots the average run time for each configuration as shown using matplotlib. This report is run automatically by the report function of project.py.

8.5.9. Impact of Version and Sharding on Writes

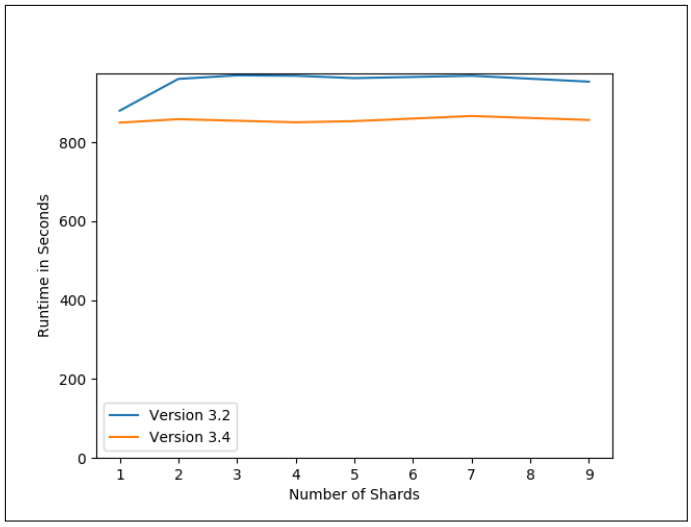


Fig. 9. Mongoimport Command - Version 3.2 vs 3.4

Figure 9 shows the MongoDB version 3.4 and 3.2 Mongoimport performance on Chameleon Cloud. For Mongoimport, version 3.4 performs better at all sharding levels, with nearly a 20% reduction in import time compared to version 3.2. Mongoimport performance appears to have been significantly improved in version 3.4.

Figure 9 can be recreated by running the program benchmark_verson_find.py passing the file benchmark_combined.csv as a parameter. It plots the average run time for each configuration as shown using matplotlib. This report is run automatically by the report function of project.py.

8.5.10. Impact of Version and Sharding on MapReduce

Figure 10 shows the MongoDB version 3.4 and 3.2 Mongoimport performance on Chameleon Cloud. Results are nearly identical for one, two, and nine shards, but surprisingly version 3.2 shows approximately 50% better performance for between three and seven shards. Given the lack of consistency across shard levels, it is possible this is environmental in a shared cloud environment, but it may be a valid finding that there is performance degredation in version 3.4 MapReduce operations.

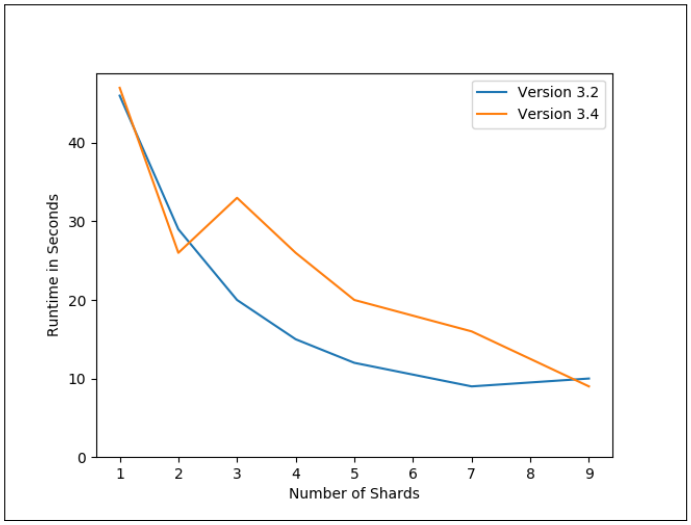


Fig. 10. MapReduce - Version 3.2 vs 3.4

Figure 10 can be recreated by running the program benchmark_verson_find.py passing the file benchmark_combined.csv as a parameter. It plots the average run time for each configuration as shown using matplotlib. This report is run automatically by the report function of project.py.

9. SUMMARY

We have created, tested, and demonstrated a fully automated program to configure and deploy a sharded MongoDB cluster to three cloud environments: Chameleon, Jetstream, and FutureSystems. Using a combination of Python, Bash, and Cloudmesh Client, the a cluster is dynamically deployed with a selected number of Config Server Replicas, Mongos Routers, Shards, and Shard Replicas and either MongoDB version 3.4 or 3.2. Functions also exist for terminating the environment, reporting on data distribution, benchmarking, and reporting on performance testing.

An automated benchmarking process to show the impact of well distributed data across shards of a large data set has been run for various configurations. The impact of MongoDB version 3.4 versus 3.2, Sharding, and Replication on performance have been assessed. Testing showed performance and stability on Chameleon Cloud to be the best of our three cloud environments. A key finding is that read performance, typically a high priority for noSQL data stores and Big Data operations, increases significantly as shards are added. Testing also showed that a predictable performance penalty is associated with replication. Our comparison of version 3.4 and 3.2 showed improved Mongoimport and find performance in version 3.4, but slightly worse MapReduce performance.

ACKNOWLEDGEMENTS

The author thanks Gregor von Laszewski and Tony Liu for technical their technical support in Thursday night office hours, particularly with Cloudmesh Client.

REFERENCES

[1] Wikipedia, “Openstack,” web page, Nov. 2016, 11/9/2016. [Online]. Available: https://en.wikipedia.org/wiki/OpenStack
[2] OpenStack, “Openstack community q&a,” web page, Nov. 2016. [Online]. Available: https://www.openstack.org/projects/openstack-faq/
[3] Chameleon, “About,” web page, Nov. 2016. [Online]. Available: https://www.chameleoncloud.org/about/chameleon/
[4] FutureSystems, “About,” web page, Sep. 2014. [Online]. Available: https://portal.futuresystems.org/about
[5] Indiana University, “Jetstream partners and collaborators,” web page, 2016. [Online]. Available: http://www.jetstream-cloud.org/partners.php
[6] Indiana University, “Indiana university’s leadership role,” web page, 2016. [Online]. Available: http://www.jetstream-cloud.org/leadership.php
[7] Chameleon, “Hardware description,” web page, Nov. 2016. [Online]. Available: https://www.chameleoncloud.org/about/hardware-description/
[8] G. von Laszewski, “Hardware,” web page, Jan. 2013. [Online]. Available: http://futuregrid.github.io/manual/hardware.html
[9] JetStream, “System specs,” web page, Apr. 2016. [Online]. Available: http://www.jetstream-cloud.org/leadership.php
[10] Python Software Foundation, “cmd — support for line-oriented command interpreters¶,” web page, 4/20/2017. [Online]. Available: https://docs.python.org/2/library/cmd.html
[11] Wikipedia, “Ansible (software),” web page, Apr. 2017, 4/20/2017. [Online]. Available: https://en.wikipedia.org/wiki/Ansible_(software)
[12] G. von Laszewski, “Cloudmesh client toolkit,” web page, Sep. 2016. [Online]. Available: http://cloudmesh.github.io/client/
[13] Wikipedia, “Mongo,” web page, Sep. 2016. [Online]. Available: https://en.wikipedia.org/wiki/MongoDB
[14] DB-Engines, “Bdb-engines ranking,” web page, Nov. 2016. [Online]. Available: http://db-engines.com/en/ranking
[15] MongoDB, “Sharded cluster components,” web page, Sep. 2016. [Online]. Available: https://docs.mongodb.com/v3.2/core/sharded-cluster-components/
[16] MongoDB, “Config servers,” web page, Nov. 2016. [Online]. Available: https://docs.mongodb.com/manual/core/sharded-cluster-config-servers/
[17] MongoDB, “Mongos,” web page, Nov. 2016. [Online]. Available: https://docs.mongodb.com/manual/reference/program/mongos/
[18] MongoDB, “Sharding,” web page, Sep. 2016. [Online]. Available: https://docs.mongodb.com/manual/sharding/
[19] MongoDB, “Shard keys,” web page, Sep. 2016. [Online]. Available: https://docs.mongodb.com/manual/core/sharding-shard-key/
[20] MongoDB, “Replication,” web page, Sep. 2016. [Online]. Available: https://docs.mongodb.com/manual/replication/
[21] MongoDB, “Replica set deployment architectures,” web page, Nov. 2016. [Online]. Available: https://docs.mongodb.com/v3.2/core/replica-set-architectures/
[22] MongoDB, “Release notes,” web page. [Online]. Available: https://docs.mongodb.com/manual/release-notes/3.4/
[23] MongoDB, “Enable internal authentication,” web page, Nov. 2016. [Online]. Available: https://docs.mongodb.com/v3.0/tutorial/enable-internal-authentication/
[24] willkoky, “Baseball on a stick,” web page, Apr. 2016. [Online]. Available: https://sourceforge.net/projects/baseballonastic/
[25] MongoDB, “mongoimport,” web page, Sep. 2016. [Online]. Available: https://docs.mongodb.com/manual/reference/program/mongoimport/
[26] MongoDB, “Mapreduce examples,” web page. [Online]. Available: https://docs.mongodb.com/manual/tutorial/map-reduce-examples/
[27] MongoDB, “Mapreduce,” web page. [Online]. Available: https://docs.mongodb.com/manual/core/map-reduce/
[28] MongoDB, “db.collection.getsharddistribution(),” web page, Nov.

2016. [Online]. Available: https://docs.mongodb.com/manual/reference/method/db.collection.getShardDistribution/
[29] C. Duffy, "How do i test if a variable is a number in bash?" web page, Apr. 2009. [Online]. Available: http://stackoverflow.com/questions/806906/how-do-i-test-if-a-variable-is-a-number-in-bash
[30] mklement0, "How can i remove the last character of a file in unix?" web page, Dec. 2014. [Online]. Available: http://stackoverflow.com/questions/27305177/how-can-i-remove-the-last-character-of-a-file-in-unix
[31] MongoDB, "Configuration file options," web page, Oct. 2016. [Online]. Available: http://docs.mongodb.org/manual/reference/configuration-options/
[32] steeldriver, "Passing named arguments to shell scripts," web page, May 2014. [Online]. Available: http://unix.stackexchange.com/questions/129391/passing-named-arguments-to-shell-scripts
[33] G. von Laszewski, "Advanced command usage," web page, Jan. 2015. [Online]. Available: http://cloudmesh.github.io/client/commands/command_advanced.html
[34] jezrael, "Converting a pandas groupby object to dataframe," web page, Aug. 2015. [Online]. Available: http://stackoverflow.com/questions/10373660/converting-a-pandas-groupby-object-to-dataframe
[35] silvio, "How to set 'auto' for upper limit, but keep a fixed lower limit with matplotlib.pyplot," web page, Sep. 2015. [Online]. Available: http://stackoverflow.com/questions/11744990
[36] EdChum, "Pandas dataframe to numpy array valueerror," web page, Aug. 2015. [Online]. Available: http://stackoverflow.com/questions/31791476/pandas-dataframe-to-numpy-array-valueerror
[37] leucos, "How to create a directory using ansible?" web page, Apr. 2014. [Online]. Available: http://stackoverflow.com/questions/22844905/how-to-create-a-directory-using-ansible
[38] El Ruso, "How to install mongodb with ansible?" web page, Feb. 2017. [Online]. Available: http://stackoverflow.com/questions/37568848/how-to-install-mongodb-with-ansible
[39] Ansible, Inc., "copy - copies files to remote locations," web page, Apr. 2017. [Online]. Available: http://docs.ansible.com/ansible/copy_module.html
[40] Lorin Hochstein, "Run command on the ansible host," web page, Sep. 2013. [Online]. Available: http://stackoverflow.com/questions/18900236/run-command-on-the-ansible-host
[41] MongoDB, "Install mongodb community edition on ubuntu," web page. [Online]. Available: https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/
[42] gwillem, "Get ansible to work on bare ubuntu 16.04 without python 2.7," web page. [Online]. Available: https://gist.github.com/gwillem/4ba393dceb55e5ae276a87300f6b8e6f

AUTHOR BIOGRAPHIES

Mark McCombe received his B.S. (Business Administration/Finance) and M.S. (Computer Information Systems) from Boston University. He is currently studying Data Science at Indiana University Bloomington.

A. CODE REFERENCES

References used in deployment, benchmarking, visualization programs are formally documented here as well as noted in a comment in the code [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [39] [40] [41] [42].

B. EXECUTION INSTRUCTIONS

The project should be run on an Ubuntu 16.04 LTS (Xenial Xerus) machine. The required modules for the project can be installed in a virtualenv virtual environment using the file project/S17-IO-3012/code/requirements.txt.

The main script, project/S17-IO-3012/code/bin/project.py, can be run to execute all functionality. Project.py functions (deploy, kill, benchmark, report, distribution) are described in help, but sample instructions are provided below for each function.

python project.py has four functions.

B.1. deploy

Runs a deployment. Takes 6 parameters:

1. Cloud - chameleon, jetstream, or kilo (futuresystems)
2. MongoDB Version - 34 for version 3.4, 32 for version 3.2
3. Config Server Replication Size - a number
4. Mongos Router Instances - a number
5. Shard Count - a number
6. Shard Replication Size - a number

Simple example:
deploy chameleon 34 1 1 1 1
More complex examples:
deploy chameleon 32 3 2 3 3
deploy kilo 34 2 2 1 1

B.2. kill

- Deletes and undefines the current cluster. No parameters.

B.3. benchmark

Runs a benchmark Mongoimport, find, and MapReduce and logs timings. Takes one required parameter - *large* or *small* (for testing purposes).

B.4. report

Regenerates PNG files in the code/report/directory based on current benchmarks

B.5. distribution

Shows the data distribution of the current configuration. For meaningful results, must be run after benchmark.

C. DIRECTORY STRUCTURE

The project/S17-IO-3012/code contains several directories.

C.1. benchmark/

Contains all benchmark timing logs

C.2. bin/

Contains all Bash and Python code

C.3. configfiles/
Contains all configuration file templates

C.4. deploy/
Contains all deployment timing logs

C.5. json/
Contains all json documents

C.6. playbooks/
Contains all Ansible YAML files

C.7. report/
Contains all reports in PNG format

C.8. stdlist/
Contains all bash script output logs

C.9. work/
Contains temporary work files

Music Predictive Analysis Project based on Lyrics

LEONARD MWANGI[1,*]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*] Corresponding authors: lmwangi@iu.edu

project-01, May 4, 2017

Being certain that lyrics of your song will lead to the next greatest hit would boost confidence to a lot of amateur artists who are faced with fears of never making it thus never attempting to make good their creativity. With Machine Learning (ML) this can be a thing of the past, these artists would have the ability to let ML models determine the viability of their lyrics becoming the next hit based on history of other songs that have made it to top. Through training, the model can certainly determine the outcome of different songs which will be depicted in this project.

Keywords: Cloud, I524

https://github.com/lmundia/sp17-i524/tree/master/project/S17-IO-3013/report/report.pdf

1. INTRODUCTION

When faced with the decision to forward their song to a recording company, amateur artists find it daunting due to uncertainty of whether their song would be recorded and if it is if it will make them wealthy. Having ability to run the lyrics through a predictive analysis process that would determine the viability of the song making it would be a huge win and confidence booster to many artists. That prediction is achievable by use of machine learning and creating a model that takes already greatest his and trains it to determine what makes the song successful. This would be done by analyzing the lyrics, the locality and time of release.

In this project, we will utilize machine learning to help determine the viability of a song becoming the next greatest hit based on the lyrics, time of production, locality and the artist. The project will utilize the greatest hits of all time [1] to train a model which will then be used to analyze larger dataset of random songs [2] and provide an in-depth analysis of the next possible hit. The project will utilize a Hadoop cluster deployed on Chameleon Cloud using CloudMesh to accomplish this analysis.

2. ENVIRONMENT

To achieve the analysis, Cloudmesh was used in order to allow the end users ability to choose a cloud of their liking. Chameleon Cloud and Jetstream were used for initial testing but changing the default cloud on CloudMesh would allow deployment to other clouds.

2.1. Cloudmesh

Cloudmesh is an open source component aimed at delivering software-defined system. It's a collection of virtualized bare-metal infrastructure, networks, application, systems and platform software unifying different clouds as a Cloud Testbeds as a Service (CTaas) [1]. Cloudmesh was developed by Professor Gregor von Laszewski in collaboration with his team at Indiana University [2]. In this environment, cloudmesh is used to automate installation of a four-node cluster.

2.2. Ansible

Ansible is also an open source platform used in automation of deployments and tasks executions [3] . It has the ability to execute scripts to a remote system thus making it easy to administer and manage hosts farm. Ansible is written in Python thus it requires python to be installed in the remote host for successful executions.

2.2.1. Ansible Playboks

Deemed as the real strength of Ansible, a playbook is a collection of instructions that defines what Ansible will do when it connects to each host [4] . Playbooks are written in YAML an easy to read data serialization language [5] . In this project, Ansible is used to deploy the required components to successfully accomplished the deployment, configuration and analysis of the defined files.

2.3. Apache Spark

Apache Spark is an open-source data processing and analytics engine designed to handle large scale dataset. Spark was originally developed by UC Berkeley but has been adopted in the Apache software stack since 2013 [6]. Spark has the ability to run as part of Hadoop, Mesos, Standalone or in a cloud environment [7]. In this project, Spark is deployed as a standalone environment and primarily used to analyze music data to in order to

identify patterns and make predictions.

2.4. MongoDB

MongoDB is a NoSQL, open source database program aimed at hosting documents in JSON-like format. MongoDB is favorable in storing unstructured data since it does not follow the traditional RDBMS structure. Its performance is also paramount when coupling related unstructured data [8]. For this project, MongoDB become the ideal database application because the source data structure is unknow and it can span in many directions where one dataset doesn't resemble the next. Putting such data in a structured database program would be cumbersome.

2.5. Python

Python is used as the programming of choice in this project due to its extensibility and also as a requirement for Ansible to execute successfully.

3. DEPLOYMENT

Environment deployment is automated as mentioned earlier using Cloudmesh and Ansible scripts. The environment is deployed on a 4-node cluster with the following resources.

3.1. Resources Table

Table 1 resources.

Table 1. resources dedicated to the environment

Node	Role	Flavor
*Node*1	*Spark*	*m*1.*small*
*Node*2	*Mesos*	*m*1.*small*
*Node*3	*MongoDB*	*m*1.*medium*
*Node*4	*Analytics*	*m*1.*small*

3.2. Cloudmesh

Code snippet below generates 4 node cluster that will be used in the inventory file for the Ansible roles. In order to use Cloudmesh Cluster, the following package levels or higher must be installed

- Cloudmesh client 4.7.2
- Pip 8.1.2
- Python 2.7.12

Algorithm 1. cloudmesh code

```
\$ cm cluster define --count 4 --image CC-Ubuntu16.04
\$ cm cluster use cluster-001
\$ cm cluster allocate
\$ cm cluster nodes > inventory.txt
```

3.2.1. format inventory file

Define roles in the inventory file and also remove the resolved name. After formatting the inventory file, the server roles and their corresponding public IP address should look similar to the output below with the right IP address.

Algorithm 2. Inventory file

```
[spark\_server]
<ip address>

[mesos\_server]
<ip address>

[mongo\_server]
<ip address>

[data\_node]
<ip address>
```

3.3. Ansible

After configuring deployment of virtual machines and configuring the inventory.txt, Ansible playbooks are used to automate deployment of the required components to the nodes defined above. The following Ansible files are included: ansible.cfg – used to define default configurations for Ansible including role_path and remote user.

Algorithm 3. ansible config file

```
[defaults]
roles\_path = roles
host\_key_checking=false
callback\_whitelist = profile_tasks
remote\_user=cc
```

3.3.1. Playbooks

The following playbooks are utilized for deployment

1. predict.yml – main file for the playbooks, used to call all the other playbooks and define their locations. Role-based playbooks are contained in scripts directory of the environment.
2. spark.yml – used to define the host, roles location and execution of Spark installation playbook. The playbooks redirect the folder structure to /roles/ansible-role-spark.
    - Tasks directory - contains main.yml playbook that's used to define all the tasks for Spark installation.
    - Defaults directory – contains main.yml that defines the default requirements for Spark installation including the spark version and download site for the distribution.
3. mesos.yml – used to define the host, roles location and execution of MongoDB installation playbook. The playbooks redirect the folder structure to /roles/ansible-role-mesos.
    - tasks directory – contains main.yml playbook that checks the operating system to determine the flavor of mesos that would be deployed. The environment contains playbooks for Debian and Redhat flavors.
    - defaults directory – contains main.yml which defines the version of mesos that will be downloaded and installed.

4. mongo.yml – used to define the host, roles location and execution of MongoDB installation playbook. The playbooks redirect the folder structure to /roles/ansible-role-mongo.
    - tasks directory - contains main.yml playbook that's used to define all the tasks for MongoDB installation. The file also includes task to create a database and initial user for the Mongo environment.
    - defaults directory – contains main.yml that defines the default requirements for MongoDB installation including the MongoD version and download site for the distribution.
    - vars directory – contains main.yml that defines the variables for the environment like data path, admin account and password.
    - template directory – contains mongod_config.j2 file which is used to define MongoDB configurations.
5. data.yml - used to define the host, roles location and execution of dataset playbook. The playbooks redirect the folder structure to /roles/ansible-role-dataset.
    - tasks directory - contains main.yml playbook that's downloads the MillionSongs dataset and copies into to MongoDB. The file also contains task to create collection and load data into MongoDB by executing a remote file myfiles.py

4. CONCLUSION

Ability for amateur artists, artists and record labels to quick determine the viability of a hit is paramount to their success and missed chances due to inexperience, fear of unknows, bad song or acting when time is not ripe can be costly. Machine learning has the ability to change these outcomes, a well-trained model can help determine with high accuracy where the song will end up.

REFERENCES

[1] G. von Laszewski, "Cloudmesh for beginners," WebPage, 2015. [Online]. Available: https://cloudmesh.github.io/introduction_to_cloud_computing/class/lesson/iaas/cloudmesh.html#cloudmesh-for-beginners

[2] G. von Laszewski, F. Wang, H. Lee, H. Chen, and G. C. Fox, "Accessing multiple clouds with cloudmesh," Indiana University, School of Informatics and Computing, 919 E. 10th Street Bloomington IN 47408, U.S.A., techreport, Jun. 2014. [Online]. Available: https://www.researchgate.net/publication/266659258_Accessing_multiple_clouds_with_Cloudmesh

[3] networklore, "What is ansible?" Webpage, Apr. 2014. [Online]. Available: https://networklore.com/ansible/

[4] Red Hat, Inc., "Playbooks," Webpage, Apr. 2017. [Online]. Available: http://docs.ansible.com/ansible/playbooks.html

[5] O. Ben-Kiki, "Yaml ain't markup language (yaml™) version 1.2," Webpage, Oct. 2009. [Online]. Available: http://www.yaml.org/spec/1.2/spec.html

[6] A. Kattt, "Spark – lightning-fast cluster computing," Webpage, Nov. 2011. [Online]. Available: https://amplab.cs.berkeley.edu/projects/spark-lightning-fast-cluster-computing

[7] Apache Spark, "Lightning-fast cluster computing," Webpage. [Online]. Available: http://spark.apache.org/

[8] MongoDB Inc., "The mongodb 3.4 manual," Webpage. [Online]. Available: https://docs.mongodb.com/manual/

Deploying CouchDB Cluster

RIBKA RUFAEL[1,*,+]

[1] *School of Informatics and Computing, Bloomington, IN 47408, U.S.A.*
[*] *Corresponding authors: rrufael@umail.iu.edu*
[+] *HID: S17-IO-3016*

project-000, May 4, 2017

This project focuses on deployment of CouchDB Cluster using Ansible playbook on Ubuntu Chameleon Cloud VMs and benchmarking of the deployment.

Keywords: CouchDB

https://github.com/cloudmesh/classes/raw/master/docs/source/format/report/report.pdf

1. INTRODUCTION

CouchDB [1] is a no sql database management system under Apache. Data is stored as documents in CouchDB. In this project CouchDB cluster of one or more Chameleon cloud VMs is deployed using Ansible playbook and benchmarking is done to measure the time it took for deployement using a TBD benchmarking tool.

2. EXECUTION PLAN

This section depicts week by week execution plan for the project

2.1. Week 1

I was able to develop an initial Ansible script that will deploy CouchDB on Ubuntu 16.04 VM , upon successul installation the script will start CouchDB and then it will stop CouchDB on the remote VM. I booted Chameleon VM using Cloudmesh and then run Ansible playbook from my local machine. The tasks in my playbook run successfully.

2.2. Week 2

Run Ansible playbook to deploy CouchDB on Chameleon Cloud VM. Benchmark time it takes to deploy CouchDB on Chameleon Cloud VM

2.3. Week 3

Extend the Ansible script developed in Week 1 to deploy CouchDB into two Chameleon Cloud VMs.

2.4. Week 4

Run Ansible playbook to deploy CouchDB on two Chameleon Cloud VMs. Benchmark time it takes to deploy CouchDB on Chameleon Cloud VMs.

2.5. Week 5

Analysis of the benchmark results for deployment of CouchDB cluster. Document results in report and finalize report.

3. TECHNOLOGIES USED

- Ansible
- Cloudmesh
- Other technologies TBD

4. DEPLOYMENT

TBD

5. BENCHMARK RESULTS

TBD

6. CONCLUSION

TBD

ACKNOWLEDGEMENTS

TBD

REFERENCES

[1] Apache Software Foundation, “Technical Overview — Apache CouchDB 2.0 Documentation,” Web Page, Mar. 2017, accessed: 2017-03-11. [Online]. Available: http://docs.couchdb.org/en/2.0.0/intro/overview.html

Analysis of USGS Earthquake Data

NANDITA SATHE[1,*]

[1] *School of Informatics and Computing, Bloomington, IN 47408, U.S.A.*
[*] *Corresponding authors: nsathe@iu.edu*

project-001, May 4, 2017

US Geological Survey's (USGS) Earthquake Hazards Program monitor and report earthquakes, assess earthquake impacts and hazards, and research the causes and effects of earthquakes [1]. The geo-spatial data it collects is available for free. Big Data Analytics tools are used to analyze this data. Machine learning algorithms are used for advanced data analysis and earthquakes prediction.

Keywords: I524, geospatial, MongoDB, Plotly, K-means clustering, DBSCAN, Python, pymongo, USGS, Ansible, Cherrypy

Code: https://github.com/cloudmesh/sp17-i524/tree/master/project/S17-IO-3017/code
Report: https://github.com/cloudmesh/sp17-i524/tree/master/project/S17-IO-3017/report/report.pdf

1. INTRODUCTION

USGS collects volumes of geospatial data pertaining to earthquakes and makes it available for analysis. The project obtains a chunk of the data using GeoJSON web service provided by USGS. The data is saved locally in MongoDB database. Data is analysed using machine learning algorithms. The output is plotted (rendered) on web browser. A light weight web server is used to respond to the web requests. Project is capable of running in cloud environment. Deployment is automated using Ansible.

2. TECHNOLOGIES USED

Technologies used for development and deployment of this project are listed below.

1. **Cloudmesh** - Cloudmesh provides Cloud Testbeds as a Service (CTaaS). It is a platform where one can manage all cloud accounts and local files enabling one to copy them between services. Project uses cloudmesh to connect to various cloud environments.

2. **Ansible** - Ansible is an IT automation tool that automates cloud provisioning, configuration management, and application deployment. Once Ansible gets installed on a control node, which is an agentless architecture, it connects to a managed node through the default OpenSSH connection type. Ptoject uses ansible for one-click deployment.

3. **Python** - Python is an object oriented, light weight programming language. Python is primary programming language used in this project.

4. **Mongo-DB** - MongoDB is an unstructured (NOSQL) database, which uses document-oriented data model. It stores data in flexible JSON-like documents. The project uses Mongo-DB to store GeoJSON data of earthquakes locally.

5. **Plotly** - Plotly is data analytics and visualization tool. One can create interactive graphs using plotly. It provides graphing libraries for Python. The plotly is used in the project as a visualization tool.

6. **Scikit-learn** - Scikit-learn provides tools for data analysis and data mining. Scikit-learn provides a wide range of learning algorithms. Scikits are the names given to the modules for SciPy, a fundamental library for scientific computing. As these modules provide different learning algorithms, the library is named as sciki-learn. In this project data classification is done using scikit learn.

7. **Cherrypy** - CherryPy is an object-oriented web application framework. It is designed for rapid development of web applications by wrapping the HTTP protocol. It is WSGI (Web Server Gateway Interface) thread-pooled web server. Cherrypy is used as a web server in the application.

3. DESIGN

Figure 1 shows main components and data flow of the application. There are four major components in the application.

3.1. WebServer

Purpose of web server is to listen to the user's HTTP GET request, call appropriate python method at the backend, get the response from the python method, and send it to the web client as a response. The server listens at port 8081.

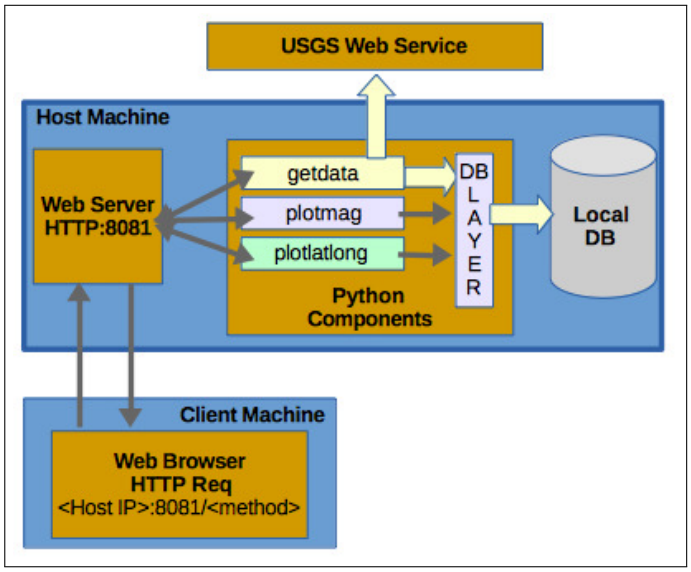


Fig. 1. Application Components

3.2. getusgsdata - getdata

getusgsdata component is responsible for fetching the data from USGS and storing it in local database. It uses dblayer common component to communicate with the local database. Running this component at least once is necessary as it downloads the data on which other components are dependent.

3.3. kmeansplot - plotmag

kmeansplot component implements the K-Means clustering algorithm and plots it using plotly library. It reads the Magnitude and Depth data from local database using dblayer common component. The algorithm classifies the data in 3 clusters based on the earthquakes magnitudes. Number 3 is selected by trial and error basis. Output of plotly.plot() function is set as *d*iv so that it returns the *d*iv with scatter plot and data, which has later been embedded into a simple HTML. This HTML is rendered on web browser as output.

3.4. dbscanplot - plotlatlong

dbscanplot implements DBSCAN algorithm for clustering the latitude-longitude data. It reads the data from local database using dblayer common component. Clusters are plotted on a globe using plotly's Scattergeo() function. Following programming statements ensure showing North and South America region on globe.

$$geo = dict(scope = ('northamerica', 'southamerica')$$

$$projection = dict(type = 'orthographic')$$

For clustering lat long data DBSCAN algorithm is used instead of K-Means because, (1) for clustering lat long data it needs an algorithm that can handle arbitrary distance function, (2) K-Means works well with linear data. Lat long data is not linear, (3) Unlike K-Means, number of clusters are not required to mention in DBSCAN. It is hard to predict clusters when locations are spread across world.

4. IMPLEMENTATION

4.1. USGS Web Service and Data Set

USGS earthquake data is fetched using the USGS web service and passing relevant parameters to it. Out of the voluminous world-wide data of decades this project uses data of earthquakes appeared in North and South America for duration of 2 years from 2015-1-1 till 2017-1-1, having magnitude over 4. To select North and South America region data, its Latitude and Longitude are taken from Search Earth Catalog tool provided by USGS. [2]. Web service returns data in JSON format.

4.2. Data Processing and Visualization

Severity of earthquakes. Severity of earthquakes is analysed by their magnitude and depth. Data is classified into clusters using K-Means clustering algorithm (Section 4.1). Clusters are plotted on a interactive scatter plot.

Region affected the most by earthquakes. Analysing Latitude-Longitude of epicenter of the earthquake shows the regions where earthquakes are frequent. Lat-long data is clustered using DBSCAN algorithm (Section 4.2). Clusters are plotted on a geo-scatter plot on a world map.

4.3. Deployment

Project is deployed using ansible. Ansible jobs are collected in a playbook and run on virtual clusters provided by Chameleon and Jetstream cloud. The tasks include cloning the git repository, installing software stack, installing dependencies and installing MongoDB.

4.4. Local Data Storage

To avoid frequent web service calls and unnecessary traffic, data once downloaded, is stored locally in MongoDB for further usage by other components. Data is fetched using pymongo library.

4.5. Steps to Execute

Steps to execute the project are explained in detail in README.md file [3].

5. CLUSTERING ALGORITHMS

5.1. K-Means Clustering

Given a target number, k, of clusters to find, K-means algorithm locates the centers of each of those k clusters and the boundaries between them. It does this using the following algorithm [4].

- Step 1: Start with a randomly selected set of k centroids
- Step 2: Determine which observation is in which cluster, based on which centroid it is closest to (using the squared Euclidean distance.

$$\sum_{j=1}^{p}(x_{ij} - X_{i^1j})^2$$

  where p is the number of dimensions)
- Re-calculate the centroids of each cluster by minimizing the squared Euclidean distance to each observation in the cluster
- Repeat 2. and 3. until the members of the clusters (and hence the positions of the centroids) no longer change.

5.2. DBSCAN

DBSCAN (Density-Based Spatial Clustering of Application with Noise) is a clustering algorithm. Given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), and marks points as outliers if they lie alone in low-density regions. Unlike K-Means Clustering, DBSCAN doesn't need to specify the number of clusters as it finds all the clusters that satisfy the requirement. Following points summarize the algorithm [5].

- Step 1: For each point in the data set, an n-dimensional sphere of radius *e*psilon is drawn around the point (if you have n-dimensional data).
- Step 2: If the number of points inside the sphere is larger than *m*in-samples, the center of the sphere is set as a cluster, and all the points within the sphere belong to this cluster.
- Step 3: Loop through all the points within the sphere with the above 2 steps, and expand the cluster whenever it satisfy the 2 rules.
- Step 4: For the points, which do not belong to any cluster, are treated as outliers.

6. BENCHMARKING

The performance of application was tested on the speed and latency while data input/output and data processing. Two different sized datasets were used for testing. They are given in Table 'Datasets'. The Table 'VM Configuration' shows details of VMs used for performance testing. Results are shown in Table 'Benchmark Results'.

Table 1. VM Configuration

Cloud	Chameleon	Jetstream
Image	Ubuntu-Server-14.04-LTS	ubuntu-14.04-trusty-server-cloudimg
OS	Ubuntu 14.04	Ubuntu 14.04
RAM CPU (Cores)	2 GB	2 GB
HDD Size	20 GB	20 GB
Flavour	m1.small	m1.small
Group	pearth	pearth
Assign Floating IP	True	True

Figure 2 shows time taken in seconds for data read and data processing with small dataset. Figure 3 shows the result with large dataset.

Performance tests are included in the python scripts themselves. As the scripts are executed, results are written in text files in 'benchmark' folder at project directory.

7. RESULT AND ANALYSIS

Table 3 shows the time taken to run ansible playbook successfully on various environments. It took maximum time to deploy the

Table 2. Datasets

Parameter	Dataset1	Dataset2
Region	North, South America	North, South America
Duration	2 years	2 years
Min Magnitude	4	3
Data Size	Small	Large

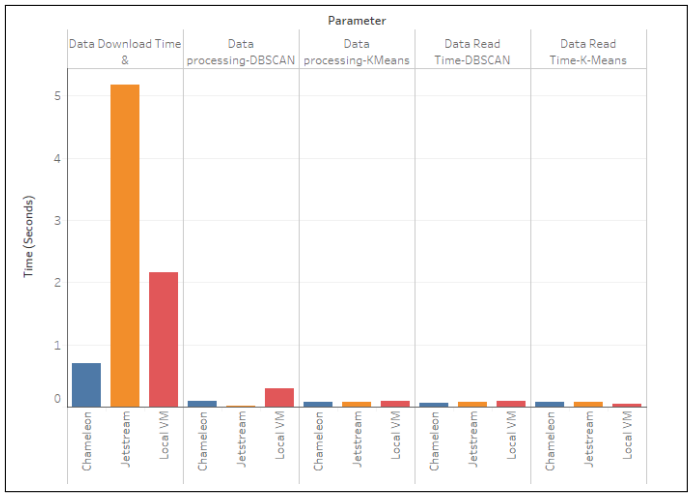


Fig. 2. Time taken for small dataset

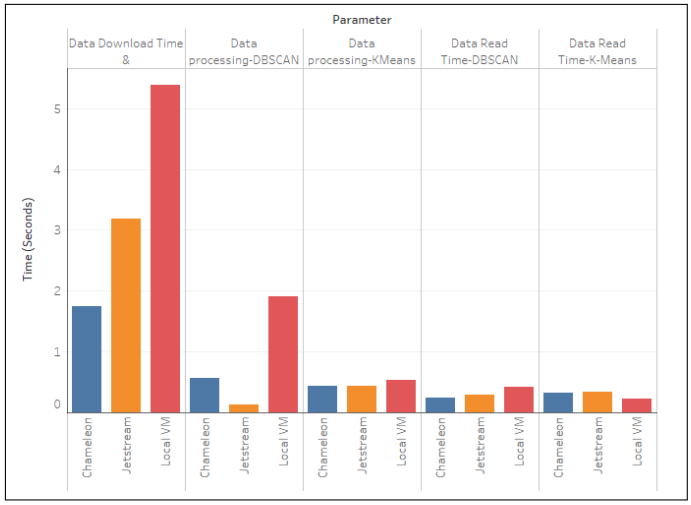


Fig. 3. Time taken for large dataset

application on jetstream. Possible reason could be slow network connectivity.

Table 3. Time taken for Deployment

Environment	Time (hh:mm:ss)
Local VM	00:07:35
Chameleon	00:08:12
Jetstream	00:10:08

Figure 4 shows default page showing application usage. Output of K-Means clustering of magnitude and depth data is given in Figure 5. Figure 6 shows a closer look at scatter plot. Figure 7 is the output of DBSCAN clustering of earthquake locations. Both the graphs are interactive. On the 'Magnitude and Depth' graph one can choose one or more clusters. Mouse hover shows magnitude and its corresponding depth. The 'Earthquake locations' globe shows lat-long information on mouse hover. The user can rotate the globe with mouse key press.

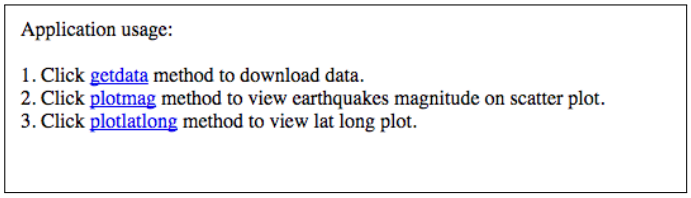


Fig. 4. Application usage

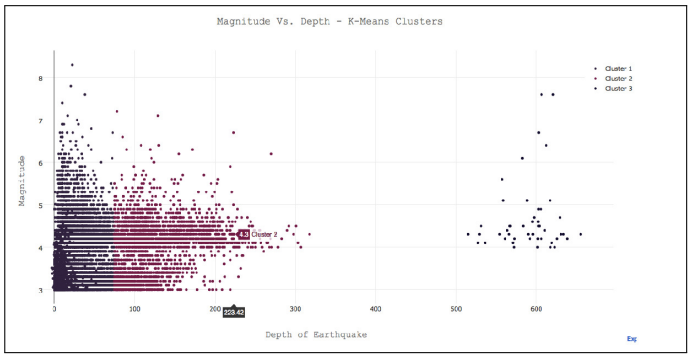


Fig. 5. Magnitude and Depth(Kms)

The focus of an earthquake is the actual point underground where rocks break. From the results it is clear that there were fewer earthquakes having the focus deep, greater than 300kms. Earthquakes having shallow focus (less than 70kms) are comparatively lesser than the ones having intermediate focus (70kms to 300km). Thus, maximum earthquakes occurred between 70kms to 300kms deep underground.

Analysis of earthquake locations shows that maximum earthquakes have happened in the Pacific coastal area of North and south America. We can confirm this result with USGS facts, which states "The majority of the earthquakes and volcanic eruptions occur along plate boundaries such as the boundary between the Pacific Plate and the North American plate. One of the most active plate boundaries where earthquakes and erup-

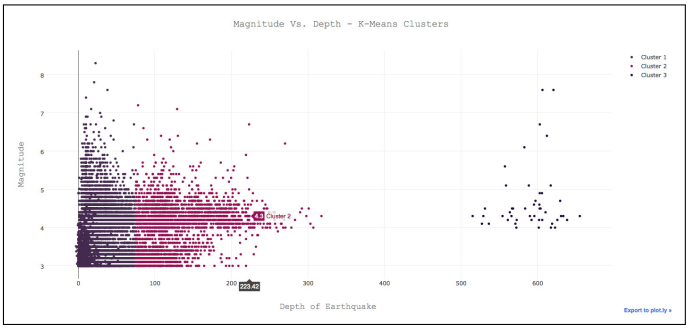


Fig. 6. Close look at the cluster

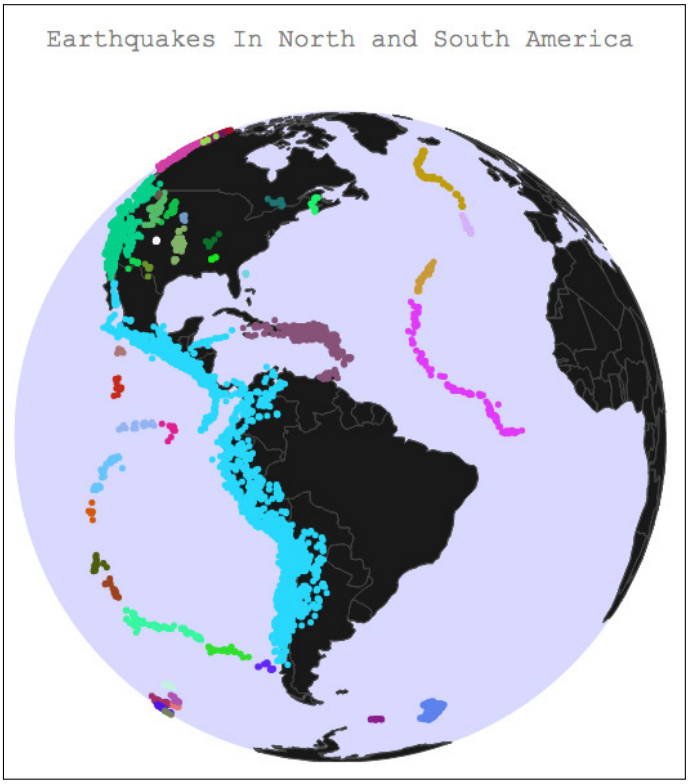


Fig. 7. Earthquake locations on globe

tions are frequent, for example, is around the massive Pacific Plate commonly referred to as the Pacific Ring of Fire [6]."

There is a subtle difference in output of K-Means and DBSCAN algorithms. K-Means worked with lesser number of clusters (3), while DBSCAN created greater number of clusters, more than 50 in this case. K-Means produced output linearly, cluster1 to cluster3. DBSCAN produced non-linear output. A cluster in Pacific ocean is Cluster28, whereas, the cluster adjacent to it is Cluster39.

8. CONCLUSION

This project was an opportunity to use Big Data open source software and projects. We can conclude that Ansible is a powerful tool for one click deployment and continuous integration. Cloudmesh client is a convenient tool to work with multiple cloud environments at the same time. Plotly is a powerful library that allows creating interactive visualization.

While researching for the algorithms we came to know that DBSCAN is better choice for classification if one doesn't want to specify number of clusters beforehand.

Analysis of earthquakes data showed interesting facts. In 2 years span from year 2015 till 2017 there were around 15,000 earthquakes of magnitude more than 3, and approx 7,100 earthquakes of magnitude more than 4.

9. ACKNOWLEDGEMENTS

This project is undertaken as part of the I524: Big Data and Open Source Software Projects course at Indiana University. The author would like to thank Prof. Gregor von Laszewski and his associates from the School of Informatics and Computing for providing all the technical support and assistance.

10. LICENSING

REFERENCES

[1] USGS, "Earthquake hazards program," Web Page. [Online]. Available: https://earthquake.usgs.gov/

[2] USGS, "Search earthquake catalog," Web Page. [Online]. Available: https://earthquake.usgs.gov/earthquakes/search/

[3] N. Sathe, "Readme," Code Repository, April 2017, accessed: 2017-4-25. [Online]. Available: https://github.com/cloudmesh/earth/blob/master/README.md

[4] B. Govan, "Clustering using scikit-learn," Web Page, May 2013. [Online]. Available: http://fromdatawithlove.thegovans.us/2013/05/clustering-using-scikit-learn.html

[5] Q. Kong, "Clustering with dbscan," Web Page, August 2016. [Online]. Available: http://qingkaikong.blogspot.in/2016/08/clustering-with-dbscan.html

[6] USGS, "Earthquake facts," Web Page. [Online]. Available: https://earthquake.usgs.gov/learn/facts.php

11. APPENDICES

Appendix A: The work on this project was distributed as follows between the authors:

Nandita Sathe. She completed all the work related to development of this application including research, testing and writing the project report.

S17-IO-3018/report/report.pdf not submitted

Twitter sentiment analysis of the Affordable Care Act in 2017

MICHAEL SMITH[1]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*] Corresponding authors: mls35@iu.edu

project001,May 4, 2017

The mission of this project is to utilized technologies and cloud computing to perform a successful sentiment analysis through software deployment written in python. The software deployment will encompass data mining, analysis of big data, and comparison of this deployment across a variety of cloud computing services. The sentiment analysis will use the social media platform twitter and python libraries that effectively extract relevant data to the project goal.

Keywords: Cloud, I524

https://github.com/cloudmesh/sp17-i524/tree/master/project/S17-IO-3019/report

1. INTRODUCTION

The current political climate of the United States is divided on many important issues. There is a disconnect between the motivations of the politicians of today and what is deemed important to the American people. The affordable care act(ACA) also known as Obamacare has been a target of the GOP, however it is uncertain if that sentiment is shared by most Americans. With a law that affects millions of Americans it is often difficult to gauge how the American people feel about the current healthcare law. Twitter is one of the biggest social medias on the internet with 67 million active users in the United States as of Q4 2016.[1] It is the goal of the project to gauge how Obamacare is viewed by twitter users who reside in the United States.

2. TECHNOLOGIES

Cloudmesh is an open source toolkit that allows the user to work across a variety clouds, virtual machines and clusters. This facilitates the ease of porting deployments to different clouds enabling the capability to benchmark cloud performance on a particular deployment. [2] The project was developed by Gregor von Laszekski and his colleagues at Indiana University. This will be the primary interface used to port the software to clouds such as chameleon cloud and futuresystems.

Ansible is open source software used for automation of provisioning of software deployments. This will be used when deploying on virtualization and cloud environments.

Chameleon cloud and futuresystems to be discussed here.

3. PYTHON

Early scripts have already been developed utilizing Twitters API and the python library tweepy to mine tweets that contain information relevant to Obamacare. Currently, the code authenticates with the twitter API, mines the tweets that contain a keyword of interest and finally a sentiment analysis which will rate a tweet by its polarity and subjectivity. For the sentiment analysis, the TextBlob library is used for its natural language processing(NLP) functionality. The final part of code outputs the tweet content and sentiment analysis into two columns into a csv file. This code will evolve to include data visualization through matplotlib and deeper analysis as the project moves closer to completion. Other python libraries will likely be added as well.

4. BENCHMARKS

Software will be deployed and benchmarked on various clouds.

5. LICENSING

TBD

6. WORK BREAKDOWN

Michael Smith is reponsible for all aspects of this project.

7. CONCLUSION

The software once finalized will be deployed across various clouds with the help of cloudmesh and ansible. Benchmark performance of the various clouds as well as analysis of the

twitter sentiment data will encompass most the final project report.

8. AUTHOR BIOGRAPHIES

Michael Smith is a senior quality control peptide chemist at Creosalus Inc. in Louisville, Kentucky. Michael possesses a MS in pharmaceutical sciences and a BS in Biology from the University of Kentucky. He will obtain his MS in Data Sciences program from Indiana University in May 2018. His current interests are python programming, data analytics, and spending time with his children.

REFERENCES

[1] Statista, "Number of monthly active twitter users in the united states." [Online]. Available: https://www.statista.com/statistics/274564/monthly-active-twitter-users-in-the-united-states/

[2] Cloudmesh, "Cloudmesh," Webpage. [Online]. Available: https://cloudmesh.github.io/

Detection of street signs in videos in a robot swarm

SUNANDA UNNI[1,*] **AND GREGOR VON LASZEWSKI**[1,**]

[1] *School of Informatics and Computing, Bloomington, IN 47408, U.S.A.*
[*] *Corresponding authors: suunni@indiana.edu*
[**] *Corresponding authors: laszewski@gmail.com*

project-1: Data analysis of Robot Swarm data, May 4, 2017

Extracting and identifying traffic signals from the videos captured by Robot swarms to help in recognizing the pattern and benchmarking the performance of the setup.

Keywords: Cloud, I524

https://github.com/cloudmesh/sp17-i524/blob/master/project/S17-IO-3022/report/report.pdf

1. INTRODUCTION

For test purpose we created some mobile videos of traffic in a simulated traffic setup. All saved video files are uploaded on the Hadoop HDFS [1]. Batch processing is enabled on the input video files to search for key images, namely the red, green and yellow signals in the images using the OpenCV [2] library's Template matching functionality. Hadoop Map reduce [1] is used for processing and analysis of the images in the videos and getting a count of the how many red or green or yellow signals are encountered.

collectd [3] is used for benchmarking of the setup with Apache Hadoop using various sized data sets and number of nodes.

2. TECHNOLOGY USED

tables need a begin table end table

Technology Name	Purpose
Hadoop [1]	map reduce
OpenCV [2] Pattern matching in video	
ansible [4]	Automated deployment
collectd [3]	Collection of statistics of setup for benchmarking

3. PLAN

tables need a begin table end table

Week	Work Item	Status
week1	Ansible deployment script for Hadoop setup	planned
week2	Ansible deployment script for OpenCV setup	planned
week3	Creating sample videos	planned
week4	OpenCV template matching script	planned
week5	Deployment and test of basic setup	planned
week6	Ansible deployment of collectd	planned
week7	Performance measurement of setup and report creation	planned
week8	Exploring different setup	planned

4. DESIGN

TBD

5. DEPLOYMENT

TBD

6. BENCHMARKING

TBD

7. DISCUSSION

TBD

8. CONCLUSION

TBD

9. ACKNOWLEDGEMENT

REFERENCES

[1] Apache Software Foundation, "Apache hadoop," Web Page, 2014. [Online]. Available: http://hadoop.apache.org/

[2] itseez.com, “Opencv- open source computer vision,” Web Page, 2017. [Online]. Available: http://opencv.org/
[3] “collectd - the system statistics collection daemon,” Web Page. [Online]. Available: https://collectd.org/
[4] “Ansible, deploy apps. manage systems. crush complexity,” Web Page. [Online]. Available: https://www.ansible.com/

Analysis of H-1B Temporary Employment-Based in Data Science Occupation

JIMMY ARDIANSYAH[1,*]

[1] *School of Informatics and Computing, Bloomington, IN 47408, U.S.A.*
[*] *jardians@indiana.edu - S17-IR-2002*

Project Proposal, May 4, 2017

This project aims to analyze The H-1B temporary employment-based visa for Data Science related occupations in the United States. We are trying to answer the number of questions related to Data Science related jobs in America's workforce based on H-1B visa.

Keywords: Apach, Hadoop, H1B, Data Science

https://github.com/jardians/sp17-i524/blob/master/project/S17-IR-2002/report/report.pdf

1. INTRODUCTION

The H-1B non-immigrant classification is a vehicle through which a qualified alien may seek admission to the United States on a temporary basis to work in his or her field of expertise. An H-1B petition can be filed for an alien to perform services in a specialty occupation. Prior to employing an H-1B temporary worker, the U.S. employer must first file a Labor Condition Application (LCA) [1] with Department of Labor Certification [2] and then file an H-1B petition with United States Citizenship and Immigration Services(USCIS). The LCA specifies the job, salary, length, and geographic location of employment. The employer must agree to pay the alien the greater of the actual or prevailing wage for the position [3].

To qualify as a specialty occupation, the position must meet one of the following requirements: (1) a bachelor's or higher degree or its equivalent is normally the minimum entry requirement for the position; (2) the degree requirement is common to the industry in parallel positions among similar organizations or, in the alternative, the position is so complex or unique that it can be performed only by an individual with a degree; (3) the employer normally requires a degree or its equivalent for the position; or (4) the nature of the specific duties is so specialized and complex that the knowledge required to perform the duties is usually associated with attainment of a bachelor's or higher degree

In the past 6 years, tech industry executive bemoan the lack of data scientists—the people who theoretically know how to look at the data your company generates, and delve into it to derive the all-important insights we keep hearing about. It's no secret that there's a shortage of data scientists in America's workforce. Many companies look to hire overseas to help ease the domestic talent shortfall (in fact, one in three data scientists are born outside the U.S.) so understanding the ins and outs of visas is rapidly becoming a business necessity [4]. To accomplish the goals, I would like to answer question like the following:

- Is it the number of petitions with Data Engineer or Scientist jobs title increasing over time?
- Which part of the US has the most Data Engineer or Scientist jobs?
- what year petitions with Data Engineer or Scientist jobs granted the most between 2011 to 2016?
- Which employers file the most petitions with Data Engineer or Scientist jobs title each year?

2. PLAN

Following table gives a breakdown of tasks in order to complete the project. Assuming week1 starts after submission of the proposal. These work items are high level breakdown on the tasks and may changes if needed.

Time	Work Item	Status
Week-1	Ansible Playbook Deployment	Planned
Week-2	ETL and Analysis	Planned
Week-3	Performance Measurement	Planned
Week-4	Report Creation	Planned

Fig. 1. Planned Schedule

3. DESIGN

I break the high-level design of the technologies used into 3 main sections– storage, ingestion, processing and analyzing.

- Storage refers to decision around the storage system such as HDFS or HBase [5]
- Ingestion refers to getting data from source and loading it into Hadoop for processing.
- Analyzing refers to running various analytical queries on processed dataset to find answer and insight to the questions presented.

4. DATASET METADATA DESCRIPTION

The columns included in the dataset download from Kaggle [6] site are followed :

- `CASE_STATUS`: Status associated with the last significant event or decision.
- `EMPLOYER_NAME`: Name of employer submitting labor condition application.
- `SOC_NAME`: the occupational code associated with the job being requested for temporary labor condition, as classified by the Standard Occupational Classification (SOC) System.
- `JOB_TITLE`: Title of the job
- `FULL_TIME_POSITION`: Y = Full Time Position; N = Part Time Position
- `PREVAILING_WAGE`: Prevailing Wage for the job being requested for temporary labor condition. The wage is listed at annual scale in USD. The prevailing wage for a job position is defined as the average wage paid to similarly employed workers in the requested occupation in the area of intended employment. The prevailing wage is based on the employer's minimum requirements for the position. YEAR: Year in which the H-1B visa petition was filed
- WORKSITE: City and State information of the foreign worker's intended area of employment
- LON: longitude of the Worksite
- LAT: latitude of the Worksite

5. DEPLOYMENT

Solution will be deployed using Ansible [7] ad-hoc commands and Linux commands. Driver script called `cc_main_driver.sh` should install all necessary software and project codes to the cluster nodes. The `cc_main_driver.sh` will copy both Python script `called cc_analyze_data.py` which analyzes and generates graphs/tables and shell script `called cc_etl_data.sh` into clusters. The `cc_main_driver.sh` will trigger `cc_etl_data.sh` to pull dataset from the web as well executes `cc_analyze_data.py` to analyze a dataset.

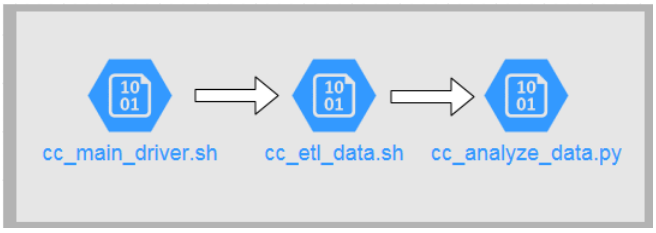


Fig. 2. Deployment Schema

6. BENCHMARKING

The original input dataset with approximately 3,000,000 rows (`h1b_3mRows`) split into two smaller datasets: 1,000,000 rows (`h1b_1mRows`) rows and 2,000,000 rows (`h1b_2mRows`). Then, I executed Python script with Linux time function (i.e: `time python ./cc_analyze_data.py`) against each of the input dataset mentioned above in order to measure both the storage size and elapsed time during the execution.

The benchmark testing on Chameleon Cloud environment revealed in the Figure-4 that elapsed processing time decreased when the number of rows in the dataset reduced. In the Figure-5, similar trend applied to disk space usage that it decreased linearly as the less number of rows need to be stored.

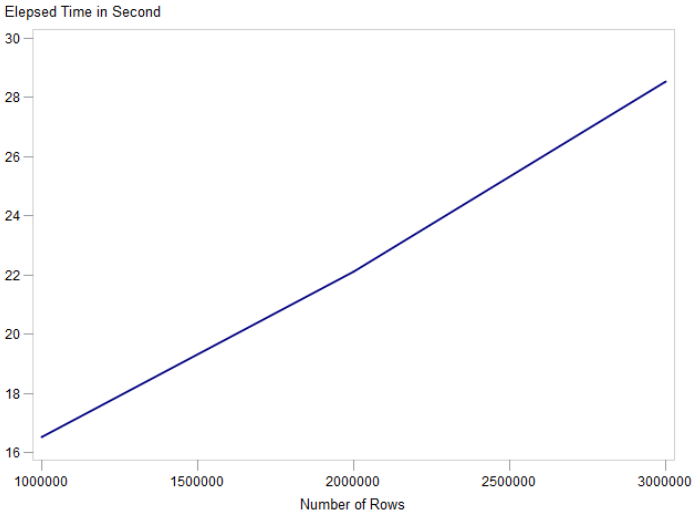


Fig. 3. Benchmark Testing - Number of Rows Vs. Elapsed Time

```
------------------------------------------------------------------------
****************************** BENCHMARK ***************************
------------------------------------------------------------------------
DATASET              REAL       USER       SYS        DISK
3000000 (h1b_3mRows) 0m28.528s  0m15.520s  0m0.384s   470 MB
2000000 (h1b_2mRows) 0m22.112s  0m09.671s  0m0.255s   312 MB
1000000 (h1b_1mRows  0m16.528s  0m05.528s  0m0.201s   156 MB
```

Fig. 4. Benchmark Testing - Number of Rows Vs. Disk Storage

7. DATA REPORT

General petition distribution between Fiscal Year(FY) 2011 to FY 2016, United States Citizenship and Immigration Services (USCIS) approved 2,615,623 petitions submitted by the employer on behalf of alien workers as indicated in the Figure-5.

Of the petitions approved during FY 2011-2016, a total 10,132 petitions, or .38 % were Data Science related occupations (i.e: Data Scientist, Data Analytics, Data Science Engineer, Statistician and Data Modelling) as shows in the Figure-6.

```
---------------------------------------------------------------------
****************** CASE STATUS DISTRIBUTION ***********************
---------------------------------------------------------------------
CERTIFIED                                                 2615623
CERTIFIED-WITHDRAWN                                        202659
DENIED                                                      94346
WITHDRAWN                                                   89799
PENDING QUALITY AND COMPLIANCE REVIEW - UNASSIGNED             15
REJECTED                                                        2
INVALIDATED                                                     1
---------------------------------------------------------------------
```

Fig. 5. General Distribution of Petition - All Jobs

```
---------------------------------------------------------------------
****************** CASE STATUS DISTRIBUTION ***********************
---------------------------------------------------------------------
CERTIFIED              10132
CERTIFIED-WITHDRAWN     1009
WITHDRAWN                391
DENIED                   245
---------------------------------------------------------------------
```

Fig. 6. General Distribution of Petition - Data Science Related Occupations

As Figure-7 indicated, petitions submitted regardless of the CASE_STATUS and all JOB_TITLE increased approximately 5 to 7 percent. For Data Science related petitions also increased especially in metropolitan areas such as San Francisco, New York and Menlo Park . The highest number of petition related to Data Science petitions to acquire H1-B visa was in the Fiscal Year 2016 as shown on the Figure-8.

```
-------------------------------------------------------------------------------
************************ PETITION PER STATE PER YEAR *************************
-------------------------------------------------------------------------------
YEAR                 2011.0  2012.0  2013.0  2014.0  2015.0  2016.0   TOTAL
STATE
ALABAMA                1487    1572    1487    1781    1873    2053   10253
ALASKA                  205     273     260     246     213     199    1396
ARIZONA                4391    5488    6389    7306    8746    9734   42054
ARKANSAS               1680    1890    2442    2329    3015    3406   14762
CALIFORNIA            65690   76402   83852   98512  115743  119741  559940
COLORADO               3630    4378    4889    5811    6827    6502   32037
CONNECTICUT            5885    7827    7447    8917   10142   10035   50253
DELAWARE               2152    2348    3172    3184    3760    3522   18138
DISTRICT OF COLUMBIA   3491    3570    3687    3727    4099    4134   22708
FLORIDA               15227   16368   15283   17644   20401   20850  105773
GEORGIA               10829   12733   13994   17728   23026   24857  103167
HAWAII                  655     725     615     602     598     557    3752
IDAHO                   638     644     635     609     778     887    4191
ILLINOIS              18595   22350   24510   27407   32768   35184  160814
INDIANA                3837    4340    4281    5509    6150    6399   30516
IOWA                   2308    2513    2607    3168    3207    2940   16743
KANSAS                 1713    2046    2233    2424    2598    2768   13782
KENTUCKY               1600    2017    1889    2170    2403    2623   12702
LOUISIANA              1615    1661    1662    1838    2702    2191   11669
MAINE                   541     586     672     714     718     687    3918
MARYLAND               8544    8350    8132    9601   10891   10738   56256
MASSACHUSETTS         14720   16556   16898   19913   23488   24891  116466
MICHIGAN               8305    9918   11535   13918   18318   20970   82964
MINNESOTA              5683    6900    7194    8996    9975    9937   48685
MISSISSIPPI             648     645     668     678     792     839    4270
MISSOURI               3756    4714    4988    6200    7182    7973   34813
MONTANA                 163     137     156     134     205     191     986
NEBRASKA               1089    1242    1388    1708    1815    2014    9256
NEVADA                 1129    1223    1119    1231    1350    1396    7448
NEW HAMPSHIRE          1185    1526    1558    1676    2078    1906    9929
NEW JERSEY            23611   27856   29794   36783   47662   48370  214076
NEW MEXICO              782     953     854     908    1005    1039    5541
NEW YORK              41769   44512   42565   48877   55017   58670  291410
NORTH CAROLINA         7783   10411   11668   13550   17413   18847   79672
NORTH DAKOTA            403     446     469     490     575     544    2927
OHIO                   8582   10426   11642   13515   16066   16344   76575
OKLAHOMA               1457    1656    1577    1846    2046    2015   10597
OREGON                 2859    3103    3712    4595    4803    4718   23790
PENNSYLVANIA          12896   15552   16779   19150   22202   23380  109959
PUERTO RICO             309     311     207     209     214     202    1452
RHODE ISLAND           1038    1323    1792    2225    2881    2458   11717
SOUTH CAROLINA         1628    1795    1672    2084    2801    2952   12932
SOUTH DAKOTA            328     286     261     281     398     348    1902
TENNESSEE              3463    4544    4268    4584    5161    5652   27672
```

Fig. 7. H1-B Petition Per Year Per State - All Jobs

```
-------------------------------------------------------------------------------
************************ PETITION PER STATE PER YEAR *************************
-------------------------------------------------------------------------------
YEAR                 2011  2012  2013  2014  2015  2016  TOTAL
STATE
ALABAMA                 8     8     9     6     4     2     37
ARIZONA                14     8     7    14    18    24     85
ARKANSAS                4     3     1     6    25    34     73
CALIFORNIA            183   219   301   508   733  1003   2947
COLORADO                5     4     6    17    18    17     67
CONNECTICUT            36    21    25    26    37    61    206
DELAWARE                9    13    14     9    20    17     82
DISTRICT OF COLUMBIA   12    12    16     8    25    25     98
FLORIDA                23    16    22    28    59    46    194
GEORGIA                19    19    27    40    84   105    294
HAWAII                NaN     3     3     3     5     4     18
IDAHO                 NaN   NaN   NaN     1     2     1      4
ILLINOIS               66    60    66   100   123   173    588
INDIANA                14    21    26    18    28    28    135
IOWA                    5     7     9     9     7    11     48
KANSAS                 12    15     9    11    18    16     81
KENTUCKY                6     4     2     1     4     9     26
LOUISIANA               2     1   NaN     1     5     3     12
MARYLAND               53    60    41    63    50    56    323
MASSACHUSETTS          51    78    92   123   193   249    786
MICHIGAN               15    18    24    25    40    64    186
MINNESOTA              18    15    20    21    26    29    129
MISSISSIPPI           NaN     4     1     2     2     2     11
MISSOURI               15    17    11    17    18    38    116
NA                      1   NaN   NaN   NaN   NaN   NaN      1
NEBRASKA                8     5     2     6    18     9     48
NEVADA                  3     9     4     5     4    11     36
NEW HAMPSHIRE           4     2     4     5     6     6     27
NEW JERSEY             96   124   142   150   168   223    903
NEW MEXICO            NaN     1   NaN   NaN     3   NaN      4
```

Fig. 8. H1-B Petition Per Year Per State - Data Science Related Occupations

As revealed in Figure-9, New Jersey, California, Massachusetts and Illinois are top locations that hires Data Science talents. Almost all technology based companies are now aware that data-driven decision making is critical if they want to succeed. As showed in the Figure-10, some of the biggest and well-known technology companies are the biggest driving force in hiring talent pool with Data Science skills.

```
---------------------------------------------------------------------
*************** TOP 25 LOCATION HIRING DATA SCIENTIST *************
---------------------------------------------------------------------
SAN FRANCISCO, CALIFORNIA                 332
NEW YORK, NEW YORK                        224
MENLO PARK, CALIFORNIA                    103
MOUNTAIN VIEW, CALIFORNIA                 101
REDMOND, WASHINGTON                        78
PALO ALTO, CALIFORNIA                      71
SAN JOSE, CALIFORNIA                       55
SUNNYVALE, CALIFORNIA                      52
BOSTON, MASSACHUSETTS                      45
BELLEVUE, WASHINGTON                       44
CHICAGO, ILLINOIS                          41
CAMBRIDGE, MASSACHUSETTS                   36
SEATTLE, WASHINGTON                        34
SAN MATEO, CALIFORNIA                      27
AUSTIN, TEXAS                              25
ATLANTA, GEORGIA                           25
REDWOOD CITY, CALIFORNIA                   21
SANTA MONICA, CALIFORNIA                   17
HOUSTON, TEXAS                             16
SANTA CLARA, CALIFORNIA                    15
SAN DIEGO, CALIFORNIA                      13
WASHINGTON, DISTRICT OF COLUMBIA           12
BURLINGTON, MASSACHUSETTS                  12
LOS ANGELES, CALIFORNIA                    12
CHARLOTTE, NORTH CAROLINA                  11
---------------------------------------------------------------------
```

Fig. 9. Top 25 Location Hiring Data Scientist

```
----------------------------------------------------------------------
************ TOP 25 COMPANY HIRING DATA SCIENTIST *****************
----------------------------------------------------------------------
MICROSOFT CORPORATION                  139
FACEBOOK, INC.                          98
UBER TECHNOLOGIES, INC.                 48
TWITTER, INC.                           31
AIRBNB, INC.                            25
GROUPON, INC.                           21
LINKEDIN CORPORATION                    20
AGILONE, INC.                           19
IBM CORPORATION                         16
WAL-MART ASSOCIATES, INC.               15
INTUIT INC.                             14
RANG TECHNOLOGIES, INC.                 13
PAYPAL, INC.                            12
SCHLUMBERGER TECHNOLOGY CORPORATION     11
APPLE INC.                              11
STITCH FIX, INC.                        10
TRIPADVISOR LLC                         10
INTEL CORPORATION                        9
THE NIELSEN COMPANY (US), LLC            9
LYFT, INC.                               8
GOOGLE INC.                              7
AMERICAN EXPRESS COMPANY                 7
CLOUDWICK TECHNOLOGIES INC.              7
ICUBE CONSULTANCY SERVICES, INC          7
ZILLOW, INC.                             7
----------------------------------------------------------------------
```

Fig. 10. Top 25 Companies Hiring Data Scientist

As shown in Figure-11, for occupations in Data Science field, the median annual compensation reported by employers of H-1B workers between FY 2011 to FY 2016 was ranged from a low of $40,000 to a high $110,000 which depends on geological location.

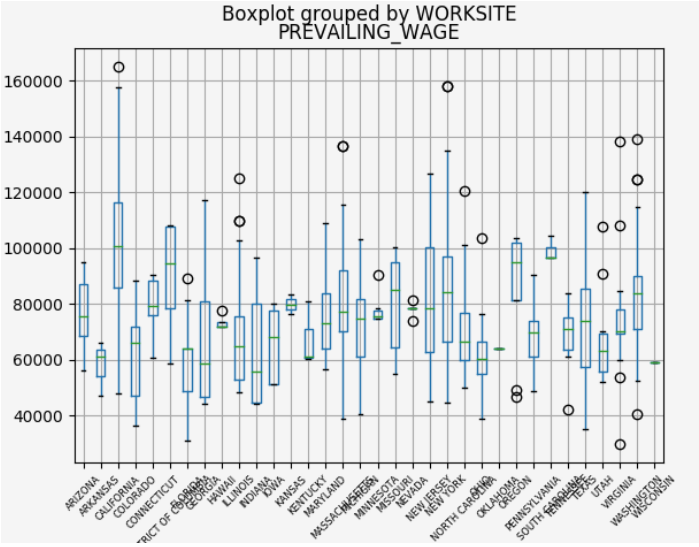


Fig. 11. Data Scientist Wage Across States

8. CONCLUSION

Overall, there is compelling evidence that the H-1B visa program is helping to alleviate acute shortages in Data Science occupations since the number of petitions submitted increased linearly from FY 2011 to FY 2016. Armed with such information, as well as indicators presented above, Data Science occupation mostly concentrated in large metropolitan areas. Well-known technology companies has indicated hired professional with Data Science skill sets.

9. ACKNOWLEDGEMENT

This work was done as part of the course "I524: Big Data and Open Source Software Projects" at Indiana University during Spring 2017. We acknowledge our Professor Gregor Von Laszewski and all Associate Instructors for helping us and guiding us throughout this project.

REFERENCES

[1] Wikipedia, "Labor condition application," Web Page, Apr. 2017, accessed: 2017-04-20. [Online]. Available: https://en.wikipedia.org/wiki/Labor_Condition_Application

[2] USCIS, "Labor certification," Web Page, Apr. 2017, accessed: 2017-04-20. [Online]. Available: https://www.uscis.gov/tools/glossary/labor-certification

[3] Wikipedia, "H-1b visa," Web Page, Apr. 2017, accessed: 2017-04-20. [Online]. Available: https://en.wikipedia.org/wiki/H-1B_visa

[4] M. Li, M. J. Wildes, and A. W. Moses, "Hiring data scientists from outside the u.s.: A primer on visas," Web Page, Mar. 2017, accessed: 2017-03-20. [Online]. Available: https://hbr.org/2016/09/hiring-data-scientists-from-outside-the-us-a-primer-on-visas

[5] Wikipedia, "Apache hadoop," Web Page, Mar. 2017, accessed: 2017-03-20. [Online]. Available: https://en.wikipedia.org/wiki/Apache_Hadoop

[6] S. Naribole, "H-1b visa petitions 2011-2016," Web Page, Mar. 2017, accessed: 2017-03-20. [Online]. Available: https://www.kaggle.com/nsharan/h-1b-visa

[7] Wikipedia, "Ansible," Web Page, Mar. 2017, accessed: 2017-03-20. [Online]. Available: https://en.wikipedia.org/wiki/Ansible

On-line advertisement click prediction - Project proposal

SAHITI KORRAPATI[1,*]

[1] *School of Informatics and Computing, Bloomington, IN 47408, U.S.A.*
[*] *Corresponding authors: sakorrap@iu.edu, S17-IR-2013*

March 13, 2017

This project aims at predicting the most suitable advertisements to be displayed on the web pages. Advertisements are selected based on the relevance. Relevance factor is calculated by ranking each ad based on the likelihood of clicking the ad if displayed. Data is obtained as CSV files from Kaggle Data sets and is stored in Hadoop Data File system(HDFS). In this project, Ansible along with Cloud mesh is used to deploy cloud architecture and the necessary soft wares on Chameleon cloud. For data exploration and analysis, as the dataset is huge in the order of 100s of gigabytes Apache Pig on Hadoop is chosen. The Pig engine in can execute the data flows in parallel by making use of Hadoop MapReduce framework.

Keywords: Hadoop, Ad click Prediction, BigData, Ansible, Chameleon cloud, Apache Pig, HDFS, MapReduce

https://github.com/sakorrap/sp17-i524/tree/master/project/S17-IR-2013/report.pdf

1. INTRODUCTION

It has been analyzed that an average American spends about 23 hours per week surfing on-line [1]. This on-line user activity is being captured by companies to perform analyzes for advertisement, recommendations and many other purposes. This has given rise to the field of "Web Analytics" and one such application is Ad Click prediction.

Many measures are available to asses the ad performance. One popular measure to asses the immediate ad response is click-through rate (CTR) of the advertisement [2] which is defined as the ratio of number of clicks on an ad to the number of times the ad is shown, expressed as a percentage. In this project, I attempted to make use of CTR along with ranking each ad based on the level of relevance to the original web page. For calculating the CTR, historical data of user activity needed to be explored. [3].

The user activity data from web that is used for prediction is enormous. Every page view, advertisement and click is being tracked by web browsers, and that level of data for millions of users potentially generates enormous volumes. The current internet giants use big data technologies to handle such large volumes of data. The current project uses Apache pig along with Hadoop for analyzing and for prediction. Hadoop is a large scale data processing system which runs on parallal processing to handle huge volumes of data. Pig is a high level language which runs on top of Hadoop's HDFS infrastructure. Pig Latin scripts are simple to comprehend and SQL-like queries which can process large volumes of data.

Ansible along with Cloudmesh is used to deploy the software and chameleon cloud for running virtual machines. Ansible is a cloud automation tool which allows easy deployment and configuration of multiple servers in one step. Cloudmesh allows us to deploy and install Hadoop instances on a cluster with any number of nodes.

2. BACKGROUND

2.1. About Data

The data set in this paper is taken from kaggle datasets. It is released by Outbrain which has 2 Billion page views and 16,900,000 clicks of 700 Million unique users, across 560 sites [4].

The dataset contains a sample of users' page views and clicks, as observed on multiple publisher sites in the United States between 14-June-2016 and 28-June-2016. Each viewed page or clicked recommendation is further accompanied by some semantic attributes of those documents [4]. It contains numerous sets of content recommendations served to a specific user in a specific context. Each context (i.e. a set of recommendations) is given a display_id. In each such set, the user has clicked on at least one recommendation. Our task is to rank the recommendations in each group by decreasing predicted likelihood of being clicked [4].

Each user in the dataset is represented by a unique id (uuid). A person can view a document (document_id), which is simply a web page with content (e.g. a news article). On each document, a set of ads (ad_id) are displayed. Each ad belongs to a campaign (campaign_id) run by an advertiser (advertiser_id). Figure 1

shows the fields in our dataset. Metadata about the document is also provided, such as which entities are mentioned, a taxonomy of categories, the topics mentioned, and the publisher [4].

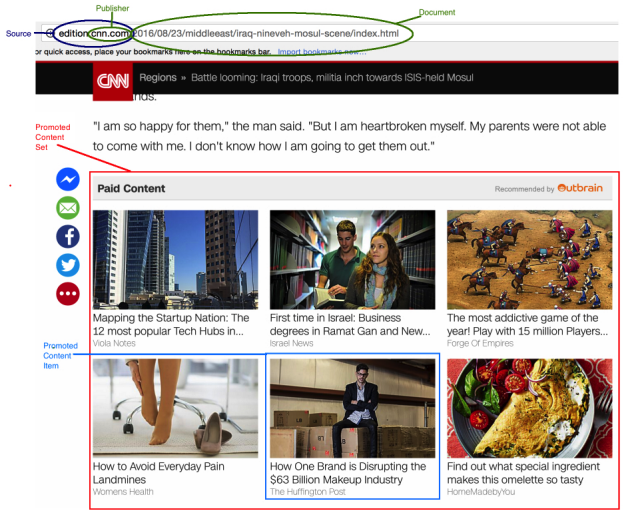


Fig. 1. Displaying Source, Publisher, Document, Promoted content set and items [4]

2.2. Ansible and Cloudmesh client toolkit

Ansible is an opensource cluster management tool which has the ability to maintain a fully immutable server architecture and design. It is a simple automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration [5]. It doesn't use agents or custom security infrastructure, so it's easy to deploy by using "YAML-language" (YAML, in the form of Ansible Playbooks).

Using Ansible, users can define any number of IP Addresses in hosts file, and create custom groups for specific tasks. These group names can be referred in the YAML file to perform specific set of tasks in each of the virtual machines. Ansible also has options to define specific roles for each task and make YAML files dynamic using global variables.

With Cloud mesh, the deployment becomes even simpler and easier. Cloud mesh client toolkit, a lightweight client interface to access heterogeneous clouds, clusters, and workstations, available as API, commandline client and commandline shell. Their quick start user manual has everything that is needed to start using the tool [6].

The current project uses Cloudmesh's "cm" command to perform multiple tasks including deployment of cluster, installing and configuring Hadoop and Pig, enabling the cloud nodes to be able to ssh with each other and assigning floating IP addresses to each node. The configuration of the cluster can be defined in Cloudmesh's YAML file and accordingly the cluster will be created with the given OS and type.

2.3. Hadoop in Big-data

Hadoop is a distributed platform designed to help manage and analyze massive data-sets that are too big or costly to put in relational databases. Rather than storing and processing the whole data in one high-end computer, data can be spread across many smaller nodes. By spreading the data across multiple nodes, the storage as well computational speed can be scaled by parallel processing [7].

Hadoop framework is developed for distributed processing of large data sets across clusters of computers using simple programming models. It can scale up from single servers to thousands of machines, each offering local computation and storage. The library is designed to deliver a highly available service on top of a cluster of computers, each of which may be prone to failures [8].

2.4. HDFS and MapReduce

Hadoop Distributed File System (HDFS) and the MapReduce parallel processing framework are two primary components at the core of Apache Hadoop. Figure 2 shows the high level architecture of Hadoop in regards to HDFS and MapReduce

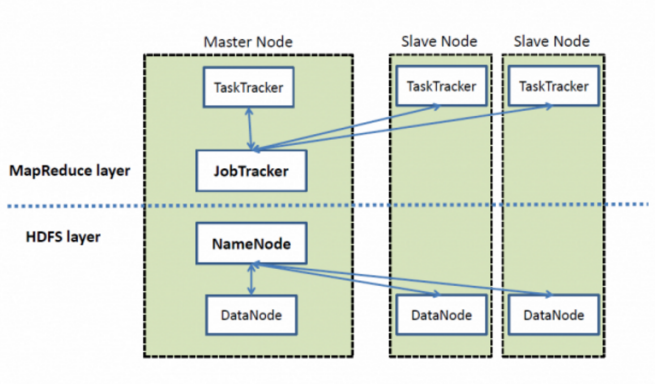


Fig. 2. High Level Architecture of Hadoop [9]

HDFS is a distributed, scalable, and portable file-system for Hadoop framework. Files are broken into blocks and spread across nodes in the cluster in HDFS. The blocks are also replicated on different nodes so that even if one of the nodes fails another one of the live nodes still has a copy of the data. There is a NameNode and DataNodes. The NameNode maintains the references on the file split up in blocks across nodes in the cluster. While reading, client process contacts the NameNode for this metadata and ask the corresponding DataNodes for those blocks for reading [7]. MapReduce is for processing files stored in a distributed environment like HDFS. A typical MapReduce application has two functions, a Mapper and a Reducer. Mappers and Reducers run as tasks on nodes in the cluster. Two processes JobTracker and TaskTracker manages MapReduce framework. The JobTracker is the master process that coordinates the Map and Reduce tasks sent across the cluster.

2.5. Apache Pig

Pig platform is designed to work with large data sets for analysis. The Pig dialect is called Pig Latin, and the Pig Latin commands get compiled into MapReduce jobs that can be run on a suitable platform, like Hadoop. Figure 3 illustrates how it makes use of MapReduce framework.

Pig is a high level language which is similar to SQL in syntax. The structure of Pig is designed such that the Pig scripts are made amenable for parallel processing. Pig scripts are automatically optimized so that the programmers can focus on semantics rather than efficiency [10]. Pig also has options to create user defined functions to perform complex tasks using Pig.

2.6. Chameleon cloud

Chameleon Cloud is a cloud platform offered for research purposes free of charge. It is maintained by the Chameleon commu-

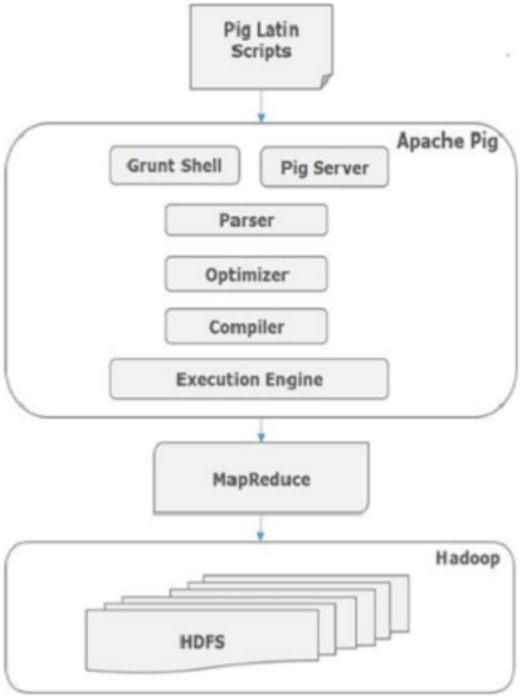


Fig. 3. Architecture of Apache Pig [9]

nity, and is funded exclusively for use by students and faculty by the National Science Foundation. Chameleon Cloud is deployed at University of Chicago and Texas Advanced Computing Center, and consists 650 multi-core cloud nodes, 5PB of total disk space, and leverage 100 Gbps connection between the sites.

The current project uses a cluster created on Chameleon cloud instance with three virtual machines on which Hadoop operates. Ubuntu 14.04 was installed on these machines with 20 GB disk space and 2 GB ram each.

2.7. Data Analysis

Data was analyzed using Pig in the MapReduce mode. The datasets clicks_train and events were joined to get a list of documents to make the predictions. The above list was joined to documents_events, documents_categories and documents_topics to generate a master dataset of all the required documents, their topics, categories and entities. The dataset promoted_content was joined with documents_events, documents_categories and documents_topics to generate a dataset which has all advertisement ids, and their related document entitiy, topic and category. A master join between the two datasets on the event, topic and category ID was performed to get matching document IDs. On this data, relevancy score for each match, by multiplying the confidence levels. The final dataset is grouped by document ID and select top 6 matching ads based on relevancy score. The top 6 ads are the recommendations to be shown on each page. This recommendation engine is purely based on the matching document properties.

3. SETUP AND CONFIGURATION

This project was setup using Ansible and Cloudmesh Client. Cloudmesh client was used to install and deploy Hadoop clusters on Chameleon Cloud. Ansible was then configured to perform a set of tasks on the namenode of the server. File movement was handled using Ansible playbooks. The final Pig script was run on the namenode Hadoop server using automated Ansible scripts.

Server	Ubuntu14.04
VCPU	2
Ram	4 GB
Disk	40 GB

Table 1. Virtual Machine Configuration

The configuration for the cluster deployed is given in Table 1. The project uses three virtual machines defined on Chameleon Cloud. One of them acted as namenode while the rest acted as datanodes.

4. WORK FLOW

The deployment of clusters was done using Cloudmesh client's "cm deploy" command. This automatically created a three node Hadoop cluster and enabled cross SSH between the nodes so that they will be able to communicate with each other. Next step was to use Ansible to transfer the data files and Pig script to cloud. Once this data transfer was done, Ansible scripts were used to move the files from cloud to hdfs dfs repository. The pig script was run using Ansible from local system and timed for benchmarking. The final step was to transfer the files back from hdfs to local system using Ansible.

5. EXPERIMENTS AND RESULTS

Multiple experiments were conducted using different sizes of input data and the time results were used to benchmark Chameleon Client for this particular analysis. The data size was taken for predicting the most likely to be clicked for 100, 2000, 5000, 10000 webpages. Each document ID corresponds to a page that needs predictions and a list 6 Ad_id are predicted for each document_id based on relevance factor calculated. The results are presented in 4, **??**, **??**.

The graphs are plotted for time versus number of webpages to which predictions are made (# documents). It is evident from 4, the time required to predict is linearly proportional to number of webpages. This shows that the time complexity is O(n) where n is number of document ids.

6. CONCLUSION

Apache Hadoop technologies can be used to process large volumes of data parallely to make quick analyses. Apache Pig is an addon for Hadoop, which is a high level language for writing queries to process data. Apache Pig is efficient in query optimization and enables parallelism for its processing. The current project uses Apache Hadoop and Pig to process large datasets related to online ad prediction. Online advertisement is one space where large amounts of data is collected and it demands the use of Hadoop like parallel processing systems to get meaningful insights from the data.

As we can see from the results, Apache Pig was able to handle reasonably large sized files in short times. The processing time increases as the input query size increases. This is a small cluster of three nodes and it is able to process such large volumes of data efficiently. These nodes can be horizontally expanded to

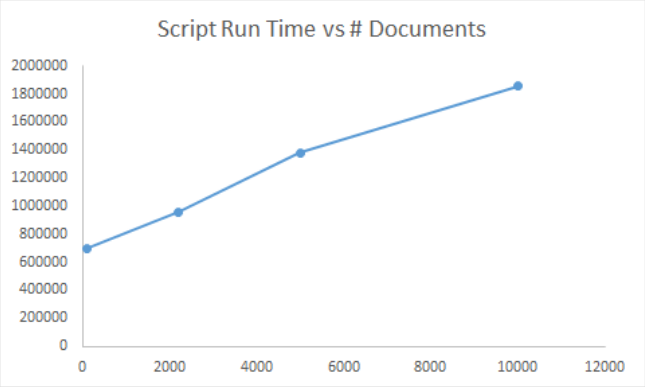


Fig. 4. Benchmarking Results: Time(in Milliseconds) taken for predicting Best 6 Ads for webpages

utilize the full extent of Hadoop's parallel processing capabilities in order to scale up.

ACKNOWLEDGEMENTS

The authors thank Professor Gregor Von Laszewski and all the AIs of big data class for the guidance and technical support.

REFERENCES

[1] D. Mielach, "Americans spend 23 hours per week online, texting," Web-page, accessed mar-13-2017. [Online]. Available: http://www.businessnewsdaily.com/4718-weekly-online-social-media-time.html
[2] marketingterms.com, "Clickthrough rate definition," Web-page, accessed mar-13-2017. [Online]. Available: http://www.marketingterms.com/dictionary/clickthrough_rate/
[3] Wikipedia, "Click-through rate," Webpage, accessed Mar-13-2017. [Online]. Available: https://en.wikipedia.org/wiki/Click-through_rate
[4] Outbrain, "Data introduction," Webpage, accessed Mar-13-2017. [Online]. Available: https://www.kaggle.com/c/outbrain-click-prediction/data
[5] Ansible, "How ansible works," Webpage, accessed Apr-24-2017. [Online]. Available: https://www.ansible.com/how-ansible-works
[6] G. von Laszewski, "Quickstart cloudmesh client," Webpage, accessed Apr-24-2017. [Online]. Available: http://cloudmesh-client.readthedocs.io/en/latest/quickstart.html
[7] Teradata Corporation, "Intro to hdfs and mapreduce," Webpage, accessed Apr-24-2017. [Online]. Available: https://www.thinkbiganalytics.com/2013/07/12/intro-hdfs-mapreduce/
[8] B. Proffitt, "Hadoop: What it is and how it works," Webpage, accessed Apr-24-2017. [Online]. Available: http://readwrite.com/2013/05/23/hadoop-what-it-is-and-how-it-works/
[9] S. P. Bappalige, "An introduction to apache hadoop for big data," Webpage, accessed Apr-24-2017. [Online]. Available: https://opensource.com/life/14/8/intro-apache-hadoop-big-data
[10] Apache Software Foundation, "Welcome to apache pig," Webpage, accessed Apr-24-2017. [Online]. Available: https://pig.apache.org/

AUTHOR BIOGRAPHIES

Sahiti Korrapati is pursuing her MSc in Data Science from Indiana University Bloomington

Flight Data Analysis Using Big Data Tools

ANVESH NAYAN LINGAMPALLI[1,*]

[1] *School of Informatics and Computing, Bloomington, IN 47408, U.S.A.*
[*] *Corresponding authors:anveling@indiana.edu*

project-S17-IR-2016, May 4, 2017

Analysis of flight data provides insights on the United States of America's Airline data by using Hadoop in the cloud environment. The On-time performance of flights operated by large air carriers are tracked and made as a report, Air Travel Consumer Report, which is a big data set. Hive component of Hadoop ecosystem, is utilized to process the big data in distributed environment. Efficient accessing and processing of the user queries is acheived by this analysis on flight data.

Keywords: Apache, Hive, Ansible, Pig

https://github.com/cloudmesh/classes/blob/master/project/S17-IR-2016/report/report.pdf

1. INTRODUCTION

Real world data is large and growing exponentially from several years. This data can be in structured or unstructured format which is popularly known as Big Data[1]. Aviation industry manages enormous amount of data, which consists of the information regarding the delayed, cancelled, diverted or on-time flights by large air-carriers[2]. This is essentially a big-data set, where statistics are publicly available as the Air Travel Consumer Report.

Access to multiple clouds is provided by a cloud manager known as Cloudmesh Client[3]. With the help of this cloud manager, Hadoop cluster is built with neccessary add-ons such as Apache Spark[4], Hive[5], Pig[6]. Cloudmesh Client is also used in the deployment of the cluster on various clouds such as Chameleon cloud[7] or Jetstream[8] cloud. Deployment part is automated with the help of ansible[9] scripts where data is extraced, stored and analysed automatically to produce the results.

Big Data analysis of this data will provide a consistent understanding and importance of the given data. With 35 million flight departures per year, data is critically important for any planning decision made by airlines and airports. The results of analysis has benefits which can help airline operations to predict and reduce redundance[10].

2. INFRASTRUCTURE

Cloudmesh client and Chameleon cloud forms the infrastructure of this analysis project.

Cloudmesh client is a toolkit which provides a client interface for accessing different clouds and clusters. It includes a commandline interface to provide abstraction from backend databases. Simplicity is one of the key and powerful features of the cloudmesh client. It makes switching between various clouds easy by providing a convenient programmable interface.

Chameleon cloud is a large scale platform which is an open research community for development of programmable cloud services. It provides wide range of services such as Infrastructure-as-a-service, platform-as-a-service and delivery of high functioning cloud environment.

3. DEPLOYMENT TOOLS

3.1. Ansible

Ansible is an open source software that provides automation for configuration management and application deployment. It facilitates a simple automation platform that makes the application easier to deploy. It also handles ad-hoc task execution and multinode orchestration. Ansible is a software which has an agent-less architecture. This is because there are no deamon processes running in the background. Components of Ansible comprises of modules, playbooks,inventory and ansible towers. Modules can control system resources like services, packages, or files. Inventory is configuration file that reflects the nodes that are available for access. Nodes are represented by hostnames or IP addresses. Playbooks are Ansible's configuration, deployment and orchestration language. These are in the YAML format. Playbooks are generally used to manage configurations and deployment on remote machines. Ansible tower makes Ansible a center for automating tasks by providing a web based console[11].

4. ANALYSIS TOOLS

4.1. Apache Hive

Hive is one of the ecosystems in Hadoop[13] framework which is built to analyze the data on hadoop cluster. Syntax of Hive is

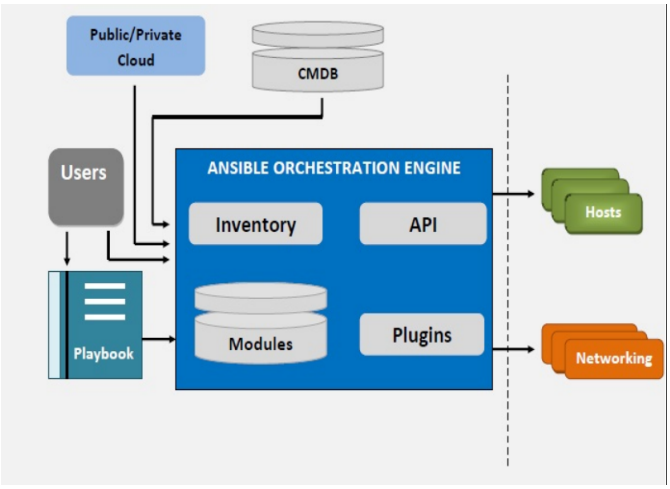


Fig. 1. Architecture of Ansible [12]

based on SQL[14], which is also known as HiveQL. MySQL or PostGreSQL[15] can be used for implementation of the queries. Hive provides tools which enable easy data extraction, transformation and data loading. Files can be stored in Hadoop Distributed File System(HDFS)[16] and accessed by Hive efficiently.

Schema of the Hive tables is stored in Hive Metastore. Metastore holds the information about tabes and partitions which are present in the data warehouse. In Hive the default Metastore is DerBy Database, which is a relational database management system provided by Apache Software. There are two components of Hive, HCatalog and WebHCat. HCatalog is a storage management layer for Hadoop which provides data processing tools such as Pig and MapReduce. WebHCat provides a service that is used to run Hadoop MapReduce, Pig, Hive jobs using REST interface[17].

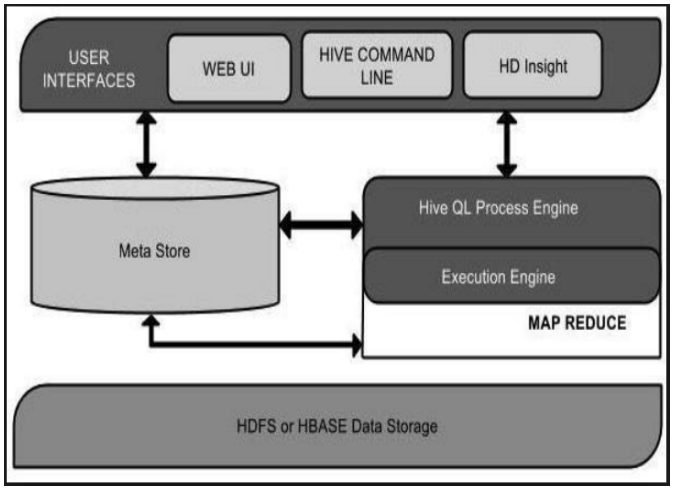


Fig. 2. Apache Hive Architecture [18]

Hive's SQL provides the basic SQL operations, such as,

- Filtering the rows from the table using WHERE clause.
- Selecting columns from table using SELECT clause.
- Joining two tables
- Aggregations of the data using 'group by' clause.
- Storing the results in a hdfs directory.

4.2. Hadoop Distributed File System

Hadoop Distributed File System provides a distributed file data storage system which spans large clusters of servers. It distributes -the storage and computation across many servers which maintains economy of the storage[16].

The file system is designed to be fault-tolerant. When HDFS takes data, the information is broken into pieces and distributed them to different nodes in a cluster. This provides parallel processing on clusters. MapReduce programming model is implemented when the applications are executed.

HDFS uses master/slave architecture, where each cluster consists of a Namenode, which manages file system operations and supports Datanodes, which manage data storage on individual compute nodes. Namenodes monitor the Datanodes in creating, deleting and replicating data by mapping them into data nodes.

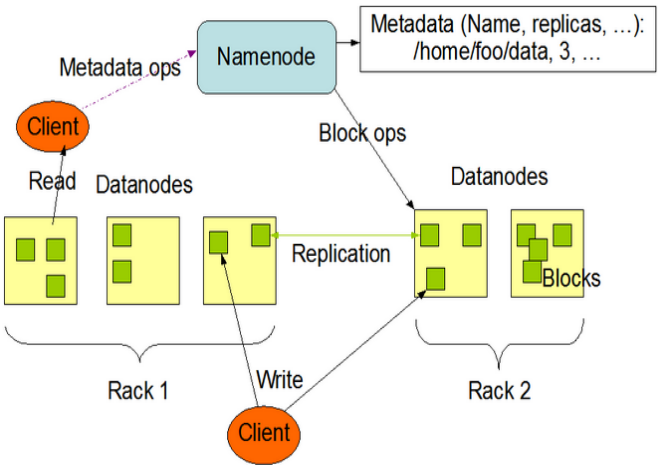


Fig. 3. Architecture of Hadoop Distributed File System [19]

5. IMPLEMENTATION

Implementation of Hive to perform data analysis consists of the following steps.

- Virtual machines are created on Chameleon Cloud with the help of Cloudmesh Client.
- Hadoop cluster is deployed on Chameleon cloud.
- Flight Data is loaded into HDFS.
- Data is then transfered to Hive tables.
- Finally, analysis is done on Hive tables using HiveQL, this can either be done using Hive interface or by installing PostgresQL interface.

5.1. Analysis in Hive Query Language

The following are the queries in Hive QL, which are similar to SQL statements.

What are the total number of flights which are cancelled?

```
SELECT year, month, count(cancelled) as Cancelled-flight
FROM Airline
WHERE cancelled = 1
GROUP BY YEAR, MONTH
ORDER BY YEAR, MONTH
LIMIT 20
```

What are the total number of flights which are diverted?

```
SELECT year, count(diverted) as Diverted-flights
FROM Airline
WHERE diverted = 1
GROUP BY month
ORDER BY month
LIMIT 10
```

Similarly, analysis of the data is done where the queries are as follows.

What is effect of flight distance on cancellations?

What is the effect of flight distance on average departure delay?

What is the monthly average departure delay?

What is the yearly average departure delay?

6. TECHNOLOGIES

- Distributed Computation and Storage:- HDFS and Hive
- Development:- PostgreSQL and Java
- Deployment:- Ansible

7. BENCHMARKING

After the deployment stage, benchmarking is implemented. This process is important as it evaluates the performance of the application and the system. This project is implemented on the chameleon cloud and the results are observed. Different cloud environments are used for creting these benchmarks

Characteristics of Chameleon cloud on which the analysis is implemented are,

Chameleon	VCPU	RAM(GB)	Storage(GB)
m1.small	1	2	20
m1.medium	2	4	40
m1.large	4	8	80

Fig. 4. Different types of chameleon clouds

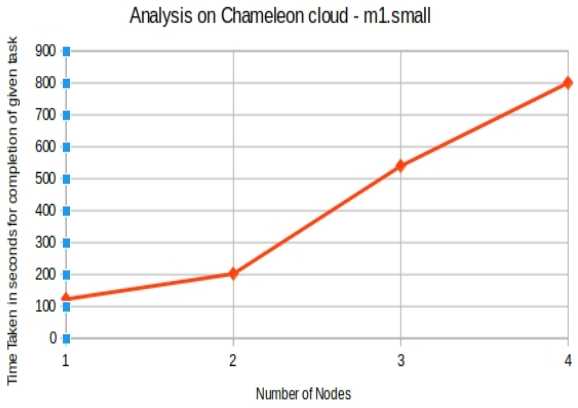


Fig. 5. Analysis time taken for a HiveQL-m1.small

The following are the observations from the analysis on chameleon cloud of different flavors.

These are the observed results when m1.small cloud is used for analysis As the number of nodes in a cluster increase, the amount of time the task takes to complete is increasing. This trend is also observed when m1.medium and m1.large flavors of Chameleon cloud are used.

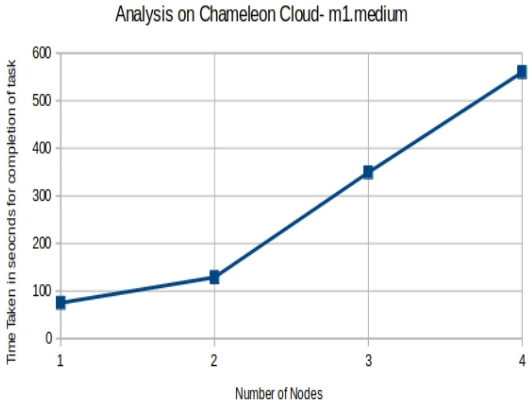


Fig. 6. Analysis time taken for a HiveQL-m1.medium

8. CONCLUSION

Deployment and the analysis of the flight data is implemented by using Apache Hive, Cloudmesh and Ansible automation. The results obtained from the qeries in Hive environment gives insights on the available flight data. Hive uses map and reduce functions internally which is taken care of Hadoop system. Observations such as number of flights cancelled, number of flights departed from an airport are made from Hive Query Language. From these results trends and patterns are observed which provide detailed analysis of the data.

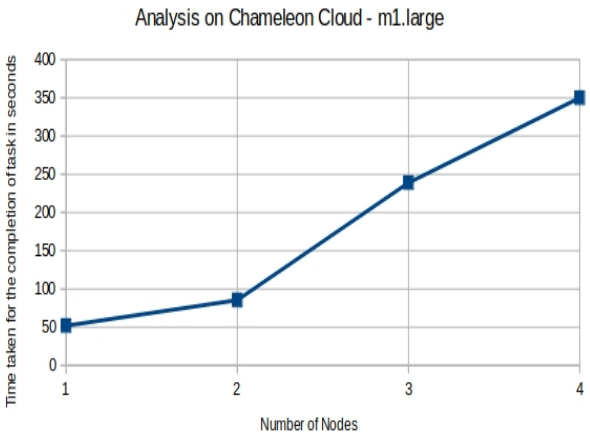


Fig. 7. Analysis time taken for a HiveQL-m1.large

9. ACKNOWLEDGEMENTS

This project is undertaken as part of the I524: Big Data and Open Source Software Projects coursework at Indiana University. We would like to thank our Prof. Gregor von Laszewski, Prof. Gregory Fox and the Associate Instructors for their help and support.

REFERENCES

[1] "Big data," Web Page. [Online]. Available: https://www.sas.com/en_us/insights/big-data/what-is-big-data.html
[2] "Aviation analysis," Web Page. [Online]. Available: https://aviationanalytics.com/airport-analytics/data-analysis/
[3] "Cloudmesh," Web Page. [Online]. Available: https://cloudmesh.github.io/
[4] "Apache spark," Web Page. [Online]. Available: http://spark.apache.org/
[5] "Apache hive," Web Page. [Online]. Available: https://hive.apache.org/
[6] "Apache pig," Web Page. [Online]. Available: https://pig.apache.org/
[7] "Chameleon cloud," Web Page. [Online]. Available: https://www.chameleoncloud.org/
[8] "Jetstream cloud," Web Page. [Online]. Available: https://jetstream-cloud.org/
[9] "Ansible webpage," Web Page. [Online]. Available: https://www.ansible.com/
[10] "Big data in aviation," Web page. [Online]. Available: http://apex.aero/2016/11/30/big-data-aviation-industry-case-becoming-data-driven
[11] "Ansible : Tutorial," Web Page. [Online]. Available: https://serversforhackers.com/an-ansible-tutorial
[12] "Architecture of ansible," Web Page. [Online]. Available: https://devops.com/ansible-automation-provisioning-configuration-management/
[13] "Apache hadoop," Web Page. [Online]. Available: http://hadoop.apache.org/
[14] "Structured query language," Web Page. [Online]. Available: https://en.wikipedia.org/wiki/SQL
[15] "Postgresql," Web Page. [Online]. Available: https://www.postgresql.org/
[16] "Hadoop distributed file system," Web Page. [Online]. Available: https://hortonworks.com/apache/hdfs/
[17] "Information on hive," Web Page. [Online]. Available: https://hortonworks.com/hadoop-tutorial/how-to-process-data-with-apache-hive/
[18] "Architecture of hive," Web Page. [Online]. Available: http://blog.cloudera.com/blog/2013/07/how-hiveserver2-brings-security-and-concurrency-to-apache-hive/
[19] "Architecture of hdfs," Web Page. [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#NameNode+and+DataNodes

S17-IR-2039/report/report.pdf not submitted

Real-time Analysis and Visualization of Twitter data

SOWMYA RAVI[1,*], SRIRAM SITHARAMAN[2], AND SHAHIDHYA RAMACHANDRAN[3]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[2] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[3] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*] Corresponding authors: sowravi@iu.edu, srirsith@iu.edu, shahrama@iu.edu

project-001, May 4, 2017

A real-time system has been developed which extracts live data from twitter, processes the data, calculates a sentiment score based on ensemble of Naive Bayes and Textblob and finally visualizes the sentiment across time and geographical locations. The system has been deployed in chameleon and jetstream cloud platforms. Live data from twitter is injected into the system using tweepy package in Python. Naive Bayes classifier was built in a two-node Apache Spark cluster by training on 3.5 Million tweets. An ensemble of the Naive Bayes classifier and the built-in classifier of Textblob module (Python) were used to determine the sentiment score. The results of the Natural Language processing algorithms were visualized in batches using d3.js and Highcharts.

Keywords: Real-time streaming, data visualization, Twitter, Natural Language Processing

Report:https://github.com/cloudmesh/sp17-i524/tree/master/project/S17-IR-P001/report/report.pdf
Ansible:https://github.com/sriramsitharaman/sp17-i524/tree/master/project/S17-IR-P001/code/ansible
PySpark:https://github.com/sowravi/data_code.git
Visualization:https://github.com/sriramsitharaman/visualization

CONTENTS

1. INTRODUCTION

In 1969, Drs. Jerry Boucher and Charles E. Osgood, psychologists at the University of Illinois, proposed the 'Pollyanna Hypothesis' which asserts that "there is a universal human tendency to use evaluatively positive words more frequently and diversely than evaluatively negative words in communicating" [1]. Such theories were hard to validate due to the absence of significant data and the lack of generality. With Social media turning into

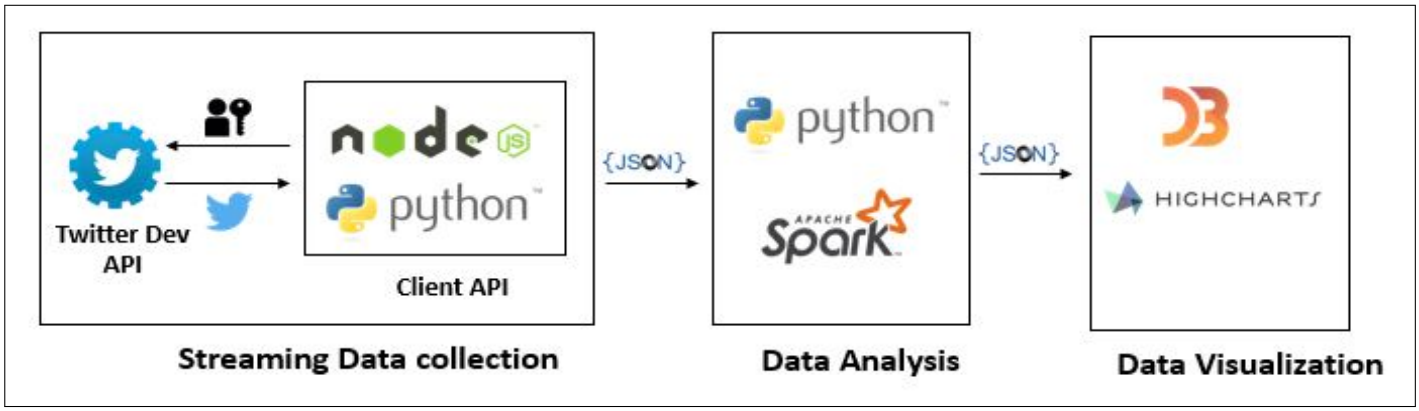


Fig. 1. System Architecture

the primary platform where people express on a day-to-day basis these claims can be analysed by sampling a portion of the data and applying Natural Language Processing algorithms to determine positivity/negativity of this data. The first step in this process involves extraction of data from Social media using the corresponding APIs. Once the data has been extracted, it is cleaned and restructured to make it suitable for analysis. The results of the Analysis are then visualized in a dashboard to better understand the outcomes.

Microblogging website like Twitter is used for analysing behaviour of people because it has emerged to be one of the predominant platforms where people express their opinions about current issues, complain about products, discuss details of ongoing events etc. Twitter data also aids in understanding behavioral patterns across diverse demographics - location, gender etc. Twitter generates nearly 200,000 tweets in less than a minute [2]. This large sample of data can then be used to test the hypothesis. Big data technologies prove to be particularly useful in storing, processing and analysing such large data sets. It also makes it possible to setup real-time systems that can output results with a latency of very few seconds.

he data for the system was extracted from Twitter using it's API through Python. This data is processed and manipulated in Python and PySpark which is finally visualized using D3.js and Highcharts. The remainder of the paper is organised as the System Architecture, Work flow, Deployment, Benchmarking and Conclusion.

2. SYSTEM ARCHITECTURE

The streaming pipeline for analysing the data consists of different components to aid in different stages of the analysis. The architecture of these components have been elaborated in this section. The system architecture is shown in the Figure 1.

2.1. Python

Python was primarily used for tweet extraction, pre-processing and analysis. Various packages were used to aid in the process. 'Tweepy' package[3] was used for extracting tweets directly from Twitter API. The user has to create an application by logging into the tweepy portal and using the unique access tokens and cosumer keys he/she can access the data from Twitter. OAuthHandler module assists in accessing the data securely. StreamListener module is used to directly extract the tweets from Twitter. 'JSON' library [4] was used to handle the JSON data output from tweepy and to provide as input to highcharts and D3.js. 'Pandas' citewww-pandas, which is Python's data analysis library was used for manipulating the data while pre-processing. 'Numpy' [5] package was used for data handling. 'zipcode' [6] package was used to extract geographical data for visualizing on maps. 'NLTK' [7] package was used for building classification model through 'Textblob' module [8]. 'wordcloud' [9] and 'matplotlib' [10] were useful in creating wordclouds of the positive and negative tweets.

2.2. Apache Spark

Spark provides a distributed cluster computing framework for processing large amounts of data. The architecture of spark is shown in Figure 2 The Spark system encompasses a Master/ Driver and Workers. The Master contains the Spark context which serves as the execution environment for Spark. Each worker node contains one or more executors which run processes. The driver manages all the workers via an external processes like YARN or Mesos [11]. Spark supports multiple languages which include Scala, Java, Python and R. In addition to the spark core framework , it has an Application Programming Interface for Streaming (Spark Streaming), SQL(Spark SQL), Machine Learning(Spark MLLib) and graph computation (Spark GraphX).

Spark is built on Hadoop and it uses the Hadoop Distributed

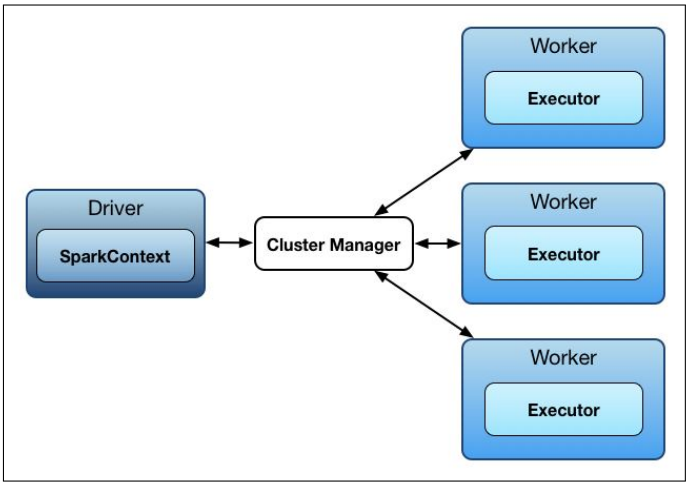


Fig. 2. Architecture of Apache spark

File System(HDFS) for Data Storage. It can also access diverse

data sources including Cassandra, HBase, and S3. Spark stores the data in the form of Resilient Distributed Data(RDD). RDD is the basic data storage unit in spark. They are read-only, logically partitioned collection of records and they are stored in a fault-tolerant manner. Each partition in an RDD may be stored and computed at a different node. RDDs are created in two ways: one is by paralleling an existing collection and the second is by referencing an existing data set[12].
The spark can be operated in either Standalone mode or in cluster mode. While operating in cluster mode, spark makes use of YARN framework for resource management and distributed computing. The YARN is responsible for allocating tasks, delivering data to the worker nodes and collecting the results of the process.

2.3. Node.js

Node.js is an open source, asynchronous, Javascript run-time environment which enables JavaScript to be used for server side scripting [13]. It has an event driven architecture that focuses primarily on throughput and scalability of web applications. It works based on callbacks rather than threading to signal completion of tasks. Though it does not support threading it allows ,multi-core usage through child processes. The cluster modules aids in balancing load across the cores. Node.js combine JavaScript with Unix network programming in order to provide concurrency without compromising on performance of the server side program. It is comparable to systems like Ruby's Event Machine or Python's Twisted. In Node.js there is no mechanism to explicitly call the event loop start. The event loop is entered after the input script is executed and exited when there are no more callbacks. It is commonly used for running server side web applications to produce dynamic web page content before the page is sent to the user's web browser.

2.4. D3.js

D3.js is a JavaScript library for manipulating website elements based on user data. D3 uses using HTML, SVG, and CSS for rendering the web page [14]. It combines powerful visualization components thereby providing a data-driven approach to web-page DOM (Document Objects Model) manipulation. D3.js offers flexibility by exposing the full capabilities of web standards such as HTML, SVG, and CSS. With minimal overhead, D3 is extremely fast, supporting large data-sets and dynamic behaviors for interaction and animation. It works by selecting elements of a HTML webpage and adding an svg element on top of it. The SVG element can vary through a series of objects like a rectangle, circle, text etc. Each of these objects has their own attributes defining their shape, label, position etc. in the web page. These shapes can also be styled by using a style class in a CSS file. Hence, D3.js in a nutshell accesses HTML, CSS and SVG components of a web page which makes rendering visualization of data in an efficient, simple and fast manner.

2.5. Highcharts

Highcharts is an online Data Visualization product created by the Highsoft team based out of Norway [15]. It is a charting library written entirely in JavaScript. It allows creation of interactive visualizations that can be hosted on web pages or web applications. The data to be visualized is input as JSON file. as input data Highcharts currently supports line, spline, area, area-spline, column, bar, pie, scatter, angular gauges, area-range, area-spline-range, column-range, bubble, box plot, error bars, funnel, waterfall and polar chart types [16].

3. WORK FLOW

3.1. Cloudmesh Client

Cloudmesh is a cloud management client that provides seamless integration to multiple clouds from command line. Some of the features of cloudmesh include heterogenous cloud integration, deployment of hadoop cluster with add-ons such as spark and pig [17]. It also enables automated builds

3.2. Ansible

Ansible is a powerful IT automation engine that helps in the deployment of of applications. It doesn't use any customized security infrastructure that and it is very easy to deploy. The YAML, a simple language, is used to write ansible scripts. The main component of the ansible is a playbook which is YAML file that contains all the processes and applications that have to be deploys in the remote system [18].
Ansible pushes small chunks of code called modules that are executed on the nodes. These modules are executed over ssh by default and removed once they are completed. Also, these modules can reside anywhere and d not require the use of database systems , servers or daemons [18].
Ansible supports the use of ssh keys as well as passwords. However, ssh keys are relatively simple to use. Another noteworthy feature of ansible is that root logins are not required and later one can sudo to any user. The Ansible authorized key module is an efficient way to control other hosts. Additional options such as kerberos or identity management systems are also available for use [18].
The Playbook is the heart of an ansible work flow. The playbook enable the user to control the infrastructure details of the project such as the number of machines to be considered for the task and the processes to be deployed in each of those machines [18].
A basic ansible playbook is shown in Figure 3 The list of hosts

```
---
- hosts: webservers
serial: 5 # update 5 machines at a time
roles:
- common
- webapp


- hosts: content_servers
roles:
- common
- content
```

Fig. 3. Structure of an Ansible Playbook [18]

is usually provided in a file and each group of hosts is usually given a name that is later reference in the playbook. While running an ansible playbook, the inventory/host file is also passed in the command. Logging into machines do not require an SSL signing in. A sample ansible inventory is shown in Figure 4.

Fig. 4. Structure of an Ansible Inventory [18]

3.3. Tweet Extraction

Python is used as the platform for collecting real-time data from Twitter. Using the 'tweepy' package, a tweet extracting module was built which authenticates the collection of data from Twitter's API. This is done by using the 'Consumer key' and an 'Access token' generated by Twitter's developer API. This Consumer key and access token are unique to the application created by the user. Of the total number of tweets authored in Twitter at a given time, only 1% -2% of the tweets can be extracted. Of these tweets that were extracted, only 1% -2% of the tweets are geo-tagged. While extracting the tweets, filter was applied on Language to extract only those that were authored in 'English' language. Tweets were collected at the rate of generation as a JSON object in Python. the data was not stored in a RDBMS/ NoSQL database since data files in the Gigabytes range slowed down the system when implemented in Chameleon cloud platform. Instead, batch processing technique was used, where the tweets were extracted and processed in batches. The results of analysis of this batch of tweets were then visualized. The results of each of the subsequent batches were aggregated with the previous batch in Python. The visualizations were also subsequently updated.

3.4. Tweet Pre-Processing

Since the extracted tweets were in JSON format, they had to be pre-processed in order to be used for analysing and calculating sentiment scores. The various steps in pre-processing have been elaborated in this section.

3.4.1. Parsing

The data had to be parsed to get convert the data in JSON format to a pandas data frame with columns populated with : textual content, location, country code, country name, state code, time of generation etc. The missing values were replaced with 'NA's.

3.4.2. Tokenization

A custom tokenizer was built that does not separate the hashtags # from the word. Similarly, @ and the textual part of the mentions were retained as a single entity. Other special cases included retaining all the characters in emoticons together as a token, extracting URLs together etc. After testing the performance of the tokenizer manually, the datasets were given as input to the tokenizer. Files with a single token on each line and an empty line following each tweet were created.

3.4.3. Word Vector Creation

For each of the tweets, word vector files were created which includes the tokens from the tweets as individual features and values were given based on their presence in each tweet. The value for that respective column was set to be 1 if the word/token was present in the tweet and 0 otherwise.

3.4.4. Feature Engineering

Two lexicons were provided: Arguing Lexicon and MPQA subjectivity lexicon. Features were created from each of these lexicons for the train and test data-set of each target.

- **Arguing Lexicon:** Arguing lexicon [19] is a hand built lexicon built by Somasundaran, et al., contains 17 ".tff" files describing patterns that represent arguing. Each file contains patterns that are specific to one type of argument. For instance, "authority.tff" contains patterns that might occur in the text that involves a person or an entity exhibiting authority in some manner. There are certain areas in the pattern files that contain words that start with "@". The corresponding definition for these words were given in 5 macro files. Modals, Spoken, Wordclasses, pronoun and intensifiers are included in the 5 macro files. For example, in "emphasis.tff" there is a pattern "(@GONNA)", the value of which is present in the "modals.tff" macro file.

  "@GONNA": ["am going to"," are going to"," is going to"," am gonna"," are gonna"," is gonna"]

  Here, "@GONNA" was replaced by the corresponding list of values shown above. It can be any one the listed values. All the 17 files were read in python as a list of patterns. Macro files were used to replace the macros present in these 17 files wherever necessary. Regex package in python was used to read and compile each pattern in each of the 17 files as regular expressions. 17 new features were created and 0/1 values were assigned if a text contains at least one regular expression from the list for each lexicon file. The created feature would have the value 1 if any one of the regular expression has a match with the input tweet, otherwise 0. This process was repeated for all the targets' train and test data-sets.

- **MPQA subjectivity lexicon** was also used in the process of creating new features. This lexicon contains a clue file which was developed by Riloff and Wiebe, 2003 as a part of their work in subjectivity expressions [20]. The clues in the file were provided in the following format:

  type=strongsubj len=1 word1=abuse pos1=verb stemmed1=y priorpolarity=negative

  A clue that is subjective in most context is considered strongly subjective (strongsubj), and those that may only have certain subjective usages are considered weakly subjective (weaksubj). "Word1" denotes the token or stem of the clue, whereas "pos1" denotes the part-of-speech of the corresponding word. Apart from subjectivity, the polarity of the word whether its positive, negative, neutral or both is given by "priorpolarity".

3.5. Classification Model

3.5.1. NLTK - Textblob

Textblob is a python wrapper(for python 2 and python 3) class of the NLTK package. It provides various modules for utilizing

the functions of NLTK. Hence, the usage requires NLTK and all NLTK corpora to be installed initially. The sentiment analyser tool in textblob gives the polarity when a sentence is given as input. It can recognize several different languages.

3.5.2. PySpark with NLTK

Spark framework comes with a powerful Machine Learning API called MLLib. The MLLib contains predefined functions for popular machine Learning Algorithms. The NaiveBayes classifier was used along with the python's NLTK toolkit to get the sentiments associated with each tweet. The resulting model was stored in a the local file system. Each subsection describes the functions and modules used to build the Naive Bayes Classifier.

The output of tweet extraction is read by the Spark Context using a JSON reader. The data is then pre-processed and cleaned by removing the stopwords from the tweets. The tweets then tokenized using the NLTK tokenizer. The tokenized tweets are used to create feature vector with features using the hashing trick. Hashing maps the data of an arbitrary size to a fixed size which is 50000 in this case. A Labelled Point vector which is a type of RDD with the label and feature vector as a key value pair. The Labelled Point vectors are useful in training supervised algorithms in MLLib. Sixty percent of the Labelled point data was used for training the Naive Bayes Classifier. The model obtained as a result of training is stored in the local file system. The model is tested on the remaining data. The predictions along with the correct labels are also stored locally in file. To process and manipulate the data the classic Hadoop functions such as Map, FlatMap and Filter were used. The functions are described in the upcoming sections.

- **Map**: Map applies a function to all the elements of a RDD. This is achieved in PySpark by using Lambda functions.
- **FlatMap**: Flat Map is similar in function to map however , it transforms an RDD to a collection and then back t an RDD. The results in a flattened output. The input and output RDDs can be of different sizes.
- **Filter** The filter applies a condition to every row of an RDD and retains rows that satisfies that condition.

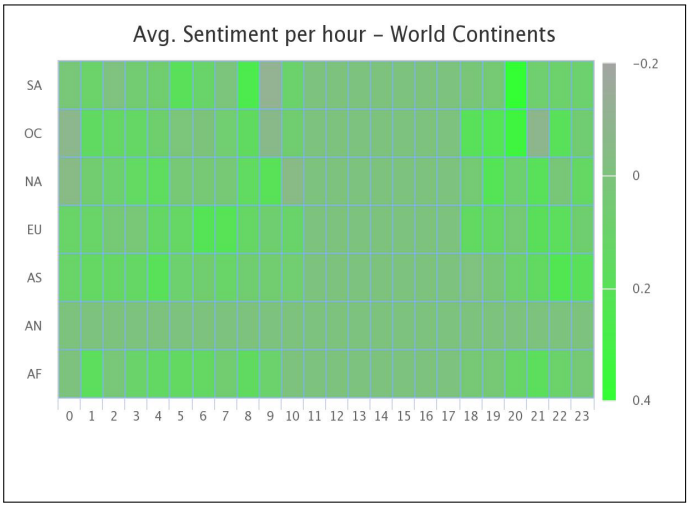


Fig. 5. Tweets Count - Time

3.6. Data Visualization

Data Visualization is an integral part of the sentiment analysis of Twitter data. Visualization tools such as D3.js and Highcharts [16] have been used to visualize the data.Visualizations created using Highcharts are updated as batches of tweets are extracted and processed in real-time. These charts change dynamically as each batch of tweet data gets processed over time. Visualization created using D3.js is directly connected to the Twitter API and thus it gets updated without any latency. A series of visualizations have been created in D3.js and Highcharts to analyze the data and project results of NLP analysis onto geographical and temporal spaces. The section below elaborates on the visualizations implemented.

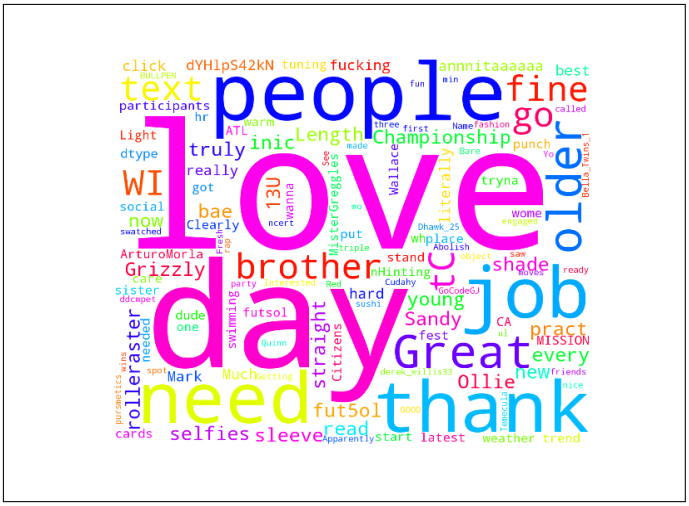


Fig. 6. Word Cloud for Positive tweets

3.6.1. Temporal Analysis

To analyze the data with respect to time, the tweets were aggregated with respect to the time stamp. The tweets generated all over the world were collected by filtering on 'English' language. The number of tweets generated from each country was plotted along the time axis to identify the period of time during the day when people tweeted the most. A matrix was created to understand the average positivity/ negativity/ neutrality of people around the world during different times of the day.

- **Count of Tweets generated vs Time:**
  The plot of the number of tweets generated in a given period of time is shown in figure 7. Line chart has been used because it clearly depicts the trend in number of tweets in a given second. The time (in UTC) is plotted along the horizontal axis and the number of tweets is plotted along the vertical axis. In the figure 7, the number of tweets have been plotted for the time period 19 : 54 : 30 to 19 : 58 : 00 on April 09, 2017. It is possible to select the time range for which we want to see the trend in more detail by clicking and dragging the mouse in the horizontal axis [21]. The tool-tip gives information about the exact time-stamp and the corresponding number of tweets generated at that time. The plot can be used to observe the peaks in activity in Twitter over a period of time.
- **Average Sentiment across Continents vs Time:**
  The chart giving the sentiment score as a function of continent and time is shown in the figure 5. Each block in this

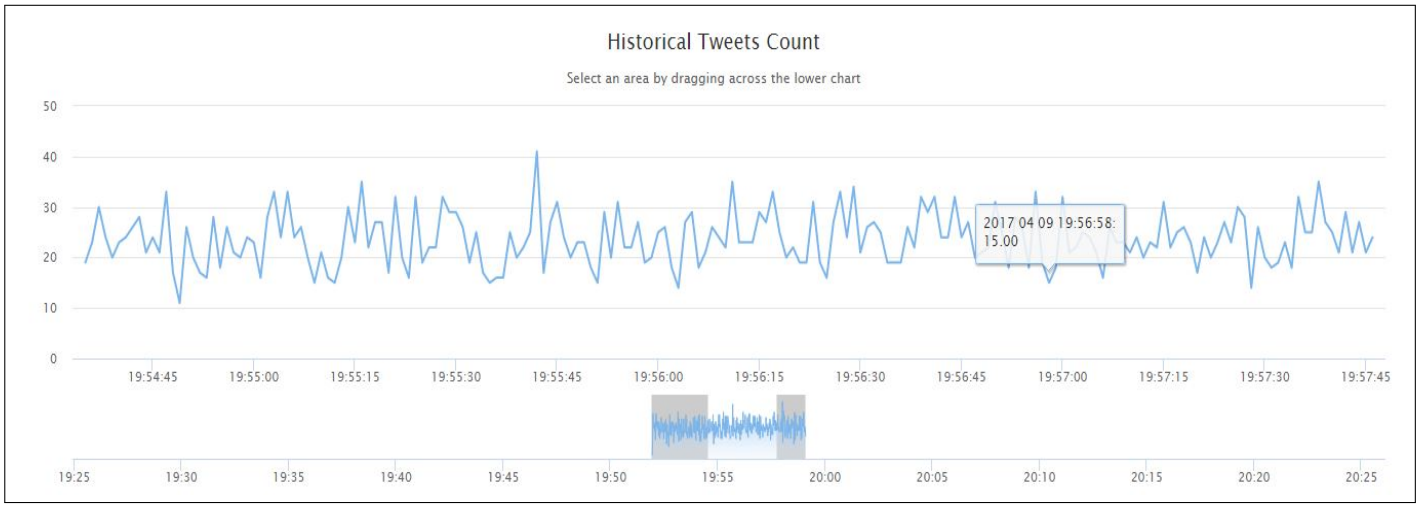


Fig. 7. Tweets Count - Time

matrix gives the average sentiment in a particular continent at a particular time [22]. The continents are given along the horizontal axis and the time periods are mentioned along the vertical axis. The tool-tip gives information about the time-period and the average sentiment score corresponding to each Continent. Since tweets are generated from as many as 180 countries, the analysis has been aggregated at the Continent level in order to obtain an overall picture. This chart depends on the sample of data chosen for plotting and hence it becomes essential to choose a stratified sample that is representative of the actual population. This can be ensured by choosing a significant number of tweets from each of the continents for the analysis. From the plot, it is seen that most tweets are generated during 7:00 AM to 9:00 AM and 19:0o Pm to 22:00 PM periods of time. Also, American continents are more active on Twitter as compared to the Asian and African continents.

3.6.2. N-grams Analysis

In a Natural Language Processing model, n-grams form important features in identifying the sentiment of a given text. N-grams can be word unigrams, bigrams, trigrams or they can also be character unigrams, bigrams, trigrams. Since the scale of Twitter data is very large word n-grams have been used for the sentiment analysis. As part of the n-gram analysis, Co-occurrence of word pairs in tweets and the most commonly occurring word unigrams in positive and negative tweets have been visualized.

- **Word Co-occurrence Matrix:**
  In Natural Language Processing, bigram combinations are used as features while developing a Sentiment Analysis model. This chart gives the co-occurrence patterns for the top 200 commonly occurring word pairs. All possible combinations of pairs of words belonging to a tweet were considered. Based on the number of times these word pairs had occurred in the tweets collected, the top 200 most commonly occurring pairs were determined. The probability co-occurrence of these word pairs were then projected onto a matrix [23] as shown in the Figure 9. This is an interactive plot where the order in which the words are displayed can be altered based on the Name, Frequency of occurrence, cluster, spectral [24] etc. The probability of co-occurrence

Fig. 8. Word Cloud for Negative tweets

has been differentiated by varying the intensity of blue used in the pixel corresponding to the pair of words. The darker the shade of blue greater is the probability of the two words occurring together in a given tweet. From the plot, it can be seen that the words like 'job', 'hiring', 'careerarc', 'work' etc. have high probability of co-occurring with each other.

- **Word Cloud:** Separate word clouds were created for tweets tagged as having a positive sentiment and those tagged as having a negative sentiment. Based on the NLP model built, a certain sentiment score was calculated for each tweet based on the n-gram features. This sentiment score value ranges from -1 to +1. Positive score indicates positivity of the tweet and negative score indicates negativity of the tweet. A score of '0' was considered as a neutral tweet. The tweets with negative and positive scores were aggregated separately and the word unigrams were plotted in a word cloud. The word cloud displays a collection of words occurring in the data [9]. The size of the word is directly proportional to the frequency of its occurrence in the data. From the positive wordcloud shown in Figure 6, it can be seen that words like 'love', 'thanks', 'fine' etc. have been

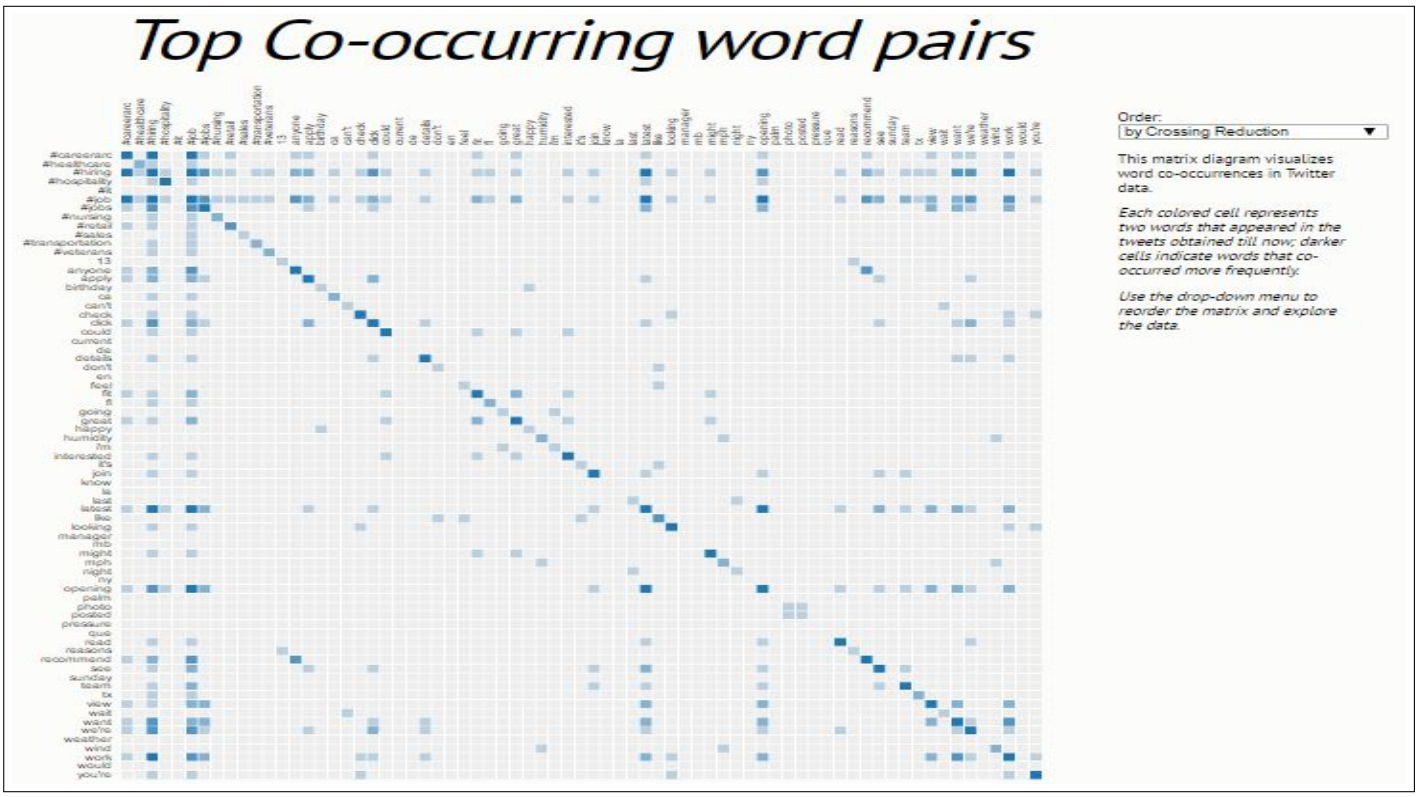


Fig. 9

frequently used. The negative wordcloud shown in Figure 8 clearly projects abusive words commonly used in tweets which in most cases are indicative of the sentiment.

3.6.3. Analysis for US-States

US is one of the major locations from where a significant number of tweets are generated. In the data that was collected from Twitter for the purpose of analysing sentiment, it was observed that majority of the tweets were from USA. Thus, Visualizations were created for analysing the amount of tweets generated and the average sentiment conveyed by these tweets across the states of the USA. Highmaps [25] were used for visualizing this data, where the values were projected onto the geographical map of US based on the latitude and longitude information. Heat maps have been used to represent this data, where the intensity of color is used to differentiate the value across multiple regions.

- **Number of tweets in US States:**
  From the Figure 10, one can see the number of tweets generated across the different states in the US. For the geo-tagged tweets available in the data, the country of tweet generation was extracted. Those tweets with Country code as 'US' were filtered and the state code was extracted for these tweets. The state code was then used to map the latitude-longitude information for each of the states. The number of tweets generated was aggregated at the State level. This was then used as JSON input to generate the graph in Highmaps [26]. The color scale shown at the bottom of the graph gives the intensity of Blue used for different frequency ranges. The number of tweets generated from each state has been projected on to the corresponding geographical location. This has been developed as an interactive graph, where particular regions can be zoomed to get a clearer view of the distribution. From the graph we can see that there is more activity along the coastal regions of the US as compared to Northern states of America. it can be seen that most tweets are generated from California on the West coast and majority of tweets are generated from the East coast of USA.

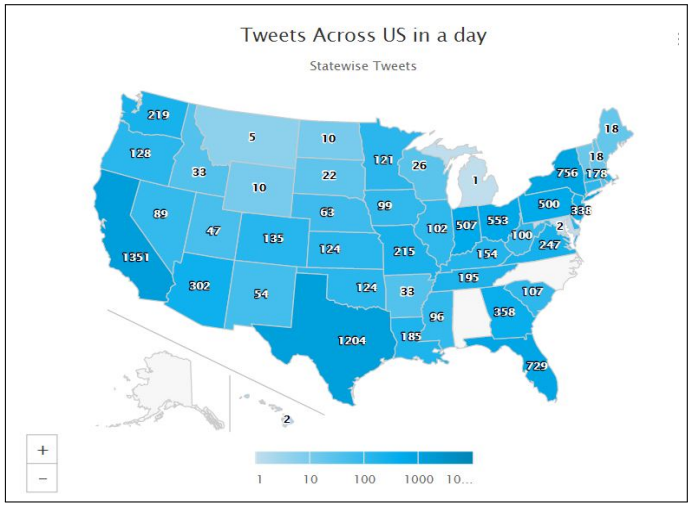


Fig. 10. Number of tweets in US States

- **Average Sentiment across US States:**
  The average sentiment across the various states of US was visualized as shown in the Figure 11. Data was extracted using a procedure similar to that explained in Section 4.3.1.

The average sentiment score was calculated by taking average of the sentiment scores of all the tweets from a given state. This graph was also developed as an interactive graph, where one can zoom into regions of interest for better understanding. The color corresponding to each class (Positive/Negative/Neutral) is shown at the bottom of the plot. In addition to creating a heat map for understanding the majority sentiment, pie charts were imposed onto the states to provide the percentage distribution of negative, positive and neutral tweets [27]. The heat map assigns the color of majority class to the state. In this way, the majority region of the pie chart will be indistinguishable from the background and one can see the distribution of other two classes in the same state. From the graph , it can be seen that majority of the states have neutral sentiment. The pie charts show that, though most states have a neutral sentiment, some states are less negative than the others.

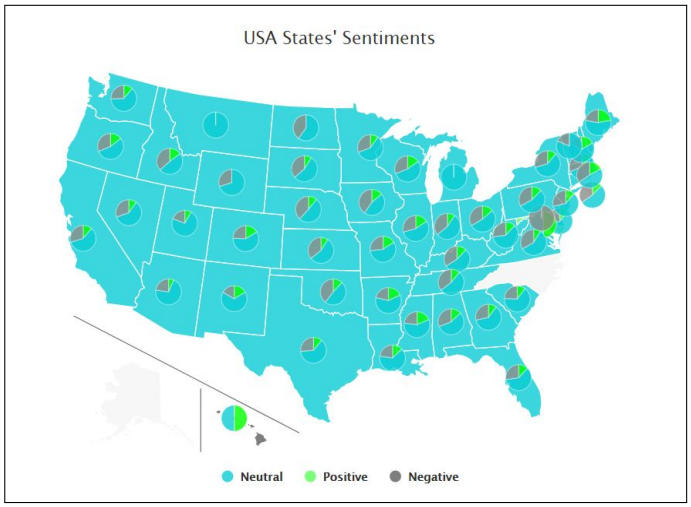


Fig. 11. Average Sentiment across US States

3.6.4. Analysis for World Countries

All tweets with geo-location information were analysed to understand the amount of tweets generated and their average sentiment with respect to countries. These values were then projected onto the world-map to identify regions of high activity and regions of positive, negative or neutral sentiment. The first graph displays real-time generation of tweets by highlighting the country from where it is generated. The latter is a heat map displaying the average sentiment across world countries in the map.

- **Number of tweets in World Countries:**
  Figure 12 shows the number of tweets generated in each country. This visualization was developed as a combination of bar chart and the geographical map. It was created using D3.js by obtaining the JSON data from node.js [28]. The visualization directly connects with the Twitter API in real-time with a delay of 2-3 seconds. Each geocoded tweet is visualized as a point of longitude and latitude on the map. When a tweet is generated, it is indicated by animated concentric circles around that point. After the animation ends it is marked by a dot with color corresponding to the color of the country in the bar chart. In parallel, a counter tracking number of tweets generated from the corresponding region is incremented and plotted on the bar chart. Only top 20 countries in terms of tweet generation are visualized in the bar chart. The country code is specified at the top of the corresponding bar. From the plot, it can be seen that maximum number of tweets are generated from US, which is in line with what was observed in the data collected.

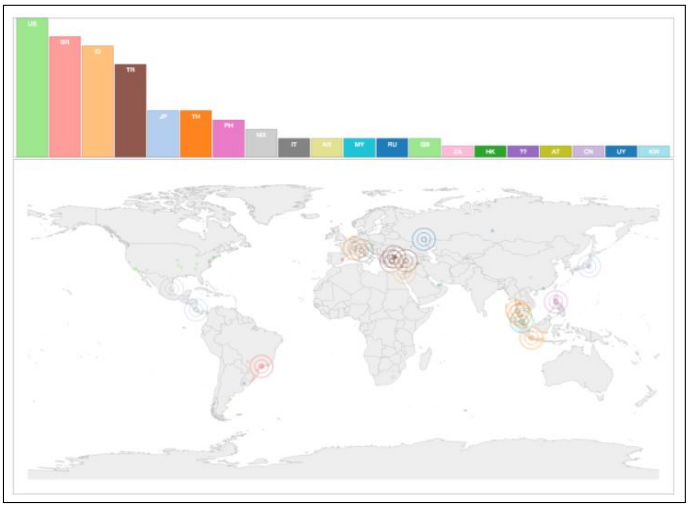


Fig. 12. Number of tweets in World Countries

4. DEPLOYMENT

4.1. Hadoop and Spark

Cloudmesh was used to deploy Hadoop and spark in the Virtual machines. A cluster with three nodes were created. One of the machines served as the master while the other two machines were the worker nodes. Two playbooks were created: the first playbook was used to install cloudmesh client and the second playbook was used to deploy a hadoop cluster with spark as an add-on. An overview of the tasks performed by the ansible playbook that deployed a hadoop cluster with spark in 3 virtual mahines is detailed below.

1. Virtualenv and Virtualenv wrapper were installed
2. the python dependencies for cloudmesh were installed.
3. Dependencies for cloudmesh client were installed
4. Install cloudmesh client in the local system
5. Generate ssh key if it doesn't exits
6. A default cloud was selected
7. The ssh key was added to cloudmesh
8. The security group was uploaded into cloudmesh
9. A cluster with three nodes was defined
10. Hadoop was deployed using spark as an add-on
11. The cross ssh connection between the three nodes was verified to enable communication between the nodes.

4.2. Implementation of Naive Bayes Classifier using spark ML-LIb

The code for the Naive Bayes classifier was uploaded in a common repository in github. An ansible script was written that fetched the code and executed it inside the master node. The model and the classified results were saved inside the master node's Hadoop distributed File system. This file was later pushed to a git repository which further utilized the results to create visualizations using D3.js

4.3. Analysis of Tweets using Python

Python was used to extract tweets from the twitter and the following packages of python was used for performing the multiple text analysis mentioned

1. Tweepy
2. Pandas
3. TextBlob
4. Numpy
5. Zipcode
6. matplotlib
7. nltk

The version of Python used was 2.7x. **Ansible-node** role was used to install python and its dependencies, followed by the installation of pip which was used to install the python packages mentioned above.

4.4. Visualization components

Two components of visualization was deployed. One using nodejs and d3.js for visualizing the tweets in their geo location in a world map. Another is the set of charts and maps using Highcharts visualizing the results of the analysis performed in the previous step.Later part of the **Ansible-node** role was used to install nodejs and its dependencies, and start the python http server for viewing the visualizations.

Table 1. Chameleon cloud Specification

Value	Chameleon
group	default
secgroup	default
image	Ubuntu-Server-14.04-LTS
flavor	m1.medium

5. BENCHMARKING

Benchmarking was done to evaluate and compare the performance of the system by varying the Cloud environment, cluster set-up, algorithms and data sizes. Benchmarking was also done on system performance in local machines. Local machine used was dual core machine with 8 GB RAM and 64bit OS. The configuration of the machines on which the analyses were run is as given below: The system was deployed in Chameleon cloud platform. The virtual machines used for the analysis had the following configuration:

5.1. Naive Bayes vs NLTK-Textblob

Two different algorithms were used to classify the tweets based on positive, negative or neutral sentiment. Naive Bayes classifier was developed in PySpark and the saved model was used for the classification task. The model was built on annotated training data and tested on data extracted from Twitter. Textblob module of the NLTK package in Python was also used for classifying the tweets. The performance of both the machine learning models were verified on a common validation set. Table 2 gives the accuracies of the two models on 10,000 tweets extracted from Twitter and annotated manually. Since Textblob had better accuracy than Naive Bayes classifier, all the analyses were done based on the sentiment scores determined using Textblob.

Table 2. Accuracy of ML models

Model	Accuracy
NLTK-Textblob	62.81%
PySpark-Naive Bayes	51.36%

5.2. Chameleon

Chameleon cloud was one of the three platforms used for deploying the real-time system. The time taken for installation of dependencies and the time taken for execution of the algorithms were recorded. Textblob tweet analyses was done using Python in a single cloud virtual machine. To run this analyses all package dependencies like Pandas, TextBlob, Numpy, Zipcode, matplotlib, nltk and node.js had to be installed in the cluster. Time taken to determine the sentiment score of tweets using textblob is shown in Figure 13.

The Naive Bayes classifier built in PySpark was implemented

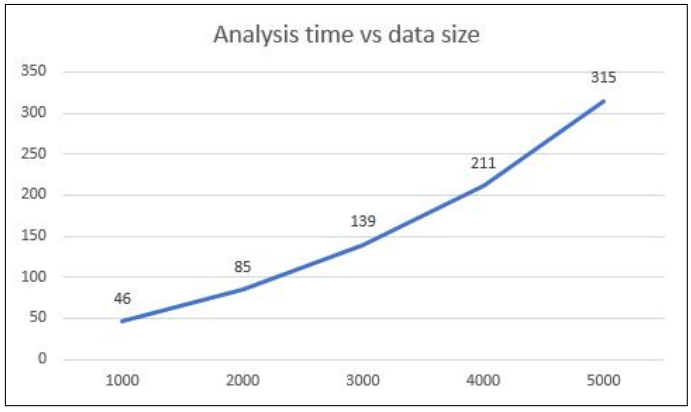


Fig. 13. Run time of Textblob

in multiple nodes within a cloud Virtual machine. To implement this model, Hadoop, Spark and HDFS had to be installed in each of the nodes in the cluster. As clearly seen from the Table 3 PySpark takes longer time to to get executed as compared to Textblob due to the parallel processing of data. Also, the installation of dependencies take significant amount of time as compared to the system execution. Figures 14, 15 and 16 give the run time of implementing the classifier in a cluster with 1,2 and 3 worker nodes respectively. It can be seen that the run-time decreases with the increase in number of nodes in the cluster.

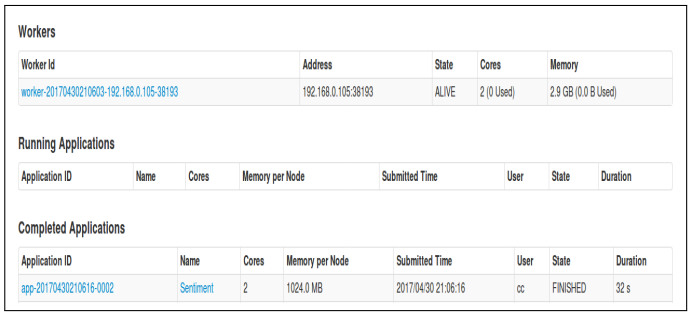


Fig. 14. Run time of Naive Bayes in 1-node cluster

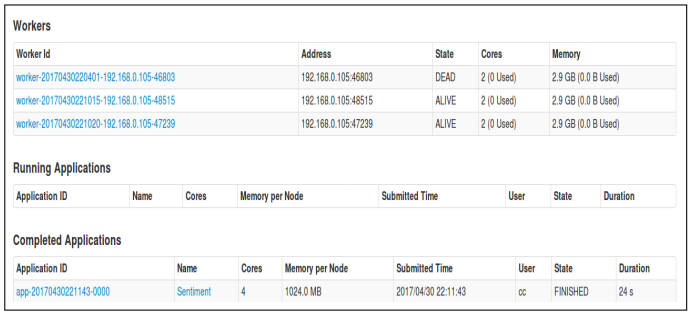


Fig. 15. Run time of Naive Bayes in 2-node cluster

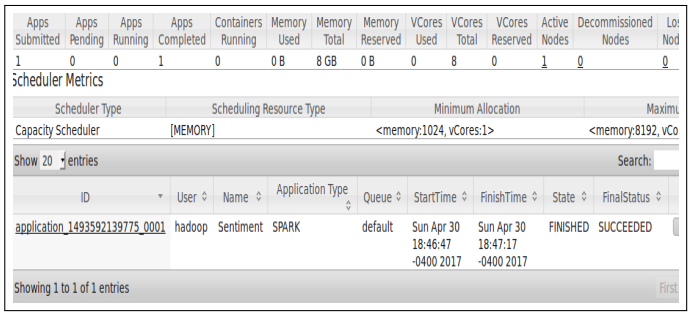


Fig. 16. Run time of Naive Bayes in 3-node cluster

Table 3. Run time for installation and execution

Task	Nodes	Runtime
Installation-Python packages	1	31s
Node.js	1	21s
Tweet Pre-processing	1	46s
Taxtblob-1000 tweets	1	0.21s
PySpark dependencies	3	148s
Naive Bayes	3	30s

6. CONCLUSION

A framework is built for visualizing the analysis performed on the real time tweets. By obtaining the tweets from the twitter streaming API, different text analysis was performed on the tweets and the results were aggregated at multiple levels (geographically, time, etc.) and was projected using charts built using High charts and D3.Js. A Naive Bayes Classifier is built on a spark cluster which was used for classifying the sentiments. Finally, multiple benchmarks were performed pertaining to deployment of required software, analysis time vs data size, and time taken for building a classifier in spark based on the number of worker nodes and cluster size.

Acknowledgement This project was a part of the Big Data Software and Projects (INFO-I524) course. We would like to thank Prof. Gregor von Laszewski, Prof. Gregory Fox, the Associate Instructors of this course for their constant help and support. Results presented in this paper were obtained using the Chameleon testbed supported by the National Science Foundation.

REFERENCES

[1] J. Boucher and C. E. Osgood, "The pollyanna hypothesis," *Journal of Verbal Learning and Verbal Behavior*, vol. 8, no. 1, pp. 1 – 8, 1969. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022537169800022

[2] I. live stats, "Twitter usage statistics | usage live stats," Web Page, Mar. 2017, accessed: 2017-3-24. [Online]. Available: http://www.internetlivestats.com/twitter-statistics/

[3] I. live stats, "Tweepy documentation|tweepy version 3.5.0," Web Page, Aug. 2009, accessed: 2017-3-24. [Online]. Available: http://tweepy.readthedocs.io/en/v3.5.0/

[4] P. software foundation, "Json encoder and decoder- python 2.7 documentation," Web Page, Mar. 2017, accessed: 2017-3-24. [Online]. Available: https://docs.python.org/2/library/json.html

[5] S. developers, "Numpy and scipy documentation," Web Page, Mar. 2017, accessed: 2017-3-24. [Online]. Available: https://docs.scipy.org/doc/

[6] P. software foundation, "zipcode-python package index," Web Page, Mar. 2017, accessed: 2017-3-24. [Online]. Available: https://pypi.python.org/pypi/zipcode

[7] N. project, "Natural language toolkit-nltk 3.0 documentation," Web Page, Mar. 2015, accessed: 2017-3-24. [Online]. Available: http://www.nltk.org/

[8] S. Loria, "Textblob-simplified text processing," Web Page, Mar. 2017, accessed: 2017-3-24. [Online]. Available: https://textblob.readthedocs.io/en/dev/

[9] A. Mueller, "Heat map | highcharts," Web Page, Mar. 2017, accessed: 2017-3-24. [Online]. Available: https://github.com/amueller/word_cloud

[10] J. H. amd Darren Dale amd Eric Firing amd Michael Droettboom and the Matplotlib development team, "overview- matplotlib documentation,"

Web Page, Mar. 2012, accessed: 2017-3-24. [Online]. Available: https://matplotlib.org/contents.html
[11] J. Laskowski, "Spark architecture," Web Page, Mar. 2017, accessed: 2017-3-24. [Online]. Available: https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-architecture.html
[12] TutorialsPoint, "Reselient distributed datasets," Web Page, Mar. 2017, accessed: 2017-3-24. [Online]. Available: https://www.tutorialspoint.com/apache_spark/apache_spark_rdd.htm
[13] L. Foundation, "Node.js," Web Page, Mar. 2017, accessed: 2017-3-24. [Online]. Available: https://nodejs.org/en/
[14] M. Bostock, "D3.js data driven documents," Web Page, Mar. 2017, accessed: 2017-3-24. [Online]. Available: https://d3js.org/
[15] Wikipedia, "Highcharts-wikipedia," Web Page, Mar. 2017, accessed: 2017-3-24. [Online]. Available: https://en.wikipedia.org/wiki/Highcharts
[16] Highcharts, "Interactive javascript charts for your webpage | highcharts," Web Page, Mar. 2017, accessed: 2017-3-24. [Online]. Available: https://www.highcharts.com/
[17] G. von Laszewski, "Cloudmesh client," Web Page, Mar. 2017, accessed: 2017-3-24. [Online]. Available: https://github.com/cloudmesh/client
[18] Ansible, "Overview: How ansible works," Web Page, Mar. 2017, accessed: 2017-3-24. [Online]. Available: https://www.ansible.com/how-ansible-works
[19] S. Somasundaran, J. Ruppenhofer, and J. Wiebe, "Detecting arguing and sentiment in meetings," in *Proceedings of the SIGdial Workshop on Discourse and Dialogue*, vol. 6, 2007.
[20] E. Riloff and J. Wiebe, "Learning extraction patterns for subjective expressions," in *Proceedings of the 2003 conference on Empirical methods in natural language processing*. Association for Computational Linguistics, 2003, pp. 105–112.
[21] Highcharts, "Master detail chart | highcharts," Web Page, Mar. 2017, accessed: 2017-3-24. [Online]. Available: https://www.highcharts.com/demo/dynamic-master-detail
[22] Highcharts, "Heat map | highcharts," Web Page, Mar. 2017, accessed: 2017-3-24. [Online]. Available: https://www.highcharts.com/demo/heatmap
[23] M. Bostock, "Les miserables co-occurrence," Web Page, Apr. 2012, accessed: 2017-3-24. [Online]. Available: https://bost.ocks.org/mike/miserables/
[24] J.-D. Fekete, "Reordering .jdfekete/reorder.js wiki," Web Page, Aug. 2015, accessed: 2017-3-24. [Online]. Available: https://github.com/jdfekete/reorder.js/wiki/Reordering
[25] Highcharts, "Highmaps demos | highcharts," Web Page, Mar. 2017, accessed: 2017-3-24. [Online]. Available: https://www.highcharts.com/maps/demo
[26] Highcharts, "Small us with data labels | highcharts," Web Page, Mar. 2017, accessed: 2017-3-24. [Online]. Available: https://www.highcharts.com/maps/demo/us-data-labels
[27] Highcharts, "Map with overlaid pie charts | highcharts," Web Page, Mar. 2017, accessed: 2017-3-24. [Online]. Available: https://www.highcharts.com/maps/demo/map-pies
[28] J. Morgan, "Real-time d3 visualization," Web Page, Oct. 2015, accessed: 2017-3-24. [Online]. Available: https://github.com/UsabilityEtc/d3-twitter-geo-stream

7. WORK BREAKDOWN

The work in this project was equally distributed between the following authors.

Sowmya Ravi: Building Spark pipeline and using Pyspark for building Naive bayes Classifier algorithm, benchmarking & troubleshooting and writing report.

Sriram Sitharaman: Creating visualization using D3.JS & NodeJs, building ansible scripts, Writing Python Scripts for Data analysis and writing report.

Shahidhya Ramachandran: Writing report, Writing Python scripts for Data Analysis, Creating visualization using HighCharts.

Aviation Data Analysis Using Apache Pig

HARSHIT KRISHNAKUMAR[1*] AND KARTHIK ANBAZHAGAN[2]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[2] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*] Corresponding authors: harkrish@iu.edu, kartanba@iu.edu

Project-002, May 4, 2017

Data science is a challenging field which gives actionable insights into data, enabling businesses to take calculated decisions. Big data techniques help and accelerate the analysis of data in real time. Big Data can be used to monitor things as diverse as flight data, traffic data, and financial transactions. With huge increase in the volume of air travel and drastic weather changes, flights delays and cancellation are on the rise. This project aims at tracking the aviation data and providing a list of the busiest airports by total flight traffic across the US. The system has been deployed in chameleon cloud platform. Apache Pig deployed on a Hadoop cluster is used to join multiple features across massive datasets to query and analyse the data across a Distributed File System.

Keywords: big data, Apache Pig, Apache Hadoop, Chameleon Cloud, Avaiation Analysis

https://github.com/cloudmesh/sp17-i524/blob/master/project/S17-IR-P002/report/report.pdf

CONTENTS

1. INTRODUCTION

Air travel is getting increasingly popular with the airlines providing cheaper fares and better services [1]. However, owing to the increased congestion in traffic during festive season and ever fluctuating weather conditions, there are cancellations and delays to the flights. Cancellations tend to be costly for both passengers and the airline companies and pose difficulties for the customers who might have appointments or connecting flights. According to new Department of Transportation filings [2], the resulting change fees and cancellation penalties passengers end up paying up to a whopping $2 billion a year.

With the increasing availability of data and improvement technology, there is growth in the area of big data and computing [3]. There has been development of new tools and frameworks to efficiently process large amounts of data. These tools rapidly process data and execute complex parallel algorithms while automatically splitting the the work across a set of available nodes. Earlier data processing applications could not handle the high volume, velocity, and variety of big data. This is the data which is difficult to handle in conventional data analytics. Big Data is now being used to monitor things as diverse as flight data [4], traffic data, and financial transactions. The challenge of Big Data is how to use it to create something that is value to the user. How to gather it, store it, process it and analyze it to turn the raw data information to support decision making. This paper aims to leverage big data by deploying a Hadoop cluster and running an analysis across large datasets containing airline cancellations and delays details using Pig Scripts and provide insights almost in real-time.

Hadoop allows to store and process Big Data in a distributed environment across group of computers. It is intended to scale up starting with a single machines and will be scaled to many

machines. In our analysis we benchmark the performance of the clusters for various data sizes and cluster configurations. We utilize Chameleon cloud to run the Hadoop cluster and perform our benchmark.

2. ARCHITECTURE

2.1. Cloudmesh Client

Cloudmesh client is a lightweight cloud client to manage virtual clusters. It enables users to access multiple cloud environments from command shell and commandline. Cloudmesh client allows to easily manage virtual machines, containers, HPC tasks, through a convenient client and API. It is a client based toolkit that is installed and run on the users computers. The installation has been done on Ubuntu 16.04 for this project. It has a layered architecture that allows easy development of new features. A resource abstraction layer allows the integration of a multitude of resources spanning HPC, Containers, and Cloud resources.

Using Cloudmesh Client's "cm define" and "cm deploy" command with the right configurations, it is possible to define and create a multi node cluster and install Hadoop with addons. The architecture of Cloudmesh Client makes the deployment of clusters simple from Ubuntu shell with basic commands.

2.2. Ansible

Ansible is an open-source automation tool that automates software provisioning, configuration management, and application deployment. Once Ansible gets installed on a control node, which is an agentless architecture [5], it connects to a managed node through the default OpenSSH connection type. The architecture of Ansible is shown in 3. As with most configuration management softwares, Ansible distinguishes two types of servers: controlling machines and nodes. First, there is a single controlling machine which is where orchestration begins. Nodes are managed by a controlling machine over SSH. The controlling machine describes the location of nodes through its inventory.

Ansible manages machines in an agent-less manner, it doesn't requires any software to be installed on the remote machines to make them manageable. By default is manages remote machines over SSH or WinRM [6], which are natively present on those platforms. These modules are temporarily stored in the nodes and communicate with the controlling machine through a JSON protocol over the standard. Ansible is decentralized, if needed, Ansible can easily connect with Kerberos, LDAP, and other centralized authentication management systems.

The design goals of Ansible includes consistency, high reliability, low learning curve, security and to be minimalistic in nature. Ansible doesn't require dedicated users or credentials - it respects the credentials that the user supplies when running Ansible. Similarly, Ansible does not require administrator access, it leverages sudo, su, and other privilege escalation methods on request when necessary [7].

The functioning of Ansible allows users to define the ip address of namenode in the hosts file, and give the required instructions in .yml file. The configuration yaml file is called Ansible Playbook, and can be run to perform all the tasks listed in it on all the servers mentioned in the hosts file. This way Ansible makes handling multiple virtual machines simple by iteratively performing a set of tasks on all the hosts. In the hosts file, individual host name can be grouped together to form host groups, which can be called for advanced installations. Ansible also has options to assign roles to perform a specified set of tasks, its own version of modularization. There are options to use variables

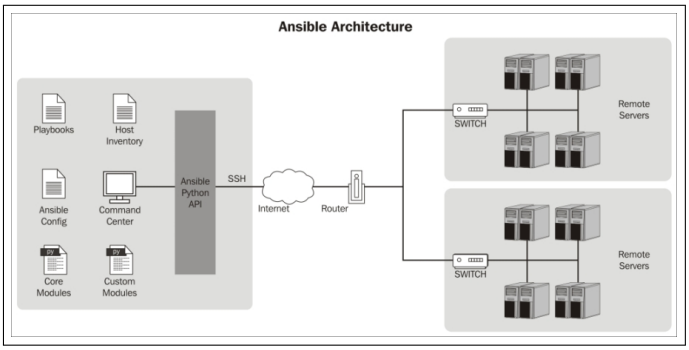


Fig. 1. Architecture of Ansible [8]

in the playbook. The current project uses Ansible to handle the movement of files between servers, and to run Shell commands on Hadoop namenode.

2.3. Chameleon Cloud

Chameleon Cloud is an open source large scale cloud platform available for the research community. The current commercial cloud platforms are mostly inaccessible for the student and research community. Chameleon project was created with funding from National Science Foundation and has 650 multi-core cloud nodes, 5PB of total disk space, and ability to leverage 100 Gbps connection between the sites. Chameleon cloud enables users to experiment on transformative concepts in deeply programmable cloud services, design, and core technologies on problems ranging from the creation of Software as a Service to kernel support for virtualization. However, Chameleon cloud of supported research includes many other areas such as developing Platforms as a Service, creating new and optimizing existing Infrastructure as a Service components, investigating software-defined networking, and optimizing virtualization technologies. [9]

The current project uses Chameleon Cloud instance created for the Big Data class at Indiana University. The current instance allows cloud created in three flavors - small, medium and large. Chameleon Cloud allows users to create any number of virtual machines with the required flavor, and allows to assign floating public IPs to be accessible from the outside. Each of the virtual machines can be created using any Operating System, and the current project uses Ubuntu14.04 version installed on Chameleon Cloud.

2.4. Hadoop

Apache Hadoop is an open-source software framework for storage and large-scale processing of data-sets on clusters of commodity hardware. Hadoop is written in Java, and it allows parallal processing of large datasets. Hadoop has four main components [10]:

1. **Hadoop Common:** These are the basic Java libraries and configuration files required for Hadoop to run
2. **Hadoop YARN:** Yarn is Hadoop's resourse navigator. It is used for resource management and job scheduling
3. **HDFS:** The Distributed File System is Hadoop's data storage. It distributes the data across Hadoop cluster
4. **Hadoop MapReduce:**It is a Yarn based system used to ro distribute Hadoop jobs across cluster

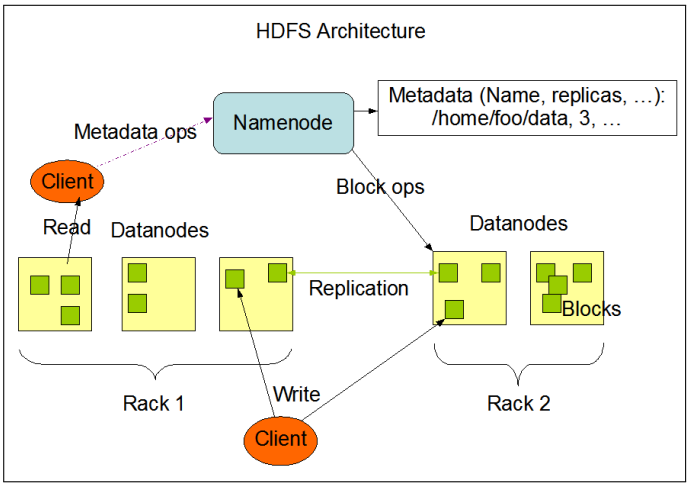


Fig. 2. Architecture of Hadoop [11]

2.4.1. HDFS

HDFS is a fault-tolerant system designed to be deployed on low-cost hardware. HDFS works well with large data sets and provides high throughput access to application data. A typical file in HDFS is in gigabytes to terabytes in size. HDFS has a master-slave architecture. A typical HDFS cluster consists of a single NameNode, a master server that manages the files system, a number of DataNodes which manages the storage attached to the node they run on. Internally, a file is split into multiple blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

The NameNode and DataNode are pieces of software designed to run on any machine. HDFS is built using the Java language. Any machine that supports Java can deploy HDFS. A deployement takes place in such a manner that one machine runs the NameNode software and each of the other machines runs one instance of the DataNode software. The presence of a NameNode is a must and the presence of a single NameNode in a cluster greatly simplifies the architecture of the system. The system is designed in such a way that user data never flows through the NameNode.

2.4.2. MapReduce

Hadoop enable distributed processing of massive structured/unstructured data across multiple commodity computers where each node contains it own data storage. This programming model of parallel processing and assimilation is the MapReduce program. MapReduce hsa two main functions - a Mapper class and a Reduce class. MapReduce serves two essential functions: It parcels out work to various nodes within the cluster or map, and it organizes and reduces the results from each node into a cohesive answer to a query.

MapReduce is composed of several components, including:

- JobTracker: the master node that manages all jobs and resources in a cluster
- TaskTrackers: agents deployed to each machine in the cluster to run the map and reduce tasks
- JobHistoryServer: a component that tracks completed jobs, and is typically deployed as a separate function or with JobTracker

2.5. PIG

PIG uses Pig Latin to analyse data in Hadoop. To perform data analysis, programmers need to write a Pig script using the Pig Latin language. Pig converts these scripts into multiple MapReduce jobs. There are various components in the Apache Pig framework:

- Parser: It checks the syntax of the script and displays the output (logical statements) in the form of DAG,
- Optimizer: performs logical optimizations such as such as projection and pushdown,
- Compiler: complies the logical output into series of MapReduce jobs,
- Execution engine: executes the MapReduce jobs are submitted to Hadoop in a sorted order.

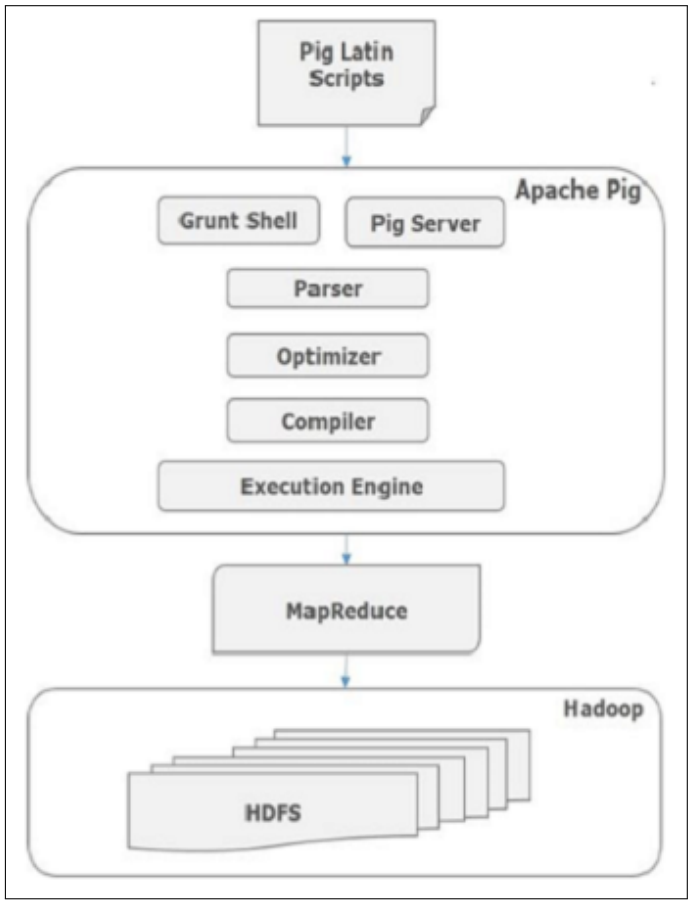


Fig. 3. Architecture of PIG [12]

3. WORK FLOW

3.1. Hadoop Deployment

The deployment was done from an Ubuntu14.04 instance running on Oracle Virtual Box. Hadoop instance with three nodes was deployed on Chameleon cloud using Cloudmesh Client

was used to automatically create the cluster and deploy Hadoop software with Pig and Spark add-ons. Three virtual machines with Ubuntu14.04 version were installed on Chameleon Cloud, and Hadoop along with Pig and Spark add-ons was deployed to the clouds with one name node and two data nodes.

3.2. Cluster Configurations

Hadoop was deployed on a cluster with three nodes. Three virtual machines were created on Chameleon Cloud for this purpose, with one instance as namenode (master) and rest of the instances as datanodes (slaves). All the three virtual machines were created with Ubuntu14.04 OS. Each of the vms had 20 GB space, with 2GB ram and one CPU. This configuration is called the "small" flavour in Chameleon Cloud. Each vm was assigned a separate floating public IP which could be used to SSH and connect.

3.3. Ansible Playbook

Ansible was installed in the oracle virtual box and configured to automatically perform file transfers and to run Pig code on the cluster. All our interactions are with the namenode of Hadoop, so the public IP of the namenode of our installation was given in the hosts file of Ansible for all purposes.

Five Ansible Playbooks were setup, each with the following purposes:

1. Transfer data and Pig Script files to namenode from local location
2. Transfer data files to hdfs location on namenode using *hdfs dfs* command
3. Run pig code using *pig <any_pig_script>* shell command on the name node
4. Transfer the output files from hdfs back to namenode
5. Transfer output files to local location

Each of these would run on the locations mentioned in hosts file on usernames CC and hadoop, which were the default users created by cloudmesh client.

4. AIRLINE ANALYSIS

4.1. Datasets

The current project focuses on analysing flight cancellations and delays data to find trends. The dataset is taken from the US Department of Transportation's Bureau of Statistics [13]. Their website releases monthly air travel statistics and summary report of all the flights information of previous month. Along with this report, they release the raw data which is openly available to be downloaded and analysed. The two datasets we use are taken from the raw data provided in the website. A brief description about the datasets is given below.

1. **Delayed Flights** contains information about all the cancelled or delayed flights ranging across the years 1987 to 2008. It has 29 columns which are described in the Table 1.
2. **Airports** is a reference dataset, which gives the airport names and locatons. The column description is shown in the Table 2.

The Delayed Flights dataset is around 250 mb in size and the airports dataset is around 250 kb since it is reference data.

Column Name	Description
Year	1987-2008
Month	1-12
DayofMonth	1-31
DayOfWeek	1 (Monday) - 7 (Sunday)
DepTime	actual departure time
CRSDepTime	scheduled departure time
ArrTime	actual arrival time
CRSArrTime	scheduled arrival time
UniqueCarrier	unique carrier code
FlightNum	flight number
TailNum	plane tail number
ActualElapsedTime	in minutes
CRSElapsedTime	in minutes
AirTime	in minutes
ArrDelay	arrival delay in minutes
DepDelay	departure delay in minutes
Origin	origin IATA airport code
Dest	destination IATA airport code
Distance	in miles
TaxiIn	taxi in time, in minutes
TaxiOut	taxi out time in minutes
Cancelled	was the flight cancelled?
CancellationCode	reason for cancellation
Diverted	1 = yes, 0 = no
CarrierDelay	in minutes
WeatherDelay	in minutes
NASDelay	in minutes
SecurityDelay	in minutes
LateAircraftDelay	in minutes

Table 1. Delayed Flights Column Information

Column Name	Description
iata	Airport Code
airport	Airport Name
city	Airport City
state	State Code
country	Country Code
lat	Lattitude
long	Longitude

Table 2. Airports Column Information

4.2. Pig Script

Pig installation runs on the Hadoop hdfs system. The script needed for this analysis was based out of the idea from the blog post written by Sumit Anand [14]. It takes the two datasets, performs basic joins, orders the rows and returns top five airports which cause most delays in all the years. This script was run on the namenode on Pig hadoop mode, with the input files given from hdfs dfs locations. This script analyses the data and outputs into hdfs location. The codes are written in Pig Latin language and can be run in Pig grunt shell.

5. BENCHMARKING

Benchmarking is a process in software development which allows developers to determine the performance of their systems. This can be done for multiple reasons including looking for improvements and future planning. The basic requirements of a benchmark are that the environmental conditions of test should be same each time it is run, and the test should be repeatable.

The current paper presents multiple levels of benchmarking, starting from the time taken to deploy Hadoop cluster with three nodes, the time taken to move files from local to namenode and hdfs, time taken to run the pig code and time taken to move files from hdfs to namenode and local. Deployment of Hadoop cluster is done using Cloudmesh Client, and the rest of the tasks are done using Ansible. For all Ansibls steps, there are individual playbook files which can be called from separate Ansible commands. Each of these steps has been timed using Ubuntu Shell's "*time*" command.

Excluding the time taken to download, install and configure Cloudmesh Client, the time taken to create three vms with the "small" configuration, deploy Hadoop with three nodes, Spark and Pig addons is 8 minutes 46 seconds. This deployment was done from Oracle Virtual Box with a ram capacity of 8 GB.

The experiment was performed on the entire dataset, and the tests were repeated with 50% and 25% of the data. The results of the benchmarking tests are given in the tables 3, 4 and 5.

6. SUMMARY

Airline industry is rapidly growing as the customers who take flights are increasing. Considering this trend, the cancellations and delays come into focus. It is imperative that Big Data technologies are deployed in this sector for quick results. This project aims at using Hadoop and Pig to run a basic analysis on Flight Delays data and benckmark the clouds' performance.

As we can see from the benchmark results, there is a drastic decrease in the first three steps (copy data to cloud, copy data to hdfs and run pig script) when the input file size is decreased. This is expected since there is less data for Hadoop to crunch. However in each of these cases, the output is top 3 airports, which is a textfile with three records. Thus we do not see much of a change in the time to copy the output (last two steps).

Overall, the time taken to run this entire process starting from copying data to namenode to getting output to local is taking less than 6 minutes for the entire dataset. Keeping in mind that our benchmark was for the "small" flavor of Chameleon Cloud (2GB ram) this is a good score, and if the cluster is vertically or horizontally scaled up, this analysis will yield results quicker. In order to take full advantage of Hadoop's parallel processing capabilities, it would be ideal if the number of nodes in the cluster were increased (horizontal scaling).

Task	Time Taken
Copy Data to cloud	1 min 33 sec
Copy Data to HDFS	23 sec
run pig script	3min 1 sec
Copy output to cloud	25 sec
Copy output to local	17 sec

Table 3. Benchmark Results for the entire data

Task	Time Taken
Copy Data to cloud	54 sec
Copy Data to HDFS	20 sec
run pig script	1 min 55 sec
Copy output to cloud	24 sec
Copy output to local	16 sec

Table 4. Benchmark Results for the 50% of the data

Task	Time Taken
Copy Data to cloud	27 sec
Copy Data to HDFS	19 sec
run pig script	1 min 6 sec
Copy output to cloud	22 sec
Copy output to local	17 sec

Table 5. Benchmark Results for the 25% of the data

ACKNOWLEDGEMENTS

The author thanks Professor Gregor Von Lazewski for providing us with the guidance and topics for the Project. The author also thanks the AIs of Big Data Class for providing the technical support.

REFERENCES

[1] R. Zoglin, "How airlines turned your vacation plans into a losing bet," Web Page, March 2015, accessed: 2015-05-28. [On-

line]. Available: https://www.bloomberg.com/news/articles/2015-05-28/airline-fares-change-fee-is-focus-of-increasing-consumer-anger
[2] S. McCartney, "Your bad luck is a windfall for airlines," Web Page, July 2009, accessed: 2009-07-30. [Online]. Available: https://www.wsj.com/articles/SB10001424052970204563304574318212311819146
[3] Apache, "Hdfs architecture," Web Page, 2016. [Online]. Available: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html
[4] K. Noyes, "For the airline industry, big data is cleared for take-off," June 2014, accessed: 2014-06-19. [Online]. Available: http://fortune.com/2014/06/19/big-data-airline-industry/
[5] Wikipedia, "Ansible," Web Page, April 2017, accessed: 2017-04-28. [Online]. Available: https://en.wikipedia.org/wiki/Ansible_(software)
[6] Ansible, "How ansible works," Web Page, Jan 2017, accessed: 2017-01-01. [Online]. Available: https://www.ansible.com/how-ansible-works
[7] Ansible, "Ansible," Web Page, Jan 2017, accessed: 2017-01-01. [Online]. Available: https://github.com/ansible/ansible
[8] M. Mohaan, "The ansible architecture," Web Page, accessed: 2017-01-01. [Online]. Available: https://www.packtpub.com/mapt/book/Networking%20/9781783550630/1/ch01lvl1sec09/The+Ansible+architecture
[9] Chameleon Cloud, "About chameleon cloud," Web Page, accessed: 2017-04-22. [Online]. Available: https://www.chameleoncloud.org/about/chameleon/
[10] Tutorials Point, "Hadoop - introduction to hadoop," Web Page, accessed: 2017-04-22. [Online]. Available: https://www.tutorialspoint.com/hadoop/hadoop_introduction.htm
[11] Hadoop, "Hdfs architecture guide," Web Page, Apr 2013, accessed: 2013-04-08. [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
[12] TutorialPoint, "Apache pig - architecture," Web Page, 2017, accessed: 2017-01-01. [Online]. Available: https://www.tutorialspoint.com/apache_pig/apache_pig_architecture.htm
[13] Bureau of Transportation Statistics, "Airlines and airports," Web Page, accessed: 2017-04-22. [Online]. Available: https://www.rita.dot.gov/bts/sites/rita.dot.gov.bts/files/subject_areas/airline_information/index.html
[14] S. Anand, "Aviation data analysis using apache pig," Web Page, accessed: 2017-04-22. [Online]. Available: https://acadgild.com/blog/aviation-data-analysis-using-apache-pig/

AUTHOR BIOGRAPHIES

Harshit Krishnakumar is pursuing his MSc in Data Science from Indiana University Bloomington
Karthik Anbazhagan is pursuing his MSc in Data Science from Indiana University Bloomington

Detecting Stop Signs in Images and Videos in a Robot Swarm

RAHUL RAGHATATE[1,*] AND SNEHAL CHEMBURKAR[1]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*] Corresponding authors: rraghata@iu.edu, snehchem@iu.edu

S17-IR-P003, May 4, 2017

This project is designed to deploy a scalable stop sign detection algorithm to process real-time image and video streams. The deployment is automated allowing for minimal user-interaction. Spark on Yarn provides the distributed computing power required to scale the stop sign detection algorithm to process big data. This system is useful in automated driving vehicles and advanced driver assisted systems (ADAS) to detect and classify the street signs and control the vehicle accordingly. A comparative benchmark is developed based on the performance of the application on multiple cloud systems.

Keywords: Street Signs, Ansible, Video Streams, OpenCV, Spark, Cloud

Report: https://github.com/cloudmesh/sp17-i524/tree/master/project/S17-IR-P003/report/report.pdf
Code: https://github.com/cloudmesh/cloudmesh.street

1. INTRODUCTION

There are many applications developed based on simple idea of object detection, like auto-tagging pictures (e.g. Facebook, Phototime), counting the number of people in a street (e.g. Placemeter), tracking an object in video streams, detecting vehicles to name a few. Based on this concept of object detection, we deploy a scalable software for stop sign detection using Spark on multiple clouds. The software deployment is automated using Ansible. Cloudmesh provides simple command line interface for defining the number of clusters as well as deploying them. Benchmarks are developed based on the performance of this software on different cloud systems. The database of street signs will be restricted to US street signs. The only publicly available dataset for US traffic signs is the LISA dataset [1] which is very huge. Hence, the video streams used for this project are captured by us using a mobile camera.

OpenCV is a computer vision library used to process video streams in Python. A lot of computing power goes into processing videos in real-time, this is where the cloud systems come in. We leverage the distributed computing power of Spark on Yarn for faster processing of images and videos. This solution is deployed on two different clouds to benchmark their performance. In this era of autonomous driving and advanced driver assisted systems (ADAS), detection and classification of traffic signs is a handy feature. Benchmarks have been created for the traffic sign detection on the German and Belgium Traffic Sign Datasets [2].

2. REQUIREMENT ANALYSIS

The following technologies are used throughout the project:

- Cloudmesh provides command line interface to connect and deploy clusters to different cloud systems.
- Ansible is an agentless, automated software deployment tool.
- Python - Programming language.
- Spark - Distributed computing engine.
- YARN - is the resource manager for Spark.
- OpenCV - Image and video analysis for street sign detection using open source computer vision libraries.The OpenCV library provides several transformations that can be applied to images(apply filters, transformation), detect and recognize objects in images.

3. SCOPE

The initial project idea was to automate the deployment of street sign detection algorithm over multiple cloud systems. As we proceeded through the project, we realized that training of Haar Cascade classifier is challenging. For a training dataset of 1000 samples the training can go on for 3-4 days. It turned out to be an exhaustive process. The resultant classifiers were unable to detect the specific signs. The details of the OpenCV training process we followed are given in the appendix. Due to difficulty training classifiers for street signs, we had to reduce the scope of this project to detect only stop signs using a pre-trained classifier available on Github []. The stop sign detection is implemented for both images and videos using Spark.

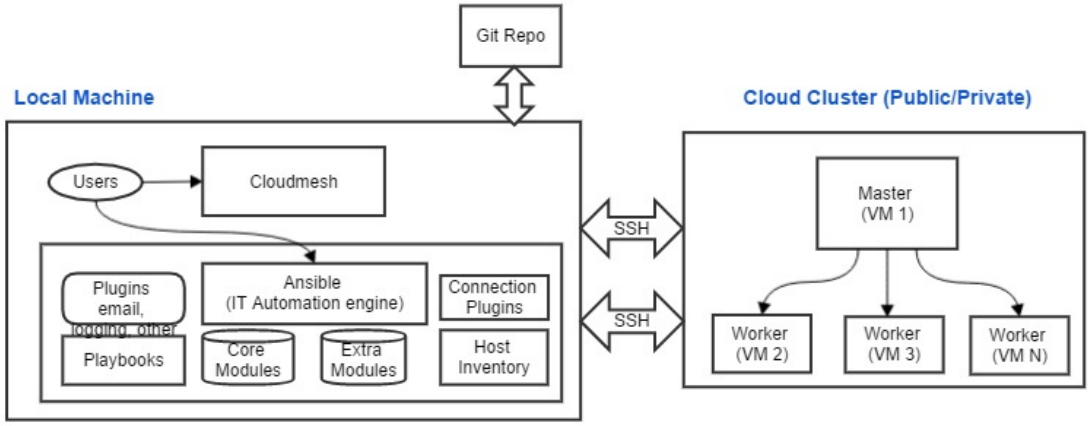


Fig. 1. System Architecture

4. SYSTEM ARCHITECTURE

Figure 1 shows an overview of our system architecture. Ansible and Cloudmesh client are installed on the host or local machine. Roles are defined in the Ansible playbook for each of the different steps in the deployment process. We execute the Ansible playbooks to instantiate the cloud machines, deploy Hadoop and Spark on them and then carryout the stop sign detection on Spark using Yarn resource manager. Once the job is submitted to Spark, the driver program initializes the SparkContext object which is responsible for the execution of the job. Input data is parallelized and sent to the worker nodes for processing. Yarn acts as the resource manager and provides executors to the worker nodes. The output is saved to the local file system on the master node and transferred to the host machine through a script. More details on the mechanics can be understood in the following sections.

5. CLOUD INFRASTRUCTURE

For the purpose of this project, we have been provided with two clouds – Chameleon [3] and Jetstream [4]. Chameleon cloud is a National Science Foundation funded experimental testbed that provides large scale cloud services to "members of the US computer Science research community and its international collaborators [3]." One can create virtual machines and manage them through the OpenStack Horizon interface. Jetstream allows researchers to leverage the computational power of cloud while retaining the look and feel of our home machines. Jetstream adds cloud based computational power to the national cyberinfrastructure [4]. Both Chameleon and Jetstream provide a cloud computing environment to researchers based on OpenStack [4]. The comparison of hardware specifications for the two cloud systems is given in table 1.

6. CLOUDMESH

Cloudmesh provides an easy interface to multiple clouds such as Chameleon and Jetstream through the command line. Cloudmesh client can be installed via pip. It is a lightweight utility that enables users to connect to different clouds from their laptops or computers. Users can customize Cloudmesh client to suite their needs of cyberinfrastructure. It provides simple command line scripts to deploy Hadoop with Spark addon to either of the clouds mentioned in section 5. A set of cluster machine instances can be defined using command:

	Chameleon	Jetstream
CPU	Xeon X5550	Haswell E-2680
cores	1008	7680
speed	2.3GHz	2.5GHz
RAM	5376GB	40TBr
storage	1.5PB	2 TB

Table 1. Comparison of Cloud Specification [4] [3]

```
cm cluster define --count 3
```

Cloudmesh commands to create and deploy Hadoop clusters with Spark are included as tasks in the ansible playbooks to automate the deployment.

7. ANSIBLE AUTOMATION

Ansible is an easy to use, opensource automation tool that is used to automate the deployment of our project on the cloud infrastructure. Ansible is an agentless tool, that is, it does not require ansible to be deployed on the remote machines. It runs only on the host machine to deploy the required processes to the remote machines through SSH authentication. Using Ansible, we can create modules for each step of the deployment process and define the roles individually. An inventory file is used to define the machines in groups as required. A sample inventory file looks like:

```
[master]
192.128.0.1
[workernode]
192.168.0.2
192.168.0.3
```

Roles are defined in Ansible for the deployment of Hadoop-Spark cluster, environment setup on the virtual machines, stop sign detection as well as to fetch the results back to the local.

8. PYTHON-OPENCV

Python is preferred due to ease of use and familiarity over other programming languages. OpenCV also provides python library to enable object detection in python using this computer vision library. The initial scope included training a Haar Cascade Classifier to detect traffic signs and then testing the classifier on test data. But as explained in appendix section - Training a Haar Cascade Classifier - a decent classifier could not be trained. The stop sign detection algorithm utilizes a pre-trained stop sign classifier available on Github to perform the detection in images and videos. The results of the algorithm are saved as images in the `~/github/cloudmesh.street/ansible/output/output/` folder on your local machine, assuming that the git repository is cloned to `~/github/cloudmesh.street/`. The output files will have a bounding box around the detected object. The signdetectionusingspark.py file is used to process both images and videos. Based on whether the input is either image or video, the path to these files has to be modified in the spark-submit task of ansible playbook.

9. SHELL SCRIPT

Shell script is used to time each of the deployment step for benchmarking. Shell script is also useful in cases where a the deployment terminates in an error and needs to be continued from some intermediate steps. Individual shell scripts are created for each tasks in case of such issues to allow execution from the point of interruption.

10. BENCHMARK

Benchmarks are created based on the performance of the software in different cloud environments. The benchmarks for Jetstream are limited due to issues with their cloud. As the test data is images and videos, spark ran into memory issues on the m1.small flavor. Hence, we have done benchmakring using the medium and large flavours only. Tables 2 and 3 reflect that the configurations for the same flavors on the two clouds are different. We cannot directly compare the performance if the two machines have different specifications. Benchmarking is done for the deployment and the analysis on Chameleon and Jetstream.

JetStream	VCPU	RAM(GB)	Storage(GB)
m1.small	2	4	20
m1.medium	6	16	60
m1.large	10	30	60

Table 2. Jetstream Flavour Specifications

10.1. Chameleon Cloud Benchmarks

10.1.1. For flavor m1. medium and 50 Test Images

Figure 2 shows the time taken for analyzing the 50 images on Chameleon cloud for 1, 2, 3, 4, and 6 node clusters. We can see that as the number of nodes increases the time taken to analyze the images reduces. Figure 3 reflects the total time required to complete the deployment. We can see that the total deployment time doesn't vary much upto 4 nodes but there is a steep increase in the deployment time for 6 node cluster.

Chameleon	VCPU	RAM(GB)	Storage(GB)
m1.small	1	2	20
m1.medium	2	4	40
m1.large	4	8	80

Table 3. Chameleon Flavour Specifications

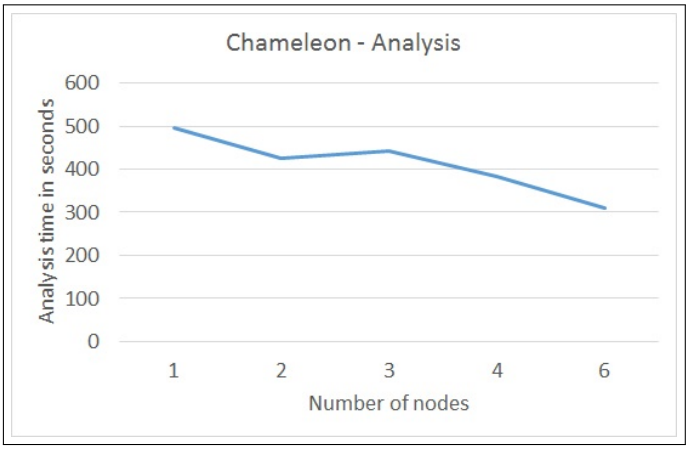


Fig. 2. Time taken by sign detection task - 50 test images

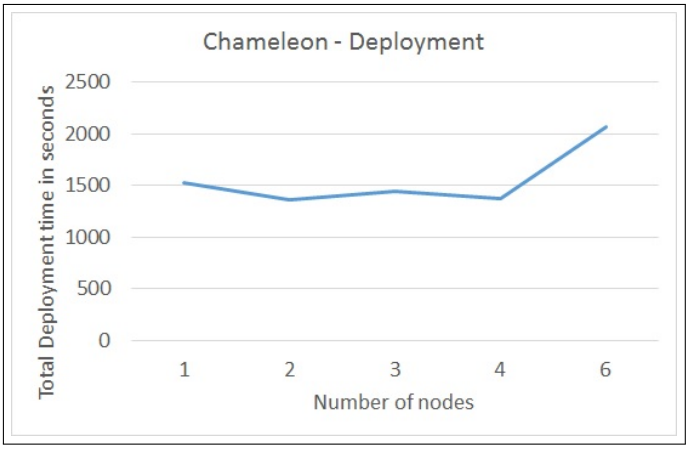


Fig. 3. Time taken for complete deployment - 50 test images

10.1.2. For flavor m1.medium and a Video Input

The video input file tested on the medium flavor is just 2 sec long but since it gets converted to frames, it comes out to 53 images that are sent to spark for processing. Figure 5 shows the time taken for analyzing a single video that is 2 seconds long on Chameleon cloud for 3, 4, 6, and 7 node clusters. We can see that as the number of nodes increases the time taken to analyze the images reduces a lot. Figure 4 reflects the total time required to complete the deployment. We can see that the total deployment

time doesn't vary much upto 4 nodes but there is a steep increase in the deployment time for 6 node cluster.

10.1.3. For flavor m1.large and a Video Input

The video input file tested on large flavor is 5 sec long and after extracting the frames, it comes out to 120 images that are sent to spark for processing. Figure 7 shows the time taken for analyzing a single video that is 5 seconds long on Chameleon cloud for 1, 2, 3, and 4 node clusters. We can see that as the number of nodes increases the time taken to analyze the reduces which is expected. Figure 6 reflects the total time required to complete the deployment. We can see that the total deployment time increases steeply at first and then starts to normalize.

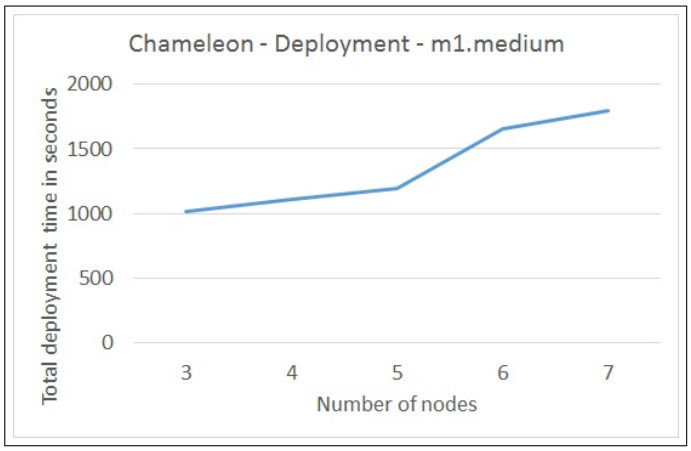


Fig. 4. Time taken for complete deployment - 1 Video (2 sec)

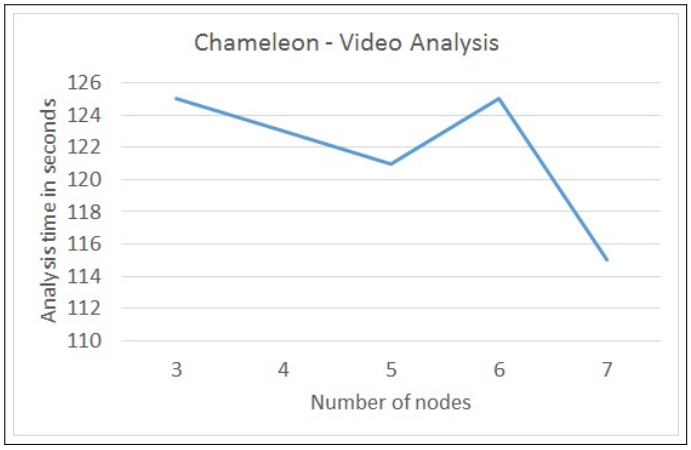


Fig. 5. Time taken for sign detection task - 1 Video (2 sec)

10.2. Jetstream Cloud Benchmarks

8 shows the time taken to the complete deployment on Jetstream when the number of images parsed are 4. It can been seen from the graph that as the number of nodes is increased the processing time is reduced. 9 and 10 reflect the performance of Jetstream cloud for 4 and 50 input images respectively. Sufficient data could not be gathered for Jetstream due to some issues with Jetstream. Hence we cannot conclude much about the performance of Jetstream.

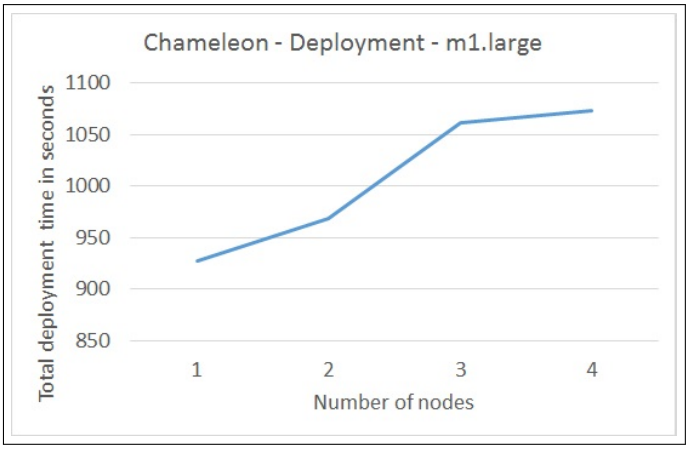


Fig. 6. Time taken for complete deployment - 1 Video (5 sec)

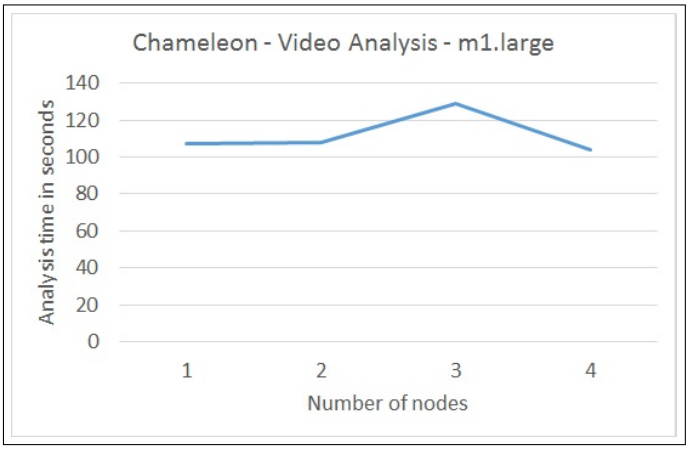


Fig. 7. Time taken for sign detection task - 1 Video (5 sec)

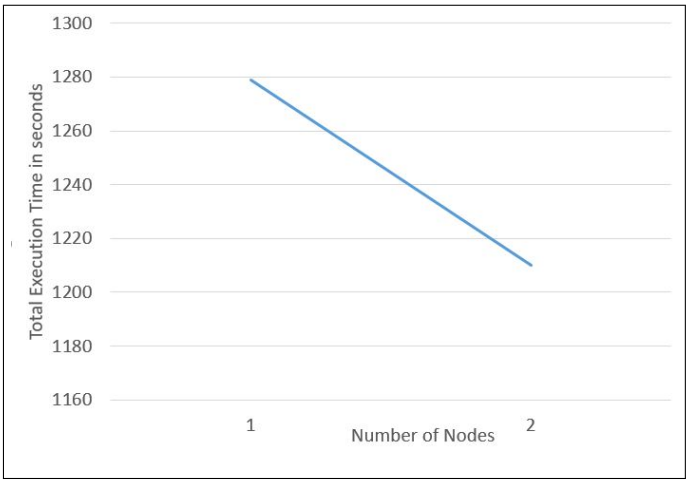


Fig. 8. Total time required to deploy for 4 test images

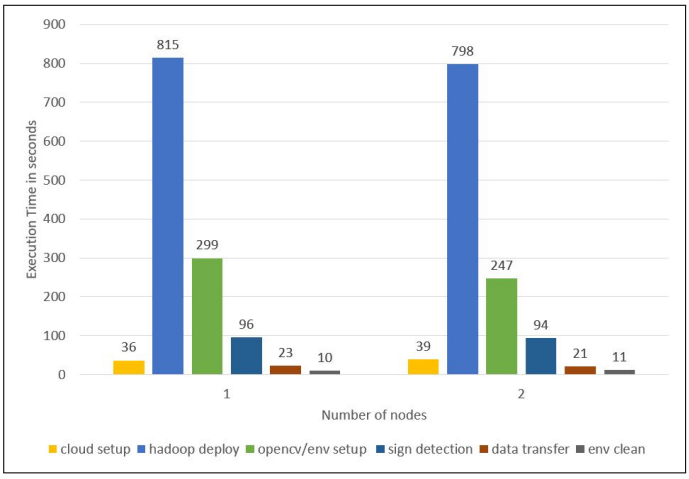


Fig. 9. Time taken by each task for 4 test images

11. USE CASES

1. Street Sign Detection for autonomous vehicles.

2. Analysis of traffic signs in Google Street View to estimate all signs ahead hence, useful in ambulance , fire brigade services, simplest path finder etc.

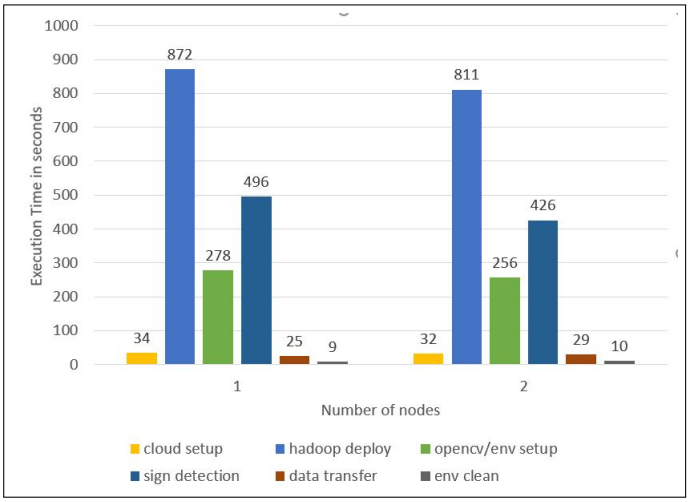


Fig. 10. Time taken by each task for 50 test images

12. FUTURE WORK

This work can be expanded to detect and classify all the U.S traffic signs which can be adapted for advanced driver assisted systems (ADAS) . Moreover, efficiency of sign detection over cloud can be increased by effective distribution of data, for e.g using Hadoop distributed file system. An extension of the stop sign detection in video streams would be to output the data as videos rather than images in realtime. As the current system is scalable, benchmarks can be developed for larger dataset with multiple classifier, similar to the German and Belgium Traffic Sign Detection and Classification benchmarks [2].

13. CONCLUSION

We have been able to successfully deploy the software to Jetstream and Chameleon clouds and test their performance. On large flavors of chameleon cloud the deployment time starts to flatten out over the curve.As the number of nodes increases the time taken to deploy Hadoop and spark to those clusters increases and on the up side the analysis on Spark is faster.

ACKNOWLEDGEMENTS

This project is undertaken as part of the I524: Big Data and Open Source Software Projects coursework at Indiana University. We would like to thank Prof. Gregor von Laszewski, Prof. Gregory Fox and the Associate Instructors for their help and support. Results presented in this paper were obtained using the Chameleon testbed supported by the National Science Foundation.

REFERENCES

[1] A. Mogelmose, M. M. Trivedi, and T. B. Moeslund, "Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 4, pp. 1484–1497, Dec 2012, accessed : 2017-03-29.

[2] M. Mathias, R. Timofte, R. Benenson, and L. V. Gool, "Traffic sign recognition #x2014; how far are we from the solution?" in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, Aug 2013, pp. 1–8, accessed : 2017-03-25.

[3] C. Cloud, "Chameleon," Web Page, accessed: 2017-04-19. [Online]. Available: https://www.chameleoncloud.org/docs/user-faq/

[4] I. University, "Jetstream," Web Page, accessed: 2017-04-11. [Online]. Available: https://jetstream-cloud.org/

[5] S. Soo, "Object detection using haar-cascade classifier," *Institute of Computer Science, University of Tartu*, 2014, accessed : 2017-03-29.

[6] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, 2001, pp. I–511–I–518 vol.1, accessed : 2017-03-29.

[7] T. Ball, "Train your own opencv haar classifier - coding robin," Webpage, Jul. 2013, accessed: 2017-04-05. [Online]. Available: http://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html

[8] G. Bradski, "Home - opencv/opencv wiki," Code Repository, accessed: 03-12-2017. [Online]. Available: https://github.com/opencv/opencv/wiki

[9] M. Gaynor, "markgaynor/stopsigns - detecting stop signs in google street view images of a provided address." Code Repository, Sep. 2016, accessed: 2017-3-21. [Online]. Available: https://github.com/markgaynor/stopsigns

AUTHOR BIOGRAPHIES

Rahul Raghatate is a graduate student at Indiana University. He will receive his Masters degree in Data Science in 2018. His research interests include Big Data and Machine Learning.

Snehal Chemburkar is a graduate student at Indiana University. She will receive her degree in Data Science in 2018. Her research interests include Big Data and Machine Learning.

A. OPENCV IMAGE PROCESSING

OpenCV provides a range of computer vision algorithms to detect objects in images. One of the simplest method for object detection is based on color. The results of Color based detection method are largely affected by the lighting conditions and one require the user to calibrate multiple times before they might get a better result in the real world [5]. Hence this technique is not very popular when detecting objects in the real world Haar features is a sophisticated technique that uses the features specific to the object in question. It been seen that working with RGB pixel values in every single pixel in the image results in computationally expensive and slow feature calculation. "A Haar-like feature considers neighboring rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region and calculates the difference between these sums. This difference is then used to categorize subsections of an image [5]." OpenCV provides a Haar feature based cascade classifier that can be used for object detection, as proposed by in [6].

A.1. Data Collection

The publicly available data set for U.S street signs is the LISA traffic dataset [1]. This dataset contains images for 47 different traffic signs. But since the data set itself is approximately 7GB, we extracted 50 images from the dataset for the purpose of testing. For our training set, we captured images of street signs and put together a few positive images for the street signs. The positive images were cropped to only contain the street sign and resized to 50x50.

A.2. Train a Haar feature-based Cascade Classifier

Based on tutorials provided in [7], [8] , we carried out multiple experiments to train a classifier to detect street signs. Since each sign needed to trained separately, we picked stop, yield, and signal ahead signs to start with. To train a classifier, we firstly required gathering at least a few positive and many negative images. The positive images are images of the object alone cropped to a size of 24x24 or 50x50 whereas the negative images should not contain the object in consideration here. In case we have a single positive image or a few positive images, OpenCV provides a utility called opencv_createsamples to generate the training and test datasets in *.vec format that is supported by the opencv_traincascade utility. The samples generated from the opencv_createsamples can be passed to the opencv_traincascade utility to get a trained classifier. Multiple experiments were carried out by differing the sample sizes (the width by height of the positive images) and varying the number of positive and negative images. As the dataset and width by height increases the computational time increases. Below are the trainings that were carried out for stop sign:

```
opencv_traincascade -data classifier -vec samples.vec
-bg negatives.txt -numStages 20 -minHitRate 0.999
-maxFalseAlarmRate 0.5 -numPos 120 -numNeg 200 -w 50
-h 50 -mode ALL -precalcValBufSize 1024
-precalcIdxBufSize 1024
```

```
opencv_traincascade -data classifier -vec samples.vec
-bg negatives.txt -numStages 20 -minHitRate 0.999
-maxFalseAlarmRate 0.5 -numPos 200 -numNeg 350 -w 50
-h 50 -mode ALL -precalcValBufSize 1024
-precalcIdxBufSize 1024
```

```
opencv_traincascade -data classifier -vec samples.vec
-bg negatives.txt -numStages 20 -minHitRate 0.999
-maxFalseAlarmRate 0.5 -numPos 600 -numNeg 100 -w 50
-h 50 -mode ALL -precalcValBufSize 1024
-precalcIdxBufSize 1024
```

When we increased the number of positive samples to 600 for the 50x50 image size, the training ran for 3.5 days. The resulting classifier was unable to detect the stop signs in the test data. After a couple more experiments and another week invested in training the classifier to no good results, we found success with a pre-trained classifier for stop signs available at [9]. The results of this classifier are shown in 11.

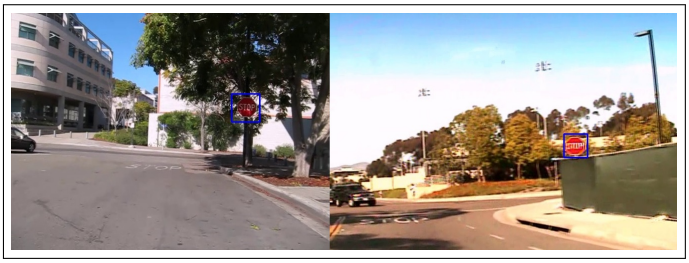


Fig. 11. Stop Sign Detection

After successful testing of stop sign classifier, we proceeded to train the classifiers for Yield sign and Signal Ahead sign. We trained 3 classifiers each for both these signs while increasing the number of positive images from 600, 900, 1200. Even after increasing the number of positive images up to 1200 the resulting classifiers were not efficient enough to detect the signs in images. As each training had resulted in a loss of approximately 3.5 days, we realized that this could not be covered as part of this project and restricted ourselves to the stop sign detection.

A.2.1. Outcomes

- From the many experiments we carried out, we learned that there is no fixed number of samples that will yield a decent result.
- Future work can be done on training the U.S traffic signs, since there are no classifiers available for them . With the growing market for autonomous vehicles and assisted driving technology, having trained classifiers for the traffic signs might prove to helpful.

B. EXECUTION SUMMARY

This section specifies the week by week timeline for project completion.

1. Mar 6 - Mar 12, 2017: During this week we created virtual machines on Chameleon cloud using Cloudmesh and submit the project proposal.

2. Mar 13-Mar 19, 2017: Deployed Hadoop cluster to Chameleon cloud using Cloudmesh and develop Ansible playbook to install the required software packages to the clusters (OpenCV, Python and dependencies)

3. Mar 20-Mar 26, 2017: Collated data for training and test data sets and trained stop sign classifier. Developed Ansible playbook to deploy Hadoop and Spark to the cloud machines.

4. Mar 27-Apr 02, 2017: Trained data for stop and yield sign classifiers using OpenCV. Developed Ansible playbook to setup the OpenCV python environment on Spark clusters.

5. Apr 03-Apr 09, 2017: Trained data for signal ahead sign using OpenCV. Test stop sign classifier on local machine and chameleon cloud.

6. Apr 10 - Apr 16, 2017: Tested classifier on Spark and created deployable software package using shell script.

7. Apr 17-Apr 23, 2017: Completed project report and developed benchmarks for the project.

Using Hadoop and Spark for Big Data Analytics: Predicting Readmission of Diabetic patients

KUMAR SATYAM[1,*], PIYUSH SHINDE[1,**], AND SRIKANTH RAMANAM[1,***]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*] Corresponding authors: ksatyam@indiana.edu
[**] Corresponding authors: pshinde@iu.edu
[***] Corresponding authors: srikrama@iu.edu

project-004, May 4, 2017

This project proposes and demonstrates the use of Hadoop and Spark on cloud to run predictive analytics using machine learning on large amount of data. Our case study is to predict the readmission likelihood for diabetes patients using their available medical history.

Keywords: Hadoop, Spark, MLlib, Ansible, Cloudmesh Client, Predictive Analysis

Report: https://github.com/cloudmesh/sp17-i524/tree/master/project/S17-IR-P004/report
Code: https://github.com/cloudmesh/diabetic

CONTENTS

1. INTRODUCTION

The idea behind this project is to introduce Hadoop-Spark distributed cluster for building predictive analytics applications. Spark allows us to build applications that are scalable and faster when compared to standalone applications performing similar analytics tasks.

We chose the case study of predicting the likelihood of a diabetes patient getting readmitted within 30 days from the date of discharge using his/her available medical data. Predictive analytics based on medical data to provide healthcare is an active area of research.Predicting readmission likelihood is aimed to providing better care to patients. Readmissions happen due to deterioration of patients' health. Readmission predictions help doctors and patients to take preventive steps to avert medical emergencies that need hospitalization.

We approached this problem as a classification problem to classify the patients into 'Yes' or 'No' classes, indicating whether the patient is likely to be readmitted or not in the next 30 days. We used different machine learning algorithms on the available data, after some pre-processing, to predict the same. The accuracy percentages obtained for all the utilized classification algorithms are included in the report.

Our other important goal is to propose an end-to-end solution that is scalable and faster than a standalone predictive analytics application. While our dataset has about 100,000 records, anticipating real-world scenarios with huge amounts of data we are proposing a Hadoop based solution. We are utilizing Spark for

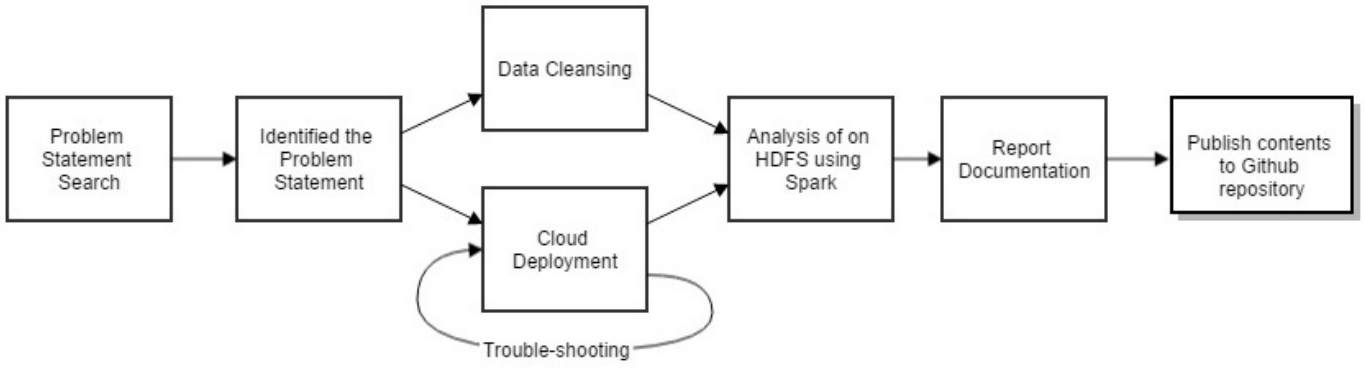


Fig. 1. Workflow of the project

its faster processing [1] and advanced analytics through packages like MLlib [2], which provides several commonly used machine learning algorithms. Finally we are implementing this solution over cloud infrastructure to meet our infrastructure requirements. This helped us demonstrate an end-to-end solution closer to real-world scenarios, where enterprises utilize cloud infrastructure in a pay per-use model. This helped us save time and resources in setting up the infrastructure.

We deployed our solution on two different clouds and obtained metrics to assess the infrastructure performance with our solution. We also deployed our solutions on a distributed Hadoop/Spark environment built on on-premise machines. We also automated the deployment of Spark cluster on cloud virtual machines using Ansible. We compared the performance metrics of all the three infrastructure choices, with respect to our application and included them in this report. Our project workflow can be seen in the Figure 1.

2. ARCHITECTURE

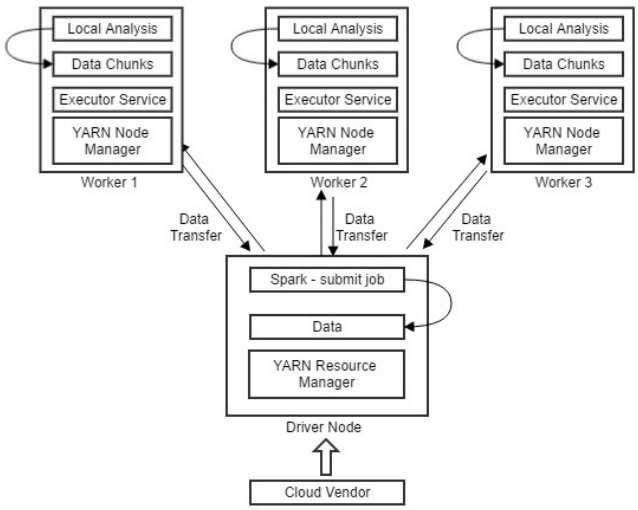


Fig. 2. Architecture Diagram

Figure 2 gives an overview of our solution's architecture. We deployed a spark driver node and three worker nodes. A driver node is a node that runs the driver program. It declares the transformations and actions on RDDs (Resilient Distributed Datasets) of data and submits such requests to the master [3]. In practical terms, the driver is the program that creates the Spark Context [4], connecting to a given Spark Master. It is a node where the yarn resource manager resides. A worker node is a node, which executes the program that involves individual data analysis task. Running the spark-submit script from the master node starts the spark job. It divides the data into data chunks and transfers them to individual worker nodes. Then a processing task is performed on the data chunks on the individual worker nodes. The processed data and analytics results are then written back to the HDFS file system as needed.

We then deployed Hadoop and Spark on these machines to setup a HDFS Spark cluster on our cloud and on-premise machines. We stored our dataset on the Hadoop's HDFS file system. Finally, we ran our predictive analytics application, which utilizes Mllib, on Spark cluster by launching it using spark-submit.

3. TECHNOLOGIES

Table 1. List of technologies used

Technology	*Usage*
Hadoop[5]/**Spark** [6]	Big Data Technologies
Python[7]	Development
MLlib[2]/**scikit-learn**[8]	Machine Learning Library
GitHub[9]	Project Repository
Ansible[10]	Application Deployment & Configuration Management
Chameleon[11], **JetStream**[12], **VirtualBox**[13]	Benchmarking
LaTex [14]	Document Preparation

We used specific technologies for specific tasks in this project as listed in Table 1:

- **Hadoop**: Apache Hadoop is a framework for processing and storing huge amounts of data, commonly known as 'Big

Data', in a distributed applications. It allows users to build scalable and highly available data applications. Hadoop has four modules: HDFS, YARN, MapReduce and Hadoop Common. Hadoop Distributed File System (HDFS) allows users to store large amounts of data. YARN is the framework for job scheduling and resource management. MapReduce supports parallel processing of large data sets stored in the distributed environment through HDFS. Hadoop Common provides utilities that support other Hadoop modules.

- **Spark**: Spark runs on Hadoop and provides faster data processing capabilities for data on HDFS. It primarily uses a new data structure called Resilient Distributed Dataset (RDD) for processing. RDD is a read only multiset of data items distributed over cluster of virtual machines. Spark also has a new feature of fault tolerance and in the event of a primary master node failure, the secondary master takes over. Spark, unlike Hadoop applications, allows the iterative reading and writing in-memory. After processing it writes the data to HDFS.

- **Python**: We chose python as per our programming language. Python is one of the programming languages supported by Spark API through pyspark. We used it because of its simple syntax and data manipulation capabilities.

- **scikit-learn**: It is an open source Python library that provides Machine Learning algorithms and other utilities to preprocess and visualize data [8].

- **MLlib**: Spark MLlib is the Spark's machine learning library provides machine learning algorithms that can be applied on Resilient Distributed Datasets [15]. It also provides other data manipulation utilities. MLlib has API available in Java, Python, Scala and R [2].

- **GitHub**: GitHub is 'a web-based Git or version control repository and Internet hosting service' [16]. We used Github repositories to store all the files related to documentation, Ansible scripts and python code.

- **Ansible**: We are using Ansible for automating deployment of our software on cloud. We used Ansible scripts to automate deployment of cloudmesh client along with its prerequisites like pip, virtualenv etc. We also used Ansible for automating deployment of Hadoop and Spark.

4. CLOUD INFRASTRUCTURE

We have setup the required infrastructure by provisioning virtual machines on two cloud vendors, Chameleon, Jetstream and our on-premise machines.

4.1. Chameleon Cloud

Chameleon is a collaborative cloud service primarily meant for research community. It allows users to explore problems ranging from the creation of Software as a Service to kernel support for virtualization. It is a good example of IAAS loaded with software defined networking and optimized virtualization technologies. We created three virtual machines on this cloud. One for Master node and two for worker nodes of Spark.

4.2. Jetstream Cloud

Jetstream is a cloud service which aims to provide researchers Jetstream's development was led by Indiana University's Pervasive Technology Institute (PTI) in collaboration with other universities [12] across the United States. This cloud service was used to provision the necessary virtual machines. We created three virtual machines on Jetstream for Spark cluster nodes.

4.3. Virtual Box

It is a fully virtualized hypervisor which gives the ability to spawn virtual machines in local commodity hardware. In fully virtualized environment, the guest OS is not aware of the underlying resources on which it is running as the hypervisor creates a complete simulation of the underlying hardware.

A brief comparison of multiple attributes of the clouds used are displayed in Table 2.

Table 2. Comparison of cloud vendors

Clouds	*Chameleon*	*Jetstream*	*VirtualBox*
CPU	Intel Xeon X5550	Dual Intel E-2680v3	Intel Core i5-6200U
RAM	2 GB	2 GB	2 GB
Number of CPU's	1	1	1
CPU Cores	1	1	2
CPU Speed	2.3 GHz	2.3 GHz	2.3 GHz

5. AUTOMATED DEPLOYMENT: ANSIBLE

The Ansible playbooks allow us to install and enable several packages in different VMs simultaneously. This removes the overhead of the logging into individual machine to install different packages and services.

For this project, we used Ansible to automate the deployment of spark and its prequisites. The Ansible script is written such that we can leverage the cloudmesh client technology to deploy the spark cluster. The steps involved in the script can be seen in Figure 3.

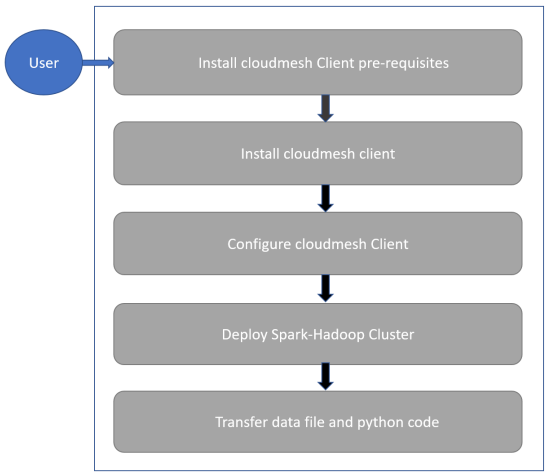


Fig. 3. Automated Cloud Deployment using Ansible

We used the below listed Ansible playbooks for our project:

1. *inventory*: This file is used by the playbook to install packages and services on target servers. This Inventory file consists of target servers and these servers can be categorized into groups like webservers, dbservers etc. Under each group, we can list the server which serves common purpose.

2. *playbook-cloudmesh-first-time-install.yml*: This file was used for deployment of cloudmesh client. It installs all the prerequisite of cloudmesh client first, checks if the ssh keys are already present, and then deploys the cloudmesh client.

3. *cloudmesh-first-time-configure.yml* : It is used to configure the cloudmesh after installation. The main tasks for this playbook include configuring cloud provider name and cloud user name, adding the ssh-key to the cloudmesh database, and upload the ssh key to the cloud.

4. *deployment-playbook.yml:* It is used to deploy the hadoop-spark cluster in chameleon and jetstream clouds. The main tasks include defining a cluster, defining a hadoop-spark stack on the cluster, syncing the clustering with github and deploying the hadoop cluster.

5. *tranfer-files-remotely.yml:* It is used to transfer pyspark codes and input files from local machine to report machines. The main tasks include reset of cloudmesh database, upgrade the python-pip, uninstall and install of cloudmesh client.

6. *upgrade-cloudmesh.yml:* It is used only when there is a need to upgrade.

We developed Ansible playbook for installation and configuration of cloudmesh client. We also developed playbook to deploy hadoop/spark cluster on different clouds. After successful deployment of Hadoop-Spark/hadoop cluster we require the python modules that perform data analysis and the data files in spark master node. For this, we developed an Ansible playbook to automate the manual process of uploading the file, authentication and downloading the file. We invoked the Ansible scripts through shell script, running on local Virtual Machines.

6. DATA CLEANING AND PRE-PROCESSING

We approached the problem from a pure data perspective to address the challenge of lacking medical domain knowledge. For some basic information, we relied on the dataset description on UCI website [17], ICD-9 [18] and earlier studies [19]. The initial data set is publicly available on the UCI Machine Learning Repository [17]. The initial data set has information extracted from the database satisfying the following criteria [19]:

- Each row corresponds to an inpatient encounter (a hospital admission).
- All of the encounters are "diabetic" encounters, that is, one during which any kind of diabetes was entered to the system as a diagnosis.
- Each encounter also corresponds to a patient stay between 1 and 14 days.
- Laboratory tests were performed during the encounter.
- Medications were administered during the encounter.

101,766 encounters were present in the data set that satisfy the above five inclusion criteria. Each encounter consists of 55 features describing the diabetic encounters, including demographics, diagnoses, diabetic medications, number of visits in the year preceding the encounter, and payer information. We defined the readmission field with 2 values: "YES," for cases where the patient was readmitted within 30 days of discharge and "NO," for both readmission after 30 days scenario and no readmission at all.

Diagnosis 1, 2 and 3 had many categorical values in the form of ICD-9 codes and had missing values. These ICD-9 codes were sorted and grouped into 9 categories, namely Circulatory, Respiratory, Digestive, Diabetes, Injury, Musculoskeletal, Genitourinary, Neoplasms and Other based on the ICD-9 codes [19]. The missing values were assigned the group 'Other'.

This data set had several features with empty fields. So we removed the features missing high percentages of data as they affect our analysis. The removed features were weight, medical specialty and payer code. The race attribute had 2% missing values which were filled by the mode value 'Caucasian'.

Observations only with unique patient ids were considered, excluding those with discharge disposition corresponding to the patient's death.

We filtered our data set according to the above-mentioned constraints and retained 62,937 encounters each corresponding to an unique patient and 55 features describing such encounter. We prepared three data-sets to test the multiple machine learning algorithms (Stochastic Gradient Descent, Gaussian Naïve Bayes, K-means and Decision Tree). Finally we also removed the patient id and encounter id as they were not relevant to learning algorithms.

We used one hot encoding to convert the categorical data features to numerical data. This creates new dummy features to represent the categorical data in numerical format.

The first data set was prepared using one hot encoding on the original data (with 62,937 observations), that resulted in 136 features representing the original 55 features.

For our second data set we implemented feature selection using a Variance Threshold algorithm that removes all low-variance features [20]. We set a variance threshold of '0.8'.

The data set B was formed using this algorithm, which helped extract 26 features.

The original data contains the age attribute grouped in 10-year intervals from 0 to 100 years. For the third data set we grouped the 10 intervals to 3 intervals by combining the age groups younger than 30, 30-60 and older than 60 years. One-hot encoding was then applied to form the third data set. This data set contained 129 features.

7. PRELIMINARY DATA ANALYSIS

The unit of our analysis is an encounter; to keep the observations independent, we only analyzed one encounter per patient. We performed early data analysis in python in a local machine. We implemented four classification algorithms using scikit-learn library on each of the 3 data sets created. The data sets were first divided in training and testing set. The training set had about 80% observations (5000 observations approx.) whereas the testing set had the remaining 20% observations (12937 observations).

Each of the algorithms provided by scikit-learn were used following in the manner:

Table 3. Results from scikit-learn

Classification Technique	*Number of Features*	*Accuracy (%)*
SGDClassifier[21]	136	90.68
	26	86.31
	129	86.96
GaussianNB[22]	136	10.43
	26	88.30
	129	9.96
KMeans[23]	136	55.26
	26	55.24
	129	55.26
DecisionTreeClassifier[24]	136	83.35
	26	82.50
	129	83.28

1. Create and fit a model using the observations and readmission of the training set.
2. Predict labels of the testing set.
3. Calculate the accuracy using the predicted labels and true labels of the testing set. The parameters needed in implementing the above-mentioned algorithms were set so as to be valid to our data, give optimum results and make the results to be reproducible.

KMeans clustering gave us approximately 55% accuracy with all the three data sets. Though it is an unsupervised learning algorithm, we used it to examine if the clustering divided the data into readmission classes, Yes and No, to an acceptable level. We concluded that clustering based on the Euclidean distance may not be the right approach for this classifying this data set. The accuracy percentages obtained for other classification algorithms can be found in the Table 3.

8. DATA ANALYSIS ON SPARK CLUSTER

We performed data analysis on Spark cluster using pyspark MLlib on data stored on HDFS.

8.1. Start the Spark service

Start the service of spark using the following command:

```
$SPARK_HOME/sbin/start-all.sh
```

Stop the service of spark using the following command:

```
$SPARK_HOME/sbin/stop-all.sh
```

Using the command 'jps' we get a list of the following services:

```
nodemanger
resourcemanger
master
namenode
applicationmaster
```

8.2. Data Storage

Uploading input data and code to Driver Node After the Spark setup is ready for the deployment, the data is pushed from the localhost to the remote spark master node. For this we used an Ansible script.

1. Ansible script

   (a) In the host files we set the target master(driver node) IP address as follows:

   ```
   [remotehosts]
   129.114.33.106 ansible_ssh_user=cc
   ```

   (b) Now , add the following entry in the yaml file to transfer the file to destination

   ```
   - hosts: remotehosts
     tasks:
       - name: Transfer file from local to
       satyam-001
         synchronize:
           src: /home/<username>/ansible
           script/ansible-spark/traindat.csv
           dest: /home/cc/
           mode: push
         delegate_to: 127.0.0.1
   ```

2. Using Github we uploaded the input data csv file and python execution code to a git repository. We installed git package in the spark driver node. We used 'wget' command with the repository path to download the data set to the virtual machine.

After the data is downloaded to the virtual cloud machine, we uploaded the file to HDFS through the following command.

```
Hdfs dfs  -put  <source file path>  <hdfs-folderpath>
```

This HDFS file serves as an input for our analytics application.

8.3. Launching Data Analysis Application

We used python programming language to develop an application that performs predictive analytics tasks. We leveraged pyspark.mllib library for machine learning algorithms.

We launched our application code using the following command

```
$ ./bin/spark-submit --class path.to.your.Class
--master yarn --deploy-mode cluster [options]
<app jar> [app options]
```

There are 4 main steps to each implementation of.

- Input formatting: MLlib classes expect RDD's of LabeledPoints. For this we parsed the data and converted each entry into a LabeledPoint , with label specifying the true output class.
- Next, the processed data frame is divided into train and test datasets.
- Train the model with the algorithm and training data
- After training, We used the model to predict the classes for test data and calculate the accuracy

Table 4. Results from Spark MLlib

Classification Technique	*Number of Features*	*Accuracy (%)*
Logistic Regression SGD[25]	136	90.88
	26	90.99
	129	90.88
Naive Bayes[26]	136	84.97
	26	85.02
	129	84.987
K-means[27]	136	55
	26	55
	129	55
Decision Tree[28]	136	91.01
	26	90.75
	129	90

9. RESULTS

Kmeans did not successfully separate the data class wise as observed in scikit-learn library, as shown in Table 4. We used supervised classification algorithms namely decision tree, logistic stochastic gradient descent and naïve bayes. We obtained good accuracies of 80-90% with classification algorithms.

Similar analysis in a real world scenario can be utilized to predict the readmission likelihood using medical records of diabetes patients. This enables doctors to pay special attention to those patients, identify the causal factors and give preventive care.

10. BENCHMARKING

We performed benchmarking to assess the performance with respect to four different aspects. (1) Spark deployment time in multiple clouds with variation in number of virtual machines. (2) Run-time of different algorithms on multiple clouds with variation in number of virtual machines. (3) File transfer time of two files of different size in multiple clouds. (4) Accuracy of multiple algorithms in Spark MLlib and scikit-learn.

10.1. Comparison of Spark Deployment Time

We compared the Hadoop-Spark deployment time with different number of machines on each of the clouds, Jetstream and Chameleon. In Table 5, we can see the Hadoop-Spark deployment times for clusters of different sizes on Chameleon and Jetstream clouds.

The variation of Hadoop-Spark deployment times on Chameleon and Jetstream clouds is shown in Figure 4. We observed that Spark deployment times on Jetstream are lesser than those observed on Chameleon, with the exception of cluster with 5 VMs. This aberration was caused due to a network issue, which was confirmed from the deployment logs.The log files recorded several unsuccessful ping requests to the cluster nodes before successful completion of deployment.

Table 5. Hadoop-Spark deployment time

Number of VMs	*Chameleon (seconds)*	*Jetstream (seconds)*
2	631.11	484.62
3	657.86	496.15
5	763.14	1399.54
7	1089	806.36

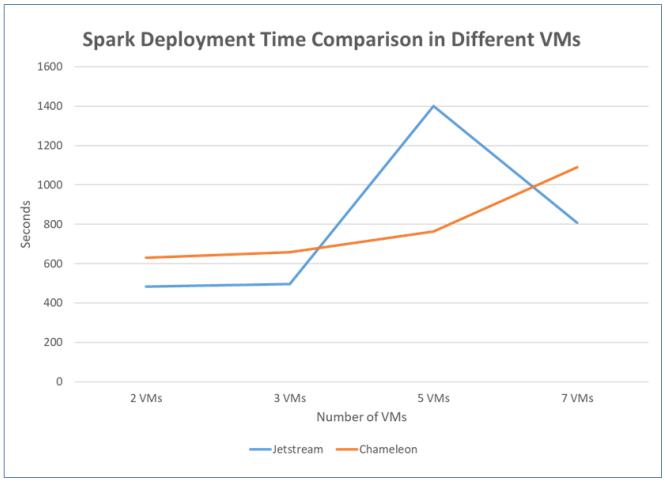


Fig. 4. Comparison of Spark deployment results

10.2. Computation time of algorithms in clouds

We can see in Table 6 the computation time required for running different Spark MLlib algorithms in multiple VMs in Chameleon and JetStream clouds.

Table 6. Computation time of algorithms in clouds in multiple VMs

Algorithms	*Number of VMs*	*Chameleon (seconds)*	*Jetstream (seconds)*
Logistic Regression SGD	1	91.67	115.55
	3	82.57	102.71
	5	31.38	57.23
Naive Bayes	1	66.09	58.48
	3	58.44	45.89
	5	23.2	36.47
Decision Tree	1	72.36	86.69
	3	60.62	71.54
	5	30.25	50.37

Figure 5 shows the performance of multiple machine learning algorithms provided by Spark MLlib on clusters in Chameleon, Jetstream and Virtual Box.

Figures 6 and 7 shows the run-time comparison for machine learning algorithms using Spark MLlib in Jetstream and Chameleon, for predicting readmission likelihood, for 3 and 5

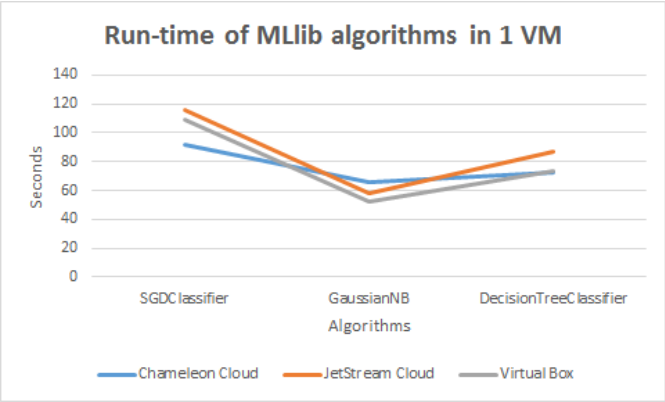


Fig. 5. Run-time in different clouds.

VMs. We can see that the run-time decreased with increase in the number of cluster nodes for all algorithms.

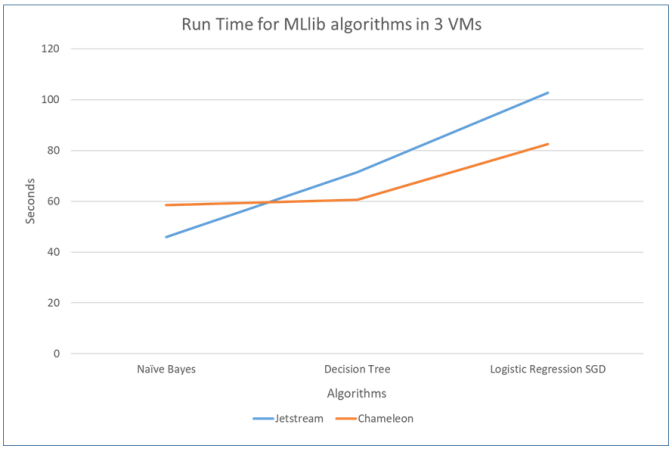


Fig. 6. Run-time in different clouds for 3 VMs

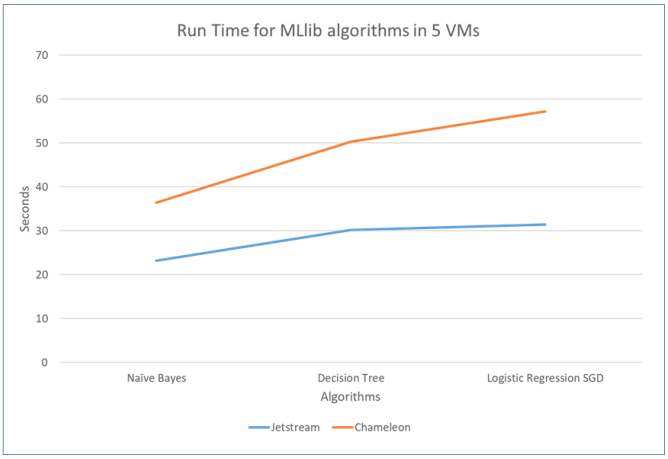


Fig. 7. Run-time in different clouds for 5 VMs

10.3. Accuracy comparison of multiple algorithms in Spark MLlib vs scikit-learn

We have recorded several metrics while running each of the four algorithms in different clouds. The four algorithms are SGD Classifier, Gaussian Naïve Bayes, kmeans and Decision Tree Classifier. The variations in run-time and deployment time are caused due to factors like:

1. *The complexity of the algorithm:* For example, kmeans clustering has O(n log n) time complexity which is worse than Gaussian Naïve Bayes and hence takes more time.

2. *The network latency between the hosts:* The VMs which are provisioned on clouds can be on different hosts spread across different racks of datacenters and even datacenters located in different geographies. This may result in network latency and affect the run-time of a program running in a distributed environment.

We compared the accuracies for different MLlib algorithms using pyspark.mllib package. We can see from Figure 8 the accuracies achieved for multiple algorithms in MLlib. While kmeans gave the lowest accuracy, rest of the algorithms gave similar higher accuracies.

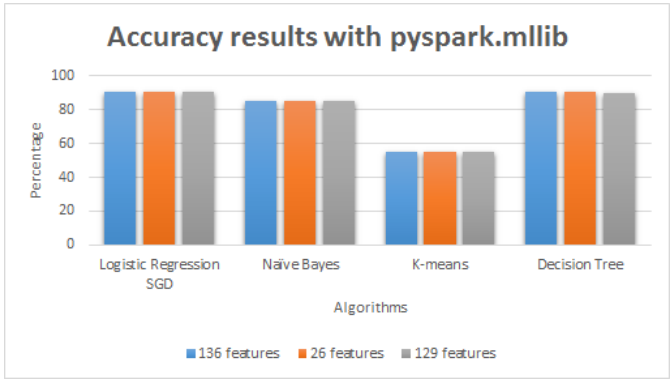


Fig. 8. Accuracy Results with pyspark.mllib

We compared the accuracies achieved from Spark MLlib and scikit-learn algorithms. Figure 9 shows accuracy results of multiple machine learning algorithms in scikit-learn vs MLlib.

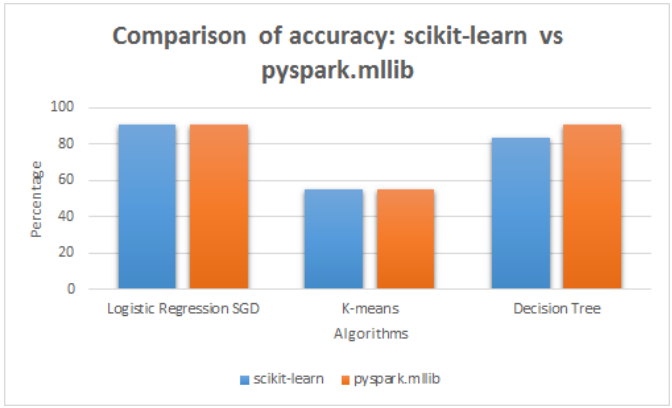


Fig. 9. Accuracy Results: scikit-learn vs pyspark.mllib

11. CONCLUSION

We demonstrated the implementation of a big data based predictive analytics solution using Hadoop, Spark and Spark's MLlib machine learning algorithms. We predicted the readmission

likelihood using MLlib with an accuracy of 90% by analyzing the medical data stored on HDFS. We also compared the results of Spark's MLlib [2] algorithms against scikit-learn's algorithms and found that both yielded similar results.

While Hadoop provided us a distributed file system for storing large amounts of data, Spark provided us big data processing capabilities and several libraries to accomplish several common processing and analytics tasks. We were successfully able to deploy the Spark-Hadoop cluster infrastructure on Chameleon, Jetstream and on-premise virtual box clouds. We developed several shell scripts and Ansible playbooks to automate the deployment. These technologies can be used to build similar solutions for real world scenarios requiring processing and analytics of big data. Our solution can also be extended to build analytics applications that process streams of data in real-time using Spark Streaming [29] and MLlib.

12. ACKNOWLEDGMENTS

This project was a part of the Big Data Software and Projects (INFO-I524) course. We would like to thank Professor Gregor von Laszewski and the associate instructors for their help and support during the course.

REFERENCES

[1] Databricks, "Apache Spark," Web Page, accessed: 2017-04-21. [Online]. Available: https://databricks.com/spark/about

[2] Apache, "Apache Spark MLlib," Web Page, accessed: 2017-04-21. [Online]. Available: http://spark.apache.org/mllib/

[3] Apache, "Class RDD<T>," Web Page, accessed: 2017-04-21. [Online]. Available: https://spark.apache.org/docs/1.6.2/api/java/org/apache/spark/rdd/RDD.html

[4] Apache, "Class RDD<T>," Web Page, accessed: 2017-04-21. [Online]. Available: https://spark.apache.org/docs/2.0.2/api/java/org/apache/spark/SparkContext.html

[5] Apache, "Welcome to Apache™ Hadoop®!" Web Page, accessed: 2017-03-12. [Online]. Available: http://hadoop.apache.org/

[6] Apache, "Apache Spark: Lightning-fast cluster computing," Web Page, accessed: 2017-03-12. [Online]. Available: http://spark.apache.org/

[7] P. S. Foundation, "python," Web Page, accessed: 2017-03-12. [Online]. Available: https://www.python.org/

[8] D. Cournapeau, "scikit-learn," Web Page, accessed: 2017-04-21. [Online]. Available: http://scikit-learn.org/stable/

[9] T. Preston-Werner, "GitHub," Web Page, accessed: 2017-04-21. [Online]. Available: https://github.com/

[10] R. Hat, "ANSIBLE," Web Page, accessed: 2017-03-12. [Online]. Available: https://www.ansible.com/

[11] C. Cloud, "Chameleon," Web Page, accessed: 2017-04-21. [Online]. Available: https://www.chameleoncloud.org/

[12] I. University, "Jetstream," Web Page, accessed: 2017-04-21. [Online]. Available: https://jetstream-cloud.org/

[13] Oracle, "VirtualBox," Web Page, accessed: 2017-04-21. [Online]. Available: https://www.virtualbox.org/wiki/VirtualBox

[14] LATEX, "The LATEX Project," Web Page, accessed: 2017-03-12. [Online]. Available: https://www.latex-project.org/

[15] Apache, "Machine Learning Library (MLlib) Guide," Web Page, accessed: 2017-04-21. [Online]. Available: http://spark.apache.org/docs/latest/ml-guide.html

[16] W. Inc., "GitHub," Web Page, accessed: 2017-04-21. [Online]. Available: https://en.wikipedia.org/wiki/GitHub

[17] U. M. L. Repository, "Diabetes 130-US hospitals for years 1999-2008 Data Set," Web Page, accessed: 2017-03-12. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Diabetes+130-US+hospitals+for+years+1999-2008#

[18] U. D. of Health and H. S. (HHS), "ICD9Data.com," Web Page, accessed: 2017-04-21. [Online]. Available: http://www.icd9data.com/

[19] B. Strack, J. P. DeShazo, C. Gennings *et al.*, "Impact of hba1c measurement on hospital readmission rates: Analysis of 70,000 clinical database patient records," *BioMed Research International*, vol. 2014, no. 781670, April 2014, published as an Article. [Online]. Available: https://www.hindawi.com/journals/bmri/2014/781670/

[20] scikit-learn developers, "sklearn.feature_selection.VarianceThreshold," Web Page, accessed: 2017-04-21. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.feature$_$selection.VarianceThreshold.html

[21] scikit-learn developers, "sklearn.linear_model.SGDClassifier," Web Page, accessed: 2017-04-21. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.linear$_$model.SGDClassifier.html

[22] scikit-learn developers, "sklearn.naive_bayes.GaussianNB," Web Page, accessed: 2017-04-21. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.naive$_$bayes.GaussianNB.html

[23] scikit-learn developers, "sklearn.cluster.KMeans," Web Page, accessed: 2017-04-21. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

[24] scikit-learn developers, "sklearn.tree.DecisionTreeClassifier," Web Page, accessed: 2017-04-21. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[25] Apache, "pyspark.mllib package," Web Page, accessed: 2017-04-21. [Online]. Available: http://spark.apache.org/docs/2.0.0/api/python/pyspark.mllib.html#pyspark.mllib.classification.LogisticRegressionWithSGD

[26] Apache, "Naive Bayes - RDD-based API," Web Page, accessed: 2017-04-21. [Online]. Available: https://spark.apache.org/docs/2.1.0/mllib-naive-bayes.html

[27] Apache, "Clustering - RDD-based API," Web Page, accessed: 2017-04-21. [Online]. Available: https://spark.apache.org/docs/2.1.0/mllib-clustering.html#k-means

[28] Apache, "Decision Trees - RDD-based API," Web Page, accessed: 2017-04-21. [Online]. Available: https://spark.apache.org/docs/2.1.0/mllib-decision-tree.html

[29] Apache, "Spark streaming," Web Page, accessed: 2017-04-23. [Online]. Available: http://spark.apache.org/docs/latest/streaming-programming-guide.html

A. APPENDIX: WORK DISTRIBUTION

- Kumar Satyam [S17-IR-2031]
  - Developed Ansible playbook for installation of cloudmesh client.
  - Developed Ansible playbook for automated deployment of Spark on different clouds.
  - Deployed the initial test of Spark cluster on 1 virtual machine of Jetstream cloud.
  - Deployed a Spark cluster on 5 and 7 virtual machines of Chameleon cloud.
  - Developed Ansible script for file transfer from local VM to Spark master node on different clouds.
  - Executed K-means python script for predicting readmission and recorded the runtime on Spark cluster on 1 VM on Chameleon cloud.
  - Developed python code for predicting readmission using Spark MLlib algorithm Decision Tree.
  - Developed python code for predicting readmission using Spark MLlib algorithm K-means.
  - Tested python code of predicting readmission using Spark MLlib algorithms on a single VM spark cluster deployed on Virtual Box.

  - Executed Naive Bayes python script for predicting readmission and recorded the runtime on Spark cluster of 1, 3 and 5 VMs on chameleon cloud.
  - Executed K-means python script for predicting readmission and recorded the runtime on Spark cluster on 1 VM on Jetstream cloud.
  - Executed Decision Tree classifier python script for predicting readmission and recorded the runtime on Spark cluster on 1, 3 and 5 VMs on Jetstream cloud.
  - Benchmarked the algorithm runtimes of chameleon and Jetstream clouds.
  - Documented the sections: benchmarking, conclusion and references.

- Piyush Shinde [S17-IR-2035]
  - Performed preliminary data analysis.
  - Developed python code for preliminary analysis using sci-kit machine learning algorithm Gaussian Naive Bayes.
  - Developed python code for preliminary analysis using sci-kit machine learning algorithm SGD classifier.
  - Developed python code for preliminary analysis using sci-kit machine learning algorithm Decision Tree.
  - Developed python code for preliminary analysis using sci-kit machine learning algorithm K-means.
  - Developed Ansible playbook for configuration of cloudmesh client.
  - Performed initial test deployment of Spark cluster on 1 virtual machine of Chameleon cloud.
  - Deployed a Spark cluster on 2 and 3 virtual machines of Chameleon cloud.
  - Executed Naive Bayes python script for predicting readmission and recorded the run-time on Spark cluster of 1, 3 and 5 VMs on Jetstream cloud.
  - Documented the sections: introduction, technologies, architecture and preliminary data analysis.
  - Executed SGD classifier python script for predicting readmission and recorded the runtime on Spark cluster of 1, 3 and 5 VMs on Jetstream cloud.

- Srikant Ramanam [S17-IR-2028]
  - Performed data cleaning.
  - Installed cloudmesh client using developed Ansible playbook.
  - Configured cloudmesh client using developed Ansible playbook.
  - Performed initial test deployment of Spark cluster on 1 virtual machine of Virtual box.
  - Deployed a Spark cluster on 2 and 3 virtual machines of Jetstream cloud.
  - Deployed a Spark cluster on 5 and 7 virtual machines of Jetstream cloud.
  - Developed shell script for triggering spark deployment Ansible playbook.
  - Benchmarked spark deployment on chameleon and Jetstream clouds using shell script.
  - Developed python code for predicting readmission using Spark MLlib's algorithm SGD classifier.
  - Executed Decision Tree classifier python script for predicting readmission and recorded the runtime on Spark cluster on 1, 3 and 5 VMs on chameleon cloud.
  - Developed python code for predicting readmission using Spark MLlib's algorithm Naive Bayes.
  - Executed SGD classifier python script for predicting readmission and recorded the runtime on Spark cluster of 1, 3 and 5 VMs on chameleon cloud.
  - Documented the sections: automated deployment using Ansible, Data cleaning and Pre-processing.

Analysis Of People Relationship Using Word2Vec on Wiki Data

ABHISHEK GUPTA[1,*] AND AVADHOOT AGASTI[1,**]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*] Corresponding authors: abhigupt@iu.edu
[**] Corresponding authors: aagasti@iu.edu

project-1: Data mining for a wiki url , May 4, 2017

Wikipedia pages of famous personalities contain details like school, spouse, coaches, languages, alma-meter etc. This information is in free form text. In this project, we extract the people information from the Wikipedia page and to establish relationships between them. Specifically, we use the data of known relationships to derive the newer relationships.

Keywords: Cloud, I524, Chemeleon, Word2Vec, Jetstream, Cloudmesh, RAM

https://github.com/cloudmesh/sp17-i524/blob/master/project/S17-IR-P005/report/report.pdf

CONTENTS

1. INTRODUCTION

Word2Vec [1] is a group of related models that are used to produce word embedding. Word2Vec is used to analyze the linguistic context of the words. In this project, we created Word2vec model using Wikipedia data and news articles. Our focus is people names occurring in the Wikipedia data and to see if Word2vec can be used to understand relationship between people. Typically Wikipedia page for people and celebrities contain the entire family and friends, colleagues information. Our idea is to use Word2vec to see if using a smaller training set of known relationships whether we can derive similar relationship for anyone who has presence on Wikipedia. This mechanism can be then used to convert the data hidden in textual format to more structured data.

We used spark [2] to load the wiki data and create word vectors. We then used vector manipulations to derive the relationships.

2. DESIGN

Figure 1 shows the overall data pipeline for the project. The data pipeline has three important stages:

- Wiki crawler: Wiki crawler runs in batch mode on a standalone machine. It can download wikipedia data as explained in section 2.1. Crawler creates CrawlDB which is a collection of text files. This crawler can be replaced or

Name	Purpose
spark [2]	data analysis
sparkML [3]	machine learning
python [2]	development
ansible [4]	automated deployment

Table 1. Technology Name and Purpose

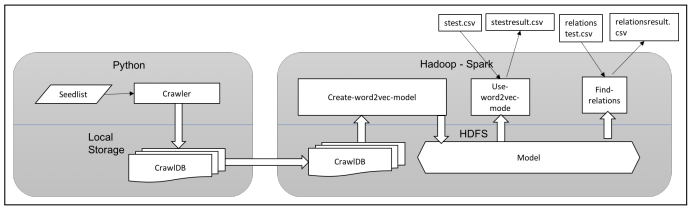


Fig. 1. Data Pipeline.

augmented with any web-crawler which can download or create the text files.

- News crawler: News crawler is responsible for downloading the news articles.
- CreateWord2VecModel: This component is responsible for creating the Word2Vec model for the text files in the CrawlDB. This model runs on spark and stores the model on HDFS. Section 2.3 describes this component in detail.
- UseWord2VecModel and FindRelations: These two components use the pre-created Word2Vec model to find synonym of a word or find the relationships. Section 2.4 describes these components in detail.

2.1. Wiki crawler

The Wiki Crawler component is useful to download the data from web. We implemented a simple crawler using Python which can deep traverse the wikipedia pages and download the text from it. In our crawler implementation, a user can specify the seed pages from wikipedia. User can also specify the maximum number of pages that are required to be downloaded. The crawler first downloads all the pages specified in the seedlist. It then extract the links from each wikipedia page and puts it in a queue which is internally maintained by the crawler. The crawler then downloads the the linked pages. Since this logic is implemented in recursive manner, the crawler can potentially download all the wikipedia pages which can be reached from the pages in the seedlist.

We followed the seedlist based crawler approach so that we can retrieve domain specific web pages. A well chosen seedlist can fetch large number of relevant web pages.

Figure 2 is the flowchart of the crawler implementation.

2.2. News crawler

News crawler is crawler implemented in Python. It executes in batch mode and download latest news article related to the topics configured in its seedlist. The news crawler uses Google APIs [5] to search the topics configured in the seedlist. Then it iterates over the result of each search, and downloads the

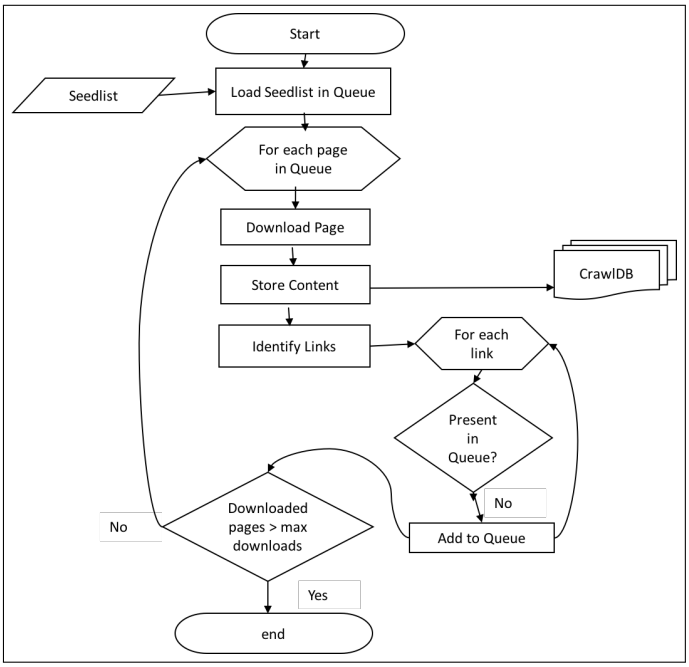


Fig. 2. Flowchart of crawler.

original HTML page contents. The textual portion of the HTML is extracted using goose python library [6]

2.3. Word2Vec model creation

CreateWord2VecModel is Spark application implemented in Python. This application is responsible for creating the Word2Vec model and storing it for later use. Figure 3 explains the steps involved in the Word2Vec model creation. We used Spark Feature Extraction [7] for implementing the steps involved in the Word2Vec model creation. These steps are explained below:

- Read crawled documents from HDFS
- For each crawled document, remove special characters from the text
- Tokenize text to create list of words
- Remove the stop words
- Create Word2Vec model
- Store the model on HDFS

Fig. 3. Steps involved in Word2Vec model creation.

2.4. Using the Word2Vec model to find synonyms and relations

The pre-created Word2Vec model can be queried to find the synonyms or relations between the words. In the context of Word2Vec, synonym of the word is the word that co-occur in similar context [8]. The *UseWord2VecModel* finds the synonyms

for the words provided in the *stest.csv* file. The results are stored in *sresults.csv* file.

The Word2Vec model is also used to find the relationships. [9] explains how the vector operations can be performed on word vectors to derive relationships. The *FindRelations* spark application performs the vector operations to find relationships. The relationtest.csv is input to the *FindRelations* application. The relationtest.csv has 3 words in each row. The application predicts the fourth word which has same relation to the thrid word as the second word related to first word. In the example below

Sachin,Anjali,Sourav

If *Anjali* is spouse of *Sachin*, then *FindRelations* application is expected to predict the first name of the person who is spouse of Sourav. The result of *FindRelations* is saved in relationsresult.csv.

3. DEPLOYMENT

The deployment on the cluster can be accomplished using 2 steps, assuming the cluster is up and running

- Step1: update hosts file *ansible-word2vec/hosts* with the IP of the master. The first node on the cluster becomes the master node.
- Step2: run the script *ansible-word2vec/run.sh*. This script will run the ansible playbooks to accomplish stage1 through stage4 of the deployment process.

Figure 4 shows the deployment stages. The two steps accomplish deployment in multiple stages as discussed in the sections below.

3.1. Stage1

As pre-requisite, we need to create a cluster with 1 or more nodes. We created a 3 node cluster using Cloudmesh [10] command line interface(CLI). Cloudmesh[10] CLI allows you to orchestrate virtual machines(VM) in a cloud environment. For this project, we have used Chameleon and Jetstream cloud providers to orchestrate the VMs using cloudmesh[10] CLI. We can orchestrate a 3 node cluster using following CLI:

```
cm reset
pip uninstall cloudmesh_client
pip install -U cloudmesh_client
cm key add --ssh
cm refresh on
cm cluster define --count 3 \
--image CC-Ubuntu14.04 --flavor m1.medium
cm hadoop define spark pig
cm hadoop sync
cm hadoop deploy
cm cluster cross_ssh
```

We are using Ubuntu14.04 image with m1.medium which comes with 2 CPU, 4GB memory. Also, the nodes created are having hadoop and spark add-ons. We can test the deployment by checking hdfs and spark-submit CLI work fine.

```
ssh cc@<cluster-ip>
sudo su - hadoop
hdfs
spark-submit
```

At this stage our cluster is ready for further deployments.

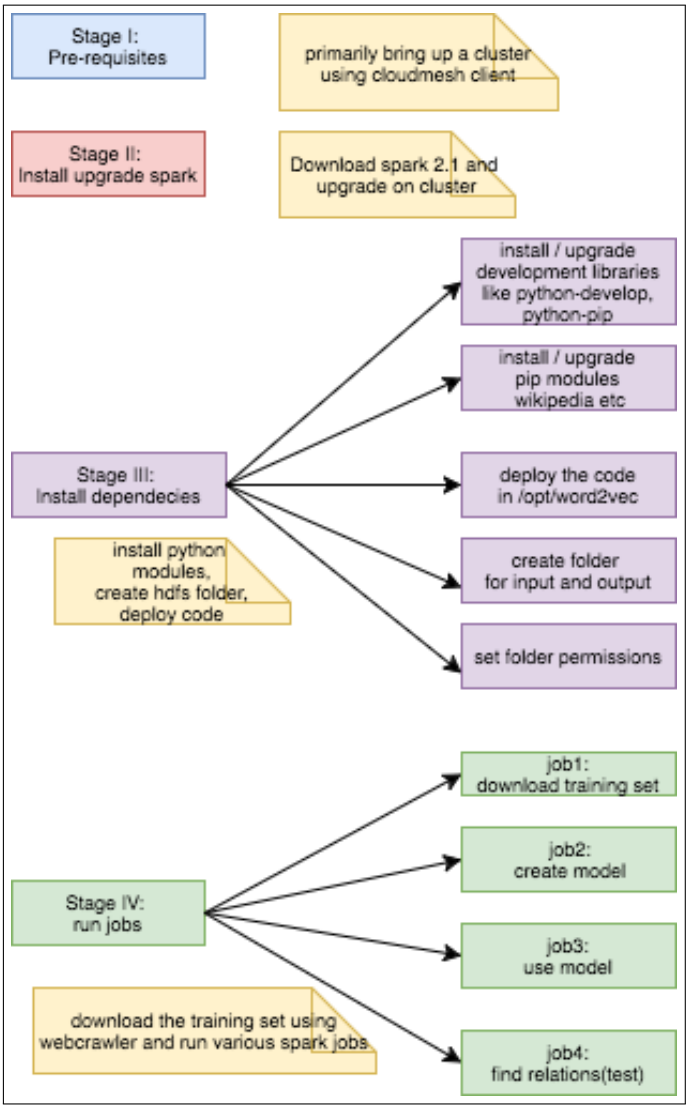


Fig. 4. Deployment stages.

3.2. Stage2

By default, cloudmesh installs spark 1.6 but our word2vec solution requires spark 2.1. We need to upgrade spark on the cluster. In order to do so we can run *install_upgrade_spark.yaml* ansible[4] playbook. This will download and unpack spark2.1 tar ball and further update the softlink to point to spark 2.1 folder.

3.3. Stage3

In this step, we upgrade the development libraries for python, and pip, install python modules like wkipedia, request etc, download the code from git repo and install it in */opt/word2vec* folder, set the folder permissions for the */opt/word2vec* folder so that it can be executed by hadoop user. These steps can be achieved using *word2vec_setup.yaml* playbook. After completing this stage, we are ready for running our word2vec solution on the cluster.

3.4. Stage4

This stage primarily deals with submitting the jobs for various purpose. Before we submit the jobs, we need to make sure input folder are created on hadfs. First, we run the crawler to download the training set and upload the data on hdfs. Further we run various jobs to created model and find relations. Along with these jobs we also run some monitoring jobs. The monitoring job queries spark metrics using

```
http://${spark_master}:4040
```

Stage4 steps can be accomplished using *word2vec_execute.yaml* playbook.

At the end of stage4 we also fetch the execution results from the cluster along with the metrics of execution times at various stages. The output files are fetched into */tmp/word2vec_results*

```
ls -1t /tmp/word2vec_results
jobs.csv
executors.csv
app.csv
stest.csv
relationstest.csv
stestresult.csv
relationsresult.csv
```

Files *jobs.csv, executors.csv, and app.csv* collect the execution time for various jobs. File *relationsresult.csv* file collects the results for sample relations corresponding to *relationstest.csv*. Similarly *stestresult.csv* collects the results corresponding to *stest.csv*.

3.5. Execution

3.5.1. Cleanup

We can execute the run from local system using *run.sh* located inside *ansible-word2vec*. Run script executes all stages sequentially on the remote system. To rerun word2vec, run the cleanup playbook *word2vec_cleanup.yaml* located inside *ansible-word2vec*. Cleanup remove the spark 2.1 binary and the soft link, remove */opt/word2vec* folder as well as any temporary files created during setup step.

3.5.2. Page size

The crawler downloads pages from wikipedia based on the config page count. To modify the page count, we can edit *ansible-word2vec/setupvariables.yml* and set the *max_pages* to desired page size. The crawler downloads individual pages and then combines the pages into a single file before submitting the spark jobs.

3.5.3. Test Results

Test results are downloaded to local machine in */tmp/word2vec_results* folder. Ansible execution log is saved in */tmp/word2vec-logfile.txt*. The log gets appended each time you execute the run script.

3.5.4. Troubleshooting

1. If the installation *run.sh* script fails in middle due to some reason, execute the cleanup script before re-triggering run script again. The run script may fail due to variety of reasons like failed to shh, hadoop not available etc
2. If the run script fails due to spark memory errors, you can modify the spark memory setting in *code/config.properties* push the code to a git feature branch for example spark_test. Modify *word2vec_setup.yaml* git section *version=master* to point to spark_test branch and execute the run script.
3. If hadoop goes into safe mode, goto the cluster namenode and execute the following

   ```
   /opt/hadoop$ bin/hadoop dfsadmin -safemode leave
   ```

   This will remove the cluster from safe mode.

4. BENCHMARKING

We used datasets of 2 different sizes to perform the benchmarking of the application. Table 2 shows the details of the two datasets. The CreateWord2VecModel spark application is most complex and time consuming application. We used this application for the benchmarking. We deployed the application on Chameleon cloud and Jetstream cloud. Table 3 shows the details for the cluster configurations on Chameleon and Jetstream clouds.

Table 2. Dataset Used for Performance Measurement

Parameter	Dataset1	Dataset2
Size of crawldb	1.4MB	7.4MB
Count of files in crawldb	100	500
Source	Wikipedia	Wikipedia

Table 3. Dataset Used for Performance Measurement

Parameter	Chameleon Cluster	Jetstream Cluster
Cluster name	cluster-005	cluster-010
Nodes	2	2
OS	Ubuntu 14.04	Ubuntu 14.04
Flavor	m1.medium	m1.medium
Secgroup	default	default
Assign floating IP	True	True
Cloud	chameleon	iujetstream

Figure 5 shows the total time taken by CreateWord2Vec application for Dataset1 and Dataset2 on the Chameleon and Jetstream cloud environments.

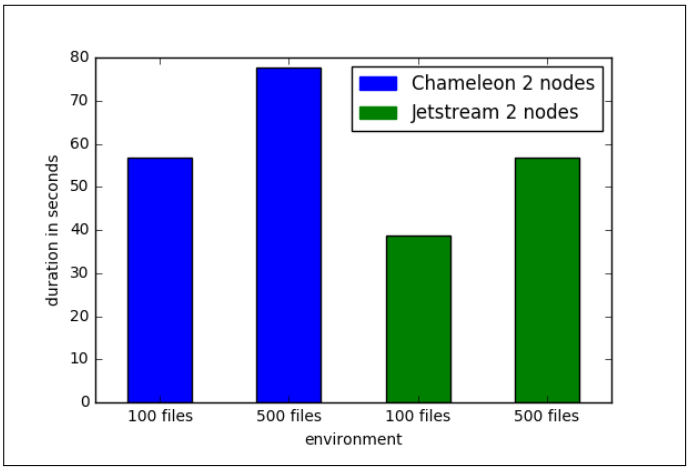


Fig. 5. Time taken by CreateWord2Vec

4.1. Working with large dataset

There are several configuration parameters added in the application to fine tune the behavior of the spark applications. When working with larger datasets, the spark applications can go out of memory. Following parameters can be configured in the *ansible-word2vec/setupvariables.yml* to handle such situation.

```
spark_executor_memory = <memory given to executor>
spark_driver_memory = <memory given to driver>
max_result_size = <maximum result size>
```

Table 4 show cluster configuration of Chameleon cloud used for 1k files dataset

Table 4. Cluster configuration used for performance measurement

Parameter	Chameleon Cluster
Cluster name	cluster-006
Nodes	4
OS	Ubuntu 14.04
Flavor	m1.medium
Secgroup	default
Assign floating IP	True
Cloud	chameleon

We tried running out solution on a 4 node cluster as described in **Table 4** with 1000 files on the crawlDB and the create model script failed with out of memory exceptions.

```
["OpenJDK 64-Bit Server VM warning:
INFO: os::commit_memory(0x00007ff67449f000, 12288, 0)
failed; error='Cannot allocate memory' (errno=12)",
"OpenJDK 64-Bit Server
VM warning: INFO:
os::commit_memory(0x00007ff6746a1000, 12288, 0)
failed; error='Cannot allocate memory' (errno=12)",
"#",
"# There is insufficient memory for the Java
Runtime Environment to continue.",
"# Native memory allocation (malloc) failed to allocate
12288 bytes for committing reserved memory.",
"# An error report file with more information is saved as:",
"# /home/hadoop/hs_err_pid25854.log"], "warnings": []}
```

We need to tune spark memory parameters here, since the words in model become too high and we need as much memory(RAM) to execute the solution or else it goes out of memory(OOM). For example for min_word_count = 5, the number of words in the model were 20839 and executor_memory of 4GB was enough. But for min_word_count of 3 which resulted in 30194 words in model, we have to give 6GB RAM. After making these changes to spark executor and driver memory we were able to run the solution successfully with 1000 files.

Figure 6 shows the total time taken by CreateWord2Vec application varying min word count on the Chameleon cloud with 4 nodes

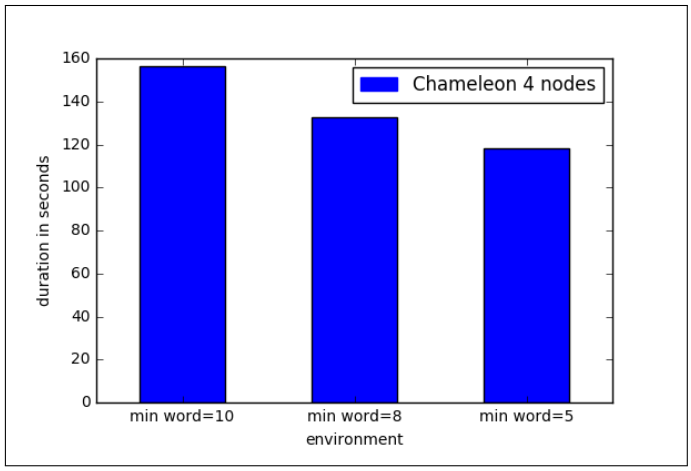


Fig. 6. Time taken on 4 node Chameleon cloud

5. DISCUSSION

The analysis of the results provided interesting insights. Section 5.1 provides key insights on our experiments with Word2Vec on Wikipedia data. Section 5.2 provide key insights on our experience of deployment and execution of Word2Vec application on Chameleon and Jetstream cloud.

5.1. Word2Vec app - key insights

We trained the Word2Vec model with the wikipedia data of Indian Cricket team members. We used *List of India ODI cricketers* as seedlist and downloaded 500 pages from Wikipedia. With this dataset, we observed that the *synonym* or correlated words were pretty accurate. Some interesting examples and its explanation:

sachin -> tendulkar

world -> cup

Sachin Tendulkar [11] is famous Indian cricket player. Naturally, the words *sachin* and *tendulkar* are highly correlated. The Word2Vec model was able to find this correlation.

Cricket World Cup [12] is one of the most viewed sporting event and considered as the flagship event of the international cricket. Hence the words *world* and *cup* are highly correlated in the context of cricket. The Word2Vec model was able to find this correlation.

However, when we tried to find the people relationships using the wikipedia dataset, we could not get good accuracy.

For example, *Anjali Tendulkar* is wife of *Sachin Tendulkar* while *Dona Ganguly* is wife of *Sourav Ganguly* who is another famous Indian Cricket player [13]. When we provided the test record of

sachin,anjali,sourav

we did not get good results.

After observing the wikipedia data, we concluded that there is lot of literature in wikipedia about the cricketers. However, there were very few mentions of their family members, coaches, schools etc. Due to this, we were not getting good results on the people relationships.

To augment the Wikipedia data, we decided to crawl news data which provide large amount of articles containing people and their relationships. We implemented the news crawler as explained in the section 2.2. After downloading about 200 news articles of 20 cricketers, our relationships discovery improved a lot. Some interesting examples are given below: *sachin,anjali,sourav,dona*

sachin,anjali,dhoni,sakshi

sachin,cricket,amitabh,hero

In first two examples, the Word2Vec model was able to identify the people-people relationships, while in third example, the Word2Vec model was able to identify the people-profession relationships.

5.2. Deployment of Word2Vec app on Chameleon and JetStream cloud - key insights

Both Chameleon and Jetstream run on openstack and work seamlessly using cloudmesh and ansible. There is a difference in the flavor for Chameleon and Jetstream, where m1.medium on Chameleon is different than m1.medium on Jetstream.

6. CONCLUSION

Using this project we conclude that we can use Word2Vev model on Wikipedia and news data to find the relationships between the people.

We further conclude that Word2Vec based analytics can be performed on public cloud systems like Chameleon cloud and Jetstream cloud. Our deployment automation, which is implemented using Cloudmesh and Ansible technology demonstrates the power of these technologies to achieve one touch deployment and execution of applications across multiple clouds.

7. ACKNOWLEDGEMENT

We acknowledge our professor Gregor von Laszewski and all associate instructors for helping us and guiding us throughout this project.

8. APPENDICES

Appendix A: Work Distribution The co-authors of this report worked together on the design of the technical solutions, implementation, testing and documentation. Below given is the work distribution

- Avadhoot Agasti
  - Implementation of wiki crawler and news cralwer in Python.
  - Implementation of CreateWord2VecModel in Spark.
  - Implementation of UseWord2VecModel and FindRelations in Spark.
  - Implementation of Python script MonitorSparkApp.
  - Analyzing the Word2Vec model results.
  - Testing of end to end flow on Chameleon cloud.
  - Performance testing and bug fixing in the spark application.
  - Writing related sections in this report.
- Abhishek Gupta
  - Implementation of Ansible scripts for deployment of Spark 2.1 which is required for the spark application.
  - Implementation of Ansible scripts for deployment.
  - Implementing the changes in the spark applications to get it working on HDFS.
  - Setting up and testing the end to end flow on Chameleon cloud.
  - Setting up and testing the end to end flow on Jetstream cloud.
  - Testing the crawler and Word2Vec applications for semantic correctness.
  - Gathering the perfomance statistics for comparison.
  - Writing related sections in this report.

REFERENCES

[1] "Word2Vec, learning vector representation of words," Web Page, accessed: 2017-02-26. [Online]. Available: https://en.wikipedia.org/wiki/Word2vec

[2] "Spark Python API (PySpark)," Web Page, accessed: 2017-02-26. [Online]. Available: https://spark.apache.org/docs/0.9.1/python-programming-guide.html

[3] "Spark ml programming guide," Web Page, accessed: 2017-02-26. [Online]. Available: https://spark.apache.org/docs/1.2.2/ml-guide.html

[4] Red Hat, Inc., "Ansible documentation," Web Page, Jan. 2015, accessed 2017-01-13. [Online]. Available: https://docs.ansible.com/ansible/index.html

[5] "Google custom search," Web Page. [Online]. Available: https://developers.google.com/custom-search/

[6] "Python-goose - article extractor," Code Repo. [Online]. Available: https://github.com/grangier/python-goose

[7] "Extracting, transforming and selecting features," Web Page, accessed: 2017-04-26. [Online]. Available: https://spark.apache.org/docs/latest/ml-features.html

[8] Y. Goldberg and O. Levy, "word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method," *CoRR*, vol. abs/1402.3722, 2014.

[9] "Vector representations of words," Web Page, accessed: 2017-04-26. [Online]. Available: https://www.tensorflow.org/tutorials/word2vec

[10] "Cloudmesh client," Code Repo. [Online]. Available: https://github.com/cloudmesh/client

[11] "Sachin tendulkar," Web Page, accessed: 2017-04-15. [Online]. Available: https://en.wikipedia.org/wiki/Sachin_Tendulkar

[12] "Cricket world cup," Web Page, accessed: 2017-04-15. [Online]. Available: https://en.wikipedia.org/wiki/Cricket_World_Cup

[13] "Sourav ganguly," Web Page, accessed: 2017-04-15. [Online]. Available: https://en.wikipedia.org/wiki/Sourav_Ganguly

cloudmesh cmd5 extension for AWS

MILIND SURYAWANSHI[1] **AND PIYUSH RAI**[2]

[1] *School of Informatics and Computing, Bloomington, IN 47408, U.S.A.*
[2] *School of Informatics and Computing, Bloomington, IN 47408, U.S.A.*

project-006, May 4, 2017

The cludmesh client will be extended to support cluster deployment on AWS using cmd5.

Keywords: Cloud, I524

https://github.com/cloudmesh/sp17-i524/blob/master/project/S17-IR-P006/report/report.pdf

1. INTRODUCTION

We are going to look at cloudmesh client [1] man pages, understand what features are already supported for cloud like chameleon and implement similar support for AWS. The new features will be provided under aws subcommand. We will study the technologies currently in-use by the project and analyze how they can be used further to achieve our objective.

2. DESIGN

TBD

3. TECHNOLOGY USED

- Cloud Mesh
- AWS
- Cmd5
- libcloud

4. STEPS

- Understand cloudmesh client.
- Propose changes and get reviewed.
- Open aws account and test basic commands.
- Make changes and test.
- Benchmark.

ACKNOWLEDGEMENTS

TBD

REFERENCES

[1] Web Page. [Online]. Available: https://github.com/cloudmesh/client

AUTHOR BIOGRAPHIES

Milind Suryawanshi received his BE (Electronics and Telecommunication) in 2010 from The University of Pune. His research interests also include Big Data analytics for intelligence and research.
Piyush Rai received his BE (Computer) in 2011 from The University of Pune. His research interests also include Big Data analytics for military intelligence and frinancial markets.

A. WORK BREAKDOWN

The work on this project was distributed as follows between the authors:

Milind Suryawanshi. TBD

Piyush Rai. TBD

Deploying a spam message detection application using R over Docker and Kubernetes

SAGAR VORA[1,*] AND RAHUL SINGH[1]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*] Corresponding authors: vorasagar7@gmail.com, rahul_singh919@yahoo.com

project-P007, May 4, 2017

In the last few decades, online spam has become one of the major problem for the sustainability of the Internet. Due to the excessive amount of spams, the quality of information available on the Internet has reduce drastically. Moreover spam messages are also creating problems among the various search engines available and the web users. This report aims at developing an application which would detect spam messages from actual meaningful messages using Pandas and R. For the purpose for parallelizing the process, we would deploy the application using Docker containers on the Kubernetes cluster using ansible scripts which would automate the deployment.

Keywords: Docker, Ansible, Kubernetes, R, Pandas, Spam, <add spam detection algorithms>

https://github.com/cloudmesh/sp17-i524/raw/master/project/S17-IR-P007/report/report.pdf

1. INTRODUCTION

Today, the Internet [1] has been adopted rapidly in the day to day life of people. It has provided a platform for information generation and consumption. Moreover, it is used on a daily basis to search for information and acquire knowledge. The online encyclopedia Wikipedia™ [2] provides a good example of a more socialized Internet because the content within Wikipedia™ is collectively generated by its users, rather than webmasters or designated editors. The ease with which content can be generated and published has also made it easier to create spam. Spam can be stated as any information which does not add value to a user of the web. Messages which are inappropriate, unsolicited, repeated and irrelevant can be all classified as spam.

So in this report, we are provinding an application that would identify valid messages and spam messages from a given dataset. For spam detection, we are using various techniques like Bayesian, <text here> ..... Moreover, deploying our application using Docker [3] containers on the Kubernetes [4] cluster will give a distributed approach.This would also speed up the process of identifying the spam messages. We have also deployed it on different cloud environments like Chameleon, JetStream, FutureSystem and have performed benchmark analysis of the application. This would let us the time taken by the algorithm on these cloud solutions.

Name of the Technology	Purpose in the Project
R	data analytics
Docker	container for the applicat
Kubernetes	cluster creation and manag
Cloudmesh Client	An client application used to ssh in various
Ansible	Automation language to deploy application

Fig. 1. Technologies used in the Project

2. SOFTWARE STACK

3. WHY KUBERNETES

<write why we choose Kubernetes>

4. WHAT IS KUBERNETES

<what is Kubernetes>

5. ARCHITECTURE

<Architecture of Kubernetes>

6. ANSIBLE

No one likes repetitive tasks, so with Ansible [5], IT admins can begin automating away the drudgery from their daily routine tasks. Ansible is a simple automation language that can perfectly describe an IT application infrastructure. Ansible is an open source automation engine which can be used to automate cloud provisioning, configuration management, and application deployment. It can also perform more advanced IT tasks such as continuous deployment or rolling out updates with zero downtime.

A major difference in Ansible and many other tools in the space is its architecture.

6.1. Architecture

Ansible is an agentless tool,it doesn't requires any software to be installed on the remote machines to make them manageable. By default is manages remote machines over SSH or WinRM, which are natively present on those platforms [**?**].

Like the other configuration management software, Ansible distinguishes between two types of servers: one being the controlling machines and other being the nodes. Ansible uses a single controlling machine where the orchestration begins. Nodes are then controlled by a controlling machine over SSH [**?**]. The location of the nodes are described by the inventory of the controlling machine.

Ansible modules are deployed by Ansible over SSH. These modules are temporarily stored in the nodes and communicate with the controlling machine through a JSON protocol over the standard output

6.2. Playbooks

Playbooks [6] are Ansible's configuration, deployment, and orchestration language. They let us control the remote systems with a policy which we might want them to enforce. If Ansible modules act as tools in your workshop, then playbooks are your instruction manuals, and your inventory of hosts are your raw material. Playbooks can be used to manage configurations of and deployments to remote machines. They can sequence multi-tier rollouts involving rolling updates, and can delegate actions to other hosts, interacting with monitoring servers and load balancers.

6.3. Ansible Galaxy

6.4. Pandas

Pandas is an opensource Python library that provides data analysis functionality with Python. Python initially lacked data analysis and modeling capability. Pandas filled out this gap by providing essential analytic functions thus saving the need to switch to a more domain specific language for data analysis.

6.5. R

R is a language and environment for statistical computing and graphics [7]. Pandas does not provide a significant statistical modeling environment as it is still a work in progress. R provides a variety of statistical model analysis, classification, clustering and graphical techniques to provide this environment. Integrating Python's efficiency with R's capability allows us to build a highly a desirable analysis model for our application.

6.6. Docker

Docker allows application developers to package their applications into isolated containers. A container comprises of only the libraries and settings that are required to make the software work. Docker automates the repetitive tasks of setting up and configuring development environments thus allowing developers to focus only on building software. A dockerized application can simply ship between platforms as the complexity of software depencies is handled by the container.

6.7. Kubernetes

Kubernetes is an open-source platform which helps in automating deployment, scaling, and operations of application containers across clusters of hosts. Kubernetes helps in faster deployment of application and scaling them on the fly. Moreover it optimizes the use of hardware by using the resources which are needed. A Kubernetes cluster can be deployed on either physical or virtual machines. We will be using Minikube which is a lightweight Kubernetes implementation which creates a VM on the local machine and deploys a simple cluster containing only one node. The Minikube CLI provides basic bootstrapping operations for working with the cluster, including start, stop, status, and delete commands.

7. DESIGN

7.1. Building the Classification model

7.1.1. CrossValidation for the training data

To address the problem of incoming spam messages, a model shall be developed using the Bayesian Classification technique to correctly classify each incoming email/text message as a spam or a legitimate one. The model aims at developing a message filter that shall correctly classify messages based on word probabilities that are extracted from the training dataset.The training dataset to build the model consists of 5574 message records. Dataset taken from [8]. The training process shall use the cross-validation feature provided by R to build the classification model and use Bayes theorem of conditional probability to predict the class of each incoming message. < I just copy pasted the above paragrah here> Donno what will come here.

To develop an efficient training model, we shall partition the data into 2 subsets - training data and classification data. We shall choose one of the subsets for training and other for testing. In the next iteration the roles of the subsets shall be reversed, i.e the training data becomes the classification one and vice versa. This operation shall be carried out until each individual record is used both as a classification and training record. We shall use the cross validation feature provided by R for this subsampling. This subsampling technique handles the underfitting problem and guarantees an effective classification model.

7.1.2. Training process

Content of each of the spam marked messages shall be processed through Naive Bayes Classifier. The classifier shall maintain a bag of words along with the count of each word occuring in the spam messages. This word count shall be used to calculate and store the word probability in a table that shall be cross-referenced to determine the class of the record on classification data [9].

A selected few words have more probability of occuring in a spam messages than in the legitimate ones. Eg: The word "Lottery" shall be encountered more often in a spam message. The classifier shall correlate the bag of words with spam and non-spam messages and then use Bayes Theorem to calculate a probability score that shall indicate whether a message is a spam or not. The results shall be verified with the results available on the training dataset and the classifier accuracy shall be calculated.

The classifier shall use the Bayesian theorem over the training dataset to calculate probabilities of such words that occur more often in spam messages and later use a summation of scores of the occurence of these word probabilities to estimate whether a message shall be classified as spam or not. After working on several samples of the training dataset, the classifier shall have learned a high probability for spam based words whereas, words in legitimate message like family member or friends names shall have a very low probability of occurence.

7.2. Classifying new data

Once the training process has been completed, the posterior probability for all the words in the new input email is computed using Bayes theorem. A threshold value shall be defined to classify a message into either class. A message's spam probability is computed over all words in its body and if the sum total of the probabilities exceeds the predefined threshold, the filter shall mark the message as a spam [10].

A higher filtering accuracy shall be achieved through filtering by looking at the message header i.e the sender's number/name. Thereby if a message from a particular sender is repeatedly marked as spam by the user, the classifier need not evaluate the message body if it is from the same sender.

8. DISCUSSION

TBD

9. DEPLOYMENT

Our application will be deployed using Ansible [5] playbook. Automated deployment should happen on two or more nodes clouds or on multiple clusters of a single cloud. Deployment script should install all necessary software along with the project code to Kubernetes cluster nodes using the Docker image.

10. CONCLUSION

TBD

11. ACKNOWLEDGEMENT

We acknowledge our professor Gregor von Laszewski and all associate instructors for helping us and guiding us throughout this project.

12. APPENDICES

TBD

REFERENCES

[1] "What is internet?" Web Page, accessed: 2017-04-12. [Online]. Available: http://searchwindevelopment.techtarget.com/definition/Internet

[2] "Wikipedia," Web Page, accessed: 2017-04-12. [Online]. Available: https://www.wikipedia.org/

[3] "What is docker?" Web Page, accessed: 2017-04-02. [Online]. Available: https://www.docker.com/what-docker

[4] "Kubernetes," Web Page, accessed: 2017-03-10. [Online]. Available: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[5] "Ansible, deploy apps. manage systems. crush complexity," Web Page, accessed: 2017-04-12. [Online]. Available: https://www.ansible.com/it-automation

[6] "Playbook," Web Page, accessed: 2017-04-12. [Online]. Available: http://docs.ansible.com/ansible/playbooks.html

[7] "R:what is R?" Web Page, accessed: 2017-04-02. [Online]. Available: https://www.r-project.org/about.html

[8] "SMS spam collection dataset," Web Page, accessed: 2017-03-10. [Online]. Available: https://www.kaggle.com/uciml/sms-spam-collection-dataset

[9] J. Provost, "Naive-Bayes vs. Rule-Learning in Classification of Email," in *Artificial Intelligence Lab*. The University Of Texas at Austin: The University Of Texas, 1999, accessed: 2017-03-10. [Online]. Available: http://mathcs.wilkes.edu/~kapolka/cs340/provost-ai-tr-99-281.pdf

[10] Wikipedia, "Naive bayes spam filtering," Web Page, January 2017, accessed: 2017-03-10. [Online]. Available: https://en.wikipedia.org/wiki/Naive_Bayes_spam_filtering

Big data Visualization with Apache Zeppelin

NAVEENKUMAR RAMARAJU[1,*] AND VEERA MARNI[1,*]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*] Corresponding authors: naveenkumar2703@gmail.com, narayana1043@gmail.com

project-008, May 4, 2017

Apache Zeppelin is an open source notebook for data analytics and visualization. In this project Apache Zeppelin is used to deploy and do visual data analytics by taking advantage of parallel computing capabilities of Spark in multiple cloud environments.

Keywords: Zeppelin, Apache, Big data, Visualization

https://github.com/cloudmesh/sp17-i524/blob/master/project/S17-IR-P008/report/report.pdf

1. INTRODUCTION

Interactive browser-based notebooks enable data engineers, data analysts and Data scientists to be more productive by developing, organizing, executing, and sharing data code and visualizing results without referring to the command line or needing the cluster details. Notebooks allow these users not only allow to execute but in order to interactively work with long work-flows There are a number of notebooks available with Spark. Although IPython remains a mature choice and great example of a data science notebooks, it has certain limitations when used for visualizations with spark which are fulfilled by Apache Zeppelin.

Apache Zeppelin[1] is a upcoming web-based notebook which brings data exploration, visualization, sharing and collaboration features to Spark. It supports Python and also has a growing list of programming languages such as Scala, Hive, SparkSQL, shell and markdown.

It is a completely open web-based notebook that enables interactive data analytics used for data ingestion, discovery, analytics, visualization and collaboration. It has built in Spark integration and supports multiple language backends like Python, Hadoop HDFS, R etc. Multiple languages can be used within same Zeppelin script and share data between them. In this project we deployed Zeppelin along with in built Spark and backend languages R and Python across cluster using Ansible. Then installed additional visualization packages provided by Apache Zeppelin Helium[2] APIs.

We also loaded a data set into Spark across cluster and perform data analytics and visualization in cloud using Zeppelin. However since the goal of this class project is focus on deployment of Big data software across multiple machines and benchmarking the time for deployments, we focused more on that through out the paper and gave less importance to the analytics that are performed after the deployment.

2. INFRASTRUCTURE

The deployment of Apache Zeppelin is done on two clouds. The clouds selected for the purpose of this project are

1. Chameleon Cloud
2. JetStream Cloud

2.1. OpenStack

OpenStack[3] is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a data center, all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface. It was created as joint project between NASA and Rackspace that is currently managed by OpenStack Foundation. It is open source software released under the Apache 2.0 license.

Both Chameleon cloud and JetStream use OpenStack. OpenStack is a free, open source cloud computing platform primarily deployed as IaaS.[4]

2.2. Chameleon Cloud

Chameleon Cloud[5] provides a large-scale platform to the open research community allowing them to explore trans-formative concepts in deeply programmable cloud services, design and core technologies. It is funded by the National Science Foundation. The testbed of Chameleon Cloud is hosted at the University of Chicago and Texas Advanced Computing Center and the University of Chicago. Chameleon provides resources to facilitate research and development in areas such as Infrastructure as a Service, Platform as a Service, and Software as a Service. Chameleon provides both an OpenStack Cloud and Bare Metal High-level Performance Computing Resources[6].

2.3. JetStream Cloud

Jetstream is led by the Indiana University Pervasive Technology Institute (PTI), will add cloud-based computation to the national cyberinfrastructure. Researchers will be able to create virtual machines on the remote resource that look and feel like their lab workstation or home machine, but are able to harness thousands of times the computing power. Jetstream will provide the following core capabilities use Virtual Machines interactively, Researchers and students can move data to and from Jetstream using Globus transfer[7], use virtual desktops and publish VMs with a Digital Object Identifier(DOI)[8].

2.4. Virtual Machine Specifications

A brief comparison of multiple attributes of the clouds discussed above are shown in the table 1.

Clouds	*Chameleon*	*Jetstream*
CPU	Intel Xeon X5550	Dual Intel E-2680v3 "Haswell"
RAM	4 GB	2 GB
Number of CPU's	1	1
CPU Cores	1	1
CPU Speed	2.3 GHz	2.3 GHz

Table 1. Comparison of cloud vendors

3. APACHE ZEPPELIN

Apache Zeppelin is Apache project under open-source license Apache 2.0. It aims to provide a web interface to analyze and format large volumes of data processed via spark in a visual and interactive way. It is a notebook style interpreter that enables collaborative analysis sessions sharing between users. Zeppelin is independent of the execution framework itself. Current version run on top of Apache Spark but it has pluggable interpreter APIs to support other data processing systems. More execution frameworks of type SQL-like backends such as Hive, Tajo, MRQL can also be added.

3.1. Background

Large scale data analysis workflow includes multiple steps like data acquisition, pre-processing, visualization, etc and may include inter-operation of multiple different tools and technologies. With the widespread of the open source general-purpose data processing systems like Spark there is a lack of open source, modern user-friendly tools that combine strengths of interpreted language for data analysis with new in-browser visualization libraries and collaborative capabilities.

Zeppelin initially started as a GUI tool for diverse set of SQL-over-Hadoop systems like Hive, Presto, Shark, etc. It was open source since its inception in Sep 2013. Later, it became clear that there was a need for a greater web-based tool for data scientists to collaborate on data exploration over the large-scale projects, not limited to SQL. So Zeppelin integrated full support of Apache Spark while adding a collaborative environment with the ability to run and share interpreter sessions in-browser.

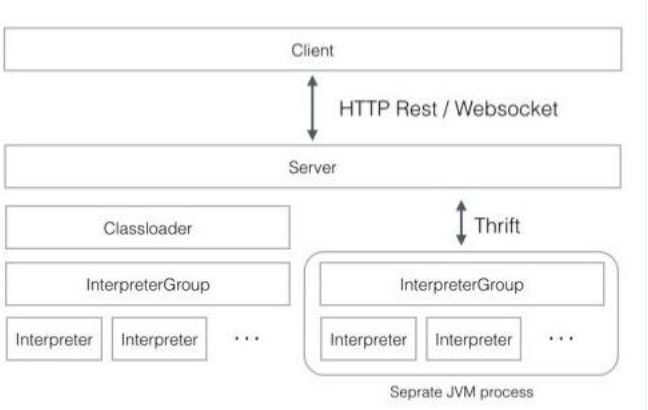


Fig. 1. Zeppelin Architecture

3.2. Zeppelin Features

Currently, Apache Zeppelin multipurpose notebook supports the following functionalities.

1. Data Ingestion
2. Data Discovery
3. Data Analytics
4. Data Visualization
5. Collaboration

3.3. Zeppelin Architecture

3.3.1. Client

Apache Zeppelin is a client based application. Analytics can be done with latest version of modern browsers. Anyone with details of the host details, port on which zeppelin is listening and access to the zeppelin interpreter can execute or view the notebooks.

3.3.2. Server

Apache Zeppelin is a web-server based application. Zeppelin listens on a port and application communicates to server through this port. This server-client based architecture facilitates sharing of notebooks and collaboration.

3.3.3. Classloader

Classloader loads the essential classes and configuration for running of Zeppelin. This also loads spark context as base interpreter. This loads additional interpreter and their classes when the new interpreters are configured and saved.

3.3.4. Multiple Language Backend

Apache Zeppelin interpreter concept allows any language/data-processing-backend to be plugged into Zeppelin. Currently Apache Zeppelin supports many interpreters listed below

1. Apache Spark
2. Python
3. JDBC
4. Markdown
5. Shell

Adding a new language backend is simple and shown in the next sections

3.3.5. Apache Zeppelin Interpreter

Apache Zeppelin interpreter is a language backend. For example to use python code in zeppelin, it is needed to have a python interpreter. Every interpreter belongs to an InterpreterGroup. Interpreters in the same InterpreterGroup can reference each other. For example SparkSqlInterpreter can reference SparkInterpreter to get the SparkContext from it while they're in the same group.

InterpreterSetting is configuration of a given InterpreterGroup and a unit of start/stop interpreter. All interpreters in the same InterpreterSetting are launched in a single, separate JVM process. The interpreter communicates with Zeppelin engine via Thrift.

3.3.6. Create your own Interpreter

To create a new interpreter we need extend org.apache.zeppelin.interpreter class and implement some methods. We can also include org.apache.zeppelin:zeppelin-interpreter:[version] artifact in our build system and put the jars under the interpreter directory with a specific directory name. Zeppelin server reads interpreter directories recursively and initializes interpreters including the new interpreter that is recently added.

There are three locations where you can store your interpreter group, name and other information. Zeppelin server tries to find the location below. Next, Zeppelin tries to find interpreter-setting.json in your interpreter jar.

zeppelin_interpreter_dir/your_own_interpreter_dir/
interpreter-settings

4. DEPLOYMENT

Zeppelin works with Spark deployed across clusters. To deploy Zeppelin and Spark across clusters, Ansible and cloudmesh client integrated with python CMD is used. Refer our code for implementation. Each component and their brief usage is explained in this section.

4.1. Cloudmesh Client

Cloudmesh client is a command line based tool to access and manage multiple cloud environments. Cloudmesh client can also be used to define security group and monitor cloud environments. Cloudmesh client is used in the project to boot, delete and define security group to enable firewall settings.

4.2. Ansible

Ansible is an automation tool to automate application deployment, maintenance and configuration management. Ansible is used to deploy jdk, openssh, spark and zeppelin across instances that are boot using cloudmesh client.

Ansible playbook is used to write and execute automated deployment. To deploy zeppelin and spark, playbooks are created with different roles like zeppelin, spark, ssh, jdk, start and stop tasks for each cloud with different variable files. Each variable file have cloud specific environment variables like cloud user, home folder and permissions for directory etc. Along with the variable files and separate role files, an individual playbook that calls start and stop tasks for zeppelin and spark. This is required to launch individual roles like start and stop to be executed through command line.

The playbook developed for deploying spark and zeppelin is configured to install pre-built version of spark and zeppelin instead of building from the code. Ansible downloads the prebuilt version and extract it. Then the extracted version is configured for cluster by setting the IP values of master and slaves of the cluster.

Version of Zeppelin and spark is configured in a variables file to make it easy to customize any version of spark or zeppelin. Ports on which zeppelin and spark listens could be configured in the same along with the locations in which the spark and zeppelin need to be installed.

4.3. Security - Cross SSH

In order for spark master and slaves to communicate, zeppelin to communicate with spark master cross SSH need to be enabled between all nodes in the cluster. This is achieved by creating a SSH public and private keys. The private key is encrypted using 'Ansible-valut' and stored in code repository. Then at deployment time same private and public key are distributed across cluster using ansible with decryption.

4.4. Putting to gether with CMD

4.5. Accessing applications

Python CMD is used to build a command line like interface to put cloudmesh client and ansible together. This interfaces with cloudmesh client to boot cloud instances with required security group and deploy applications using ansible.

Python CMD interface takes number of instances required to boot and launches the number of instances by using cloudmesh client. Once the instances are booted, the details of the instances are stored into an inventory file and config files. Now using the generated inventory and config file ansible playbook is launched. The interface has options to set cloud and user details.

This interface also has capability to deal with local network ip and floating IP based deployment. This enables in-network deployment by using less scarce resources without creating floating IP address. To do this we could set master ip through interface and cloud based spark-zeppelin infrastructure with master-slave using only one floating IP.

The interface also can be used to start and stop zeppelin and/or spark. It calls respective ansible playbooks to execute the task by using the inventory files created earlier.

Zeppelin Notebook is accessible via latest version of browsers like Firefox or Chrome. Zeppelin is configured to listen on port 8080 and Spark UI is available on port 8082 of master instance. By launching the master ip and port 8080, Zeppelin is launched and Spark master instance can be configured in Zeppelin. Now Zeppelin is ready to do visual analytics by taking advantage of Spark parallel computing capability.

Spark applications can also be launched using the command line interface created. This invokes start-all utility of spark to launch master and worker nodes. It is useful in the instances where spark need to run as application instead of using Zeppelin for only analytics.

4.6. Boot and Deployment Time

The interface prints time taken to boot the instances and time taken to deploy the application across all the nodes. The time printed are used for benchmarking the deployment. Booting happens in sequence as Chameleon cloud supports sequential booting operations only. Once booting is completed the deployment is done in parallel across different machines in cluster. Refer to the benchmarking section for the time took to deployment time on different clouds.

5. DATASET DESCRIPTION

This is real-world dataset[9] collected from a Portuguese marketing campaign related with bank deposit subscription. The business goal is to find a model that can explain success of a contact, i.e. if the client subscribes the deposit. Such model can increase campaign efficiency by identifying the main characteristics that affect success, helping in a better management of the available resources (e.g. human effort, phone calls, time) and selection of a high quality and affordable set of potential buying customers.

The increasingly vast number of marketing campaigns over time has reduced its effect on the general public. Furthermore, economical pressures and competition has led marketing managers to invest on directed campaigns with a strict and rigorous selection of contacts. Such direct campaigns can be enhanced through the use of Business Intelligence (BI) and Data Mining (DM) techniques.

In the benchmarking section we have used three codes to benchmark the Zeppelin software that we have deployed on the Chameleon and Jetstream clouds. All the codes run on the same data set and written in SQL. The code is explained in more detail in the visualization section below

6. BENCHMARKING

There are 2 different approaches used in benchmarking which are used for the deployments on clouds where the deployment has been done. They are as follows

1. Deployment Benchmarking
2. Analytical Benchmarking

Deployment Benchmarking
This benchmarking deals with the time taken for deploying Apache Zeppelin across machines. Graphs are plotted to visualize the time taken for deployment of Apache Zeppelin with number of machines on x-axis and time taken on the y-axis. The command line script also includes code to record the time taken for the deployment. When the VM's are booted inside the command line wrapper the results also include the amount of time taken to deploy Apache Zeppelin on the virtual machines. The time taken for deploying Apache Zeppelin on different number of machines can be recorded and plotted on a graph to show analyze the increase in amount of time as the number of machines increases. Ideally it is excepted that the graph in the curve flattens out as with increase in the number of machines.

Various factors that influence the deployment benchmarking are as follows.

1. The dependencies that need to be installed on all machines in order to deploy the software.
2. The network traffic can effect the time taken for deployment. For example a bad network might introduce delay in downloading the software on to the machines.
3. The number of machines the software has to be deployed on.

Analytical Benchmarking
This benchmarking deals with the time taking for running the analytics on clouds. The same analytics are performed on all clouds on which Apache Zeppelin was deployed and the performance is plotted on graphs

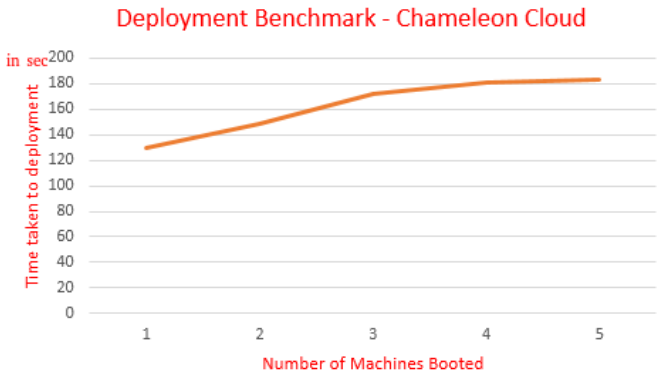


Fig. 2. Jetstream Deployment Benchmarking

Various factors that influence the analytical benchmarking are as follows.

1. The size of the data set that the scientist is working on. As the size of the data set it take more time to download the data set and split it across machines.
2. The way the machines are configured. If all the machines lie on the same hardware then the network over head is largely reduced decreasing delays in processing.
3. The complexity of the algorithm. A highly complex algorithm can take longer time time than a simpler algorithm.
4. The size of data set can also effect the running as the algorithm time complexity will increase with the size of the dataset.

Since Cloudmesh client doesn't allow parallel boot of virtual machines the boot time is neglected in the deployment benchmarking

6.1. Chameleon Cloud

The Benchmarking for on Chameleon Cloud is done and explained in detail the below 2 sections. The benchmarking is only performed after all the machines are successfully booted and ready for deployment.

6.1.1. Deployment Benchmarking

Once all the machines are boot the ansible-playbook script is started automatically and the time taken to deploy Apache Zeppelin on the machines is clocked before the start of the deployment and after the end of the deployment. The difference of the end time and start time is the total deployment time. The graph for deployment benchmarking on chameleon cloud explains the various times taken to deploy Apache Zeppelin across machines with changes in the number of machines on Chameleon Cloud.

The time taken for deployment on a single machine is the lowest of all and the time taken for deploying more machines increases with the number of machines. However the graph also starts to flatten out after five machines. Since the deployment is done using ansible playbook the process is parallelized and all the softwares are installed at the same time across all the machines. This process is reflected in the deployment graph shown for chameleon cloud.

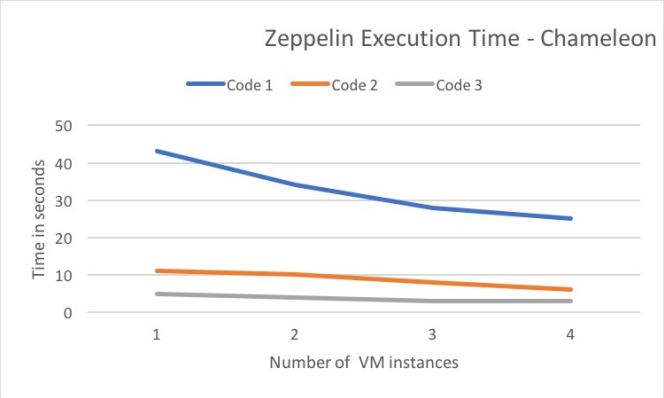


Fig. 3. Chameleon Analytics Benchmarking

6.1.2. Analytical Benchmarking

After the deployment of Apache Zeppelin on Chameleon Cloud, the code for the analytics is run on the Apache Zeppelin and the run time is clocked. The run time to process and generate the visualizations is plotted on the y-axis and the number of VMs in the cluster is given on the x-axis. The table below explains this in detail.

Table 2. Analytical Benchmarking Chameleon Cloud
Time taken to run codes Vs Machines Count

VM Count	Code#1	Code#2	Code#3
1	43	11	5
2	34	10	4
3	28	8	3
4	25	6	3

6.2. Jetstream Cloud

Similar to Chameleon Cloud, in Jetstream also cloudmesh allows only serial booting of VMs. Hence the boot time of the VMs is ignored in the process of benchmarking the deployments on the Jetstream Cloud.

6.2.1. Deployment Benchmarking

The benchmarking in the Jetstream case is similar to that of the deployment in the Chameleon cloud. The same ansible-playbook script is started automatically and the time taken for deployments are recorded similarly. The below graph explains the amount time taken to deploy Apache Zeppelin on Jetsream cloud when the number of machines are varied.

From the Jetstream deployment benchmarking figure it can be seen that the time taken for deploying zeppelin across virtual machines stops to grow and flattens out as the number of virtual machines start to increase. It can also be noted that there is an increase in the number time taken for deployment the number of virtual machines is less than 4. The primary reason for this initial increase due the additional overhead the master node has to handle for setting up communication with the worker nodes

6.2.2. Analytical Benchmarking

Similar to the analytical benchmarking in the chameleon we also performed the same experiment on Jetstream cloud. The results

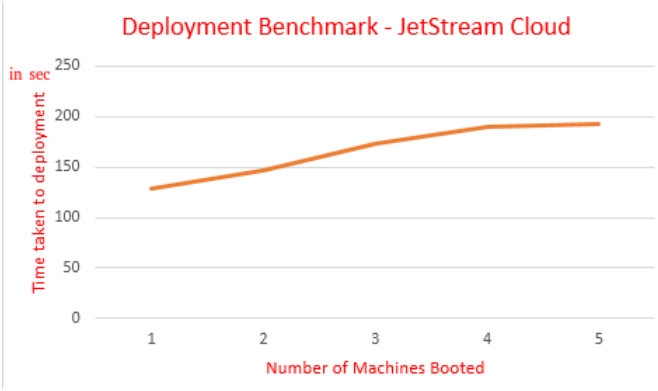


Fig. 4. Jetstream Deployment Benchmarking

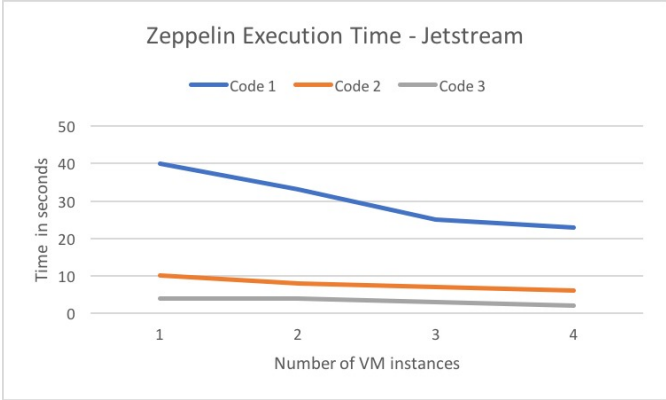


Fig. 5. Jetstream Analytics Benchmarking

are presented in the table below.

Table 3. Analytical Benchmarking Jetstream Cloud
Time taken to run codes Vs Machines Count

VM Count	Code#1	Code#2	Code#3
1	40	10	4
2	33	8	4
3	25	7	3
4	23	6	2

Analytics deployment shows a slight decrease in time as number of nodes increased.

7. VISUALIZATION WITH ZEPPELIN

All the codes for visualization are written in SQL on Zeppelin. Zeppelin has options to change the type of plots in a click and better present the results. The basic features of zeppelin are presented below through the code examples.

7.0.1. Code 1

This piece of code counts all the people who are below the age of 30, groups them age and then orders the counts by age. The plot below shows this in detail.

```
select age, count(1) value
```

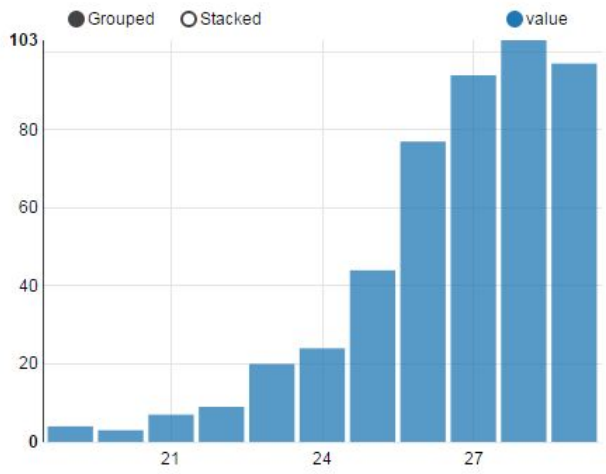


Fig. 6. Histogram / Bar Chart

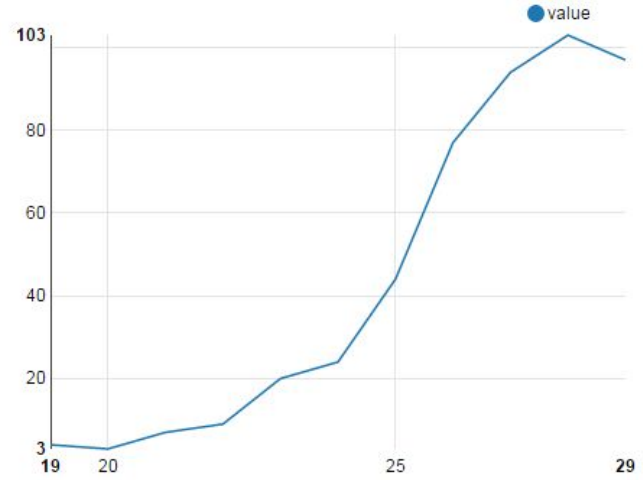


Fig. 8. Line Chart

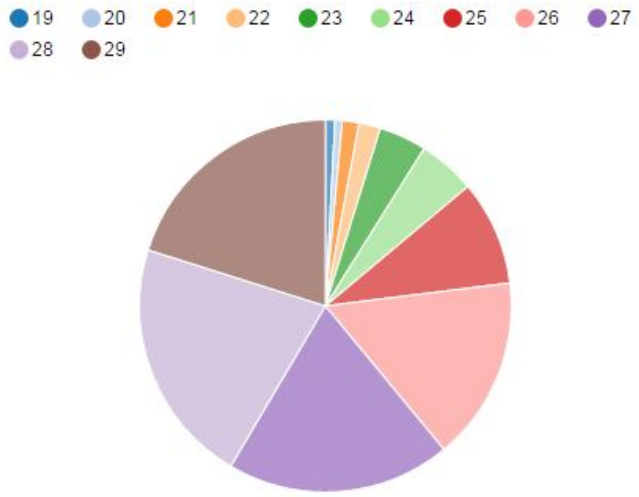


Fig. 7. Pie Chart

```
from bank
where age < 30
group by age
order by age
```

A few of the different Types of Visualizations on the same code can be seen the figures below and many others can be explored at on Zeppelin basic tutorials available inbuilt in the Zeppelin notebook. The figures 4, 5 and 6 show how Zeppelin plots Bar Chart, Pie Chart and Line Charts on the Bank data described above.

7.0.2. Code 2

```
select age, count(1) value
from bank
where age < ${maxAge=30}
group by age
order by age
```

In this code above the maxAge parameter acts as an place holder and excepts an user input which is an integer. After the system process the user input then the place holder is replaced by the input values and Zeppelin executes the code and presents the results.

7.0.3. Code 3

The user input can also be a string and it shown in the code below. Below code takes a string as input and processes the query based on the input received

In the below query the records are picked based on the marital status column, grouped by the age column and then ordered by age to show the count of people on who and their age given their marital status.

```
select age, count(1) value
from bank
where marital="${marital=single,single|divorced|married}"
group by age
order by age
```

8. SUPPLEMENTAL MATERIAL

Apache Zeppelin is Quick start tutorials are available on the Apache Zeppelin webpage[10] and can be accessed for free of cost. There are also other Zeppelin works available in the form of notebooks on Zeppelin hub[11].

9. CONCLUSION

We are successfully able to deploy Apache Zeppelin across clouds with varying number of machines. The deployment time flattens out after four clusters in both Chameleon and Jetstream clouds. The time taken to run similar zeppelin analytic queries on both clouds are almost similar when the other parameters are fixed.

10. EXECUTION PLAN

The following subsections act as a timeline regarding how we broke the project up week-by-week in order to complete the entire project by the desired deadline. This project execution plan is a final draft of the project was implemented during the second half of the semester.

10.1. March 6,2017 - March 12,201

This week we discussed about the planned how to implement the project in and came up with approximate deadlines for tasks. We have also revisted the tutorials on the class webpage and referred to official documentation of Apache Zeppelin and came up with a workflow for implementing this project.

10.2. March 13,2017 - March 18,201

This week we have installed Cloudmesh on our local machines, completed the tutorials on Cloudmesh present on the class website. We have also accessed on chameleon cloud accounts to boot Vitrual Machines on cloud and logged in successfully into the Virtual Machines.

We have discussed about building a command shell through which we can deploy the clusters with less effort. Hence we looked completed the tutorials on CMD and CMD5 available on the class website. This tutorials have helped us in coming up with a basic outline of the shell that we should develop in order to meet the requirements for deploying Apache Zeppelin on various clouds. We have made a decision to use CMD module in python for this purpose.

10.3. March 19,2017 - March 26,201

During this week we have completed the development of the command shell which can start a given number of virtual machines are return their details like the machine name, floating IPs, Static IPs to a file. Other methods like delete, setCloud, getStaticIps, getFloatingIps are also included in the command shell developed over the week. The description for all methods is documented and can accessed from with the shell.

We have discussed the over the deployment of Apache Zeppelin and came up with the dependencies that need to installed on the machines before zeppelin in deployed on to them. We have revisted the ansible tutorials on the class website as we will using ansible to deploy Apache Zeppelin on various clouds

10.4. March 27,2017 - April 2,201

Developed and tested code to deploy the Apache Zeppelin on the clusters. Upon successful deployment we have opened ports so that Apache Zeppelin can be accessed through web-interfaces.

10.5. April 10,2017 - April 16,201

Integrated the deployment code into the command shell developed previously and tested the deployment chameleon cloud. We have ran into issues with security and VM accessibility. We have fixed the below issues over the week.

1. Fixed deployment issues that might arise to lack of availability of floating point IPs on the chameleon cloud.

2. Fixed security issues and checked if the notebook is accessible through the external web-browsers.

During the week we have also worked on analytics which can be performed on the Apache Zeppelin that has been previously installed on the cloud from a web-page on in external machine. More details about the analytics are discussed in the analytics section below.

10.6. April 17,2017 - April 23,201

Review of deployment and developing the final draft of the report for submission.

ACKNOWLEDGEMENTS

This work was done as part of the course "I524: Big Data and Open Source Software Projects" at Indiana University during Spring 2017. Thanks to our Professor Gregor von Laszewski and associate instructors for their help and support during the course.

AUTHOR BIOGRAPHIES

Naveenkumar Ramaraju is a graduate student in Data Science at the School of Informatics and Computing Indiana University. He is interested in Machine learning, data science and big data.

Veera Marni received his Bachelors in Technology in Electronics and Communication from SRM University, India and will be receiving his Masters in Data Science from Indiana University in Dec 2017. His research interests are Machine Learning and Big Data. He will be working as Data Scientist intern at Proteous Digital Health during the summer 2017.

11. WORK BREAK DOWN:

Naveenkumar Ramaraju is responsible for the following

- Ansible scripts for deployment
- Benchmarking on Jetstream

Veera Marni is responsible for the following

- Python wrapper using CMD
- Benchmarking on Chameleon
- Zeppelin notebook

REFERENCES

[1] Apache Zeppelin, “Zeppelin 0.7.0 Documentation,” Web Page, Apache Software Foundation, Mar. 2017. [Online]. Available: https://zeppelin.apache.org/docs/0.7.0/

[2] “Helium,” Web Page, Apache Software Foundation, Apr. 2017. [Online]. Available: https://cwiki.apache.org/confluence/display/ZEPPELIN/Helium+proposal

[3] Open Stack, “Open Stack,” Web Page, Apache Software Foundation, Apr. 2017. [Online]. Available: https://www.openstack.org/software/

[4] LaaS, “LaaS,” Web Page, Apache Software Foundation, Apr. 2017. [Online]. Available: https://en.wikipedia.org/wiki/Logging_as_a_service

[5] Chameleon Cloud, “Chameleon Cloud,” Web Page, Apache Software Foundation, Apr. 2017. [Online]. Available: https://www.chameleoncloud.org/about/chameleon/

[6] “Bare metal high level performance computing resources,” Web Page, Apache Software Foundation, Apr. 2017. [Online]. Available: https://cloudtweaks.com/2013/08/bare-metal-cloud-meeting-the-demand-for-high-performance-cloud-solutions/

[7] Globus Transfer, “Globus Transfer,” Web Page, Apache Software Foundation, Apr. 2017. [Online]. Available: https://www.chameleoncloud.org/about/chameleon/

[8] DOI, “DOI,” Web Page, Apache Software Foundation, Apr. 2017. [Online]. Available: https://en.wikipedia.org/wiki/Digital_object_identifier

[9] S. Moro, R. Laureano, and P. Cortez, “Using data mining for bank direct marketing: An application of the crisp-dm methodology,” in *Proceedings of the European Simulation and Modelling Conference - ESM'2011*, P. N. et al., Ed. Guimaraes, Portugal: EUROSIS, Oct. 2011, pp. 117–121.

[10] “Tutorial,” Web Page, Apache Software Foundation, Apr. 2017. [Online]. Available: https://zeppelin.apache.org/docs/0.5.5-incubating/tutorial/tutorial.html

[11] “Zeppelin hub,” Web Page, Apache Software Foundation, Apr. 2017. [Online]. Available: https://www.zeppelinhub.com/viewer

Cloudmesh Docker Extension

KARTHICK VENKATESAN[1] AND ASHOK VUPPADA[2]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.

S17-IR-P009, May 4, 2017

Cloudmesh client is a simple client to enable access to multiple cloud environments from a command shell and command line. The users can manage their set of resources right from their workstation.Currently, cloudmesh client supports managing Virtual Machines across multiple clouds. In this project, we have added the capability to manage/provision docker and swarm containers to the cloudmesh client through a simple and extensible command line interface.

Keywords: Cloud, I524

Report: https://github.com/cloudmesh/sp17-i524/blob/master/project/S17-IR-P009/report/report.pdf
Code: https://github.com/cloudmesh/cloudmesh.docker

1. INTRODUCTION

Docker is an open platform for developing, shipping, and running applications. Docker enables to separate the applications from infrastructure so that we can deliver software quickly. With Docker, one can manage the infrastructure in the same ways we manage our applications[1].

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow running many containers simultaneously on a given host. Containers are lightweight because they don't need the extra load of a hypervisor, but run directly within the host machine's kernel. This means we can run more containers on a given hardware combination than if you were using virtual machines. We can even run Docker containers on host machines that are virtual machines.

Clouldmesh Client[2] capability is detailed in Figure 1,it aims at managing vm instances in multiple heterogeneous clouds remotely via a command line interface.In this project we have added the capability to provision and manage docker[1] containers and swarm[3] services to cloudmesh client[2].

1.1. Docker Mode

The key objects of Docker engine are images, containers and networks.An image is a read-only template with instructions for creating a Docker container. A container is a runnable instance of an image. The Docker Module built in cloudmesh docker application has the capabilities to manage Docker hosts and the underlying objects running on multiple remote VM's as shown in Figure 2.

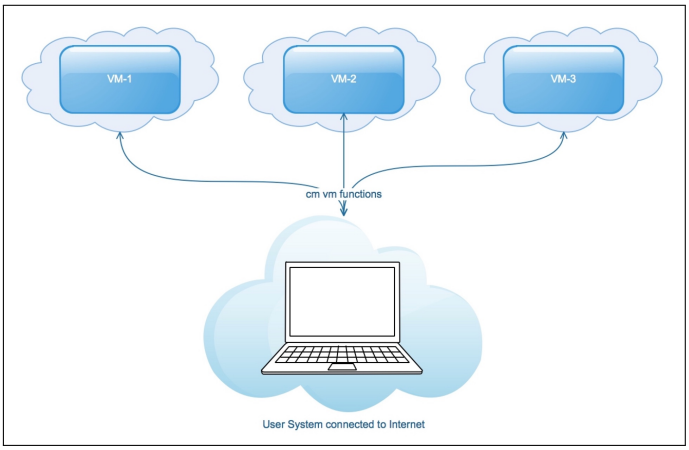


Fig. 1. Cloudmesh client

1.2. Swarm Mode

A swarm[3] is a cluster of Docker engines, or nodes, participating in a cluster where we deploy services. The Swarm Mode of Docker orchestrates swarm services in standalone containers on Docker instances.

1.2.1. Node

A node is an instance of the Docker Engine participating in the swarm.We can run one or more nodes on a single physical computer or cloud server.

To deploy the application to a swarm, we submit a service definition to a manager node. The manager node dispatches units of work called tasks to worker nodes.

Manager nodes also perform the orchestration and cluster

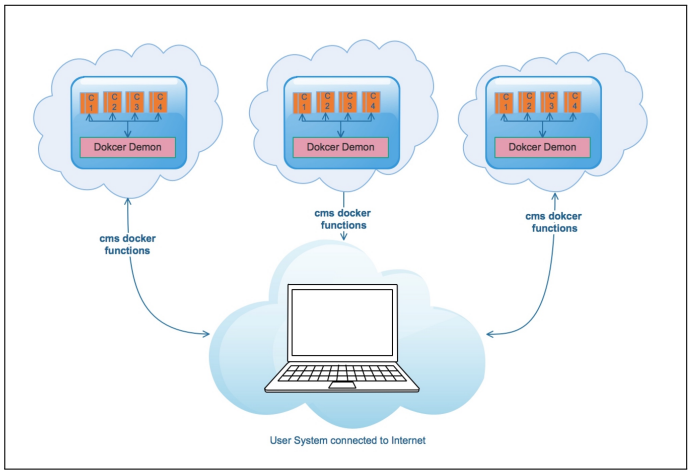


Fig. 2. Docker Module

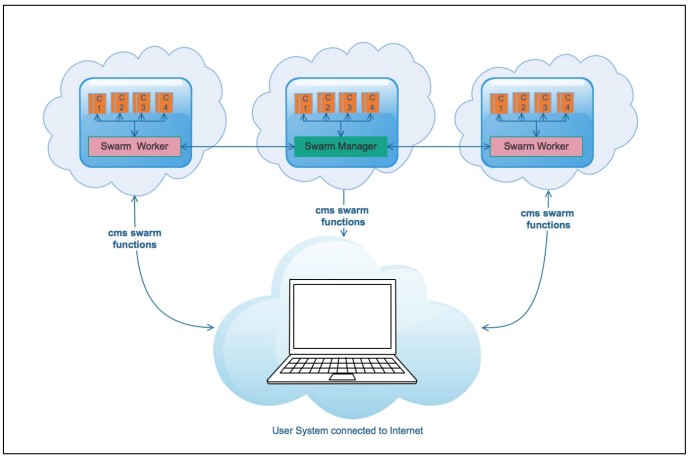


Fig. 3. Swarm Module

management functions required to maintain the desired state of the swarm. Manager nodes elect a single leader to conduct orchestration tasks.

Worker nodes receive and execute tasks dispatched from manager nodes. By default manager nodes also run services as worker nodes, but we can configure them to run manager tasks exclusively and be manager-only nodes. An agent runs on each worker node and reports on the tasks assigned to it. The worker node notifies the manager node of the current state of its assigned tasks so that the manager can maintain the desired state of each worker.

1.2.2. Services and Tasks

A service is the definition of the tasks to execute on the worker nodes. It is the central structure of the swarm system and the primary root of user interaction with the swarm.When we create a service, we specify which container image to use and which commands to execute inside running containers.

In the replicated services model, the swarm manager distributes a specific number of replica tasks among the nodes based upon the scale we set in the desired state.

For global services, the swarm runs one task for the service on every available node in the cluster.

A task carries a Docker container and the commands to run inside the container. It is the atomic scheduling unit of swarm. Manager nodes assign tasks to worker nodes according to the number of replicas set in the service scale. Once a task is assigned to a node, it cannot move to another node. It can only run on the assigned node or fail.

The Swarm Module built in cloudmesh.Docker application has the capabilities to create and manage a swarm cluster running of multiple remote VM's as shown in Figure 3.

1.3. Remote vs Local use

Users can choose to use cloudmesh docker application from a remote terminal outside the network of the data centre as in Figure 2 or locally from a provisioning or configuration server inside the data centre as in Figure 4.We have analysed this difference in the application usage in depth in the project and have provided detailed benchmark results for both modes of use.

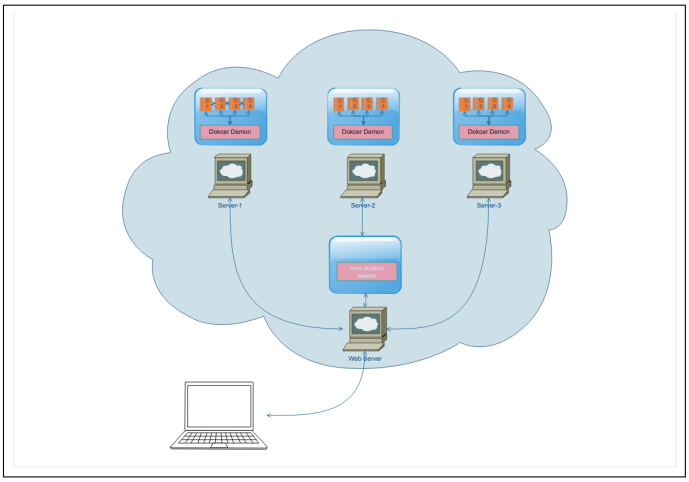


Fig. 4. Docker/Swarm Remote

2. CLOUDMESH DOCKER APPLICATION ARCHITECTURE

The architecture of the application is depicted in Figure 5.The commands developed can be broadly classified as 'action commands','Inquiry commands'. The action commands are which would create or alter an entity.The entity can be a host/container/node. We use the corresponding API call to get the latest values for the changed entity.Docker API module is developed for addressing the docker commands, and Swarm API module is used for swarm commands. The Inquiry commands have two flavours a list and refresh mode.The list commands fetch the data locally from the Database and the refresh command will refresh the current state of the corresponding entity from the hosts.

2.1. Technologies Used

Name	Purpose
docker [1]	Docker Server and Api for managing containers and services
mongodb [4]	Nosql DBMS
Python-eve [5]	Restful webservices interface to mongoDB
python [6]	Development
ansible [7]	Automated deployment

Table 1. Technology Name and Purpose

2.2. MongoDB and Python-eve

The cloudmesh docker application uses MongoDB[4] for data storage.The access to the database is all through restful services supported through Python-eve[5].The following are the entities for which collections are defined in Eve and MongoDB.

1. Host
2. Image
3. Container
4. Network
5. Service
6. Node

A key benefit of using a NoSQL database like MongoDB is that it allowed us to store the data in the DB in the native form as returned by the Docker API without the need for much marshalling of the data.

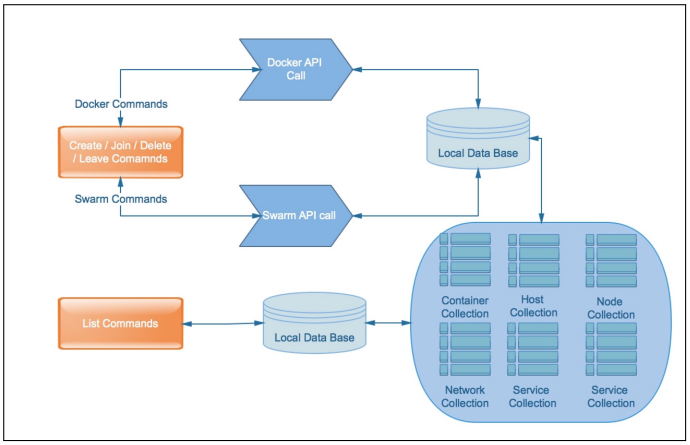


Fig. 5. cloudmesh docker application architecture

2.3. Security

We have not yet fully baked in security aspects into the client.We propose that security between the various aspects of the application will be handled as below

1. The docker daemon on the remote Hosts / VM are to be exposed to be accessed by cloudmesh Docker application.We envision that there would be separate configuration machines either within the network of the user's infrastructure or outside which will have a secure access.The docker daemons will be configured to accept connections only from this remote host.Alternatively, if there are multiple configuration machines, then specific security groups need to created between the client machine and each docker host for secure access.
2. The Python-eve client currently runs on a local machine and will accept connections only from the localhost from were the client is run.We also envision adding a user access control to the Python-eve services which will facilitate making the service and the Mongo DB to be deployed on a remote central server different from the client and allowing only authenticated users to use the application.

2.4. Cloudmesh Common

The application makes extensive use of common functions and tightly integrated into common functions available in the cloudmesh common repository for display formatting, YAML config management and timers

2.5. Ansible

As part of the project we have built Ansible[7] scripts to automate the installation of Docker in remote hosts and also deployment of Docker images in these remote docker hosts.Below is the list of Ansible scripts that are built and used in the project

1. Install Docker in remote hosts and enable them for remote API access
2. Install Docker images in Docker hosts.As part of the script, the local docker files are synced with remote hosts, and the images are built.
3. Setup /etc/hosts for remote hosts.This script allows to setup host names in remote hosts which allow the applications running on these hosts to be configured to access other applications on the network by the standard host names instead of the IP address.

2.6. Project Repository

The source code and the detailed deployment and use instructions are available at [8]

3. DOCKER COMMANDS

1. **Host Set/Add** This command will Set/Add the docker host on which the user wants to operate.The host details will be captured in the database with this command.

```
cms docker host docker1 docker1:4243
```

2. **Host List** This command will list the hosts available.The output would display the Ip, Name, Port,and if the host is swarm manager and the swarm manager Ip. Please note the Swarm manager Ip will be blank if the host is manager or not part of swarm.

```
cms docker host list
```

Table 2. cms docker host list

Ip	Name	port	Swarmmode	SwarmManagerIp
docker1	docker1	4243	Manager	
docker2	docker2	4243	Worker	docker1
docker3	docker1	4243	Host	

3. **Host Delete** This command will delete the host from the setup.This would also delete the host details from the database.User can do a host list to see the updated host details.

```
cms docker host delete docker1:4243
```

4. **Image List** This command will display the images available on the hosts available in the database. It would display the Ip of the host, the Image Id, repository and the size of the image.Please note that this command would display the results from the local dB.

```
cms docker image list
```

Table 3. cms docker image list

Ip	Id	Repository	Size(GB)
docker1	5545f4e3b27e	cloudmesh:docker	5.59
docker2	45f4e3b2799e	elasticsearch:swarm	0.45

5. **Image Refresh** This command would refresh the images across the hosts available. The results are updated to the local data base.

```
cms docker image refresh
```

6. **Container Create** This command will create a container on a given host.The arguments for this command is the name of the container and the image from which the container needs to be created.The image in the argument must be available through image list command above on the given host.

```
cms docker container create test1 \
elasticsearch:docker
```

7. **Container Start** This command will start a container.The container should have been already created using the create command.

```
cms docker container start test1
```

8. **Container Stop** This command will stop and container which is running.

```
cms docker container stop test1
```

9. **Container List** This command will display the list of containers running across the hosts.The output contains the Ip, Container Id, Name, Image, status and start time of the container.The details would be shown from the local database maintained.

```
cms docker container list
```

Table 4. cms docker container list

Ip	Id	Name	Image	Status	StartedAt
docker1	5545f4e3b27e	test1	image1	exited	12.00PM

10. **Container Refresh** This command will refresh the current state of the containers across the hosts. This application will connect to all the hosts available in the database and run the native Docker container list to get the latest information and update the local database for refreshing the data.

```
cms docker container refresh
```

11. **Container Delete** This command will delete the container on the host. The argument required is the container name.The updated container list can be viewed by running cms docker container list command.

```
cms docker container delete test1
```

12. **Container Run** This command will create and run a container in one step.The arguments for the function are the name of the container and the image from which it needs to be run.

```
cms docker container run test1 /
elasticsearch:docker
```

13. **Container Pause** This command will pause the container which is currently running.

```
cms docker container pause test1
```

14. **Container Unpause** This command will unpause the container which is currently paused.

```
cms docker container unpause test1
```

15. **Network Refresh** This command will refresh the network details across all the available hosts in the database and update the results to the local database.

```
cms docker network refresh
```

16. **Network List** This command will display the network details available in the local database.The output would display the host Ip where the network is established, the network Id , name and containers on the network.

```
cms docker network list
```

Table 5. cms docker network list

Ip	Id	Name	Containers
docker1	5545f4e3b27e	network1	test1

4. SWARM COMMANDS

1. **Host Set/Add** The command will add or set the input host to the current host.The swarm commands following this command will be executed on the host setup in this step. The host details will also be captured in the database as part of this command.

```
cms swarm host docker1 docker1:4243
```

2. **Host List** This command will list the hosts available. It displays the Ip, Name,Port,and if the host is swarm manager and the swarm manager Ip. Please note the Swarm manager Ip will be blank if the host is manager or not part of swarm.

```
cms swarm host list
```

Table 6. cms docker host list

Ip	Name	port	Swarmmode	SwarmManagerIp
docker1	docker1	4243	Manager	
docker2	docker2	4243	Worker	docker1
docker3	docker1	4243	Host	

3. **Host Delete** This command will delete the host from the database and also the associate container,service,networkand image objects of the host.

```
cms swarm host delete docker1:4243
```

4. **Image List** This command will display the images available on a host. It would display the Ip of the host, the Image Id , repository and the size of the image.Please note that this command would display the results from the local dB.

```
cms swarm Image list
```

5. **Swarm Create** This command will initialise the swarm mode on the current host.There are no arguments required for this command. After this command is run the current host would become the swarm manager.

```
cms swarm create
```

6. **Swarm Join** This command will join the current host to a swarm node either as a manager or a worker. User needs to setup a new current host with cms swarm host command and run cms swarm join so that current host would be joined with the swarm created.User needs to pass the address or host name of the swarm manager and the mode in which the current node will operate in the swarm.

```
cms swarm join docker3 docker4:4243 worker
```

(assuming docker3 is already a swarm manager)

7. **Swarm Leave** This command is applicable for the swarm manager or worker, this would remove the host from the swarm.If a manager has multiple workers, workers need to be removed(leave) before the manager can leave.

```
cms swarm leave
```

8. **Network Create** This command will create the network which can be used by the swarm containers later.The argument it needs is the name of the containers.

```
cms swarm network create network1
```

9. **Network List** This command will display the network details from the local database.The output will display the host Ip where the network is established, the network Id, name and containers on the network.

```
cms swarm network list
```

Table 7. cms swarm network list

Ip	Id	Name	Containers
docker1	5545f4e3b27e	network1	test1

10. **Network Refresh** This command will refresh the network details across the Docker hosts available in the database and update the results in the local database.

```
cms swarm network refresh
```

11. **Network Delete** This command will delete an existing network.The input required for this command is just the network name.

```
cms swarm network delete network1
```

12. **Service Create** This command will create a service, the arguments required are the image name and the name of service.This command will record the service details into the local database.This command also takes several configurable options such as Replica Count , Replica Mode and Network which can be passed as advanced options as detailed in Section 5.

```
cms swarm service create elasticsearch \
elasticsearch:swarm
```

13. **Service List** This command will list the current services running , the data displayed is from the local database, if the most current details are required user can run service refresh command below.

```
cms swarm service list
```

The number of replicas below indicates the number of containers which are running the services.

Table 8. cms swarm service list

Ip	Id	Name	Image	Replicas
docker1	5545f4e3b27e	elasticsearch	elastic:swarm	3

14. **Service Delete** This command will delete a running service, the argument required is the service name.This command will also delete the service details into the local database.

```
cms swarm service delete elasticsearch
```

15. **Service Refresh** This command will refresh the services status for all the hosts available in the database and refresh the service details in the local database.

```
cms swarm service refresh
```

16. **Node List** This command will display the list of the nodes across the hosts available.The results come from the local database. The command will display the node Id, node Ip, Role, Status and Manager Ip.

```
cms swarm node list
```

Table 9. cms swarm node list

Id	Ip	Role	Status	Manager Ip
5545f4e3b27e	docker3	Manager	Ready	
7645f4f4b27e	docker2	Worker	Ready	docker4

17. **Image Refresh** This command will refresh the images details across the hosts available in the database and store the results in the local database .

```
cms swarm image refresh
```

18. **Image List** This command will display the images available on a host. It would display the Ip of the host, the Image Id, repository and the size of the image.Please note that this command would display the results from the local dB.

```
cms swarm image list
```

Table 10. cms swarm image list

Ip	Id	Repository	Size(GB)
docker1	5545f4e3b27e	cloudmesh:docker	5.59
docker2	45f4e3b2799e	elasticsearch:swarm	0.45

19. **Container Refresh** This command will refresh the current state of the containers across the hosts available in the local database. This command would connect to the host and run the native Docker container list to get the latest information and update the local database refreshing the data.

```
cms swarm container refresh
```

20. **Container List** This command will display the list of containers running across the hosts.This would return the Ip, Container Id, Name, Image, status and start time of the container.The details would be shown from the local database maintained.

```
cms swarm container list
```

Table 11. container list

Ip	Id	Name	Image	Status	StartedAt
docker1	5545f4e3b27e	test1	image1	exited	12.00PM

5. DOCKER AND SWARM COMMANDS ADVANCED OPTIONS

All the commands in the docker and the swarm module support optional advanced parameters that allow customising the docker containers, services and networks that are created.A detailed list of these arguments is available at [9].The advanced options can be added to commands as simple name value pairs.

6. USE CASE - ELASTICSEARCH CLUSTER

Elasticsearch[10] is an open-source, broadly-distributable, readily-scalable, enterprise-grade search engine. Accessible through an extensive and elaborate API, Elasticsearch can power extremely fast searches that support your data discovery applications[2]

Using Cloudmesh client, Ansible and Cloudmesh Docker application we deployed and provisioned an Elasticsearch cluster on remote hosts in Chameleon cloud in Docker and swarm

mode.We benchmarked the cluster using esrally[11] have compared the results between the elastic search clusters in Docker and swarm mode.

The hardware specifications used on both the clouds is detailed in Table 12

Table 12. Deployment Hardware Specification

	Chameleon	Aws
VM	3	3
Containers	5	5
OS	Ubuntu 16.04	Ubuntu 16.04
Flavor	m1.large	t2.large
VCPU	4	2
Memory	8 GB	8 GB
Storage	80 GB	80 GB

6.1. Elasticsearch Cluster Docker Mode

For provisioning, the Elasticsearch cluster in Docker hosts below are the steps done

1. Created 3 Virtual Machines using Cloud Mesh Client .2 of the Virtual Machines are to be used for the docker Elasticsearch cluster, and 1 Virtual machine is the Benchmark server for the Kibana and esrally docker images

2. Using Ansible scripts Install Docker in 3 Virtual Machines and enable the docker daemon for remote access.

3. Using Ansible scripts Install Images of Elasticsearch on hosts for Docker cluster and the Image of Esrally in the Benchmark server.

4. Using the Cloudmesh Docker application we start four containers 2 in each of the virtual machines.To enable clustering of Elasticsearch applications running in the docker containers, we need set the below parameters in container creation.

```
network_mode=host
environment=
["http.host=0.0.0.0",
"transport.host=0.0.0.0",
"discovery.zen.ping.unicast.hosts=/
docker1,docker2"]
```

   The network mode set to host allows the Elasticsearch containers use the underlying Virtual Machines network for networking and leveraging the Elasticsearch unicast discovery find and form a cluster along with the other Elasticsearch instances running in other containers either on the same host or different hosts.

6.2. Elasticsearch Cluster Swarm Mode

For provisioning, the Elasticsearch cluster in Docker hosts in swarm mode below are the steps done

1. Created 3 Virtual Machines using Cloud Mesh Client .2 of the Virtual Machines are to be used for the docker Elasticsearch cluster, and 1 Virtual machine is the Benchmark server for the esrally docker image.

2. Using Ansible scripts Install Docker in 3 Virtual Machines and enable the docker daemon for remote access.

3. Using Ansible scripts Install Images of Elasticsearch on hosts for Docker cluster and the Images of Kibana and Esrally in the Benchmark server.

4. Using the Cloudmesh Docker application we first create a swarm cluster with the two docker hosts.Then we create a service in the Swarm Manager Node.Along with the creation of the service we pass parameters to specify the number of replicas , the network to be used, the mode of replication and the service name.

```
ServiceMode.mode="replicated"
ServiceMode.replicas=4
EndpointSpec.ports=["9200:9200"]
networks=["elastic_cluster"]
env=["SERVICE_NAME=elasticsearch"]
```

   Swarm mode containers cannot use the underlying host network as in the docker mode to enable the communication between the swarm containers we created an "overlay" network in the swarm manager.This network is passed in the service creation.So every container that is created by the swarm mode Manager will run on this network.In the swarm mode to enable elastic search unicast discovery on the start of the elastic search cluster using the Service name environmental variable we identify other containers available in the cluster and dynamical set the

```
discovery.zen.ping.unicast.hosts
```

   parameter to enable elastic search to find and form a cluster with other Elasticsearch applications in the swarm.

6.3. Elasticsearch cluster Docker and Swarm mode benchmark results

Table 13 summarises the benchmark results between the clusters in the docker and swarm modes.The results indicate that barring minor differences the clusters in both the docker and swarm modes have similar results.So we can conclude that the docker swarm mode in spite of having the additional overhead need for networking and cluster management has nearly nil impact on the performance of the application deployed on it.The above finding combined with the benefit of the inbuilt scalability and fault tolerance capabilities of the docker swarm mode make it a clear winner.

7. BENCHMARKING CLOUDMESH DOCKER

We performed benchmarking of the cloudmesh docker application for Docker and swarm commands.The benchmark was performed both in remote mode (Cloudmesh docker client is run on a network outside the cloud data centre) and local mode (Cloudmesh Docker client is run from a VM inside the cloud data centre). We performed the benchmarking for both the options on both the Amazon Webservices[12] and Chameleon cloud[13]. The results are plotted and tabulated as below

Each of the benchmark runs were performed 100 times for a defined set of operations similar to the steps performed for setting up an elastic search cluster in Docker and swarm.The results were gathered as a CSV file and plotted using Ipython[14].

The hardware specifications used on both the clouds is detailed in Table 14

Table 13. Elastic search Benchmark Results Docker Vs Swarm

Operation	Unit	Docker	Swarm
Flush time	min	0.9709	1.34333
Indexing time	min	117.888	136.951
Merge throttle time	min	75.5648	87.8035
Merge time	min	146.693	179.403
Refresh time	min	27.4014	32.6458
articles_monthly_agg_cached	ops/s	20.0178	20.0175
articles_monthly_agg_uncached	ops/s	20.0085	20.0093
default	ops/s	20.0133	20.007
force-merge	ops/s	1.75528	0.943048
index-append	docs/s	535.527	461.233
index-stats	ops/s	49.8993	50.2674
node-stats	ops/s	49.6913	50.2767
phrase	ops/s	20.0127	20.0129
scroll	ops/s	1.31822	0.457152
term	ops/s	20.0126	20.011

Table 14. Cloud Hardware Specification

	Chameleon	Aws
VM	3	3
Containers	5	5
OS	Ubuntu 16.04	Ubuntu 16.04
Flavor	m1.large	t2.large
VCPU	4	2
Memory	8 GB	8 GB
Storage	80 GB	80 GB

7.1. Docker Mode - Results

Below are the categories of the bench mark results

1. Chameleon Docker Mode Local Client Figure 6
2. Chameleon Docker Mode Remote Client Figure 7
3. Aws Docker Mode Local Client Figure 8
4. Aws Docker Mode Remote Client Figure 9

Based on the benchmark results we can infer the below details

1. In the battle of the clouds, Aws is around 20 percent faster than Chameleon cloud in docker mode
2. Cloudmesh docker operations for the docker command performed in a local network are between 25 and 30 percent faster.We also noticed some network issues when performing the test from a remote network, however, we chose to ignore those outliers in the plot.

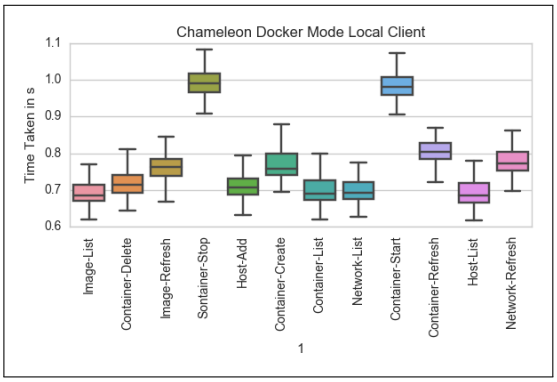


Fig. 6. Chameleon Docker Mode Local Client

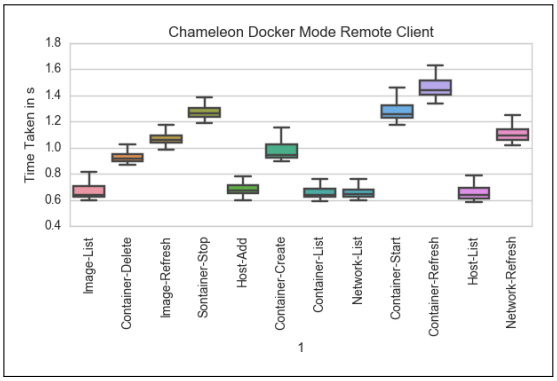


Fig. 7. Chameleon Docker Mode Remote Client

3. The standard deviation of the response times is significantly lower for Aws than Chameleon indicating that Aws is much more stable and reliable in performance than the chameleon cloud
4. The mean container create times range between 0.5 to 1 s which is significantly faster than normal VM boot times on the cloud.

7.2. Swarm Mode - Results

Below are the categories of the Benchmark results

1. Chameleon Swarm Mode Local Client Figure 11
2. Chameleon Swarm Mode Remote Client Figure 10
3. Aws Swarm Mode Local Client Figure 12
4. Aws Swarm Mode Remote Client Figure 13

Based on the benchmark results we can infer the below details

1. In the battle of the clouds, Aws is around 20 percent faster than Chameleon cloud in swarm mode
2. Cloudmesh swarm operations for the swarm command performed in a local network are between 25 and 30 percent faster.
3. The standard deviation of the response times is significantly lower for Aws than Chameleon indicating that Aws is much more stable and reliable in performance than the chameleon cloud

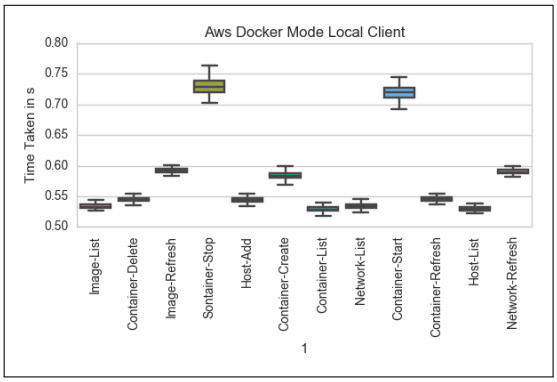


Fig. 8. Aws Docker Mode Local Client

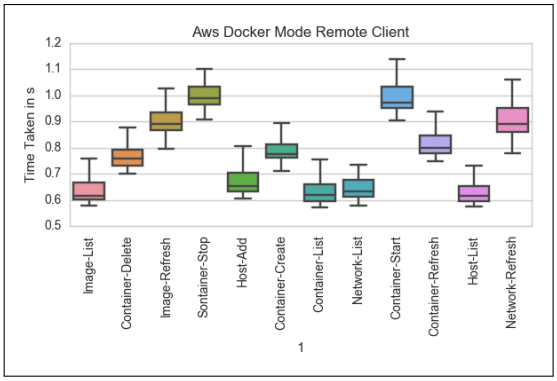


Fig. 9. Aws Docker Mode Remote Client

4. The mean service create times range between 2 to 1.6 s for four replicated containers which is significantly faster than normal boot times for a similar number of VM on the same cloud.

Table 15. Docker Mode AWS VS Chameleon Local Vs Remote

		Aws		Chameleon	
		Local	Remote	Local	Remote
Image-List	mean	0.534	0.661	0.695	0.704
Image-List	std	0.004	0.128	0.039	0.197
Container-Delete	mean	0.545	0.785	0.721	0.951
Container-Delete	std	0.004	0.090	0.040	0.115
Image-Refresh	mean	0.592	0.925	0.763	1.139
Image-Refresh	std	0.005	0.109	0.040	0.298
Sontainer-Stop	mean	0.730	1.017	0.992	1.299
Sontainer-Stop	std	0.014	0.122	0.041	0.113
Host-Add	mean	0.544	0.691	0.710	0.727
Host-Add	std	0.004	0.114	0.038	0.194
Container-Create	mean	0.584	0.798	0.767	1.007
Container-Create	std	0.006	0.075	0.042	0.164
Container-List	mean	0.529	0.655	0.697	0.689
Container-List	std	0.004	0.110	0.038	0.141
Network-List	mean	0.534	0.668	0.700	0.679
Network-List	std	0.005	0.098	0.035	0.115
Container-Start	mean	0.720	1.018	0.985	1.310
Container-Start	std	0.011	0.145	0.044	0.169
Container-Refresh	mean	0.546	0.824	0.805	1.509
Container-Refresh	std	0.004	0.070	0.033	0.208
Host-List	mean	0.530	0.659	0.693	0.708
Host-List	std	0.004	0.125	0.042	0.273
Network-Refresh	mean	0.591	0.946	0.780	1.137
Network-Refresh	std	0.005	0.171	0.043	0.151

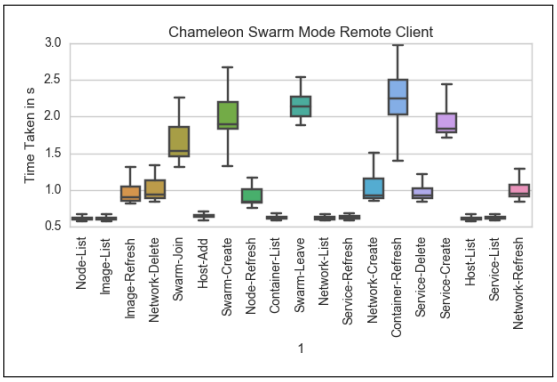


Fig. 10. Chameleon Swarm Mode Remote Client

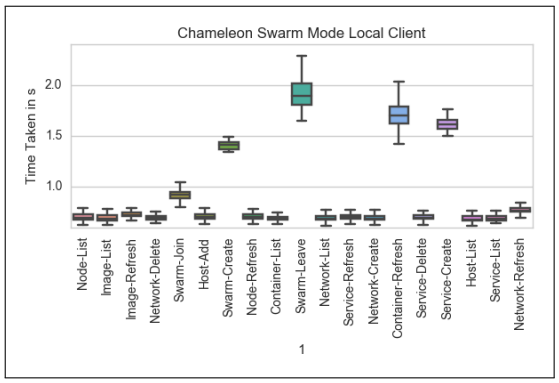


Fig. 11. Chameleon Swarm Mode Local Client

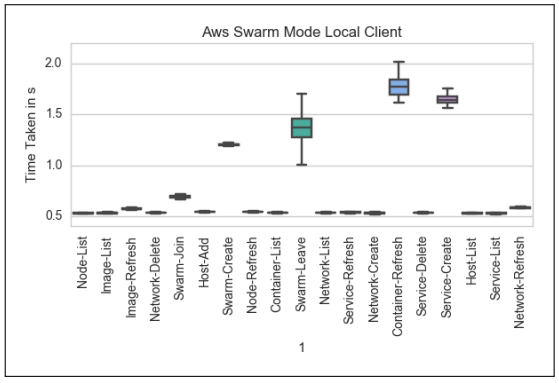


Fig. 12. Aws Swarm Mode Local Client

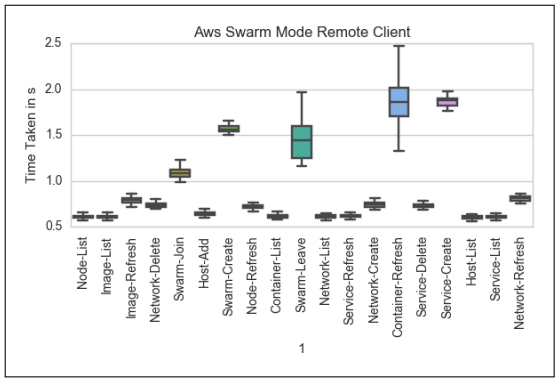


Fig. 13. Aws Swarm Mode Remote Client

Table 16. Swarm Mode AWS VS Chameleon Local Vs Remote

		Aws		Chameleon	
		Local	**Remote**	**Local**	**Remote**
Node-List	mean	0.529	0.612	0.704	0.631
Node-List	std	0.004	0.022	0.037	0.056
Image-List	mean	0.532	0.614	0.696	0.626
Image-List	std	0.004	0.041	0.039	0.053
Image-Refresh	mean	0.571	0.818	0.732	0.981
Image-Refresh	std	0.006	0.139	0.035	0.220
Network-Delete	mean	0.536	0.822	0.701	1.024
Network-Delete	std	0.005	0.280	0.035	0.247
Swarm-Join	mean	0.690	1.178	0.925	1.666
Swarm-Join	std	0.015	0.345	0.053	0.270
Host-Add	mean	0.542	0.653	0.714	0.665
Host-Add	std	0.005	0.062	0.035	0.076
Swarm-Create	mean	1.201	1.640	1.320	1.963
Swarm-Create	std	0.046	0.342	0.213	0.340
Node-Refresh	mean	0.542	0.738	0.714	0.911
Node-Refresh	std	0.004	0.107	0.039	0.127
Container-List	mean	0.533	0.622	0.696	0.638
Container-List	std	0.004	0.039	0.033	0.050
Swarm-Leave	mean	1.419	1.460	1.932	2.143
Swarm-Leave	std	0.275	0.288	0.241	0.186
Network-List	mean	0.532	0.625	0.698	0.625
Network-List	std	0.004	0.050	0.035	0.027
Service-Refresh	mean	0.536	0.631	0.710	0.645
Service-Refresh	std	0.004	0.089	0.039	0.099
Network-Create	mean	0.531	0.781	0.698	1.009
Network-Create	std	0.005	0.156	0.035	0.162
Container-Refresh	mean	1.666	1.846	1.693	2.273
Container-Refresh	std	0.374	0.348	0.181	0.386
Service-Delete	mean	0.535	0.781	0.704	0.991
Service-Delete	std	0.004	0.205	0.039	0.140
Service-Create	mean	1.661	1.905	1.636	1.938
Service-Create	std	0.071	0.164	0.115	0.273
Host-List	mean	0.529	0.608	0.693	0.629
Host-List	std	0.004	0.030	0.033	0.083
Service-List	mean	0.528	0.617	0.698	0.639
Service-List	std	0.005	0.052	0.035	0.075
Network-Refresh	mean	0.583	0.834	0.776	1.009
Network-Refresh	std	0.005	0.145	0.039	0.126

8. CONCLUSION

In this project, we have successfully integrated docker and swarm capabilities into the cloudmesh client.We have also demonstrated its use for a practical use case of Setting up an Elastic search cluster in Docker and swarm modes.We have also benchmarked the commands for multiple clouds(AWS and Chameleon) in both local and remote modes and detailed the results and insights.The Ansible scripts as part of the project along with the capabilities built in the cloudmesh docker application provide for a seamless capability in deploying and provisioning applications in Docker and swarm containers.

9. ACKNOWLEDGEMENT

We acknowledge our professor Gregor von Laszewski and all associate instructors for helping us and guiding us throughout this project.

REFERENCES

[1] Docker Inc., "Docker," Web Page, 2017, accessed 2017-04-23. [Online]. Available: https://www.docker.com/

[2] G. von Laszewski, F. Wang, H. Lee, H. Chen, and G. C. Fox, "Accessing Multiple Clouds with Cloudmesh," in *Proceedings of the 2014 ACM International Workshop on Software-defined Ecosystems*, ser. BigSystem '14. New York, NY, USA: ACM, 2014, pp. 21–28. [Online]. Available: http://doi.acm.org/10.1145/2609441.2609638

[3] Docker Inc., "Swarm," Web Page, 2017, accessed 2017-04-23. [Online]. Available: https://docs.docker.com/engine/swarm/

[4] MongoDB, Inc., "MongoDB," Web Page, 2017, accessed 2017-04-23. [Online]. Available: https://www.mongodb.com/

[5] N. Iarocci, "Python Eve," Web Page, 2017, accessed 2017-04-23. [Online]. Available: http://python-eve.org/

[6] Python Software Foundation, "Python," Web Page, 2017, accessed 2017-04-23. [Online]. Available: https://www.python.org/

[7] Red Hat, Inc., "Ansible," Web Page, 2017, accessed 2017-04-23. [Online]. Available: https://www.ansible.com/it-automation

[8] K. Venkatesan, A. Vuppuda, and G. von Laszewski, "Cloudmesh docker," Code Repository, Apr. 2017, accessed 2017-04-23. [Online]. Available: https://github.com/cloudmesh/cloudmesh.docker

[9] Docker Inc., "Docker python sdk," Web Page, 2017, accessed 2017-04-23. [Online]. Available: https://docker-py.readthedocs.io/en/stable/

[10] Elasticsearch BV, "ElasticSearch," Web Page, 2017, accessed 2017-04-23. [Online]. Available: https://www.elastic.co/

[11] D. Mitterdorfer, "ESRally," Web Page, 2017, accessed 2017-04-23. [Online]. Available: http://esrally.readthedocs.io/en/latest/quickstart.html

[12] Amazon Web Services, Inc., "Amazon Webservices," Web Page, 2017, accessed 2017-04-23. [Online]. Available: https://aws.amazon.com/?nc2=h_lg

[13] National Science Foundation, "Chameleon," Web Page, 2017, accessed 2017-04-23. [Online]. Available: https://www.chameleoncloud.org/

[14] IPython development team, "Ipython," Web Page, 2017, accessed 2017-04-23. [Online]. Available: https://ipython.org/

AUTHOR BIOGRAPHIES

Karthick Venkatesan received his BE(Bachelor of Engineering) from Anna University ,India. He is currently pursuing MS in Data Science at Indiana University Bloomington.

Ashok Vuppada received his BE(Bachelor of Engineering) from Andhra University ,India. He is currently pursuing Graduate Certificate in Data Science at Indiana University Bloomington.

A. APPENDICES

A.1. Appendix A: Work Distribution

The co-authors of this report worked together on the design of the technical Solutions, implementation, testing and documentation. Below given is the work distribution

- Karthick Venkatesan
  - Design and Implementation of Docker and Swarm Commands.
  - Integration of Docker and Swarm Commands to Docker API.
  - Integration to cloudmesh.common,cloudmesh.rest, cloudmesh.cmd5 repositories.
  - Framework definition and wrapper class built for Python-eve
  - Ansible scripts for Docker image installation and setup of etc hosts
  - Test scripts for Docker and Swarm command
  - Dockerfile for installation of cloudmesh.docker
  - Create Benchmark scripts for Local and Remote Benchmarking on Chameleon and AWS
  - Execute Benchmark scripts for Chameleon and Aws and plot the results in Ipython
  - Scripts for setup of Elasticsearch Docker cluster
  - Benchmark Elastic search swarm cluster using ESRally and document results
  - Writing related sections in this report.
- Ashok Vuppuda
  - Design of Docker and Swarm Commands.
  - Integration into cloudmesh.rest
  - Python-eve integration and implementation for Docker and Swarm Modes
  - Ansible scripts for Docker installation
  - Test application on Aws and Chameleon clouds
  - Execute Benchmark scripts for Chameleon and Aws and plot the results in Ipython
  - Benchmark Elastic search Docker cluster using ESRally and document results
  - Writing related sections in this report.

Deployment of Vehicle Detection application on Chameleon clouds

ABHISHEK NAIK[1,*] AND SHREE GOVIND MISHRA[2,*]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[2] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*] Corresponding authors: absnaik810@gmail.com, shremish@indiana.edu

project-001, May 4, 2017

This project focuses on the deployment of Vehicle Detection application on multiple Chameleon clouds using Ansible playbook. It also focuses on the benchmarking of the deployment results and its analysis.

Keywords: Vehicle detection, Ansible, Cloudmesh, OpenCV, Haar Cascades, Cloud, I524

https://github.com/cloudmesh/sp17-i524/blob/master/project/S17-IR-P010/report.pdf

1. INTRODUCTION

Vehicle Detection forms an integral part of the development of new technologies like fully self-driving cars, etc. One of the techniques to perform such detection is by using Haar Cascades [1]. This technique has been applied to vehicle detection by creating a haar-cascade cars.xml file which has been trained using 526 rear-end images of cars. In this project, we would be extending this vehicle detection approach to enable it to run on multiple clouds. This deployment would initially be done on the localhost, followed by deployment on the cloud. Cloudmesh client would be used for cloud management and Ansible scripts would be used for software stack deployment. The vehicle detection application would then run remotely onto the clouds. The result would be an image (a .jpg file) with all the vehicles in it detected with a red rectangle around it and sent back to the local host. Appropriate benchmarking would be carried out at each iteration using some benchmarking technique.

2. REQUIREMENTS

The requirement was to deploy the Vehicle detection application onto the cloud's Virtual Machines (VMs). The processing, which included detection of vehicles by drawing a red rectangle around them, had to be carried out on the cloud VMs. The output image that was generated had to be then resent back to the local machine.

3. SOFTWARE STACK

For this project, we have used the following software/applications:

- Ansible [2]:
  Ansible is an open source platform that is used for automation. Configuration management, task automation and application deployment can be carried out using Ansible. In our project, we have used Ansible to manage the configuration and automate the deployment of the software stack onto the clouds. We run Ansible scripts via localhost entering the IP addresses of the cloud's VMs and the software/applications are installed in those VMs. Thus, we are using Ansible to orchestrate the software deployment via playbooks using the inventory.txt file containing the IP addresses.

  Ansible forms connections to our cloud VMs and then pushes 'Ansible modules', which are nothing but small programs, to them. Ansible executes these modules and then removes them upon completion. Ansible thus helped us in lightweight development, in the sense that other than an editor and the terminal, it did not require any other components to be installed. Although we used SSH keys for the connections and with Ansible, Ansible also supports the use of passwords.

- Cloudmesh client [3]:
  Cloudmesh is basically used for cloud VM management. It provides easy access to different cloud environments via a command shell and command line. In our project, we have used Cloudmesh client for dynamic management of the cloud VMs. Cloudmesh client enabled us to boot up the virtual machines on the clouds, assign floating IPs, etc. Thus, we have used it to carry about these activities as well as add our SSH keys to a local database and then SSH to the remote VMs on the clouds using various cloudmesh client commands and utilities.

- Python [4]:
  Python is an object oriented, high level programming lan-

guage. It is an interpreted language and provides built-in data structures with dynamic typing and binding. It has an easy and simple yet intuitive syntax. In our project, we are using Python (and Pip [5]) for the installation and management of software written in Python.

- Git [6]:
  Git is a version control system that can be used for tracking and change management when multiple people are involved in a project. It helps control and coordinate the working among a group of people. We have used Git to manage our work within our team. It has helped us by providing a central place wherein we could commit our work and make it easily accessible to others. Also, the Vehicle Detection Application (outlined below) application that we are using in our project has been developed and hosted on Github by the creator Andrews Sobral. We have forked his repository and using his application for the detection of vehicles.

- OpenCV [7]:
  OpenCV is a collection (library) of various functions used to execute computer vision related applications. The Vehicle Detection Application runs on OpenCV. OpenCV includes both, a trainer as well as a detector. It also has many pre-trained classifiers. We are using the Haar classifier [1].

- Vehicle Detection Application [8]:
  The Vehicle Detection Application has been developed by Andrews Sobral. It basically uses a haar-cascade cars.xml for vehicle detection. The haar-cascade was trained using 526 rear-end images of the cars. The video size used is 360x240 pixels. In our project, we are running this application on the clouds.

- Bash script [9]:
  Bash typically helps us in the automatic execution of various Linux commands by creating a .sh file. We have used a file myScript.sh in order to carry out the benchmarking of the project. It should be noted that once a shell script is created, appropriate permissions (or privileges) must be set so that it is executable. We call the ansible script to be executed within this shell script. We can thus say that the Bash script is the starting point of our project - we just have to run this script and then the software stack including the Vehicle detection application would be deployed, the application would be run and the generated image file would be rerouted to our local machine. In order to carry out benchmarking, we just did a minor modification to this routine (edited the inventory.txt file).

4. DEPLOYMENT

For successful project deployment, the first step was gaining access to the clouds. Cloudmesh client was used for gaining this access. In order to set up this cloudmesh client, we had to configure the cloudmesh.yml file with our credentials and other details. We mainly used Chameleon clouds in our project so we only edited the cloudmesh.yml file parts that corresponded to chameleon clouds. If we were to use Jetstream, we would have required to edit the Jetstream part as well. We had to edit the values in such a way as to ensure that the info command did not yield any To Be Decided (TBD) values. In case it did, then it meant that there were possibly some errors and we had to revisit the cloudmesh.yml file and correct those. If there were no TBD values at all, then it meant that we were good and could now access the clouds. Once we had access to the clouds, we booted a VM and then assigned a floating IP to it. Post this, we used the SSH command to log in into the VM. Once we were within the VM, we could use it as a normal local Ubuntu machine. Although we didn't require it in this project, cloudmesh also provided the functionality to use a different image on the cloud VMs.

When we need a new VM, we simply booted a new VM and assigned a new floating IP address to it. This resulted in allocation of a new VM which we could then use as a new additionL machine. By proper set up in the Ansible scripts, one of these machines could be used as a master and the other one(s) as slaves, if there is a need in some project.

The second step was the identification of the software stack that would be required. In order to understand this, we first deployed the software and application on the local machines and then on the clouds. This step helped us understand the software needed as well as the dependencies amongst them. Once we identified the software, we developed an ansible script to deploy these software and applications dynamically in an automated way.

Ansible is an open source automation platform. It mainly uses .yml files for its working. The file 'inventory.txt' contains the details of the hosts (their IP addresses) wherein the software stack is to be installed. Similarly, we can also mention the ansible username in this file. Thus, in our case, the entry in the inventory.txt file is as below:

```
129.114.110.83 ansible_ssh_user=cc
```

The 'playbook.yml' file lists the hosts, variables and the roles. The hosts contains the details of the machine where the software are to be installed. The variables section lists the variables that are being used elsewhere, for e.g., '`dwnld_dr`' denoting the download directory. The advantage of using variables is that the values do not need to be hard coded and thus they can be changed everywhere with a minor update. The roles section lists the various software that need to be installed, in order. In our case, the entry of the playbook.yml file is as below:

```
---
- hosts: all
      vars:
    dwnld_dr: ~/downloads}
    ocv_ver: 2.4.13.2}
    repository: {{ repository }}
    tmp: /tmp/vehicledetection
       roles:
    - git
    - python
    - upgrade
    - opencv
    - vehicledetection
```

The 'roles' directory contains the details about all the software that are to be installed. Since we have installed four software, we have four directories within it. Each of these four directories is named after the software that it is supposed to install. Thus, the directory 'git' installs git, 'python' installs python and so on. Each of these directories in turn contain two directories - 'defaults' and 'tasks'. The 'defaults' directory contains a file 'main.yml' that lists the temporary variables like the download directory to be used, the version to be downloaded, etc that

is specific to its software. The 'tasks' directory contains a file named 'main.yml' that lists the tasks (i.e., the activities) that need to be carried out step-by-step. These activities are executed in a sequential order. A snapshot of one of the main.yml files used in our project is as below:

```
---
- name: install Git on Ubuntu machine
  become: yes
  apt: name=git state=present
```

Thus, the first statement says that we are installing Git on the Ubuntu. The next statement says that sudo privileges would be required for installation of Git. The last statement specifies the package name and its state. Ansible thus enabled us to dynamically deploy the software stack and configure the system as required for proper installation.

5. EXECUTION

For the execution of this project, we had set the tasks as below:

5.1. Week 1

In the first week, we deployed the vehicle detection application on the localhost by using bare commands. The main aim of this step was to ensure that the application worked. This step was critical in the sense that it helped us understand the various dependencies among the various software in the software stack. It also helped us understand the environmental variables like `OpenCV_FOUND` and `OpenCV_DIR` that we had to setup, since they are specific to the local system on which the application is run. This step in reality took us more than a week to find out about `OpenCV_DIR`, `OpenCV_FOUND` and their specific expected values. Once this step was executed successfully, we wrote down an Ansible script to carry out the software deployment. Since we spent substantial amount of time in debugging the `OpenCV_FOUND` and `OpenCV_DIR` errors, we could not write the scripts for the entire software stack. We wrote it only for Git; but it provide us with a good start.

5.2. Week 2

In this week, our aim was to reserve and access a Chameleon cloud using cloudmesh client. We then planned to carry out software stack deployment on it, and carry out benchmarking. However, we spent more than four days in trying to debug the `OpenCV_DIR` and `OpenCV_FOUND` errors that we encountered in the first week. Nonetheless, we found out a solution and tested that the application works as expected. As a consequence of the error we faced, we could not spend much time in developing ansible scripts for further software stack deployment. However, by the end of second week, we had configured our cloudmesh.yml file. We were yet to test it, though.

5.3. Week 3

For week 3, we tested our configuration of the cloudmesh.yml file by actually gaining access to the Chameleon clouds. For this, we learned about the various commands that cloudmesh had to offer and booted up a VM in the clouds. Later, we assigned a floating IP address to it and then logged in into the remote VM via SSH. Once we were into the VM, we could operate on it just like a local normal VM. We spent the next couple of days in writing down ansible scripts for software stack deployment. Once this was complete, we ran those scripts on the local host to test if they worked as expected, or if they needed any changes.

5.4. Week 4

In this week, we carried out minor changes to the ansible scripts that we had written. We booted up a cloud VM using cloudmesh client and logged in into it. We then ran the ansible scripts from our local hosts by entering the floating IP of our cloud VMs in the inventory.txt file. While the scripts for git ran successfully, the script for OpenCV failed. After much debugging and testing, we found a solution which involved updating the cache (running an equivalent of sudo apt-get update) which resulted in OpenCV being installed successfully on the clouds. We faced another challenge when cloning the Vehicle detection project from github. We circumvented this problem by making the ansible script run the command as a sudo user.

5.5. Week 5

In the final fifth week, we aimed to carry out benchmarking and write a report about our observations. We first carried out benchmarking by deploying the software stack on the local machine and taking the readings. Then, we deployed it onto the Chameleon clouds and noted down the observations. Finally we noted down all these observations in a detailed report.

5.6. Week 6

By this week, the majority of our project was ready. We, however, still had one significant change to make - re-factor the code in order to make it suitable to be deployed on the clouds. Since we were interested in *saving* the image so as to send it back to the local machine, we replaced the cvShowImage() attribute to cvSaveImage(), passing it similar parameters. This method created a snapshot of the video and placed the output video file in the same directory as that of the main programs. In our case, the image file denoting the detected vehicles looks as shown in Figure 1.

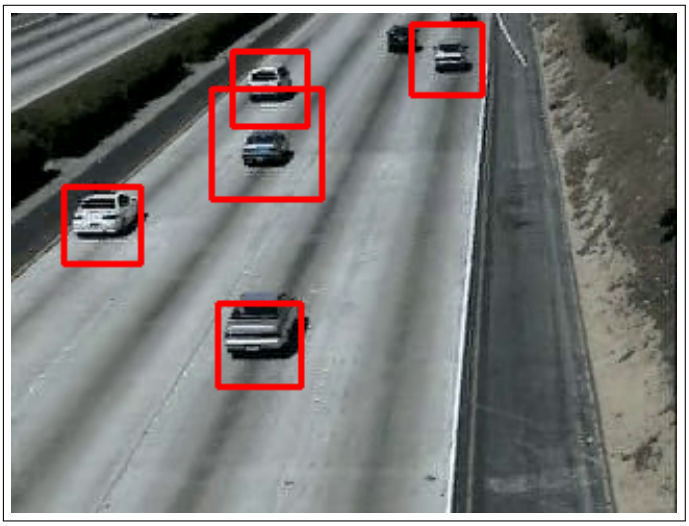


Fig. 1. Image showing the detected vehicles sent back to local machine from the cloud VM

Other than this, we were also getting a warning "Gtk - Could not open display". After some debugging, we found out that the reason for this error was that within the vehicle detection application, the algorithm was trying to create a new window. As per our requirement, we didn't have any need to open a window and display the contents there. Hence we simply removed this

call that the algorithm made and instead added some other code due to which the image would be re-routed to the local host.

Post these two changes, the application ran successfully on the clouds, carrying out all the processing on the remote VMs. Once the required output file was created on the remote VM, it would create a directory on the host machine with the name as the floating IP address assigned earlier. Within this directory, another directory named dev containing the vehicledetection folder was created. The final image was placed within this vehicledetection folder. Thus, as can be seen, ansible provides a robust and very secure way to access the systems (local as well as remote).

6. BENCHMARKING RESULTS AND ANALYSIS

For benchmarking, we first deployed the software stack onto the local machine and then deployed it on the clouds. We noted down the observations in each case. The parameters for benchmarking as well as the observed values have been noted below. In order to reduce any errors, we have considered average values. Thus, we installed each software (from scratch) 3 times on the local machine and 3 times on the clouds. Then, we took the average of all these values in order to find out the exact time needed for its deployment on the platform.

Also, for measuring the time, we used a shell script Setup.py. This script displays the current time and then calls onto the Ansible playbook. This Ansbile script then installs all the software on the software stack and then final comes to an end. Once the Ansible script ends, the shell script again denotes the latest time. The difference between these two timings displayed at the beginning and at the end give us the total software installation time required. While testing it for different software, we had edited the playbook.yml file to contain only the single software under consideration. In this way, we understood the time required for the installation of just a single software.

6.1. Deployment benchmarking

Deployment benchmarking included observing the software deployment times for each of the software in the software deployment stack. We started off with the first software, Git. On the local machine, it was installed in 2.567 seconds, while it took 10.447 seconds to be installed on the clouds. The next command, upgrade, took 35.837 seconds on the local machine and 155.03 seconds on the clouds. Python took 4.554 seconds to be installed on the local machine and 30.748 seconds to be installed on the clouds. OpenCV, is a heavy software having lots of dependencies on other software. We thus had to install it dependencies first before we could compile and install it. Due to all this, OpenCV took 1420.373 to be installed on the clouds and 276.014 seconds on the local machine. Vehicle detection is a lightweight application and as such, it took 11.987 to install on the clouds and just 2.04 seconds to be installed on the local machine. All these values have been tabulated in Table 1.

Table 1. Deployment Benchmark

	Local VM avg time	Cloud VM avg time
Git	2.5863333333	10.494
Upgrade	35.531	155.5903333333
Python	4.2873333333	30.5766666667
OpenCV	277.032	1419.5266666667
Vehicle Detection	2.1833333333	11.7166666667

6.2. Elasticity benchmarking

Elasticity metric aims to test the dependency of one software on the other ones. In other words, it aims to test a software's sensitivity to changes in another software. In our project, we noticed that some of the software had a high dependency (and sensitivity), while others had relatively low (or no) dependency. For benchmarking the software elasticity, we referred its ansible script. We counted the number of tasks that had to be successfully executed before the software in question could be considered to be successfully installed. For e.g., in case of the Vehicle Detection application, before running the 'make' command, we need to carry out 3 steps - getting the Vehicle detection package source, changing the directory and running cmake. Thus, the elasticity factor associated with the Vehicle Detection project is 3. We carried out a similar analysis of the other software from the software stack. Table 2 lists these observations.

Table 2. Elasticity benchmark

Software/Application	Dependency count
Git	0
Upgrade	NA
Python	4
OpenCV	14
Vehicle Detection	3

6.3. Re-installation benchmarking

Ansible scripts do a good job of installing the software. However, even before the software/applications are installed, ansible first tests to make sure that the software is not already installed on the system. In case it is, then it simply skips over the installation process of that software and then continues with the next one. This is important, since it does not try to update/reinstall the existing software since that would lead to conflicts with the existing package versions or features. Also, since the re-installation of packages is just skipped, in case of failures, there is no need to edit the inventory.txt file to deploy only the failed software - it can be kept as it is and the ansible scripts can be run from scratch. Only the failed software would be installed. This also helps in reducing the installation time.

6.4. Workload benchmarking

For Workload benchmarking, we found out the disk utilization of the various software from the software stack. While software such as Git were pretty light, others like OpenCV were pretty heavy. They not only required a long time to install, but also consumed a lot of disk space. Figures 2 and 3 denote the output of the 'df -h' command on the cloud VMs. As shown in Figure 2, we had used only 1.3 GB of system space before installing OpenCV while we ended up consuming 4.4 GB of it after its installation. The usage thus increased from 7 percent to 24 percent.

6.5. Security and trust factors

In our project we have used cloudmesh client which in turn uses the SSH protocol to make secure connections to the VMs

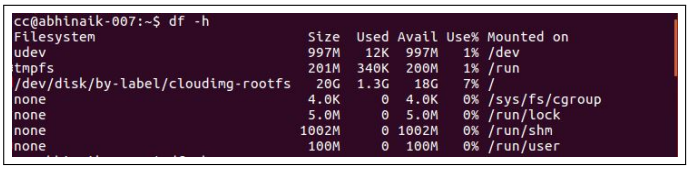


Fig. 2. Usage statics before OpenCV installation

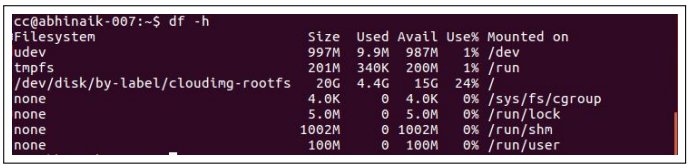


Fig. 3. Usage statics after OpenCV installation

on the clouds [10]. SSH is a cryptographic network protocol used for establishing secured connections over the network. It is thus used for access to shell accounts on Unix like operating systems. Since SSH is being used, trustworthy connections can be established between the local machine and the remote VMs on the clouds. Along with this, Ansible also helps in security enforcement. Some of the qualities like being agentless, capability to support SSH, no unnecessary changes, modular structure, etc [11].

6.6. Latency and reliability benchmarking

We used the ping facility for benchmarking the latency [12] of the network. Latency measures the time it takes for a packet to reach the cloud VM (with a specific IP) from the local host. We used the following command to test this:

```
ping 129.114.33.88 -c 10
```

In this command, the '-c 10' option makes 10 requests to the cloud VMs. In the end, it displays a summary of the results. We can thus observe that the minimum round trip time (rtt) was 40.499 seconds, while the maximum was 72.198. Along with all this, we also noticed that there was 10 percent packet loss, since out of the 10 packets that were sent, only 9 of them could make it back to the local host. However, repeated experiments showed that the system was indeed reliable and that this was a small exception. Figure 4 shows the latency test results

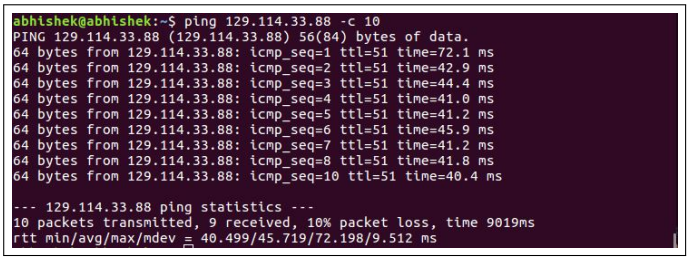


Fig. 4. Latency Test results

7. CONCLUSION

In this project, we used cloudmesh client which is a client that enables us to access and manage various cloud environments. We used it to gain access to Chameleon cloud VMs by using the SSH protocol. Ansible is an automation platform that helps us in automating the software deployment. We used Ansible scripts to install all our software along with the configuration of the project. Once the software stack was installed, we installed the Vehicle detection application on top of it. The Vehicle detection application running on OpenCV used haar-classifiers to detect the vehicles. It created an image wherein the vehicles were marked with a red rectangle. This image, which was generated on the cloud, was then redirected to the local machine, using ansible fetch. On the local machine, it was saved deep inside the directory named with the IP address. Once this end-to-end process was done, we did benchmarking of the system by using shell scripts. We typically focused on the deployment, elasticity, installation capacity, workload, security, latency and reliability.

8. ACKNOWLEDGEMENTS

This project was undertaken as a part of the course objective for I524: Big Data and Open Source Software Projects at Indiana University, Bloomington. We would like to thank Prof. Gregor Von Laszewski and all the TAs for their help. Similarly we would also like to thank Andrews Sobral for providing us with the Vehicle Detection application that we ran on the cloud VMs.

REFERENCES

[1] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Third IEEE International Conference on e-Science and Grid Computing (e-Science 2007)*. Cambridge, MA: Institute of Electrical and Electronics Engineers (IEEE), Dec. 2001. [Online]. Available: http://wearables.cc.gatech.edu/paper_of_week/viola01rapid.pdf

[2] Wikipedia, "Ansible (software)," Web page, online; accessed 10-Mar-2017. [Online]. Available: https://en.wikipedia.org/wiki/Ansible_(software)

[3] G. von Laszewski, "Cloudmesh/client," Code Repository, accessed: 2017-3-10. [Online]. Available: https://github.com/cloudmesh/client

[4] Python Software Foundation, "What is python? executive summary," Web page, online; accessed 21-Mar-2017. [Online]. Available: https://www.python.org/doc/essays/blurb/

[5] Wikipedia, "pip (package manager)," Web page, online; accessed 20-Mar-2017. [Online]. Available: https://en.wikipedia.org/wiki/Pip_(package_manager)

[6] Wikipedia, "Git," Web page, online; accessed 19-Mar-2017. [Online]. Available: https://en.wikipedia.org/wiki/Git

[7] Wikipedia, "Opencv," Web page, online; accessed 19-Mar-2017. [Online]. Available: https://en.wikipedia.org/wiki/OpenCV

[8] A. Sobral, "Vehicle detection by haar cascades with opencv," Code Repository, accessed: 2017-3-10. [Online]. Available: https://github.com/andrewssobral/vehicle_detection_haarcascades

[9] Wikipedia, "Bash (unix shell)," Web page, online; accessed 11-Mar-2017. [Online]. Available: https://en.wikipedia.org/wiki/Bash_(Unix_shell)

[10] Wikipedia, "Secure shell," Web page, online; accessed 18-Mar-2017. [Online]. Available: https://en.wikipedia.org/wiki/Secure_Shell

[11] Red Hat, Inc., "The inside playbook," Web page, online; accessed 21-Mar-2017. [Online]. Available: https://www.ansible.com/blog/security-automation

[12] Red Hat, Inc., "The inside playbook," Web page, online; accessed 21-Mar-2017. [Online]. Available: https://www.ansible.com/blog/security-automation

Head Count Detection Using Apache Mesos

ANURAG KUMAR JAIN[1,*], PRATIK SUSHIL JAIN[1,*], AND RONAK PAREKH[1,*]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*] Corresponding authors: jainanur@iu.edu, jainps@iu.edu, parekhr@iu.edu

Project P011, May 4, 2017

Apache Mesos can be used for scaling of applications and providing highly available clusters. It provides the functionality to use resources from multiple agents and execute the application using the offered resources. Deployment of a custom face detection application is done using Ansible on Apache Mesos and analysis of the deployments are benchmarked. The face detection application was created using OpenCV. Ansible playbooks are used to deploy Apache Mesos on virtual machines created on chameleon cloud. For benchmarks, deployment of Mesos on different systems and different sizes of virtual machines were carried out.

Keywords: Cloud, I524, Apache Mesos, OpenCV, Face detection, Mesospehere, Ansible, Big Data, Zookeeper, Marathon, Haar Cascade

https://github.com/cloudmesh/sp17-i524/blob/master/project/S17-IR-P011/report.pdf

1. INTRODUCTION

There are a number of applications which require the counting of people in an image. The application for detecting headcount in an image can be in the field of attendance management and tagging images on social media. The purpose of this project is to deploy a head count detection framework on Apache Mesos to enable distributed processing of image data over the cloud. Apache Mesos is built using the same principles as the Linux kernel but at a different level of abstraction. It runs on every machine and provides applications with API's for resource management and scheduling across different cloud infrastructures [1]. Ansible is used to automate the deployment of the head count detection application on Apache Mesos which will be deployed on Chameleon cloud. The VGG face detection dataset is used for benchmarking the performance of Mesos on different versions of the Linux Operating System. The head count detection is restricted to detection of the number of frontal faces in an image [2]. This is due to the field of face detection and image processing being vast and thus the head count detection is restricted to detecting faces based on simple criteria. The performance benefits of Apache Mesos in terms of high availability, scalability and fault tolerance are leveraged by deploying the application on a virtual machine which offers its computing resources to Mesos for fast computation.

2. WHY MESOS?

Mesos can be used to implement a decentralized scheduling approach. In this approach, each framework decides which offers to accept or reject. There are many incentives that are provided by any decentralized system. The incentives provided by Apache Mesos system includes short tasks, no minimum allocation, scale down and not accepting unknown resources [3].

3. WHAT IS MESOS?

Mesos is built using the same principles as the Linux kernel. However, it is at a different level of abstraction. The Mesos kernel runs on every machine and provides applications such as Hadoop, Spark, Kafka, Elastic Search with API's for resource management and scheduling across entire cloud environments and data center [3].

3.1. Design Philosophy

Mesos aims to provide a resilient core to enable various frameworks for efficient cluster sharing. As cluster frameworks are rapidly evolving and highly diverse, this overriding design philosophy focussing on defining a minimal interface enabling efficient resource sharing across frameworks and otherwise push control of task scheduling and execution to the frameworks. Pushing control to the frameworks has two benefits. It allows frameworks to implement diverse approaches to various problems in the cluster and to evolve these solutions independently. This also keeps Mesos simple and minimizes the rate of change required, which makes it easier to keep Mesos scalable and robust. Higher level libraries implementing common functionality are built on top of Mesos even though Mesos provides a low-level interface. Including the functionality in libraries rather than in Mesos allows it to remain small and flexible and lets the libraries evolve independently.

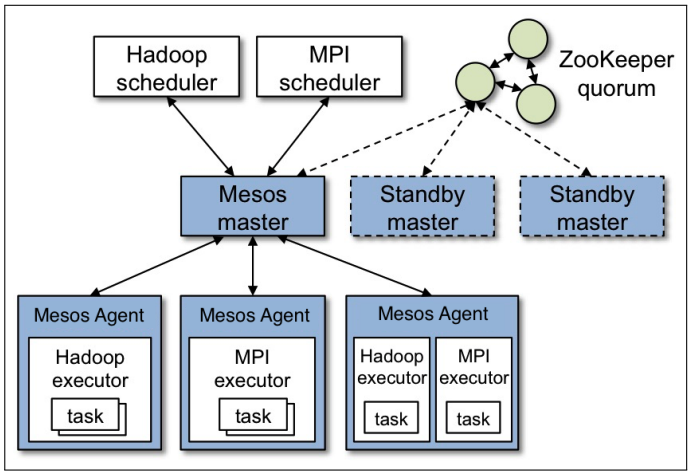


Fig. 1. Mesos architecture [4].

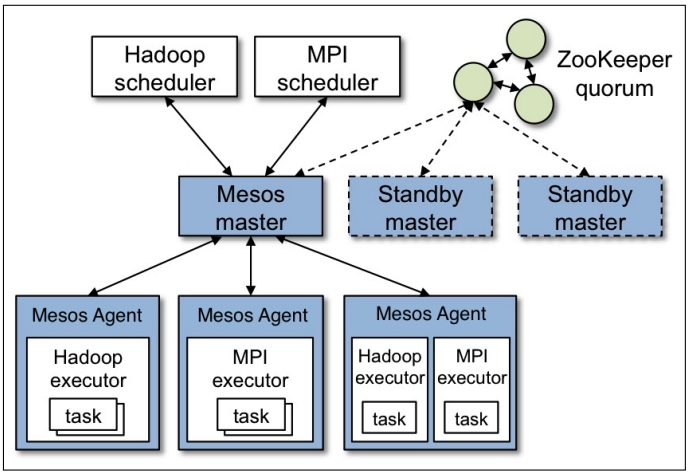


Fig. 2. Example Flow [4].

3.2. Architecture

Mesos consists of a master daemon through which an agent running on each cluster node is managed. It also consists of a Mesos Framework that runs tasks on these agents. The master enables the fine-grained sharing of resources including CPU and RAM across all the frameworks by making resource offers. Each resource offer contains a list of <agent ID, resource1:amount1, resource2: amount2,...> [4]. The master decides how many resources to offer to each framework based on an organizational policy such as strict priority or fair sharing. A diverse set of policies is supported by employing a modular architecture that makes it easy to add new allocation modules via a plug-in mechanism [1].

A framework running on top of Mesos consists of two components: an executor process that can be launched on agent nodes to run framework tasks and a scheduler that registers with the master to the resource offered. While the master determines how many resources are offered to each framework, the framework scheduler selects which of the offered resources to use. When a framework accepts offered resources, the description is passed to Mesos [4].

The sequence of events in the figure are as follows:

1. Agent 1 reports to the master that it has 4 CPU's and 4GB of memory free. An allocation policy module is invoked by the master which tells it that framework 1 should be offered all available resources [4].

2. The master sends a resource offer to framework 1 which includes details of which resources are available on agent 1.

3. A reply is sent back to the master by the frameworks scheduler with information regarding two tasks to run on the agent, using <2CPU's, 1 GB RAM> for the first task, <1CPU and 2 GB RAM> for the second task [4].

4. Finally the master sends the task to the agent which allocates appropriate resources to the frameworks executor which in turn launches the two tasks. The allocation module may now offer the unallocated resources to framework 2 [4].

The resource offer process repeats when new resources become free or when tasks are finished.

4. MESOS FRAMEWORK

A Mesos Framework is used to manage tasks. A Mesos framework is highly available only if it continues to manage tasks correctly in the presence of a variety of failure scenarios. There are failure conditions which the framework should consider. The conditions are as follows:

- A framework scheduler which is connected to a Mesos master might fail. It may be due to crashing or losing network connectivity. If the master has been configured to use the high availability mode, this will promote a replica of another Mesos master to become the new leader. In this case, the scheduler re-registers with the new master and ensure the task state is consistent [5].

- A host may fail where a framework scheduler is running. While creating the framework, the framework authors should ensure that multiple copies of the scheduler run on different nodes and a backup copy is promoted to become the new leader where a previous leader fails. This is done to ensure that the framework remains available and can continue to run new tasks [5].

- The host might fail on which a task is running. The node might not have failed but the Mesos agent on the node might be unable to communicate with the Mesos master due to a network partition [5].

4.1. Internal Working of Framework

When the agents are connected to the Mesos master, they offer resources to the Mesos master in terms of CPU's, RAM and disk space. These resources are available for the master to use till the agents are in the active state. There can be multiple states for the Mesos agents such as Active state, Deactivated state and Unreachable state. When an agent is successfully able to connect to the master, it is in the active state. When an agent is not able to advertise its IP address to the master, the agent state is changed to deactivated. If there are network outages and the agent is unable to reach the master, it goes in the unreachable state. The master is able to use the resources offered by the agent only if the agent is in the active state. When the face detection framework is started on Mesos master, the master searches for any available resource offers from the active agents. It breaks the execution of the framework into multiple tasks and uses the resource offers

to execute the task. The task runs on multiple CPU's if available, and after completion of each task a status update is sent to the Mesos master. Each individual task is assigned a unique task id. If an agent doesn't update the status of the task due to network outage or failure of the agent, the Mesos master persists the task id and it assigns the task id to a new agent which has resources available and the task is resumed. The framework state changes to active when the tasks of the framework are running and changes to completed when the execution of the framework is finished.

4.2. Highly Available Frameworks

Mesos provides unreliable messaging between components by default: Messages are delivered at most once by default. Framework authors should take into consideration that the messages they sent might not be received and should be prepared to take corrective action. Timeouts are used to detect whether a message is lost. There are situations where the framework attempts to launch a task but the message does not reach the Mesos master [5]. To address such issues a framework scheduler should set a timeout after attempting to launch a new task. The scheduler should take corrective action if the scheduler does not receive a status update and thereby launching a new copy of the task [4].

In general, distributed systems cannot distinguish between 'lost' messages and messages that are delayed. There might be a case where the scheduler might see a status update for the attempt of the first task launch immediately after its timeout has fired but it has already begun to take corrective action. Mesos provides ordered message delivery between any pair of processes. If a framework sends a message M1 and M2 to the master, then the master might not receive any messages, or it may receive only M1 or, only M2 or, M1 followed by M2. But it will never receive messages in the order M2 followed by M1 [5]. Mesos also provides reliable delivery of the task status updates. The agent persists task status updates to disk and then forwards them to the master. The master sends status updates to the appropriate framework scheduler. When a scheduler acknowledges a status update, the master forwards the acknowledgment back to the agent, which allows the stored status update to be garbage collected. If the agent does not receive an acknowledgment for a task status update within a specific amount of time, it will repeatedly resend the status update to the master, which will again forward the update to the scheduler. Hence, task status updates will be delivered "at least once", assuming that the agent and the scheduler both remain available [5].

The information about the active tasks and registered frameworks is stored by the Mesos master in memory. Persistent storage is not used and it does not ensure that the information is preserved even after the master is failed. This feature helps the Mesos master to scale to large clusters with many tasks and frameworks [5]. If all the Mesos masters are unavailable or crashed, the cluster should continue to operate. The existing Mesos agents and user tasks should continue running but new tasks cannot be scheduled and frameworks will not receive resource offers or status updates about previously launched tasks. Mesos does not dictate how frameworks should be implemented and does not try to assume responsibility for how frameworks should deal with failures [5].

4.3. Designing Highly Available Frameworks

The patterns followed for designing highly available networks are as follows:

- Frameworks run multiple scheduler instances to tolerate scheduler failures. Only one of the scheduler instances is the leader at any particular given point in time. This instance is connected to the Mesos master, it receives resource offers and task status updates and launches new tasks. The other available scheduler replicas are followers. They are used only when the leader fails and in that case one of the followers is chosen to become the new leader via polling [5].

- The election of a new leader is done via a mechanism and schedulers need that mechanism when the scheduler has failed. This mechanism is typically provided using a coordination service. This coordination service can be Apache ZooKeeper [5].

- After election is conducted and a new leading scheduler is elected, the new leader should reconnect to the Mesos master. When the master is registered, the framework should set the id field in its FrameworkInfo to the ID that was assigned to the failed scheduler instance. This ensures that a new session is not started by the master and the master recognizes the connection. Thus, it will continue the session used by the failed scheduler instance.

- After connecting to the Mesos master, the newly elected leading scheduler should maintain the consistency of its local state with the current state of the cluster. If the previous leading scheduler attempted to launch a new task and then failed immediately, the task might have launched successfully and the newly elected leader will start to receive updates regarding it [5]. Frameworks typically use a consistent distributed data store to record information about active and pending tasks. The coordination service that is used to elect the master can be used for this purpose. Mesos replicated logs can also be used to achieve this. The data store is used to record the actions that the scheduler intends to take, before it takes them. If a scheduler intends to launch a task, it first writes this intent to its data store. It then sends a launch task message to the Mesos master. This helps in cases when the instance of the scheduler fails and a new scheduler is promoted to become the leader, the newly elected leader then consults the data store to find all the possible tasks that might be running on the cluster.

  The approach is called as write-ahead logging pattern. The key aspects of this approach can be summarized as one when the scheduler must persist its intent before launching task. If the task is launched and then the scheduler fails before it can write to the data store, the new scheduler instance is unaware of the new task. In this case, the new scheduler instance will begin receiving the task status updates for a task it has no knowledge of. Another aspect is that the scheduler should ensure that its intent has been durably recorded in the data store before it continues to launch the task [5].

5. LIFE CYCLE OF A TASK

A Mesos task transitions through a sequence of states. The agent is the actual source from which the state of a task can be determined. The agent on which the task is running provides information regarding the actual state of the task. The framework scheduler learns about the current state of the task by communicating with the Mesos master [4]. It does learn by listening for

task status updates and performing task state reconciliation. The task states are represented by the frameworks using a state machine. It can have one initial state and possibly several terminal states [5].

TASK_STAGING state is the first state of the task. The task begins in the TASK_STAGING state and is in this state when the framework sends the request to the master. The master receives the request to launch the task but the task has not yet started to run [5].

TASK_STARTING state is intended to be used by executor which are custom implemented. This state can be used to describe that a custom executor is aware of the task and might have started fetching its dependencies but it hasn't yet started [5].

TASK_RUNNING is the state when a task transitions to after it has begun running successfully. The framework should perform reconciliation when it attempts to launch a task but does not receive a status update within a particular timeout interval. For unknown tasks, the master will reply with the TASK_LOST status. The framework uses this to distinguish between the tasks that are slow to launch and the tasks that the master is unaware about [5].

TASK_KILLING state is an optional state which is intended to indicate that the request has been received by the executor but the corresponding task has not yet been killed. This might be useful for tasks to be killed smoothly. Executors should not generate this state unless the framework has the TASK_KILLING_STATE capability [5].

There are several terminal states such as TASK_FINISHED which depicts when a task is completed successfully, TASK_FAILED indicates that the task is aborted with an error, TASK_KILLED indicates that a task was killed by the executor, TASK_LOST depicts that the task was running on an agent that has lost contact with its master and TASK_ERROR indicates that a task is launched but its attempt to launch the task has failed because of an error in the task specification [5].

6. COMPARISON OF MESOS, DOCKER AND KUBERNETES

Mesos, Kubernetes and Docker fall into DevOps infrastructure management tools known as Container Orchestration Engines(COEs) [6]. An abstraction layer of protocols are provided by COEs between a pool of resources and the containers of application that run on these resources. COEs also solve the problem of taking multiple discrete resources in the cloud and combine them into a single pool. This pool can be used to deploy a variety of applications. These tools provide a set of features such as Container scheduling, High Availability, Health checks, Service discovery and Load balancing. Container scheduling comprises of performing functions which start and stop containers, distributing containers amongst pooled resources, failure recovery of containers, rebalancing containers from hosts that are failed to hosts that are healthy and scaling of applications [6]. High availability of application and container is provided by these tools. The health checks are used to determine the container and application health while service discovery is used to determine the services which are located on a network [6].

6.1. Kubernetes Container Orchestration Capailities

Kubernetes project originated to resolve the issue of running containers at a massive scale. Kubernetes uses YAML-based deployment model. It is used to manage the scheduling the containers on host machines. It also includes features such as built-in auto-scaling, secrets management, load balancing and volume management. It requires less third party software than Mesos and Swarm. It consists of a concept of pods which are groups of containers that are scheduled together to form a service. It doesn't support single node master installations with highly available clusters. The learning curve of Kubernetes is steeper than Swarm.

6.2. Swarm Container Orchestration Capabilities

Docker Swarm is a native Docker Container Orchestration Engine. Swarm is tightly integrated with the Docker API which makes it extremely suitable for its use with Docker. Swarm uses the same primitives as applicable to single host docker cluster. It simplifies managing container infrastructure as there is no need to configure a different orchestration engine. Swarm also uses a YAML-based deployment model using Docker Compose. Its main features includes auto-healing of clusters, overlay networks with DNS, and usage of multiple masters for high availability. Swarm does not support native auto-scaling as well as external load balancing. Swarm includes the capability of ingress load balancing but it requires a third party load balancer for external load balancing.

6.3. Mesos Container Orchestration Capabilities

Apache Mesos takes a more distributed approach to managing datacenter and cloud resources. Mesos uses Zookeeper to manage multiple masters and form a high-availability cluster. Other container management frameworks can be run on top of Mesos including Kubernetes, Apache Aurora, Chronos, and Mesosphere Marathon. The Mesosphere DC/OS, distributed datacenter operating system is based on Apache Mesos. Mesos takes a modular approach to managing containers which allow flexibility in the type of applications that can be run on Mesos and also the scale to which the applications can be run. It can scale upto tens of thousands of nodes and is used by Twitter, Airbnb, Yelp and eBay. The most important features are multiple types of container engines including Docker and its own Containerizer which comes along with a web UI, and its ability to run on multiple Oses including Linux, OS X and even Windows. Due to its complexity and flexibility, Mesos has a steeper learning curve than Docker Swarm.

7. ANSIBLE

Ansible is an open source automation engine which can be used to automate cloud provisioning, configuration management, and application deployment [7]. It is designed to be minimal in nature, secure, consistent, and highly reliable [8]. In many respects, it is unique from other management tools and aims to provide large productivity gains. It has an extremely low learning curve and seeks to solve major unsolved IT challenges under a single banner [7].

7.1. Architecture

One of the primary differences between Ansible and many other tools in the space is its architecture. Ansible is an agentless tool, it doesn't require any software to be installed on the remote machines to make them manageable. By default it manages remote machines over SSH or WinRM, which are natively present on those platforms [9].

Like many other configuration management softwares, Ansible distinguishes two types of servers: controlling machines

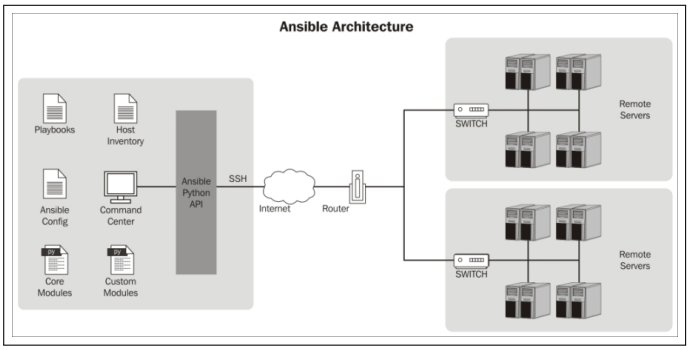


Fig. 3. Ansible Architecture [10].

and nodes. Ansible uses a single controlling machine where the orchestration begins. Nodes are managed by a controlling machine over SSH. The location of the nodes is described by the inventory of the controlling machine [8].

Modules are deployed by Ansible over SSH. These modules are temporarily stored in the nodes and communicate with the controlling machine through a JSON protocol over the standard output [9].

The design goals of Ansible includes consistency, high reliability, low learning curve, security and to be minimalistic in nature. The security comes from the fact that Ansible doesn't require users to deploy agents on nodes and manage remote machines using SSH or WinRM. If needed, Ansible can connect with LDAP, Kerberos, and other centralized authentication management systems [11].

7.2. Advanced Features

The Ansible Playbook language includes a variety of features which allow complex automation flow, this includes conditional execution of tasks, the ability to gather variables and information from the remote system, ability to spawn asynchronous long running actions, ability to operate in either a push or pull configuration, it also includes a check mode to test for pending changes without applying change, and the ability to tag certain plays and tasks so that only certain parts of configuration can be applied [8]. The features allow your applications and environments to be modeled simply and easily, in a logical framework that is easily understood. Ansible has low overhead and is much smaller when compared to other tools like Puppet [12].

7.3. Playbook and Roles

Playbook is what Ansible uses to perform automation and orchestration. They are Ansible's configuration, deployment, and orchestration language. They can be used to describe policy you need your remote systems to enforce, or a set of steps in a general IT process [13].

At a basic level, playbooks can be used to manage configurations and deployments to remote machines. While at an advanced level, they can be utilized to sequence multi-tier rollouts which involve rolling updates, and can also be used to delegate actions to other hosts, interacting with monitoring servers and load balancers at the same time.

Playbooks consist of series of plays, which are used to define automation across a set of hosts, known as the 'inventory'. These 'play' generally consists of multiple 'tasks', that can select one, many, or all of the hosts in the inventory where each task is a call to an Ansible module - a small piece of code for doing a specific task. The tasks may be complex, such as spinning up an entire cloud formation infrastructure in Amazon EC2 or Chameleon. Ansible includes hundreds of modules which help it perform a vast range of tasks [9].

Similar to many other languages Ansible supports encapsulating Playbook tasks into reusable units called 'roles'. These roles can be used to easily apply common configurations in different scenarios, such as having a common web server configuration role which can be used in development, test, and production automation. The Ansible Galaxy community site contains thousands of customizable rules that can be reused used to build Playbooks.

Playbook can be used to combine multiple tasks to achieve complex automation [13]. Playbook and Ansible can be easily used in implementing a cluster-wide rolling update that consists of consulting a configuration/settings repository for information about the involved servers, configuring the base OS on all machines and enforcing the desired state. It can also be used in signaling the monitoring system of an outage window prior to bringing the servers offline and signaling load balancers to take the application servers out of a load balanced pool [9].

7.4. Integration Of Cloud And Infrastructure

Ansible can easily deploy workloads to a variety of public and on-premise cloud environments. This capability includes cloud providers such as Amazon Web Services, Microsoft Azure, Rackspace, Cloudmesh, and Google Compute Engine, and local infrastructure such as VMware, OpenStack, and CloudStack. This includes not just compute provisioning, but storage and networks as well and the capability doesn't end here. As noted, further integrations are easy, and more are added to each new Ansible release. As Ansible is open source, anyone can make his/her contributions [9].

8. OPENCV

OpenCV stands for Open Source Computer Vision Library. It is an extensively used open source machine learning and computer vision software library. OpenCV was built with a motive to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being an open source library licensed under BSD-licensed product, OpenCV makes it easy for anyone to utilize and modify the code [14].

The library has a number of optimized algorithms, which includes a comprehensive set of both classic and advanced computer vision and machine learning algorithms. The algorithms can be used for a variety of purposes ranging from face detection, object identification, classification of human actions to tracking moving objects, extracting 3D models of objects. It has C++, C, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of single instruction, multiple data (SIMD) and Streaming SIMD Extensions instructions when available [14].

9. FACE DETECTION

Face detection is a specific case of object-class detection. In object-class detection, the task is to find the locations and sizes of all objects in an image that belong to a given class. Face-detection algorithms focus on the detection of frontal human faces. It is analogous to image detection in which the image of

a person is matched bit by bit. Image matches with the image stored in database. Any facial feature changes in the database will invalidate the matching process [15].

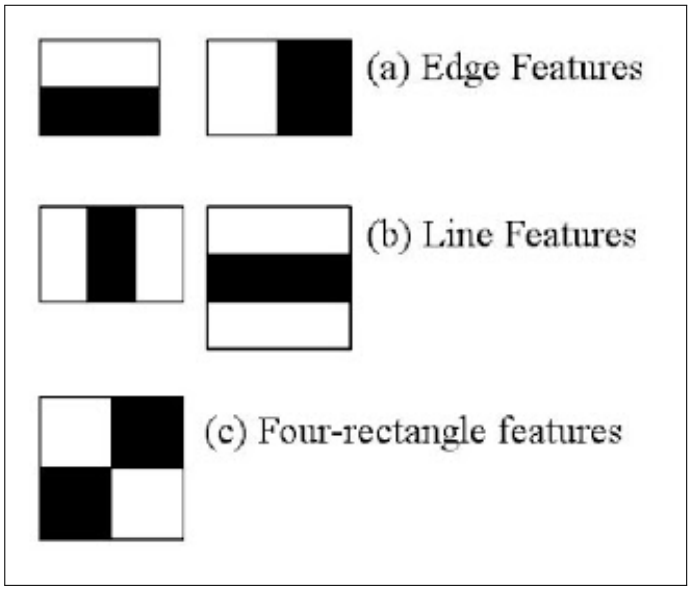


Fig. 4. Features [15].

A number of approaches are used but the most common one which our project utilized is a method where the possible human eye regions are detected by testing all the valley regions in the gray-level image. The basic idea is eyes (white) is separated by darker shades (skin color) above and below it. Then the genetic algorithm is used to generate all the possible face regions which include the eyebrows, the iris, the nostril and the mouth corners [15].

Each possible face candidates is normalized to reduce lightning effect caused due to uneven illumination and the shirring effect due to head movement. The fitness value of each candidate is measured based on its projection on the eigen-faces. After a number of iterations, all the face candidates with a high fitness value are selected for further verification. At this stage, the face symmetry is measured and the existence of the different facial features is verified for each face candidate.

We have utilized face detection Haar cascade provided by OpenCV. The Haar cascade is a classifier achieved using machine learning algorithm known as AdaBoost [16]. Initially, the AdaBoost algorithm required a lot of positive images (images of faces) and negative images (images without faces) to train the face detection classifier. Then the features are extracted from it. Each feature is a single value obtained by subtracting sum of pixels under imaginary white rectangle from sum of pixels under black rectangle [15].

All possible sizes and locations of each kernel are used to calculate plenty of features. For each feature calculation, we need to find sum of pixels under white and black rectangles. Among all the features generated, most of them are irrelevant. To select best features, they are applied to each and every feature on all the training images. For each feature, it finds the best threshold which will classify the faces to positive and negative. There will be errors or misclassifications but when results from all the features are combined the results are excellent. Out of all the features, the features with the minimum error rate, are the features that best classifies the face and non-face images.

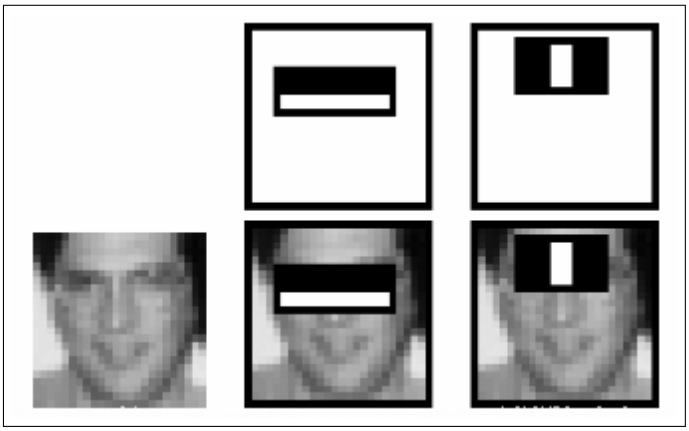


Fig. 5. Basic idea behind features [15].

Final classifier is a weighted sum of these weak classifiers/features. It is considered as weak classifier because it can't classify the image alone, but together with other classifiers, forms a strong classifier. The combined results obtained from all the features help to identify if a face is present in the image, and if it is then the classifier also tells the user about the dimensions of the rectangle and the starting x, y coordinate of that rectange on the image [16].

10. IMPLEMENTATION

In our implementation, we used Ansible version 2.2.1.0, Mesos version 1.2.0, Python version 2.7.12 and OpenCV version 3.2. The deployment was done on chameleon cloud. Firstly, cloudmesh client is set up on the local machine and ansible is installed. A new security is created and the ports 5050 and 5051 are exposed. The default security group was also included which exposed ports 22, 80 and 443. Port 22 is used for accessing the remote machine. The virtual hardware template(flavor) used is m1.medium and the image used was CC-Ubuntu16.04-20161214. A virtual hardware template defines the size of RAM, disk size, and number of cores. The flavor m1.medium specifies 2 VCPU's used, 40 GB root disk, 4096 MB RAM. The cluster was setup to have 3 virtual machines(nodes). Amongst the 3 virtual machines, one was used as Mesos master and the remaining two were used as Mesos agents(slaves). The profile details for cloudmesh client have been set up. The cloudmesh client setup was validated. The system has been deployed using Ansible roles. For the deployment, the system has been divided into multiple roles such as inventory role(r_inventory) , role for create swap file(r_swap), mesos(r_mesos) role, master(r_master) role, agent(r_agent) role, role for copying framework(r_copy_framework) and role for running framework(r_run_framework).

11. ROLES

The functionality of the Ansible roles are explained as below:

11.1. Inventory

The role r_inventory was used for inventory setup on the local machine. Firstly, the operating cluster was selected and its node details were obtained. These nodes were added to the known hosts file under the ssh directory. The inventory was created selecting two nodes as Mesos agents and one as Mesos master and were tagged accordingly in the inventory file.

11.2. Create Swap file

Swap space in Linux is used when the amount of physical memory is full. If the system needs more memory resources and the RAM is full, inactive pages in memory are moved to the swap space. Initially, the built was unsuccessful when attempts were made with RAM size of 2GB, 4GB and 8GB. It was observed, the virtual machines created had no swap space defined. The swap size was increased to 2GB after which Mesos was successfully installed. All these observations led to the conclusion that for successfully installing Mesos on a virtual machine minimum of 2GB swap space is required. The playbook r_create_swap successfully creates a swap file of 5GB on all the remote machines.

11.3. Mesos

The Mesos package is downloaded from the Mesos website. The latest version of Apache Mesos is 1.2.0 and it is uncompressed on the virtual machines. All the existing softwares on the virtual machines are updated. The dependencies required for Mesos 16.04 are openjdk-8-jdk, python-dev, build-essential, python-virtualenv, libcurl4-nss-dev, libsasl2-dev, libsasl2-modules, maven, libapr1-dev, libsvn-dev, zlib1g-dev, git. These packages are specific to different distribution and versions of OSes. Now, the dependencies required for face detection framework are installed. The dependencies include python-pip, numpy and opencv-python to support face detection framework. In Mesos, applications are deployed as frameworks. Mesos is configured using the configure command. In order to install Mesos on the virtual machines, the make and make check commands are executed.

11.4. Mesos Master

The Mesos master is run on the host which was selected as the Mesos master in the inventory file. The working directory for Mesos is selected. The IP address of this virtual machine is advertised using the tag –advertise_ip. The Mesos master runs on default port 5050. The Mesos master can be validated by entering the IP address followed by the port number of the Mesos master in a Web browser on a local machine.

11.5. Mesos Agent

The Mesos agent is run on the hosts which were selected as the Mesos agents in the inventory file. The working directory for Mesos is selected. The IP address of the Mesos agents are advertised and the IP address of the master to which the agents will be connected is specified. In the inventory role, the IP address of the master node was saved in a separate file which is copied to the remote agents in this role and the IP address of the master will be used by the agents to connect to the master node. The agent runs on the default port 5051.

11.6. Copy and Run framework

The face detection framework files are copied from the local machine to all the remote virtual machines including both the Mesos master node and Master agent nodes and are placed in their respective directories. The face detection framework is then run on the master node.

12. FRAMEWORK IMPLEMENTATION

The test framework provided by mesos have been modified to run face detection application. The framework initializes all the requirements through face_detection_framework file after which face_detection_executor runs. The executor creates threads which then run on agents. The executor is designed such that it can be easily modified to support another python application by just changing the call to the main function in run_task.

The variable dataset and database_url are the only variables that need to be modified to change the dataset from which the images need to extracted and head counts need to be taken. Here, the dataset variable tells the dataset name and the dataset_url tells the location from where the zip folder can be downloaded. The program downloads and unzips the folder to home directory of the user and this default path can be changed by changing the global_path variable. After unzipping the files, the program removes the zip folder as it is no longer required.

Once the dataset is extracted, the file list is given to the face detection program implemented using OpenCV library. The OpenCV library uses cascade files, where the cascade files are simply classifiers which can identify faces in an image or dogs in an image or any other object depending on the cascade file used. For our application, face detection cascade has been used and the link to cascade file is given in cascase_url. One can easily change this cascade file and change the way this program functions, i.e. one can just provide the hyperlink in the variable cascade_url depending on the cascade file user wants to use and the function will identify dogs, cats, eyes or any other object which the cascade file is trained to identify. The user can find these cascade files online, easily or could train and create one if required.

The program identifies faces, marks them in a rectangle and saves the output in the output folder. The number of images has been restricted to 50 for testing purposes but this can easily be changed by updating the no_files and images_per_file variable as per the requirement. After the results have been obtained the dataset folder is deleted. The results obtained contains images with faces marked in blue square and the count is maintained in results.txt file. These results are saved on the agent machines which is then copied to local machine using fetch command automatically.

Whenever new tests are run, the results are appended to results.txt file in place of just replacing the results. This has been done as a user might decide to run the application for different dataset or for different images while keeping the head counts for older datasets.

13. BENCHMARK

The benchmarks are shown in Table 1 which compare the time to run the make command and the make check on different VMs. The time required to install, setup a cluster and install prerequisites has been excluded as they are negligible and on average take 5 minutes on each VM and just vary in few seconds on average.

The results shown in table 2 were run on VM's running Ubuntu 16.04 with flavor as medium and swap size of 5 GB. The results were taken for 1 image to see how much time initialization takes. The results show that there is significant improvement when number of agents gets increased.

14. CONCLUSION

The head count detection framework is running successfully on Apache Mesos and the output is generated on the local machine which gives the source of the image and the number of head counts detected in the image along with it. Apache Mesos

Table 1. Benchmarks

Operating System	Flavor	Virtual Size Allocated	RAM CPU Cores (in GB)	Swap Size (in GB)	HDD Size (in GB)	Time to run make (in hh:mm:ss)	Time to run make install (in hh:mm:ss)
Ubuntu 14.04	small	1	2	2	20	00:43:56	00:48:07
Ubuntu 14.04	medium	2	4	5	40	00:35:23	00:42:32
Ubuntu 14.04	large	2	8	5	80	00:33:27	00:37:21
Ubuntu 16.04	small	1	2	2	20	00:40:32	00:48:32
Ubuntu 16.04	medium	2	4	2	40	00:37:56	00:46:68
Ubuntu 16.04	medium	2	4	5	40	00:19:31	00:32:16
Ubuntu 16.04	medium	2	4	7	40	00:21:47	00:29:43
Ubuntu 16.04	large	2	8	5	80	00:20:31	00:31:23

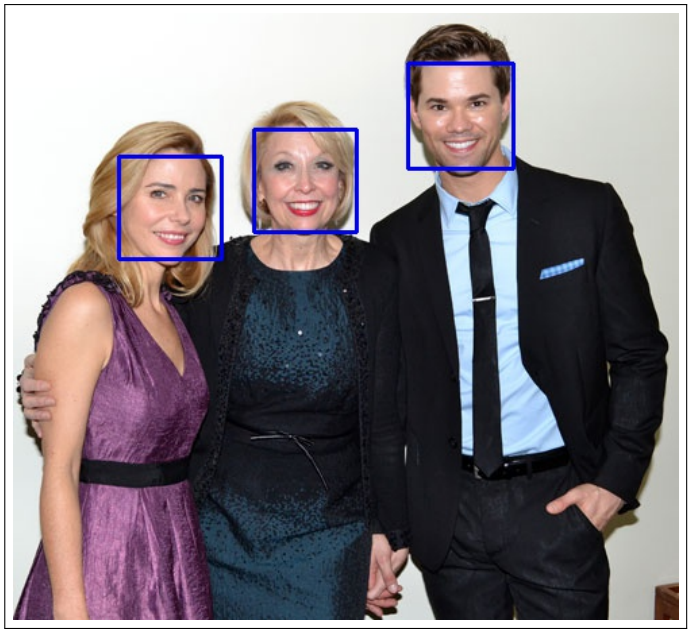


Fig. 6. Sample Output Image.

Fig. 7. Sample Result.

Table 2. Application Benchmarks

Number Of Images	Run Time with 1 Agent (in hh:mm:ss)	Run Time with 2 Agents (in hh:mm:ss)
1	00:00:28	00:00:45
50	00:01:21	00:01:15
100	00:02:12	00:01:52
500	00:05:08	00:03:25
1000	00:10:21	00:07:32

was successfully deployed on Linux 16.04 and 14.04 systems and the time required to install and deploy Mesos was successfully benchmarked. Implementation of the head count detection framework was successfully tested on Apache Mesos with a cluster of 1 master with 1 agent and 1 master with 2 agents. Ansible was used to successfully deploy Mesos on chameleon cloud and it is seen that Apache Mesos handles the containerization of the framework with ease. The future scope would include deployment of the headcount detection framework over 1,000 virtual machines and test the scalability of Apache Mesos. Along with scalability, the reliability can also be tested by running multiple masters and checking the fault tolerance of Apache Mesos.

REFERENCES

[1] Apache Software Foundation, "Apache mesos," Web Page, Mar. 2014, accessed 2017-04-01. [Online]. Available: http://mesos.apache.org/

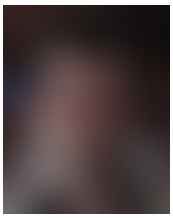
[2] Wikipedia, "Face detection," Web Page, Feb. 2017, online; accessed 09-March-2017. [Online]. Available: https://en.wikipedia.org/wiki/Face_detection

[3] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, Ion Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in *8th USENIX Symposium on Networked Systems Design and Implementation*, 2010. [Online]. Available: http://mesos.berkeley.edu/mesos_tech_report.pdf

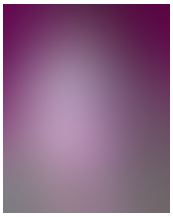
[4] Apache Software Foundation, "Apache mesos-architechture," Web Page, Mar. 2014, accessed 2017-05-20. [Online]. Available: http://mesos.apache.org/documentation/latest/architecture/

[5] Apache Software Foundation, "Apache mesos-designing highly available mesos frameworks," Web Page, Mar. 2014, accessed 2017-05-20. [Online]. Available: http://mesos.apache.org/documentation/latest/high-availability-framework-guide/
[6] Sumo Logic, "Kubernetes vs mesos vs swarm - sumo logic," Web Page, 2017, accessed 2017-04-01. [Online]. Available: https://www.sumologic.com/devops/kubernetes-vs-mesos-vs-swarm/
[7] Wikipedia, "Ansible (software)-wikipedia," Web Page, Feb. 2017, accessed: 2017-02-20. [Online]. Available: https://en.wikipedia.org/wiki/Ansible_(software)
[8] Red Hat Inc., "Ansible in depth," Web Page, Feb. 2016, accessed: 2017-02-23. [Online]. Available: https://www.ansible.com/ansible-in-depth-whitepaper
[9] Red Hat Inc., "How ansible works," Web Page, Feb. 2016, accessed: 2017-02-14. [Online]. Available: https://www.ansible.com/how-ansible-works
[10] Madhurranjan Mohaan, Ramesh Raithatha, "The ansible architecture-learing ansible," in *Learning Ansible*. Packt Publishing Limited, 2011. [Online]. Available: https://www.packtpub.com/mapt/book/Networking+and+Servers/9781783550630/1/ch01lvl1sec09/The+Ansible+architecture
[11] Red Hat Inc., "Ansible documentation," Web Page, Feb. 2016, accessed: 2017-02-15. [Online]. Available: https://docs.ansible.com/ansible/index.html
[12] UpGuard Inc., "Ansible vs puppet," Web Page, Feb. 2017, accessed: 2017-02-24. [Online]. Available: https://www.upguard.com/articles/ansible-puppet
[13] Red Hat Inc., "Playbooks," Web Page, Feb. 2016, accessed: 2017-02-24. [Online]. Available: https://docs.ansible.com/ansible/playbooks.html
[14] OpenCV Team, "OpenCV," Web Page, Feb. 2017, online; accessed 09-March-2017. [Online]. Available: http://opencv.org/
[15] OpenCV Dev Team, "Face detection using haar cascades," Web Page, Feb. 2014, accessed 2017-03-11. [Online]. Available: http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html#face-detection
[16] Wikipedia, "Adaboost-wikipedia," Web Page, Apr. 2017, accessed: 2017-04-23. [Online]. Available: https://en.wikipedia.org/wiki/Ansible_(software)

AUTHOR BIOGRAPHIES



Anurag Kumar Jain will receive his Masters (Computer Science) in 2018 from The Indiana Univeristy Bloomington. His research interests include Artificial Intelligence and Big Data.



Pratik Sushil Jain will receive his Masters (Computer Science) in 2018 from Indiana University Bloomington. His research interests include Big Data and Machine Learning.

Ronak Parekh will receive his Masters (Computer Science) in 2018 from Indiana University Bloomington. His research interests include Big Data, Cloud Computing and Machine Learning.

Optical Character Recognition

SABER SHEYBANI[1] AND SUSHMITA SIVAPRASAD[1]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
* Corresponding authors: sheybani@umail.iu.edu,sushsiva@umail.iu.edu

project-000, May 4, 2017

Optical Character Recognition is a technology for converting images of texts, into machine encoded text format. In this project, the input data is in standard image formats and our goal is to recognize the words/letters in the image as accurately as possible and convert the dataset into TXT format. The heart of OCR as a pattern recognition system is a classifier. The images will go through a pre-processing phase which will convert them into a form that is compatible for feeding to the classifier. Then the classifier will determine the class of each glyph in the image. The whole recognition system is installed and operated on the cloud.

Keywords: OCR, Ansible, K-Nearest Neighbor

Report: https://github.com/cloudmesh/sp17-i524/blob/master/project/S17-IR-P012/report/report.pdf
Code: https://github.com/cloudmesh/cloudmesh.ocr/tree/master/code

1. INTRODUCTION

The age of digitalization has made it quintessential to store documents in a digital form for the purpose of allowing valuable information to be stored, edited and searchable in the middle of billions of records. OCR technology is used quite frequently by every industry that has handwritten, scanned or photographic data. This project proposal gives a detailed view on how the OCR technology works, the preprocessing done to remove the noise and the algorithms used to recognize the images. We have delved deep into some of the basic concepts used in the implementation process. We have also discussed some important applications of this technology in the real world and created a benchmark.

2. EXECUTION PLAN

This execution plan is to show the distribution of work over the time period of the course. The steps followed to achieve the final results.

1. **6 March - 12 March 2017** To implement the OCR, we first looked for a dataset to train the algorithm and test it. The desired library to use to write the codes and the complexity of the final goal of the project was discussed.

2. **13 March -19 March 2017** Looking for preprocessing steps in order to cleanse noise from the image and convert the image into a standard form which makes it adequate for later processing.

3. **20 March - 26 March 2017** Starting to work with Ansible and Cloudmesh Client for running tasks on virtual nodes.

4. **27 March- 02 April 2017** starting to deploy individual modules for preprocessing, to the virtual machines on Chameleon Cloud service, using Ansible. Cloudmesh Client was used to reserve the virtual machines on the cloud services.

5. **03 April- 09 April 2017** Executing a preliminary form of character recognition using K Nearest Neighbour classification and a large dataset.

6. **10 April - 16 April 2017** Implementing segmentation of image into lines, words and letters.

7. **17 April - 23 April 2017** Executing benchmark and finalizing the project report.

3. BACKGROUND

3.1. OCR Technology

Optical Character Recognition is a technology which is used to convert different types of documents that can be in the form of scanned documents, including handwritten or machine-written into an editable and searchable form [1]. The images can be in either the basic black & white or colored. The technology analyzes the structure of the document and divides it into small refined segments, so that each segment contains one character. Finally, individual characters are singled out one by one and fed to a classification algorithm which will return the closest letter that the individual character could possibly be identified with.

4. SYSTEM CONFIGURATION AND TECHNOLOGIES USED

4.1. System Configuration

The python codes and the libraries are run on a Ubuntu 16.04 LTS.

4.2. Technologies Used

1. **Python Programming Language** : An object oriented programming language with an open-source license. We chose Python for this project as there are many libraries available in Python that helps in creating a code for OCR easier. Packages such as PIL, Pillow and OpenCV are one of the few libaries that helps in image processing.

2. **OpenCV** : OpenCV is an open source computer vision library. It is used for building computer vision applications. It consists of more than 2500 algorithms including both machine learning and computer vision algorithms [2]. These algorithms are devised for facial and image recognition, track any moving objects etc. We have used OpenCV 2 for creating a KNN classifier and a few of our preprocessing techniques.

3. **Ansible** : Ansible is an IT automation tool. It uses YAML in order to issue the state of the server [1]. Ansible implements the internal command that is required to reach that state which depends on the operating system. The ansible playbook which consists of these internal commands can be applied to any server or service. There is no requirement to instal an additional software on the target system as the commands are run over an SSH session.

4. **Cloudmesh Client** : It is an open source client interface tool that provides us with an easy-to-use interface for accessing cloud services, creating single and multiple VMs, clusters and workstations.We can manage the resources we would like to use and customize them as per our requirement to run the projects.
   It provides an interface to execute jobs on High Performance Computing clusters [3].The users can use just one platform to manage all of the cloud resources. Cloudmesh client creates a local copy of the data which results in clouds with similar configurations to be created as well. The default features of the Cloudmesh Client allows easy control of the cloud as well. Cloudmesh includes an API, commandline client and a commandline shell.

5. **Chameleon Cloud** :Chameleon cloud provides OpenStack Cloud (kilo) using the KVM virtualization technology [4]. It is an Infrastructure as a Service(IaaS) platform that allows us to create as well as manage the virtual environment. The virtual machine that we use here is compatible with KVM. Chameleon Cloud also gives access to the bare-metal computing resources, which allows administrative rights to use cloud for computing experiments [5].

5. METHOD SURVEY

Optical Character Recognition have already been developed in numerous ways, focusing on different goals. We did a survey on the possible approaches for character recognition.

The main components of every OCR system can be enumerated as feature extraction component, and the classifier.

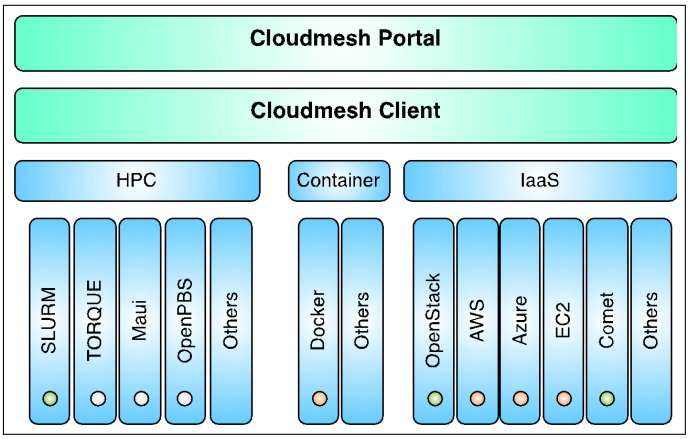


Fig. 1. Architecture of Cloudmesh Client [3]

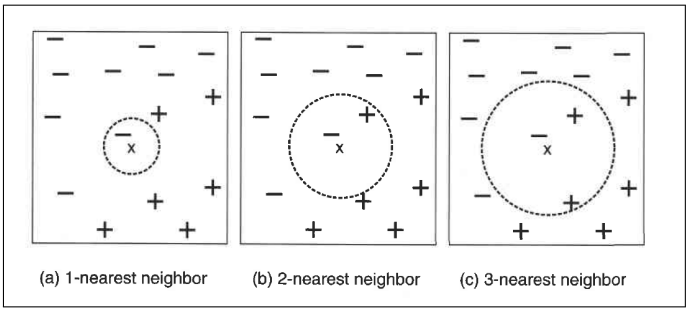


Fig. 2. Example of k-NN classification [7]

Feature extraction methods can be separated into two groups: Template matching, and Structural classification [6]. In **Template matching**, the individual pixels of the image are directly used as features. For each possible character, there is one class and one template feature set, associated with it. In classification, a similarity metric will evaluate the distance of the input character to each of the templates. And thus, the input will be represented with the class of the most similar template.

In **structural classification** structural features of every character, such as loops and curves, are used as features.

There are various types of classification methods. A number of methods that have been used for OCR are Nearest-Neighbor, Artificial Neural Networks, and Support Vector Machines. K-Nearest Neighbor and Artificial Neural Networks are discussed further as the candidates for our job.

5.1. K Nearest Neighbour

It is a non parametric algorithm where each of the training data is considered to have a set of vectors and a class label associated with each vectors. The training set will have the labels for the classes , given the test set it calculates the distance between the training set and the test set and calculate the nearest points. A single value of 'K' is given which allows us to decide how many neighbours influence the classification. Figure 2 displays a schema of KNN classification.

5.2. Feed Forward Neural Networks

Artificial Neural Network is a paradigm in computing, inspired by the structure of biological nervous systems. It consists of a network of processing units, where the output of each unit is a nonlinear function of its weighted inputs that come from

other units. Such network can be trained to solve different kinds of problems, including classification and clustering. A feed forward neural networks is one in which the neurons are organized in a number of layers and each layer only feeds to the next one, but not to the previous one (no feedback). However, in a back-propagation process, the errors from one iteration of classification will be fed back from the output to the network, in order to modify and improve the network for next iterations.

Due to simplicity and relatively good performance of KNN, we choose it as the classifier for this project. There is an easy to use implementation of it in OpenCV library. Our KNN uses euclidean distance as the distance metric to calculate the nearest neighbours for the 'K' value.

Euclidean Distance

$d(p,q) = \sqrt{\sigma(p_i - q_i)^2}$

For the value of K, 5 is chosen so that it is neither too small and sensitive to the noise nor too large which might result in including the points from other classes.

6. PREPROCESSING TECHNIQUES

The steps in an OCR full session are as follows: Preprocessing: The input images need to be segmented into units that each of them keep only one glyph (symbol). Also, the colored or grayscale images will be binarized. Feature extraction: The glyphs will be decomposed into features like lines, closed loops, line direction, and line intersections. Character recognition: The image features will be fed to the classifier and they will be compared with stored glyph features and the nearest match will be chosen.

Preprocessing is required on the raw images that we are using to filter out the required subject and distinguish from any other unwanted objects from the image such as watermarks, background subjects etc. We have conducted different preprocessing techniques in order to remove noise and convert the image into a grey scale format as color images requires more complex methods of processing

6.1. Noise Reduction Techniques

Noise reduction is done for extracting out any unwanted bit-pattern, there are linear as well as non-linear techniques for this. Linear : In this method is used to remove any isolated pixel noise from the image. Here the required output filter is taken as a linear combination of the neighborhood pixels Non- Linear : These kind of filters are used to replace the value of a particular pixel in order to remove any kind of impulse noise

6.2. Histogram Based Method

It gives a value to the intensity of the pixel and plot it on a histogram , where darker the image , more the data points would be on the left and center of the histogram . Lighter the image , more the data points would be on the right side of the histogram. Using a histogram equalization method the contrast on the image can be improved in this case. In the histogram equalization method , an image is divided into blocks of pixels and an histogram equalization is done. This allows us to distinguish the images we actually require from the other background images . It allows us to enhance the visibility of the characters' present on the image.

6.3. Median Filter

It is a non-linear noise reduction technique , it is a low pass filter. In this case the pixel values are taken for an area on the image and an average of the pixel value is taken and assigned to the center pixel in that area. Figure 3 displays a schema of how median filter works. It is an effective means for removing the salt and pepper noise which are random lines occurring on the image due to poor quality of the picture or if the image wasn't scanned well [8]. Figure 4 shows the result of applying a median filter on a scanned image, we can see the reduction in dots and other marks on the image, making it more smooth and usable.

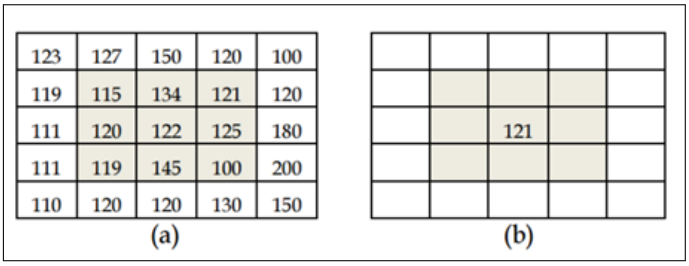


Fig. 3. Averaging of a pixel in median filter [8]

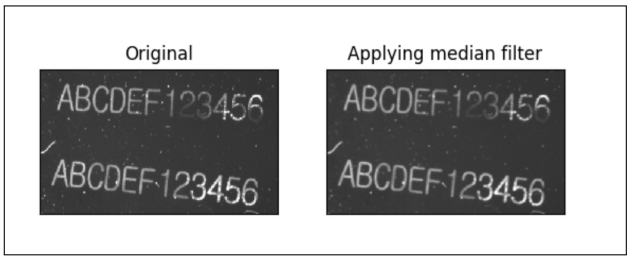


Fig. 4. Result of applying median filter on a scanned image

6.4. Gaussian Blur

Gaussian Blur filtwe is a low pass filter which is used to eliminate isolated pixel noise. Image smoothing is done here using gaussian filters where the weighted average of the pixel values is computed with the gaussian coefficients as weight. The filter provides a smooth texture to the noisy image.

Fig. 5. After applying gaussian filter on the image preproccesed with median filter

6.5. Binarization

Otsu's method [9]: Otsu's method concludes finding the best intensity threshold to separate two classes, often background vs foreground but not always. The algorithm tries to find a separation point that has the minimum weighted within class variance. If the input images are grayscale, the algorithm will simply find a threshold that any intensity below that will be considered as the background and the intensity of the corresponding pixels will be rounded to zero. Similarly, the intensities above the threshold will be rounded to 1. The resulting image (array) will be binary.

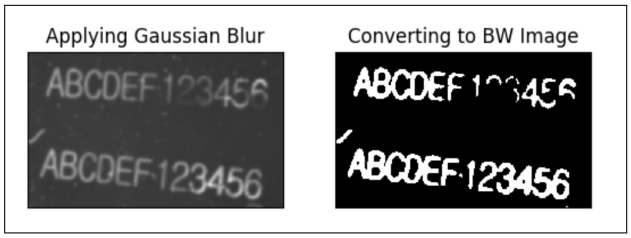


Fig. 6. Applying binarization after applying gaussian filter

6.6. Segmentation

Segmentaion of the image happens in multiple levels, namely lines, words and letters. Application of all of these three will enable the conversion of whole pages of scanned documents into text format. In this project, segmentation at all levels is implemented using projection profile approach. Projection of the intensities on the vertical axis will differentiate the rows that contain some text, from the ones that only include the background. For word segmentation if the number of background columns are more than a threshold, the two letters will be considered as incorporating different words. The background columns are detected using projection on the horizontal axis, but only for the rows that are included in the current line. Thresholds are defined using expert views in typography [10].
The letter segmentation is simply done after one background column is found. The figure 7 displays the application of our segmentation algorithm on a sample image.

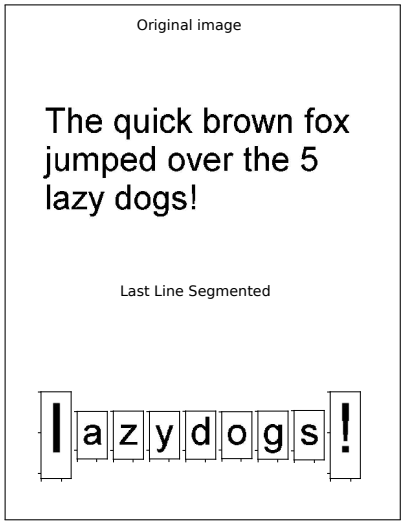


Fig. 7. Applying segmentation

7. APPLICATION

OCR converts images to machine-readable text. That will make it the initial tool that needs to be used for processing any documents or simply any written material in a digital image, which has been captured by a camera [11]. It's output can be stored significantly more compact than scanned images. But beyond that, it enables us to process the output information for numerous applications. Examples of these applications include creating a narrator machine to help the visually impaired read nondigital documents and signs, or automatic recognition of automobile number plates.

8. LICENSE

The project is developed under the open source Apache License 2.0. The license file is included in the Git repository of the project. The packages and softwares used in developing the project include OpenCV, Python 2.7, Ansible, Cloudmesh. All of these packages are open-source. The license file for OpenCV is included in the project repository (CV_LICENSE.txt).

9. CLOUD DEPLOYMENT

Automatic deployment of program on virtual environment was done using Ansible [12] .The jobs were collected, organized in Playbooks [13] and run on virtual clusters provided by Chameleon Cloud [14]. The tasks include Installing the essential libraries on the remote machine and running the program. The VM reservations were done using Cloudmesh Client. Three Ansible playbooks are used, one for deploying the software stack, one for running the OCR codes on a standard dataset and another one for running it on an arbitrary image. The software stack required for running the program includes the following components:

1. **Git** The Git module is needed for cloning the OCR Python codes. It is installed using apt module.
2. **OpenCV** OpenCV, as discussed in the previous sections, has become the standard library of computer vision. It is used in this project for multiple purposes. As this library is extensive, its installation may take a significant time. Hence, we tried to examine all the possible ways to install it. Building from the source, which is the recommended way for installing it (and any other package) took more than 20 minutes. Even after that, connecting it to the existing Python installation on the VMs was problematic. The alternative ways include using PyPi, Conda (Anaconda package management system), and apt. After trial and error, Apt appeared to be the best solution with installation time of less than 5 minutes and hassle-free python integration. It can be easily installed on VMs using Ansible apt module. So we chose this method.
3. **OCR Code** The codes of the program were checked in the project repository. An Ansible role cloned these codes into each node of the cluster.
4. **Dataset** An Ansible role was used to download and extract the dataset. Unarchive module was used for this purpose.

10. DATASET

The dataset used in our project for training the KNN is Chars74k [15]. This dataset provides an extensive collection of English letters and digits, in handwritten form or in various computer fonts

[16]. In this project, a subset of the data, including characters from computer fonts with 4 variations (combinations of italic, bold and normal) was used. The glyphs in this subset include 0-9, A-Z, and a-z. For each glyph, there are 1016 different files, each with one font. However, some of the fonts are similar to each other.
Before this dataset, the MNIST handwritten digits database were used for preliminary purposes [17].

11. BENCHMARKING

After wrapping all the necessary material, the code was run in a role and the result was saved in text files, using another role. The accuracy of classification is printed in the console of the local machine. The program was first deployed on single VMs that were reserved manually using Cloudmesh Client. Later on, clusters of 2-4 nodes were used to measure the execution time of deployment. Note that for running the benchmark, there is a trade-off between the runtime and accuracy of classification. The more samples are used for training the dataset, the higher the accuracy will be achieved and of course the longer execution time is needed.

11.1. Sample size analysis

Running a dataset of 50 images as the test and train data with K=5, takes around 20 seconds and yieds an accuracy of 65%. The time complexity of the algorithm is O(n2), thus, using 100 samples will multiply the runtime by 4. However, as the dataset used contains 74k images, there is still significant diversity among 100 images. As a result, using 100 images only improves the classification accuracy up to 75%. Changing the number of nodes does not affect the runtime, unless paralellization is exploited.

Each VM on Chameleon Cloud has only 1 core, 2050076 kB (2 GB) of RAM, and 20 GB of storage. Due to small size of RAM, we had to keep the number of training samples under 200.

11.2. Cluster size analysis

In this section, the sample size is around 50 different fonts for training the classifier and another 50 for testing it.

For the cluster of **2 nodes**: It took 67 seconds to deploy (allocate) the cluster, 153 seconds to install the software stack and dataset on the nodes, and 169 seconds to run the benchmark on them. The accuracy of classification on one of the nodes was 72.23 and on the other, it was 70.84.

For the cluster of **3 nodes**: It took 98 seconds to deploy (allocate) the cluster, 170 seconds to install the stack and 180 seconds to run the benchmark.

For the cluster of **4 nodes**: It took 131 seconds to deploy (allocate) the cluster, 175 seconds to install the stack and 187 seconds to run the benchmark. As we can see, the higher number of nodes will only add to the time of cluster creation. As the nodes are quite similar, the execution time of the tasks which run in parallel is similar.

11.3. OCR on arbitrary images

As the arbitrary images can be significantly different from the standard dataset, the sample size needs to be much higher. However, due to small size of RAM, for running OCR on arbitrary images, we only used a sample size of 200. That causes the accuracy of OCR to be very low, around 20%.

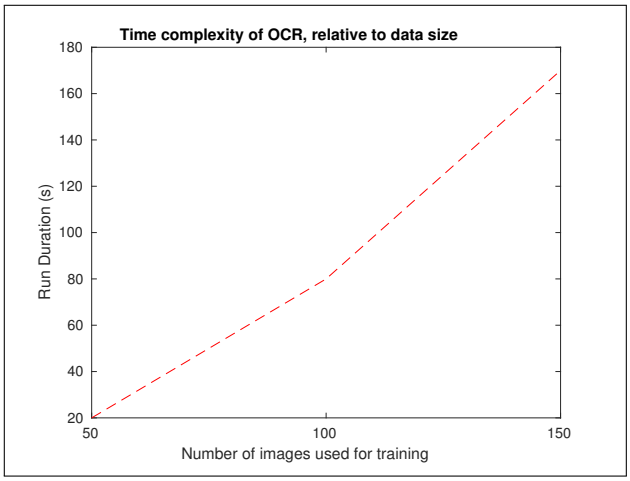


Fig. 8. Time complexity of running OCR on Chameleon Cloud

12. REPRODUCIBILITY

Instructions on reproduction of the project are provided in the project repository [18].

13. FUTURE WORK

For improving the current OCR system, various kinds of sophisticated structural features can be added to feature extraction. It can also be extended to operate on languages other than English. In order to improve the performance, the power of multiple cores/VMs can be used for parallelization, so that after segmentation, each VM will operate on one line of the text. Thus, multiple lines can be processed at the same time. For improving the recognition accuracy, a lexicon can be used to evaluate the validity of the words that have been formed by OCR, and then possibly correct them to the closest word in the lexicon.

14. ACKNOWLEDGEMENT

A very special thanks to Professor Gregor von Laszewski and the teaching assistants Miao Zhang and Dimitar Nikolov for all the support and guidance. This project proposal is written during the spring 2017 semester course I524: Big Data and Open Source Software Projects at Indiana University Bloomington.

AUTHOR BIOGRAPHIES

Saber Sheybani received his B.S. (Electrical Engineering - Minor in Control Engineering) from University of Tehran. He is currently a PhD student of Intelligent Systems Engineering - Neuroengineering at Indiana University Bloomington.

Sushmita Sivaprasad is a graduate student in Data Science at Indiana University under the department of Informatics and Computing. She had completed her bachelors in Electronics and Communication from SRM University , India and her master's in International Business from Hult International Business School, UAE.

15. WORK BREAKDOWN

The following was the work distribution followed for the project,

- **Sushmita Sivaprasad** : Researched on the background of implementing the ocr technology, briefed on the system configurations and the technologies that can be used. She implemented the preprocessing steps to remove the noise and inaccuracies in the image file before implementing the K means algorithm. Contributed to the writing of the report.
- **Saber Sheybani** : Implemented the cloud deployment on chameleon cloud and has done the benchmarking of the on the chameleon cloud. Contributed to the writing of the report.

REFERENCES

[1] "What is ocr and ocr technology," Web Page, 2017. [Online]. Available: https://www.abbyy.com/en-us/finereader/what-is-ocr/

[2] "About opencv," Web Page, accessed: 2017-4-6. [Online]. Available: http://opencv.org/about.html

[3] G. V. Laszewski, "Cloudmesh, an introduction," presentation, p. 31, 2015. [Online]. Available: https://drive.google.com/file/d/0Bx_sUfI4VkKSVG9KOE8xU05KREE/view

[4] "Openstack kvm user guide," Web Page. [Online]. Available: https://www.chameleoncloud.org/docs/user-guides/openstack-kvm-user-guide/

[5] "Bare metal user guide," Web Page. [Online]. Available: https://chameleoncloud.org/docs/bare-metal-user-guide/

[6] E. Borovikov, "A survey of modern optical character recognition techniques," *arXiv preprint arXiv:1412.4183*, 2014.

[7] V. K. Pang Ning Tan, Michael Steinbach, *Introduction to Data Mining*. Pearson Education, 2006.

[8] Y. Alginahi, "Preprocessing techniques in character recognition," pp. 9–10. [Online]. Available: http://cdn.intechopen.com/pdfs/11405.pdf

[9] N. Otsu, "A threshold selection method from gray-level histograms," *Automatica*, vol. 11, no. 285-296, pp. 23–27, 1975.

[10] G. Dowding, *Finer points in the spacing and arrangement of type*. Hartley & Marks Publishers, 1995.

[11] "Optical Character Recognition," Web Page, Mar. 2017. [Online]. Available: https://en.wikipedia.org/wiki/Optical_character_recognition

[12] "Ansible," Web Page. [Online]. Available: https://www.ansible.com/

[13] "Playbook," Web Page, Mar. 2017. [Online]. Available: http://docs.ansible.com/ansible/playbooks.html

[14] "A configurable experimental environment for large-scale cloud research," Web Page, Jan. 2017. [Online]. Available: https://www.chameleoncloud.org/

[15] "Character recognition in natural images," Web Page. [Online]. Available: http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/

[16] T. E. de Campos, B. R. Babu, and M. Varma, "Character recognition in natural images." in *VISAPP (2)*, 2009, pp. 273–280.

[17] C. J. B. Yann LeCun, Corinna Cortes, "The mnist database of handwritten digits," Web Page. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[18] "Readme," Web Page. [Online]. Available: https://github.com/cloudmesh/cloudmesh.ocr/tree/master/code

Weather Data Analysis

VISHWANATH KODRE[1], SABYASACHI ROY CHOUDHURY[1], AND ABHIJIT THAKRE[1]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[1] Corresponding authors: sabyasachi087@gmail.com, vkodre@gmail.com, athakre@gmail.com

project-fillmeout, May 4, 2017

The project aims to analyze climate data with focusing on use of Hadoop Framework for data analysis and Ansible for automating deployment and monitoring.

Keywords: Cloud, I524

Report: https://github.com/cloudmesh/....fillmeout
Code: https://github.com/cloudmesh/....fillmeout

WARNING:This project has not yet completed a security review and detailed explanation and a separate security section will be added by the team to discuss the security. The team has also to explain if the use cloudmesh cluster and cross_ssh and if not why not. If they do not they need to reinvent this logic. We could not immediatly find in the code the portion that deals with security, so it is also important to explicitly explain to us where and how this is done.

1. INTRODUCTION

The study of environmental science and climatic changes around has been done for decades, the study has always been predictive based on the past experiences and forecasting of the weather conditions around us. With use of modern days technologies it determining the climatic changes and with analysis done around it has helped human being to prepare and face the natural calamities. Though with current equipment weather department has strengthen their arms but has not been able to be full proof and many time its not been able to predict/ forecast the climatic changes effectively. The study of the whether data and geo graphical changes is ongoing evolving process. Thus more and more researcher needs modern days tools and technologies to leverage it and forecast more accurately.

1.1. Objective

The goal of this is to study the weather data and analyze the relationship between the geo graphical changes such change in geo magnetic field and/or natural disaster. With use of Hadoop for distributed data analysis aims to finds any pattern that might exists between these parameters. The course of the analysis will also provides visualization of these parameters in order to identify any pattern in a more intuitive way. By leveraging the power ansible for application deployment over cluster and monitoring the application performance to determine scalability and throughput. The conclusion will be determine by establishing any existing pattern, analysis done over it and by visualizing it.

2. DATA SOURCES

Weather data has been recorded since 19th century. This data can be used to estimate climate changes and forecasting. The same data can be can be used to find any existing pattern with natural disasters. Following sources has been compiled for weather, natural disaster and geo magnetic fields.

- Weather-Data[1]
- Natural Disaster[2]
- Geo Magnetic Field[3]

3. HIGH LEVEL DESIGN

The design of the application is thought of leveraging power of Hadoop as main processing unit of analysis with deployment on the cluster environment where application requires multiple processing units for execution, database for persistence and visualization tools for graphical outputs. The project is divided into following steps:

- Data cleaning and persistence - The raw data cannot be use directly for analysis. First data has to be parsed and required parameters will be extracted. Then this extracted data will be dumped into a NoSql database.
- Core Analysis Program - Core analysis program will be responsible for figuring out any hidden patterns between aforesaid parameters. Program will compare natural disasters occurred, geo-magnetic orientation and climate data set on a given location and duration and compute relationship

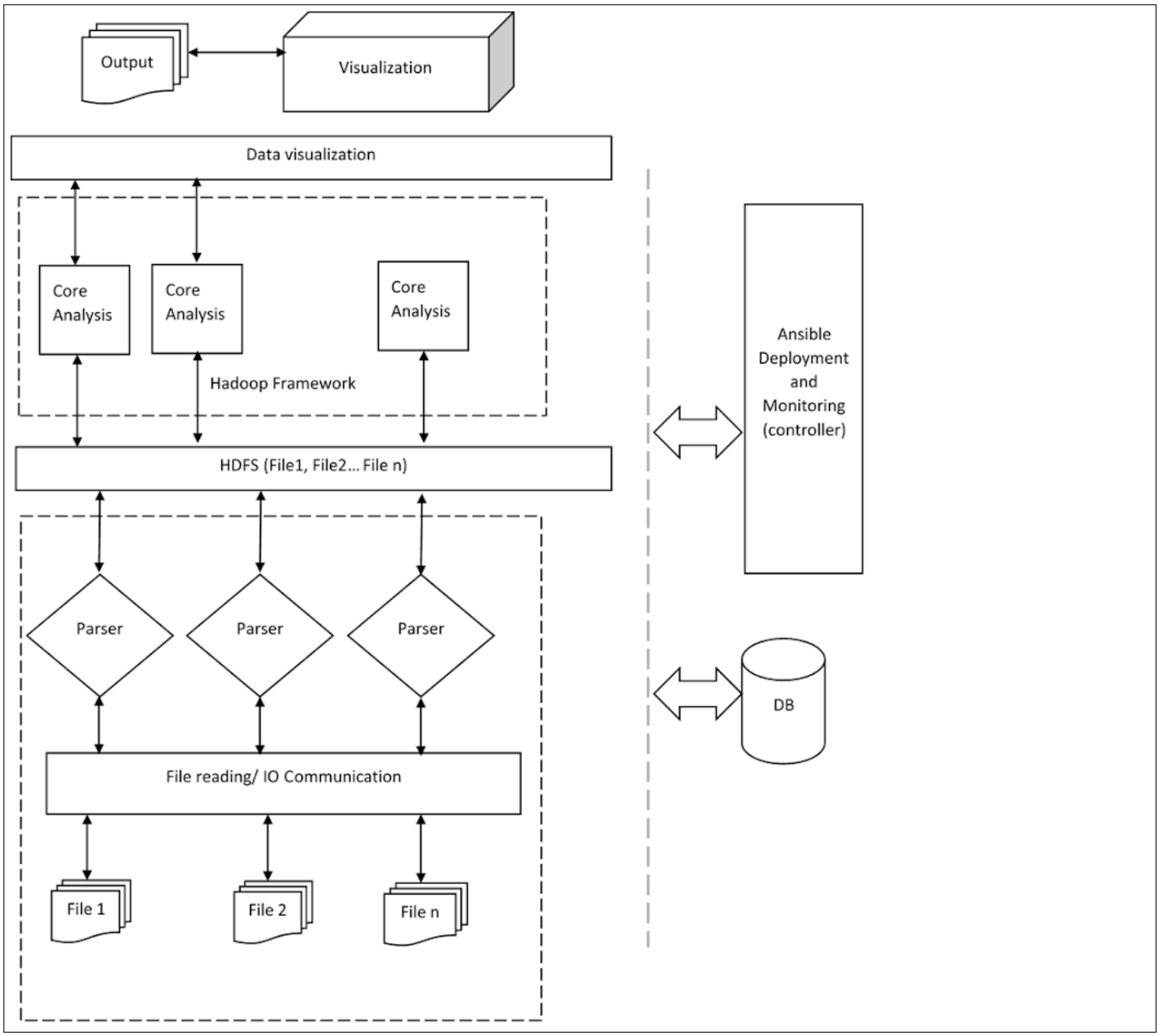


Fig. 1. Architecture

between them. The program will be an MapReduce implementation and is the heart of the application. The program will be executed through Hadoop framework. Hadoop will execute the program in a distributed manner.

- Deployment and Monitoring - The application needs multiple processing units and monitoring system. Ansible will be used for deployment and manage nodes for program execution.Ansible will be responsible for following tasks i) Deployment and configuration of Hadoop on the multiple nodes. ii) Starting Hadoop servers, inserting/reading data. iii) Execution of the commands to run the analysis using Hadoop to filter the input data and write response to HDFS or some output file. iv) This output can be then passes to the visualization step as the input data.

- Visualization - Finally once the programs completes execution, using the scikit-tool or other visualization tool kit and the output file, graphs and patterns depicting the relationship can be plotted more intuitive representation.

- BenchMarking - The application can be benchmarked for the scalability by addition more nodes and checking the performance for strong scaling. The report will be represented in tabular format.

4. DATA CURATION

Getting data ready is the very first and basic step for analysis. We have chose NCDC [4] as our source of data. NCDC [4] exposes few rest apis for accessing weather data. Following will give you a brief understanding on the apis used for getting the required data.

1. Datasets : This groups data into monthly daily , yearly pattern. There are eleven different datasets. We will be choosing GSOM (Global Summary Of Monthly) as our primary datasets. (URL : https://www.ncdc.noaa.gov/cdo-web/api/v2/datasets ,Attributes : GSOM). For this project we will use GSOM only.

2. Data Categories: This groups data into data category like Temperature, Pressure etc. We will consider only few. (URL : https://www.ncdc.noaa.gov/cdo-web/api/v2/datacategories, Attributes : "TEMP" (Air Temperature) , "PRES" (Pressure) , "EVAP" (Evaporation) and "VELOCITY" (Velocity)). For our project we will be using PRCP, SNOW, TMAX, TMIN and TAVG.

3. Data Types: This group Data Categories into further smaller sub types. (URL : https://www.ncdc.noaa.gov/cdo-web/api/v2/datatypes?datasetid=GSOM&datacategoryid=TEMP , Attributes : "TAVG" (Average Temperature), "TMAX" (Maximum Temperature) and "TMIN" (Minimum Temperature)) Other Data Categories does not have sub type.

4. Location Categories: This groups data in terms of location. (URL : https://www.ncdc.noaa.gov/cdo-web/api/v2/locationcategories , Attributes : "CITY" , "CLIM_DIV" (Climate Division), "CLIM_REG" (Climate Region) , "CNTRY" (Country) ,"ST" (State)). For this project we will be downloading data for India only.

5. Location: Groups data in terms of country, state etc. (URL : https://www.ncdc.noaa.gov/cdo-web/api/v2/locations?locationcategoryid=CNTRY, Attributes : "FIPS:IN" (INDIA) , "FIPS:IO" (Indian Ocean))

6. Station : This collects stations details based on the location id. (URL : https://www.ncdc.noaa.gov/cdo-web/api/v2/stations?locationid=FIPS:IN , Attributes: ids , mindate and maxdate). All of the stations details will be collected.

7. Data : This collects actual weather data for the given stationId, startdate and enddate. (URL : https://www.ncdc.noaa.gov/cdo-web/api/v2/data ?datasetid=GSOM&stationId=GHCND:IN1NCBC0005 &startdate=1970-10-10&enddate=1971-10-10). Since the project only requires only 5 attributes, further filter is applied while invoking the API. The data received will be saved into database.

4.1. Collecting Data

NCDC uses token based authentication for security and with each token , no more than 10,000 hits are allowed per day. Well 10,000 seems huge but truly its not. We have targeted one country (India) and 5 attributes chiefly precipitation, snow, maximum temperature, minimum temperature and average temperature for each month. Total number of weather data stations in the country is around 3500 and each station has a data range of 30 to 50 years. This leads to a total of 100,000 hits or more. To handle this, we need to load data incrementally, i.e. the curation program should have the ability to resume the download from the last save point. Before we dive more into logical section , lets first see the technology stack for data collection.

4.2. Technology Stack

We have considered python , Apache thrift and Apache Hbase for data curation step. We will be covering a short introduction of the steps to configure the system, before diving into the core logic of curation. Since we are going to use Apache Hadoop for our analysis purpose, Hbase comes as a natural choice of NoSql database. Java is the default language for both Hadoop and Hbase. So now the question is how to connect to Hbase using python and the solution is Apache Thrift or thrift. Thrift is a library from apache which generates hbase client. Thrift supports many third party languages including python. The following steps will be required to install thrift first.

4.2.1. Install Apache Thrift [5]

1. Download Thrift from "http://redrockdigimark.com/apachemirror/th 0.10.0.tar.gz".

2. Extract the file and run ./configure.

3. Execute sudo make install or simply make to generate the binaries. If using make, the thrift binaries has to put into path manually (can be found under compiler/cpp/).

4. After thrift is available in path , it can be tested by running "thrift –version".

5. Then the python module has to be created to be used within python. To do this download "https://github.com/apache/hbase/blob/master/hbase-thrift/ src/main/resources/org/apache/hadoop/hbase/thrift/Hbase.thrift" file.

6. Then execute "thrift –gen py <path/to/Hbase.thrift>". This will generate "gen-py" folder.

7. Add this folder to python path by export PYTHONPATH=<path/to>/gen-py/:$PYTHONPATH.

For details you can check Installation steps.Installation of Hbase will be discussed later along with Hadoop. Now thrift is ready to use. Execute "/hbase thrift start" to start the thrift server. Hbase has to be started separately.

4.2.2. Install Happybase [6]

Happybase is a wrapper program written on thrift to facilitate data access layer in python in a more readable and swift way. Rather than using thrift client directly we will be using happybase python module. To install happybase run "pip install happybase". Once done test the python library by running the following test code

Listing 1. Connect-Hbase

```
import happybase as hbase

connection = hbase.Connection('localhost')
print connection.tables()
```

5. HBASE AND TABLE STRUCTURE

Hbase [7] is a column based key-value database. We have considered two tables for weather data 'wda_stations' and 'wda_weather'. The structure of the tables are as follows

Listing 2. WDA_STATIONS Table

```
[{station-id-key : {
    station : {name : {} , id : {}},
    date_range : {
        min_date : {} ,
        max_date : {}
     },
    location : {
        latitude : {},
        longitude : {}
    }
}}]
```

Listing 3. WDA_WEATHER Table

```
[{yyyy-mm-key : {
    weather : {station-id : {
        data-type : {value}
}}}}]
```

5.1. Table Definition

Stations table will consist of all the weather stations with its location and the date range within which it is was active. Location will have the latitude and longitude. The station id will be the key for the table. NCDC uses a uniquely identifiable key for the stations across the globe. This makes the best attribute for the primary key irrespective of country and region. Weather table is a more completed table and has dynamic column family. The key for the table is the year and month in (YYYY-MM) format. Each key contains all the stations with their weather parameter. For example say a key 1967-01 (meaning January 1967) is having 10 stations with station id sid-1 to sid-10. Each station id will having 4 parameters or data types (TAVG,TMAX,TMIN and PRCP). And finally each parameter have their values.

6. DATA PERSISTENCE LAYER

The project is structured with python packages. We have a specific package for persistence as 'iu.i524.S17IRP013.dao'. The dao has the scripts to connect hbase and performs db operations. The base script is 'hbase_connect.py' which does the connection. The connection is made through Apache Thrift Rest Api. It has the connection object , which is used by the other two dao classes.

6.1. Station Dao

This script does all CRUD operations on stations table. On startup it will create the table if not available. It has 2 main functions insert and retrieve. The insert function takes key value pair as argument and put it into hbase table. Another function and the important one is 'get_station_data' which takes the start row as argument and returns next ten records. This is done via scan command of hbase which supports record limitation.

Listing 4. Stations Dao

```
def get_station_data(start_row=''):
    st_list = dict()
    count = 0
    if(start_row == ''):
        for key, data in table.scan(limit=10):
            st_list[key] = {'min_date':data['date_range:
                min_date'], 'max_date':data['date_range
                :max_date'] , \
                            'station_id': data['station:
                                id'], \
                            'latitude':data['location:
                                latitude'], \
                            'longitude':data['location:
                                longitude']}
    else:
        for key, data in table.scan(limit=11, row_start=
             start_row):
            count = count + 1
            if(count > 1):
                st_list[key] = {'min_date':data['
                    date_range:min_date'], 'max_date':
                    data['date_range:max_date'], \
                            'station_id': data['station:
                                id'], \
                            'latitude':data['location:
                                latitude'], \
                            'longitude':data['location:
                                longitude']}
    return st_list
```

6.2. Weather Dao

Weather dao is similar to station in dao in all respect except it deals with weather table. Apart from the getting records from the weather table, there is another method to get all keys from the table. We will see in later sections why we need the entire key list from the table.

7. WEATHER DATA PERSISTENCE

Once happybase, thrift and hbase is functional, we are ready to download our weather data. Weather data download is divided into two steps 1) Getting weather stations details for a given country and 2) Getting Weather data for a given station. Lets see them individually :

- Download Weather Stations - To consume rest services we have used python's inbuilt request response module. Let us walk you through the code. We have two main python script one for data access and another for rest consumption. "stations_dao.py" is for accessing table "wda_stations" in hbase. The functions are self explaining and hence will not

be repeated here. "weather_services.py" is for consuming NCDC rest services. To load all stations , we will be using the code *'FIPS:IN'* i.e the region code for India.

Listing 5. Get Weather Stations

```
def get_stations(country='FIPS:IN', offset=0):
    url = NCDC_API + NCDC_SERVICES.STATIONS + \
    '?locationid=' + country + '&offset=' + str(
        offset)

    response = requests.get(url, headers=HEADERS)
    return response.json()

def load_stations():
    limit = 25
    offset = 0
    nbr_of_records = 0
    result = Services.get_stations(offset=offset)
    while(result != {}):
        nbr_of_records = nbr_of_records \
        + insert_station(result['results'])
        offset = offset + limit
        result = Services.get_stations(offset=
            offset)
    print str(nbr_of_records) \
    + ' stations loaded successfully !!'
```

The function 'load_stations' calls 'get_stations' till all the available stations are downloaded. 'insert_station' is the *DAO (Data Access Object)* call and inserts the dataset into hbase table.

- Download Weather Data - With all stations in the table, we invoke the data api of NCDC for downloading weather data. The argument for getting weather data is station id and date range. The two main methods for loading weather data are as follows

Listing 6. Get Weather Stations

```
def load_weather_data():
    start_row = ''
    st_list = stations.get_station_data(start_row=
        start_row)
    while st_list != {}:
        for key, value in st_list.items():
            get_weather_data(startDate=value['
                min_date'], \
            endDate=value['max_date'], stationId=
                key, station_details=value)
            start_row = key
        print 'Weather Data loaded till station id
            = ' + key
        st_list = stations.get_station_data(
            start_row=start_row)

def get_weather_data(station_details, startDate='
    1968-01-01', endDate='1970-01-01', stationId='
    GHCND:IN001011001'):
    date_range_list = AppUtil.get_year_list(
        start_date=startDate, end_date=endDate)
    for date_range in date_range_list:
        limit = 25
        offset = 0
        result = Services.get_weather_data(
            startDate=date_range['min_date'],\
            endDate=date_range['max_date'],
            stationId=stationId, offset=offset)
        while(result != {}):
            # print result['results']
            insert_result(result['results'],
                station_details)
            offset = offset + limit
            result = Services.get_weather_data(
                startDate=date_range['min_date'],\
                endDate=date_range['max_date'],
                stationId=stationId, offset=offset)
```

The 'load_weather_data' function retrieves station data from the station table in a batch of ten. For each station id, it is then passed to 'get_weather_data' function to get weather data for the given station id. This data is again persisted into wda_weather table for future analysis and usage. These are the main functions to download weather data. All other functions are helper functions to enable the download.

8. INSTALL PYTHON PROGRAM

Our python programs are structure and packaged. So to use it, we need to install the packages , in order to make them available in the classpath or pythonpath. There is a setup.py script provided with the main source folder. This script installs the entire package into python core folder. To execute this *pip* must be pre-installed. Once pip is available run the following command

Listing 7. Install python packages

```
sudo pip install  <path_to_src_folder>/src
```

Use sudo or super user to install as other may have permissions issues. Once they are installed , we are ready to run our map reduce program.

9. EXECUTION

The application expects all the data in hbase are prefetched and ready. For downloading the datasets into hbase follow the steps

- Install the python packages. Check all installations hbase, hadoop , thrift and happybase.
- Start Hadoop and Hbase and Apache Thrift.
- Open up python cli and run the command 'iu.i524.S17IRP013.hadoop.init.DataSetup.py'.
- The above command will take time and will dump data into hbase table.
- It also created hdfs input folder as '/wda/input' and write 'wda_row_keys.txt' file into it. This file will have all the keys (year-month format) from weather table.

9.1. Run MapReduce

We need hadoop streaming api to execute our ma reduce program. Hadoop streaming api reads from hdfs and write it onto the standard io. This data is then read by our map program *wda_mapper.py* and further sent to reducer program *wda_reducer.py*.

Listing 8. Run MapReduce

```
bin/hadoop jar <path_to_streaming_jar>{share/hadoop/
    tools/lib}/hadoop-streaming-2.7.3.jar \
-file <path_to_mapper>/run/wda_mapper.py -mapper
    wda_mapper.py \
-file <path_to_reducer>/run/wda_reducer.py -reducer
    wda_reducer.py \
-input /wda/input -output /wda/output
```

The above will execute the map reduce analysis program and write the output into hdfs '/wda/output' folder. Once the program is finished check the output with following command :

Listing 9. Run MapReduce

```
hdfs dfs -ls /wda/output
hdfs dfs -cat /wda/output/<name_of_the_file>{part-r
    -00000}
Output >>
{'TAVG': ('GHCND:IN001020700', '35.02'),
'TMAX': ('GHCND:IN001020700', '43.08'),
'PRCP': ('GHCND:IN001011000', '863.7'),
'TMIN': ('GHCND:IN001020700', '14.08')}
```

10. DEPLOYMENT USING ANSIBLE

Ansible is open source automation tool. It can be used for deployment of software, configuration management and automation in the execution of application. It also serves for monitoring of the state of the application. As per current state of our project we have used ansible for deployment of software in our project. The script deploys Java, Hadoop, Hbase on the independent clusters. It also configures the properties in the key files. This script can further be extended to build the analysis , deploy and run the analysis module. Monitor the progress of the execution and display the reports, however currently it is work to be done.

In the project a separate directory structure is created for deployment. This folder consists of some key files and folders. We can dig into the details for each of them in below section.

1. Inventory.
2. Playbook.yml.
3. Roles —> Roles is a directory and we can going to explore the roles used in the project as we proceed.

10.1. Inventory Configuration

Inventory file contains the list of hostname or nodes that can be accessible by ansible. These nodes are then used in the script for deployment and configuration. In the current project chameleon server nodes were created and configured. These IP address can be changed dynamically. This file can also contain the group inside Which multiple ip can be configured. The deployment of hadoop was in clustered mode where we have on master and multiple slaves. In the inventory.configuration file there are multiple section with header which mentioned the list of nodes/IP address those needs to be included in the group. For example in the below file we have two groups.

Listing 10. Inventory

```
[weatherClusterMaster]
    <ipaddress1> ansible_ssh_user=cc
[weatherClusterSlave]
    <ipaddress2>  ansible_ssh_user=cc
    <ipaddress3> ansible_ssh_user=cc
```

1. WeatherClusterMaster : Node/Instance with ipaddress1 falls under the master cluster.
2. WeatherClusterSlave: Node/Instance with ipaddress2, ipaddress3 falls under the slave cluster.

The mapping of IP address to the group from the inventory file is used in the playbook.yml to represent them as a group. There are some script which needs to be part of the master cluster deployment however it don't need to be executed on the slave cluster. This problem is address using the playbook.yml roles

10.2. Playbook

Playbook works on mentioned host of group. It mentions the roles those will be operated on each of the host. The main.yml inside the task folder within each roles will be applied to The cluster.

Listing 11. Playbook Modules

```
---
- hosts: weatherClusterMaster
 remote\_user: root roles:
     - java
     - hadoop
     - hbase
     - master
 - hosts: weatherClusterSlave remote\_user: root
 roles:
     - java
     - hadoop
     - hbase
```

As mentioned above the hosts mentions the instance on which the scripts are going to be executed. So all the ipaddress those are mentioned under the weatherClusterMaster will undergo the execution of scripts those are mentioned in each of the roles for that task. These script will be executed using the remote_user mentioned in the script. In above example from project, both the cluster will execute the script mentioned inside java, hadoop and hbase role however master cluster additionally will execute the script from the master role folder as well. These could be the addition scripts like configuration of slave addresses in the slaves file and many such task which needs to be executed only for master instance.

10.3. Roles

Roles folder contains the list of the roles mentioned in the playbook.xml. There could be multiple roles which can be executed on node or group of nodes. In current project there are four roles i.e. java, hadoop, base and master. The name are very much self explainatory. Java : — Java role installs java on the remote machine . Roles contains multiple directory. The task directory is the main directory which mentions the list of task / script those needs to be executed on the nodes.

Current project mentions 4 different task in java. All these task takes care of installing java on the remote node.

Hadoop — Hadoop along with Hbase is the major role. The task file mentioned the below list of task

1. Download hadoop.
2. Extract hadoop.
3. Set java home/ Hadoop home configuration in bashrc.
4. Set Java home/Hadoop home path to env.rc
5. Create and Configure logging file.
6. Update core-site.xml
7. Configure slaves
8. Update other configuration files like core-site.xml, hdfs-site.xml, yarn-site, mapper- site.xml.
9. Replace hosts.

Sources fIle
The sources folder consists of other key folders like

1. Vars This is yml file which maintains the mapping for placeholder names used in the configuration files.

2. Meta : This file can contain it dependency roles and other details.

3. Templates : This contains the template configuration file.

After Hadoop, HBase, Thrift is installed and configured using similar way. You can refer to the main.yml inside the task folder in for the script details.

11. BENCHMARKING

The aforesaid program has been executed on futuresystems. The execution time has been collected by increasing nodes from 1 (single) to 3 (multiple) with 2cpus each. The graph indicates the

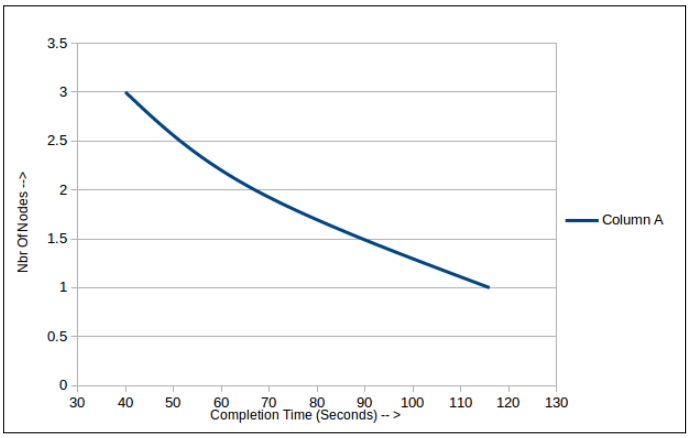


Fig. 2. Benchmarking

reduction in time with increase in nodes. Since the data used for the sample is not big enough , the time taken for starting the framework is considerably high as compared to total time taken by the analysis program itself. So only the map reduce time or

```
17/05/01 12:12:19 INFO mapreduce.Job:  map 0% reduce 0%
17/05/01 12:13:29 INFO mapreduce.Job:  map 33% reduce 0%
17/05/01 12:13:30 INFO mapreduce.Job:  map 67% reduce 0%
17/05/01 12:13:41 INFO mapreduce.Job:  map 83% reduce 0%
17/05/01 12:13:42 INFO mapreduce.Job:  map 100% reduce 0%
17/05/01 12:14:08 INFO mapreduce.Job:  map 100% reduce 67%
17/05/01 12:14:10 INFO mapreduce.Job:  map 100% reduce 100%
17/05/01 12:14:15 INFO mapreduce.Job: Job job_1493620614560_0001 completed successfully
17/05/01 12:14:17 INFO mapreduce.Job: Counters: 51
```

Fig. 3. Map Reduce Snapshot

the program time has been considered and not the ancillary time which includes time to start the hadoop framework.

12. SCOPE OF EXTENSIONS

The application is only distribute the data load from the hbase table. But fetching data from NCDC is sequential. Though its only a one time job , it takes considerable amount of time to download. The same application can be extended to use Hadoop MapReduce for downloading and persisting the raw data from Rest APIs. The same fundamentals can be used for this purpose. Initial weather stations can be downloaded sequentially and then the keys are to be distributed in HDFS for further weather data download.

12.1. Limitation

There is a limitation in download imposed by NCDC. NCDC uses token bases authentication which allows only 10000 invocation per day. To distribute the download among multiple nodes requires multiple tokens.

13. PROBLEM AND WORKAROUND

Usually Hadoop is run with java and connects directly with the Hbase to retrieve the data from database. However in the current implementation we have python script which is used for the analysis. The challenge was to fetch the data from Hbase by running the Hadoop program in python. There were some of the library available which connects Hadoop to Hbase however those were not reliable and supported by Apache.

Workaround for Connecting Hadoop to Hbase

In order to execute the python script on Hadoop in cluster, there was a work around considered. In this work around, all the keys were stored in the hadoop file system. These key were then divider as per the nodes and passed to the mapper method. For each key, mapper method called the hbase to fetch the data. This call to Hbase was via apache thrift and happybase

Let drill down more on what is Apache thrift.

Apache thrift is mechanism or framework which can be talk between two services which may or may not been written in same language. Apache thrift has its own Interface definition language, its compiler can generate client code and server code in same or different language and hence can be used as a bridge to talk between the two different language.

Apache thrift is very similar to SOAP. The way SOAP uses UDDI for publishing and discovering the service, thrift uses zoo keeper for finding services. Apache thrift was connecting to the Base with the help of Apache

HappyBase is a developer-friendly Python library to interact with Apache HBase. Below the surface, HappyBase uses the Python Thrift library to connect to HBase.

14. CONCLUSION

MapReduce (MR) with Hadoop is an efficient framework for distributed computing. It can be run on any commodity hardwares and virtual machines. It has also some useful plugins available for shared computing which can share dataset without doing any IO operations,ex - Twister and Spark. Python with Hadoop on other hand is not so great combination as the communication requires additional layer i,e Standard IO. We have seen earlier that there is no support for Hbase communication within the hadoop as well. FutureSystems and Jetstream works well with the framework. Chameleon VMs did not have all ports opened (except 22 for ssh) which is a must for Hadoop to work in cluster as Hadoop uses IPC protocol for inter node communication.

REFERENCES

[1] “Weather data,” Web page. [Online]. Available: https://www.ncdc.noaa.gov/

[2] “Natural disaster,” Web page. [Online]. Available: http://www.emdat.be/

[3] “Geo magnetic field data,” Web page. [Online]. Available: https://geohazards.usgs.gov/mailman/listinfo/geomag-data

[4] “Ncdc rest api,” Web page, Jan. 2017, accessed: 2017-04-25. [Online]. Available: https://www.ncdc.noaa.gov/cdo-web/webservices/v2

[5] BRUNDESH, “Install apache thrift with python,” Web Page, Jan. 2017, accessed: 2016-01-25. [Online]. Available: https://acadgild.com/blog/connecting-hbase-with-python-application-using-thrift-server/

[6] Happybase, “Install happybase,” Web Page, Apr. 2017. [Online]. Available: https://happybase.readthedocs.io/en/latest/installation.html
[7] A. Hbase, “Apache hbase,” Web Page, Apr. 2017, accessed: 2017-04-27. [Online]. Available: https://hbase.apache.org/

Analysis of Airline delays data using Spark and HDFS

BHAVESH REDDY MERUGUREDDY[1,*] AND NITEESH KUMAR AKURATI[1,**]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*] Corresponding authors:bmerugur@iu.edu
[**] Corresponding authors: akuratin@iu.edu

project-P014, May 4, 2017

Airline delays data is analyzed by developing an automated process for deploying Hadoop and Spark on Chameleon and Jetstream cloud computing environments. The data set used is publicly available and analyzed for obtaining various results like average delay of an airline and an airport. The automation process is carried out using Ansible scripts and a cloud manager called Cloudmesh Client is used to interact with the clouds. Spark is used as the cluster computing framework and Hadoop Distributed File System is used as the distributed storage system for the data sets. Benchmarking is done after the analysis to determine the efficiency and performance of the system.

Keywords: Ansible, Spark, Cloudmesh Client, Hadoop, YARN

https://github.com/cloudmesh/classes/blob/master/project/S17-IR-P014/report/report.pdf

1. INTRODUCTION

Analysis of airline delays data by deployment of Hadoop and Spark on Chameleon and Jetstream clouds is the main focus of the project. A data set having the airlines information such as flight arrival time, departure time and average delays is considered. This data set is available to everyone. Cloudmesh Client is used as the cloud manager which provides command line to access multiple clouds. It is used to create a Hadoop cluster with Spark as an add-on. The cluster is then deployed on Chameleon and Jetstream clouds by using the Cloudmesh Client. Ansible scripts are written and the Cloudmesh Client interacts with these scripts to automate the deployment.

Ansible scripts are written for extracting data sets from the published zip file and deploying them on the clouds. Hadoop Distributed File System is used to store the extracted data sets. A program is written in Spark to perform the data analysis. Spark runs on the Hadoop cluster and accesses the HDFS for retrieving the data sets. There are several results that are obtained from this analysis. Top ten airports that have delays are identified, average delay per an airline and per an airport is determined and top ten airlines with comparatively more delays are identified.

The program is deployed by using an Ansible script. Bar graphs are drawn for the analysis performed. Along with the deployment and analysis, benchmarking is done to evaluate the performance of the program on each node of a cluster and on different clouds. The efficiency of the program is determined by varying the sizes of the data set and comparing the results.

2. INFRASTRUCTURE

Infrastructure for the project includes Cloudmesh Client, Chameleon and Jetstream clouds. Cloudmesh Client is used to access multiple clouds from a single command line. Chameleon and Jetstream provide cloud computing environments for the system.

Cloudmesh Client is a toolkit that provides a standardized interface for accessing various workstations, clusters and heterogeneous clouds. It acts as a manager that allows users to manage the available set of resources. Cloudmesh Client plays an essential role in the deployment process by handling the interactions between users and virtual machines being used in the clouds [1].

Cloudmesh Client provides several services which make it easy for the users to manage the virtual machines in the clouds. The "vm boot" command in Cloudmesh Client is a single instruction for creating virtual machines. Security rules can be uploaded to the clouds by using "secgroup" command from Cloudmesh. Key management in the clouds is simplified Cloudmesh's key add and upload commands. Deletion of the virtual machines created can be easily carried out by specific commands defined in Cloudmesh.

Cloudmesh Client makes it easy for the users to switch virtual machines from one cloud to other by specifying the name of the cloud. Cloudmesh provides a command shell that allows users to develop and run scripts and each command can be called by the user from the command line. Cloudmesh Client essentially provides virtual machine management through a convenient programmable interface.

2.1. Chameleon Cloud

Chameloen is a project aimed at providing large-scale open research platform for cloud design and services. The project receives funding from the National Science Foundation (NSF). Chameleon provides a wide range of services like developing platforms-as-a-service, optimizing virtualization technologies and infrastructure-as-a-service components [2]. Chameleon allows full user configurability of the software stack, ranging from provisioning of bare metal to the delivery of high functioning cloud environments, by supporting a graduated configuration system.

The Chameleon testbed is hosted at the University of Chicago and the Texas Advanced Computing Center. It consists of 5PB of total disk space with 650 multi-core cloud nodes. A portion of the testbed is dedicated for supporting experiments with large disk, high memory and co-processor units. Chameleon facilitates integration of clouds and networks enhancing their capabilities.

2.2. Jetstream

Jetstream is a cloud computing environment that can be used by researchers as a configurable infrastructure. They are provided with interactive computing and data analysis resources [3]. Jetstream allows researchers to create their own private computing system with customizable virtual machines. Jetstream's operational software environment is based on OpenStack and has a web-based user interface. It provides a library of virtual machines for performing specific analysis tasks. It can be used for tailoring workflows for both small scale and larger scale environments. It can also be used as the backend to science gateways to supply research jobs to HTC or other HPC resources.

Table 1 shows the specifications used from both Jetstream and Chameleon cloud environments.

Table 1. Hardware Specifications of Chameleon and Jetstream

	Chameleon	Jetstream
CPU	Xeon X5550	Haswell E-2680
cores	1008	7680
speed	2.3GHz	2.5GHz
RAM	5376GB	40TBr
storage	1.5PB	2 TB

3. SOFTWARE STACK

Following are the deployment and analysis tools used in the project.

3.1. Ansible

Ansible is an open-source software that facilitates automation of configuration management and application deployment. Ansible consists of controlling machines and nodes. Controlling machine starts the orchestration and manages the nodes over SSH [4]. Resources are not consumed by Ansible when the nodes are not being managed. This is due to the fact that there are no daemons that run for Ansible in the background. This makes Ansible a software with an agent-less architecture. This architecture prevents the nodes from polling the controlling machine thereby reducing the overhead on the network as shown in figure 1.

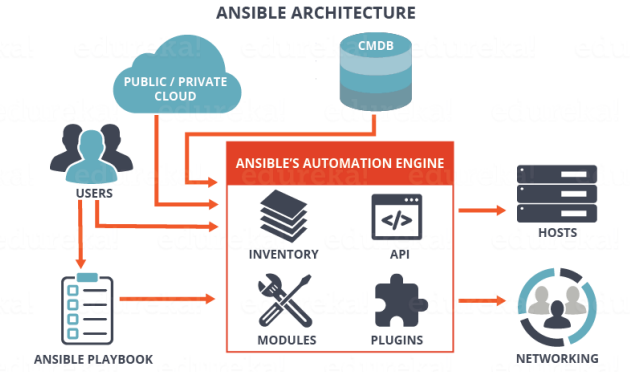


Fig. 1. Ansible Architecture [5]

Modules, Inventory, Playbooks and Ansible Tower are the components of the Ansible architecture. In Ansible, a module is work unit written in a scripting language. It is idempotent and standalone. Inventory is a configuration file that lists the nodes that are accessible by Ansible. It allows the users to add a set of nodes to a group. Nodes are generally represented by IP addresses or hostnames.

Playbooks are YAML format files which consist of configurations and express deployment in Ansible. A group of hosts are mapped to a set of roles through Playbook. Ansible Tower is a web-based console which makes Ansible a center for automating tasks. Ansible is consistent and minimal in nature. Ansible does not deploy agents to nodes which makes it very secure.

3.2. Apache Spark

Apache Spark is an open source framework that provides cluster-computing capabilities. Spark allows its users to program different clusters by providing an interface [6]. It facilitates fault-tolerance and data parallelism. Spark makes use of a data structure called as resilient distributed dataset (RDD) which is distributed over different virtual machines in a cluster.

RDDs are immutable which means that they cannot be changed once they have been created. They provide mechanisms for exploratory data analysis and iterative algorithms for processing dataset iteratively. Spark interfaces with systems like Cassandra, Hadoop Distributed File System (HDFS) and Amazon S3 for distributed storage and interacts with Hadoop YARN for cluster management.

Task scheduling, dispatching and some fundamental I/O functionalities are achieved in Spark through the Spark Core. It is an application programming interface which reflects functional programming. Functions similar to map and reduce are provided by the interface which produces new RDDs as output by taking in the required RDDs. RDDs make use of different types of Java, Scala or Python objects. The operations of RDDs are fault-tolerant and lazy. Structured and semi-structured data is supported in Spark through Spark SQL that processes a new data model called DataFrames. Spark SQL provides ODBC/JDBC server and command-line interfaces.

RDD transformations are performed on the data by the Spark Streaming component. It takes in data and performs streaming analytics. Spark MLlib is a machine learning framework that simplifies machine learning pipelines in Spark [6]. MLlib

is provided with several statistical and machine learning algorithms. This reduces the overhead of performing classification and regression, correlations, linear regression, support vector machines and k-means clustering method. A simple spark architecture can be observed in figure 2. Apache Spark consists of a graph processing component known as GraphX. It depends on RDDs and generally used for graphs that are immutable.

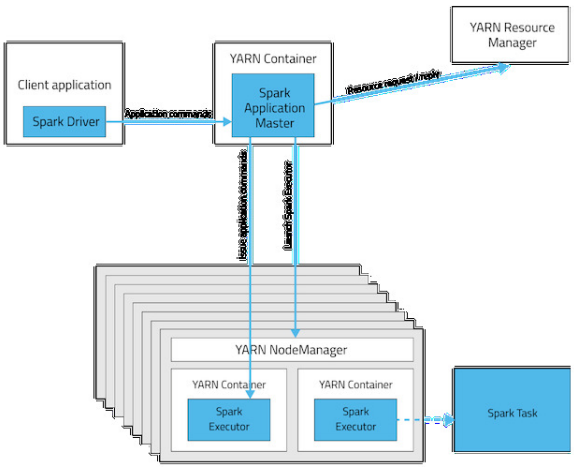


Fig. 2. Spark with Yarn Architecture [7]

Generally, map and reduce functions use variables which are defined outside the functions in Spark driver. New copies of each variable are provided to the tasks running on the cluster but the driver is not provided with the updates of these copies [8]. To solve the problem, Spark makes use of shared variables called accumulators. An accumulator can be considered as a container used for aggregating data across different tasks running on multiple executors.

Accumulators are designed for distributed sums and counters and can be effectively used for distributed computations [9]. They act as read-only variables for the executors and can only be read by the driver programs. Accumulators are not thread-safe but they are serializable. They can be safely sent over the wire for execution after being referenced in the code in executors. Accumulators even help in the debugging process by counting the events.

3.3. Hadoop Distributed File Systen

The Hadoop Distributed File System (HDFS) is a distributed file storage system that provides reliable and scalable data storage. It is a fault-tolerant storage system. It spans large clusters of commodity servers [10]. It supports thousands of servers and a billion files. HDFS distributes storage and computation across many servers making the combined storage resource grow with demand and remain economical at every amount of storage.

HDFS allows the users to connect the nodes across several clusters in which the data is distributed. It provides high throughput access to large datasets [11]. The data files can be accessed by the users in a streaming manner as the data files are stored as a continuous file system. MapReduce programming model is employed when applications are executed. HDFS has a write-once-read-many model which simplifies data coherency and lightens the requirements of concurrency control. It allows only one writer to write data at a given point of time. It appends bytes to the end of a stream and stores the streams in the order they were written.

HDFS provides portability across heterogeneous operating systems and ensures efficiency by processing the distributed data in parallel. It automatically redeploys processing logic in the failure situations by maintaining multiple copies of data. Rather than processing data close to logic, HDFS processes logic closer to data. It is accessible in different ways.

A web browser can be used to browse files in HDFS. It consists of a single node called name node and several data nodes that store data as blocks within the files. The name node is responsible for regulating client access to files and managing the namespace of the file system. This includes opening, closing and renaming files and directories. Name node monitors the data nodes in creating, deleting and replicating data blocks by mapping them to the data nodes. Each data node contains an open server socket through which remaining data nodes read or write data.

To be fault-tolerant, HDFS replicates file blocks according to the number that an application specifies. It optimizes replica placement by using an intelligent replica placement model which in turn, ensures reliability and efficiency. HDFS supports large files by placing each file block on a different data node. To overcome failures, it makes use of heartbeat messages for detecting connectivity between data nodes and the name node. Data nodes are required to send heartbeat messages to the name node periodically and the failure is detected when name node stops receiving the messages. In this situation, the data node is marked as dead and removed from the system. When the data node count reaches a limit value, replication is done by the name node. Figure 3 shows the architecture of HDFS.

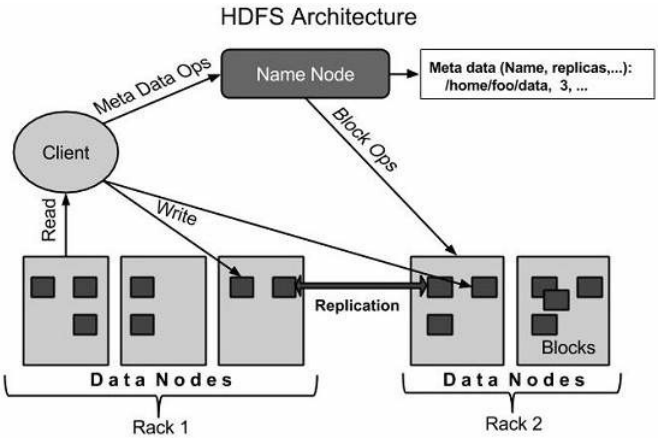


Fig. 3. HDFS Architecture [12]

HDFS supports data block rebalancing to avoid the used space for data nodes from being underutilized. If the free space on a data node is too low, it automatically moves blocks from one data node to other. Rebalancing is also done when new nodes are added to the cluster. It ensures integrity of data stored in HDFS. The file system performs checksum validation on the files by storing computed checksums in separate files in the namespace of actual data [11]. All other HDFS functionalities are similar to that of other distributed file systems.

3.4. YARN

Yet Another Resource Negotiator (YARN) is a technology used for cluster management. Hadoop supports a broad range of ap-

plications through YARN as it decouples MapReduce's scheduling mechanism and resource management from the data processing component [13]. YARN consists of a node manager and a central resource manager. Node manager monitors the operations of cluster nodes while the resource manager manages the Hadoop system resources which are used by the applications. YARN separates HDFS from MapReduce which improves the efficiency of the Hadoop environment in processing different operations.

The resource manager is responsible for governing a cluster by assigning applications to the underlying resources. Resources like bandwidth and memory are orchestrated by the resource manager to the underlying node managers [14]. Applications that run within YARN are managed by the ApplicationMaster. YARN allocates resources through ApplicationMasters and monitors the underlying applications through node managers. ApplicationMasters are responsible for execution of containers and negotiation of resources from the resource manager. They are assumed as buggy as they are user code and a security issue.

The node manager manages the nodes within a cluster by providing per-node services within that cluster. YARN uses the data nodes and name nodes from HDFS layer. Data node is used for replicated storage services across a cluster while the name node is used for metadata services. Execution of YARN is initiated by a client application that sends a request. ApplicationMaster is then triggered by resource manager to represent the application.

In the cluster, the ApplicationMaster negotiates containers for the application at each node by making use of a resource-request protocol. After the completion of the application, it unregisters the containers from the resource manager. YARN improves the ability to scale Hadoop clusters to large confiurations by reducing the overhead on resource manager and making the ApplicationMaster responsible for the management of job execution. Moreover, it allows a parallel execution of different programming models like machine learning and graph processing.

YARN allows users to create distributed applications which are more complex than the ones developed by the traditional MapReduce paradigm. It provides a scope for customized development by exposing the underlying framework [14]. This makes it more robust and it does not need to be segregated from other distributed frameworks that reside on the cluster. YARN frees up resource overhead that has been dedicated to the distributed frameworks which simplifies the complexity of the overall system.

As YARN provides customized development, it becomes more difficult to build YARN applications. This is due to the development of ApplicationMaster which is required after launching resource manager on a client request. YARN initially allocates a certain number of resources within a cluster. It processes the application and provides touchpoints to monitor the progress of the application. After this process, it releases resources and performs a cleanup when the finds the status of the application as complete. YARN provides many services which are beyond the scope of traditional MapReduce.

4. DATASET

Airlines delay data set is used as the data for analysis. It is analyzed using Pyspark. It is published by the United States department of transportation as the flight related information. This data is free for anyone to use and analyze. Here, we get flight arrival and departure times and delays for all flights taking off in a certain period.

Data is obtained by mentioning a year or a period of time within which the flight information is required. Three files containing this information namely airlines.csv, airports.csv and flights.csv are available in the form of a zip file. The flights.csv file contains the following fields: Flight ID, airline, airport, departure, arrival and delay. Airlines.csv has airline ID and airline name. The airports.csv file consists of airport ID and airport name. These files are placed in the local file system or in HDFS. The spark program reads the files from either location. If the files are placed in HDFS, "hdfs://" is to be given as a prefix to the file path.

5. DEPLOYMENT

The deployment process is driven by Ansible playbooks and Cloudmesh Client commands and scripts. The process is initiated on user's local Ubuntu instance. The commands are executed in local machine as well as virtual machines on the cloud.

- Cloudmesh Client is used to access multiple clouds from the command line. This makes it easier to switch to another cloud in case of a failure.
- After the Cloudmesh Client installation, ssh key is to be added to the Cloudmesh database and uploaded to all the active clouds.
- The configuration file of the Cloudmesh Client is to be modified by making Chameleon and Jetstream as active clouds.
- Security rules are then added to the user's security profile after which security group is uploaded to communicate with the virtual machines.
- A virtual cluster is to be created on the cloud by specifying the number of nodes.

Cloudmesh provides one line command for doing so. In order to make use of the nodes, floating IPs need to be assigned to the created nodes. Cluster creation fails when the cloud runs out of floating IPs. Floating IP is required for the communication between the servers and ssh from the client to the cloud. A Hadoop cluster is defined on top of the cluster we defined. Table 2 shows the resources on the cloud that have been used.

Table 2. Resources on clouds

	Chameleon	Jetstream
Flavor	m1.medium	m1.medium
OS	Ubuntu 14.04	Ubuntu 14.04
secgroup	default	default
Nodes	3	3

- Similar to the cluster previously defined, multiple specifications can be defined for the Hadoop cluster and one specification has to be activated.
- After this, Hadoop cluster can be deployed by synchronizing the Big Data stack.

- To use Spark as an add-on in the Hadoop cluster, Spark is to be passed as an argument while defining the Hadoop cluster.
- The details of the specification of the Hadoop cluster can be viewed by using the "cm hadoop avail" command.
- The Spark cluster can then be deployed by using "cm hadoop sync" and "cm hadoop deploy" commands.
- The process of uploading the data set to HDFS and running the Spark program on the uploaded data set is automated through an Ansible script.
- Installing the analysis code into the repository is also automated.

6. ANALYSIS

Airlines dataset publicly available from US Government website is used for performing analysis and finding out various insights. The dataset is downloaded by identifying the goals to be accomplished. The dataset consists of three files, they are flights.csv, airlines.csv, airports.csv. The flights.csv has key informtion like the departure, delay of various airlines and airports. The airlines.csv and airports.csv files contains the code for an airline and airport respectively and their corresponding names. The airline and airport files can be used as lookup files. The flight data is the key for the analysis.

Initially, the flight data is parsed to create a flight tuple with all the fields in the flight class to be members of the named tuple, just like class and its objects. The following functions are implemented they are parse, split and notHeader. Parse is used to parse each indiviual row in the flights.csv and convert it into a named tuple. The split function is used to split each column value in a row based on comma since the dataset is comma seperated values. After loading the SparkContext, the airlines data is parsed to eliminate the header and split it accordingly.

Similarly, the airports data is parsed. The flights data is parsed such that each individual row is converted into a named tuple. By using this parsed data, output is obtained by performing various transformations and actions.

The process is to transform the flights RDD by applying filters and map functions for getting the delay based on two instances i.e, airports and airlines. Then ReduceByKey and CombineByKey actions are performed by aggregating and computing the average delay in case of each airport and sorting functions are applied to sort the output in descending order and based on that, the top ten airports to avoid are obtained given the average delay per airport. Since the codes of various airlines and airports are the only ones available, lookup operations are performed by using countAsMap() operation with airports and airlines dataset and by using broadcast, the lookup information is passed on to all workers and executors within them.

The top ten airlines to avoid are found by computing the total minutes of delay per airline over a period of time. Figure 4 shows the top ten airlines obtained by the analysis. The analysis can be further improvised by using various Machine Learning techniques which helps in predicting the delays over a period of time ahead and provides various insights which are helpful in making better decisions in choosing airlines and airports to commute.

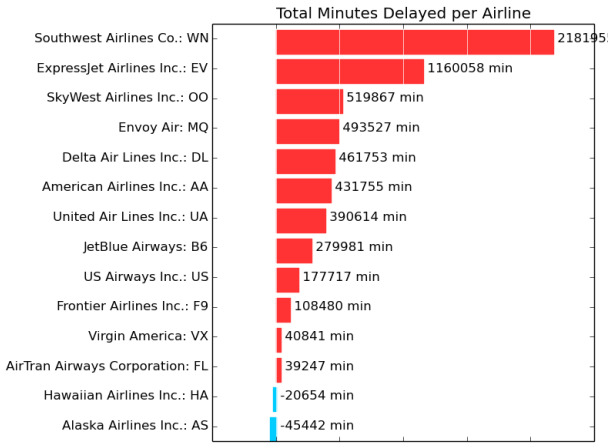


Fig. 4. Top ten airlines to avoid

7. BENCHMARKING

Benchmarking is carried out after the deployment and data analysis. It is done to evaluate the efficiency and performance of the system. As a part of benchmarking, the flight and airline data analysis performed is evaluated by running the code on Jetstream and Chameleon cloud computing environments.

The data is transferred to Hadoop Distributed file system and analysis is done by using Spark with YARN. To evaluate the efficiency of the analyis, the dataset has been used in varying sizes. Datasets with increasing size in rows have been considered for this purpose. As it is known that the transformations in Spark are lazy, the results are not evaluated right away. This makes it even more efficient.

Python's time module is used for obtaining the current time. Time for running the analysis is found out by determining timestamps both before and after running the code, named beforeTime and afterTime respectively. The required time is determined by subtracting beforeTime from afterTime. Due to Spark's lazy evaluation, the time module used is wrapped around the Spark actions. Figure 5 shows the time, in seconds, taken by the analysis in Jetstream cloud computing environment with different dataset sizes.

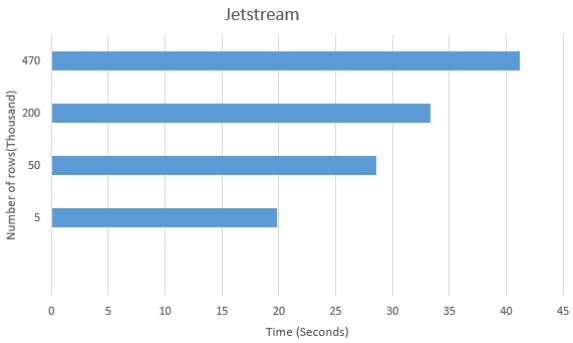


Fig. 5. Performance on Jetstream

Figure 6 shows the performance of the analysis on Chameleon cloud environment.

As the analyis code, data and packages are installed on the clusters through Ansible playbook, the time taken for the au-

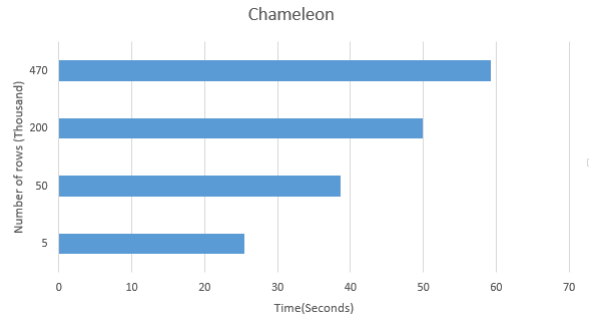


Fig. 6. Performance on Chameleon

tomation on Chameleon and Jetstream clouds is determined and the values obtained on each cloud are compared. The comparison is shown in figure 7.

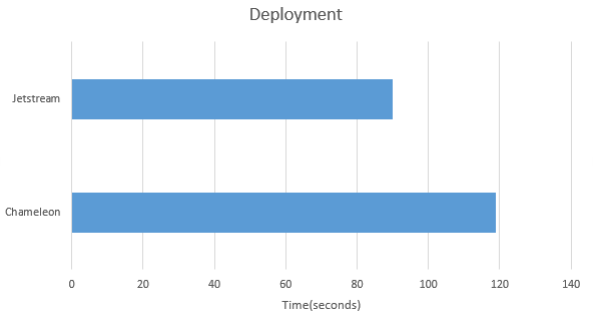


Fig. 7. Deployment in Chameleon and Jetstream

8. TIMELINE

Week by week timeline for project completion is specified in this section.

1. March 6 - March 12, 2017: Created virtual machines on Chameleon cloud using Cloudmesh.
2. March 13 - March 19, 2017: Deployed Hadoop cluster to Chameleon cloud using Cloudmesh.
3. March 20 - March 26, 2017: Acquired data for performing analysis and submitted the project proposal.
4. March 27 - April 02, 2017: Created virtual machines on Jetstream cloud using Cloudmesh and deployed Hadoop cluster to the cloud using Cloudmesh.
5. April 03 - April 09, 2017: Perfomed analysis on the data using Apache Spark on top of Hadoop stack.
6. April 10 - April 16, 2017: Developed Ansible playbook to deploy Hadoop and Spark to the cloud machines.
7. April 17 - April 23, 2017: Completed project report and developed benchmarks for the project.

9. WORK BREAKDOWN

Below is the work distribution for the implementation, testing and documentation of the project.

- Bhavesh Reddy Merugureddy
  - Creating and deploying clusters on Jetstream.
  - Acquiring the data and performing analysis on flight data.
  - Writing transformations and actions required for the analysis using Spark.
  - Setting up and testing the end to end flow on Jetstream cloud.
  - Performing benchmarking for the analysis on Jetstream by varying the data set size.
  - Writing related sections in this report.
- Niteesh Kumar Akurati
  - Creating and deploying clusters on Chameleon.
  - Collecting airport data and performing analysis.
  - Implementation of Ansible scripts for deployment of code, data and the required packages.
  - Setting up and testing the end to end flow on Chameleon cloud.
  - Performing benchmarking for the analysis on Chameleon by varying the data set size.
  - Writing related sections in this report.

10. CONCLUSION

Airline delays data has been analyzed by the deployment of Hadoop and Spark on Chameleon and Jetstream cloud computing environments. A publicly available data set containing flight and airline related data is taken for analysis. Cloudmesh Client is used as the cloud manager to access multiple clouds and deploy a Hadoop cluster on Chameleon and Jetstream clouds. Ansible scripts are written for extracting data sets from the published zip file and deploying them on the clouds. Hadoop Distributed File System is used to store the extracted data sets. A program is written in Spark to perform the data analysis which is deployed by using an Ansible script. Bar graphs are drawn for the analysis performed. Apart from the deployment and analysis, benchmarking is done to evaluate the performance of the program on each node of a cluster and on different clouds. The efficiency of the program is determined by varying the sizes of the data set and comparing the results.

11. ACKNOWLEDGEMENTS

This project is undertaken as part of the I524: Big Data and Open Source Software Projects coursework at Indiana University. We would like to thank our Prof. Gregor von Laszewski, Prof. Gregory Fox and the Associate Instructors for their help and support.

REFERENCES

[1] G. von Laszewski, "Cloudmesh client toolkit," webpage, 2015. [Online]. Available: http://cloudmesh-client.readthedocs.io/en/latest/
[2] "About chameleon," webpage. [Online]. Available: https://www.chameleoncloud.org/about/chameleon/
[3] P. Lindenlaub, "System overview," webpage. [Online]. Available: https://iujetstream.atlassian.net/wiki/display/JWT/System+Overview
[4] "Ansible(software)," webpage. [Online]. Available: https://en.wikipedia.org/wiki/Ansible_(software)

[5] R. Ahmed, "What is ansible? – configuration management and automation with ansible," webpage. [Online]. Available: https://www.edureka.co/blog/what-is-ansible/
[6] "Apache spark," webpage. [Online]. Available: https://en.wikipedia.org/wiki/Apache_Spark
[7] "Running spark applications on yarn," webpage. [Online]. Available: https://www.cloudera.com/documentation/enterprise/5-6-x/topics/cdh_ig_running_spark_on_yarn.html
[8] A. Sethi, "Introduction to accumulators : Apache spark," webpage. [Online]. Available: https://blog.knoldus.com/2016/05/13/introduction-to-accumulators-apache-spark/
[9] "Accumulators," webpage. [Online]. Available: https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-accumulators.html
[10] "Apache hadoop hdfs," webpage, 2017. [Online]. Available: https://hortonworks.com/apache/hdfs/
[11] H. J, "An introduction to the hadoop distributed file system," webpage. [Online]. Available: https://www.ibm.com/developerworks/library/wa-introhdfs/
[12] "Hadoop - hdfs overview," webpage. [Online]. Available: https://www.tutorialspoint.com/hadoop/hadoop_hdfs_overview.htm
[13] M. Rouse, "Apache hadoop yarn (yet another resource negotiator)," webpage. [Online]. Available: http://searchdatamanagement.techtarget.com/definition/Apache-Hadoop-YARN-Yet-Another-Resource-Negotiator
[14] M. Nelson and M. Jones, "Moving ahead with hadoop yarn," webpage. [Online]. Available: https://www.ibm.com/developerworks/library/bd-hadoopyarn/

Deployment of a Storm cluster

VASANTH METHKUPALLI[1,*] AND AJIT BALAGA[1,**]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*] Corresponding authors:mvasanthiiit@gmail.com
[**] Corresponding authors: ajit.balaga@gmail.com

project-P015, May 4, 2017

Apache Storm is a free and open source distributed realtime computation system. Storm makes it easy to reliably process unbounded streams of data, doing for realtime processing what Hadoop did for batch processing. Storm is simple, can be used with any programming language. Storm has many use cases: realtime analytics, online machine learning, continuous computation, distributed RPC etc. Storm is really fast at data processing: a sample benchmark clocked it at over a million tuples processed per second per node. It is scalable, fault-tolerant, guarantees that data will be processed, and is easy to set up and operate. Storm integrates with the queueing and database technologies we already use. Storm is currently being used to run various critical computations in Twitter at scale, and in real-time, this led us to explore and deploy it on various cloud. In this paper we try to deploy storm on various clouds and benchmark the performance on various data, doing real time processing on sample datasets. First, we describe the architecture of Storm, its deployment and its methods for distributed scaleout and fault-tolerance. Storm is in active development at Twitter.

Keywords: Storm, Ansible, Java, Python

https://github.com/cloudmesh/classes/blob/master/project/S17-IR-P015/report/report.pdf

CONTENTS

A. INTRODUCTION

Currently modern data processing environments require processing complex computation on streaming data in real-time. Places like Twitter where each interaction with a user requires making a number of complex decisions, often based on data that has just been created, this mandates for a real time data processing system, Storm currently delivers on this account and provides many other services which we will see in the coming sections. Storm is designed to be, some of these things we observed while running our sample projects for deployment:

1. Scalable : Nodes may be easily added or removed from the Storm cluster without disrupting existing data flows through Storm topologies see Fig 3.

2. Resilient : Fault -tolerance is crucial to Storm as it is often deployed on large clusters, and hardware components can fail. The Storm cluster must continue processing existing topologies with a minimal performance impact.

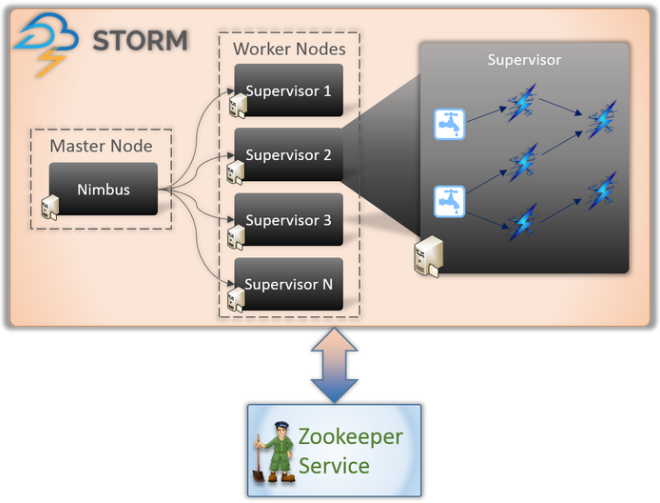


Fig. 1. Storm Architecture

3. Extensible : Storm topologies may call arbitrary external functions (e.g. looking up a MySQL service for the social graph), [1] and thus needs a framework that allows extensibility.

4. Efficient : Since Storm is used in real-time applications, it must have good performance characteristics. Storm uses a number of techniques, including keeping all its storage and computational data structures in memory.

5. Easy to Administer : A critical part of storm features or development is that it should be easy to administer. Given that there a lot of computations going on at every stage, tools should be developed which warn the user and development team of any major conflicts arising.

B. DATA MODEL AND ARCHITECTURE

Storm data processing architecture consists of streams of tuples flowing through topologies [2] . A topology is a directed graph where the vertices represent computation and the edges represent the data flow between the computation components. Vertices are further divided into two disjoint sets, spouts and bolts. Spouts are tuple sources for the topology. Typical spouts pull data from queues, such as Kafka [3] or Kestrel. On the other hand, bolts process the incoming tuples and pass them to the next set of bolts downstream. Note that a Storm topology can have cycles. From the database systems perspective, one can think of a topology as a directed graph of operators.

B.1. Storm Overview

Storm runs on a distributed cluster. Clients submit topologies to a master node, called the Nimbus. The nimbus is responsible for distributing and coordinating the execution of the topology. The actual work is done on worker nodes. Each worker node runs one or more worker processes. At any point in time, a single machine may have more than one worker processes, but each worker process is mapped to a single topology. Note more than one worker process on the same machine may be executing different part of the same topology. Each worker process runs a JVM, in which it runs one or more executors. Executors are made of one or more tasks. The actual work for a bolt or a spout is done in the task. Thus, tasks provide intrabolt or intraspout parallelism, and the executors provide intratopology parallelism. Worker processes serve as containers on the host machines to run Storm topologies. Note that associated with each spout or bolt is a set of tasks running in a set of executors across machines in a cluster. Data is shuffled from a producer spout or bolt to a consumer bolt (both producer and consumer may have multiple tasks). This shuffling is like the exchange op erator in parallel databases.

Storm supports the following types of partitioning strategies [2]:

1. Shuffle grouping, which randomly partitions the tuples.

2. Fields grouping, which hashes on a subset of the tuple attributes or fields.

3. All grouping, which replicates the entire stream to all the consumer tasks.

4. Global grouping, which sends the entire stream to a single bolt.

5. Local grouping, which sends tuples to the consumer bolts in the same executor.

The partitioning strategy is extensible and a topology can define and use its own partitioning strategy. Each worker node runs a Supervisor that communicates with Nimbus. The cluster state is maintained in Zookeeper [4], and Nimbus is responsible for scheduling the topologies on the worker nodes and monitoring the progress of the tuples flowing through the topology. Loosely, a topology can be considered as a logical query plan from a database systems perspective. As a part of the topology, the programmer specifies how many instances of each spout and bolt must be spawned. Storm creates these instances and also creates the interconnections for the data flow. We note that currently, the programmer has to specify the number of instances for each spout and bolt. Part of future work is to automatically pick and dynamically changes this number based on some higher-level objective, such as a target performance objective.

B.1.1. Storm Internal Architecture

In this section, we describe the key components of Storm), and how these components interact with each other.

B.1.2. Nimbus and Zookeeper

Nimbus plays a similar role as the “JobTracker” in Hadoop, and is the touchpoint between the user and the Storm system. Nimbus is an Apache Thrift service and Storm topology definitions are Thrift objects. To submit a job to the Storm cluster (i.e. to Nimbus), the user describes the topology as a Thrift object and sends that object to Nimbus. With this design, any programming language can be used to create a Storm topology.

As part of submitting the topology, the user also uploads the user code as a JAR file to Nimbus. Nimbus uses a combination of the local disk(s) and Zookeeper to store state about the topology. Currently the user code is stored on the local disk(s) of the Nimbus machine, and the topology Thrift objects are stored in Zoo keeper.

The Supervisors contact Nimbus with a periodic heartbeat protocol, this comes in very handy so we can periodically verify ifthe nodes are working, advertising the topologies that they are currently running, and any vacancies that are available to run more topologies. Nimbus keeps track of the topologies that need

assignment, and does the match -making between the pending topologies and the Supervisors.

All coordination between Nimbus and the Supervisors is done using Zookeeper. Furthermore, Nimbus and the Supervisor daemons are fail- fast and stateless, and all their state is kept in Zookeeper or on the local disk(s). This design is the key to Storm's resilience. If the Nimbus service fails, then the workers still continue to make forward progress. In addition, the Supervisors restart the workers if they fail.

However, if Nimbus is down, then users cannot submit new topologies. Also, if running topologies experience machine failures, then they cannot be reassigned to different machines until Nimbus is revived. An interesting direction for future work is to address these limitations to make Storm even more resilient and reactive to failures. All the above workings, whether Nimbus and Zookeeper are working properly are viewed in the UI of the storm deployment, and can be viewed from the localhost of that particular node.

B.1.3. Supervisor

The supervisor runs on each Storm node. It receives assignments from Nimbus and spawns workers based on the assignment. It also monitors the he alth of the workers and respawns them if necessary. The main thread reads the Storm configuration, initializes the S upervisor's global map, creates a persistent local state in the file system, and schedules recurring timer events. There are three types of events, which are:

1. The heart beat event, which is scheduled to run every 15 seconds, and is runs in the context of the main thread. It reports to Nimbus that the supervisor is alive.

2. The synchronize supervisor event, which is executed every 10 seconds in the event manager thread. This thread is responsible for managing the changes in the existing assignments. If the changes include addition of new topologies, it downloads the necessary JAR files and libraries, and immediately schedules a synchronize process event .

3. The synchronize process event, which runs every 3 seconds under the context of the process event manager thread. This thread is responsib le for managing worker processes that run a fragment of the topology on the same node as the supervisor. It reads worker heartbeats from the local state and classifies those workers as either valid, timed out, not started, or disallowed. A "timed out" work er implies that the worker did not provide a heart beat in the specified time frame, and is now assumed to be dead. A "not started " worker indicates that it is yet to be started because it belongs to a newly submitted topology, or an existing topology who se worker is being moved to this supervisor. Finally, a "disallowed " worker means that the worker should not be running either because its topology has been killed, or the worker of the topology has been moved to another node.

C. MILESTONES

- Previous Project idea brainstorming
- Performing Analysis on local VM- April 17th, 2017
- Deploying Storm and Hadoop on Chameleon Cloud and Jetstream- April 20th, 2017
- Analysis on the distributed cloud environment- April 27th, 2017
- Benchmarking- April 27th, 2017
- Final update with report- May 1st, 2017

D. TECHNOLOGIES

Usage	Technologies Used
Distributed Computation and Storage:	Storm
Development:	Python and Java
Deployment:	Ansible, Bash Shell script
Project Repository:	GitHub
Document Preparation:	LaTex

E. DEPLOYMENT AUTOMATION

E.1. Automation with Ansible

Ansible Playbook is used as the application and configuration deployment tool. Deploying the hadoop and spark framework into the cluster environment. Ansible will help push configurations to the environment automatically based on playbooks written for various configurations. For this project, we used Ansible to automate the deployment of storm, zookeeper and other prequisites. The Ansible script is written such that we can leverage the cloudmesh client technology to deploy the spark cluster. When creating Zookeeper cluster we had a communication issue, similar issue was found when creating a Storm cluster, we resolved this issue by uploading a secgroup with the details necessary and added it to the cluster.

The Ansible playbook package constitutes the following files.

- ansible.cfg: This file contains all the configuration information necesasary for the storm deployment. In our project we refer to the hosts file to look up for the ips necessary for deployment.
- hosts: Efforts are made to automate the generation of this file, however currently the user after creating a cluster in the cloudmesh client has to update the information the user ip addresses and id's in three columns, chameleon, nimbus and supervisors fields. This enables the user to select the ip address in which they desire to have the nimbus node and all other configuration.
- install.yml: However, the above step is one of the few steps we had to configure manually, however, running the command ansible-playbook install.yml, installs all the dependencies and packages necessary for strom deployment. However, the install sets up the cluster with supervisor and nimbus nodes. The startup of these services however are to be done manually by the below bash scripts.
- nimbus.sh: This script is intended to start the nimbus node to receive all the packets(data) for processing.
- startStorm.sh: This starts the storm cluster.
- submit.sh Used to submit jobs to run, in the storm cluster.
- start-zookeeper.yml: This file is still in developmental stage where we want to automate the above two steps, however, there is a small problem in exiting the storm cluster after once it has started. We are working on it to fix this.

- supervisor.sh: This script file is used to start the worker nodes on which the data processing can run
- ui.sh: Running this script file will open the localhost, where we can view the current jobs running, topologies, cluster information, spouts, bolts, etc. Sample screenshots showing the working of this are shown in the benchmarking figures.
- In the templates folder, we have hosts.j2, myid.j2, storm.yaml.j2, zoo.cfg.j2, these files are necessary for proper deployment of storm and zookeeper cluster, so we created templates of the configuration files and intend to use them.

E.2. Bash Shell Script

First we automated the entire deployment using python shell script, and later changed the deployment by using Ansible playbooks. In the python shell script, we have mainly used 4 files(.sh) to automate the entire process, from updating the apps directory to configuring the host file in the zookeeper cluster. As a part of our project we are submitting even the python shell script files for everyone to view the initial stages of project development.

F. CLOUDMESH

Cloudmesh client is a simple client to enable access to multiple cloud environments form a command shell and command-line[reference for cloudmesh].The entire application is built on python which essentially needs no prerequisite knowledge and hence could see this as a ready-to-use tool. For our project since we need access to clouds for deployment purposes this comes as a welcome tool to ease us with the entire deployment process. Many thanks to Gregor von Laszewski and others collaborators for supporting our project directly and indirectly in the form of this tool which enabled us to deploy the project easier without any hassles. For this project, have tested the installation on Ubuntu 16.04.

G. CLOUD DEPLOYMENT

We selected two clouds for deployment our project: Chameleon Cloud, and Jetstream. In our automated deployment and benchmarking process, first we create a cluster of particular number of required nodes, update the ip addresses and the user id of all the clouds, then we run the ansible script to deploy the entire project on the cloud. However, lot of improvements can be made, such as automating the process of ip addresses for the cluster deployment and exiting the cloud by deleting the files and the cloud resources once deployed and benchmarked. The entire process of deploying the project on various clouds have been tried and tested, various times were measure on the context of installation time right from the beginning. Based on the above results we benchmarked our results on various cluster sizes.

H. BENCHMARKING

H.1. Chameleon Cloud

Following are the results of our benchmarking on deploying the storm cluster on Chameleon cloud. Few things, we noticed that a 3-node cluster is not viable for running storm jobs and hence we ran all our jobs on 5-nodes or above clusters, this was one error we came across when we tried deploying a 3-node cluster and overcame this by following code blogs related to troubleshooting mentioned in the code references and solved the issue. We ran a sample-storm job on a 5node cluster to see if its working, following are the snapshots where in we can see the UI of storm, showing all the resources used among many other things. Similar tests were made on a 7-node and a 9-node cluster and results were stored in a log page.

Table 1. Hardware Specifications of Chameleon

	Chameleon
CPU	Xeon X5550
cores	1008
speed	2.3GHz
RAM	5376GB
storage	1.5PB

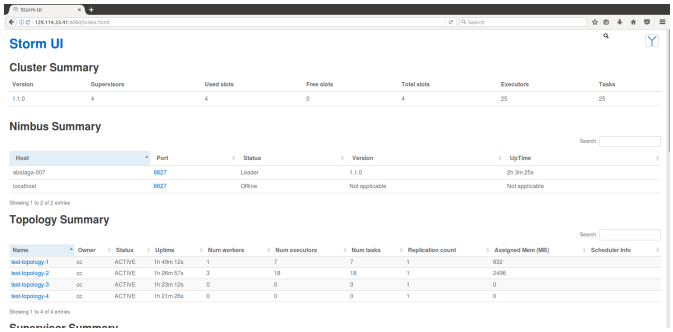


Fig. 2. Index Page

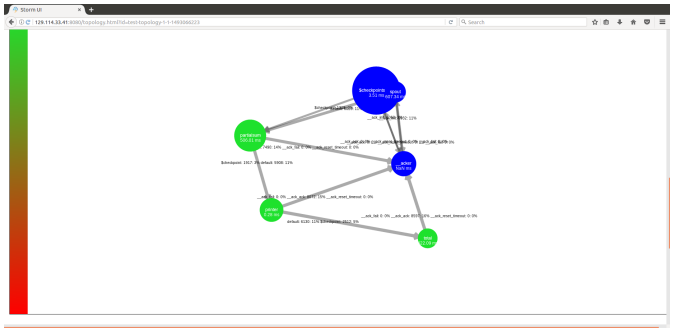


Fig. 3. Sample Topology on a 5-node cluster

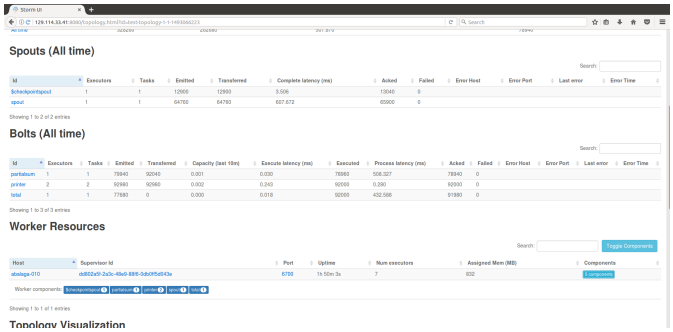


Fig. 4. Worker nodes and description on a 5-node cluster

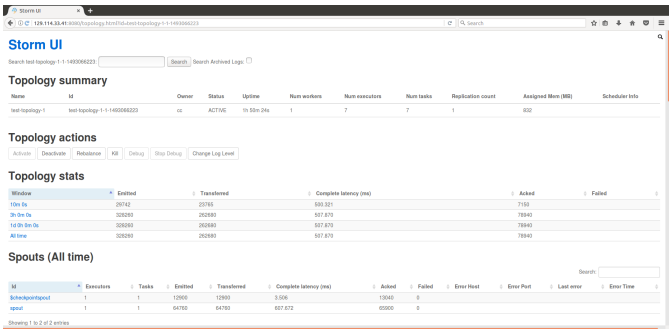


Fig. 5. Toplogy Summer on a 5-node cluster

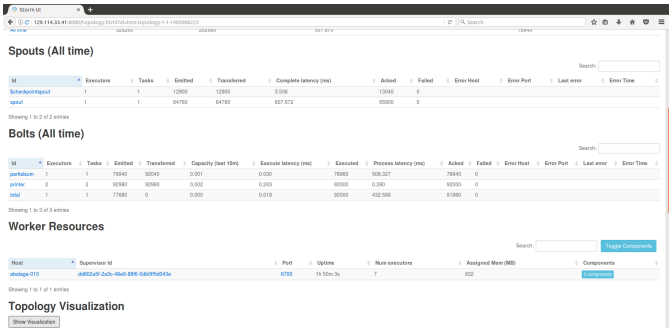


Fig. 6. Spouts and bolts summary on a 5-node cluster

Also we calculated the time it took to deploy a 5-node, 7-node and a 9-node cluster on the chameleon cloud. Following are the results obtained and can be illustrated below.

Installation Time	5 Node cluster	7 Node cluster	9 Node Cluster
Real	143.91	188.19	387.536
User	13.928	28.432	36.192
Sys	3.964	8.872	9.988

Fig. 7. Table showing the time taken to deploy

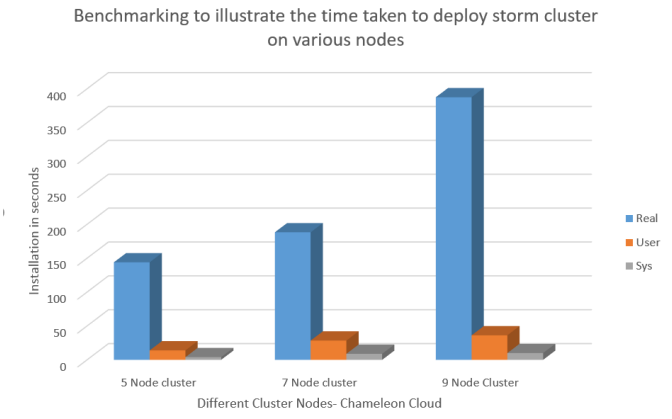


Fig. 8. Bar diagram to compare the time taken to deploy

H.2. JetStream

We had slight issues booting the Virtual Machines on the jetstream cloud. In that scenario we decided to go ahead and deploy the cluster only on the chameleon cloud and test its performance. However, it can be noted that the deployment is fairly similar even in the case of JetStream using the Cloudmesh client.

I. CODE REFERENCES

Following were the code resources used in deploying the cluster at various stages.

- Storm-[5][6][7][8][9][10][11][12][13][14]
- Zookeeper-[15][16][17][18][19][20][21][22]
- Running Topologies on a cluster-[23][24][25][26][27][28]
- Zookeeper Troubleshooting-[29][30][31][32][33][34]
- Storm Troubleshooting-[35][36][37][38]
- Storm defaults yaml-[39]

J. SUMMARY

We have created, tested and demonstrated a fully automated program to configure and deploy a Storm cluster which can be deployed on cloud environments. We currently deployed it on Chameleon cloud and expect to do the same with any other cloud environment with slight changes in the ansible script.Also, another thing we observed while deploying a Zookeeper cluster that installing it in standalone mode makes it prone to breakdown, since if the Zookeeper cluster fails in this scenario, the entire storm cluster fails to run, however installing it on server mode leads to the scenario where in if one or more clusters fail to run, the load is handled by the other nodes in the cluster. We used a combination of Python, bash, cloudmesh client and ansible in fully deploying the cluster. We did a benchmark test on Chameleon cloud to measure the time taken to deploy 5, 7 and 9 node clusters, so far it has shown satisfactory results and we are striving to take it beyond to other cloud environments as well.

K. ACKNOWLEDGMENTS

This project was a part of the Big Data Software and Projects (INFO-I524) course. We would like to thank Professor Gregor von Laszewski and the associate instructors for their help and support during the course. Special mention to cloudmesh client which made most of the gruelling taks straighforward.

REFERENCES

[1] N. Bronson, Z. Amsden, G. Cabrera, P. Chakka, P. Dimov, H. Ding, J. Ferris, A. Giardullo, S. Kulkarni, H. C. Li *et al.*, "Tao: Facebook's distributed data store for the social graph." pp. 49–60, 2013.

[2] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham *et al.*, "Storm@ twitter," ACM, pp. 147–156, 2014.

[3] J. Kreps, N. Narkhede, J. Rao *et al.*, "Kafka: A distributed messaging system for log processing," pp. 1–7, 2011.

[4] "Apache ZooKeeper - Home." [Online]. Available: https://zookeeper.apache.org/

[5] "Setting up a Storm Cluster." [Online]. Available: http://storm.apache.org/releases/1.0.3/Setting-up-a-Storm-cluster.html

[6] "How to Deploy Apache Storm on AWS | Cloud Academy." [Online]. Available: http://cloudacademy.com/blog/how-to-deploy-apache-storm-on-aws/

[7] "Deploy and manage Apache Storm topologies on Linux-based HDInsight | Microsoft Docs." [Online]. Available: https://docs.microsoft.com/en-us/azure/hdinsight/hdinsight-storm-deploy-monitor-topology-linux

[8] "Running a Multi-Node Storm Cluster - Michael G. Noll." [Online]. Available: http://www.michael-noll.com/tutorials/running-multi-node-storm-cluster/
[9] "How to Deploy an Apache Storm Cluster to the Amazon Elastic Compute Cloud (EC2) – Knowm.org." [Online]. Available: http://knowm.org/how-to-deploy-an-apache-storm-cluster-to-the-amazon-elastic-compute-cloud-ec2/
[10] "nathanmarz/storm-deploy: One click deploy for Storm clusters on AWS." [Online]. Available: https://github.com/nathanmarz/storm-deploy
[11] "java - Apache Storm Topology deployment - Stack Overflow." [Online]. Available: http://stackoverflow.com/questions/34037377/apache-storm-topology-deployment
[12] "Apache Storm Cluster Architecture." [Online]. Available: https://www.tutorialspoint.com/apache_storm/apache_storm_cluster_architecture.htm
[13] "Apache Storm: What We Learned About Scaling." [Online]. Available: https://www.loggly.com/blog/what-we-learned-about-scaling-with-apache-storm/
[14] "How Spotify Scales Apache Storm | Labs." [Online]. Available: https://labs.spotify.com/2015/01/05/how-spotify-scales-apache-storm/
[15] "ZooKeeper Administrator's Guide." [Online]. Available: https://zookeeper.apache.org/doc/r3.3.2/zookeeperAdmin.html
[16] "ZooKeeper Cluster (Multi-Server) Setup | myjeeva blog." [Online]. Available: https://myjeeva.com/zookeeper-cluster-setup.html
[17] "Howto Setup Apache Zookeeper Cluster on Multiple Nodes in Linux." [Online]. Available: http://www.thegeekstuff.com/2016/10/zookeeper-cluster-install/
[18] "Setting up Apache ZooKeeper Cluster | Apache ZooKeeper Tutorials." [Online]. Available: http://www.allprogrammingtutorials.com/tutorials/setting-up-apache-zookeeper-cluster.php
[19] "Clustering Zookeeper." [Online]. Available: http://www.mastertheintegration.com/core-apache-projects/zookeeper/clustering-zookeeper.html
[20] "How To Setup a Zookeeper Cluster - Beginners Guide." [Online]. Available: https://devopscube.com/how-to-setup-a-zookeeper-cluster/
[21] "Cluster(Multi server) setup for zookeeper exhibitor - Stack Overflow." [Online]. Available: http://stackoverflow.com/questions/17343475/clustermulti-server-setup-for-zookeeper-exhibitor
[22] "Zookeeper Quick Guide." [Online]. Available: https://www.tutorialspoint.com/zookeeper/zookeeper_quick_guide.htm
[23] "Running Topologies on a Production Cluster." [Online]. Available: http://storm.apache.org/releases/1.0.3/Running-topologies-on-a-production-cluster.html
[24] "java - How to submit a topology in storm production cluster using IDE - Stack Overflow." [Online]. Available: http://stackoverflow.com/questions/15781176/how-to-submit-a-topology-in-storm-production-cluster-using-ide
[25] "Deploy and manage Apache Storm topologies on Linux-based HDInsight | Microsoft Docs." [Online]. Available: https://docs.microsoft.com/en-us/azure/hdinsight/hdinsight-storm-deploy-monitor-topology-linux
[26] "Setting up Storm and Running Your First Topology | Harold Nguyen's Blog." [Online]. Available: http://www.haroldnguyen.com/blog/2015/01/setting-up-storm-and-running-your-first-topology/
[27] "Running Topologies on a Storm Cluster | CoreJavaGuru." [Online]. Available: http://www.corejavaguru.com/bigdata/storm/running-topologies-on-a-cluster
[28] "Is there a way to create Dynamic Topologies in Apache Storm? - Hortonworks." [Online]. Available: https://community.hortonworks.com/questions/70650/is-there-a-way-to-create-dynamic-topologies-in-apa.html
[29] "Zookeeper - three nodes and nothing but errors - Stack Overflow." [Online]. Available: http://stackoverflow.com/questions/11498507/zookeeper-three-nodes-and-nothing-but-errors
[30] "Zookeeper can not run becuase of Invalid config - Stack Overflow." [Online]. Available: http://stackoverflow.com/questions/37738826/zookeeper-can-not-run-becuase-of-invalid-config
[31] "zookeeper-user - Starting zookeeper in replicated mode." [Online]. Available: http://zookeeper-user.578899.n2.nabble.com/Starting-zookeeper-in-replicated-mode-td5205720.html
[32] "Solved: Re: zookeeper gets permission problem on /var/lib/... - Cloudera Community." [Online]. Available: http://community.cloudera.com/t5/Cloudera-Manager-Installation/zookeeper-gets-permission-problem-on-var-lib-zookeeper/m-p/30749
[33] "java - what is zookeeper port and its usage? - Stack Overflow." [Online]. Available: http://stackoverflow.com/questions/18168541/what-is-zookeeper-port-and-its-usage
[34] "zookeeper-user - Problem with leader election." [Online]. Available: http://zookeeper-user.578899.n2.nabble.com/Problem-with-leader-election-td7579173.html
[35] "Apache Storm: Could not find leader nimbus from seed hosts - Stack Overflow." [Online]. Available: http://stackoverflow.com/questions/36742451/apache-storm-could-not-find-leader-nimbus-from-seed-hosts
[36] "Solutions for Storm Nimbus Failure - Hortonworks." [Online]. Available: https://community.hortonworks.com/articles/8844/solutions-for-storm-nimbus-failure.html
[37] "Apache Storm is not running properly and worker.log generating exceptions - Stack Overflow." [Online]. Available: http://stackoverflow.com/questions/38551229/apache-storm-is-not-running-properly-and-worker-log-generating-exceptions
[38] "java - Apache Storm Supervisor not Running the Bolt - Stack Overflow." [Online]. Available: http://stackoverflow.com/questions/28667548/apache-storm-supervisor-not-running-the-bolt
[39] "storm/defaults.yaml at master · apache/storm." [Online]. Available: https://github.com/apache/storm/blob/master/conf/defaults.yaml

Predicting Customer Churn Using Apache Spark Machine Learning

YATIN SHARMA, DIKSHA YADAV[1,*]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*] Corresponding authors: yatins@indiana.edu, yadavd@iu.edu

project-001, May 4, 2017

This report presents a reference implementation to the Customer Churn Prediction Project that is built using Apache Spark Machine Learning. In this report, we discuss the models used for predicting customer churn along with their accuracies. We will also discuss the software deployment on Chameleon and Jetstream cloud computing environments using Ansible and Cloudmesh Client scripts.

Keywords: Prediction, Bigdata, Apache Spark, ML, Hadoop, ,Ansible, Cloudmesh, Analytics

https://github.com/cloudmesh/sp17-i524/tree/master/project/S17-IR-P016/report/report.pdf

CONTENTS

1. INTRODUCTION

One of the most important business metrics is churn rate- describing the rate at which your customers stop doing business with you. Our aim is to build a predictive model which uses data science to predict in advance which customers are at risk of leaving. Such model will allow the company to be proactive and focus on such customers. We used Apache Spark[1] Machine Learning library(ML)[2] for fitting a predictive model on our dataset. Detailed analysis and modeling was carried out in Python Programming language and different machine learning algorithms were tested on the same dataset. Application deployment was initially done on localhost, followed by deployment on two different clouds. Finally Performance metrics on each of the infrastructure was compared and has been included the report.

2. ARCHITECTURE

Our application was prototyped on smaller dataset locally and was then modified to run on cluster. Fig 1 below describes the runtime architecture of a distributes spark application.

In distributed mode, Spark uses a master/slave architecture with one central coordinator and many distributed workers(2 in our case). The central coordinator is called the driver. The driver communicates with the distributed worker called executors. The driver runs in its own Java Process and each executor is a separate Java process. A driver and its executers are together termed a Spark Application. A Spark application is launched on a set of machines using an external service called cluster manager. We used Hadoop YARN[4] as our cluster manager in cloud

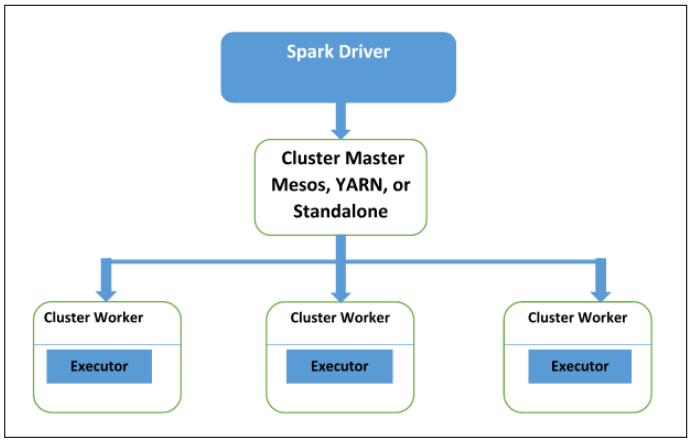


Fig. 1. Runtime architecture of spark in distributed mode[3]

deployments.

2.1. The Driver

The driver is the process where main() method of our program runs. It is the process running the user code that creates a SparkContext, creates RDDs(Resilient Distributed Dataset), and performs transformations and actions.

2.2. Executors

Spark executors are worker processes responsible for running individual tasks in a given spark job.Executors have two roles. First, they run the tasks that make up the application and return the result to the driver. Second, they provide in-memory storage for the RDDs.

2.3. Hadoop YARN

YARN is cluster manager that provides a data processing framework to run on a shared resource pool, typically on the same node as the Hadoop filesystem(HDFS)[5]. In our project, an HDFS Spark cluster was set up on cloud using Cloudmesh Client.[6] The dataset was stored on the same HDFS system which was then accessed by our spark appication for quick data processing. Using YARN in spark is also straightforward with the help of spark-submit as follows:

```
$ ./bin/spark-submit --master yarn yourapp
```

2.4. Launching a program

Spark provides a single script to submit our program to it called spark-submit. Through various options, spark-submit can connect to different cluster managers and control how many resources the application gets. For example following command can be used to launch a spark application.

```
$ ./bin/spark-submit --class path.to.your.Class
--master yarn --deploy-mode cluster [options]
<app jar> [app options]
```

3. TECHNOLOGIES

Table 1 lists the set of technologies that were used in building this project.

- **Python**: Python is a high level programming language having various popular libraries for data analysis and machine learning algorithms, making it our preferred choice fo the project.

Technology	*Usage*
Python[7]	Development
Hadoop[8]	Big Data Technology
Spark[9]	Big Data Technology
Apache Spark ML[2]	Machine Learning Library
Cloudmesh Client[6]	Cloud Resource Manager
GitHub[10]	Project Repository
Ansible[11]	Application Deployment & Configuration Management
Chameleon[12], **JetStream**[13]	Cloud deployment & Benchmarking

Table 1. Technologies used

- **Hadoop**: Apache Hadoop is a framework that allows processing and storing huge amounts of data across a distributed resource pool. Therefore, in order to improve overall performance and scalability Hadoop was installed at the foundation of each virtual machine in the cloud.

- **Spark**: Apache Spark is a cluster computing platform designed to be fast and general purpose. It extends the popular MapReduce model to efficiently support more types of computations including interactive queries and stream processing. We will primarily use Spark's machine learning library to build a predictive model on our dataset.

- **ML**: ML is Spark's primary machine learning library. It is DataFrame[14] based API that makes practical machine learning scalable. The main concepts in Spark ML are:

  1. DataFrame A DataFrame is a Dataset organized into named columns.[14]
  2. Transformer A Transformer is an abstraction that includes feature transformers and learned models. It converts one dataframe into another.[15]
  3. Estimator An Estimator is an algorithm which can be fit on a DataFrame to produce a Transformer. For example, training/tuning on a DataFrame and producing a model.
  4. Pipeline A Pipeline is specified as a sequence of stages, and each stage is either a Transformer or an Estimator. It represents a workflow, which is run in a specific order.[16]

- **Cloudmesh Client**: Cloudmesh Client provides an application programming interface(API), which allows us to manage a set of cloud resources. It standardize access to various clouds and clusters. In our project, Cloudmesh Client is used to set up Hadoop virtual cluster with Spark on two cloud environment. Following tasks are performed using Cloudmesh Client:

1. Uploading Public Key- using Cloudmesh's key add and upload commands.
2. Uploading Security Rules- using Cloudmesh's secgroup commands.
3. Creating Hadooop/Spark Cluster- using Cloudmesh's deploy command.

- **GitHub**: GitHub is 'a web-based Git or version control repository and Internet hosting service' [10]. We used Github repositories to store our files related to documentation, ansible scripts and python code.
- **Ansible**: Ansible is an open source automation platform that automates Application Deployment, and many other IT needs. In our project, Ansible is used to manage the configuration and automate deployment of software stacks on to the clouds. Ansible scripts are run via localhost by providing it with the IP addresses of the cloud VM's set up using Cloudmesh Client.

4. CLOUD INFRASTRUCTURE

Two clouds were selected for deployment: Chameleon Cloud and Jetstream. For successful deployment, following applications/utilities were used.

4.1. Chameleon Cloud

Chameleon is a collaborative cloud service funded by National Science Foundation primarily meant for research community. In our project three virtual machines were created on this cloud. One acting as master node and the other two as executor nodes for our spark application.

4.2. Jetstream Cloud

Jetstream is a cloud service led by Indiana University's Pervasive Technology Institute (PTI) in collaboration with other universities [13] across the United States. In our project three virtual machines were created on this cloud. One acting as master node and the other two as executor nodes for our spark application.

4.3. Cloud Hardware Comparison

Table 2. below shows a comparison of key computing resources on Chameleon and Jetstream Cloud environment.

Clouds	*Chameleon*	*Jetstream*
CPU	Intel Xeon X5550	Dual Intel E-2680v3 "Haswell"
Cores	1008	7680
RAM	5376 GB	40 Tbr
Speed	2.3 GHz	2.5 GHz
Storage	1.5 PB	2TB

Table 2. Cloud Hardware Specifications.

5. GETTING STARTED

The following section describes the process involved in building the spark application to predict the customer churn.

5.1. Loading the data

The 'Customer Data' used in the project contained 21 columns and 7000 rows. The data was stored in Hadoop distributed file system (HDFS) configured on the cloud cluster. Each row in the dataset represents an observed customer and each column represents attribute of that customer. Dataset was loaded into Spark DataFrame [14] along with the specified schema. After the dataframe was instantiated, it was queried using SQL queries with the help Pyspark DataFrame API.

5.2. Feature Engineering

Every single information we use to represent a customer is called a "feature" and the activity of finding useful features is called "feature engineering." In the Customer data set the data is labeled with two classes – Yes (Churned) and 0 (Not Churned). The features for each customer consist of:

Labels–> Churn: Yes or No

Features–> customerID,gender,SeniorCitizen,Partner, Dependents, PhoneService,MultipleLines,InternetService, OnlineSecurity, OnlineBackup,DeviceProtection,TechSupport, StreamingTV, StreamingMovies,Contract,PaperlessBilling, PaymentMethod, MonthlyCharges,TotalCharges

5.3. Define Feature Array

In order for the features to be used by a machine learning algorithm (Spark ML), It is useful for combining raw features and features into a single feature vector, which are vectors of numbers representing the value for each feature. Next, VectorAssembler[17] is used to transform and return a new dataframe with all of the feature columns in a vector column. Finally we use this dataframe to train our model.

5.4. Model Training

Fig 2 below shows the workflow used in training our machine learning algorithm.

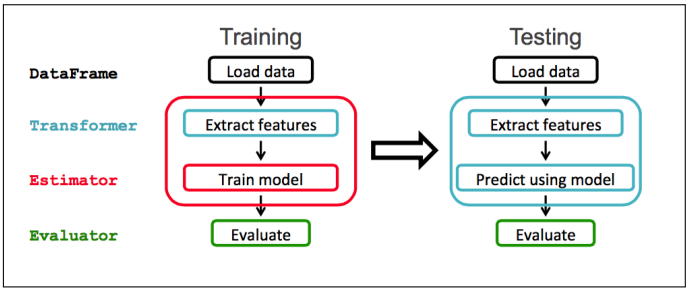


Fig. 2. Spark ML Workflow [18]

We used the following Machine Learning Algorithm and compared them using metrics like accuracy, AUROC and confusion matrix.

1. Random Forest
2. Decision Tree
3. Support Vector Machine

For each of the algorithms, the data was split into training and testing subsets with a ratio of 70 percent for training and 30 percent for testing.The predictive performance of each of the algorithm was evaluated by comparing predictions on the

testing data set with true values (known as ground truth) using a variety of metrics such as Accuracy, AUROC and Confusion Matrix.

6. MODEL EVALUATION

We used Spark ML's inbuilt classification evaluators to evaluate the predictions of our algorithms. Table 3 below shows the accuracy for each of the algorithm used.

Classification Algorithm	*Accuracy (%)*
Decision Tree	83.75
Support Vector Machine	86.82
Random Forest	84.02

Table 3. Predictive Accuracy

7. DEPLOYMENT

The project deployment scripts were designed to be simple and reproducible. The deployment consists of Ansible playbooks and Cloudmesh Client scripts. It is run on user's local machine(Ubuntu 16.04). Deployment scripts will install all the necessary software along with project codes onto the cluster nodes, run spark job on the cloud and finally fetch result on the users local machine. The following section describes how to use the deployment scripts to install and run the project in the cloud.

7.1. Requirements

In order to run the project deployment scripts, one must have Ansible and Cloudmesh Client configured and installed on their local machine.

1. Cloudmesh Client: In our project Cloudmesh Client is used to set up Hadoop virtual cluster with Spark on two cloud environment. In order to run a virtual machine in the cloud, /.cloudmesh/cloudmesh.yaml configuration file should be configured to fill in detail about our cloud accounts such as username, password, project name, etc. Following tasks are performed in our project using Cloudmesh Client:

   (a) Uploading Public Key- using Cloudmesh's key add and upload commands.

   ```
   $ cm key add --ssh
   $ cm key upload
   ```

   (b) Uploading Security Rules- using Cloudmesh's sec-group commands.

   ```
   $ cm secgroup upload
   ```

   (c) Creating Hadooop/Spark Cluster- using Cloudmesh's deploy command.

   ```
   $ cm cluster define --count 3
    --image CC-Ubuntu14.04
   $ cm hadoop define spark
   $ cm hadoop sync
   $ cm hadoop deploy
   ```

2. Ansible Scripts: In our project, the entire cloud interaction including spark application installation, uploading data, setting up environment, running the spark application, and fetching the result is performed via Ansible playbooks. Ansible script was used in the following way:

   (a) `Local Configuration` : In the host file the IP address of remote namenode of the cluster is specified as follows:

   ```
   [predict]
   129.114.33.144 ansible_ssh_user=cc
   ```

   (b) `predict_setup.yaml` : sets up environment, copies files/code and installs dependencies on the namenode of the cluster.

   (c) `predict_execute.yaml` : This moves the downloaded data to HDFS through the following command:

   ```
   $ hdfs dfs  -put <source> <hdfs-folderpath>
   ```

   This HDFS file will serve as an input to our spark application. Finally the spark application is submitted to the namenode, which runs it on distributed framework and fetches result from remote to local after job completion.

8. BENCHMARKING

After the Hadoop Spark cluster is set up on the cloud, a benchmarking process is run to asses the performance of our deployment and Spark job on the two clouds. Figure 3 below compares the time set up environment, install dependencies, run application and fetch result from both the clouds. For measuring the time we used shell script to calculate the time elapsed between begin and completion of Ansible playbooks.

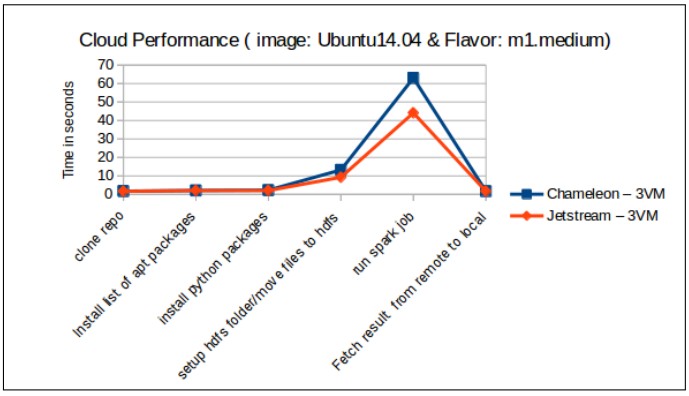


Fig. 3. Time Comparison in seconds

First time installation for the dependencies and apt-packages takes considerable amount of time(10-15 minutes) and hence are not included in the above comparison graphs. Specifically, Spark application performs 23 percent faster on 3 node Jetstream cluster when compared to similar cluster on chameleon.

9. CONCLUSION

We have built and tested a fully automated program to configure, deploy and run Apache Spark's Machine Learning algorithm for customer churn prediction on two cloud environments(Chameleon and Jetstream). The use of Ansible Galaxy to

automate deployment and Cloudmesh Client to setup Virtual Cluster on cloud helped in reproducibility and scalability.

10. ACKNOWLEDGMENT

The authors would like to thank Gregor von Laszewski and course TA's for their technical support and guidance.

11. WORK BREAKDOWN

The work in this project was equally distributed between the following authors.

Diksha Yadav: Data analysis and building Spark pipeline for machine learning algorithm in stand alone mode, benchmarking and troubleshooting.

Yatin Sharma: Code deployment, building ansible scripts, benchmarking and troubleshooting.

REFERENCES

[1] A. S. Foundation, "Overview - spark 2.1.0 documentation," Web Page, accessed: 03-12-2017. [Online]. Available: http://spark.apache.org/docs/latest/index.html

[2] "Ansible documentation," Web Page. [Online]. Available: http://docs.ansible.com/ansible/index.html

[3] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, *Learning Spark: Lightning-Fast Big Data Analysis*. O'Reilly Media, 2015.

[4] "Cloudmesh – my cloudmesh web documentation," Web Page. [Online]. Available: https://cloudmesh.github.io/

[5] "Cloudmesh – my cloudmesh web documentation," Web Page. [Online]. Available: https://cloudmesh.github.io/

[6] "Cloudmesh – my cloudmesh web documentation," Web Page. [Online]. Available: https://cloudmesh.github.io/

[7] "Welcome to python.org," Web Page. [Online]. Available: https://www.python.org/

[8] "Welcome to apache™ hadoop®!" Web Page. [Online]. Available: http://hadoop.apache.org/

[9] "Apache spark lightning-fast cluster computing," Web Page. [Online]. Available: http://spark.apache.org/

[10] "Github," Web Page. [Online]. Available: https://github.com/

[11] "Ansible is simple it automation," Web Page. [Online]. Available: https://www.ansible.com/

[12] "Home | chameleon cloud," Web Page. [Online]. Available: https://www.chameleoncloud.org/

[13] "Jetstream home," Web Page. [Online]. Available: https://jetstream-cloud.org/

[14] "Spark sql and dataframes," Web Page. [Online]. Available: http://spark.apache.org/docs/latest/sql-programming-guide.html

[15] "Ml pipelines," Web Page. [Online]. Available: https://spark.apache.org/docs/2.0.2/ml-pipeline.html#transformers

[16] "Ml pipelines," Web Page. [Online]. Available: https://spark.apache.org/docs/2.0.2/ml-pipeline.html

[17] "Extracting, transforming and selecting features - spark.ml," Web Page. [Online]. Available: http://docs.ansible.com/ansible/index.html

[18] "Predicting breast cancer using apache spark machine learning," Web Page. [Online]. Available: https://mapr.com/blog/predicting-breast-cancer-using-apache-spark-machine-learning-logistic-regression/