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-GG-0014/report/report.pdf not submitted

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

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

Prototyping a Virtual Robot Swarm with ROS and Gazebo

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, April 9, 2017

Our virtual robot swarm prototype accomplishes a *TBD task* by allocating portions of the task to each virtual robot (VR). As each VR works on its piece of the task, it communicates relevant information back to the master VR. Upon completion, the master VR collates the results and creates a human-readable report. The virtual swarm utilizes the *Robot Operating System* to control the virtual robots, *Gazebo* to simulate the task completion and *RVIZ* to visualize the process. We use *Ansible* to deploy the software to a distributed computing environment. The importance of our effort centers on some super-special conclusion I do not yet grasp.

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

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

1. INTRODUCTION

Stating the Problem: 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. It follows 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 requires much more time and effort than a single virtual robot. Collecting the results from a swarm also requires additional effort.

Our Contribution: We create a cross-platform system to quickly and relatively easily deploy and manage a swarm, as well as evaluate the swarm's operational effectiveness. Or maybe something else...I'm not sure, yet. Automate the deployment of a virtual robot swarm that will accomplish some arbitrary task; capture data from the swarm as it completes its task; report back the results in a human-readable format.

2. VIRTUAL ROBOT SWARM COMPONENTS

2.1. Robot Operating System (ROS) [1]

TBD; will include a discussion of a) how to obtain and install ROS and b) ROS graph concepts. The latter topic will introduce ROS' core components, namely a node, publications, suscriptions, topics and services. This section should probably cover ROS packages and ROS client libraries (primarily C++ and Python)

2.2. Gazebo [2]

TBD; again, introducing core Gazebo concepts. In this case, will include a) world files, b) model files and c) its client-server model. It should also include non-obvious limitation examples, e.g., a gripper arm driven by a single screw instead of multiple screws. In addition, it should allude to any limitations that affect our simulation.

2.3. Ansible

TBD; briefly describe Ansible - what it is, salient features, etc.

2.4. Testing Environment

TBD; briefly describe cloudmesh

3. VIRTUAL ROBOT SWARM PROJECT IMPLEMENTATION

3.1. VR Swarm task

TBD; discuss the task to be accomplished by the swarm, as well as how the information collected during task completion will be communicated back to the master node for collation and reporting.

3.2. Deployment

TBD; document the Ansible steps needed to successfully deploy ROS and Gazebo on multiple computers; will include references for obtaining major components, including adding new repositories if needed.

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;

3.7. Updating Software

TBD; discuss the methods used to implement software updates on each node; remain cognizant of battery levels *I have no idea how I might accomplish an over-the-air update of ROS. This point intimidates me.*

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

[2] 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/

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, April 24, 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

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

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.

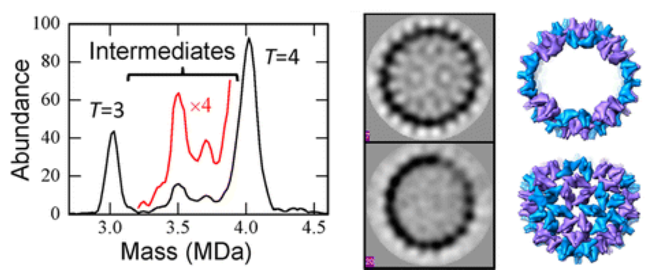


Fig. 1. The chart to the left displays an accurate measurement of the Hepatitis B virus (HBV) created by the research group [2]. This detailed mass information is used to create the images shown in the middle and to the right, which show 2-D and 3-D models of ions within the HBV. [2]

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. 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].

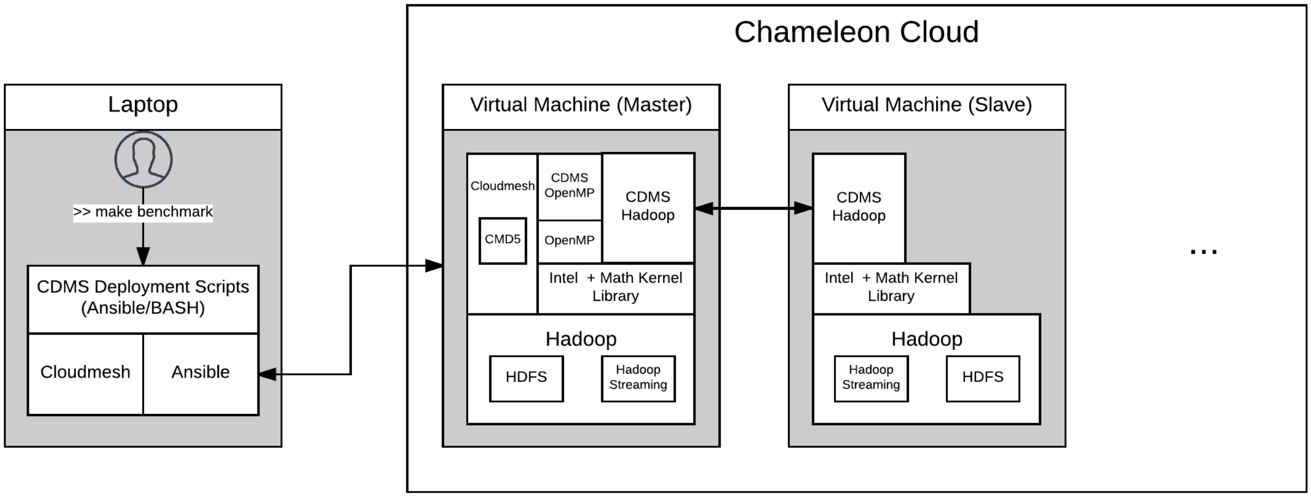


Fig. 3. The figure above provides a visual representation of the underlying architecture required for the Charge Detection Mass Spectrometry Cloud Computing workflow. The necessary software (i.e. Cloudmesh, Ansible, Hadoop, Intel, and etc.) and compute resources (i.e. Laptop and Virtual Machines) are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

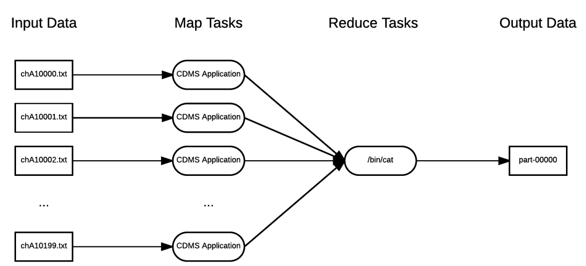


Fig. 4. The diagram shown above indicates the MapReduce style analysis of the CDMS Hadoop Streaming version of the application. Each two (2) MB input file is processed independently by the CDMS application and the results are aggregated with a single *cat* reduce task.

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. Peforamnce 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.

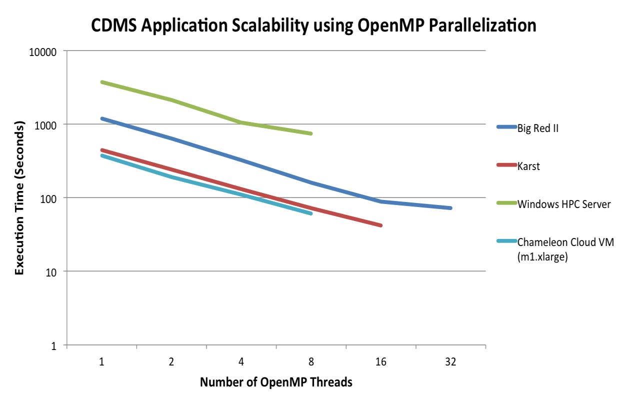


Fig. 5. The figure above shows the scalability (i.e. reduction in time-to-solution) as the number of OpenMP threads increase on local servers, Supercomputers and Clouds.

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.

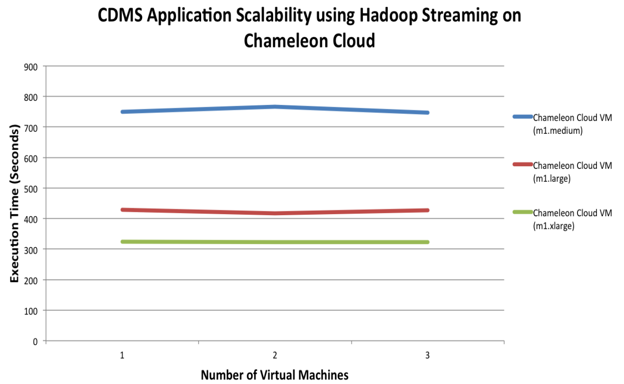


Fig. 6. The image above shows the scalability (i.e. reduction in time-to-solution) of the Charge Detection Mass Spectrometry Hadoop Streaming version of the application in a Chameleon Cloud virtual cluster. The performance information includes timing results for three (3) different virtual machine flavors.

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

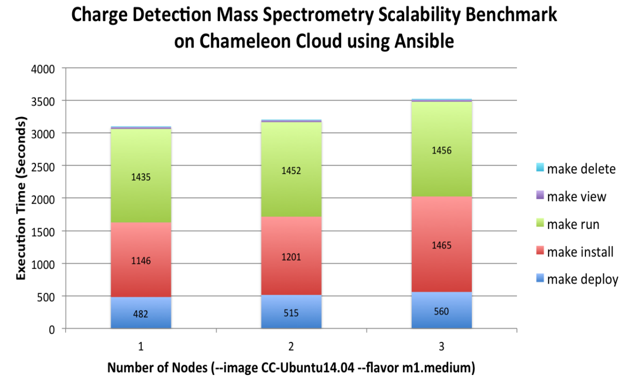


Fig. 7. The figure above indicates the time-to-solution for a full benchmark of the deployment of one (1), two (2) and three (3) virtual machines in the Chameleon Cloud. This benchmark includes depoying the virtual machines, installing all of the software, running the serial, OpenMP and Hadoop Streaming 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, April 24, 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

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.

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.

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].

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].

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 [**?**]. 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.

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].

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

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.

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.

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.

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].

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.

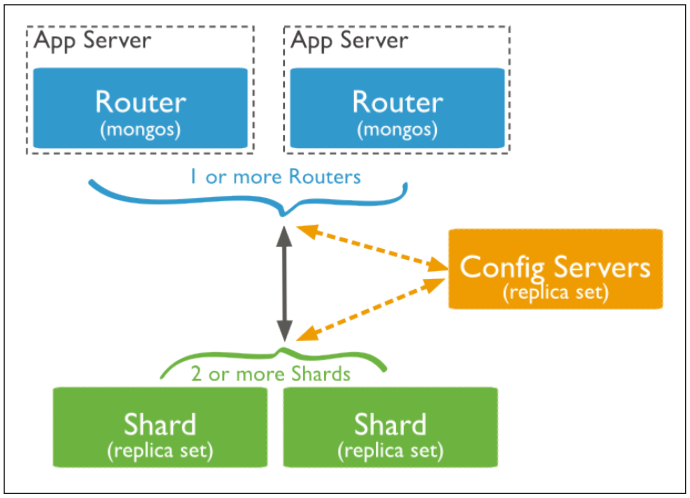


Fig. 1. Sharded MongoDB Architecture [15]

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.

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.

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.

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.

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).

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.

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.

Deployment Process

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

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.

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.

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.

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 3.

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.

Table 3. 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*

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 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 4. 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.

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.

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.

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.

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.

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.

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].

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.

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 5 and 6 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 5. 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 6. 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

Benchmarking Analysis

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.

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. While performance is inconsistent, mostly due to environmental factors, all clouds show and overall decline in run time as the number of shards increase, 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.

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,

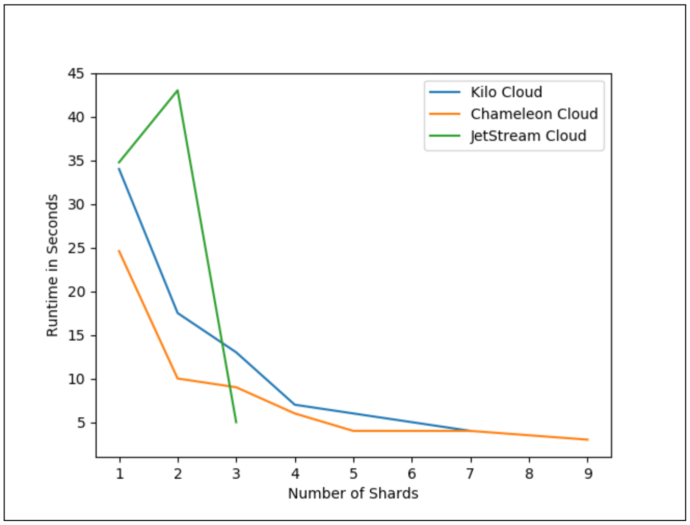


Fig. 2. Find Command - Sharding Test

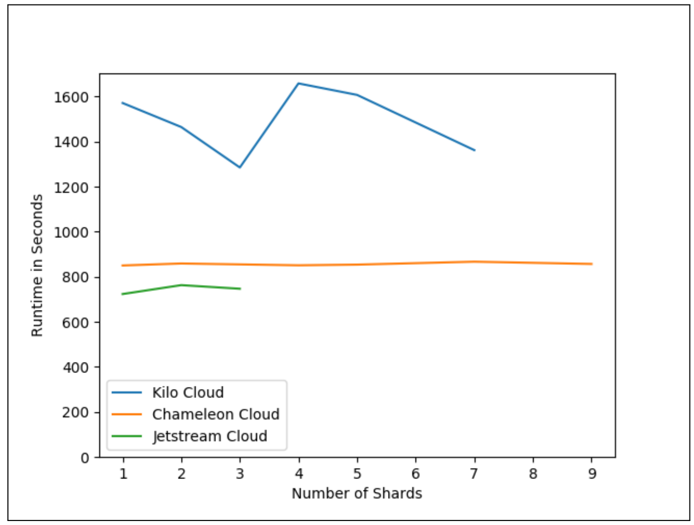


Fig. 3. Mongoimport Command - Sharding Test

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 was very erratic for this test, making it difficult to draw meaningful conclusions.

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.

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.

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. Interestingly, Jetstream performance is surprisingly consistent when compared to other tests and even faster than Chameleon Cloud.

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.

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. 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.

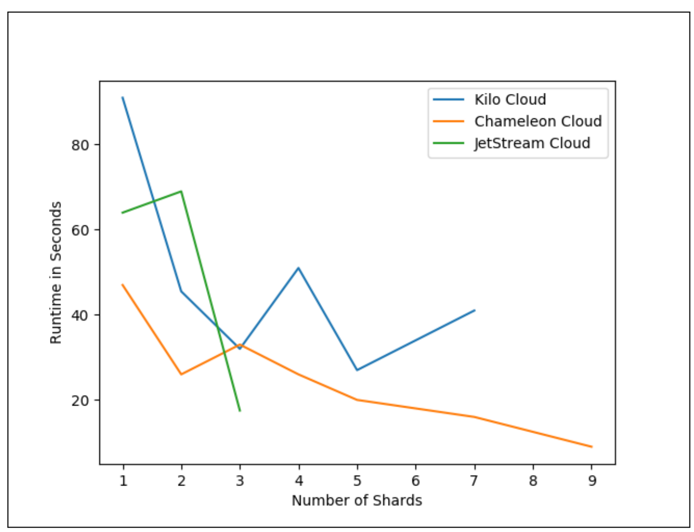


Fig. 4. MapReduce - Sharding Test

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.

Impact of Replication on Reads

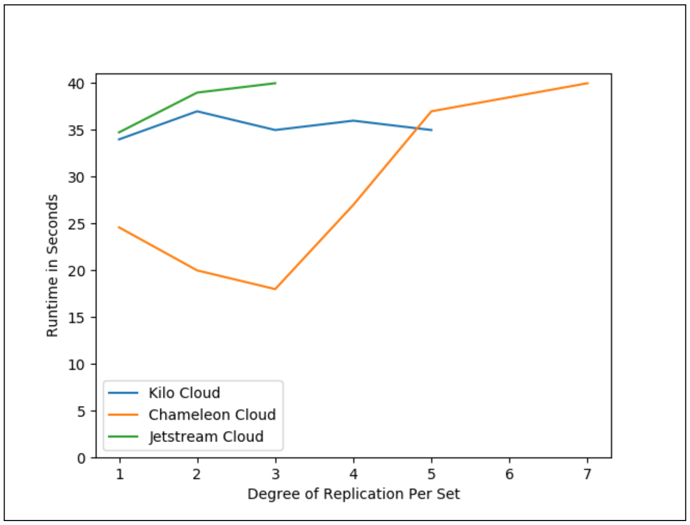


Fig. 5. Find Command - Replication Test

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 our sharding mongoimport test, 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.

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. 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.

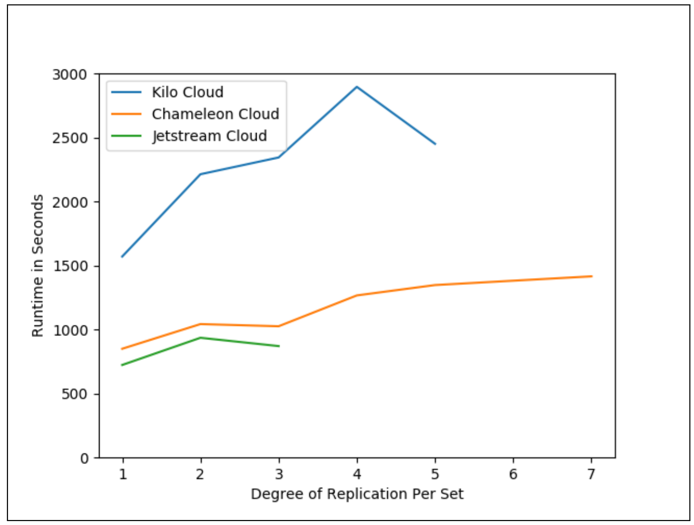


Fig. 6. Mongoimport Command - Replication Test

Performance on JetStream was best for the sharding levels that could be run, slightly better than Chameleon. 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.

Impact of Replication on MapReduce

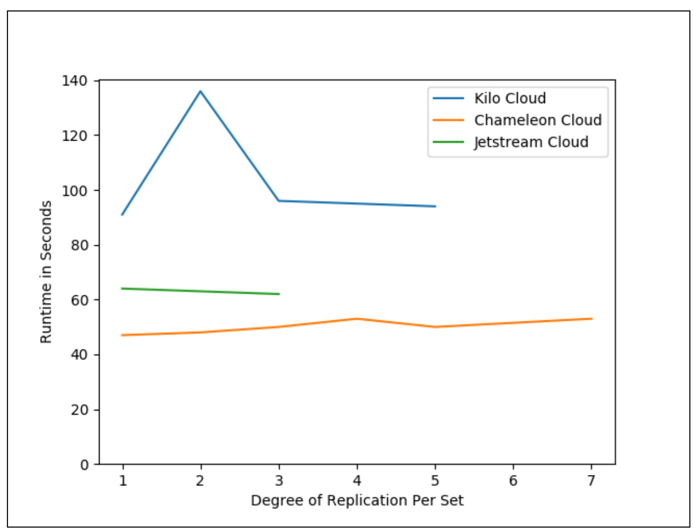


Fig. 7. MapReduce - Replication Test

As shown in Figure 7, other than a performance hit that is shown for all two shard tests on Jetstream (which appears to be an environmental issue), replication appears to have no impact on MapReduce operations. 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.

Impact of Version and Sharding on Reads

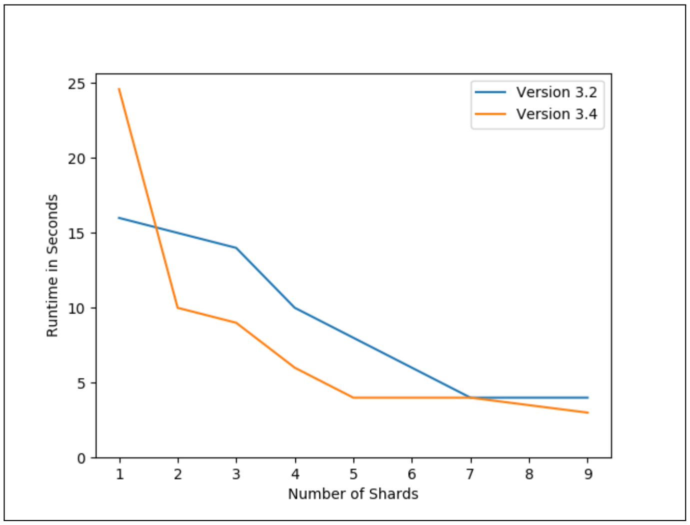


Fig. 8. Find Command - Version 3.2 vs 3.4

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.

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.

Impact of Version and Sharding on Writes

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.

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 degradation in version 3.4 MapReduce operations.

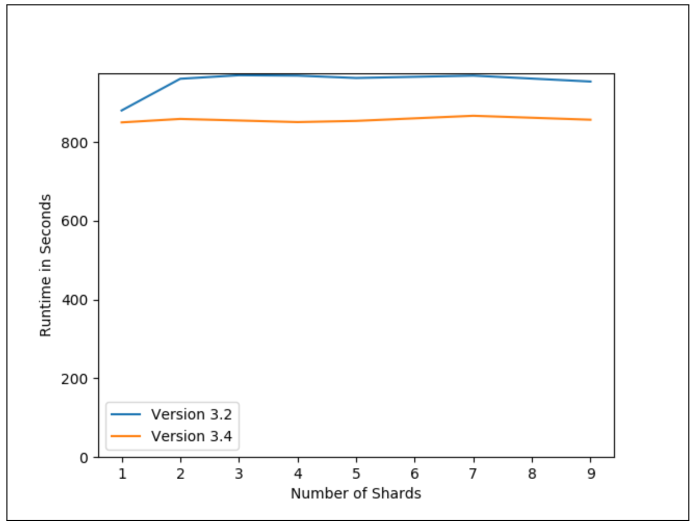


Fig. 9. Mongoimport Command - Version 3.2 vs 3.4

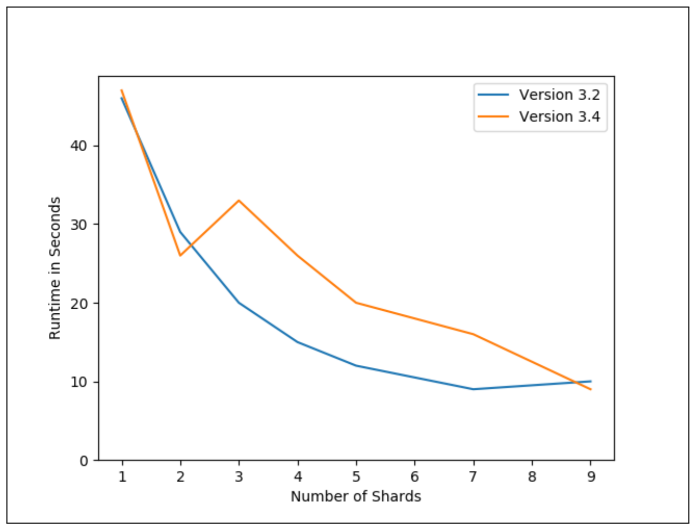


Fig. 10. MapReduce - Version 3.2 vs 3.4

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.

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/

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.

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].

EXECUTION INSTRUCTIONS

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.

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

kill

- Deletes and undefines the current cluster. No parameters.

benchmark

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

report

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

distribution

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

DIRECTORY STRUCTURE

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

benchmark/

Contains all benchmark timing logs

bin/

Contains all Bash and Python code

configfiles/

Contains all configuration file templates

deploy/
Contains all deployment timing logs

json/
Contains all json documents

playbooks/
Contains all Ansible YAML files

report/
Contains all reports in PNG format

stdlist/
Contains all bash script output logs

work/
Contains temporary work files

Proposal for 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.com

project-01, April 9, 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.

The following components will be utilized to accomplish the project:

- Ansible
- Apache Mesos
- Apache Spark
- MongoDB
- Million Song Dataset
- Billboard charts
- Python Scripts

2. COMPONENTS ROLES

ANSIBLE

Will be used to install software packages and define roles to different nodes in the cluster.

APACHE MESOS

Will act as the scheduler for the environment.

APACHE SPARK

Due to Sparks ability to parallel process, we'll utilize it to process the dataset to achieve the required performance while providing in-depth analysis.

MONGODB

MongoDB will be used as the repository for the dataset.

3. MILLION SONGS DATASET

This is a freely-available community maintained dataset [1]. The dataset will be used by ML as the source of random songs that will be analyzed for results. This project will utilize a subset of the dataset due to time and size of our development environment.

BILLBOARD CHARTS

In conjunction with Million Songs Dataset, Billboard charts [2] will be used to determine the greatest hits of all time, which will be used to train the model on how to determine a great hit.

PYTHON SCRIPTS

Scripts will be used to train the model and determine the next greatest hit.

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] labrosa, “Million song dataset,” WebPage, 2012. [Online]. Available: https://labrosa.ee.columbia.edu/millionsong/

[2] “http://www.billboard.com/charts,” WebPage, 2017. [Online]. Available: http://www.billboard.com/charts

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, April 9, 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, April 9, 2017

Geo-spatial data fits into definition of Big Data as it has all three 'V's viz. high-velocity, high-volume and high-variety. Big Data Analytics tools now allow us to analyze the huge volumes of geo-spatial data. Data of earthquakes that take place globally is a major part of crucial geo-spatial data. This application analyzes data related to earthquake which can be utilised in further research. 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].

Keywords: I524, geospatial, MongoDB, D3.js, Apache Spark, Python, USGS, Ansible

https://github.com/nsathe/sp17-i524/blob/master/project/S17-IO-3017/report/report.pdf

1. INTRODUCTION

The USGS estimates that several million earthquakes occur in the world each year, although many go undetected because they occur in remote areas or have very small magnitudes [2]. Thus earthquakes pose significant risk globally to the mankind. USGS collects volumes of geospatial data pertaining to earthquakes and makes it available for analysis. This project intends to analyze this data. The application will be deployed on cloud. Deployment will be automated using Ansible.

2. TECHNOLOGIES USED

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

1. Cloudmesh - For connecting to different cloud environments.
2. Ansible -For deploying software and associated packages.
3. Python - Writing script for data analysis and data processing
4. Apache Spark - For data processing
5. Mongo-DB - For storing Geo-spatial data
6. D3.js - As a visualization tool

3. EXECUTION PLAN

This is how I intend to execute the project on week-by-week basis. Although my intention is to follow the plan deligently, it is possible that because of technical and other un-foreseen challenges deadlines may be pushed ahead.

1. **6 Mar 2017 - 12 Mar 2017** Create virtual machines on Chameleon cloud using Cloudmesh and submit the project proposal.
2. **13 Mar 2017 - 19 Mar 2017** Deploy Mongo DB to Chameleon cloud using Cloudmesh and develop initial Ansible playbook to install the required software packages.
3. **20 Mar 2017 - 26 Mar 2017** Write script in Python for downloading USGS data at run-time. Write Python and Spark scripts for data analysis and processing.
4. **27 Mar 2017 - 02 Apr 2017** Implement visualization using D3.js. Update Ansible playbook to install D3js package.
5. **03 Apr 2017 - 09 Apr 2017** Test on different cloud systems. Define quantitative benchmarks. Tentatively benchmarks will be for data insertion time and data processing time.
6. **10 Apr 2017 - 16 Apr 2017** Create deployable software package in Python.
7. **17 Apr 2017 - 23 Apr 2017** Update and finalize the Project Report

4. BENCHMARK

As mentioned in Section 3, benchmarks are tentatively for data insertion time and data processing time on different clouds.

5. 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.

6. LICENSING

TBD

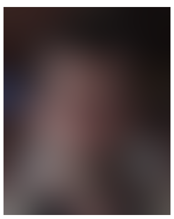
7. CONCLUSION

TBD

REFERENCES

[1] USGS, “Earthquake hazards program,” Web Page. [Online]. Available: https://earthquake.usgs.gov/
[2] ——, “About us - program overview,” Web Page. [Online]. Available: https://earthquake.usgs.gov/aboutus/

AUTHOR BIOGRAPHIES



Nandita Sathe is PMP certified project manager by profession. She will obtain MS in Data Sciences from Indiana University in May 2018. Her interests are in data analytics and machine learning.

8. WORK BREAKDOWN

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,April 9, 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, April 9, 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, April 26, 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

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?

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

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.

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

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.

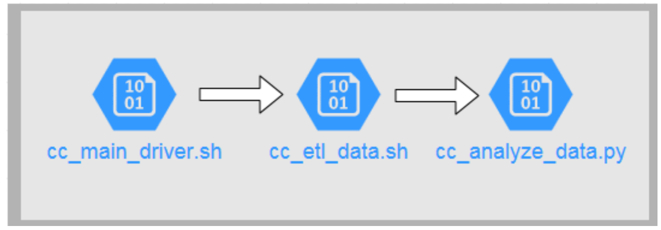


Fig. 2. Deployment Schema

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.

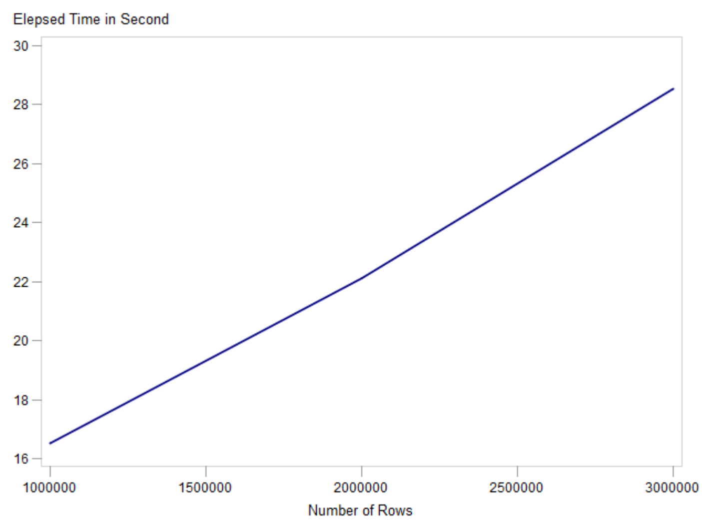


Fig. 3. Benchmark Testing - Number of Rows Vs. Elapsed Time

```
--------------------------------------------------------------
****************************** 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

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 / STATE	2011.0	2012.0	2013.0	2014.0	2015.0	2016.0	TOTAL
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 / STATE	2011	2012	2013	2014	2015	2016	TOTAL
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.

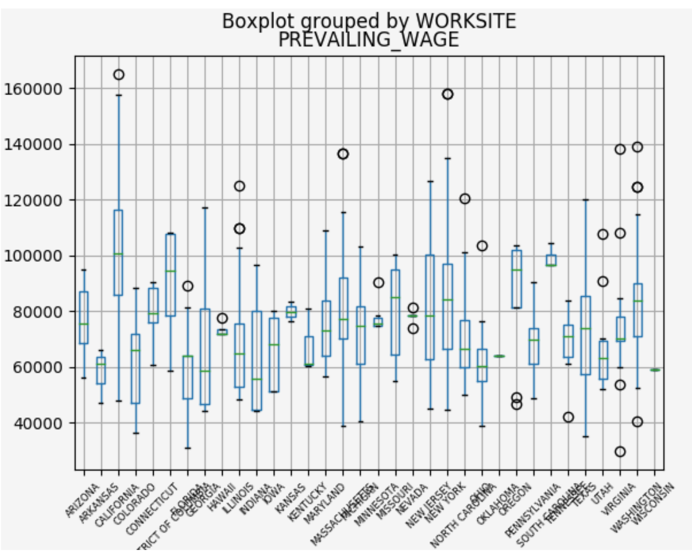


Fig. 11. Data Scientist Wage Across States

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.

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

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 based on relevance by ranking each ad based on the liklihood of clicking. Data is obtained as CSV files from Kaggle Datasets and is stored in Hadoop Data File system(HDFS). In this project, various Bigdata tools and softwares are used to carry out the analytical computations efficiently. And Ansible is used to deploy all the necessary softwares on a cloud.

Keywords: Hadoop, Ad click Prediction, BigData, Ansible, Chameleon cloud

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 analysis for advertisements or recommendations or any other purpose. This has given rise to the field of "Web Analytics" and one such application is Ad Click prediction. Many measures are available to assess the ad performance. One such measure to assess the immediate ad response is click-through rate (CTR) of the advertisement [2] which is defined as the ratio of a number of clicks on an ad to the number of times the ad is shown, expressed as a percentage [3]. The user activity data that is used for prediction is enormous. To handle such large volumes of data Big data technologies come handy. The dataset that we are dealing with in this project is released by Outbrain which is 2 Billion page views and 16,900,000 clicks of 700 Million unique users, across 560 sites [4]. The data is anonymized and is in CSV file format. Paraquet compressing along with impala/drill to be decided to query and analyze the data compressed in files that are stored in HDFS. Programming language for analyzing the data is yet to be decided between JAVA and Python. Ansible is used to deploy the software and chameleon cloud for running virtual machines.

2. BACKGROUND

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]. The dataset 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].

2.1. Data fields description

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].

3. SETUP AND CONFIGURATION

TBD

4. WORK FLOW

TBD

5. EXPERIMENTS AND RESULTS

TBD

6. LICENSING

TBD

7. CONCLUSION

TBD

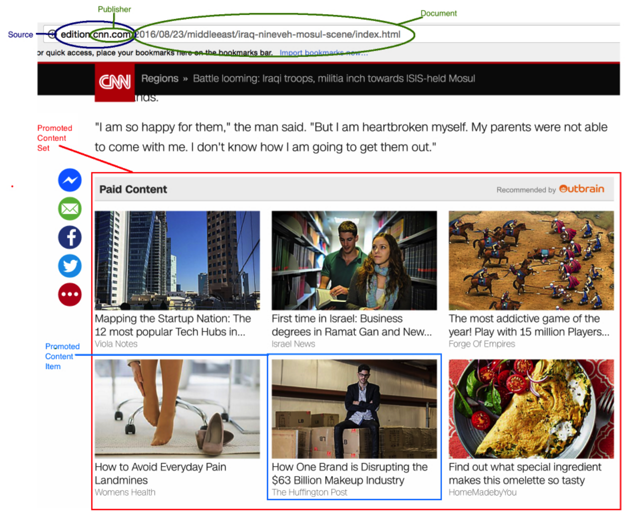


Fig. 1. Displaying Source, Publisher, Document, Promoted content set and items [4]

8. EXECUTION SUMMARY

1. Feb 23 - Mar 6, 2017 Exploring Python, Ansible and Chameleon cloud
2. Mar 10 - Mar 13, 2017 Exploring the datasets available and come up with the project proposal
3. Mar 14 - Mar 20, 2017 Decide on the architecture and workflow
4. Mar 21 - Mar 24, 2017 Configure and setup the workbench
5. Mar 25 - Mar 30, 2017 Prepare high level design
6. Mar 31 - Apr 6, 2017 Implementation and Testing
7. Apr 7 - Apr 11, 2017 Automate the deployment process using Ansible
8. Apr 12 - Apr 17, 2017 Optimizing and work on benchmarking
9. Apr 18 - Apr 22, 2017 Project review and completing any pending work
10. Apr 23 - Apr 24 Complete the report and commit the code

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

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, April 16, 2017

Analysis of flight data provides insights on the United States of America's Airline data. 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

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[1]. This statistics is publicly available in the Air Travel Consumer Report. 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[2].

2. MILESTONES

- Performing Analysis on local VM
  - Loading Data into HDFS
  - Pre-processing of the data using Pig
  - This data into Hive tables
- Analysis on the distributed cloud environment
- Visualizing the results using Tableau
- Benchmarking
- Final update with report

3. TECHNOLOGIES

- Distributed Computation and Storage:- HDFS, Hive and Pig
- Development:- Python and Java
- Deployment:- Ansible

4. DEPLOYMENT

Ansible Playbook is used as the application and configuration deployment tool. Deploying the Hive and Pig framework into the cluster environment.

5. BENCHMARKING

TBD

REFERENCES

[1] "Aviation analysis," Web Page. [Online]. Available: https://aviationanalytics.com/airport-analytics/data-analysis/
[2] "Big data in aviation," Web page. [Online]. Available: http://apex.aero/2016/11/30/big-data-aviation-industry-case-becoming-data-driven

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

Real-time Visualization of Happiness Quotient across English regions based on 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, April 14, 2017

This project involves development of a real-time system which streams live data from twitter to visualize the "Happiness index" across the English-speaking regions in the world. Live data from twitter is injected into the system using streaming API in spark. All possible tweets are taken into consideration for analyzing the overall happiness level of people tweeting from different locations. Suitable classifier will be built to identify if the tweet is positively biased. The results of the Language processing algorithm will be visualized in real-time using d3.js.

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

https://github.com/cloudmesh/sp17-i524/tree/master/project/S17-IR-P001/report/report.pdf

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 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. This large sample 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.

This project involves extraction of Twitter data using it's API through Python. This data is stored and manipulated in Cassandra which is then fed to Kafka. Kafka streamlines this data for analysis in Spark which is finally visualized using D3.js.

2. EXECUTION SUMMARY

The approximate schedule for completion of this project has been outlines in the section below:

1. Mar 6 - Mar 12, 2017 Create virtual machines on Chameleon, FutureSystems and Jetstream clouds using Cloudmesh and submit the project proposal.

2. Mar 13-Mar 19, 2017 Deploy Hadoop cluster to the clouds using Cloudmesh and create Ansible playbook to install the required software packages (Cassandra,D3.js,Kafka etc.) to the clusters and to upload the twitter data.

3. Mar 20-Mar 26, 2017 Pre-processing of the tweets to create required features for using in the Natural Language Processing algorithm. Building a language model to estimate the Happiness quotient

4. Mar 27-Apr 02, 2017 Develop an interactive visualization of the analysed data in D3.js

5. Apr 03-Apr 09, 2017 Continuing with the D3.Js visualization and connecting with streaming data from twitter to convert it into a live dashboard.

6. Apr 10 - Apr 16, 2017 Create software package that can be readily deployed in Python

7. Apr 17-Apr 23, 2017 Complete the partially done Project Report

3. 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.

3.1. Cassandra

3.2. Apache Kafka

Apcha Kafka is used as the queuing system for sending pulses of data to Pyspark. The tweets are pushed from Cassandra/Python to Kafka as and when the tweet is generated with a latency of less than a second. Kafka system is coordinated by three main components- Producer, Consumer and broker. Producer produces the data over topics, Brokers control the workload alloccated to the consumers and the consumers process the data.

4. WORKFLOW

The project will make use of the following four components.

1. Apache Spark

2. Apache Kafka

3. Apache Cassandra

4. D3.js

The Architecture of the system is shown in Fig.1

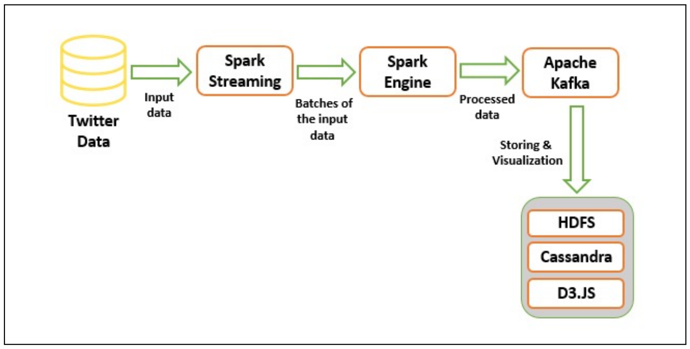


Fig. 1. Architecture

4.1. Python

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. Tweets get collected at the rate of generation as a json object. This data is then parsed to extract date, textual content and location which can then be stored in Cassandra.

4.2. Phase 1: Streaming Twitter data using Apache Spark

Spark is a high speed in-memory data engine which is specialized to perform tasks such as streaming or requiring repeated access to datasets. The objective is to obtain the happiness index of tweets from different English speaking countries around the world. Thus it will require a fairly large amount of tweets collected over a time frame(TBD). Spark's framework provides the facility to work with a variety of data formats including text. Spark streaming which is the extension of the core spark API aids in the streaming process and delivers it to the core engine. The data is then passed to Apache Kafka which helps in pipeline processing

4.3. Phase 2: Kafka

Kafka, a queueing system serves as an ingestion backbone to Apache spark. It is a super-fast, low-latency, distributed and partitioned stream processing service. Kafka being highly reliable and scalable, is perfect for integrating the huge stream of twitter data to a data sink.

4.4. Phase 3: Apache Cassandra

Cassandra was chosen as the database because of it's high scalability and reliability. Cassandra used along with spark streaming and kafka forms an excellent base for real time analytics. Cassandra being a NoSQL database is well suited to store unstructured textual data. A feature that makes Cassandra stand out is that it is a column oriented database which makes it horizontally scalable too.

4.5. Phase 4: D3.js

Real time visualization of the processed data streamed from Kafka message queuing service would be created to view the results from the analytics performed on the twitter data.

5. CONCLUSION

Put in some conclusion based on what you have researched

Acknowledgement Put in the information for this class and who may sponsor you. Examples will be given later

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

Flight Price Prediction

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-001, April 9, 2017

This project aims at tracking live flight status and flight pricing in the US. Live flight data streams are obtained using Python APIs and stored in Big Data Hadoop Distributed File Systems. This paper explores the use of Apache Hive to store data streams and analyse the data. The analyses will be presented in real time using d3.js.

Keywords: big data, apache hive

https://github.com/cloudmesh/sp17-i524/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. More often, customers tend to look for flights in the last minute, which is exploited by third party vendors who look to gain more profits in the rush hour. Skyscanner is a travel fare aggregator website and travel metasearch engine which helps users find the lowest rates from multiple travel sites, as well as instant comparisons for hotels and car hire removing the need for customers to search across different airlines for prices [1].

A metasearch engine (or aggregator) is a search tool that uses another search engine's data to produce their own results from the Internet [2]. Metasearch [3] engines take input from a user and simultaneously send out queries to third party search engines for results. Sufficient data is gathered, formatted by their ranks and presented to the users. The Skyscanner Live Pricing allows developers to access live pricing information on prices for different flights, by making requests to the Live Pricing API.

In this project, we would be querying the Skyscanner Live Pricing API using Apache HIVE and deploying the data on cloud (1-TBD & 2-TBD). Cloudmesh would be used for cloud management and the software stack deployment would be done through Ansible. We would benchmark performance of our analysis across mutilple clouds. We would be presenting a real-time visualization of the cheapest air fare and the most likely travel destination analysis in D3.js.

2. WORKFLOW

The project will make use of Python APIs to retrieve live flight prices information from Skyscanner and dump it in Apache HIVE database [4]. SQL Analyses are performed on this data and the results of analyses are stored in HIVE and presented in an interactive dashboard or website. The dashboard will take the onward and return journey locations, and the date of travel as inputs from users and show different price ranges for different dates commencing from the next available flight, for a period of three months. This aims to provide the users an idea as to when is the safe time to book flight tickets and beyond which date will the prices shoot up.

3. EXECUTION SUMMARY

The schedule for completion of this project has been outlined below:

1. Mar 06-Mar 12, 2017 Creating virtual machines on Chameleon cloud using Cloudmesh and coming up with a project proposal

2. Mar 13-Mar 19, 2017 using cloudmesh to set up Hadoop clusters and installing the required software packages

3. Mar 20-Mar 26, 2017 Fetching the data from Skyscanner API and adding it to our HIVE database

4. Mar 27-Apr 02, 2017 Running few data mining/time series models to predict the ticket prices

5. Apr 03-Apr 09, 2017 Review the work done and find out scopes for improvement and creating a benchmark report
6. Apr 10-Apr 16, 2017 Presenting the work in D3.Js in real-time as a visualization of the analysis
7. Apr 17-Apr 23, 2017 Complete the Project Report

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] B. J. Jansen, A. Spink, and C. Ciamacca, "An analysis of travel information searching on the we," Pennsylvania State University, Paper 10(2), 101-118, 2018, accessed: 2017-3-14. [Online]. Available: https://faculty.ist.psu.edu/jjansen/academic/jansen_travel_searching.pdf
[2] S. Berger, *Sandy Berger's Great Age Guide to Online Travel*, 1st ed. Greenwich, CT, USA: Que Publishing, 2005. [Online]. Available: https://www.amazon.com/Great-Guide-Internet-Sandy-Berger/dp/0789734427
[3] E. J. Glover, S. Lawrence1, W. P. Birmingham, and C. L. Giles, "Architecture of a metasearch engine that supports user information needsarchitecture of a metasearch engine that supports user information needs," in *Eigth International Conference on Information Knowledge Management*. ACM New York, NY, USA, 1999, pp. 210–216. [Online]. Available: http://www.researchgate.net/publication/2596239_Architecture_of_a_Metasearch_Engine_that_Supports_User_Information_Needs/file/d912f5131046ecac21.pdf
[4] L. Leverenz, "Gettingstarted with hive, apache software foundation," Web Page, Sep. 2016. [Online]. Available: https://cwiki.apache.org/confluence/display/Hive/GettingStarted

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, April 26, 2017

The aim of this project is to deploy a software package to detect street signs in a video stream. This will be a scalable system over Hadoop based cloud ecosystem to incorporate multiple video feeds and parallel real-time processing of the feeds. A comparative benchmark will be developed based on the performance of package on multiple cloud systems.

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

https://github.com/cloudmesh/sp17-i524/raw/master/project/S17-IR-P003/report/report.pdf

INTRODUCTION

Detecting objects in images has always been keen area of interest in the field of computer vision. There are many applications developed based on this simple idea like auto tagging pictures (e.g. Facebook, Phototime), counting the number of people in a street(e.g. Placemeter), classifying pictures, detecting vehicles, etc. On the similar grounds, we are building a software package which can be deployed easily on cloud infrastructure and establish a platform to detect different street signs in a video stream. A benchmark will be developed based on performance of this software on different cloud systems. The database of street signs will be restricted to US street signs. The video streams used for this project are simulated or captured using mobile camera.

The purpose of this project is to deploy a street sign detection algorithm on a cloud infrastructure to enable distributed processing of the images and videos. Detection and classification of street signs is an important feature in the era of autonomous driving vehicles. The market for autonomous driving and advanced driver assisted systems (ADAS) has increased the interest in the field of traffic sign detection techniques. Benchmarks have been created for the traffic sign detections on the German and Belgium Traffic Sign Datasets [1]. The only publicly available dataset for US traffic signs is the LISA dataset [2] which very huge. In this project, we have collected data by clicking photos of street signs, grabbing images from google and from the LISA dataset [2].

Image processing is a vast field and due to limited knowledge in this field we restrict ourselves to the basic image processing required to detect objects in street signs. Processing images and videos in real time requires a lot of computing power and 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 the deployment and processing of the algorithm for different workloads.

REQUIREMENT ANALYSIS

We are using following technologies for complete project development and deployment:

- Cloudmesh provides command line interface to connect to different cloud systems.
- Ansible - Ansible is an automated software deployment tool that only runs on the host machine.
- Python - Programming language.
- Spark - Distributed computing engine.
- YARN - is the resource manager for Spark.
- OpenCV [3] - Image/video analysis for street sign detection using open source computer vision libraries.The OpenCV library provides several features to manipulate images(apply filters, transformation), detect and recognize objects in images.

METHODOLOGY

1. Data gathering for street signs and video streams
2. Deploy Hadoop clusters on cloud using Cloudmesh
3. Develop Ansible script to install OpenCV on cloud

4. Build a model to detect or track street signs using OpenCV. We plan on implementing two programs-
    - Read an image and run the Haar cascade classifier to detect the signs in the image and
    - use the video stream and detect signs in real time.
5. To detect street signs, we will be using Haar based cascade classifier which detect objects in an image. As detecting signs and categorizing them are two different problems and use two different approaches. Hence, we will benchmark detection first and will work on categorization as future development.
6. Test the performance of software package on 3 different clouds or on the same cluster with multiple nodes.
7. Create benchmarks based on the above results

EXECUTION SUMMARY

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

1. Mar 6 - Mar 12, 2017: Create 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.

SYSTEM ARCHITECTURE

Figure 1 shows an overview of our system architecture, the host machine being our laptop in this case. Ansible and Cloudmesh client are installed on this host 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 street detection processing on Spark using Yarn resource manager. When we submit our job to Spark, the driver program initializes the SparkContext object which is responsible for the execution of the job. The 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.

CLOUD INFRASTRUCTURE

For the purpose of this project, we have been provided with two clouds – Chameleon and Jetstream. 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 [4]." Jetstream allows researchers to leverage the computational power of cloud while retaining the look and feel of home machines. Jetstream adds cloud based computational power to the national cyberinfrastructure, which was led by the Indiana University Pervasive Technology institute [5].

CLOUDMESH

Cloudmesh provides an easy interface to multiple clouds such as Chameleon and Jetstream through the command line.

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

defines a set of cluster machine instances. 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.

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
```

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. 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 [6].” OpenCV provides a Haar feature based cascade classifier that can be used for object detection, as proposed by in [7].

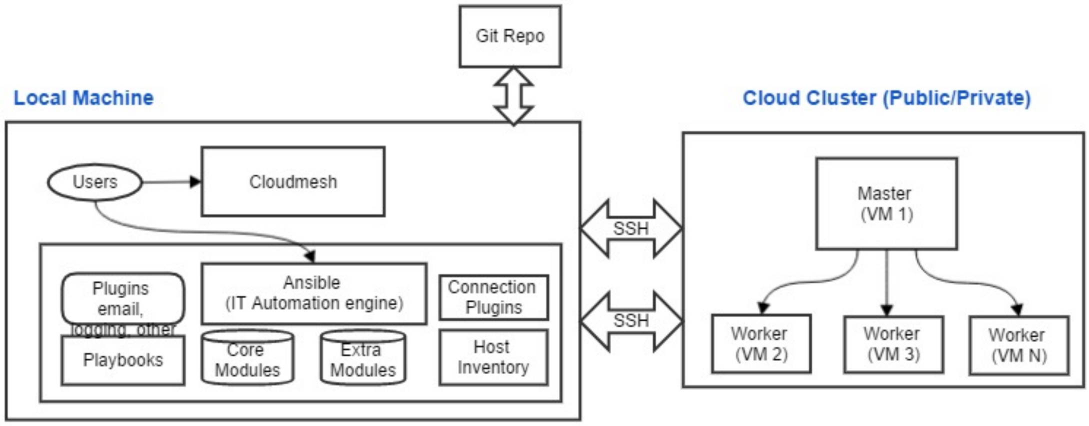


Fig. 1. System Architecture

Data Collection

The publicly available data set for U.S street signs is the LISA traffic dataset [2]. 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.

Train a Haar feature-based Cascade Classifier

Based on tutorials provided in [8], [2] and [3], 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 figure 2.

Fig. 2. 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.

Learnings

- 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.

PROGRAMMING ENVIRONMENT

Programming for the analysis module of the project was done in Python. OpenCV provides a library that can be used for running computer vision algorithms in Python. We leverage the Pyspark API provided by Spark to execute this module in Spark distributed environment. Ansible deployment scripts are written in YAML format which is similar to giving commands in plain English. Finally, the benchmarking for the project is done using shell script to calculate the processing times for each of the steps in the deployment method.

BENCHMARK

Benchmarks will be created based on the performance of the software in different cloud environments.Figure 3 shows the graph for time taken for 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. Figure 4 and figure 5 reflect the performance of Chameleon cloud for 4 and 50 input images respectively. Figure 6 and figure 7 reflect the performance of Jetstream for 4 and 50 input images respectively. There are a few outliers which can be attributed to the network delays. Overall these benchmarks clearly reflect that as the number of nodes increases the processing time decreases. Due to limited resources on Jetstream, the processing times for 3 node clusters could not be obtained.

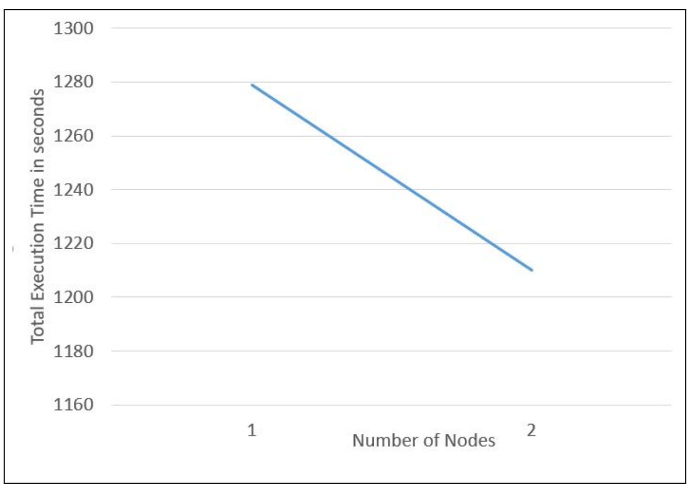


Fig. 3. Total time on Jetstream with 4 Input Images

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.

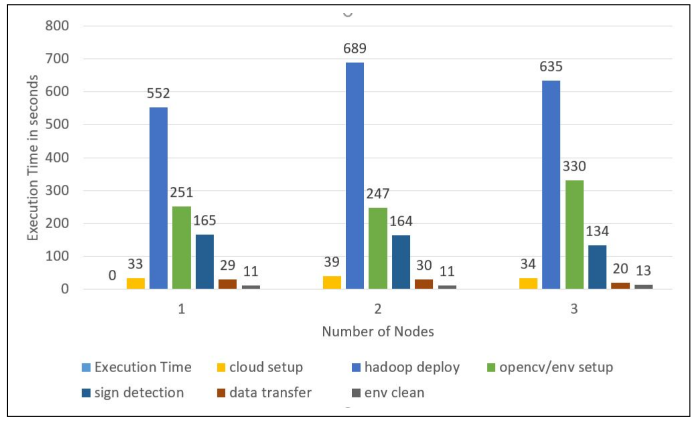


Fig. 4. Performance on Chameleon Cloud with 4 Input Images

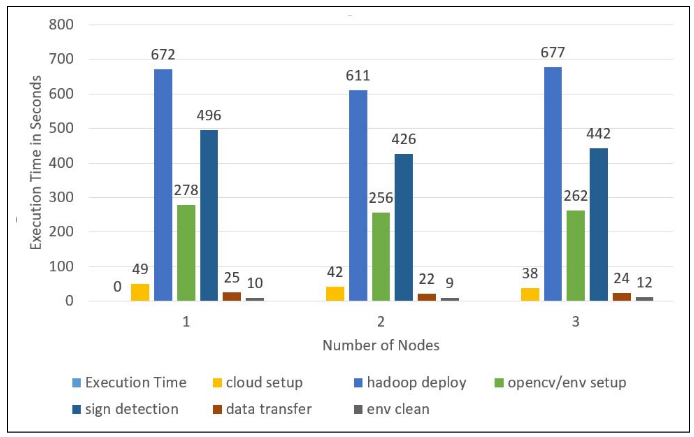


Fig. 5. Performance on Chameleon Cloud with 50 Input Images

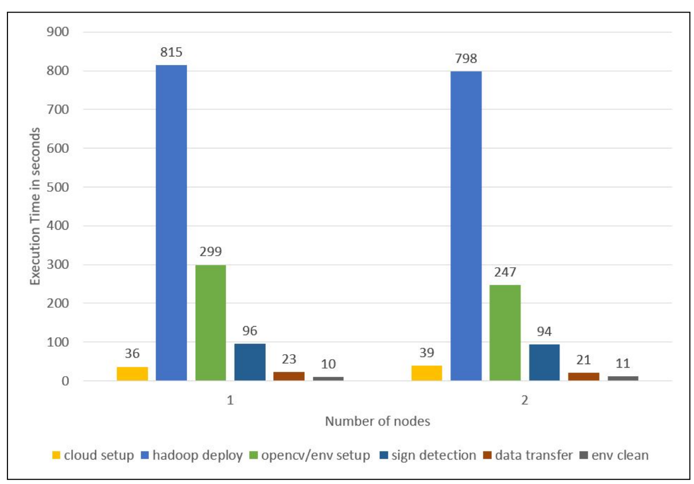


Fig. 6. Performance on Jetstream with 4 Input Images

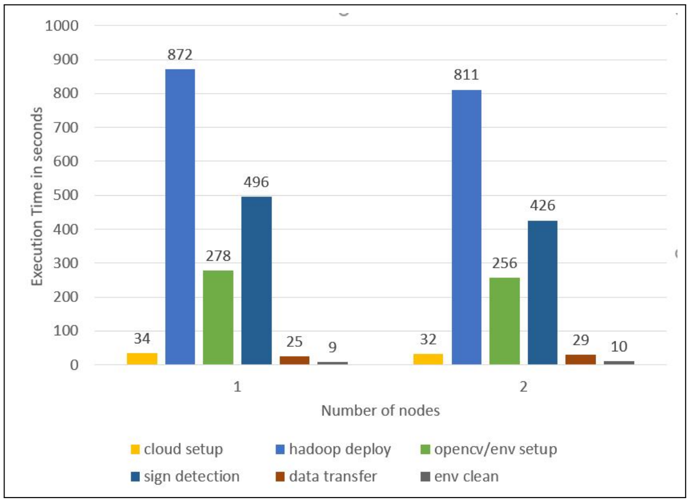


Fig. 7. Performance on Jetstream with 50 Input Images

FUTURE WORK

This work can be expanded to detect and classify all the U.S traffic signs. Benchmarks can be developed for them similar to the German and Belgium Traffic Sign Detection and Classification benchmarks.

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. Results presented in this paper were obtained using the Chameleon testbed supported by the National Science Foundation.

REFERENCES

[1] 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.

[2] 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.

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

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

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

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

[7] 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.

[8] 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

[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 will receive his Masters (Data Science) in 2018 from The Indiana Univeristy Bloomington. His research interests include Big Data and Machine Learning.

Snehal Chemburkar will receive her Masters (Data Science) in 2018 from Indiana University Bloomington. Her research interests include Big Data and Machine Learning.

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-000, April 24, 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

https://github.com/cloudmesh/classes/blob/master/project/S17-IR-P004/report/report.pdf

CONTENTS

1. INTRODUCTION

The idea behind this project is to introduce Hadoop/Spark over cloud infrastructure as a scalable and faster solution for predictive analysis using machine learning.

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. 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. We also compared the results of Spark's Mllib algorithms against scikit-learn's algorithms and found that both yielded similar results.

We approached the solution 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 [1], ICD-9 [2] and earlier studies [3]. We followed a workflow as shown in figure 1.

Our other important goal is to propose an end-to-end solution that is scalable and faster. While our dataset is 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 its faster processing [4] and advanced analytics through packages like Mllib [5] which provides several

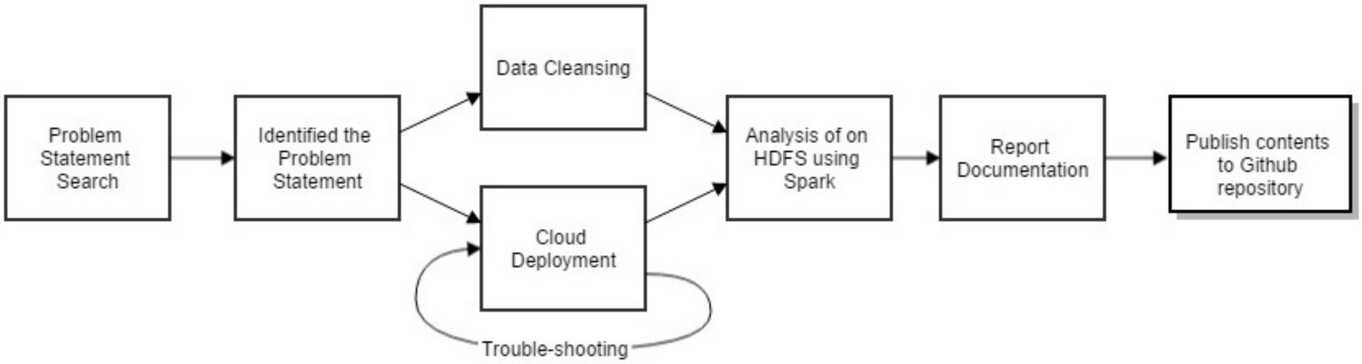


Fig. 1. Workflow of the project

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 compared the performance metrics of all the three infrastructure choices, with respect to our solution and included them in this report.

2. ARCHITECTURE

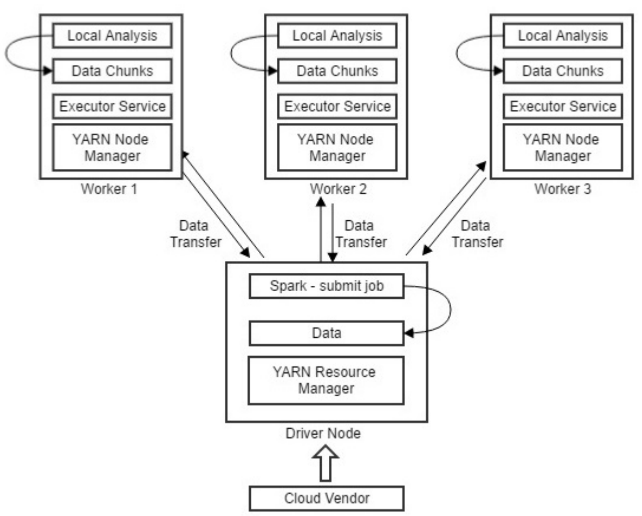


Fig. 2. Architecture Diagram

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 [6]. In practical terms, the driver is the program that creates the Spark Context [7], 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

Technology	*Usage*
Hadoop[8]/**Spark** [9]	Big Data Technologies
Python[10]	Development
MLlib[5]/**scikit-learn**[11]	Machine Learning Library
GitHub[12]	Project Repository
Ansible[13]	Application Deployment & Configuration Management
Chameleon[14], **JetStream**[15], **VirtualBox**[16]	Benchmarking
LaTex [17]	Document Preparation

Table 1. List of technologies used

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 [11].

- **MLlib**: Spark MLlib is the Spark's machine learning library provides machine learning algorithms that can be applied on Resilient Distributed Datasets [18]. It also provides other data manipulation utilities. MLlib has API available in Java, Python, Scala and R [5].

- **GitHub**: GitHub is 'a web-based Git or version control repository and Internet hosting service' [19]. 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 [15] 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.

Clouds	*Chameleon*	*Jetstream*	*VirtualBox*
CPU	Intel Xeon X5550	Dual Intel E-2680v3 "Haswell"	Intel Core i5-6200U
RAM	4 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

Table 2. Comparison of cloud vendors

5. AUTOMATED CLOUD DEPLOYMENT: ANSIBLE

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.

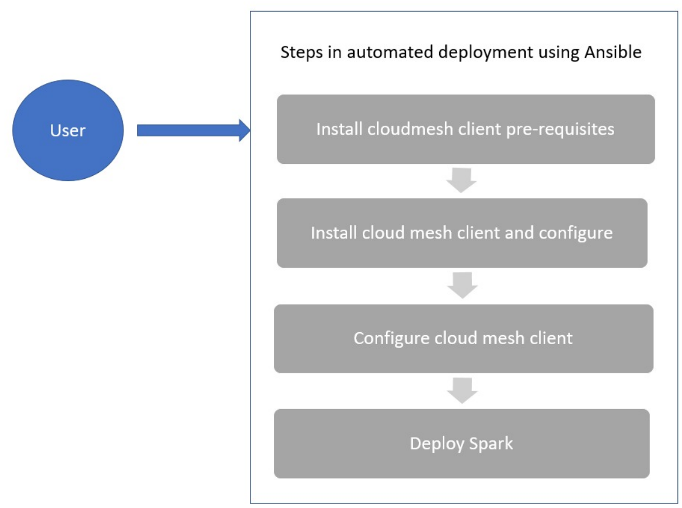


Fig. 3. Automated Cloud Deployment using Ansible

The Ansible playbook package constitutes 4 files.

1. *ansible.cfg*: This file consists of configuration information.
2. *playbook-cloudmesh-first-time-install.yml*: This file was used for deployment of cloudmesh client.
3. *host*: This file contains the list of hosts.
4. *hadoop-spark-playbook.yml*: This consists of the ansible code for redeployment of all the pre-requisites packages for

cloudmesh client. This also deployment of spark using cloudmesh client.

The following procedure has to be followed to run the Ansible script:

1. Install cloudmesh client for the first run 'playbookcloudmesh-first-time-install.yml' with the following command. This automates the deployment of cloudmesh client, which is a pre-requiste for installing Spark over our cloud infrastructure.

```
ansible-playbook
playbook-cloudmesh-first-time-install.yml
--ask-sudo-pass -vvvv
```

2. Open, /.cloudmesh/cloudmesh.yaml and edit the following section,the entry with <> should be customized as per your credentials.

```
profile:
     firstname: <first name>
     lastname: <last name>
     email: <email id>
     user: <chameleon/jetsream/other cloud username>
```

3. Change the entry of active cloud to use your preferred cloud. Example with chameleon is present below:

```
active:
  - chameleon
clouds:
  ...
```

4. Under a cloud(chameleon/jetstream/..) change the following entry, the entry with <> should be customized as per your credentials:

```
credentials:
OS_PASSWORD: <enter your chameleon cloud password
here>
OS_TENANT_NAME: CH-818664
OS_TENANT_ID: CH-818664
OS_PROJECT_NAME: CH-818664
OS_USERNAME: <username>
```

5. Also change, the type of OS and flavor you want for hadoop/spark cluster provided by your preferred cloud:

```
default:
        flavor: m1.medium
        image: CC-Ubuntu14.04
```

6. Edit the file hadoop-spark-playbook.yml in the section: 'name:Preparing cloudmesh- setting default user', the entry with <> should be customized as per user credentials:

```
- name: Preparing cloudmesh- setting default user
become_user: "{{ lookup('env','USER') }}"
shell: cm default user=<chameleon/jetstream user>
```

7. Now run the following command to deploy hadoop/spark cluster which will run the hadoop-spark-playbook.yml. It will again upgrade cloudmesh client, so it is up to date:

```
ansible-playbook hadoop-spark-playbook.yml
--ask-sudo-pass -vvvv
```

6. DATA CLEANSING AND PRE-PROCESSING

The initial data set is publicly available on the UCI Machine Learning Repository [1]. The initial data set has information extracted from the database satisfying the following criteria [3]:

1. Each row corresponds to an inpatient encounter (a hospital admission).
2. All of the encounters are "diabetic" encounters, that is, one during which any kind of diabetes was entered to the system as a diagnosis.
3. Each encounter also corresponds to a patient stay between 1 and 14 days.
4. Laboratory tests were performed during the encounter.
5. 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 [3]. 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:

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.

Classification Technique	*Number of Features*	*Accuracy (%)*
SGDClassifier	136	90.68
	26	86.31
	129	86.96
GaussianNB	136	10.43
	26	88.30
	129	9.96
KMeans	136	55.26
	26	55.24
	129	55.26
DecisionTreeClassifier	136	83.35
	26	82.50
	129	83.28

Table 3. Results from scikit-learn

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

9. RESULTS

Classification Technique	*Number of Features*	*Accuracy (%)*
SGDClassifier	136	90.88
	26	90.99
	129	90.88
GaussianNB	136	84.97
	26	85.02
	129	84.987
KMeans	136	55
	26	55
	129	55
DecisionTreeClassifier	136	91.01
	26	90.75
	129	90

Table 4. Results from Spark MLlib

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.

These can be found in the below table. 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

In the figure 4 we are displaying the time taken to run each of the four algorithms in different clouds. The four algorithms are SGD Classifier, Gaussian Naïve Bayes, kmeans and Decision Tree Classifier.

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 compute resources used. For example the chameleon cloud takes less time to execute any other algorithm because the RAM configuration for the VMs of chameleon cloud is better than any other VMs.
3. 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 across geographies. This may result in network latency and affect the runtime of a program running in a distributed environment.

10.1. Computation time of algorithms in clouds

Figure 4 shows the performance of multiple machine learning algorithms using Spark MLlib in multiple clouds.

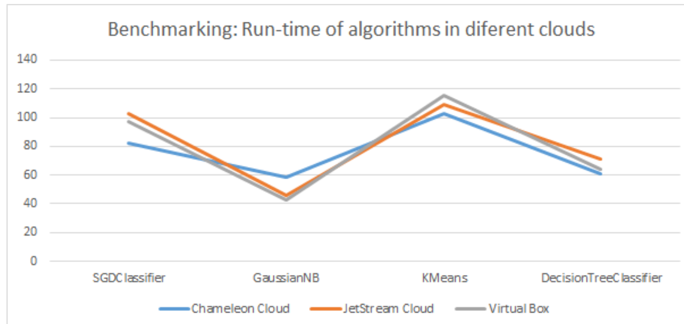


Fig. 4. Run-time in different clouds.

10.2. Comparison of multiple algorithms in Spark MLlib vs scikit-learn

We can see from figure 5 that accuracies between multiple algorithms in MLlib are almost smiliar.

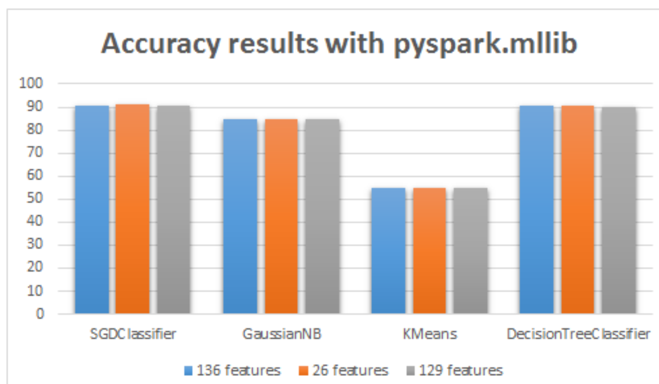


Fig. 5. Accuracy Results with pyspark.mllib

Figure 6 shows accuracy results of multiple machine learning algorithms in scikit-learn vs MLlib.

11. TROUBLESHOOTING

We encountered several errors while running different commands in the Linux terminal. We have listed some of the commands and errors encountered while executing them followed by the steps to resolve them.

1. **Command**: cm ssh key upload

   Error 1: Permission denied (public key)

   Error 2: Problem uploading key <username> to cloud chameleon: The request you have made requires authentication

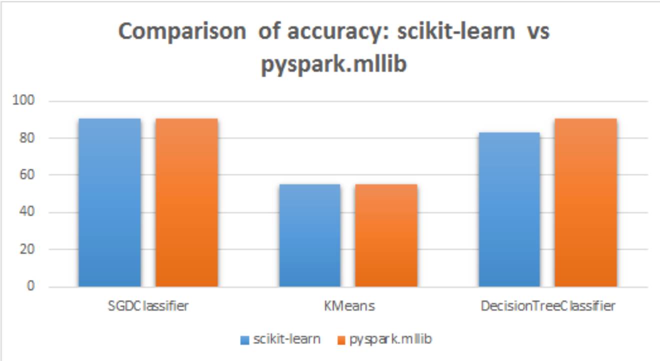


Fig. 6. Accuracy Results: scikit-learn vs pyspark.mllib

Resolution:
cm ssh-key delete <username>-001
cm ssh-key delete <username>-001 cloud = chameleon
cm ssh-key add
cm ssh-key upload
cm key refresh

2. **Command**: cm hadoop deploy

   Error: IndexError: list index out of range

   Resolution: Use the following image:

   cm cluster define –count 3 –image CC-Ubuntu14.04

3. **Command**: cm hadoop deploy

   Error: INFO: Waiting for cluster to be accessible DEBUG: Running cmd: ansible all -m ping -u ubuntu stack-0011 | UNREACHABLE! => { "changed": false, "msg": "SSH encountered an unknown error during the connection. We recommend you re-run the command using -vvvv, which will enable SSH debugging output to help diagnose the issue", "unreachable": true }

   Resolution: Use the following image:

   cm cluster define –count 3 –image CC-Ubuntu14.04

4. **Command**: cm hadoop sync

   Error: no github account associated

   Resolution:

   git config –global user.name "username" git config –global user.email "emailid"

5. **Command**: apt-get install git

   Error: Permission denied when running "apt-get install git" command

   Resolution: Exit from the current hadoop user using *exit* command. The user will change to 'cc' that has the root privileges. Install the package through the 'cc' user log-in back as the hadoop user to use the installed package .

The "Cm refresh on" command can be used most of the times to resolve multiple errors.

12. 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 with an accuracy 90% by analyzing the data stored on HDFS. 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 used MLlib library's algorithms to achieve similar results as obtained with scikit-learn library. These technologies can be used to build similar solutions for real world scenarios requiring processing and performing analytics over big data. Similar applications can be built leveraging Spark's ability to process streams of data and support for machine learning algorithms.

13. 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] 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#
[2] U. D. of Health and H. S. (HHS), "ICD9Data.com," Web Page, accessed: 2017-04-21. [Online]. Available: http://www.icd9data.com/
[3] C. G. J. L. O. S. V. K. J. C. Beata Strack, Jonathan P. DeShazo and J. N. Clore, "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/
[4] Databricks, "Apache Spark," Web Page, accessed: 2017-04-21. [Online]. Available: https://databricks.com/spark/about
[5] Apache, "Apache Spark MLlib," Web Page, accessed: 2017-04-21. [Online]. Available: http://spark.apache.org/mllib/
[6] ——, "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
[7] ——, "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
[8] ——, "Welcome to Apache™ Hadoop®!" Web Page, accessed: 2017-03-12. [Online]. Available: http://hadoop.apache.org/
[9] ——, "Apache Spark: Lightning-fast cluster computing," Web Page, accessed: 2017-03-12. [Online]. Available: http://spark.apache.org/
[10] P. S. Foundation, "python," Web Page, accessed: 2017-03-12. [Online]. Available: https://www.python.org/
[11] D. Cournapeau, "scikit-learn," Web Page, accessed: 2017-04-21. [Online]. Available: http://scikit-learn.org/stable/
[12] T. Preston-Werner, "GitHub," Web Page, accessed: 2017-04-21. [Online]. Available: https://github.com/
[13] R. Hat, "ANSIBLE," Web Page, accessed: 2017-03-12. [Online]. Available: https://www.ansible.com/
[14] C. Cloud, "Chameleon," Web Page, accessed: 2017-04-21. [Online]. Available: https://www.chameleoncloud.org/
[15] I. University, "Jetstream," Web Page, accessed: 2017-04-21. [Online]. Available: https://jetstream-cloud.org/
[16] Oracle, "VirtualBox," Web Page, accessed: 2017-04-21. [Online]. Available: https://www.virtualbox.org/wiki/VirtualBox
[17] LATEX, "The LATEX Project," Web Page, accessed: 2017-03-12. [Online]. Available: https://www.latex-project.org/
[18] Apache, "Machine Learning Library (MLlib) Guide," Web Page, accessed: 2017-04-21. [Online]. Available: http://spark.apache.org/docs/latest/ml-guide.html
[19] W. Inc., "GitHub," Web Page, accessed: 2017-04-21. [Online]. Available: https://en.wikipedia.org/wiki/GitHub

[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

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 , April 22, 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

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

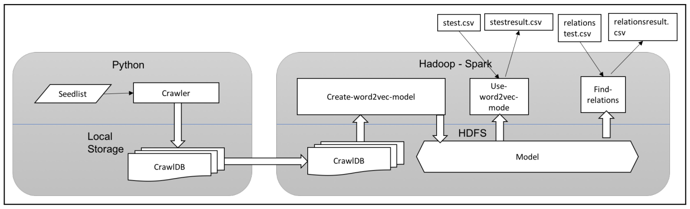


Fig. 1. Data Pipeline.

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

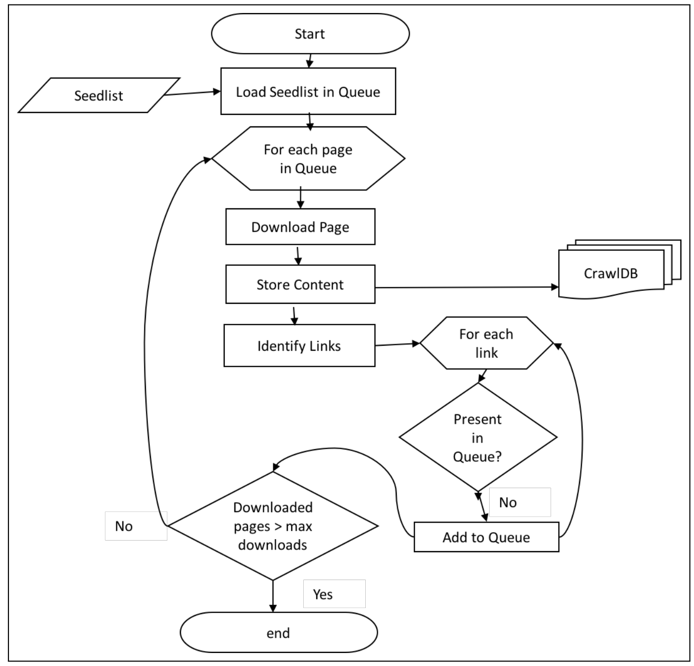


Fig. 2. Flowchart of crawler.

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

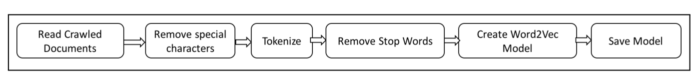


Fig. 3. Steps involved in Word2Vec model creation.

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.

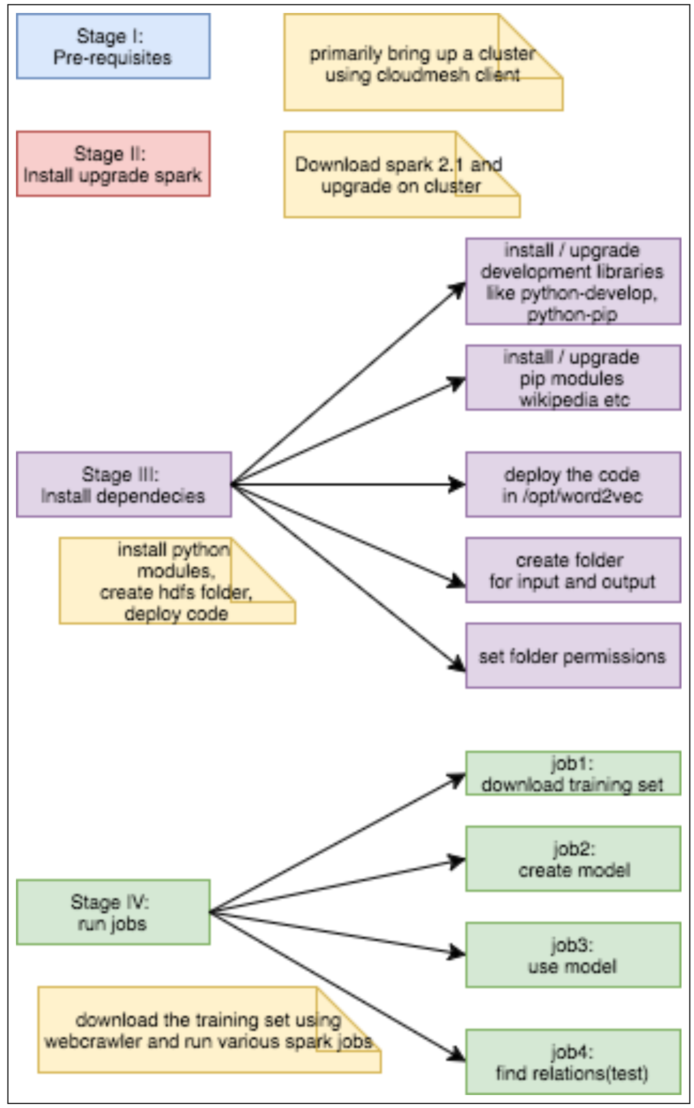


Fig. 4. Deployment stages.

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 on the Chameleon and Jetstream cloud environments.

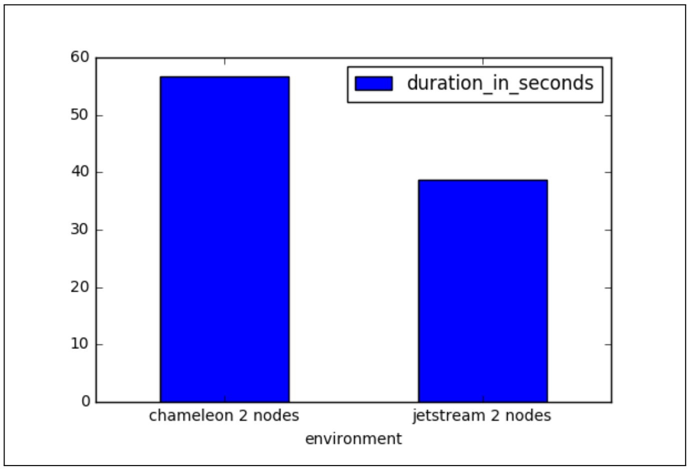


Fig. 5. Time taken by CreateWord2Vec for Dataset1

Figure 6 shows the total time taken by CreateWord2Vec application for Dataset2 on the Chameleon and Jetstream cloud environments.

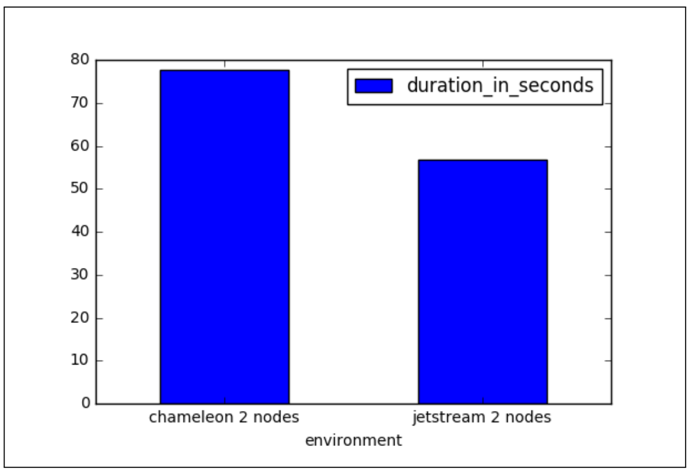


Fig. 6. Time taken by CreateWord2Vec for Dataset2

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 config.propertise to handle such situation.

```
spark_executor_memory = <memory given to executor>
spark_driver_memory = <memory given to driver>
max_result_size = <maximum result size>
```

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, April 9, 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 and Pandas 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-1, April 9, 2017

A classification model shall be built by working on a training data set of 5574 text messages, each marked as a spam or a legitimate message. The model shall be used to correctly predict the class of any new incoming text message as a spam or a legitimate one. The application shall be deployed using Ansible scripts over Kubernetes cluster while data manipulation and classification shall be done using Pandas and R in conjunction.

Keywords: Docker, Ansible, Kubernetes, R, Pandas

https://github.com/cloudmesh/sp17-i524/raw/master/project/S17-IR-P007/report/report.pdf

test

1. INTRODUCTION

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 [1]. 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.

2. SOFTWARE STACK

Name of the Technology	Purpose in the Project
R and Pandas	data analytics
Docker	container for the application
Kubernetes	cluster creation and management

Fig. 1. Technologies used in the Project

2.1. 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.

2.2. R

R is a language and environment for statistical computing and graphics [2]. 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.

2.3. 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 [3]. A dockerized application can simply ship between platforms as the complexity of software depencies is handled by the container.

2.4. Kubernetes

Kubernetes[4] 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.

3. DESIGN

3.1. Building the Classification model

3.1.1. CrossValidation for the training data

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.

3.1.2. 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 [5].

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.

3.2. 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 [6].

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.

4. DISCUSSION

TBD

5. DEPLOYMENT

Our application will be deployed using Ansible [7] 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.

6. CONCLUSION

TBD

7. ACKNOWLEDGEMENT

We acknowledge our professor Gregor von Laszewski and all associate instructors for helping us and guiding us throughout this project.

8. APPENDICES

TBD

REFERENCES

[1] "SMS spam collection dataset," Web Page, accessed: 2017-03-10. [Online]. Available: https://www.kaggle.com/uciml/sms-spam-collection-dataset

[2] "R:what is R?" Web Page, accessed: 2017-04-02. [Online]. Available: https://www.r-project.org/about.html

[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] 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

[6] Wikipedia, "Naive bayes spam filtering," Web Page, January 2017, accessed: 2017-03-10. [Online]. Available: https://en.wikipedia.org/wiki/Naive_Bayes_spam_filtering

[7] "Ansible, deploy apps. manage systems. crush complexity," Web Page, accessed: 2017-03-12. [Online]. Available: https://www.ansible.com/

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, April 24, 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].

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.

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.

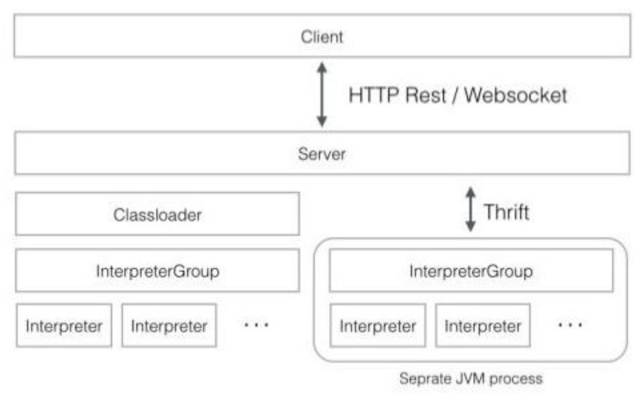


Fig. 1. Zeppelin Architecture

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

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

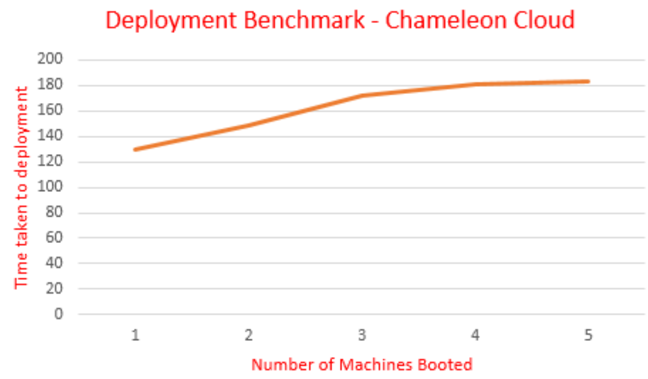


Fig. 2. Jetstream Deployment Benchmarking

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.

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 1. 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

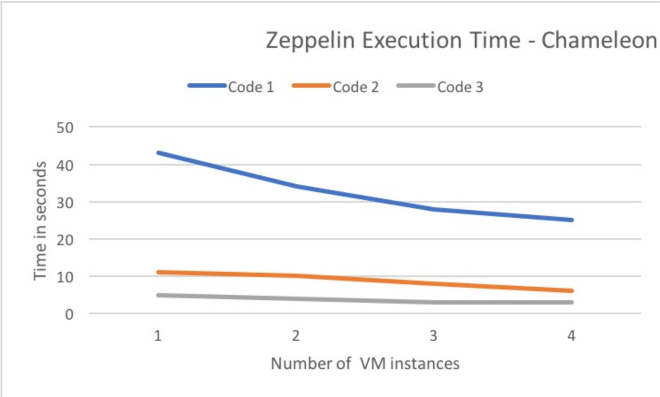


Fig. 3. Chameleon Analytics Benchmarking

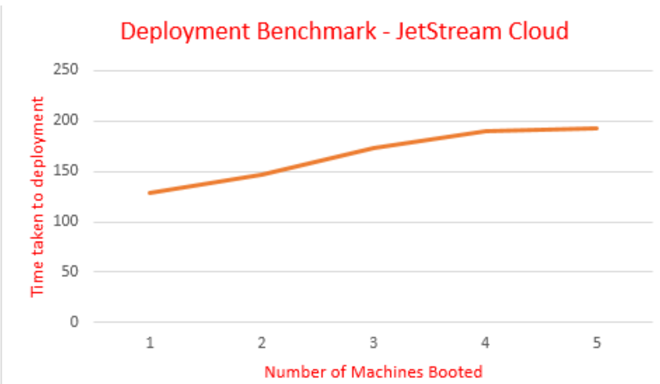


Fig. 4. Jetstream Deployment Benchmarking

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 are presented in the table below.

Analytics deployment shows a slight decrease in time as number of nodes increased.

Table 2. 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

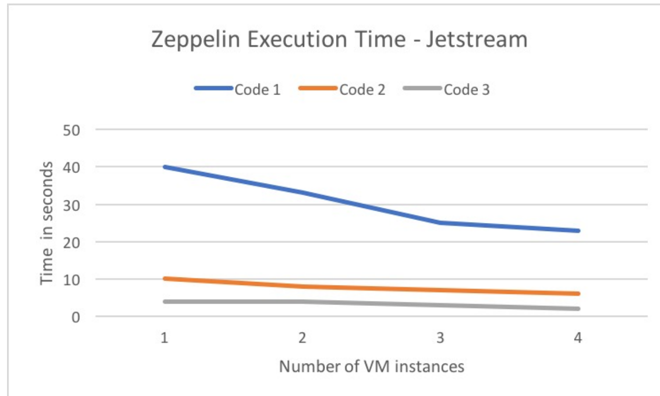


Fig. 5. Jetstream Analytics Benchmarking

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
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

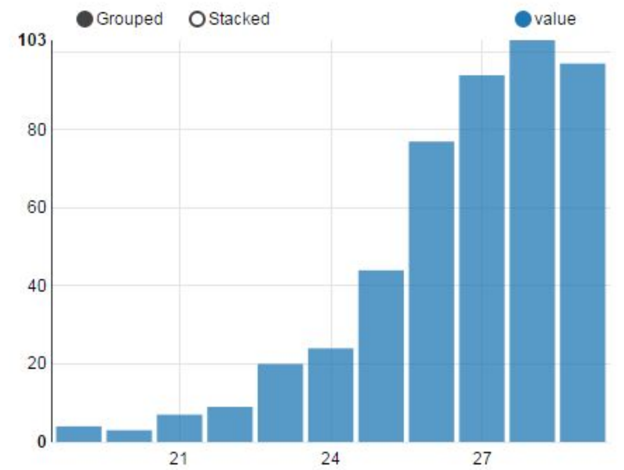


Fig. 6. Histogram/ Bar Chart

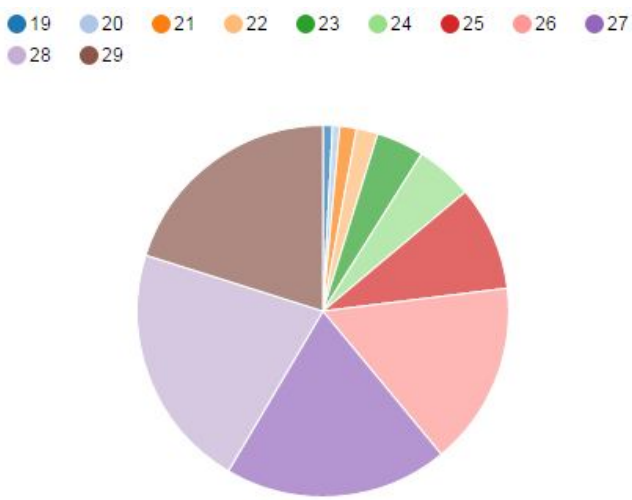


Fig. 7. Pie Chart

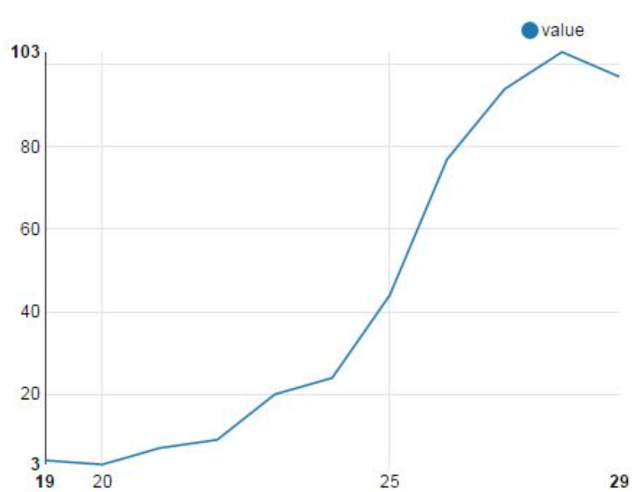


Fig. 8. Line Chart

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.

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.

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

A. APPENDIX

A.1. 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.

A.1.1. 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.

A.1.2. 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.

A.1.3. 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

A.1.4. 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.

A.1.5. 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.

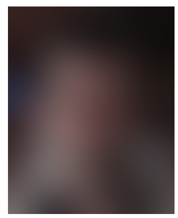
2. Fixed security issues and checked if the notebook is accessible through the external web-browsers.

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.

A.1.6. April 17,2017 - April 23,201

Review of deployment and developing the final draft of the report for submission.

AUTHOR BIOGRAPHIES



Naveenkumar Ramaraju yet to update his Bio

Veera Marni yet to update his bio

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, April 24, 2017

Cloudmesh client is a simple client to enable access to multiple cloud environments form a command shell and command line. The user's 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 cloudmesh client through a simple and extensible command line interface.

Keywords: Cloud, I524

https://github.com/cloudmesh/sp17-i524/blob/master/project/S17-IR-P009/report/report.pdf

1. INTRODUCTION

In the time of Big data and Micro-Services,it is common to have a set of services running on multiple clouds.Docker[1] with the build and ship model made the micro-services architecture work better.Since there would multiple containers running across multiple remote clouds managing them can become tedious.In project we have added the capability to provision and manage docker[1] and swarm[2] containers to cloudmesh client[3].

Clouldmesh Client[3] capability is detailed in Figure 1,it aims at managing vm instances in multiple heterogeneous clouds remotely via a command line interface.

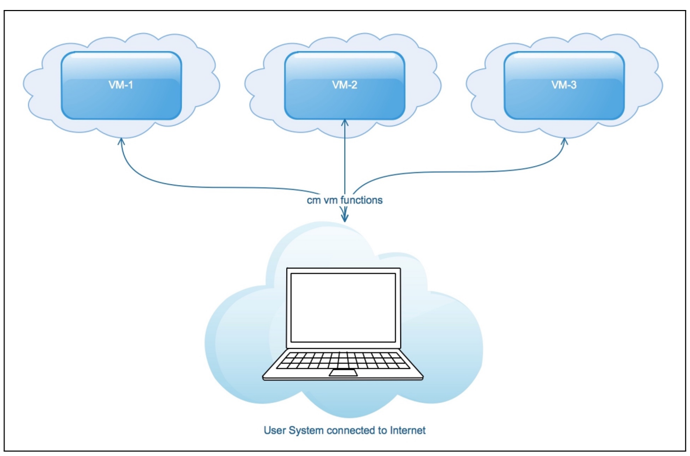


Fig. 1. Cloudmesh client

The cloudmesh docker application build can be broadly divided into Docker and Swarm modules.

The Docker Module capabilities are detailed in Figure 2. This module would help user perform various docker functions mentioned in Section 3

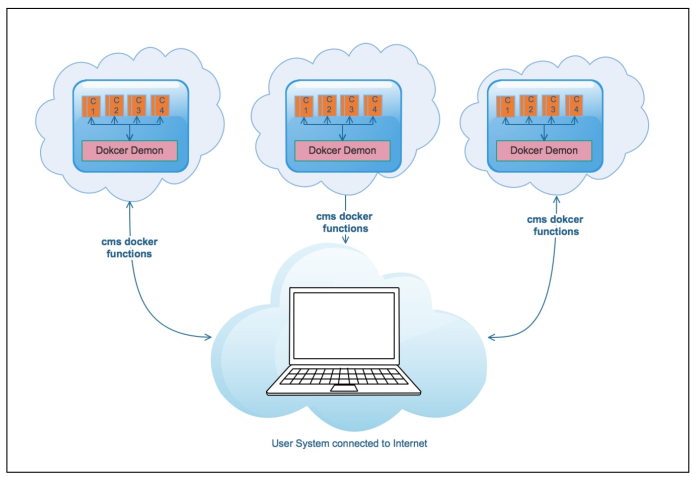


Fig. 2. Docker Mode

The Swarm module facilitates performing various swarm operations Figure 3.

Users can choose to use cloudmesh docker application from a remote terminal outside the network of the data center as in Figure2 or locally from a provisioning or configuration server inside the data center as in Figure 4

2. APPLICATION ARCHITECTURE

The architecture of the application is depicted in Figure 5.The commands developed can be broadly classified as 'action com-

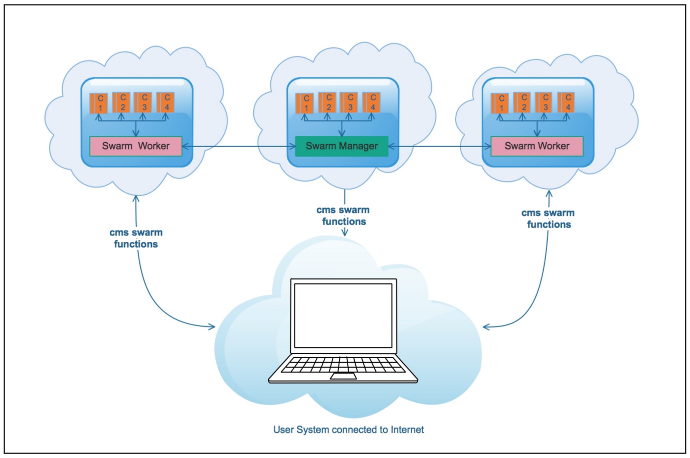


Fig. 3. Swarm Mode

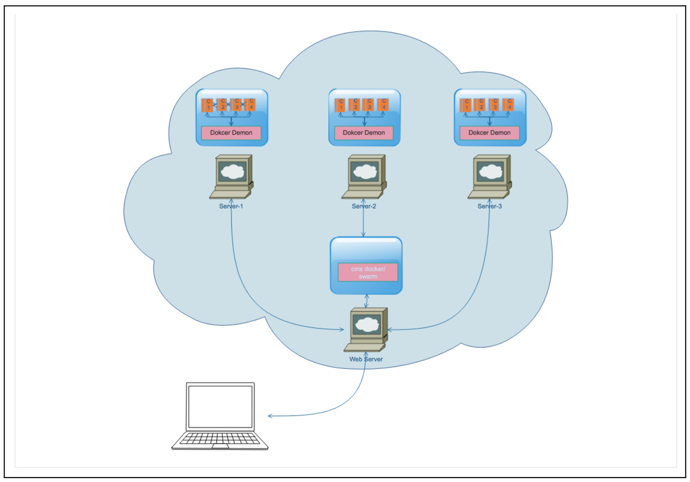


Fig. 4. Docker/Swarm Remote

mands','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
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 data base like mongoDB is that it allowed us to store the data in the the DB in the native form as returned by the Docker Api without the need for much marshaling of the data .The application uses the cloudmesh.rest repository for managing mongo and eve services.

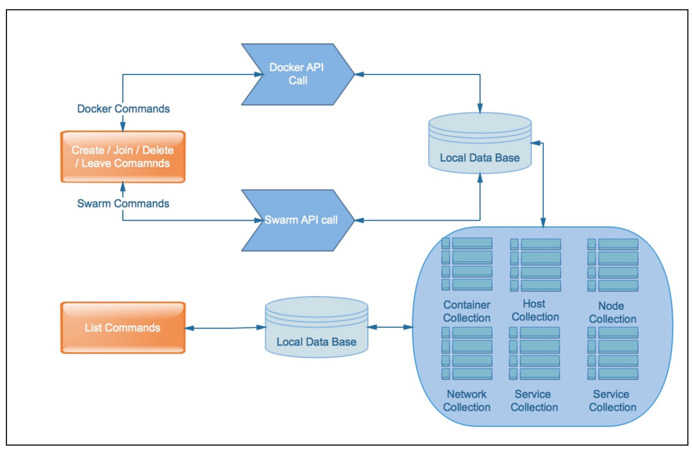


Fig. 5. cms docker extension application architecture

2.3. 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

2.4. Ansible

As part of the project we have built Ansible[7] scripts to automate 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 and IP in remote hosts which allows the applications to be configured to access by the standard host names instead of the IP address of the machine.

3. DOCKER COMMANDS

1. **Host Set/Add** This command is to be used to setup the docker host on which the user wants to operate .The docker commands following would be executed on the host setup in this step. The host details will be also captured in the database with this command.

```
cms docker host docker1 docker1:4243
```

2. **Host List** This command would list the hosts available.This command would display the Ip,Name,Port,and if the host is swarm manager and the swarm manager Ip. Please note the Swarmmanager Ip would 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 would be used to delete a 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 would be used to 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 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 would update the local data base , and display the updated reults to the user.

```
cms docker image refresh
```

6. **Container Create** This command would used to create a container on a given host.The arguments for this command would be 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 would be used to start a container.The container should be already created using the above command.

```
cms docker container start test1
```

8. **Container Stop** This command would be used to stop and container which is running.The container can be started using the above command

```
cms docker container stop test1
```

9. **Container List** This command would 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 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 would refresh the current state of the containers across the hosts, This command would connect to the host 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 would be used to delete a required container on the host. The arguments required are 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 would be used to run a container instead of creating and starting in two steps.This 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 would pause the container which is currently running.Use can run a cms docker container list to observe the status change.

```
cms docker container pause test1
```

14. **Container Unpause** This command would unpause the container which is currently paused.Use can run a cms docker container list to observe the status change.

```
cms docker container unpause test1
```

15. **Network Refresh** This command would refresh the network across the docker containers and hosts the updated results would be stored in the local database.

```
cms docker network refresh
```

16. **Network List** This command would display the results of the refreshed network.This would display the host ip where the network established , the network id , name and containers in 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 would be used to setup the current host.The docker commands following would be executed on the host setup in this step. the host details will be also captured in the database with this command.

```
cms swarm host docker1 docker1:4243
```

2. **Host List** This command would list the hosts available,This command would display the Ip,Name,Port,and if the host is swarm manager and the swarm manager Ip. Please note the Swarmmanager Ip would 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 would be used to delete a 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 swarm host delete docker1:4243
```

4. **Image List** This command would be used to 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 would create swarm on the host in use.There is no arguments required for this command, After this command is run the current host status would be treated as 'manager'.user can run a node list or host list to see the updated result.To setup the current host user needs to use cms swarm host ADDR command shown above.

```
cms swarm create
```

6. **Swarm Join** This command would be applicable for the host which is not manager,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 in the last step.User needs to pass the swarm host Ip and address the host being joined

```
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 let the host leave swarm.If manager has multiple workers ,workers needs to be removed(leave) before manager can leave.This command would treat current host is to be removed(leave) the swarm.User may need to set up the current host before processing the command.

```
cms swarm leave
```

8. **Network Create** This command would be used to create the network which can be used by the swarm containers later.The arguments it would need is the name of the containers.

```
cms swarm network create network1
```

9. **Network List** This command would display the results of the refreshed network.This would display the host ip where the network established , the network id , name and containers in 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 would refresh the network across the docker containers and hosts the updated results would be stored in the local database.

```
cms swarm network refresh
```

11. **Network Delete** This command would be used to delete an existing network.The inputs required for this command is just the network name.

```
cms swarm network delete network1
```

12. **Service Create** This command would be used to create a service,the arguments required are the image name and the name of service.This command would record the service details into the local database.

```
cms swarm service create elasticsearch \
elasticsearch:swarm
```

13. **Service List** This command would list the current services running ,the data being displayed would be from the local data base, 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 would be used to delete a running service,the arguments required are the service name.This command would delete the service details into the local database.

```
cms swarm service delete elasticsearch
```

15. **Service Refresh** This command would be used to refresh the services status based on the current condition.This command would refresh the local database so that service list would show the updated results.

```
cms swarm service refresh
```

16. **Node List** This command would display the list of the nodes across the hosts available.The results would come from the local database. The command would display the node id,,Id,Role,status and Manager Ip.

```
cms swarm node list
```

17. **Image Refresh** This command would refresh the images across the hosts available the results would update the local data base , user can run docker image list to view the updated results.

```
cms swarm image refresh
```

Table 9. cms swarm node list

Id	Ip	Role	Status	Manager Ip
5545f4e3b27e	docker3	Manager	Ready	
7645f4f4b27e	docker2	Worker	Ready	docker4

18. **Image List** This command would be used to 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 would refresh the current state of the containers across the hosts, This command would connect to the host and run the native docker container list to get the latest information and update the local database for refreshing the data.

```
cms swarm container refresh
```

20. **Container List** This command would 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. USE CASE - ELASTICSEARCH CLUSTER

Elasticsearch[8] 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 a Elasticsearch cluster on remote hosts in Chameleon cloud in docker and swarm mode .We benchmarked the cluster using esrally[9] have compared the results between the elastic search clusters in docker and swarm mode.

5.1. 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 4 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.

5.2. 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 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 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 the underlying host network as in the docker mode to enable the communication between the swarm containers we created a "overlay"

Table 12. 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

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 start of the elastic search cluster using the Servicename 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.

5.3. Elasticsearch cluster Docker and Swarm mode benchmark results

6. 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 center) and local mode (Cloudmesh docker client is run from a VM inside the cloud data center) . We performed the benchmarking for both the options on both the Amazon Webservices[10] and Chameleon cloud[11]. The results are plotted and tabulated as below

Each of the benchmark runs was performed 100 times for a defined set of operations similar to the steps performed for setting up a elastic search cluster in docker and swarm.The results were gathered as a csv file and plotted using a Ipython[12].

The hardware specifications used on both the clouds is listed below

6.1. Docker Mode - Results

Below are the categories of the bench mark results

1. Chameleon Docker Mode Local Client Figure 6

Table 13. Hardware Specification

Parameter	Chameleon	Aws
VM	3	3
OS	Ubuntu 16.04	Ubuntu 16.04
Flavor	m1.large	m1.large
Secgroup	default	default
Assign floating IP	True	True

2. Chameleon Docker Mode Remote Client Figure 7
3. Aws Docker Mode Local Client Figure 8
4. Aws Docker Mode Remote Client Figure 9

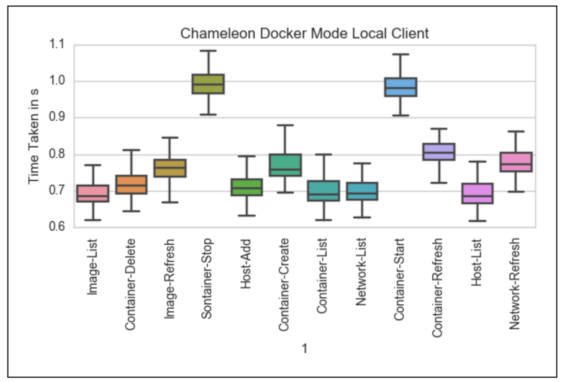


Fig. 6. Chameleon Docker Mode Local Client

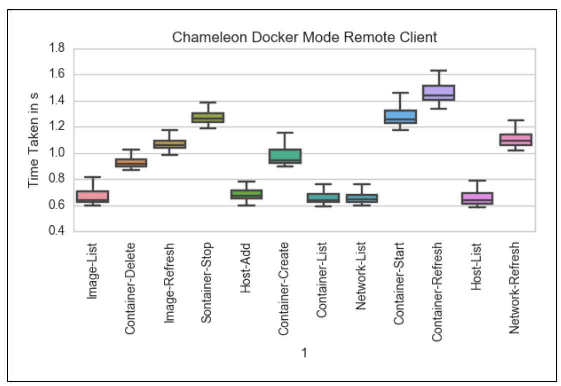


Fig. 7. Chameleon Docker Mode Remote Client

Based on the benchmark reults 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.
3. The standard deviation of the response times is significantly lower for Aws than Chameleon indicating that Aws is much

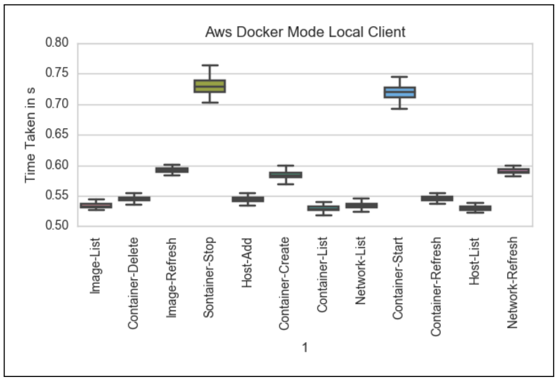


Fig. 8. Aws Docker Mode Local Client

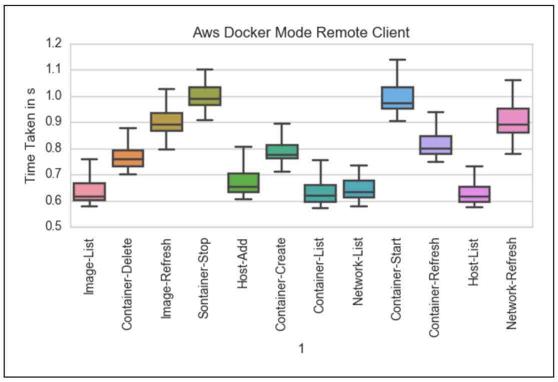


Fig. 9. Aws Docker Mode Remote Client

more stable and reliabe 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 .

6.2. 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 reults 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 reliabe in performance than the chameleon cloud

Table 14. 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

4. The mean service create times range between 2 to 1.6 s for 4 replicated containers which is significantly faster than normal boot times for a similar number of VM on the same cloud .

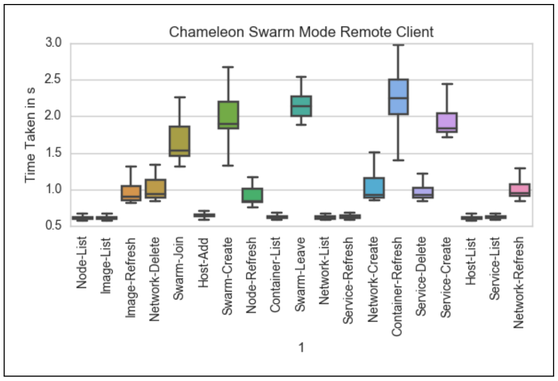


Fig. 10. Chameleon Swarm Mode Remote Client

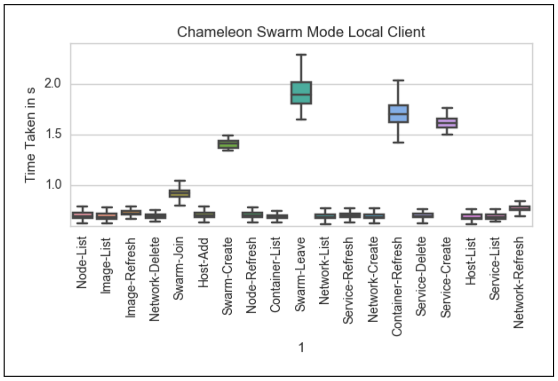


Fig. 11. Chameleon Swarm Mode Local Client

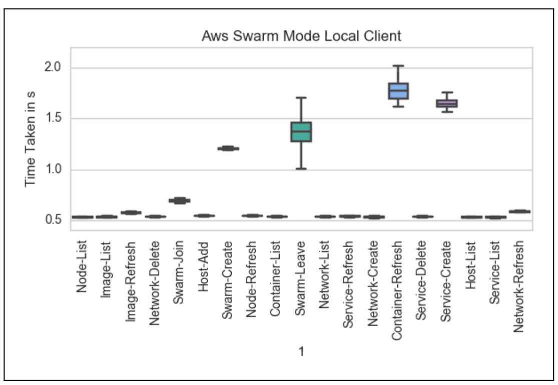


Fig. 12. Aws Swarm Mode Local Client

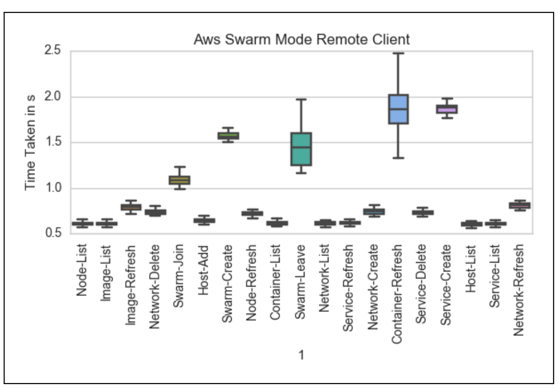


Fig. 13. Aws Swarm Mode Remote Client

Table 15. 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

7. CONCLUSION

In this project we have succefully integrated docker and swarm capabilities into cloudmesh client.We have also demonstrated its use for a practical use case of setting up a 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.

8. ACKNOWLEDGEMENT

We acknowledge our professor Gregor von Laszewski and all associate instructors for helping us and guiding us throughout this project.

9. 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

- 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 dcoument reults
  - 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 dcoument reults
  - Writing related sections in this report.

REFERENCES

[1] Docker Inc., “Docker,” Web Page, 2017, accessed 2017-04-23. [Online]. Available: https://www.docker.com/
[2] ——, “Swarm,” Web Page, 2017, accessed 2017-04-23. [Online]. Available: https://docs.docker.com/engine/swarm/
[3] 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
[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] Elasticsearch BV, “ElasticSearch,” Web Page, 2017, accessed 2017-04-23. [Online]. Available: https://www.elastic.co/
[9] D. Mitterdorfer, “ESRally,” Web Page, 2017, accessed 2017-04-23. [Online]. Available: http://esrally.readthedocs.io/en/latest/quickstart.html
[10] Amazon Web Services, Inc., “Amazon Webservices,” Web Page, 2017, accessed 2017-04-23. [Online]. Available: https://aws.amazon.com/?nc2=h_lg
[11] National Science Foundation, “Chameleon,” Web Page, 2017, accessed 2017-04-23. [Online]. Available: https://www.chameleoncloud.org/
[12] IPython development team, “Ipython,” Web Page, 2017, accessed 2017-04-23. [Online]. Available: https://ipython.org/

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, April 9, 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 1, 2, 3 and 4 clouds. Cloudmesh client would be used for cloud management and Ansible scripts would be used for software stack deployment. Appropriate benchmarking would be carried out at each iteration using some benchmarking technique or tool (TBD).

2. SOFTWARE STACK

- Ansible
- Cloudmesh
- TBD

3. EXECUTION PLAN

The execution plan that would be followed is as under:

3.1. Week 1

Deployment of the vehicle detection application on the localhost, using Ansible playbook. Benchmarking of the important factors that are chosen for observation. Analysis of the benchmarking results thus obtained.

3.2. Week 2

Reservation of a single Chameleon cloud using cloudmesh client. Deployment of the vehicle detection application on the cloud, using Ansible playbook. Benchmarking of the important factors that are chosen for observation. Analysis of the benchmarking results thus obtained.

3.3. Week 3

Reservation of 2 Chameleon clouds using cloudmesh client. Deployment of the vehicle detection application on the clouds, using Ansible playbook. Benchmarking of the important factors that are chosen for observation. Analysis of the benchmarking results thus obtained.

3.4. Week 4

Reservation of 3 and 4 Chameleon clouds using cloudmesh client. Deployment of the vehicle detection application on the clouds, using Ansible playbook. Benchmarking of the important factors that are chosen for observation. Analysis of the benchmarking results thus obtained.

3.5. Week 5

Final analysis of the benchmarking results obtained so far. Studying the effect of software stack deployment on multiple clouds. Publishing the final report and conclusion.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.

4. DEPLOYMENT

As per the current plan, we would first identify the software stack that would be required to run the vehicle detection application on the localhost, which is a Ubuntu 16.04.02 machine with 3 GB RAM. This software deployment would be done using an Ansible script [2]. Ansible is an automation engine which we would use to orchestrate the software deployment via playbooks using the inventory.txt file. Important factors like the time required for deployment, the 'ease' of deployment (TBD), etc. would be benchmarked using some benchmarking technique or tool (TBD).

The next step would be to deploy this software stack on a single cloud. We would be using the cloudmesh client [3], a command line based client, to reserve a cloud and enable access to it. We would consider the feasibility of deploying the software stack on the cloud, by careful observation of the benchmarking results obtained in the previous step of deploying it on the localhost. Depending upon the outcomes, we would be able to identify the minimum system requirements that would be typically required for the deployment of the software stack. This deployment would again be achieved by using Ansible scripts [2]. These results would again we benchmarked for further analysis.

As the next step, we would carry out this deployment on two clouds. Ansible would again be used for deployment of the software stack and cloudmesh client for the management of the clouds. The results obtained would again be benchmarked and analysis performed. We would iterate this for 3 and 4 clouds. More clouds might be considered depending upon the time constraints of the project.

Finally the observations and analyses would be published as a report.

5. BENCHMARKING RESULTS AND ANALYSIS

TBD

6. CONCLUSION

TBD

7. ACKNOWLEDGEMENTS

TBD

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/progrium/buildstep

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.*

project-000, April 9, 2017

By deploying our face detection application utilizing Apache Mesos, we will try to achieve high throughput by parellalizing processing of images for head count task on multiple nodes each having thousand of pictures.

Keywords: Cloud, I524

https://github.com/cloudmesh/sp17-i524/tree/master/project/S17-IR-P011

1. INTRODUCTION

Counting the number of people in a image has been a challenge in the field of computer vision [1]. Given a huge number of images, finding the number of people in each image can become a cumbersome task. We try to solve this issue using our distributed approach utilizing power of Apache Mesos [2] , where services have to be deployed (TBD). We run the OpenCV [3] face detection algorithms on smaller data in multiple nodes and obtain the head count of the people in each picture.

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 [4].

ACKNOWLEDGEMENTS

This project is undertaken as part of I524: Big Data And Open Source Software Projects at Indiana University, Bloomington. We would like to Prof. Gregor von Laszewski and Assosiate Instructors for their help.

REFERENCES

[1] Wikipedia, "Face detection," Web Page, Feb. 2017, online; accessed 09-March-2017. [Online]. Available: https://en.wikipedia.org/wiki/Face_detection
[2] Apache Software Foundation, "Apache mesos," Web Page, Mar. 2014, accessed 2017-04-01. [Online]. Available: http://mesos.apache.org/
[3] Wikipedia, "OpenCV," Web Page, Feb. 2017, online; accessed 09-March-2017. [Online]. Available: https://en.wikipedia.org/wiki/OpenCV
[4] 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

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, April 16, 2017

Optical Character Recognition is a technology for converting images into machine encoded text format. In this project, the input data is in PNG format 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 is a classification algorithm which will be implemented using Python programming language. The algorithm will be deployed using the Ansible technology [1] to remote virtual clusters.

Keywords: OCR,ansible,classification

https://github.com/SushmitaSivaprasad/sp17-i524/tree/master/project/S17-IR-P012/report/report.pdf

1.INTRODUCTION

This project proposal provides an overview on how we plan on implmenting the OCR technology. It gives a background on the kind of technology that has been used , delving into some of the basic concepts used in the implentation process.We have also discussed some important applications of this technology in the real world.

2.BACKGROUND

2.1 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 papers (raster images) or PDF into an editable and searchable form [2]. The images can be in either the basic black & white or multicolored. The technology first analyzes the structure of the document and divides it into smaller segments. 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.

2.2 Ansible

Ansible is an IT automation tool. It uses YAML in order to issue the state of the server [2]. 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 across 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.

2.3 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.

3.ANSIBLE DEPLOYMENT

We will be using Ansible [1] for running the OCR algorithm. The jobs will be collected and organized in a Playbook [3] and run on virtual clusters provided by Chameleon Cloud [4]. The tasks will include Installing the essential libraries on the remote machine and running the program.

4.OCR IMPLEMENTATION

Optical Character Recognition have already been developed in numerous ways, focusing on different goals. For our purpose, various classification algorithms such as K-Nearest Neighbor and Neural Network (multilayer perceptron) can be used. For this project, a feedforward, back-propagation Neural Network will be used. The steps 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 neural network and they will be compared with stored glyph features and the nearest match will be chosen, after multiple iterations of classification by the network.

5.PRE PROCESSING THE DATA

Preprocessing Techniques :

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

5.1 Binarization

Otsu's method [5] 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. .

5.2 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

5.3 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.

5.4 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. 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.[6]Figure 2 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.

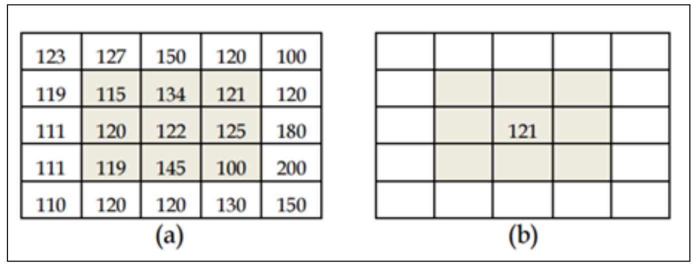


Fig. 1. Averaging of a pixel in median filter [6]

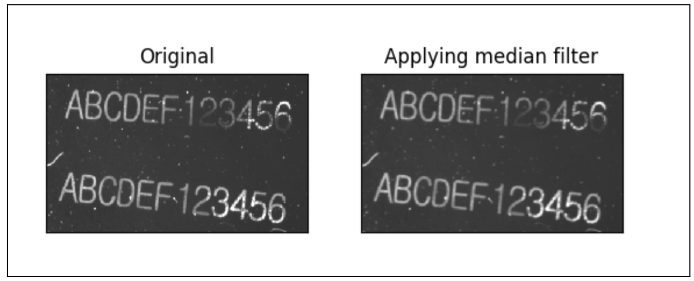


Fig. 2. After applying median filter on a scanned image

6.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[7]. 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.

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.

REFERENCES

[1] "Ansible," Web Page. [Online]. Available: https://www.ansible.com/
[2] "What is OCR and OCR Technology," Web Page, 2017. [Online]. Available: https://www.abbyy.com/en-us/finereader/what-is-ocr/
[3] "Playbook," Web Page, Mar. 2017. [Online]. Available: http://docs.ansible.com/ansible/playbooks.html
[4] "A configurable experimental environment for large-scale cloud research," Web Page, Jan. 2017. [Online]. Available: https://www.chameleoncloud.org/
[5] N. Otsu, "A threshold selection method from gray-level histograms," *Automatica*, vol. 11, no. 285-296, pp. 23–27, 1975.
[6] Y. Alginahi, "Preprocessing techniques in character recognition," pp. 9–10. [Online]. Available: http://cdn.intechopen.com/pdfs/11405.pdf
[7] "Optical Character Recognition," Web Page, Mar. 2017. [Online]. Available: https://en.wikipedia.org/wiki/Optical_character_recognition

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-000, April 9, 2017

The project aims to analyze any relationship between change in climate, geo- magnetic field and natural disasters with focusing on use of Hadoop Framework for data analysis and Ansible for automating deployment and monitoring.

Keywords: Cloud, I524

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

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 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

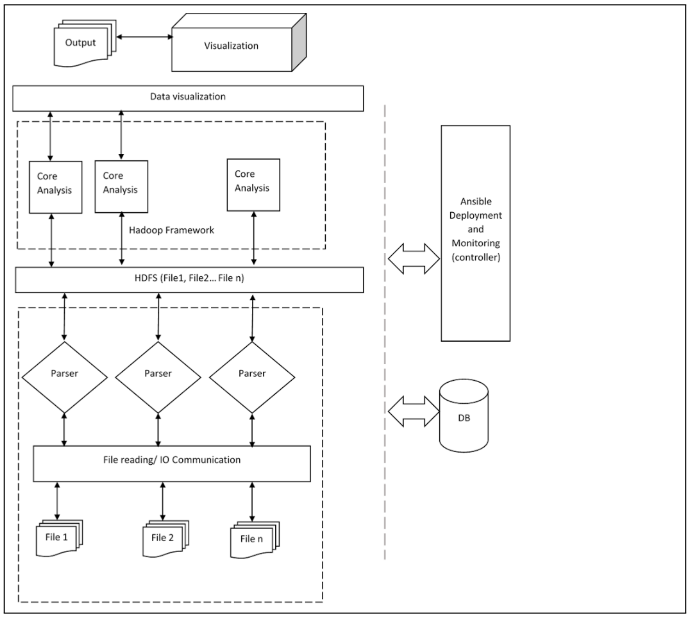


Fig. 1. Architecture

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.

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

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, April 24, 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

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.

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.

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.

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.

SOFTWARE STACK

Following are the deployment and analysis tools used in the project.

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.

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.

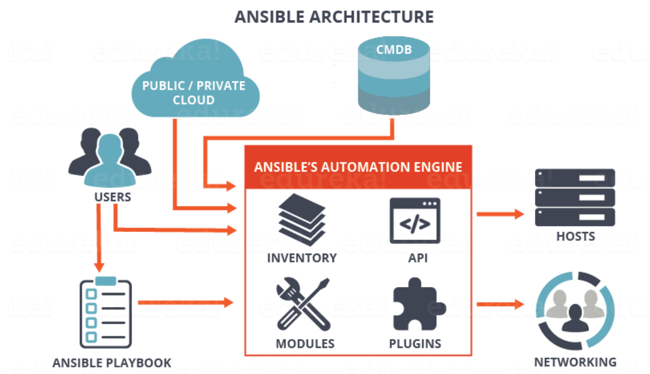


Fig. 1. Ansible Architecture [5]

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. Apache Spark consists of a graph processing component known as GraphX. It depends on RDDs and generally used for graphs that are immutable.

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

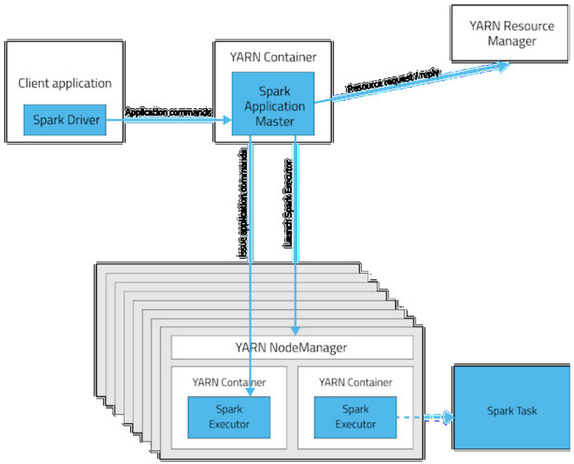


Fig. 2. Spark with Yarn Architecture
[7]

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.

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.

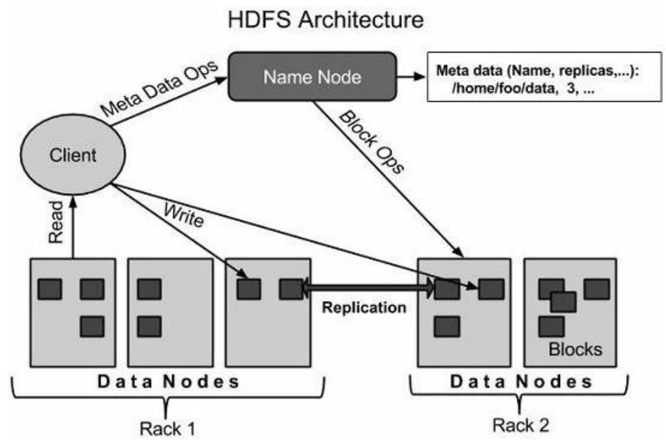


Fig. 3. HDFS Architecture
[12]

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.

YARN

Yet Another Resource Negotiator (YARN) is a technology used for cluster management. Hadoop supports a broad range of applications 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.

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.

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.

- Similar to the defined cluster, 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.

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.

The method of analysis goes as follows 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 we uniquely various transformations and actions to get the output. The process is transform the flights rdd by applying filters and map functions to get the delay based on two instances i.e, airports and airlines. Then we performed ReduceByKey and CombineByKey actions by aggregating and computing the average delay in case of each airport, and applied sorting function to sort the output in descending order and based on that we extracted the top ten airports to avoid provided the average delay per airport. Since, only the codes of various airlines and airports are available, lookup operations are performed by using countAsMap() operation with airports and airlines dataset and by using broadcast various the lookup information is passed on to all workers and executors within them. Also, finding the top ten airlines to avoid by computing the total minutes of delay per airline over a period of time. The following analysis can be further improvised by used various Machine Learning techniques which helps us to predict the delays over a period of time ahead and gives us various insights which helps us to make better decisions in choosing airlines and airports to commute.

TIMELINE

BENCHMARKING

Benchmarking is the process that is carried out after the deployment and data analysis. It is done to evaluate the efficiency and performance of the system.

RESULTS

WORK BREAKDOWN

CONCLUSION

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, April 16, 2017

This project focuses on deployment of Apache Storm using Ansible playbook on Chameleon Cloud VM, future systems cloud and benchmarking of the deployment.

Keywords: Storm, Ansible, Java, Python

https://github.com/cloudmesh/classes/blob/master/project/S17-IR-P015/report/report.pdf

INTRODUCTION

Apache Storm is a distributed stream processing computation framework under Apache. In this project, Storm cluster of one or more Chameleon cloud VMs and future sytems cloud is deployed using Ansible playbook and benchmarking is done to measure the time it took for deployement using a TBD benchmarking tool[1][2].

APACHE STORM

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, and is a lot of fun to use!. Storm has many use cases: realtime analytics, online machine learning, continuous computation, distributed RPC, ETL, and more. Storm is fast: a benchmark clocked it at over a million tuples processed per second per node. It is scalable, fault-tolerant, guarantees your data will be processed, and is easy to set up and operate.Storm integrates with the queueing and database technologies you already use. A Storm topology consumes streams of data and processes those streams in arbitrarily complex ways, repartitioning the streams between each stage of the computation however needed. Read more in the tutorial[1][3].

MILESTONES

- Performing Analysis on local VM
- Deploying Storm and Hadoop on FutureSystems and Chameleon Cloud
- Analysis on the distributed cloud environment
- Benchmarking
- Final update with report

TECHNOLOGIES

- Distributed Computation and Storage:- Storm
- Development:- Python and Java
- Deployment:- Ansible

DEPLOYMENT

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.[4][2]

BENCHMARKING

TBD

REFERENCES

[1] "Apache Storm." [Online]. Available: http://storm.apache.org/
[2] "Apache ZooKeeper - Releases." [Online]. Available: http://zookeeper.apache.org/releases.html
[3] "Storm (event processor) - Wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Storm_(event_processor)
[4] "Apache Storm." [Online]. Available: http://storm.apache.org/

Machine Learning for Customer churn prediction using big data analytics

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, April 22, 2017

This project involves use of machine learning algorithms to identify customers who are most likely to discontinue using the service or product.

Keywords: Prediction, Bigdata, Apache Spark, MLlib, Hadoop, Analytics

https://github.com/cloudmesh/sp17-i524/tree/master/project/S17-IR-P016/report/report.pdf

CONTENTS

INTRODUCTION

We will use Apache Spark[1] machine learning library for fitting a predictive model on a massive dataset. Detailed analysis and modeling will be carried out in Python Programming language.

EXECUTION SUMMARY

The tentative schedule for this project has been outlined below:

1. March 13-March 19, 2017: Create virtual machines on Chameleon, FutureSystems and Jetstream clouds
2. March 13-March 19, 2017: Deploy Hadoop cluster to the clouds and install the required software packages to the clusters and also finalize data.
3. March 20-March 26, 2017: Data Preprocessing and applying transformation to extract features from the data.
4. March 27-April 09, 2017: Use MLlib to train and evaluate various machine learning algorithms and choose best based on various performance metrics.
5. April 10 - April 16, 2017: Create deployable software packages in Python.
6. April 17-April 23, 2017: Complete Project Report.

WORKFLOW

The project will make use of the following four components.

1. Apache Spark
2. Hadoop
3. Spark MLlib

DEPLOYMENT

We will deploy our application using Ansible[2] playbook. Deployment of Master/slave nodes will be done hadoop/spark distributed cluster environment. Different cloud systems that will be used in the project include Chameleon,FutureSystems and JetStream.

BENCHMARKING

Performance of the Hadoop/Spark clusters deployed on different clouds will compared for benchmarking.

CONCLUSION

TBD

ACKNOWLEDGEMENT

TBD

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