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 23, 2017

Apache Zeppelin is an open source notebook for data analytics and visualization. In this project we deploy Apache Zeppelin in cluster and visualize data stored in Spark across cluster using Apache Zeppelin interpreter that employs Python and Scala in same notebook.

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 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 notebook it is has certain limitations when used for visualizations when used 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 support Python, but also 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 aim to deploy Zeppelin 0.7 along with in built Spark and backend languages R and Python across cluster using Ansible. Then install additional visualization packages provided by Apache Zeppelin Helium APIs.

We also aim to load 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 will be focusing more on that through out the paper and will give less importance to the analytics that are performed after the deployment.

2. 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.

2.1. 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.

2.2. 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.

2.3. 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

2.4. 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.

2.5. 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.

2.6. April 17,2017 - April 23,201

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

3. INFRASTRUTURE

The deployment of Apache Zeppelin is done on 2 clouds. The clouds selected for the purpose of this project are

1. Chameleon Cloud
2. JetStream Cloud

3.1. OpenStack

OpenStack[2] is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacenter, 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 laaS.[3]

3.2. Chameleon Cloud

Chameleon Cloud[4] provides a large-scale platform to the open research community allowing them 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.

3.3. 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[5], use virtual desktops and publish VMs with a Digital Object Identifier(DOI)[6].

4. DEPLOYMENT

4.1. Deployement process

—- Naveen—- Introduction ansible galaxy and detailed explanation on zeppelin was deployed

4.2. Security

4.3. Deployment Timing

5. APACHE ZEPPELIN

Apache Zeppelin is Apache project under open-source license Apache2. 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 addded.

5.1. 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.

5.2. Current Status

Currently, Apache Zeppelin multipurpose notebook supports the following needs.

1. Data Ingestion
2. Data Discovery

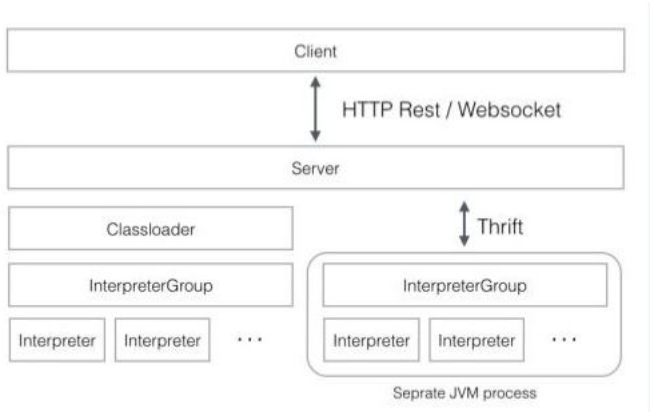


Fig. 1. Zeppelin Architecture

3. Data Analytics
4. Data Visualization
5. Collaboration

5.3. Zeppelin Architecture

5.4. Multiple Language Backend

Apache Zeppelin interpreter concept allows any language/data-processing-backend to be plugged into Zeppelin. Currenlty 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

5.5. 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.

5.6. 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

6. DATASET DESCRIPTION

This is real-world dataset[7] 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.

7. 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 in the 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.

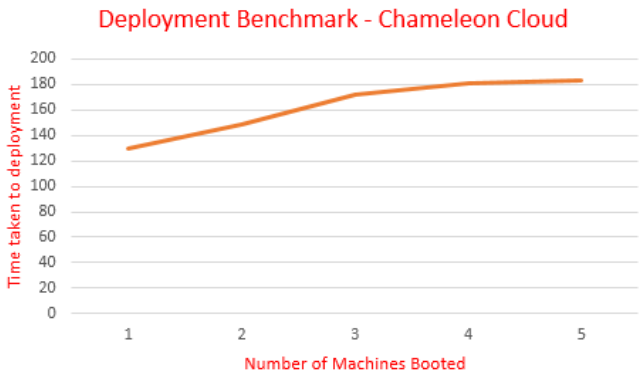


Fig. 2. Jetstream Deployment Benchmarking

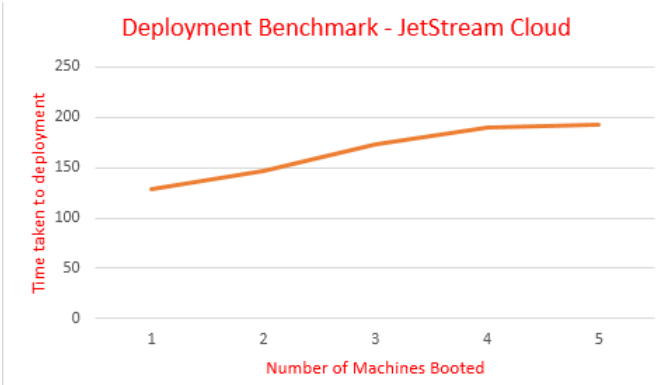


Fig. 3. Jetstream Deployment Benchmarking

Analytical Benchmarking

This benchmarking deals with the time taking for running the analytics on the clouds. The same analytics are performed on all the 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 doesnot allow parallel boot of virtual machines the boot time is neglected in the deployment benchmarking

7.1. 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.

7.1.1. 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 times 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 is process is reflected in the deployment graph shown for chameleon cloud.

7.1.2. 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	coode#3
1	43	11	5
2	34	10	4
3	28	8	3
4	25	6	3

7.2. 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.

7.2.1. 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

7.2.2. 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.

Table 2. Analytical Benchmarking Jetstream Cloud Time taken to run codes Vs Machines Count

VM Count	Code#1	Code#2	coode#3
1	40	10	4
2	33	8	4
3	25	7	3
4	23	6	2

8. VISUALIZATION WITH ZEPPELIN

9. SUPPLEMENTAL MATERIAL

TBD

ACKNOWLEDGEMENTS

TBD

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] Open Stack, "Open Stack," Web Page, Apache Software Foundation, Apr. 2017. [Online]. Available: https://www.openstack.org/software/

[3] LaaS, "LaaS," Web Page, Apache Software Foundation, Apr. 2017. [Online]. Available: https://en.wikipedia.org/wiki/Logging_as_a_service

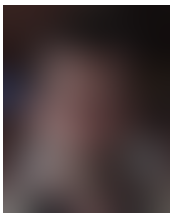
[4] Chameleon Cloud, "Chameleon Cloud," Web Page, Apache Software Foundation, Apr. 2017. [Online]. Available: https://www.chameleoncloud.org/about/chameleon/

[5] Globus Transfer, "Globus Transfer," Web Page, Apache Software Foundation, Apr. 2017. [Online]. Available: https://www.chameleoncloud.org/about/chameleon/

[6] DOI, "DOI," Web Page, Apache Software Foundation, Apr. 2017. [Online]. Available: https://en.wikipedia.org/wiki/Digital_object_identifier

[7] 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.

AUTHOR BIOGRAPHIES



Naveenkumar Ramaraju yet to update his Bio

Veera Marni yet to update his bio