Deployment of a Storm cluster

VASANTH METHKUPALLI[1,*] AND AJIT BALAGA[1,**]

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

project-P015, May 10, 2017

Apache Storm is a free and open source distributed realtime computation system. Storm makes it easy to reliably process unbounded streams of data, doing for realtime processing what Hadoop did for batch processing. Storm is simple, can be used with any programming language. Storm has many use cases: realtime analytics, online machine learning, continuous computation, distributed RPC etc. Storm is really fast at data processing: a sample benchmark clocked it at over a million tuples processed per second per node. It is scalable, fault-tolerant, guarantees that data will be processed, and is easy to set up and operate. Storm integrates with the queueing and database technologies we already use. Storm is currently being used to run various critical computations in Twitter at scale, and in real-time, this led us to explore and deploy it on various cloud. In this paper we try to deploy storm on various clouds and benchmark the performance on various data, doing real time processing on sample datasets. First, we describe the architecture of Storm, its deployment and its methods for distributed scaleout and fault-tolerance. Storm is in active development at Twitter.

Keywords: Storm, Ansible, Java, Python

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

1. INTRODUCTION

Currently modern data processing environments require processing complex computation on streaming data in real-time. Places like Twitter where each interaction with a user requires making a number of complex decisions, often based on data that has just been created, this mandates for a real time data processing system, Storm currently delivers on this account and provides many other services which we will see in the coming sections. Storm is designed to be, some of these things we observed while running our sample projects for deployment:

1. Scalable : Nodes may be easily added or removed from the Storm cluster without disrupting existing data flows through Storm topologies see Fig 3.

2. Resilient : Fault -tolerance is crucial to Storm as it is often deployed on large clusters, and hardware components can fail. The Storm cluster must continue processing existing topologies with a minimal performance impact.

3. Extensible : Storm topologies may call arbitrary external functions (e.g. looking up a MySQL service for the social graph), [1] and thus needs a framework that allows extensibility.

4. Efficient : Since Storm is used in real-time applications, it must have good performance characteristics. Storm uses a number of techniques, including keeping all its storage and computational data structures in memory.

5. Easy to Administer : A critical part of storm features or development is that it should be easy to administer. Given that there a lot of computations going on at every stage, tools should be developed which warn the user and development team of any major conflicts arising.

2. DATA MODEL AND ARCHITECTURE

Storm data processing architecture consists of streams of tuples flowing through topologies [2] . A topology is a directed graph where the vertices represent computation and the edges represent the data flow between the computation components. Vertices are further divided into two disjoint sets, spouts and bolts. Spouts are tuple sources for the topology. Typical spouts pull data from queues, such as Kafka [3] or Kestrel. On the other hand, bolts process the incoming tuples and pass them to the next set of bolts downstream. Note that a Storm topology can have cycles. From the database systems perspective, one can think of a topology as a directed graph of operators.

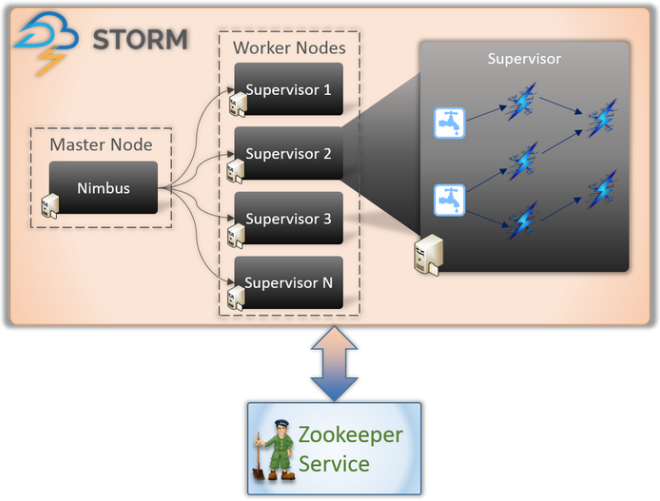

Fig. 1. Storm Architecture

2.1. Storm Overview

Storm runs on a distributed cluster. Clients submit topologies to a master node, called the Nimbus. The nimbus is responsible for distributing and coordinating the execution of the topology. The actual work is done on worker nodes. Each worker node runs one or more worker processes. At any point in time, a single machine may have more than one worker processes, but each worker process is mapped to a single topology see Fig 1. Note more than one worker process on the same machine may be executing different part of the same topology. Each worker process runs a JVM, in which it runs one or more executors. Executors are made of one or more tasks. The actual work for a bolt or a spout is done in the task. Thus, tasks provide intrabolt or intraspout parallelism, and the executors provide intratopology parallelism. Worker processes serve as containers on the host machines to run Storm topologies. Note that associated with each spout or bolt is a set of tasks running in a set of executors across machines in a cluster. Data is shuffled from a producer spout or bolt to a consumer bolt (both producer and consumer may have multiple tasks). This shuffling is like the exchange op erator in parallel databases.

Storm supports the following types of partitioning strategies [2]:

1. Shuffle grouping, which randomly partitions the tuples.
2. Fields grouping, which hashes on a subset of the tuple attributes or fields.
3. All grouping, which replicates the entire stream to all the consumer tasks.
4. Global grouping, which sends the entire stream to a single bolt.
5. Local grouping, which sends tuples to the consumer bolts in the same executor.

The partitioning strategy is extensible and a topology can define and use its own partitioning strategy. Each worker node runs a Supervisor that communicates with Nimbus. The cluster state is maintained in Zookeeper [4], and Nimbus is responsible for scheduling the topologies on the worker nodes and monitoring the progress of the tuples flowing through the topology. Loosely, a topology can be considered as a logical query plan from a database systems perspective. As a part of the topology, the programmer specifies how many instances of each spout and bolt must be spawned. Storm creates these instances and also creates the interconnections for the data flow. We note that currently, the programmer has to specify the number of instances for each spout and bolt. Part of future work is to automatically pick and dynamically changes this number based on some higher-level objective, such as a target performance objective.

2.1.1. Storm Internal Architecture

In this section, we describe the key components of Storm), and how these components interact with each other.

2.1.2. Nimbus and Zookeeper

Nimbus plays a similar role as the "JobTracker" in Hadoop, and is the touchpoint between the user and the Storm system. Nimbus is an Apache Thrift service and Storm topology definitions are Thrift objects. To submit a job to the Storm cluster (i.e. to Nimbus), the user describes the topology as a Thrift object and sends that object to Nimbus. With this design, any programming language can be used to create a Storm topology.

As part of submitting the topology, the user also uploads the user code as a JAR file to Nimbus. Nimbus uses a combination of the local disk(s) and Zookeeper to store state about the topology. Currently the user code is stored on the local disk(s) of the Nimbus machine, and the topology Thrift objects are stored in Zoo keeper.

The Supervisors contact Nimbus with a periodic heartbeat protocol, this comes in very handy so we can periodically verify ifthe nodes are working, advertising the topologies that they are currently running, and any vacancies that are available to run more topologies. Nimbus keeps track of the topologies that need assignment, and does the match -making between the pending topologies and the Supervisors.

All coordination between Nimbus and the Supervisors is done using Zookeeper. Furthermore, Nimbus and the Supervisor daemons are fail- fast and stateless, and all their state is kept in Zookeeper or on the local disk(s). This design is the key to Storm's resilience. If the Nimbus service fails, then the workers still continue to make forward progress. In addition, the Supervisors restart the workers if they fail.

However, if Nimbus is down, then users cannot submit new topologies. Also, if running topologies experience machine failures, then they cannot be reassigned to different machines until Nimbus is revived. An interesting direction for future work is to address these limitations to make Storm even more resilient and reactive to failures. All the above workings, whether Nimbus and Zookeeper are working properly are viewed in the UI of the storm deployment, and can be viewed from the localhost of that particular node.

2.1.3. Supervisor

The supervisor runs on each Storm node. It receives assignments from Nimbus and spawns workers based on the assignment. It also monitors the he alth of the workers and respawns them if necessary. The main thread reads the Storm configuration, initializes the S upervisor's global map, creates a persistent local state in the file system, and schedules recurring timer events. There are three types of events, which are:

1. The heart beat event, which is scheduled to run every 15 seconds, and is runs in the context of the main thread. It reports to Nimbus that the supervisor is alive.

2. The synchronize supervisor event, which is executed every 10 seconds in the event manager thread. This thread is responsible for managing the changes in the existing assignments. If the changes include addition of new topologies, it downloads the necessary JAR files and libraries, and immediately schedules a synchronize process event .

3. The synchronize process event, which runs every 3 seconds under the context of the process event manager thread. This thread is responsib le for managing worker processes that run a fragment of the topology on the same node as the supervisor. It reads worker heartbeats from the local state and classifies those workers as either valid, timed out, not started, or disallowed. A "timed out" work er implies that the worker did not provide a heart beat in the specified time frame, and is now assumed to be dead. A "not started " worker indicates that it is yet to be started because it belongs to a newly submitted topology, or an existing topology who se worker is being moved to this supervisor. Finally, a "disallowed " worker means that the worker should not be running either because its topology has been killed, or the worker of the topology has been moved to another node.

table must be cited

3. TECHNOLOGIES

Table 1. TABLE CAPTION MISSING

Usage	Technologies Used
Distributed Computation and Storage:	Storm
Development:	Python and Java
Deployment:	Ansible, Bash Shell script
Project Repository:	GitHub
Document Preparation:	LaTex

4. DEPLOYMENT AUTOMATION

4.1. Automation with Ansible

Ansible Playbook is used as the application and configuration deployment tool. Deploying the hadoop and spark framework into the cluster environment. Ansible will help push configurations to the environment automatically based on playbooks written for various configurations. For this project, we used Ansible to automate the deployment of storm, zookeeper and other prequisites. The Ansible script is written such that we can leverage the cloudmesh client technology to deploy the spark cluster. When creating Zookeeper cluster we had a communication issue, similar issue was found when creating a Storm cluster, we resolved this issue by uploading a secgroup with the details necessary and added it to the cluster.

The Ansible playbook package constitutes the following files.

- ansible.cfg: This file contains all the configuration information necesasary for the storm deployment. In our project we refer to the hosts file to look up for the ips necessary for deployment.

- hosts: Efforts are made to automate the generation of this file, however currently the user after creating a cluster in the cloudmesh client has to update the information the user ip addresses and id's in three columns, chameleon, nimbus and supervisors fields. This enables the user to select the ip address in which they desire to have the nimbus node and all other configuration.

- install.yml: However, the above step is one of the few steps we had to configure manually, however, running the command ansible-playbook install.yml, installs all the dependencies and packages necessary for strom deployment. However, the install sets up the cluster with supervisor and nimbus nodes. The startup of these services however are to be done manually by the below bash scripts.

- nimbus.sh: This script is intended to start the nimbus node to receive all the packets(data) for processing.

- startStorm.sh: This starts the storm cluster.

- submit.sh Used to submit jobs to run, in the storm cluster.

- start-zookeeper.yml: This file is still in developmental stage where we want to automate the above two steps, however, there is a small problem in exiting the storm cluster after once it has started. We are working on it to fix this.

- supervisor.sh: This script file is used to start the worker nodes on which the data processing can run

- ui.sh: Running this script file will open the localhost, where we can view the current jobs running, topologies, cluster information, spouts, bolts, etc. Sample screenshots showing the working of this are shown in the benchmarking figures.

- In the templates folder, we have hosts.j2, myid.j2, storm.yaml.j2, zoo.cfg.j2, these files are necessary for proper deployment of storm and zookeeper cluster, so we created templates of the configuration files and intend to use them.

4.2. Bash Shell Script

First we automated the entire deployment using python shell script, and later changed the deployment by using Ansible playbooks. In the python shell script, we have mainly used 4 files(.sh) to automate the entire process, from updating the apps directory to configuring the host file in the zookeeper cluster. As a part of our project we are submitting even the python shell script files for everyone to view the initial stages of project development.

5. CLOUDMESH

Cloudmesh client is a simple client to enable access to multiple cloud environments form a command shell and command-line[reference for cloudmesh].The entire application is built on python which essentially needs no prerequisite knowledge and hence could see this as a ready-to-use tool. For our project since we need access to clouds for deployment purposes this comes as a welcome tool to ease us with the entire deployment process. Many thanks to Gregor von Laszewski and others collaborators for supporting our project directly and indirectly in the form of this tool which enabled us to deploy the project easier without any hassles. For this project, have tested the installation on Ubuntu 16.04.

6. CLOUD DEPLOYMENT

We selected three clouds for deployment of our project: Chameleon Cloud, Jetstream, and Kilo. In our automated deployment and benchmarking process, first we create a cluster of particular number of required nodes, update the ip addresses and the user id of all the clouds, then we run the ansible script to deploy the entire project on the cloud. However, lot of improvements can be made, such as automating the process of ip addresses for the cluster deployment and exiting the cloud by deleting the files and the cloud resources once deployed and benchmarked. The entire process of deploying the project on various clouds have been tried and tested, various times were measure on the context of installation time right from the beginning. Based on the above results we benchmarked our results on various cluster sizes.

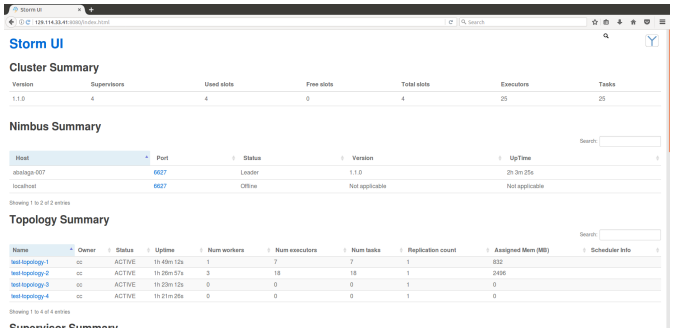

Fig. 2. Index Page

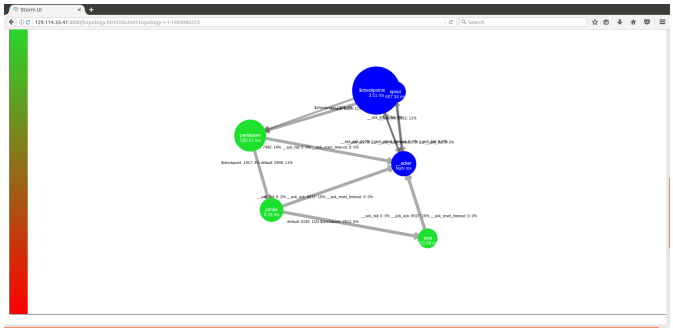

Fig. 3. Sample Topology on a 5-node cluster

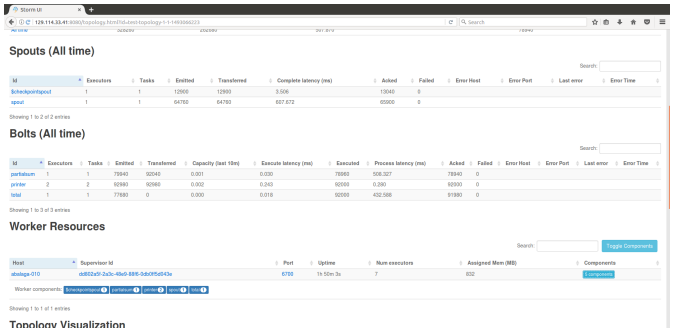

Fig. 4. Worker nodes and description on a 5-node cluster

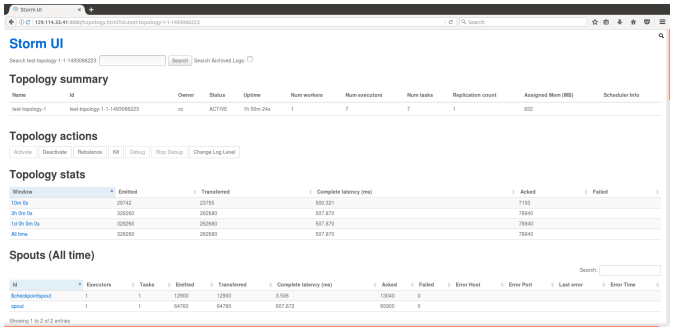

Fig. 5. Toplogy Summer on a 5-node cluster

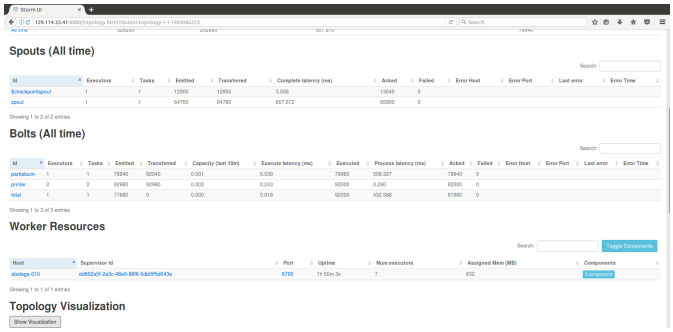

Fig. 6. Spouts and bolts summary on a 5-node cluster

7. BENCHMARKING

7.1. Chameleon Cloud

Following are the results of our benchmarking on deploying the storm cluster on Chameleon cloud. Few things, we noticed that a 3-node cluster is not viable for running storm jobs and hence we ran all our jobs on 5-nodes or above clusters see Fig 4, this was one error we came across when we tried deploying a 3-node cluster and overcame this by following code blogs related to troubleshooting mentioned in the code references and solved the issue see Fig 6. We ran a sample-storm job on a 5node cluster to see if its working, following are the snapshots where in we can see the UI of storm, illustrated in see Fig 2, a sample toplogy is also shown in the UI in Fig 3, also showing the resources used among many other things. Similar tests were made on a 7-node and a 9-node cluster and results were stored in a log page see Fig 5.

Table 2. Hardware Specifications of Chameleon

	Chameleon
CPU	Xeon X5550
cores	1008
speed	2.3GHz
RAM	5376GB
storage	1.5PB

Also we calculated the time it took to deploy a 5-node, 7-node and a 9-node cluster on the chameleon cloud. Following are the results obtained and can be illustrated below.

TABLE MUST BE CITED

Table 3. TABLE CAPTION

Install Time	5 node cluster	7 node cluster	9 node cluster
Real:	143.91	188.19	387.536
User:	13.928	28.432	36.192
Sys:	3.964	8.872	9.988

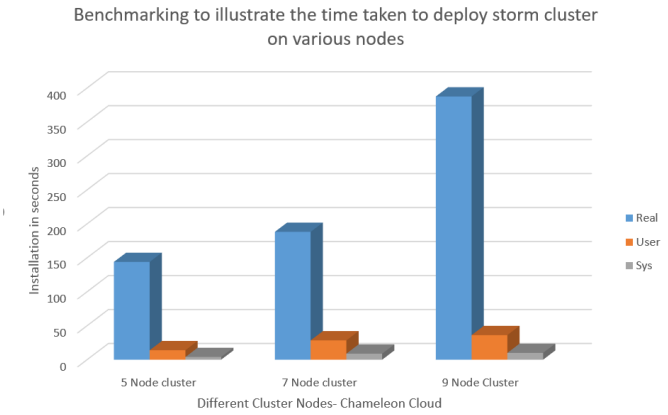

Fig. 7. Bar diagram to compare the time taken to deploy on Chameleon

7.2. JetStream

Automated the deployment of cloud on Jestream cloud resources on various cluster node sizes of 5, 7 and 9 just like we did in the case of chameleon cloud. Below is a tabluar description of the image flavor used on Jetstream and time taken to deploy the cluster on Jetstream.

Table 4. Virtual Machine Configuration for Jestream

VM Flavor Name	m1.medium
vCPUs	6
RAM	16GB
Local Storage	60GB

Below is the bar description of the various times it took in deploying various size clusters.

7.3. Kilo- FutureSystems

Automated the deployment of our project on Kilo cloud based on various cluster node sizes of 5, 7 and 9 just like we did in the case of chameleon cloud. Below is a tabluar description of the image flavor used on Kilo and time taken to deploy the cluster on Kilo.

THIS IS NOT HAW A TABLE GOES INTO A PAPER

Table 5. CAPTION MISSING

Install Time	5 node cluster	7 node cluster	9 node cluster
Real:	478.426	592.267	798.894
User:	24.232	38.14	46.208
Sys:	5.988	11.54	13.568

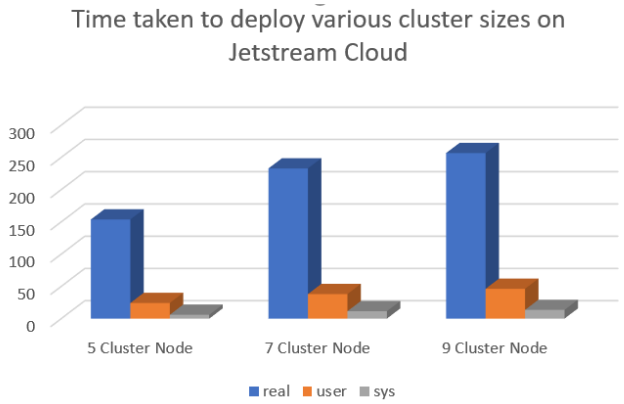

Fig. 8. Bar diagram to compare the time taken to deploy on Jetstream

Install Time	5 node cluster	7 node cluster	9 node cluster
Real:	478.426	592.267	798.894
User:	53.728	61.104	77.388
Sys:	7.048	11.224	16.116

Below is the bar description of the various times it took in deploying various size clusters.

8. BENCHMARKING ON VARIOUS CLOUDS

Comparison between installation time on various clouds with different cluster sizes. From the above deployment, following inferences were made

- Kilo cloud relatively long time when it came to installing java and its related packages

Table 6. Hardware Specifications of Kilo

	Kilo
Image	Ubuntu-14.04-64
flavor	m1.small
cores	1024
speed	2.66GHz
RAM	3072GB
storage	335TB

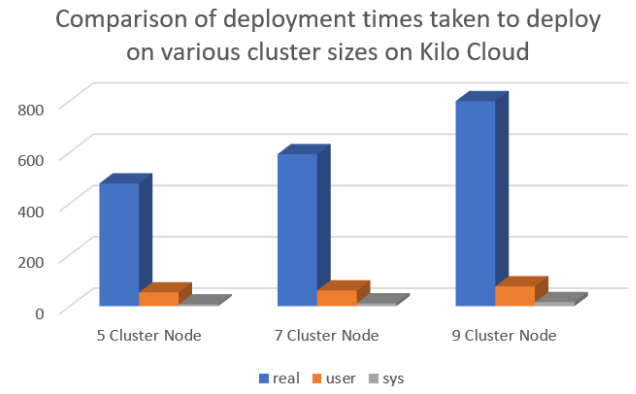

Fig. 9. Bar diagram to compare the time taken to deploy on Kilo

- Fairly, all the clusters took the same time in starting the storm cluster and running the storm-starter job.

Also a plot has been made to illustrate the various installation times on different clouds at one place, below image illustrates that.

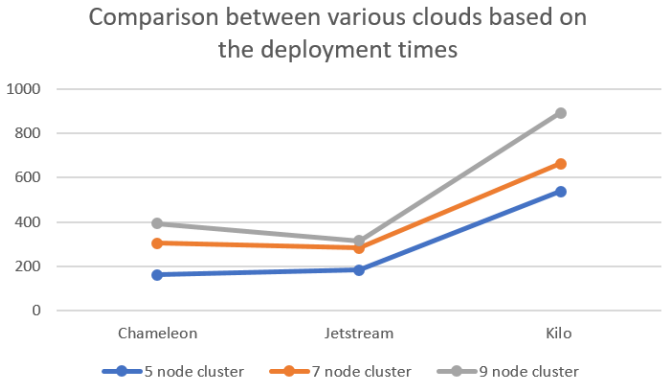

Fig. 10. Graph plot to compare the time taken to deploy on various clouds

Install Time	Chameleon	Jetstream	Kilo
5 node:	161.802	183.95	539.202
7 node:	305.494	282.603	664.595
9 node:	393.716	316.451	892.398

9. CODE REFERENCES

Following were the code resources used in deploying the cluster at various stages.

- While coding configurations related to Storm we used the following resources, we have looked at the sample use cases of the storm application at Twitter and Spotify. Code resources for Storm-[5][6][7][8][9][10][11][12][13][14]
- While configuring the Zookeeper cluster, we came across many solutions, the important question we came across is whether to use a stand-alone mode or a server mode configuration also while researching for various cluster sizes, we have come to the conclusion that storm is better deployed on cluster sizes more than 5. However, the server mode configuration is quite relevant to our project, so used many resources from below for coding the required configurations. Zookeeper-[15][16][17][18][19][20][21][22]
- The following code resources were used in analyzing the toplogies which were created part of development of the storm cluster. We went through many use-cases such as like in the case of Spotify while researching this topic. Running Topologies on a cluster-[23][24][25][26][27][28]
- While encountering various issues with Zookeeper, following code resources were used, of these we examined whether dynamic toplogies where possible to create, problems with running a 3 node cluster, and other config files configuration. Zookeeper Troubleshooting-[29][30][31][32][33][34]
- While checking the log files to check if storm is working or not, had to explore the below resources to make sure it was up and running. Storm Troubleshooting-[35][36][37][38]
- Storm yaml defaults file details were obtained from the below code resources.-[39]

10. SUMMARY

We have created, tested and demonstrated a fully automated program to configure and deploy a Storm cluster which can be deployed on any cloud environment using cloudmesh. We currently deployed it on Chameleon, Jetstream and Kilo cloud. Benchmarking strategy is visioned by viewing the time it takes to deploy a particular cluster on various cloud environments. We did a benchmark test on Chameleon, Jetstream and Kilo cloud to measure the time taken to deploy 5, 7 and 9 node clusters, so far it has shown satisfactory results and we are striving to take it beyond to other cloud environments as well, outside the cloudmesh usage.

11. ACKNOWLEDGMENTS

This project was a part of the Big Data Software and Projects (INFO-I524) course. We would like to thank Professor Gregor von Laszewski and the associate instructors for their help and support during the course. Special mention to cloudmesh client which made most of the gruelling taks straighforward.

REFERENCES

[1] N. Bronson, Z. Amsden, G. Cabrera, P. Chakka, P. Dimov, H. Ding, J. Ferris, A. Giardullo, S. Kulkarni, H. C. Li *et al.*, "Tao: Facebook's distributed data store for the social graph." pp. 49–60, 2013.

[2] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham *et al.*, "Storm@ twitter," ACM, pp. 147–156, 2014.

[3] J. Kreps, N. Narkhede, J. Rao *et al.*, "Kafka: A distributed messaging system for log processing," pp. 1–7, 2011.

[4] "Apache ZooKeeper - Home." [Online]. Available: https://zookeeper.apache.org/

[5] "Setting up a Storm Cluster." [Online]. Available: http://storm.apache.org/releases/1.0.3/Setting-up-a-Storm-cluster.html

[6] "How to Deploy Apache Storm on AWS | Cloud Academy." [Online]. Available: http://cloudacademy.com/blog/how-to-deploy-apache-storm-on-aws/

[7] "Deploy and manage Apache Storm topologies on Linux-based HDInsight | Microsoft Docs." [Online]. Available: https://docs.microsoft.com/en-us/azure/hdinsight/hdinsight-storm-deploy-monitor-topology-linux
[8] "Running a Multi-Node Storm Cluster - Michael G. Noll." [Online]. Available: http://www.michael-noll.com/tutorials/running-multi-node-storm-cluster/
[9] "How to Deploy an Apache Storm Cluster to the Amazon Elastic Compute Cloud (EC2) – Knowm.org." [Online]. Available: http://knowm.org/how-to-deploy-an-apache-storm-cluster-to-the-amazon-elastic-compute-cloud-ec2
[10] "nathanmarz/storm-deploy: One click deploy for Storm clusters on AWS." [Online]. Available: https://github.com/nathanmarz/storm-deploy
[11] "java - Apache Storm Topology deployment - Stack Overflow." [Online]. Available: http://stackoverflow.com/questions/34037377/apache-storm-topology-deployment
[12] "Apache Storm Cluster Architecture." [Online]. Available: https://www.tutorialspoint.com/apache_storm/apache_storm_cluster_architecture.htm
[13] "Apache Storm: What We Learned About Scaling." [Online]. Available: https://www.loggly.com/blog/what-we-learned-about-scaling-with-apache-storm/
[14] "How Spotify Scales Apache Storm | Labs." [Online]. Available: https://labs.spotify.com/2015/01/05/how-spotify-scales-apache-storm/
[15] "ZooKeeper Administrator's Guide." [Online]. Available: https://zookeeper.apache.org/doc/r3.3.2/zookeeperAdmin.html
[16] "ZooKeeper Cluster (Multi-Server) Setup | myjeeva blog." [Online]. Available: https://myjeeva.com/zookeeper-cluster-setup.html
[17] "Howto Setup Apache Zookeeper Cluster on Multiple Nodes in Linux." [Online]. Available: http://www.thegeekstuff.com/2016/10/zookeeper-cluster-install/
[18] "Setting up Apache ZooKeeper Cluster | Apache ZooKeeper Tutorials." [Online]. Available: http://www.allprogrammingtutorials.com/tutorials/setting-up-apache-zookeeper-cluster.php
[19] "Clustering Zookeeper." [Online]. Available: http://www.mastertheintegration.com/core-apache-projects/zookeeper/clustering-zookeeper.html
[20] "How To Setup a Zookeeper Cluster - Beginners Guide." [Online]. Available: https://devopscube.com/how-to-setup-a-zookeeper-cluster/
[21] "Cluster(Multi server) setup for zookeeper exhibitor - Stack Overflow." [Online]. Available: http://stackoverflow.com/questions/17343475/clustermulti-server-setup-for-zookeeper-exhibitor
[22] "Zookeeper Quick Guide." [Online]. Available: https://www.tutorialspoint.com/zookeeper/zookeeper_quick_guide.htm
[23] "Running Topologies on a Production Cluster." [Online]. Available: http://storm.apache.org/releases/1.0.3/Running-topologies-on-a-production-cluster.html
[24] "java - How to submit a topology in storm production cluster using IDE - Stack Overflow." [Online]. Available: http://stackoverflow.com/questions/15781176/how-to-submit-a-topology-in-storm-production-cluster-using-ide
[25] "Deploy and manage Apache Storm topologies on Linux-based HDInsight | Microsoft Docs." [Online]. Available: https://docs.microsoft.com/en-us/azure/hdinsight/hdinsight-storm-deploy-monitor-topology-linux
[26] "Setting up Storm and Running Your First Topology | Harold Nguyen's Blog." [Online]. Available: http://www.haroldnguyen.com/blog/2015/01/setting-up-storm-and-running-your-first-topology/
[27] "Running Topologies on a Storm Cluster | CoreJavaGuru." [Online]. Available: http://www.corejavaguru.com/bigdata/storm/running-topologies-on-a-cluster
[28] "Is there a way to create Dynamic Topologies in Apache Storm? - Hortonworks." [Online]. Available: https://community.hortonworks.com/questions/70650/is-there-a-way-to-create-dynamic-topologies-in-apa.html
[29] "Zookeeper - three nodes and nothing but errors - Stack Overflow." [Online]. Available: http://stackoverflow.com/questions/11498507/zookeeper-three-nodes-and-nothing-but-errors
[30] "Zookeeper can not run becuase of Invalid config - Stack Overflow." [Online]. Available: http://stackoverflow.com/questions/37738826/zookeeper-can-not-run-becuase-of-invalid-config
[31] "zookeeper-user - Starting zookeeper in replicated mode." [Online]. Available: http://zookeeper-user.578899.n2.nabble.com/Starting-zookeeper-in-replicated-mode-td5205720.html
[32] "Solved: Re: zookeeper gets permission problem on /var/lib/... - Cloudera Community." [Online]. Available: http://community.cloudera.com/t5/Cloudera-Manager-Installation/zookeeper-gets-permission-problem-on-var-lib-zookeeper/m-p/30749
[33] "java - what is zookeeper port and its usage? - Stack Overflow." [Online]. Available: http://stackoverflow.com/questions/18168541/what-is-zookeeper-port-and-its-usage
[34] "zookeeper-user - Problem with leader election." [Online]. Available: http://zookeeper-user.578899.n2.nabble.com/Problem-with-leader-election-td7579173.html
[35] "Apache Storm: Could not find leader nimbus from seed hosts - Stack Overflow." [Online]. Available: http://stackoverflow.com/questions/36742451/apache-storm-could-not-find-leader-nimbus-from-seed-hosts
[36] "Solutions for Storm Nimbus Failure - Hortonworks." [Online]. Available: https://community.hortonworks.com/articles/8844/solutions-for-storm-nimbus-failure.html
[37] "Apache Storm is not running properly and worker.log generating exceptions - Stack Overflow." [Online]. Available: http://stackoverflow.com/questions/38551229/apache-storm-is-not-running-properly-and-worker-log-generating-exceptions
[38] "java - Apache Storm Supervisor not Running the Bolt - Stack Overflow." [Online]. Available: http://stackoverflow.com/questions/28667548/apache-storm-supervisor-not-running-the-bolt
[39] "storm/defaults.yaml at master · apache/storm." [Online]. Available: https://github.com/apache/storm/blob/master/conf/defaults.yaml

A. WORK BREAKDOWN

- Ajit Balaga: Responsible for initial deployment of storm cluster on various clouds and clusters of different sizes, played an instrumental role in understanding the storm toplogy and its architecture.

- Vasanth Methkupalli: Responsible for deployment of the storm cluster on various clouds of different cluster sizes using ansible, almost automated the entire deployment of storm cluster on cloud.