Deployment and performance analysis of a Storm cluster on various cloud environments

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, September 21, 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. We intend to perform a deployment and performance benchmarking to realise the storm on various clouds. We wish to identify and examine the best clouds for storm in terms of deployment and performance.

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:

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

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.

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.

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.

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

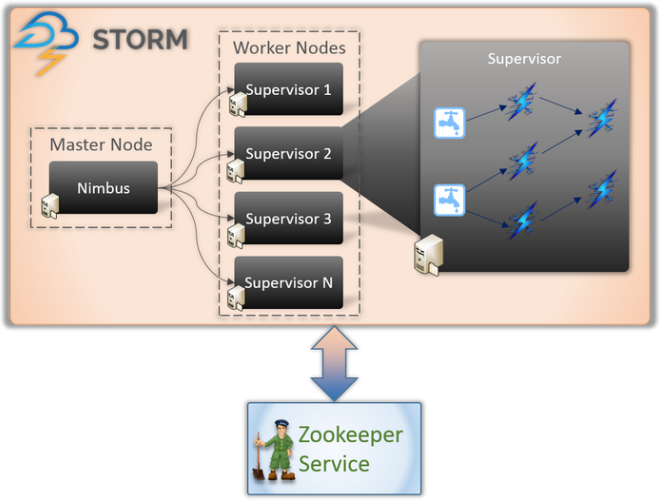

Fig. 1. Storm Architecture

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.

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]:

Shuffle grouping, which randomly partitions the tuples.

Fields grouping, which hashes on a subset of the tuple attributes or fields.

All grouping, which replicates the entire stream to all the consumer tasks.

Global grouping, which sends the entire stream to a single bolt.

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:

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.

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 .

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. Timed out worker implies that the worker did not provide a heart beat in the specified time frame. Not started worker indicates that it is yet to be started because it belongs to a newly submitted topology. 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.

3. TECHNOLOGIES

Following are the technologies used as a part of development of the entire project see Table 1.

Table 1. Technologies

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

4. CLOUDMESH

Cloudmesh client is a simple client to enable access to multiple cloud environments form a command shell and commandline. The entire application is built on python which essentially needs no prerequisite knowledge and is as a ready-to-use tool. For our project, as we need access to clouds for deployment, this comes as a welcome as it helps us with the entire deployment process. Many thanks to our professor, 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.

5. 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 a particular size, update the ansible hosts file and run the ansible script to deploy the entire project on the cloud. The entire process of deploying the project on various clouds have been tried and tested, and the results have been collected for benchmarking. The results of benchmarking have been presented after a brief comparision and deployment strategy.

6. CLOUD FLAVOR AND IMAGE COMPARISON

The following table shows a comparison of key computing resources on Chameleon, FutureSystems, and Jetstream cloud environments. See Table 2 for the comparison.

Table 2. Cloud Hardware Specification Comparison

Cloud	Kilo	Chameleon	Jetstream
Image	Ubuntu-14.04	Ubuntu-14.04	ubuntu-14.04
Flavor	m1.small	m1.small	m1.small
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

7. DEPLOYMENT AUTOMATION

7.1. Automation with Ansible

Ansible Playbook is the primary tool used for deployment. Ansible will help push configurations to the environment automatically based on playbooks written. For this project, we have used Ansible Roles to automate the deployment of storm and zookeeper clusters. The Ansible script installs the zookeeper and storm on all nodes and starts the cluster dynamically using handlers. These tasks are handled by the respective roles. CMD5 provided by the cloudmesh client is used to create a secgroup, a cluster and deploying the software automatically.

The following commands are provided with the CMD5 storm extension:

cloud: Set the cloud on which to deploy the cluster

name: Set the name of the cluster

size: Set the size of the cluster

flavor: Set the flavor to use

image: Set the image to use

cluster info: Displays the information to be used to create the cluster

cluster deploy: Does a multitude of tasks, creates a secgroup and uploads it, leverages cloudmesh client to create vms on the cloud and create a hosts file, runs the ansible script to install software and manage handlers to ensure that processes have been started.

cluster status: Checks the ports to ensure that zookeeper and storm nimbus have been started and are running.

cluster delete: deletes the cluster vms and deletes the secgroup from the cloud

submit: Submits the jar file to nimbus. The jar file has to be included in the same folder as the script to run it.

The important command in the above is the "storm cluster deploy" command. It utilizes four roles created using ansible, namely, setup, zookeeper, nimbus and supervisors. The roles are explained as follows:

Setup: Installs software packages, java jdk, java jre, supervisord on all nodes in the cluster. Creates the folders for zookeeper and storm for installation, and supervisord logs. Updates the hosts file so the nodes can communicate with each other.

Zookeeper: Downloads and unpackages zookeeper files into the folder created above. Creates a "myid" file updates the config file on each node. Creates a supervisord config file to run the process in the background. Finally, starts the zookeeper cluster.

Nimbus: Installs maven on nimbus nodes. Downloads and unpackages storm files into the folder created above. Updates the storm config files and creates supervisord config file to run nimbus and ui in the background. Starts the storm nimbus and ui on the master node.

Supervisors: Downloads and unpackages storm files into the folder created above. Updates the storm config files and creates supervisord config file to run storm workers (supervisors) in the background. Starts the storm workers on the slave nodes.

8. DEPLOYMENT BENCHMARKING

Following zookeeper recommendations, deployment testing was done starting with a cluster size of 5. Zookeeper recommends a minimum cluster size of 5 and scale the clusters in odd numbers. This way, the server cluster can ensure reliability.

8.1. Chameleon Cloud

Chameleon cloud was used for development and most of the scripts were run and tested on this cloud. The ansible scripts developed on this cloud were then tested on the remaining clouds to ensure they are platform independent. The results of deployment are presented in the table 3 and illustrated in the form of a bar plot (Fig 2).

Table 3. Table illustrating the various times it took to deploy on Chameleon cloud

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

8.2. JetStream Cloud

Automated deployment on Jestream cloud of various cluster node sizes of 5, 7 and 9 and results are presented as follows, in the form of a table 4 and illustrated as a figure 3.

A bar description of the time taken for deployment on different size clusters see Fig 3.

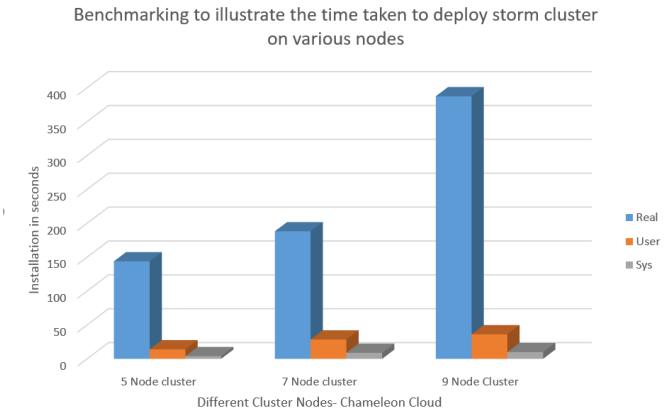

Fig. 2. Bar diagram to compare the time taken to deploy on Chameleon

Table 4. Illustrating the various times it took to deploy on Jetstream

Install Time	5 node cluster	7 node cluster	9 node cluster
Real:	153.73	232.923	256.675
User:	24.232	38.14	46.208
Sys:	5.988	11.54	13.568

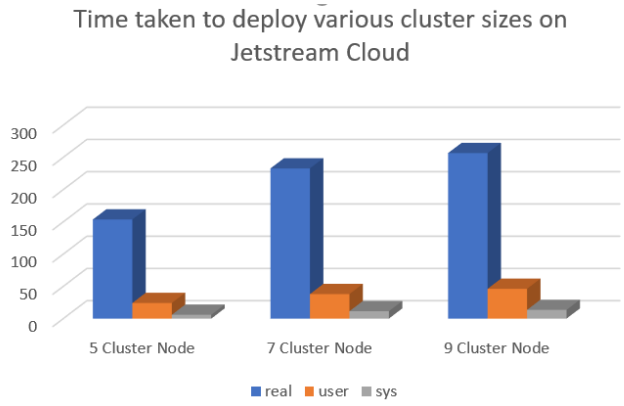

Fig. 3. Bar diagram to compare the time taken to deploy on Jetstream

8.3. Kilo Cloud - FutureSystems

Automated deployment on Kilo cloud of various cluster node sizes of 5, 7 and 9 and results are presented, in the form of a table 5 and illustrated as a figure 4

A bar description of the time taken for deployment on different size clusters see Fig 4.

Table 5. Illustrating the various times it took to deploy on Kilo

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

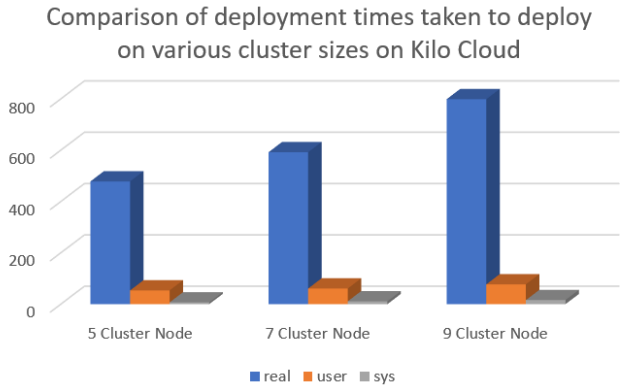

Fig. 4. Bar diagram to compare the time taken to deploy on Kilo

9. DEPLOYMENT SUMMARY ON VARIOUS CLOUDS

Comparison between installation time on the three clouds with differing cluster sizes is done in the figure 5. The following inferences were made:

- Kilo cloud - Not conducive to development or production as the creation of vms on the cloud and the installation of software took a relatively long time.
- Chameleon and Jetstream clouds have similar rates of deployment. As observed, jetstream is more suitable for production.
- All the clusters took similar amount of time for starting the zookeeper and storm processes.

The table showcases the times plotted. see Table 6

Table 6. Time to deploy various cluster sizes 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

A plot has been illustrated to distinguish the installation time on the different clouds see Fig 5.

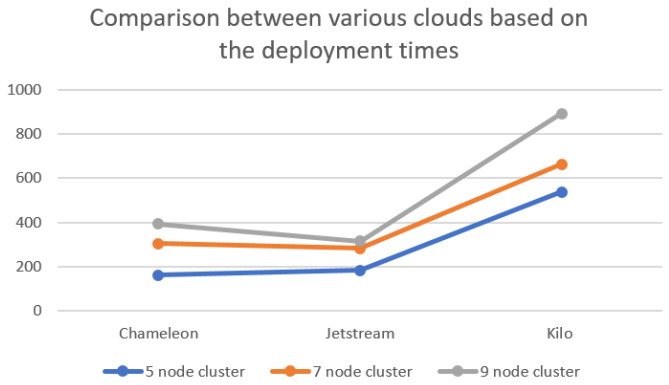

Fig. 5. Graph plot to compare the time taken to deploy on various clouds

10. ANALYSIS OF PERFORMANCE BENCHMARKING ON VARIOUS CLOUDS:

As a part of this Project we would like to examine the difference in terms of performance between clouds and analyze the results to discern the advantages of one cloud over the other. To test the performance, a topology from the storm-starter project was used. The topology is based on the basic word count topology with a few changes. The first difference is the rate of production of sentences. The rate may be set using arguments. This enable us to perform stress testing on clusters of different sizes and clouds. The second difference is the number of worker nodes to be used. This property allows us to accurately define the number of storm supervisor nodes to run the topology on and gather the results effectively. The final difference is the time for which the topology runs. This is an important feature as the metrics are printed with a period of 30 secs. This implies that every 30s, the number of sentences that have been executed successfully is printed and a new set of sentences are generated for the next 30 secs. We have chosen a value of 5 minutes as this gives us an accurate idea of the effectiveness of the cluster. The definitions of throughput and latency are mentioned as follows: ***Latency*** is the time required to perform some action or produce a result. ***Throughput*** is the number of such actions performed per unit of time.

A result is produced/completed on a storm cluster when an "ack" is received. The number of such acks defines the throughput of the cluster. For example, if we have received 12,600 acks during a 30s time period, then the throughput rate is 420/sec. Higher throughput rates are desired for better computing capability with lower latency.

Analysis for the entire cloud based on the mentioned parameters and tests were performed, the strategy has been done by writing a shell script, performance.sh which runs for different input sizes for a given cluster and collects the output in a file. This test has been performed on all clusters in a given cloud, repeated on different clouds. All the results are tabulated in the next sections, a detailed analysis based on the results obtained is also elucidated clearly.

10.1. Hardware:

All the nodes on each of the clouds deployed had the following features in common:

Image: Ubuntu-14.04-64

Flavor: m1.small

Nodes: Depending on the cluster size various node sizes are deployed

10.2. Experimental Results and Analysis:

Following were the inferences and conclusions derived from the performance benchmarking, with respect to the ThroughputVs-Latency topology.

- 5 node cluster: The throughput graph starts small and rises equally upto a rate of 150000, at which point, the throughput on chameleon cloud falls rapidly. The results on kilo cloud indicate that a 5 node cluster is sub-optimal for this cloud. Jetstream cloud scaled very well with increase in throughput. The results are tabulated here see Table 7 and illustrated here see Fig 6.
- 7 node cluster: Chameleon cloud performed better on a 7 node cluster but the throughput falls rapidly after 300000. Kilo cloud performed sub-optimally at all throughput levels. Jetstream cloud scaled exceptionally well with an increase in throughput. The results are tabulated here see Table 8 and illustrated here see Fig 7.
- 9 node cluster: The difference in performance between chameleon cloud and jetstream cloud is negligible. Kilo cloud performs well upto a throughout value of 150000 and falls afterwards. Jetstream cloud is consistent at all throughput levels and chameleon cloud performed equally well. The results are tabulated here see Table 9 and illustrated here see Fig 8
- While significant improvements were observed on clusters of larger sizes, the results are not consistent at all times for chameleon cloud. Jetstream cloud provided consistent results and can be used for production clusters as shown by its performance results. Kilo cloud is not suitable for deployment or for production. The results are consolidated in the form of a graph see Fig 9.

Table 7. Throughput vs Latency on all 5 node clusters on 3 clouds:

	5 node cluster		
Throughput:	Chameleon	Jetstream	Kilo
3000:	2940	2760	2906
15000:	13958	13654	15093
30000:	27946	28178	30095
150000:	140454	142084	137352
300000:	35417.5	275074	10506
450000:	14946.667	414982	6268

Table 8. Throughput vs Latency on all 7 node clusters on 3 clouds:

	7 node cluster		
Throughput:	Chameleon	Jetstream	Kilo
3000:	2702	2734	2875
15000:	13502	13766	14348
30000:	27364	28110	29160
150000:	140124	141894	41275
300000:	280170	284356	431.25
450000:	67857.5	412488	435

Table 9. Throughput vs Latency on all 9 node clusters on 3 clouds:

	9 node cluster		
Throughput:	Chameleon	Jetstream	Kilo
3000:	2700	2754	2848
15000:	14722.22	13784	14337
30000:	27724	27282	29455
150000:	151562.22	141580	149440
300000:	305566.667	286708	8832
450000:	418002	427174	7312

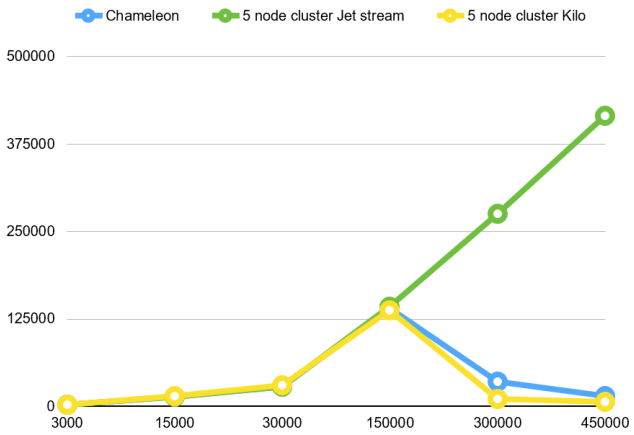

Fig. 6. Graph plot to compare the peformance on 5 nodes across various clouds

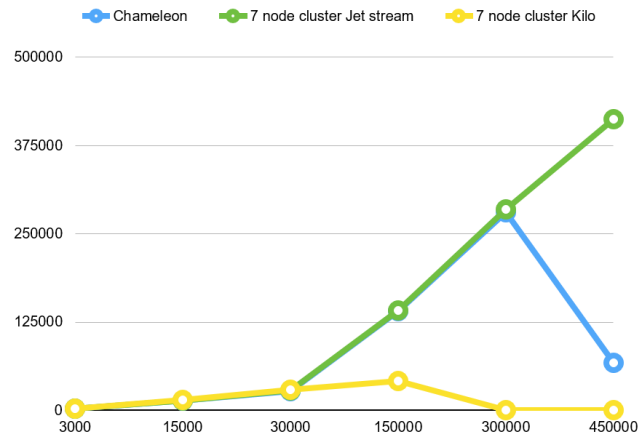

Fig. 7. Graph plot to compare the performance on 7 nodes across various clouds

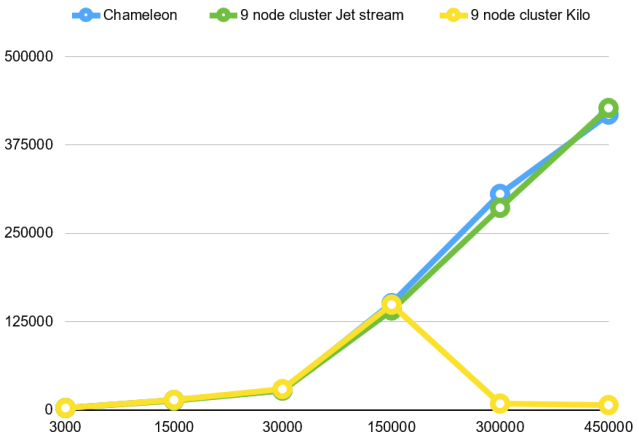

Fig. 8. Graph plot to compare the performance on 9 nodes across various clouds

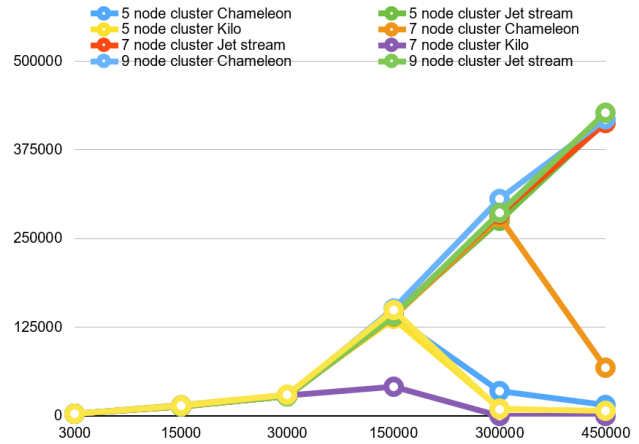

Fig. 9. Graph plot to compare the performance on all the nodes across various clouds

11. SUMMARY

We have created, tested and demonstrated a fully automated method to configure and deploy a Storm cluster which can be deployed on any cloud environment using cloudmesh. We have deployed it on Chameleon, Jetstream and Kilo cloud. Benchmarking strategy is envisioned 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. A performance benchmarking has also been done to better analyze the various cloud performances on deployment of a storm cluster. We have measured the Throughput and Latency for different lengths of input to check the performance on various clouds. All the results have been illustrated and various conclusions drawn based on the data obtained from the complete deployment of the storm cluster.

12. ACKNOWLEDGMENTS

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

13. 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 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 many resources to make sure it was up and running. Storm Troubleshooting-[35][36][37][38]

- Storm yaml defaults file details were obtained from the many code resources.-[39]

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/8168541/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 deployment of storm cluster on various clouds and clusters of different sizes, played an instrumental role in understanding the storm and its architecture. Developed the cmd5 storm commands extension and ansible scripts for deployment, responsible for perfomance benchmarking on chameleon cloud, jetstream and 9 node cluster on kilo cloud.
- 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. Performed deployment benchmarking on various clouds and analysis. Also, responsible for performance benchmarking on Kilo cloud and analysis of the entire Performance bencmarking.

LIST OF FIGURES

LIST OF TABLES