Cloud File Systems

Experimentation

By
Group E
Cloud Computing
UT DALLAS

RAJA JAYA CHANDRA MANNEM
mrjayachandra@gmail.com 972.730.4304

Organization of presentation

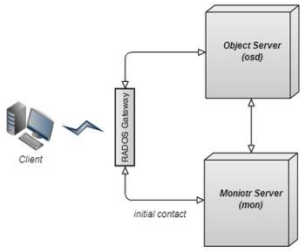
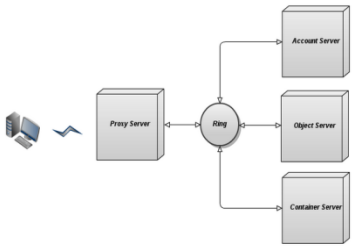
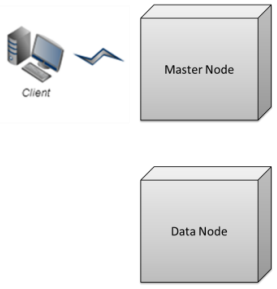
- Cloud file systems design space/ challenges
- Comparison of the file systems under study
  - Hadoop/HDFS
  - HBASE(scope limited)
  - OpenStack Swift
  - Ceph
  - Lustre
- Feature Analysis / testing
  - Hadoop HDFS
  - Swift
  - Ceph (installation)
  - Benchmarking tools: Iozone/Netmist
  - Project execution plan

Cloud file systems design space/challenges

- Major design perspectives:
  - Availability
  - Reliability
  - Fault tolerance
  - Replication
  - Scalability
  - Latency
  - I/O throughput
  - CPU Utilization
  - Metadata management
  - Consistency
  - Virtualization
  - And many more…

  - Adapt appropriate file systems based on cloud user/system requirements; appreciate standard designs

Comparison of File Systems

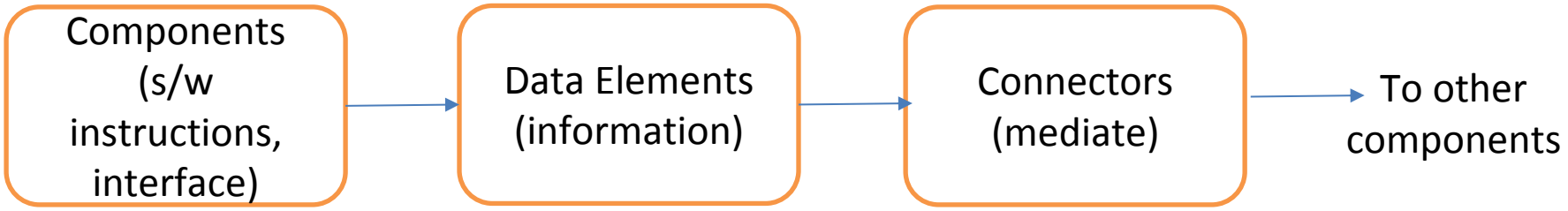
	Ceph	**Swift**	**HDFS**
Consistency	Strongly consistent	Eventually consistent	Eventually consistent
Storage	•Block storage, Object storage	Object storage	Object storage
Data placement method	CRUSH (algorithm)	Ring (static mapping structure)	Namenode specifies the location of the data
Latency	No centralized metadata server reducing access latency	centralized metadata server, reason for latency	centralized metadata server, reason for latency
Components	object server, monitor server, RADOS	Account, Container, Object	Namenodes, Datanodes
Replica management	yes(dynamic)	No	No
Architecture			

Unable to Install Lustre

- Ubuntu is not a supported OS for Luster Server
- The OS that support Lustre are
- EL 5.4, i686 and x86_64 only (Lustre 1.8.2)
  OEL 5.3, i686 and x86_64 only (Lustre 1.8.1.1 and 1.8.2)
  Red Hat Enterprise Linux 5
  SuSE Linux Enterprise Server 10
  SuSE Linux Enterprise Server 11, i686 and x86_64 only (Lustre 1.8.1 and later)
  Linux kernel 2.6.16 or greater
- In order to install Lustre we need to CentOS ,which doesn't support other file systems .

Rest API

- What is REST architecture?(from testing perspective)



- Mainly used with HTTP/HTTPS

Why REST in our analysis?

- Rest properties best suited for distributed file systems.
  - –Performance
  - –Scalability
  - –Simplicity of interfaces
  - –Modifiability of components to meet changing needs(while application is running)
  - –Portability of component deployment
  - –Reliability

Feature Analysis/testing

HDFS : Plain requests

- Read: hadoop fs -cat /tmp/file123
- Put: hadoop fs -put /home/hduser/hdfs_script/file123 /tmp
- ListDir: hadoop fs -ls /tmp
- Delete: hadoop fs -rm /tmp/2mbfile.csv

Feature Analysis/testing

- HDFS REST requests : implemented through WebHDFS API.

- Rest Read: curl -i -L http://localhost:50070/webhdfs/v1/tmp/file123?op=OPEN
- Rest Put: curl -i -X PUT http://localhost:50070/webhdfs/v1/tmp/2mbfile.csv?user.name=hduser&op=CREATE
  curl -i -X PUT -T /home/hduser/hdfs_script/2mbfile.csv http://slave1:50075/webhdfs/v1/tmp/2mbfile.csv?op=CREATE&user.name=hduser&overwrite=false
- Rest ListDir: curl -i http://localhost:50070/webhdfs/v1/tmp?op=LISTSTATUS
- Rest Delete: curl -i -X DELETE http://localhost:50070/webhdfs/v1/tmp/2mbfile.csv?user.name=hduser&op=DELETE

Feature Analysis/testing

- Swift : Plain requests
- Admin account setup issues for testing requests.
- Testing in progress

Feature Analysis/testing

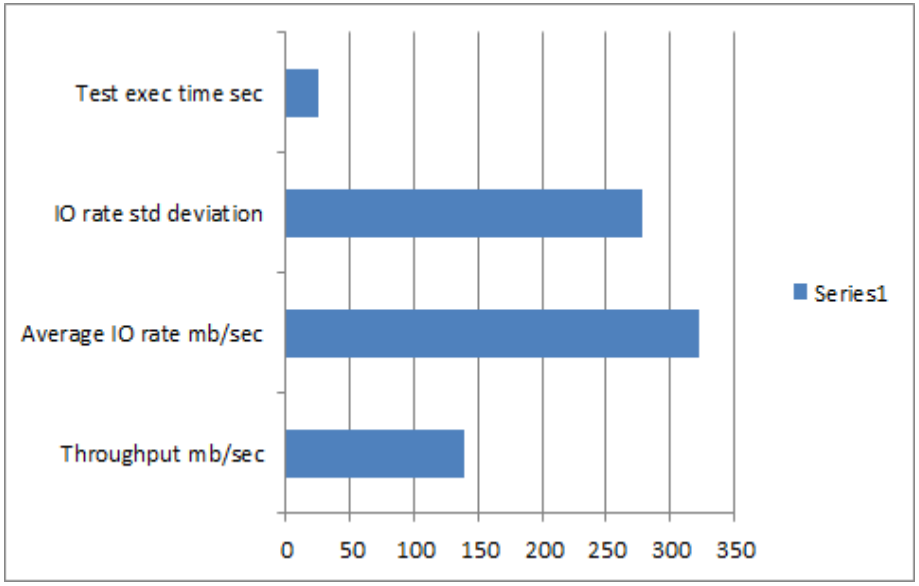
- Swift RESTful requests : implemented through SwAUTH API
- Testing in progress

Feature Analysis/testing

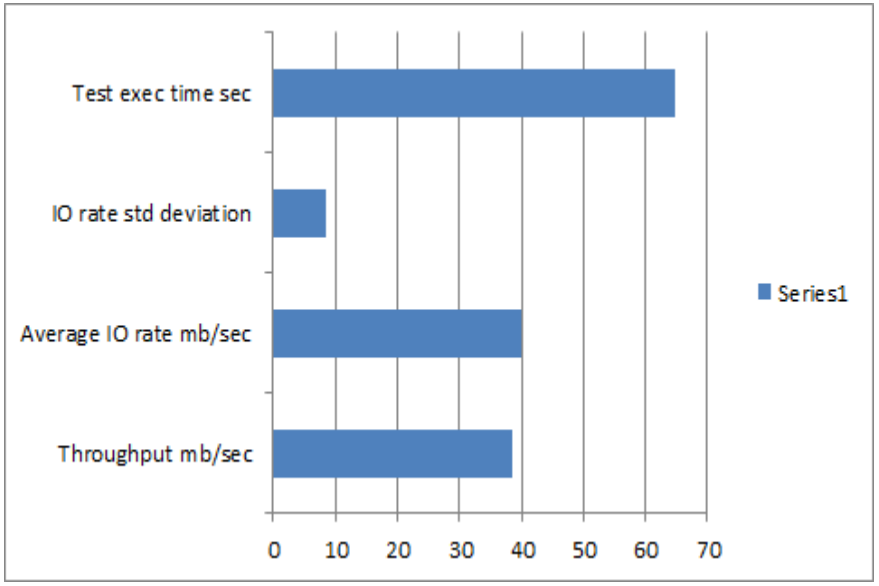
- Ceph
- Installation success
- Testing and RESTful interface in progress

Testing:HDFS

TESTDFSIO _READ OPERATION RESULTS

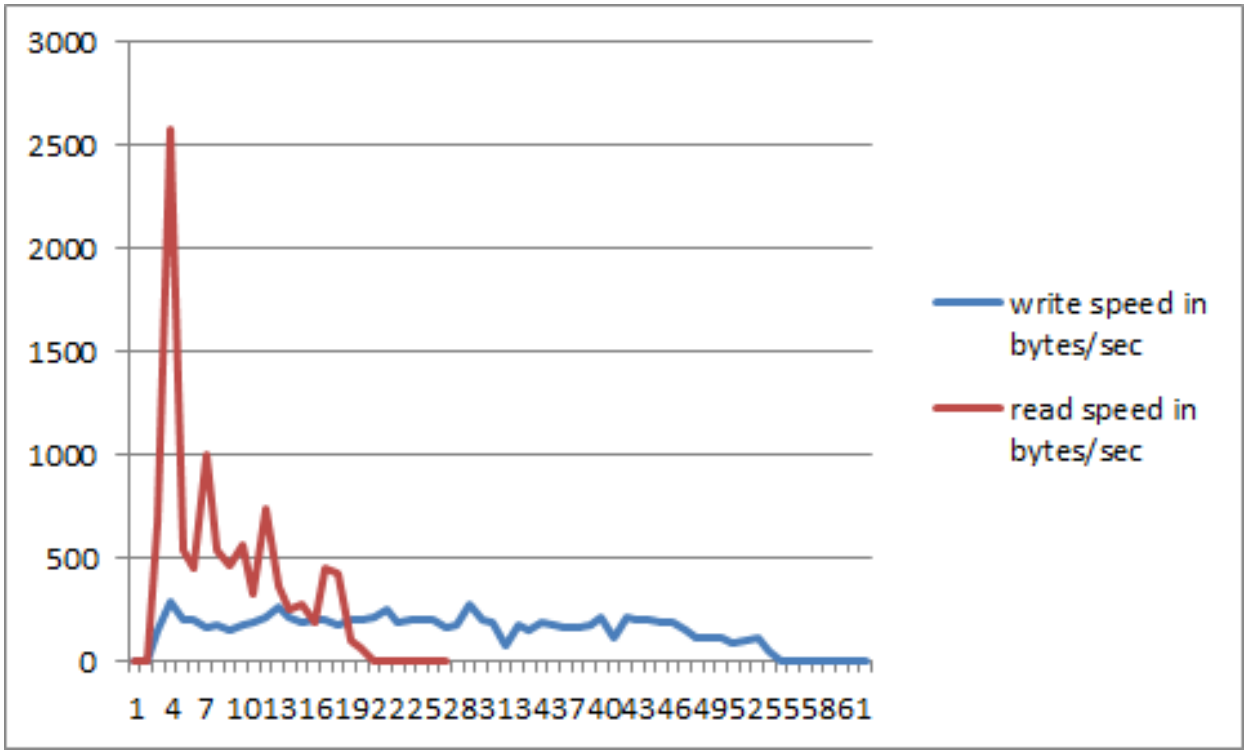


TESTDFSIO_WRITE OPERATION RESULTS

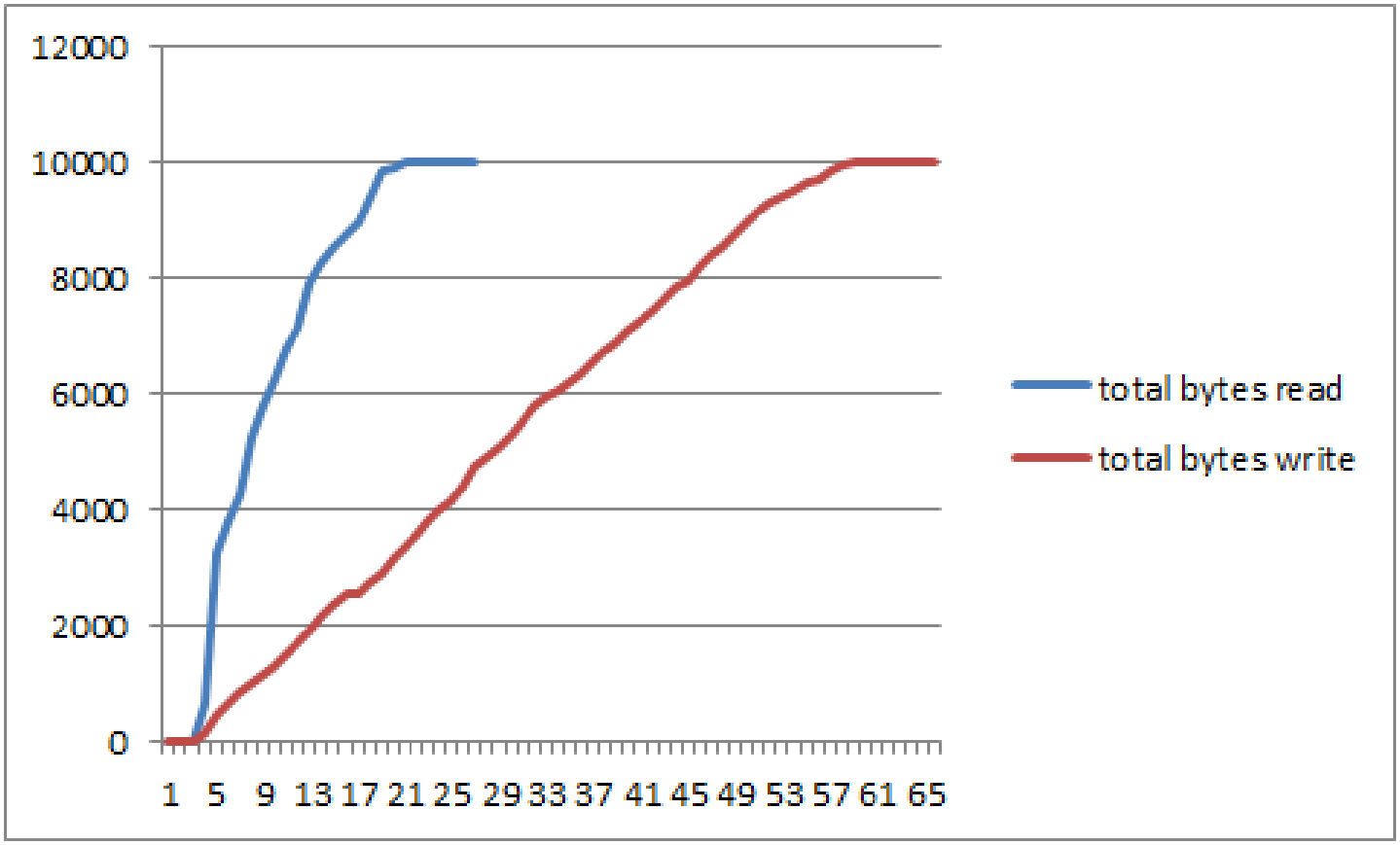


Testing:HDFS

Throughput: Read Vs Write

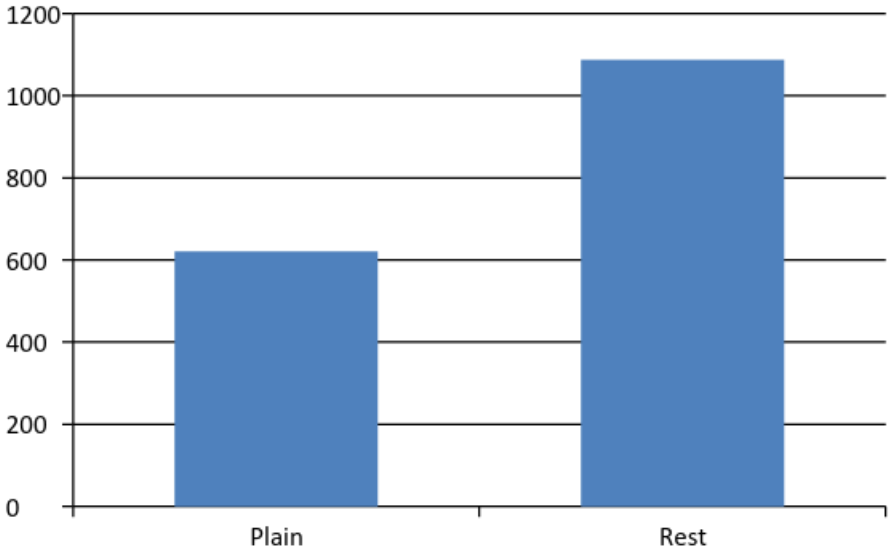


Testing : HDFS

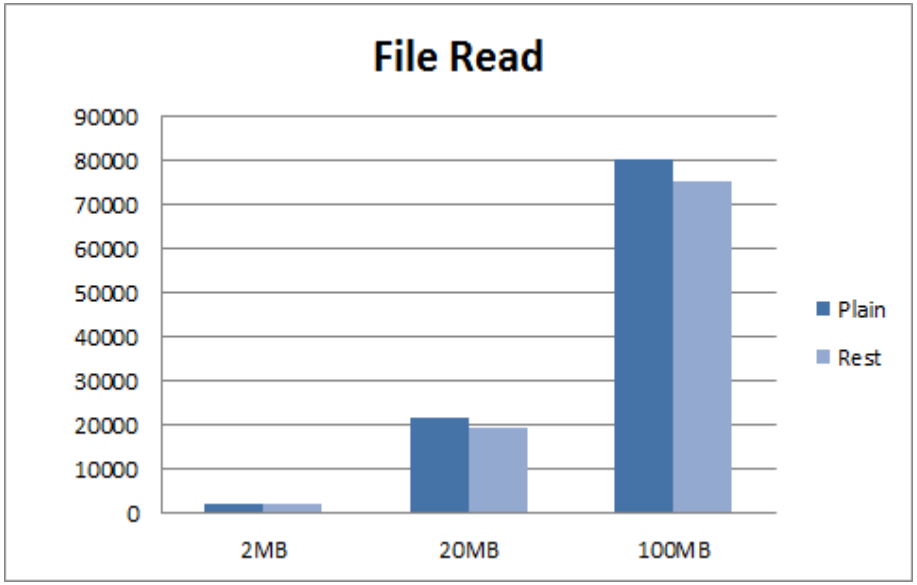


Plain Versus REST API: HDFS

ListDir

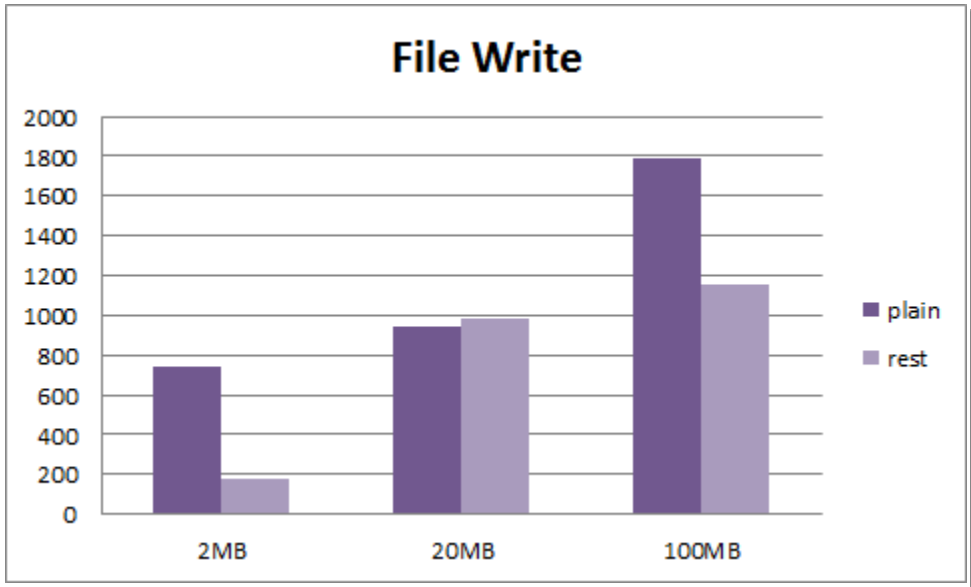


cat

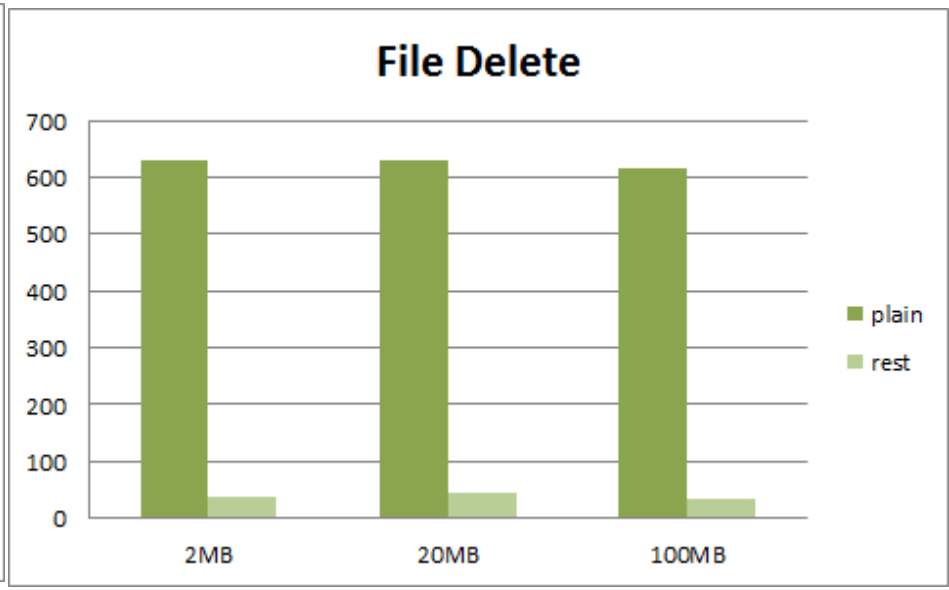


Plain Versus REST API: HDFS

put

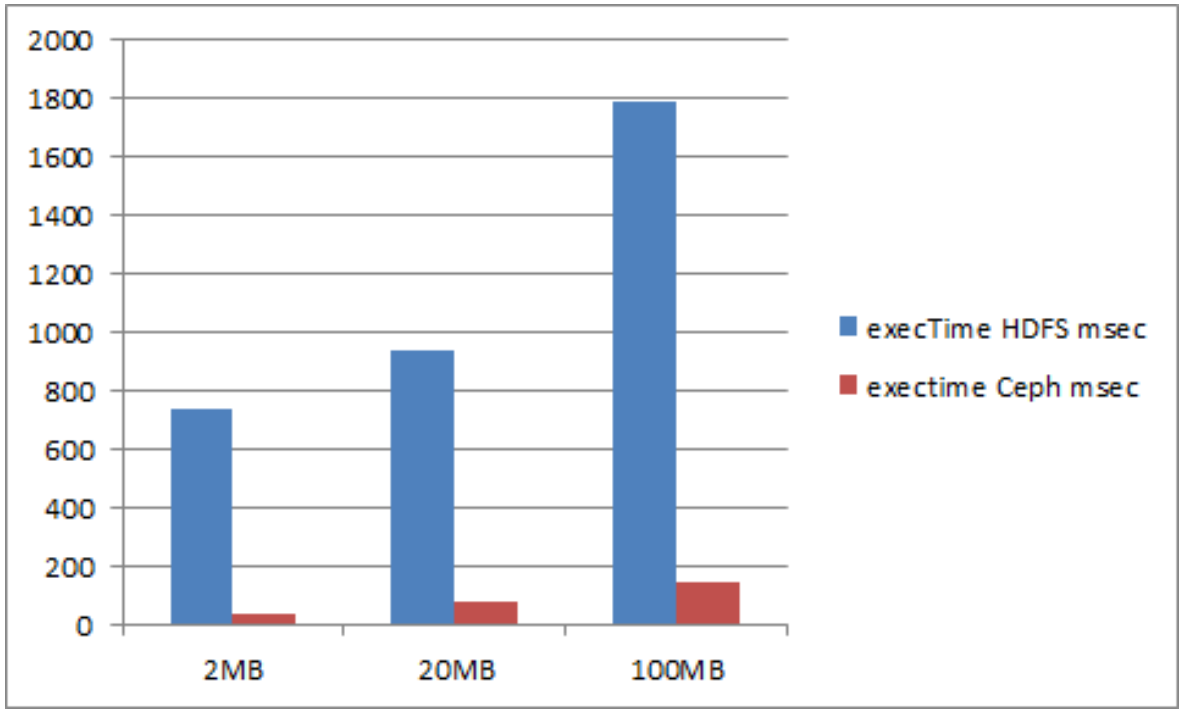


delete



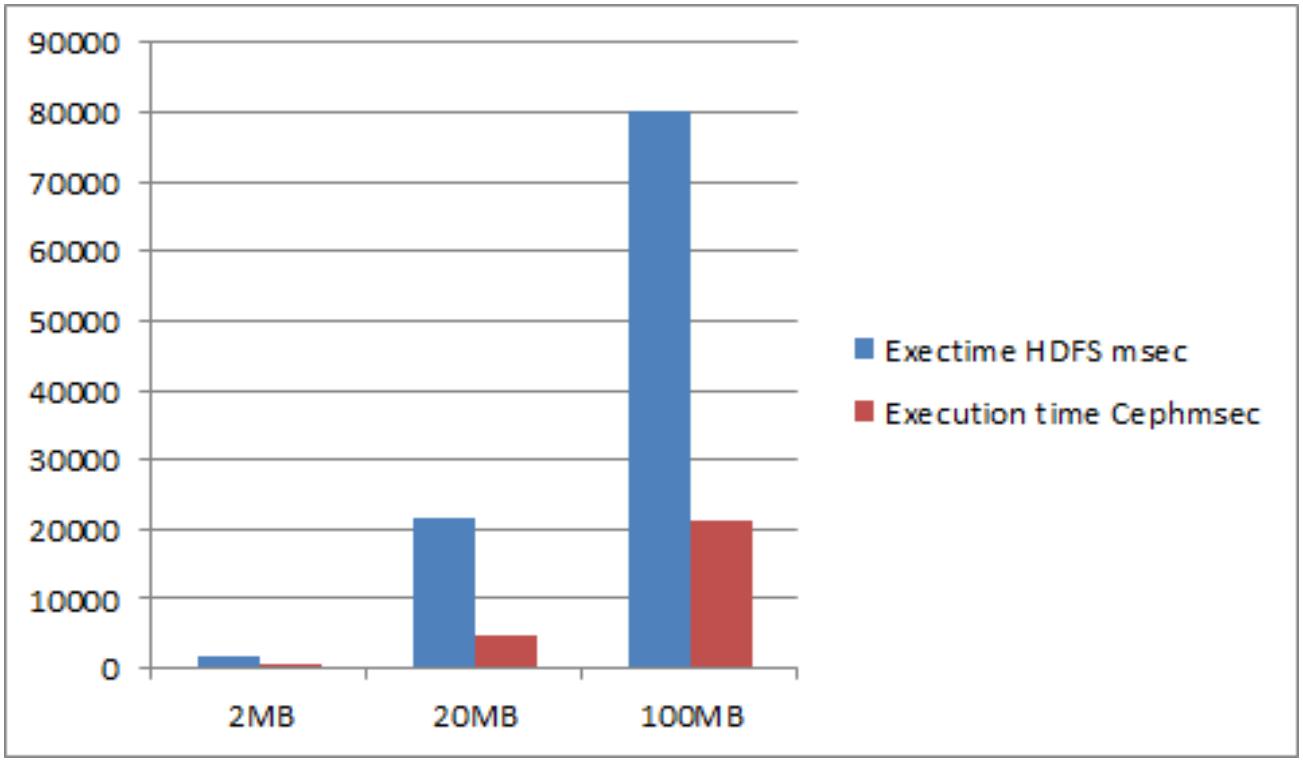
Plain HDFS Vs Plain CEPH

Write



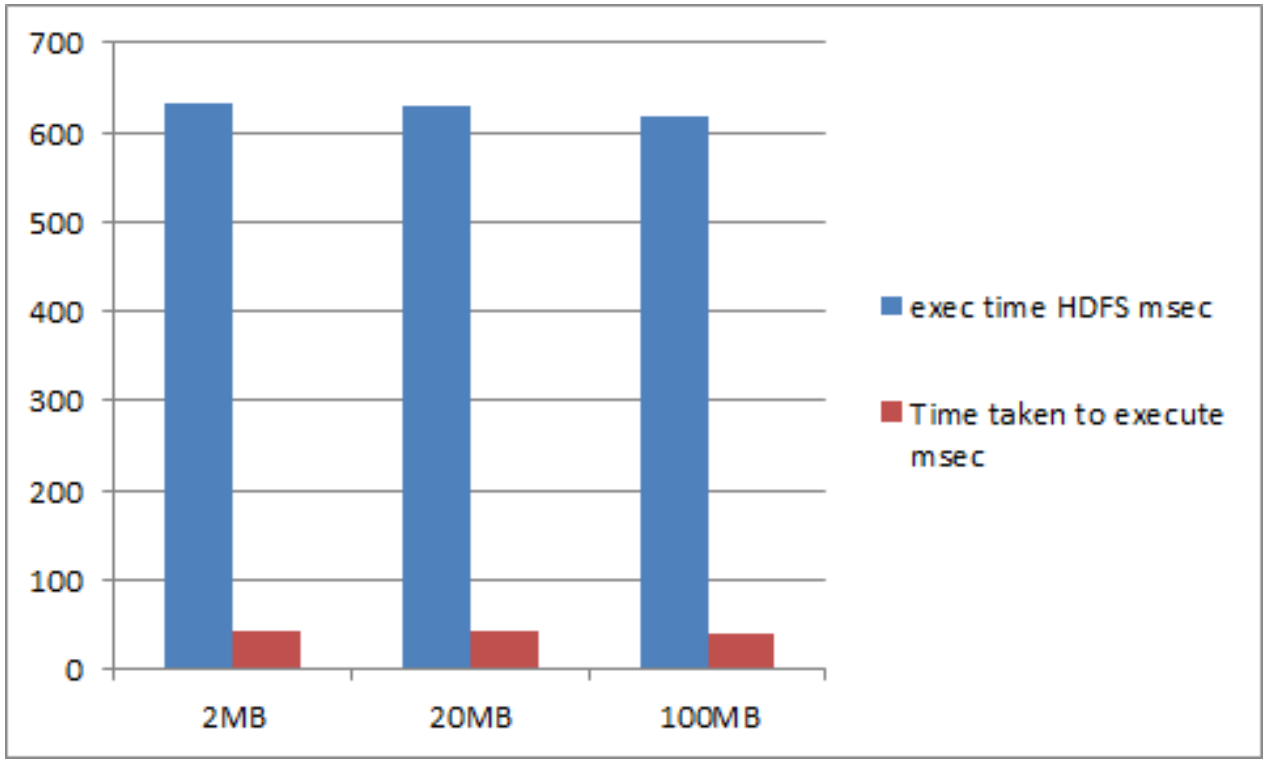
Plain HDFS Vs Plain CEPH

Read



Plain HDFS Vs Plain CEPH

Delete



Fault Tolerance and Load Balancing

Initially, total = 30.84 GB

DataNode	Slave1	Slave2	Slave3
Usage	13.04 GB	9.12 GB	8.68 GB

After uploading 10 GB of data, 20 GB Data added (replication factor = 2)

DataNode	Slave1	Slave2	Slave3
Usage	18.65 GB	15.4 GB	16.02 GB

After killing Slave3

DataNode	Slave1	Slave2	Slave3
Usage	25.03 GB	25.03 GB	0

After restarting Slave3

DataNode	Slave1	Slave2	Slave3
Usage	9.01 GB	25.03 GB	16.02 GB

Next Steps

- RESTful requests for Ceph and Swift
- Plain requests for Swift
- Scalability for HDFS, Swift and Ceph
- Fault tolerance and load balancing for Ceph and Swift
- Netmist integration

Questions?

Thank you !!!