Automation of Rolling Upgrade for Hadoop Cluster without Data Loss and Job Failures

May 17, 2017
Hiroshi Yamaguchi & Hiroyuki Adachi
About Us

Hiroshi Yamaguchi
• Hadoop DevOps Engineer

Hiroyuki Adachi
• Hadoop Engineer
Largest Portal Site in Japan

- >90 million DUB*
- >69.8 Billion PV/month
- >34 million MAU
- 74% Smartphone Users
- 82% PC Users
- 2.7 Billion Shopping Items
- >100 Services
- No.1 App. publisher

* Daily Unique Browsers

Copyright © 2017 Yahoo Japan Corporation. All Rights Reserved.
Overview of Our Data Platform

- PV
- Conversion
- Recommendation
- Ad Targeting

Data Platform

- kafka
- Access Logs
- Web Server

MySQL
TERADATA
Cassandra

Yahoo Japan
1. Our Hadoop Cluster
2. Issues Involved in Previous Upgrade
3. Upgrade Approach
4. How to Upgrade
5. Results
6. Conclusion
Our Hadoop Cluster
Overview

- HDP 2.3 + Ambari 2.2
- Almost all core components are configured with HA
  - HA: NameNode, ResourceManager, JournalNode, Zookeeper
  - Non HA: Application Timeline Server
- Secured with Kerberos
- 800 slave nodes
  - DataNode/NodeManager
- Other components
  - HiveServer2, Oozie, HttpFS

800 Nodes

128GB Mem
Xeon E5-2683v3
Data Volume

- Total stored data: 37 PB
- Increases about 50 ~ 60 TB/day

Increase in data volume
Work Load

- Multi-tenant cluster
  - ETL / E-Commerce / Advertising / Search ...
  - 20K ~ 30K jobs/day
- Average resource utilization is over 80%
- Data processing needs are growing

YARN work load of the day
Issues Involved in Previous Upgrade
Previous Upgrade

- Performed in Q4 2015
- Without using Ambari
- Manual Express Upgrade including above mentioned steps
Previous Upgrade

- **Upgrade**
  - Restarting all components for updates to come into effect

- **Check**
  - Component’s normality checks
  - Wait till missing blocks to be fixed

It took 12 hours
Issue 1: 12 Hours of Cluster Down

• Why took so long?
  • Large number of slave nodes
  • Multiple nodes failed to start after upgrade
  • Job failed due to data loss
  • Must recover missing blocks
Issue 2: Finding Right Schedule

- Challenging to find a right schedule
  - Cluster is multi-tenant, shared by hundreds of services
  - Picking a day with minimal impact
    e.g. A day without weekly or monthly running jobs
Issue 3 : Coordinating Resource Allocation

- Post-Upgrade: Restarting all jobs simultaneously caused resource exhaustion
- Jobs were recovered by precedence

Diagram:
- Upgrade Operation
- Cluster down
- Resource Allocation
- ETL Jobs
- Scheduled Jobs
- Ad hoc Jobs

Copyright © 2017 Yahoo Japan Corporation. All Rights Reserved.
Issues to be Addressed

• Storage
  • Should not affect HDFS Read/Write
  • Should prevent missing blocks

• Processing
  • To keep Components (Hive, Oozie ...) working
  
Impact-less upgrade operation
Upgrade Approach
Possible Upgrading Methods

1. Ambari Provided Express Upgrade
2. Ambari Provided Rolling Upgrade
3. Manual Express Upgrade
4. Manual Rolling Upgrade
Possible Upgrading Methods

1. Ambari Provided Express Upgrade
2. Ambari Provided Rolling Upgrade
3. Manual Express Upgrade
4. Manual Rolling Upgrade

1’st and 2’nd are not suitable for our environment
3’rd has several issues as explained earlier
Approaches for Upgrading

- Impact-less Rolling Upgrade
  - Grouping components e.g. NameNode
  - Upgrading & Checking each component group
• Total upgrade time increases as slave nodes are upgraded one-by-one for safety
Approaches for Upgrading

- Automating upgrade and check process to reduce operation cost
Summary

• Impact-less rolling upgrade is possible
• Automating upgrade process reduces operation cost
How to Upgrade
Target Cluster

- HDP 2.3.x + Ambari 2.2.0
- Master nodes
  - HA: NameNode, ResourceManager, JournalNode, Zookeeper
  - Non-HA: Application Timeline Server
- Slave nodes
  - 800 DataNode/NodeManager
- Others
  - HiveServer2, Oozie, HttpFS
Apache Ambari

Web UI
REST API
Ambari Provided HDP Upgrade Methods

- Express Upgrade
  - Brings down entire cluster
- Rolling Upgrade
  - Cannot control
    - Load balanced HiveServer2 restart
    - Collective DataNode restart

But we can’t adopt either of these methods
Extending Ambari’s Operations

• Safety restart and our environment specific operations
  • Ambari Custom Service
    • Additional operations such as NN failover
  • API wrapper scripts
    • Additional confirmation of service normality
    • Precise control
For Non-stop Upgrade

Controlling with Custom Ambari Services and Scripts

Controlling with Ansible
Upgrade Flow Control

Controlling with Custom Ambari Services and Scripts

Controlling with Ansible
Ansible

• Configuration management tool
• Why we chose?
  • Easy to learn
  • Agent less, push based system
  • Can control upgrading workflow
Overview of Upgrading with Ansible

- Configuration management
- Controlling upgrade sequence

ansible-playbook -i hosts/target_cluster play_books/upgrade_master.yml
Make backups of critical metadata: FsImage, Hive Metastore, Oozie DB.

Registers repository for the target (HDP) version.

Installs target (HDP) version on all hosts.

Upgrades to target version, Some component upgrades in parallel.

Finalizes the upgrades.

https://docs.hortonworks.com/HDPDocuments/Ambari-2.1.1.0/bk_upgrading_Ambari/content/_manual_minor_upgrade.html
Upgrading Each Component

Master
- Zookeeper
- NN, JN, JHS, ATS, RM

3 hours

Application
- HttpFS, Oozie
- Hive Metastore, HiveServer2

4 hours

Client
- MapReduce2, Spark, YARN, HDFS, Oozie, Hive, Zookeeper

0.5 hours

Slave
- DataNode
- NodeManager

70 hours
Preventing Job Failures

Controlling with Custom Ambari Services and Scripts

Controlling with Ansible
Ambari Custom Service

- Ambari can implement custom service
- Operational commands
  - NameNode F/O
  - Load balancer pool add/remove
- Can operate as existing Ambari services
Ambari Custom Service

- Add service definition xml and python scripts
- No need of manually executing commands on a server
- Prevents operation miss
Ambari CLI

- In-house script for cluster admins
- A wrapper script using API’s of Ambari, NameNode, ResourceManager etc.
- Provides safe operations
  - Pre and post restart check for each component
  - Preventing data loss
Safety Restart for HiveServer2

Client -> Load Balancer -> HiveServer2
Safety Restart for HiveServer2

Client

_loadBalancer_

Established Connection

Wait for jobs to be finished

⚠️

HiveServer2
Safety Restart for HiveServer2

Client

Load Balancer

Established Connection

HiveServer2

⚠ Wait for jobs to be finished
Safety Restart for HiveServer2

Client

Load Balancer

HiveServer2

UPGRADE
Safety Restart for HiveServer2

Client

Load Balancer

HiveServer2
Preventing Data Loss

Controlling with Custom Ambari Service and Scripts

Controlling with Ansible
Safety Restart for DataNode

Rack 1  Rack 2  Rack 3  Rack 4  Rack n

Missing Blocks: 0
Under Replicated Blocks: 0
Corrupt Blocks: 0

Group of DataNodes at a time

Upgrade, restart and wait for Missing Blocks to be 0
Safety Restart for DataNode

Upgrade, restart and wait for Missing Blocks to be 0
Replication Vs Erasure Coding

Replication

Erasure Coding (EC)
Safety Restart for DataNode (EC)

Upgrade, restart and wait for Missing Blocks to be 0
Test Jobs for Each Component

- HDFS Read/Write
  - `hdfs dfs -put /tmp/test`

- MapReduce
  - `yarn jar hadoop-mapreduce-examples.jar pi 5 10`

- Hive, Hive on Tez
  - `hive -e "select x from default.dual group by x"`

- Pig, Spark

- HttpFS, Oozie
  - `curl --negotiate -u "https://.../webhdfs/v1/?op=liststatus"`
Results
Results of Non-stop Upgrade

- Hadoop 2.3.x to 2.3.y upgrade
- 7 minutes of downtime
- 0.3% of job failure
- 0% data loss
## Results of Non-stop Upgrade

<table>
<thead>
<tr>
<th>Component</th>
<th>Non-stop</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDFS</td>
<td>✅</td>
<td></td>
</tr>
<tr>
<td>HttpFS</td>
<td>✅</td>
<td></td>
</tr>
<tr>
<td>MapReduce</td>
<td>✅</td>
<td></td>
</tr>
<tr>
<td>Hive</td>
<td>✅</td>
<td></td>
</tr>
<tr>
<td>Pig</td>
<td>✅</td>
<td></td>
</tr>
<tr>
<td>Spark</td>
<td>△</td>
<td>Disconnected from Hive Metastore, few jobs failed</td>
</tr>
<tr>
<td>HiveServer2</td>
<td>△</td>
<td>Checksum error occurred for some jobs</td>
</tr>
<tr>
<td>Oozie</td>
<td>△</td>
<td>Non-HA component</td>
</tr>
</tbody>
</table>

✅ successful upgrade without any job failures
△ Upgrade was affected up to some extent
Problems During the Upgrade

- NameNode restart
- DataNode restart
- Spark and Hive job failure
NameNode Restart

Restart standby NN (nn02) → Failover nn01 → nn02

ERROR: no enough memory

• NN metadata was huge and it took long to free memory
• As a workaround, stop NN → wait → start NN

Restart standby NN (nn01)
DataNode Restart

• Restart failed due to existence of old pid file
• 2 DataNode Processes
  • Parent: /var/run/hdfs/hadoop-hdfs-datanode.pid
  • Child: /var/run/hdfs/hadoop_secure_dn.pid

Usually gets deleted automatically, but

Starting regular datanode initialization
Still running according to PID file /var/run/hdfs/hadoop_secure_dn.pid
Spark (v1.5) Job Failure

Spark job → Connecting → Hive Metastore
Spark (v1.5) Job Failure

- Spark job
- Hive Metastore

Disconnected

Does not reconnect

Restart to upgrade
Hive Job Failure

```
def setup_hiveserver2():
    ...
    copy_to_hdfs("mapreduce", ...)```

- Uploaded library timestamp: yyyy
- Cached library timestamp: xxxx
- java.io.IOException: .../mapreduce.tar.gz changed on src filesystem

Restart HiveServer2

Upload necessary tarballs

HiveServer2

HDFS

Client
Conclusion
Non-stop Upgrade

• Hadoop 2.3.x to 2.3.y upgrade
• 7 minutes of downtime
• 0.3% of job failure
• 0% data loss
• No need of separate resource scheduling
## Comparison of Upgrade Results

<table>
<thead>
<tr>
<th></th>
<th>Previous</th>
<th>Proposed (without automation)</th>
<th>Proposed (with automation)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cluster Downtime</strong></td>
<td>12h</td>
<td>0min</td>
<td>7min</td>
</tr>
<tr>
<td><strong>Maintenance Time</strong></td>
<td>12h</td>
<td>72h</td>
<td>72h</td>
</tr>
<tr>
<td><strong>Man-hour</strong></td>
<td>108h</td>
<td>648h</td>
<td>21h</td>
</tr>
</tbody>
</table>

- **User impact**
- **Operating cost**

---

**Note:** The proposed upgrade results with automation significantly reduce cluster downtime and maintenance time, while minimizing user impact and operating costs.
Future Work

- Improving non-stop upgrade by bringing down
  - Cluster downtime to 0
  - Job failures to 0
- To be able to perform major upgrades (upgrades involving NameNode metadata changes)
- Automating of preparation stage and handle failures with recovery operations
- Contributing to OSS