**<Customer Name>**

HDP Production Environment  
Health Check Report

12 November 2017

Version 1.0

Table of contents:

1. Introduction 8

1.1. Objectives 8

5.1. 1.2. Intended Audience 8

2. Engagement Approach 9

3. Current State Architecture 9

3.1. CPU 10

5.2. 3.2. Memory 10

5.3. 3.3. Disks 10

5.4. 3.4. Network 11

4. Observations summary 11

5. OS level observations 12

4.1. OS version 12

4.2. Hostnames 13

4.3. User session system resources limits 14

4.4. Block devices read-ahead configuration 14

4.5. Linux I/O scheduler queue size 14

4.6. Kernel readahead size during a sequential read operations 15

4.7. Maximum allowed size of an I/O request for a single operation 15

4.8. XFS filesystem mount options 15

4.9. Linux kernel swappiness 16

4.10. Linux disk caching for Hadoop workload 16

4.11. TCP Selective Acknowledgements (SACK) 17

4.12. Duplicate SACKs 18

4.13. Max number of packets in the INPUT queue 18

4.14. Number of TCP keepalive probes 18

4.15. Interval between subsequential keepalive probes 18

4.16. TCP connection receive buffer 19

4.17. TCP connection transmit buffer 19

4.18. Nine system daemons configured for 0, 1 and 6 runlevels 19

4.19. Inconsistent software profile for the HDP metastores 20

4.20. Disaster recovery on the RDBMS metastore backend 20

4.21. /var file system size 20

4.22. Separate system /var file system and Hadoop one 21

4.23. Disable Transparent Huge Pages at boot time 21

4.24. Logical Volumes on the cluster datanodes 21

4.25. No single sign-on implemented for the Linux authentication 22

4.26. Conducting the OS configuration changes in batches 22

6. General HDP observations 23

5.1. Current HDP version 23

5.2. Current Ambari version 23

5.3. Current Java version 24

5.4. Current Java versions mismatch 24

5.5. Missing components in current Java versions 25

5.6. Kerberos authentication 25

5.7. Ranger authorization 26

5.8. HDFS Transparent Data Encryption 27

5.9. Knox Gateway 27

5.10. SmartSense 27

5.11. Conducting the general HDP configuration changes in batches 29

6. Zookeeper observations 29

6.1. Zookeeper Server heap size 29

6.2. The number of ticks to connect to the ZooKeeper leader node 29

6.3. Timeout on waiting connection 30

6.4. Number of heartbeat ticks to the leader node 30

6.5. Conducting the general HDP configuration changes in batches 30

7. HDFS observations 31

7.1. HDFS NameNode metadata redundancy in all HDP environments 31

7.2. Accidental chance of erasing all HDFS metadata by human error 31

7.3. HDFS checkpoint process interval is too long 32

7.4. Number of new edit log transactions is too small 32

7.5. Supported compression codecs 32

7.6. Avro serializers and deserializers 33

7.7. HDFS balancing bandwidth 33

7.8. Max Number of Concurrent Moves 34

7.9. Asynchronous edit logs in the Namenode 34

7.10. HDFS block location tracking disabled 34

7.11. NameNode log performance 35

7.12. HDFS and YARN files share the same physical disks 35

7.13. Dedicated Service RPC Port 35

7.14. Dedicated Lifeline RPC Port for HDFS HA 36

7.15. RPC Handler Count is too low 37

7.16. Service RPC Handler Count is too low 37

7.17. Number of DataNode server threads is too low 37

7.18. RPC Client Backoff on Client RPC port (upgrade required) 38

7.19. FairCallQueue on Client RPC Port (upgrade required) 38

7.20. Conducting the HDFS configuration changes in batches 39

8. YARN observations 40

8.1. Amount of physical memory allocated for containers 40

8.2. YARN Secure Containers 41

8.3. YARN CGroups Isolation 41

8.4. Number of CPU cores that can be allocated for containers 42

8.5. The maximum vcores allocation for every container request at the RM 43

8.6. The maximum memory allocation for every container request at the RM 43

8.7. The ResourceCalculator implementation in the Capacity Scheduler 43

8.8. Capacity Scheduler container preemption 44

8.9. YARN NodeManager health check script 44

8.10. YARN Timeline Server 1.5   44

8.11. Summary storage for ATS 45

8.12. Maximum number of retires for timeline service client 45

8.13. Client policy for timeline operations 45

8.14. Conducting YARN configuration changes in batches 46

9. MapReduce2 observations 47

9.1. MapReduce Map Memory Settings 47

9.2. MapReduce Reduce Memory Settings 48

9.3. MapReduce Sort Allocation Memory 48

9.4. MapReduce History Server heap size 49

9.5. Conducting MapReduce2 configuration changes in batches 49

10. HBase 49

10.1. Maximum Region File size 49

10.2. Online Region Merge 50

10.3. Managed Compactions 50

10.4. Sync all data and metadata of a block to disk on the block closure 50

10.5. Write HDFS data to disk immediately after it is written 51

10.6. Nagle's algorithm for the TCP socket connection on the client 51

10.7. Block pinning for HFile HDFS blocks 51

10.8. HDFS favored-nodes feature in HBase balancer 52

10.9. Parallel Garbage Collection of the Eden Space 52

10.10. Free memory allocation watermark 52

10.11. Conducting HBase configuration changes in batches 52

11. Hive 54

11.1. Hive Tez container size 54

11.2. Hive Metastore HA 54

11.3. HiveServer2 HA 55

11.4. Zookeeper Discovery for HiveServer2 HA 55

11.5. Group by in the mapper byBucketizedHiveInputFormat 55

11.6. Sort-merge bucketed map-side join enforcement 55

11.7. HiveServer2 to close operation after not beingaccessed for this duration of time 56

11.8. Check interval for session/operation timeout in HiveServer2 56

11.9. Maximum number of retries on stats publisher/aggregator 56

11.10. Support sub-directories for Hive tables/partitions 57

11.11. ORC splits to include metadata about the stripes in the file 57

11.12. Merge small files at the end of a map-reduce job 57

11.13. Table directories default permissions 57

11.14. Compress Hive intermediate files 58

11.15. Data size per reducer 58

11.16. Column Position Alias in GROUP BY and ORDER BY clauses 58

11.17. Tracking table and keys used in joins and groupby’s 59

11.18. Tracking column access in the QueryPlan 59

11.19. Aggregations with no group-by in a single reducer 59

11.20. Restrict certain risky Hive queries 59

11.21. Maximum number of dynamic partitions 60

11.22. Maximum number of dynamic partitions on a node 60

11.23. Maximum number of HDFS files created by a MapReduce job 60

11.24. Conducting Hive configuration changes in batches 60

12. Tez observations 64

12.1. The maximum heartbeat interval between the AM and RM 64

12.2. Number of allowed counters for the executing DAG 64

12.3. Class to use for logging history data 65

12.4. Slow-start behavior 65

12.5. Generating counters per I/O 65

12.6. Conducting Spark configuration changes in batches 66

13. Spark observations 66

13.1. Spark version 66

13.2. Memory to use for the YARN Application Master in client mode 67

13.3. Amount of memory per executor process 67

13.4. Max number of executors 67

13.5. JobConf caching in the HadoopRDD 68

13.6. Logs the effective SparkConf as INFO 68

13.7. Conducting Spark configuration changes in batches 68

14. Kafka 69

14.1. Ambari Management for Kafka cluster 69

14.2. Kafka topics run as user root 69

14.3. Some topics have only one replica 69

15. Flume 70

15.1. Flume version 70

16. Revision History 70

# Introduction

The document forms the deliverable of the HDP health check and architecture review engagement commissioned by and in process of conducting onsite at <Customer Name> between <Start Date> and <End Date>.

This document records the observations, findings and recommendations, allowing <Customer Name> to maintain a baseline of the current environment configuration and understand the recommended best practice system and architecture changes needed to optimally deploy and scale Hadoop within the organization.

The scope of the review includes the Production HDP environment currently deployed at <Customer Name> . During the course of the engagement, Professional Services performed a system health check and made proposal recommendations on addressing existing issues and future upgrade for the incumbent environments.

## Objectives

The purpose of this document is to achieve the following objectives:

* Deliver results of the health check
* Provide proposals

## 1.2. Intended Audience

The document serves as a reference and intended to be reviewed by the following teams within <Customer Name> :

* Business and Technology Decision Makers
* Architecture Team: Data Architects, Enterprise Architects, Hadoop Architects,  ETL Architects
* Hadoop Operational Support Team
* ETL Developers
* Data Science Team
* Planning Team
* IT Security Team

# Engagement Approach

Hortonworks adopted the following approach during the course of the engagement.

* Reviewed each of the critical cluster environment configuration parameters
* Documented the results of the observations including recommendations
* Provided the infrastructure and architecture recommendations in line with findings
* Provided an implementation plan in line with the findings above
* Provided <Customer Name> a high level presentation of the findings and recommendations
* Prepared the final documentation and recommendations (this report)

# Current State Architecture

The following section outlines the current state architecture deployed at <Customer Name> today:

All XX datanodes are the **<Hardware Make and Model>** servers.

## CPU

Model 380 servers belong to the 300 series dual socket capable systems and use Intel processors (models with a '0' in the last digit use Intel processors and models with a '5' in the last digit use AMD processors).

The processors in all 341 servers are Intel Xeon CPU E5-2690 v3 @ 2.60GHz. The E5-2690V3 model:

* 12 cores
* 24 threads
* 2.60 GHz base frequency.

## 3.2. Memory

The XXX model has 24 DIMM slots available (2 processors, 12 DIMM slots per processor). All XX datanodes have:

* 16 DDR4 Registered (RDIMM) 16384MB size @ 2133 MHz clock speed each
* 8 available slots.

It means we can potentially add 8 more 16GB @ 2133 MHz RDIMMs, i.e. 128GB RAM without replacing existing RDIMMs.

## 3.3. Disks

The configuration has:

* 600GB - 1 disk (/dev/sda)
* 6001GB – 12 disks (/dev/sdb - /dev/sdm)

All disks are configured as LOGICAL VOLUME devices, i.e the physical topology of the RAIDs is unknown; likely they are RAID-0 volumes.

**Notes**:

## 3.4. Network

Two Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network Cards.

The model:

* Has dual ports
* Has 10GbE data rate per port

# Observations summary

There are 124 recommendations proposed as result of the health check.

The breakdown of the recommendations by components:

* OS - 25
* General HDP - 10
* Zookeeper - 4
* HDFS - 19
* YARN - 14
* MapReduce2 - 4
* HBase - 10
* Hive - 23
* Tez - 5
* Spark - 6
* Kafka - 3
* Flume - 1

# OS level observations

There are 25 changes recommended on the OS level are proposed as part of the health check.

## OS version

Currently we have CentOS installed on all HDP nodes. CentOS is a Linux distribution that provides a free, community-supported computing platform functionally equal (or almost equal) to its upstream source, Red Hat Enterprise Linux.

The installed version is CentOS 6.6. The version is outdated and CentOS 6.6 no longer gets any updates, nor any security fixes.

Not having security updates leaves the OS vulnerable to attacks using known software vulnerabilities.

**Example**: recent WannaCry ransomware attack used an exploit of Windows' SMB protocol that was fixed by a critical security patch released 2 months earlier. Due to lack of applying security updates the malware infected more then 230,000 computers in over 150 countries.

In addition version 6 is already out of full updates and only maintenance updates are provided for CentOS6:

|  |  |  |  |
| --- | --- | --- | --- |
| **CentOS version** | **Release date** | **Full updates** | **Maintenance updates** |
| **3** | 19 March 2004 | 20 July 2006 | 31 October 2010 |
| **4** | 9 March 2005 | 31 March 2009 | 29 February 2012 |
| **5** | 12 April 2007 | 31 January 2014 | 31 March 2017 |
| **6** | 10 July 2011 | 10 May 2017 | 30 November 2020 |
| **7** | 7 July 2014 | Q4 2020 | 30 June 2024 |

The best practice advocates the use the most recent stable Operating Systems due to new features and improved security. Hortonworks recommends <Customer Name> to upgrade the OS to the most recent supported version of CentOS; in this case there are two options:

* CentOS 6.9 + updates
* CentOS 7.3-1611 + updates

In general, mixing even Linux releases of the same major versions is not recommended: it can create some hard-to-debug situations, even though on a surface it "works".

## Hostnames

Currently all hostnames of the Production cluster are in the uppercase. In case of the cluster kerberizaton it will immediately cause "Server not found in Kerberos database (7) - UNKNOWN\_SERVER".

RFC 4120 “The Kerberos Network Authentication Service (V5)” section 6.2.1 “Name of Server Principals” explicitly prohibits uppercase in the hostnames:

[Where the name of the host is not case sensitive (for example, with Internet domain names) the name of the host MUST be lowercase.](https://www.ietf.org/rfc/rfc4120.txt)

It is critical to convert the hostnames to lowercase before the cluster kerberizaton.

## User session system resources limits

Currently there are no limits on the system resources that can be obtained in a user-session.

In order to avoid "unable to create new native thread" errors for various Hadoop services it is important to put a the following line in /etc/pam.d/login as the last line for that service:

session required pam\_limits.so

## Block devices read-ahead configuration

Currently the read-ahead configuration for all Hadoop block devices is 256 sectors (512 bytes each) that are not sufficient for busy clusters.

The best practice is to increase the readahead size to 1024 or 2048 sectors in order to improve the performance of sequential file reads on large files. The change can be done by:

**blockdev --setra 4096 /dev/sd*X***

Preferable way to preserve readahead size for block devices is to implement changes through a Linux Device Management udev rule.

## Linux I/O scheduler queue size

Currently the maximum number of read and write requests that can be queued at one time (before the next process requesting a read or write is put to sleep) is 512.

The best practice is to increase the queue length (number of schedulable requests) to 4096 by setting **/sys/block/sd\*/queue/nr\_requests**.

Preferable way is to implement changes through a Linux Device Management udev rule.

## Kernel readahead size during a sequential read operations

Currently the kernel readahead size during a sequential read operation is 128 KB.

The best practice is to increase the maximum amount of read-ahead data to 4096 by setting **/sys/block/sd\*/queue/read\_ahead\_kb**.

Preferable way is to implement changes through a Linux Device Management udev rule.

## Maximum allowed size of an I/O request for a single operation

Currently the maximum allowed size of a single I/O request is 512.

The best practice is to increase the I/O size of a single operation to 1024 by setting **/sys/block/sd\*/queue/max\_sectors\_kb**.

Preferable way is to implement changes through a Linux Device Management udev rule.

## XFS filesystem mount options

Currently the mount options for Hadoop file systems are default, i.e. "rw, suid, dev, exec, auto, nouser, async and relatime".

The best practice is to mount XFS file systems for Hadoop following mounts options:

**defaults,noatime,nodiratime,****logbufs=8**

where:

* noatime – do not update inode access times on this filesystem
* nodiratime – do not update directory inode access times on this filesystem
* logbufs – number of in-memory log buffers (max is 8).

In order to preserve changes over reboots we need to update /etc/fstab file with recommended mount options.

## Linux kernel swappiness

All RHEL/CentOS kernels > 2.6.32-303 changed behavior when vm.swappiness=0 is configured: previously setting the paremeter to 0 was to reduce the tendency to swap userland processes, but not disable that completely, i.e. see little swapping was expected instead of OOM.

Since the production servers run on the 2.6.32-504 kernel, we should avoid OOM termination conditions even though swap is not being used and memory does not appear to be actually exhausted by setting **vm.swappiness = 1**.

## Linux disk caching for Hadoop workload

The default values of the highest percentage of the memory to hold dirty data (vm.dirty\_ratio) and lowest percentage of the memory where pdflush stops writing dirty data (vm.dirty\_background\_ratio) do not fit Hadoop I/O profile.

 The best practice is to use optimal for Hadoop values for dirty data in memory. Hortonworks recommends to set parameters for Hadoop (some experimentation may be necessary to find the perfect settings for specific workload):

* vm.dirty\_ratio = 70
* vm.dirty\_background\_ratio = 5

i.e. pdflush kernel threads start writing the dirty pages to disk in background when their number in the page cache hits 5% of the system memory, but will not force synchronous I/O until it gets to 70% full.

## TCP Selective Acknowledgements (SACK)

TCP Selective Acknowledgments (SACK) is TCP extension defined in [RFC 2018 - TCP Selective Acknowledgment Options](http://tools.ietf.org/html/rfc2018).

A basic TCP Acknowledgment (ACK) only allows the receiver to advise the sender which bytes have been received. When packet loss occurs, this requires the sender to retransmit all bytes from the point of loss, which can be inefficient. SACK allows a sender to specify which bytes have been lost and which bytes have been received, so the sender can retransmit only the lost bytes.

There is some research available in the networking community which shows enabling SACK on high-bandwidth links can cause unnecessary CPU cycles to be spent calculating SACK values, reducing overall efficiency of TCP connections. This research implies these links are so fast, the overhead of retransmitting small amounts of data is less than the overhead of calculating the data to provide as part of a Selective Acknowledgment.

Unless there is high latency or high packet loss, it is most likely better to keep SACK turned off over a high performance network.

SACK can be turned off with kernel tunables: net.ipv4.tcp\_sack = 0

## Duplicate SACKs

Currently TCP sends "duplicate" SACKs.

It is a good idea to prohibit sending "duplicate" SACKs by setting net.ipv4.tcp\_dsack = 0

## Max number of packets in the INPUT queue

Currently the max number of packets in the INPUT queue when the interface receives packets faster than kernel can process them is 1,000.

It could not optimal for Hadoop workloads and it is a good idea to increase at least to 30,000 by setting net.core.netdev\_max\_backlog = 30000.

## Number of TCP keepalive probes

Currently number of unacknowledged probes to send before considering the connection dead and notifying the application layer is 9.

It could not optimal for Hadoop workloads and it is a good idea to decrease the number of keepalive probes to 5 by setting net.ipv4.tcp\_keepalive\_probes = 5.

## Interval between subsequential keepalive probes

Currently the keepalive process resends probes every 75 seconds. If no ACK response is received for tcp\_keepalive\_probes consecutive times, the connection is marked as broken.

It could not optimal for Hadoop workloads and it is a good idea to decrease interval to not more than 15 seconds by setting net.ipv4.tcp\_keepalive\_intvl = 15.

## TCP connection receive buffer

Currently min, default, max receive buffer for each TCP connection is 4 KB, 16 KB and 4 MB accordingly.

It could not optimal for Hadoop workloads and it is a good idea to increase a connection receive buffer to min 4 KB, default 256 KB and max 16 MB by setting: net.ipv4.tcp\_rmem = 4096 262144 16777216.

## TCP connection transmit buffer

Currently min, default, max transmit buffer for each TCP connection is 4 KB, 16 KB and 4 MB accordingly.

It could not optimal for Hadoop workloads and it is a good idea to increase a connection transmit buffer to min 4 KB, default 256 KB and max 16 MB by setting: net.ipv4.tcp\_wmem = 4096 262144 16777216.

## Nine system daemons configured for 0, 1 and 6 runlevels

Currently there are nine system daemons:

* acpid
* auditd
* cpuspeed
* lvm-monitor2
* messagebus
* network
* ntpd
* sshd
* sysstat

are configured to run on the Halt (0), Single-User Mode (1) and Reboot (6) runlevels, that they supposed to be turned off.

It is highly advisable to turn the daemons off on these runlevels.

## Inconsistent software profile for the HDP metastores

The metastores are not consolidated and running using both MySQL and PostgreSQL RDBMS backends.

Common practice is to use one metastore to simplify administration, backups and disaster recovery policies. Also it decreases skillsets required to support the RDBMS databases.

Hortonworks recommends <Customer Name> to use only one RDBMS type for the metastores.

## Disaster recovery on the RDBMS metastore backend

No real-time replication between the primary and backup RDBMS for the metastores backend.

Better practice is to Master-Master replication for the metastores that allows replicating every committed transaction and have warm standby database ready.

Hortonworks recommends to configure Master-Master in Active-Passive mode (MMAP) replication for MySQL databases.

## /var file system size

At least once a Production cluster master node run out the space in the /var file system, that caused number of system processes failures and leaded to unplanned server reboot.

It is recommended to have sufficient space on the /var file system in order to stay in the free space safe zone.

## Separate system /var file system and Hadoop one

Currently all Hadoop logs are located on the /var file system, that in addition to space requirements adds extra I/O on the underlying disk(s).

It can be a good idea to create a separate file system dedicated to Hadoop processes logs.

## Disable Transparent Huge Pages at boot time

Currently the most of Hadoop nodes have Transparent Huge Pages disabled. However, at least one node with THP enabled was discovered. THP feature is known to perform poorly in Hadoop cluster and results in excessively high CPU utilization.

The best practice is to disable the THP by appending the following to the kernel command line in grub.conf:

**transparent\_hugepage=never**

**Note**: Certain ktune and/or tuned profiles specify to enable THP when they are applied. Please refer RedHat KB article 422283 on addressing THP in the ktune and tuned profiles.

## Logical Volumes on the cluster datanodes

Currently LVMs are configured on the datanodes. Per file system partitioning best practices, we should avoid using LVM since it adds latency and could become a bottleneck.

It is a good idea to wait until we want to rebuild datanodes (example: for major OS upgrade) and create all conventional partitions.

## No single sign-on implemented for the Linux authentication

 User accounts are authenticated from local credentials; integration with a Single sign-on source is not configured.

Best practice is to use a centralized authentication service to maintain central control over account and authentication data to prevent credentials to become inconsistent or out-of-date. It is a good idea to:

* Use a centralized authentication service (LDAP or Active Directory)
* Ensure that authentication information is never sent over the network  unencrypted
* Ensure that the root account has a local password, to allow recovery in case  of network outage or authentication server failure

## Conducting the OS configuration changes in batches

Following changes can be done in one batch:

* Block devices read-ahead configuration
* Linux I/O scheduler queue size
* Kernel readahead size during a sequential read operations
* Maximum allowed size of an I/O request for a single operation
* Linux kernel swappiness
* Linux disk caching for Hadoop workload
* TCP Selective Acknowledgements (SACK)
* Duplicate SACKs
* Max number of packets in the INPUT queue
* Number of TCP keepalive probes
* Interval between subsequential keepalive probes
* TCP connection receive buffer
* TCP connection transmit buffer

The rest of changes required to be independent and not combined unless we are reinstalling the OS or building a new environment.

# General HDP observations

There are 10 changes recommended in the general HDP level are proposed as part of the health check.

## Current HDP version

The Production environment has HDP 2.4.0.0-169 installed.

There are two major versions (HDP-2.5 and HDP-2.6) are released since and there are 7 minor (bugfixes) releases:

* HDP-2.4.2.0
* HDP-2.4.3.0
* HDP-2.5.0.0
* HDP-2.5.3.0
* HDP-2.5.5.0
* HDP-2.6.0.3
* HDP-2.6.1.0

Best practice is to use the most recent stable HDP release.

Hortonworks recommends <Customer Name> to upgrade Hadoop to the HDP-2.6.1, that is the most recent stable release.

## Current Ambari version

The Production environment has Ambari 2.2.1.1 installed.

There are 8 versions released since:

* 2.2.1.1
* 2.2.2.0
* 2.4.0.0
* 2.4.0.1
* 2.4.1.0
* 2.4.2.0
* 2.4.3.0
* 2.5.0.3
* 2.5.1.0

Best practice is to use the most recent stable Ambari release.

Hortonworks recommends <Customer Name> to upgrade Ambari to the version 2.5.1.0, that is the most recent stable release.

## Current Java version

Oracle JDK 1.7 and OpenJDK 7 are marked deprecated as of HDP 2.6.0 and will be removed in HDP 3.0.0.

Hortonworks recommends <Customer Name> to start planning JDK upgrade to 1.8.0. The current recommended release is Oracle JDK 1.8.0\_77. The location of exact binaries in the official Hortonworks repository:

<http://s3.amazonaws.com/public-repo-1.hortonworks.com/ARTIFACTS/jdk-8u77-linux-x64.tar.gz>

## Current Java versions mismatch

Currently at least some servers have more than one version of Java likely installed in the JAVA\_HOME. Example:

[root@NVMBD2AJV030D00 tmp]# /usr/bin/javac -version

javac 1.8.0\_72

[root@NVMBD2AJV030D00 tmp]# /usr/bin/java -version

java version "1.7.0\_91"

Java(TM) SE Runtime Environment (build 1.7.0\_91-b31)

Java HotSpot(TM) 64-Bit Server VM (build 24.91-b03, mixed mode)

[root@NVMBD2AJV030D00 tmp]#

Java is important component of any Hadoop framework and must be stable and consistent across the cluster including the same version/release. It is important to fix any Java mismatches.

## Missing components in current Java versions

Currently at least some servers have missing Java binaries. Example:

[root@NVMBD2AFU210D00 tmp]# /usr/bin/javac -version

-bash: /usr/bin/javac: No such file or directory

[root@NVMBD2AFU210D00 tmp]# ll /usr/bin/javac

lrwxrwxrwx 1 root root 23 Jul  6  2015 /usr/bin/javac -> /etc/alternatives/javac

[root@NVMBD2AFU210D00 tmp]# ll /etc/alternatives/javac

lrwxrwxrwx 1 root root 42 Jul  6  2015 /etc/alternatives/javac -> /u01/app/oracle/java/jdk1.7.0\_75/bin/javac

[root@NVMBD2AFU210D00 tmp]# ll /u01/app/oracle/java/jdk1.7.0\_75/bin/javac

ls: cannot access /u01/app/oracle/java/jdk1.7.0\_75/bin/javac: No such file or directory

[root@NVMBD2AFU210D00 tmp]#

Java is important component of any Hadoop framework and must be stable and consistent across the cluster including the same version/release. It is important to fix any Java missing binaries/libraries.

## Kerberos authentication

Strong authentication and establishing a user’s identity is the basis for secure access in Hadoop. Hadoop uses Kerberos for strong authentication and identity propagation. Kerberos is a third party authentication mechanism, in which users and services rely on a third party (the Kerberos server) to authenticate each to the other. The Kerberos server itself is known as KDC (Key Distribution Center).

Currently we do not have Kerberos configured and the production cluster kerberized.

The best practices in security is to kerberize Hadoop environments and the first step in implementing security. Additional argument for kerberization case of the Production environment is to enable Linux Control Group Isolation in order set proper CPU shares for OS, HBase and YARN processes.

## Ranger authorization

Ranger provides a centralized platform to define, administer and manage security policies consistently across Hadoop components:

* HDFS
* YARN
* Hive
* HBase
* Storm
* Knox
* Solr
* Kafka
* NiFi

Using the Apache Ranger console, security administrators can easily manage policies for access to files, folders, databases, tables, or column. These policies can be set for individual users or groups and then enforced consistently across HDP stack.

Ranger also provides security administrators with deep visibility into their Hadoop environment through a centralized audit location that tracks all the access requests in real time and support multiple destination sources including Solr and HDFS.

Currently we do not have Ranger framework implemented in the <Customer Name> environments. The best practice in security is to have all policies centrally managed by Ranger.

## HDFS Transparent Data Encryption

The Ranger Key Management Service (Ranger KMS) provides a scalable cryptographic key management service for HDFS data at rest encryption.

Ranger KMS is based on the Hadoop KMS originally developed by the Apache community and extends the native Hadoop KMS functionality by allowing system administrators to store keys in a secure database.

Currently we do not have Ranger KMS implemented in the <Customer Name> environments. If we have sensitive data (personal information, financial data, etc.) in Hadoop it is advisable to have it in encrypted format.

## Knox Gateway

The Knox Gateway provides perimeter security in order to allow securely extend Hadoop access to new users with maintaining compliance with security policies. Basically Knox Gateway is a secure entry point for Hadoop clusters.

For many user tasks it eliminates requirements to have SSH edge nodes.

Currently we do not have Knox Gateway implemented in the <Customer Name> environments. In order to allow secure access to the environments for number of end users it is a good idea to implement Knox.

## SmartSense

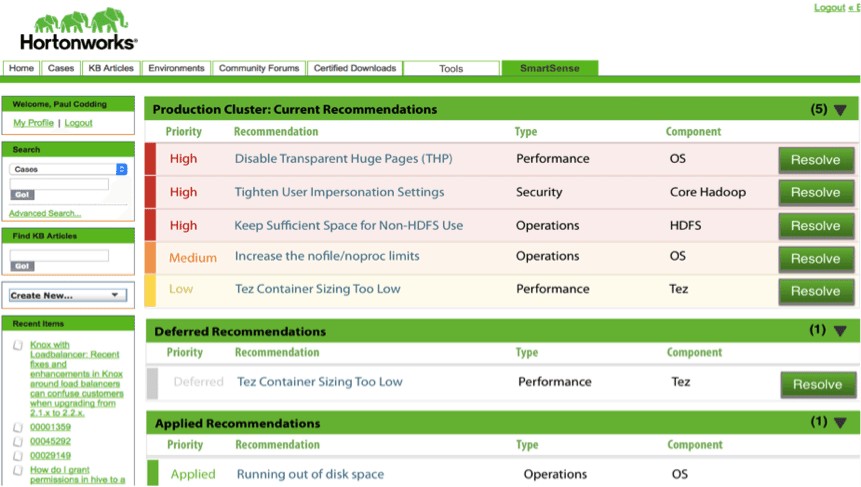
SmartSense is a collection of tools and services that help admins of Hortonworks Data Platform quickly resolve current issues and act on proactive recommendations that help avoid future issues.

Basically, SmartSense has fundamentally changed the model for delivering Hadoop support from reactive to proactive monitoring and recommendations that are actually delivered even prior to experiencing an issue.

SmartSense collects the data and analyzes the bundle, new recommendations are displayed in the Hortonworks Support Portal. Each recommendation includes:

* Description/justification of the recommended change
* Steps to follow in order to apply the recommendation
* List of the impacted components
* Description of associated risks of implementing the recommendation
* List of the hosts that will be affected by the recommendation

Example page with SmartSense recommendations:



## Conducting the general HDP configuration changes in batches

Java consistence and stability changes are top priority. The recommendations below should be implemented together as one batch:

* Current Java versions mismatch
* Missing components in current Java versions

The rest of changes required can independent and not combined unless we are building a new environment.

# Zookeeper observations

There are 4 changes recommended in the Zookeeper configuration. The changes are related to the Zookeeper processes heap sizes and the connection retries/timeouts.

## Zookeeper Server heap size

Currently the Zookeeper server heap size is 1 GB.

It is a good idea to increase the heap to 4 GB by setting SERVER\_JVMFLAGS= "-Xms4096m -Xmx4096m" in the zookeeper-env.sh configuration file.

## The number of ticks to connect to the ZooKeeper leader node

Currently the number of ticks that the initial synchronization phase can take is 10. The initLimit property defines how many ticks to wait during connection attempts to the ZooKeeper leader node. Example: if initLimit is set to 5 and tickTime is 2,000 millseconds, then a ZooKeeper node waits for 5\*2 seconds before giving up on the connection and deciding to elect a new leader.

It is a good idea to increase timeout ZooKeeper uses to limit time the ZooKeeper servers in quorum have to connect to a leader to 20 by setting initLimit=20 in the zoo.cfg configuration file.

## Timeout on waiting connection

Currently the time basic unit used by ZooKeeper is 5 seconds.

It is a good idea to decrease connection timeout to 2 seconds by setting tickTime=2000 in the zoo.cfg configuration file.

## Number of heartbeat ticks to the leader node

Currently the number of ticks that can pass between sending a request and getting an acknowledgement from a leader is 5.

It is a good idea to increase number of ticks to 10 in the zoo.cfg configuration file.

## Conducting the general HDP configuration changes in batches

The configuration change can be cumulative: all four proposed changes could be done together.

In order to have the Zookeeper service uninterrupted it can be done one by one in rolling mode.

# HDFS observations

There are 4 changes recommended in the HDFS configuration. The changes are related to different HDFS properties and some of the can be done after the HDP upgrade to the recent version.

## HDFS NameNode metadata redundancy in all HDP environments

Only one HDFS metadata (fsimage) directory exists per a NameNode that is not accordance the best practices.

The suggested best practice is to have the name table (fsimage) mirrored across minimum two local directories.

Hortonworks recommends <Customer Name> to set dfs.namenode.name.dir = /data/hadoop/hdfs/namenode,*<the 2nd local directory>* in the hdfs-site.xml configuration file.

## Accidental chance of erasing all HDFS metadata by human error

HDFS NameNode metadata has format allowed today. This could potentially lead to a catastrophic human error. This is applicable for all HDP environments at <Customer Name> .

Best practice is to disable this property. Hortonworks recommends <Customer Name> to set dfs.namenode.support.allow.format = false in the hdfs-site.xml configuration file.

Note: this configuration change must be done after the initial HDFS configuration and the first HDFS metadata format.

## HDFS checkpoint process interval is too long

The number of seconds between two periodic HDFS checkpoints is set 21,600 (6 hours).

Best practice is to merge the content of the most recent fsimage with all edits every hour or less in order to decrease of NameNode startup delay as the NameNode reapplies all the edits. Hortonworks recommends <Customer Name> to set dfs.namenode.checkpoint.period = 3,600 (default value) in the hdfs-site.xml configuration file.

## Number of new edit log transactions is too small

The number of new edit log transactions between two periodic HDFS checkpoints is set 1,000,000.

Best practice is to increase the number of new edit log transactions to 10,000,000 in order to avoid too often checkpoint of the namespace. Hortonworks recommends <Customer Name> to set dfs.namenode.checkpoint.txns = 10000000 in the hdfs-site.xml configuration file.

## Supported compression codecs

The list of the compression codec classes that can be used for compression/decompression includes only GzipCodec, DefaultCodec, Snappy codec is available.

Best practice is to enable all supported compression codecs. Hortonworks recommends <Customer Name> to set io.compression.codecs to following values (comma separated) in the core-site.xml configuration file:

* org.apache.hadoop.io.compress.DefaultCodec
* org.apache.hadoop.io.compress.GzipCodec
* org.apache.hadoop.io.compress.BZip2Codec
* org.apache.hadoop.io.compress.DeflateCodec
* org.apache.hadoop.io.compress.SnappyCodec
* org.apache.hadoop.io.compress.Lz4Codec

## Avro serializers and deserializers

The list of serialization classes that can be used for obtaining serializers and deserializers include only org.apache.hadoop.io.serializer.WritableSerialization.

Better practice is to make Avro serializers and deserializers also enables that allows supporting schema evolution (i.e. different versions of the schema at the same time). Hortonworks recommends <Customer Name> to set value of io.serializations to default in the core-site.xml configuration file:

* org.apache.hadoop.io.serializer.WritableSerialization
* org.apache.hadoop.io.serializer.avro.AvroSpecificSerialization
* org.apache.hadoop.io.serializer.avro.AvroReflectSerialization

## HDFS balancing bandwidth

HDFS balancing bandwidth is 6,250,000 (6 MB/second) that is not always sufficient for unbalanced clusters.

Best practice is to increase to realistic values for the HDFS network: 256 MB/second (268,435,456) in order to utilize 10Gb interfaces. After some experimentation the balancing bandwidth can be increased to 512 MB/second (536,870,912).

The configuration change can be done dynamically by executing command:

hdfs dfsadmin -setBalancerBandwidth 268435456

To preserve this value on HDFS restarts Hortonworks recommends <Customer Name> to set dfs.datanode.balance.bandwidthPerSec = 268435456 in the hdfs-site.xml configuration file.

The configuration change can be done independently from any other configuration changes, adding the value to hdfs-site.xml configuration file can be done as part of any batch.

## Max Number of Concurrent Moves

Currently the maximum number of concurrent block moves that a Datanode is allowed for balancing the cluster is 5 (default value).

Best practice is to set Max Number of Concurrent Moves to a small multiple of the number of disks allocated for HDFS in the Datanodes. A higher value may not be useful, but only increases disk contention.

In our case it is 2\*12 = 24.

Hortonworks recommends <Customer Name> to set dfs.datanode.balance.max.concurrent.moves = 24 in the hdfs-site.xml configuration file.

## Asynchronous edit logs in the Namenode

Currently the Namenode uses the traditional synchronous edit logs.

Best practice is to enable async audit logging. It can minimize the impact of audit log I/Os on namenode performance. Hortonworks recommends <Customer Name> to set dfs.namenode.edits.asynclogging = true in the hdfs-site.xml configuration file.

## HDFS block location tracking disabled

HDFS disk metadata allowing clients to query a DataNode for the storage information of a list of blocks is not enabled.

Better practice is to enable for performance short-circuit local read properties including block location tracking. Hortonworks recommends <Customer Name> to set dfs.datanode.hdfs-blocks-metadata.enabled = true in the hdfs-site.xml configuration file.

## NameNode log performance

The NameNode logging level is set to track block status changes that significantly increases amount of data written to the NameNode log.

Common practice is to limit NameNode logging in production clusters. Hortonworks recommends <Customer Name> to suppress normal block state change messages from BlockManager in the NameNode in the log4j.properties configuration file:

log4j.logger.BlockStateChange=WARN

## HDFS and YARN files share the same physical disks

HDFS, YARN localized files and logs share the physical disks that causes additional I/O contention due to different I/O profiles.

The best practice is to allocate separate physical disks for HDFS and YARN. Hortonworks recommends <Customer Name> to format Hadoop allocated disks in the way when each disk is dedicated to either HDFS, or YARN. Exact ratio of disks allocated for particular framework needs to be determined by some experimentation.

## Dedicated Service RPC Port

No service RPC ports configured.

Currently there are no dedicated RPC address for HDFS Services communication. Datanodes and all other services are connecting to the same address that handles all clients requests.

The best practice is to configure a service RPC ports in order to allocate a dedicated port for DataNodes to report status via block reports and heartbeats. The port is also used by Zookeeper Failover Controllers for periodic health checks by the automatic failover logic. For HDFS HA the properties/values should look like:

* dfs.namenode.servicerpc-address.jcpprdha.nn1 = NVMBD2AFU210D00.prd.rjil.ril.com:8040
* dfs.namenode.servicerpc-address.jcpprdha.nn2 = NVMBD2AFV190D00.prd.rjil.ril.com:8040

in the hdfs-site.xml configuration file.

Note: setting this property requires HDFS downtime, i.e. shutting down all components for the configuration change.

## Dedicated Lifeline RPC Port for HDFS HA

Currently there are no Lifeline protocol ports configured in all environments.

The best practice is to offload of NameNode HA service health checks (reporting DataNode liveness and basic health information to the NameNode) to a separate RPC server. It allows a DataNodes to report that it is still alive to the NameNode via a fallback protocol, separate from the existing heartbeat messages. This can prevent the NameNode from incorrectly marking a DataNode as stale or dead in busy clusters where heartbeat processing is suffering delays.

For Production HDFS HA the properties in the hdfs-site.xml should look like:

* dfs.namenode.lifeline.rpc-address.jcpprdha.nn1 = NVMBD2AFU210D00.prd.rjil.ril.com:8050
* dfs.namenode. lifeline.rpc-address.jcpprdha.nn2 = NVMBD2AFV190D00.prd.rjil.ril.com:8050

Note: setting this property requires HDFS downtime, i.e. shutting down all components for the configuration change.

## RPC Handler Count is too low

Currently the value of the RPC Handler Count is too low. The Hadoop RPC server consists of a single RPC queue per port and multiple handler (worker) threads that dequeue and process requests. If the number of handlers is insufficient, then the RPC queue starts building up and eventually overflows.

Best practice is to set the RPC handler count property dfs.namenode.handler.count to 20\*log2(number of datanodes) with an upper limit of 200.

In our case it is 20\*log2(341) = 170.

Hortonworks recommends <Customer Name> to set dfs.namenode.handler.count = 170 in the hdfs-site.xml configuration file.

## Service RPC Handler Count is too low

Currently the Service RPC Handler Count is too low.

Best practice is to set the Service RPC handler count to 50% of the RPC handler count in busy clusters.

## Number of DataNode server threads is too low

Currently the number of server threads that handle remote procedure calls has default value (10). It could be less than optimal on the busy clusters.

Best practice is to set number of DataNode server threads to higher value. Value 40 can be a good start.

Hortonworks recommends <Customer Name> to set dfs.datanode.handler.count = 40 in the hdfs-site.xml configuration file.

## RPC Client Backoff on Client RPC port (upgrade required)

In the HDP-2.4.2 and more recent HDP version RPC Congestion Control can be configured in order to help Hadoop Services respond more predictably under high load: when the NameNode RPC queue is full it sends an explicit signal back to the client. Instead of waiting for a request that may never complete, the client throttles itself by resubmitting the request with an exponentially increasing delay.

Best practice is to enable RPC Congestion Control by setting ipc.8020.backoff.enable = true in the core-site.xml configuration file after the cluster upgrade.

Notes:

* Do not enable this setting for the Service RPC port or the DataNode lifeline port.
* Do not enable this setting on HDP versions earlier than HDP−2.4.2.

## FairCallQueue on Client RPC Port (upgrade required)

The RPC FairCallQueue can be configured to replace the single RPC queue with multiple prioritized queues. The RPC server maintains a history of recent requests grouped by user. It places incoming requests into an appropriate queue based on the user's history. RPC handler threads will dequeue requests from higher priority queues with a higher probability.

Best practice is to set ipc.8020.callqueue.impl = org.apache.hadoop.ipc.FairCallQueue in the core-site.xml configuration file after the cluster upgrade.

Notes:

* Do not enable this setting for the Service RPC port or the DataNode lifeline port.
* Do not enable this setting on HDP versions earlier than HDP−2.4.2.

## Conducting the HDFS configuration changes in batches

The changes can be done in batches:

**Batch 1:**

* HDFS NameNode metadata redundancy
* Accidental chance of erasing all HDFS metadata by human error

No downtime required; the change can be rolling.

**Batch 2:**

* HDFS checkpoint process interval is too long
* Number of new edit log transactions is too small

No downtime required; the change can be rolling.

**Batch 3:**

* Supported compression codecs
* Avro serializers and deserializers

No downtime required; the change can be rolling.

**Batch 4:**

* Asynchronous edit logs in the Namenode
* NameNode log performance in all environments

No downtime required; the change can be rolling.

**Batch 5:**

* HDFS balancing bandwidth
* Max Number of Concurrent Moves
* HDFS block location tracking disabled
* RPC Handler Count is too low
* Service RPC Handler Count is too low
* Number of DataNode server threads is too low

No downtime required; the change can be rolling.

**Batch 6:**

* Dedicated Service RPC Port
* Dedicated Lifeline RPC Port for HDFS HA

Downtime is required; the change cannot be rolling.

**Batch 7:**

* RPC Client Backoff on Client RPC port (after upgrade)
* FairCallQueue on Client RPC Port (after upgrade)

HDP upgrade is required.

Downtime is required; the change cannot be rolling.

# YARN observations

## Amount of physical memory allocated for containers

The physical memory allocated for YARN containers is 159,744 MB (156 GB).

Best practice is to allocate for YARN containers free memory on a Nodemanager.

In our case we have 256GB of RAM on all worker nodes and:

* 6GB reserved for the OS
* 32GB allocated for HBase RegionServer heap
* 4GB allocated for HDFS DataNode Heapsize
* 8GB allocated for YARN NodeManager Heapsize
* 4GB allocated for Ambari Metrics Monitor

Based on the memory allocation we can increase yarn.nodemanager.resource.memory-mb up to 200GB in the yarn-site.xml configuration file.

Notes:

* This change should be done atomically (one at a time)
* Rolling restart of the components is required
* Necessary tests are highly recommended to verify the stability of the change

## YARN Secure Containers

Linuc Control Groups (CGroups) is a Linux kernel feature and currently only exposed via the LinuxContainerExecutor.

YARN secure containers needs to be enabled by setting

yarn.nodemanager.container-executor.class = org.apache.hadoop.yarn.server.nodemanager.LinuxContainerExecutor in the yarn-site.xml configuration file.

## YARN CGroups Isolation

Currently memory is the only supported resource for applications on YARN: a NodeManager monitors the memory usage and if the memory usage of any container exceeds its allocation, it kills that container. This ensures that containers are always strictly limited to the memory requested. But the NodeManager cannot ensure that containers don’t exceed their vcore allocation.

Best practice is to implement CGroups isolation to ensure that the container launched is limited to the percentage of CPU that it was allocated and set yarn.nodemanager.linux-container-executor.resources-handler.class = org.apache.hadoop.yarn.server.nodemanager.util.CgroupsLCEResourcesHandler in the yarn-site.xml configuration file. Also Cgroups needs to be configured on the OS level.

Notes:

* HDP-2.4.0 supports CGroups on Centos6/RHEL6. HDP-2.4.0 does not support the default CGroups on Centos7/RHEL7
* LinuxContainerExecutor should be used as the container-executor class
* CGroups require that the HDP cluster be Kerberos enabled

## Number of CPU cores that can be allocated for containers

The number of CPUs allocated for YARN containers is 48, i.e. equals to the number of vcores on the servers.

Best practice is to reserve some CPUs for the non-YARN services: OS, HDFS DataNode, YARN NodeManager, Ambari Metrics Monitor, etc.

In our case reasonable configuration could be 8 CPUs reserved for non-YARN processes and we can set yarn.nodemanager.resource.percentage-physical-cpu-limit = 85 in the yarn-site.xml configuration file.

The values can be adjusted after performance impact analysis.

**Note**: this feature requires Control Groups (CGroups) isolation enabled.

## The maximum vcores allocation for every container request at the RM

Currently the maximum allocation for every container request is 8 vcores, but the number of available vcores for YARN is 40 on every NodeManager of the cluster.

Best practice is to set the maximum virtual CPU cores allocation for every container request at the RM to number of vcores available for YARN NodeManager. In our case it is 40, i.e. yarn.scheduler.maximum-allocation-vcores = 40 in the yarn-site.xml configuration file.

## The maximum memory allocation for every container request at the RM

Currently the maximum memory allocation for every container request is 159744 MB, but the amount of available memory for YARN is approximately 200GB on every NodeManager of the cluster.

Best practice is to set in the yarn-site.xml configuration file yarn.scheduler.maximum-allocation-mb to amount of memory available for YARN NodeManager. In our case it is 204800 MB.

## The ResourceCalculator implementation in the Capacity Scheduler

Currently the ResourceCalculator implementation uses only memory to compare resources in the scheduler, but we need also to take into account CPU.

to set in the yarn-site.xml yarn.scheduler.capacity.resource-calculator = org.apache.hadoop.yarn.util.resource.DominantResourceCalculator.

## Capacity Scheduler container preemption

Currently YARN preemption is not enabled, i.e. under-served queues can begin to claim their allocated cluster resources without having to wait for other queues' applications to finish running.

Best practices is to enable preemption by setting yarn.resourcemanager.scheduler.monitor.enable = true in the yarn-site.xml configuration file.

## YARN NodeManager health check script

 NodeManager health check script is not implemented.

It could be a good idea to develop and implement health checker script that will be invoked by the health checker service. A health check script is not mandatory, but if no script is specified, only the disk checker status will be used to determine the health of the node. yarn.nodemanager.health-checker.script.path.

## YARN Timeline Server 1.5

 YARN Timeline Server 1.0 is configured and running in the PROD environment.

YARN Timeline Server 1.5 is released in HDP-2.2.9 and includes changes and fixes to improve scalability across the whole stack, introduces new client APIs, a new storage architecture and supports a plugin mechanism to work with YARN applications. YARN Timeline Server 1.5 can be enabled by setting yarn.timeline-service.version = 1.5 in yarn-site.xml.

Note:

* Upgrading from ATS v1 to v1.5 may cause data stored in ATS v1.0 storage to be inaccessible.
* The upgrade process is not rolling: applications may lose previous data stored in ATS v1.0 during upgrade.

## Summary storage for ATS

 Currently org.apache.hadoop.yarn.server.timeline.LeveldbTimelineStore class is used for ATS summary storage.

The most significant YARN ATS 1.5 change is the new EntityGroupFSTimelineStore summary storage. The best practices is to set yarn.timeline-service.store-class = org.apache.hadoop.yarn.server.timeline.EntityGroupFSTimelineStore in yarn-site.xml.

## Maximum number of retires for timeline service client

 The maximum number of retires for Timeline Service client is 30 that is usually too conservative.

It is reasonable to decrease the max number of retires in the stable clusters and set yarn.timeline-service.client.max-retries = 3 in the yarn-site.xml.

## Client policy for timeline operations

Currently failure to obtain a delegation token is considered as application failure; in case of ATS outage there are no applications can be submitted.

It is a good idea to allow jobs can continue to get submitted even if the ATS is down by setting yarn.timeline-service.client.best-effort = true in yarn-site.xml.

## Conducting YARN configuration changes in batches

**Batch 1:**

* Amount of physical memory allocated for containers

No downtime required; the change can be rolling. However, it is a major change and must be tested carefully.

**Batch 2:**

* YARN Secure Containers

Downtime is required; the change cannot be rolling. It is a major change and must be tested carefully.

**Batch 3:**

* YARN CGroups Isolation

Downtime is required; the change cannot be rolling. It is a major change and must be tested carefully.

**Batch 4:**

* Number of CPU cores that can be allocated for containers
* The maximum vcores allocation for every container request at the RM
* The maximum memory allocation for every container request at the RM

No downtime required; the change can be rolling.

**Batch 5:**

* The ResourceCalculator implementation in the Capacity Scheduler
* Capacity Scheduler container preemption

No downtime required; the change can be rolling.

**Batch 6:**

* YARN NodeManager health check script

No downtime required; the change can be rolling.

**Batch 7:**

* YARN Timeline Server 1.5
* Summary storage for ATS
* Maximum number of retires for timeline service client
* Client policy for timeline operations

Downtime is required; the change cannot be rolling.

# MapReduce2 observations

## MapReduce Map Memory Settings

Currently we have configured 4096 MB to request from the scheduler for each map task and 3276 MB MapReduce Map Java heap size, i.e. 820 MB is the maximum memory allocated in the container for all Java libraries associated with a MapReduce job.

I most cases it is sufficient, but it is a good idea to follow conservative approach and reserve 1536 MB (1.5 GB) for the Java libraries. In this case the configuration can be changed to:

* mapreduce.map.memory.mb = 5125 MB
* mapreduce.map.java.opts = -Xmx3584mb

## MapReduce Reduce Memory Settings

Currently we have configured 4096 MB to request from the scheduler for each reduce task and 3276 MB MapReduce Reduce Java heap size, i.e. 820 MB is the maximum memory allocated in the container for all Java libraries associated with a MapReduce job.

I most cases it is sufficient, but it is a good idea to follow conservative approach and reserve 1536 MB (1.5 GB) for the Java libraries. In this case the configuration can be changed to:

* mapreduce.reduce.memory.mb = 5125 MB
* mapreduce.reduce.java.opts = -Xmx3584mb

## MapReduce Sort Allocation Memory

Currently we have 1630 MB allocated for the buffer memory to use while sorting files that is 50% of the current MapReduce Map Java heap size (3276 MB).

It is a good idea to allocate between 30-50%, i.e. to increase mapreduce.task.io.sort.mb to 1792 MB.

## MapReduce History Server heap size

Currently we have 8GB heap size configured for the MapReduce History Server.

It is a good idea to decrease the heap size to lower values. 2048 MB could be a good value to test.

## Conducting MapReduce2 configuration changes in batches

All configuration changes can be done in one batch:

* MapReduce Map Memory Settings
* MapReduce Reduce Memory Settings
* MapReduce Sort Allocation Memory
* MapReduce History Server heap size

# HBase

There are 9 changes recommended for HBase are proposed as part of the health check.

## Maximum Region File size

Currently the Maximum Region File size is set to 10 GB, that cause creating 271,936 regions across 340 Region Servers (average 800 regions per Region Servers).

It is a good idea to increase the Maximum Region File size to 20 GB by setting hbase.hregion.max.filesize = 21474836480 in the hbase-site.xml. This value is more relevant to the best practices to keep not more than 300-500 regions per a Region Server.

## Online Region Merge

Currently there are average 800 regions per Region Servers and it could be a good idea to consider merging the adjacent regions (where the end key of a region is the start key of the other region).

The merging can be done by the HBase shell merge\_region command.

Note: this feature is not considered stable yet. We need to follow up with Technical Support before starting merging the regions.

## Managed Compactions

Currently major compactions are managed by HBase with default hbase.hregion.majorcompaction value of 7 days. It could trigger major compaction in peak hours or other non-desirable time.

It is a good idea to turn off automatic major compactions set hbase.hregion.majorcompaction = 0 in the hbase-site.xml.

Note: major compactions are absolutely necessary for StoreFile cleanup, the only variant is when they occur. They can be administered through the HBase shell and cron.

## Sync all data and metadata of a block to disk on the block closure

Currently data is not synced to disk on the block closure that potentially can cause HBase data loss when a cluster suffers a power loss.

It is a good idea to make the DataNodes flush and sync all data and metadata of a HDFS block file to disk when the block is closed by setting dfs.datanode.synconclose = true in the hdfs-site.xml. See HDFS-1539 for details.

## Write HDFS data to disk immediately after it is written

Default OS policy is to wait up to 30 seconds before triggering writeback, that may be not the best policy for HBase with heavy writes.

It is a good idea in HBase clusters with a lot of writes to instruct the operating system to enqueue all written data to the disk immediately after it is written by setting dfs.datanode.sync.behind.writes = true in the hdfs-site.xml.

## Nagle's algorithm for the TCP socket connection on the client

Currently small outgoing messages are buffered until full packet and sent all at once. This algorithm has negative effect on larger writes by imposing latency due to the ACK waiting for the previous packets.

It is a good idea to disable Nagle's algorithm in HBase clusters by setting ipc.client.tcpnodelay = true in the core-site.xml.

## Block pinning for HFile HDFS blocks

HDFS block pinning is not enabled. It causes HFile HDFS blocks movement during HDFS balancing and significantly impacts HBase RegionServer data locality.

It could be a good idea to implement HDFS block pinning for HBase objects by setting dfs.datanode.block-pinning.enabled = true in the HDFS configuration and specify a list of favorable datanodes in the file creation in the DistributedFileSystem API. It will allow to keep a block replica to be pinned to a favorable datanode and not moved during HDFS balancing. See HDFS-6133 for details.

Note: this feature is not considered stable yet. We need to follow up with Technical Support before enabling it.

## HDFS favored-nodes feature in HBase balancer

Currently the HDFS favored-nodes feature is not used by HBase.

It could be a good idea to enable the HDFS favored-nodes feature by setting the HBase balancer class hbase.master.loadbalancer.class = org.apache.hadoop.hbase.favored.FavoredNodeLoadBalancer.

Note: this feature is not considered stable yet. We need to follow up with Technical Support before enabling it.

## Parallel Garbage Collection of the Eden Space

Currently the Eden Space (the pool from which memory is initially allocated for most objects) does not use parallel copying collector.

It is a good idea to enable parallel copying collector by adding -XX:+UseParNewGC to the Region Server start option in the hbase-env.sh.

## Free memory allocation watermark

Currently the free memory allocation watermark is 90112, that is too low for HBase Region Servers.

It is a good idea to increase the free memory allocation watermark to at least 1 GB in order to allow memory allocations for HBase without waiting to reclaim memory by setting vm.min\_free\_kbytes = 1048576.

## Conducting HBase configuration changes in batches

Due to importance of HBase stability we should carefully conduct configuration changes on the HBase side and test them.

**Batch 1:**

* Maximum Region File size

Downtime is required; the change cannot be rolling.

**Batch 2:**

* Managed Compactions

Downtime is required; the change cannot be rolling.

**Batch 3:**

* Sync all data and metadata of a block to disk on the block closure
* Write HDFS data to disk immediately after it is written
* Nagle's algorithm for the TCP socket connection on the client

No downtime required; the change can be rolling.

**Batch 4:**

* Block pinning for HFile HDFS blocks

This configuration change should be confirmed with Technical Support.

**Batch 5:**

* HDFS favored-nodes feature in HBase balancer

This configuration change should be confirmed with Technical Support.

**Batch 6:**

* Parallel Garbage Collection of the Eden Space

No downtime required; the change can be rolling.

**Batch 7:**

* Free memory allocation watermark

No downtime required; the change can be rolling.

# Hive

## Hive Tez container size

We have hive.tez.container.size set to 5,120 MB, but hive.tez.java.opts max heap size is configured to 6,964MB, that can cause task failures due to OOM errors.

to decrease max heap size to 4,096 MB by setting hive.tez.java.opts = -server -Xmx**4096**m -Djava.net.preferIPv4Stack=true -XX:NewRatio=8 -XX:+UseNUMA -XX:+UseParallelGC -XX:+PrintGCDetails -verbose:gc -XX:+PrintGCTimeStamps in the hive-site.xml

## Hive Metastore HA

Currently Hive connects to only one URI to make metadata requests to a remote metastore (MySQL database).

The best practice is to have at least one metastore by adding additional Hive Metastore for HA purposes and updating hive.metastore.uris Hive property.

## HiveServer2 HA

Currently there is only HiveServer2 process runs in the production cluster.

The best practice is to have minimum two HiveServer2 instances for HA purposes by adding additional HS2.

## Zookeeper Discovery for HiveServer2 HA

Currently there is no practical need to enable Zookeeper Discovery for HiveServer2, but after adding mode HS2 instances it could beneficial.

The best practice is to configure Dynamic Service Discovery through ZooKeeper.

## Group by in the mapper byBucketizedHiveInputFormat

Currently group by in the mapper by using BucketizedHiveInputFormat is disabled.

If the bucketing/sorting properties of the table exactly match the grouping key, whether to perform the group by in the mapper by using BucketizedHiveInputFormat. It is a good idea to enable this feature by setting hive.map.groupby.sorted = true in the hive-site.xml.

## Sort-merge bucketed map-side join enforcement

Currently if the user asked for sort-merge bucketed map-side join, and it cannot be performed, the query fails.

It is a good idea to prevent queries from failing if the user requests a sort-merge bucketed map-side join that cannot be performed. It can be done by setting hive.enforce.sortmergebucketmapjoin = false in the hive-site.xml.

## HiveServer2 to close operation after not beingaccessed for this duration of time

Currently closing operations not accessed for defined duration of time is disabled in HiveServer2.

It is a good idea to configure checking for operations in terminal state only (FINISHED, CANCELED, CLOSED, ERROR) and closing them after 24 hours (1000\*60\*60\*24) by setting hive.server2.idle.operation.timeout = 86400000 in the hive-site.xml.

## Check interval for session/operation timeout in HiveServer2

Currently there are no checks for session/operation timeout in HiveServer2.

It is a good idea to configure checking for session/operation timeouts to 6 hours (1000\*60\*60\*6) by setting hive.server2.session.check.interval = 21600000 in the hive-site.xml.

## Maximum number of retries on stats publisher/aggregator

Currently there are no retries on failure when stats publisher/aggregator gets an exception updating intermediate database.

The best practices is to configure at least one retry in case of connection failure by setting hive.stats.retries.max = 1 in the hive-site.xml. See HIVE-2127 for details.

## Support sub-directories for Hive tables/partitions

Currently Hive does not support sub-directories for tables/partitions for MapReduce that disables many Hive optimizations.

It is a good idea to enable MapReduce support of sub-directories for Hive tables/partitions by setting hive.mapred.supports.subdirectories = true in the hive-site.xml. See MAPREDUCE-1501 for details.

## ORC splits to include metadata about the stripes in the file

Currently there are no metadata about the stripes are written to the ORC files, that prevents Hive clients or HiveServer2 from reading it remotely and sending to the tasks.

It is a good idea to include metadata about the stripes in the ORC file by setting hive.orc.splits.include.file.footer = true in the hive-site.xml.

## Merge small files at the end of a map-reduce job

Currently small files at the end of a map-reduce job are not merged, that leads to possible creating large number of small files.

It is a good idea to configure merging small files at the end of a map-reduce job by setting hive.merge.mapredfiles = true in the hive-site.xml.

## Table directories default permissions

Currently table directories are created with permissions derived from dfs umask, that could be potential access control issue since we are not using Ranger access controls.

It is a good idea to let table directories inherit the permissions of the warehouse or database directory by setting hive.warehouse.subdir.inherit.perms = true in the hive-site.xml.

## Compress Hive intermediate files

Currently there is no compression for intermediate files, that cause reading, writing and transferring over the network not compressed intermediate files.

The best practice is to compress intermediate files by setting hive.exec.compress.intermediate = true in the hive-site.xml.

## Data size per reducer

Currently the data size per reducer is 64 GB that could trigger launching large number of reducers and extra overhead on the cluster resources.

It is a good idea to increase the data size per reducer by setting hive.exec.reducers.bytes.per.reducer = 134217728 in the hive-site.xml.

## Column Position Alias in GROUP BY and ORDER BY clauses

Currently Column Position Alias in GROUP BY and ORDER BY clauses of queries is disabled.

It is a good idea to enable the Column Position Alias by setting hive.groupby.orderby.position.alias = true in the hive-site.xml.

## Tracking table and keys used in joins and groupby’s

Currently join and group by keys on tables are not derived and maintained in the QueryPlan. It complicates identification how tables are accessed and if they should be bucketed. It is a good idea to enable this feature by setting hive.stats.collect.tablekeys = true in the hive-site.xml. See HIVE-3501 for details.

## Tracking column access in the QueryPlan

Currently column accesses are not tracked in the QueryPlan. I complicates identification how tables are accessed and if there are wasted columns that can be trimmed.

It is a good idea to enable column accesses tracking in the QueryPlan by setting hive.stats.collect.scancols = true in the hive-site.xml.

## Aggregations with no group-by in a single reducer

Currently there is no certain aggregation queries delegation the final aggregation stage to a fetch task, which increases the query time.

It is a good idea to execute aggregation queries with no group-by clause (example: *select count(1) from table*) final aggregations in a single reduce task by setting hive.fetch.task.aggr = true in the hive-site.xml.

## Restrict certain risky Hive queries

Currently any query (including abusive in terms of resources) can be run by an inexperienced analyst.

It is a good idea to restrict execution of some risky queries by setting hive.mapred.mode nonstrict = strict in the hive-site.xml.

## Maximum number of dynamic partitions

Currently the maximum number of dynamic partitions is 5000 that may not be sufficient for large DML operations.

It is a good idea to configure maximum number of dynamic partitions allowed to be created by setting hive.exec.max.dynamic.partitions = 16384 in the hive-site.xml.

## Maximum number of dynamic partitions on a node

Currently the maximum number of dynamic partitions allowed to be created in each mapper/reducer node is 2000 that may not be sufficient for large DML operations.

It is a good idea to configure maximum number of dynamic partitions allowed to be created in each mapper/reducer node by setting hive.exec.max.dynamic.partitions.pernode = 4000 in the hive-site.xml.

## Maximum number of HDFS files created by a MapReduce job

Currently the maximum number of HDFS files created by all mappers/reducers in a MapReduce job is 10000 that may not be sufficient for large DML operations.

It is a good idea to increase the maximum number of HDFS files created by all mappers/reducers by setting hive.exec.max.created.files = 32768 in the hive-site.xml.

## Conducting Hive configuration changes in batches

The most configuration changes do not require downtime and can be done in rolling mode. However, the Hive changes impact on query exection and it is advisable to make one change at a time and test the change.

**Batch 1:**

* Hive Tez container size

No downtime required; the change can be rolling.

**Batch 2:**

* Hive Metastore HA

No downtime required; the change can be rolling.

**Batch 3:**

* HiveServer2 HA

No downtime required; the change can be rolling.

**Batch 4:**

* Zookeeper Discovery for HiveServer2 HA

Downtime is required; the change cannot be rolling.

**Batch 5:**

* Group by in the mapper by BucketizedHiveInputFormat

No downtime required; the change can be rolling.

**Batch 6:**

* Sort-merge bucketed map-side join enforcement

No downtime required; the change can be rolling.

**Batch 7:**

* HiveServer2 to close operation after not being accessed for this duration of time

No downtime required; the change can be rolling.

**Batch 8:**

* Check interval for session/operation timeout in HiveServer2

No downtime required; the change can be rolling.

**Batch 9:**

* Maximum number of retries on stats publisher/aggregator

No downtime required; the change can be rolling.

**Batch 10:**

* Support sub-directories for Hive tables/partitions

No downtime required; the change can be rolling.

**Batch 11:**

* ORC splits to include metadata about the stripes in the file

No downtime required; the change can be rolling.

**Batch 12:**

* Merge small files at the end of a map-reduce job
* No downtime required; the change can be rolling.

**Batch 13:**

* Table directories default permissions
* No downtime required; the change can be rolling.

**Batch 14:**

Compress Hive intermediate files

* No downtime required; the change can be rolling.

**Batch 15:**

Data size per reducer

* No downtime required; the change can be rolling.

**Batch 16:**

* Column Position Alias in GROUP BY and ORDER BY clauses
* Tracking table and keys used in joins and group by’s
* Tracking column access in the QueryPlan

No downtime required; the change can be rolling.

**Batch 17:**

* Aggregations with no group-by in a single reducer
* No downtime required; the change can be rolling.

**Batch 18:**

Restrict certain risky Hive queries

* No downtime required; the change can be rolling.

**Batch 19:**

* Maximum number of dynamic partitions
* Maximum number of dynamic partitions on a node
* Maximum number of HDFS files created by a MapReduce job

No downtime required; the change can be rolling.

# Tez observations

## The maximum heartbeat interval between the AM and RM

Currently the maximum heartbeat interval between the Application Master and ResourceManager is set to 250 milliseconds.

Increasing this setting reduces the communication between the AM and the RM that reduces load on the ResourceManager on large clusters. Best practice is to set tez.am.am-rm.heartbeat.interval-ms.max = 1000 (1 second) in the tez-site.xml.

## Number of allowed counters for the executing DAG

Currently number of allowed counters for the executing DAG is set 2000 that could cause “Too many counters” error on large jobs.

It is a good idea to increase max number of counters to 5000 by setting tez.counters.max = 4096 in the tez-site.xml. See AMBARI-14215 for details.

## Class to use for logging history data

Currently the class to use for logging history data is set to is set to ATSHistoryLoggingService

If we decide to change ATS to v1.5 we need to use ATSV15HistoryLoggingService as the history data class by setting tez.history.logging.service.class = org.apache.tez.dag.history.logging.ats.ATSV15HistoryLoggingService in the tez-site.xml.

## Slow-start behavior

in case of a ScatterGather connection once this fraction of source tasks have completed, all tasks on the current vertex can be scheduled. The total number of mappers that have to finish, where it starts to decide and run reducers in the nest stage is determined by the following parameters:

* tez.shuffle-vertex-manager.min-src-fraction (set to 0.2)
* tez.shuffle-vertex-manager.max-src-fraction (set 0.4)

Setting the properties to 0.2 and 0.4 accordingly could start reduce tasks earlier and waste resources; good to start with the defaults (0.25f for min and 0.75f for max) and tune it later if needed in the tez-site.xml:

* tez.shuffle-vertex-manager.min-src-fraction = 0.25
* tez.shuffle-vertex-manager.max-src-fraction = 0.75.

## Generating counters per I/O

Currently generating counters per I/O is enabled, i.e. CounterGroups/CounterNames are unique per Vertex + Src|Destination instead of unique per vertex.

This behavior is useful for debugging/tuning and in normal circumstance can be disabled by setting tez.task.generate.counters.per.io = false in the tez-site.xml.

## Conducting Spark configuration changes in batches

Tez configuration changes are the client side change and can be done in any order without restarting services (unless Ambari requires it). The changes below can be done in one batch:

* The maximum heartbeat interval between the AM and RM
* Number of allowed counters for the executing DAG
* Class to use for logging history data
* Slow-start behavior
* Generating counters per I/O

# Spark observations

## Spark version

Current version of Spark is 1.6.0 and considered to be old, but still maintained by the Apache Spark community:

|  |  |  |
| --- | --- | --- |
| **Version** | **Latest version** | **Release date** |
| 0.5 | 0.5.1 | 2012-10-07 |
| 0.6 | 0.6.2 | 2013-02-07 |
| 0.7 | 0.7.3 | 2013-07-16 |
| 0.8 | 0.8.1 | 2013-12-19 |
| 0.9 | 0.9.2 | 2014-07-23 |
| 1.0 | 1.0.2 | 2014-08-05 |
| 1.1 | 1.1.1 | 2014-11-26 |
| 1.2 | 1.2.2 | 2015-04-17 |
| 1.3 | 1.3.1 | 2015-04-17 |
| 1.4 | 1.4.1 | 2015-07-15 |
| 1.5 | 1.5.2 | 2015-11-09 |
| 1.6 | 1.6.3 | 2016-11-07 |
| 2.0 | 2.0.2 | 2016-11-14 |
| 2.1 | 2.1.1 | 2017-05-02 |

Consider upgrade Spark to v2.1.1 since it is a major update with new features introduced.

## Memory to use for the YARN Application Master in client mode

Amount of memory to use for the YARN Application Master in client mode is set to 512 MB.

It could be a good idea to set spark.yarn.am.memory = 1GB in the spark-defaults.conf since the current value may not be sufficient for some applications.

## Amount of memory per executor process

Currently amount of memory to use per executor process is set to 1 GB that could be not sufficient for large jobs.

It is a good idea to increase memory for executor processes by setting spark.executor.memory = 16 GB in the spark-defaults.conf.

## Max number of executors

Currently we have dynamic allocation is enabled and max number of executors is set to 10 that could be not sufficient for large jobs.

It is a good idea to increase the upper limit of executors by changing spark.dynamicAllocation.maxExecutors = 100 in the spark-defaults.conf.

## JobConf caching in the HadoopRDD

JobConf caching in the HadoopRDD could impact GC on a large jobs (1000s of RDD) due to caching too many configuration objects.

The best practice is to disable JobConf caching in the HadoopRDD by spark.hadoop.cacheConf = false in the spark-defaults.conf. See AMBARI-15521 for details.

## Logs the effective SparkConf as INFO

Currently we do not log the effective SparkConf as INFO when a SparkContext is started. The information configuration information is light in terms of size and can be useful in troubleshooting and performance tuning.

The best practice to enable the effective SparkConf logging when a SparkContext is started by setting spark.logConf = true in the spark-defaults.conf.

## Conducting Spark configuration changes in batches

Spark configuration changes are the client side change and can be done in any order without restarting services (unless Ambari requires it). The changes below can be done in one batch:

* Memory to use for the YARN Application Master in client mode
* Amount of memory per executor process
* Max number of executors
* JobConf caching in the HadoopRDD
* Logs the effective SparkConf as INFO

# Kafka

## Ambari Management for Kafka cluster

currently there is a dedicated Zookeeper quorum is used for Kafka and Ambari does not manage the quorum.

Create a dedicated Ambari Server for the Kafka cluster. It will allow us to decouple Kafka from the main Hadoop environment that simplifies management, security and upgrades of both Kafka and Hadoop clusters.

## Kafka topics run as user root

Currently Kafka topics run as user root.

We should avoid running any service under privileged accounts unless it is absolutely necessary. The best practice is to run Kafka topics under dedicated non-privileged accounts.

## Some topics have only one replica

Currently all nodes of the Kafka cluster have BOD disk configuration and  some topics have only one replica. In this case a disk failure leads to data loss.

The best practice is to have minimum 2 replicas.

# Flume

## Flume version

Current version of Flume is 2.2.6.0-2800.

It is advisable to upgrade Flume at least to version of the rest of components.

# Revision History

| Author | Date | Version | Revision Description |
| --- | --- | --- | --- |
|  |  |  |  |