IntelliPaat

Hadoop Administration (2)

Agenda

In this session we will cover :

- Administration and Maintenance
  - Namenode/Datanode directory structures and files
  - File system image and Edit log
  - Namenode High Availability
  - Adding and removing nodes
  - Hadoop File system Recovery

Agenda

- Monitoring and Troubleshooting
  - Using logs and stack traces for monitoring and troubleshooting
  - Using open-source tools to monitor the cluster

- Job Scheduler: Map reduce job submission flow
  - How to schedule Jobs on the cluster
  - Capacity Schedule, Fair Scheduler and its configuration

Administration and Maintenance

File system image and Edit log

- fsimage – fsimage file contains the complete state of the file system at a point in time. Every file system modification is assigned a unique, monotonically increasing transaction ID. An fsimage file represents the file system state after all modifications up to a specific transaction ID.

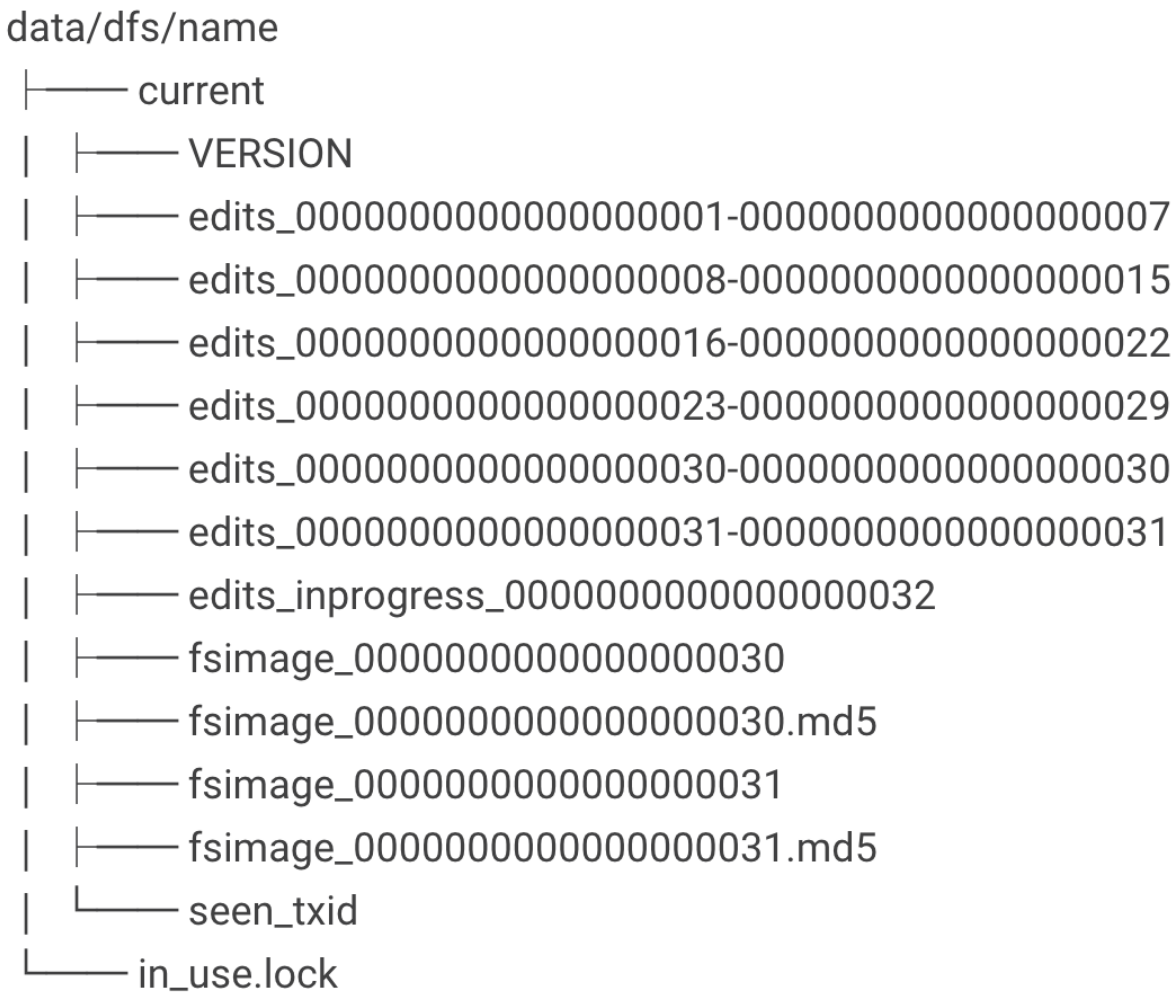
- edits – edits file is a log that lists each file system change (file creation, deletion or modification) that was made after the most recent fsimage.

- Checkpointing is the process of merging the content of the most recent fsimage with all edits applied after that fsimage is merged in order to create a new fsimage. Checkpointing is triggered automatically by configuration policies or manually by HDFS administration commands.

Namenode directories and files

```
[training@hadoop intellipaat]$ ls /var/lib/hadoop-hdfs/cache/hdfs/dfs/name/
current  in_use.lock
[training@hadoop intellipaat]$ ls /var/lib/hadoop-hdfs/cache/hdfs/dfs/name/current/VE
RSION
/var/lib/hadoop-hdfs/cache/hdfs/dfs/name/current/VERSION
[training@hadoop intellipaat]$ ls /var/lib/hadoop-hdfs/cache/hdfs/dfs/name/current/ed
its_^C
[training@hadoop intellipaat]$ ls /var/lib/hadoop-hdfs/cache/hdfs/dfs/name/current/fs
image_0000000000000000137^C
[training@hadoop intellipaat]$ ls /var/lib/hadoop-hdfs/cache/hdfs/dfs/name/current/se
en_txid
/var/lib/hadoop-hdfs/cache/hdfs/dfs/name/current/seen_txid
```

Namenode directories and files

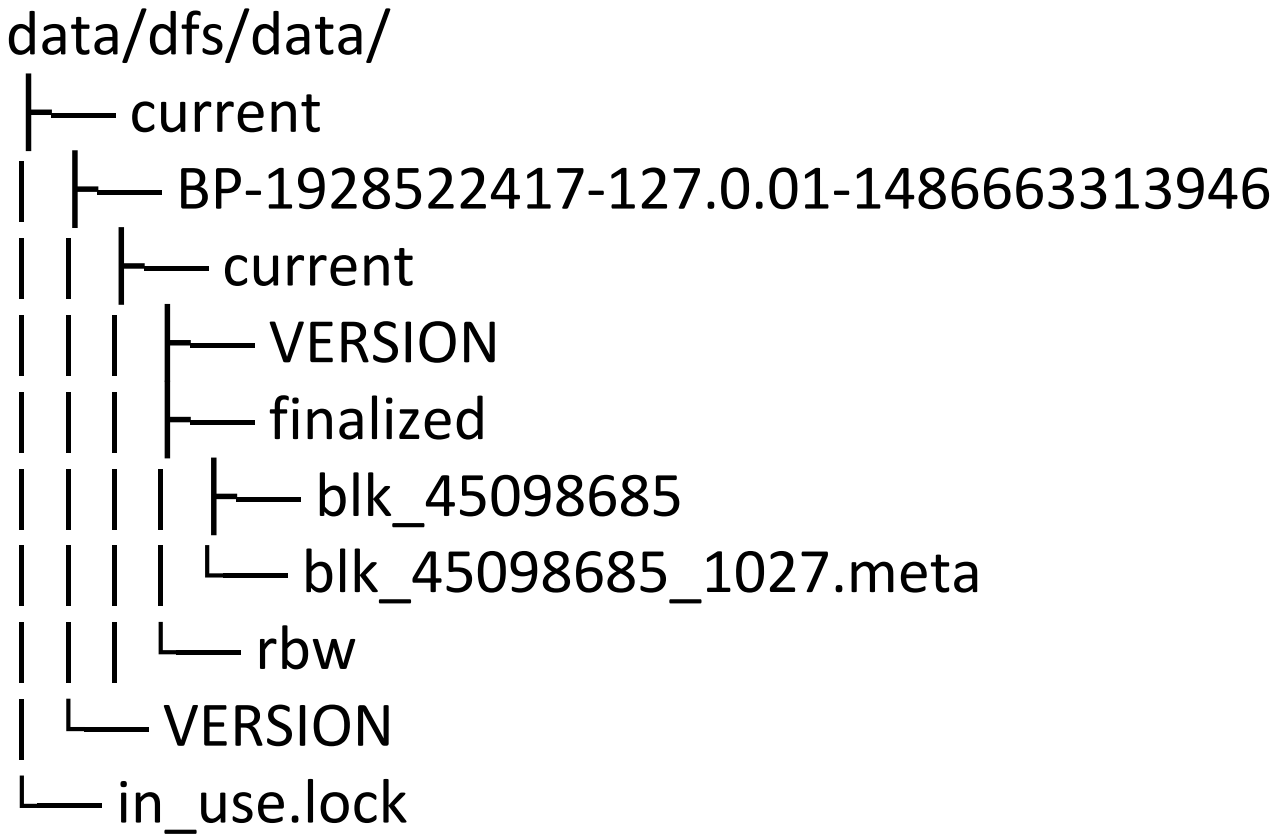


Namenode directories and files

- VERSION File

```
#Thu Feb 09 13:01:53 EST 2017
namespaceID=1453867033
clusterID=CID-640b2bb2-f746-4cc9-a7de-ee4d74786331
cTime=0
storageType=NAME_NODE
blockpoolID=BP-1928522417-127.0.0.1-1486663313946
layoutVersion=-40
```

Datanode directories and files



Datanode directories and files

- VERSION

```
#Fri Feb 10 15:04:39 EST 2017
storageID=DS-1522345539-127.0.0.1-50010-1486663688108
clusterID=CID-640b2bb2-f746-4cc9-a7de-ee4d74786331
cTime=0
storageType=DATA_NODE
layoutVersion=-40
```

Namenode failure and Recovery

- Namenode is most important component in Hadoop, if it goes down cluster will be useless. Before Hadoop 2.0 this was single point of failure. But now it supports HA through secondary namenode.

- The NameNode can write its edit logs to multiple local directories.

    - dfs.namenode.name.dir - Give comma-delimited list of directories then the name table is replicated in all of the directories, for redundancy.

Namenode failure and Recovery

- Two NameNodes configured for HA - Active and Standby. Standby is acting as a slave, maintaining enough state to provide failover capability.
- For standby NameNode to be synchronized with active NameNode, both communicate with each other through Journal Node.
- Standby NameNode polls edit logs at journal nodes and updates itself.

Namenode failure and Recovery

- Configure dfs.namenode.shared.edits.dir in hdfs-site.xml
  dfs.namenode.shared.edits.dir - the location of the shared storage directory
- The URI should be in the form:
  qjournal://<host1:port1>;<host2:port2>;<host3:port3>/<journalId>

  For example, if the JournalNodes for this cluster were running on the machines node1.example.com, node2.example.com, and node3.example.com, and the nameservice ID were mycluster, you would use the following as the value for this setting (the default port for the JournalNode is 8485)

Removing nodes

- Create a file excludes on your local file system.
- Map this location to dfs.exclude (hdfs-site.xml) and mapred.exclude (mapred-site.xml)properties
- As **hdfs** user issue command, hadoop dfsadmin -refreshNodes
- As **mapred** user issue command, yarn rmadmin -refreshNodes (for YARN)
- As **mapred** user issue command, hadoop mradmin -refreshNodes (for V1)

Adding nodes

- Create a file includes on your local file system.
- Map this location to dfs.include (hdfs-site.xml) and mapred.include (mapred-site.xml)properties
- As **hdfs** user issue command, hadoop dfsadmin -refreshNodes
- As **mapred** user issue command, yarn rmadmin -refreshNodes (for YARN)
- As **mapred** user issue command, hadoop mradmin -refreshNodes (for V1)

Hadoop File system Recovery

- To recover corrupt files in HDFS.
  hadoop fsck /path/to/directory

- To recover corrupt metadata in Namenode
  hadoop namenode -recover

Monitoring and Troubleshooting

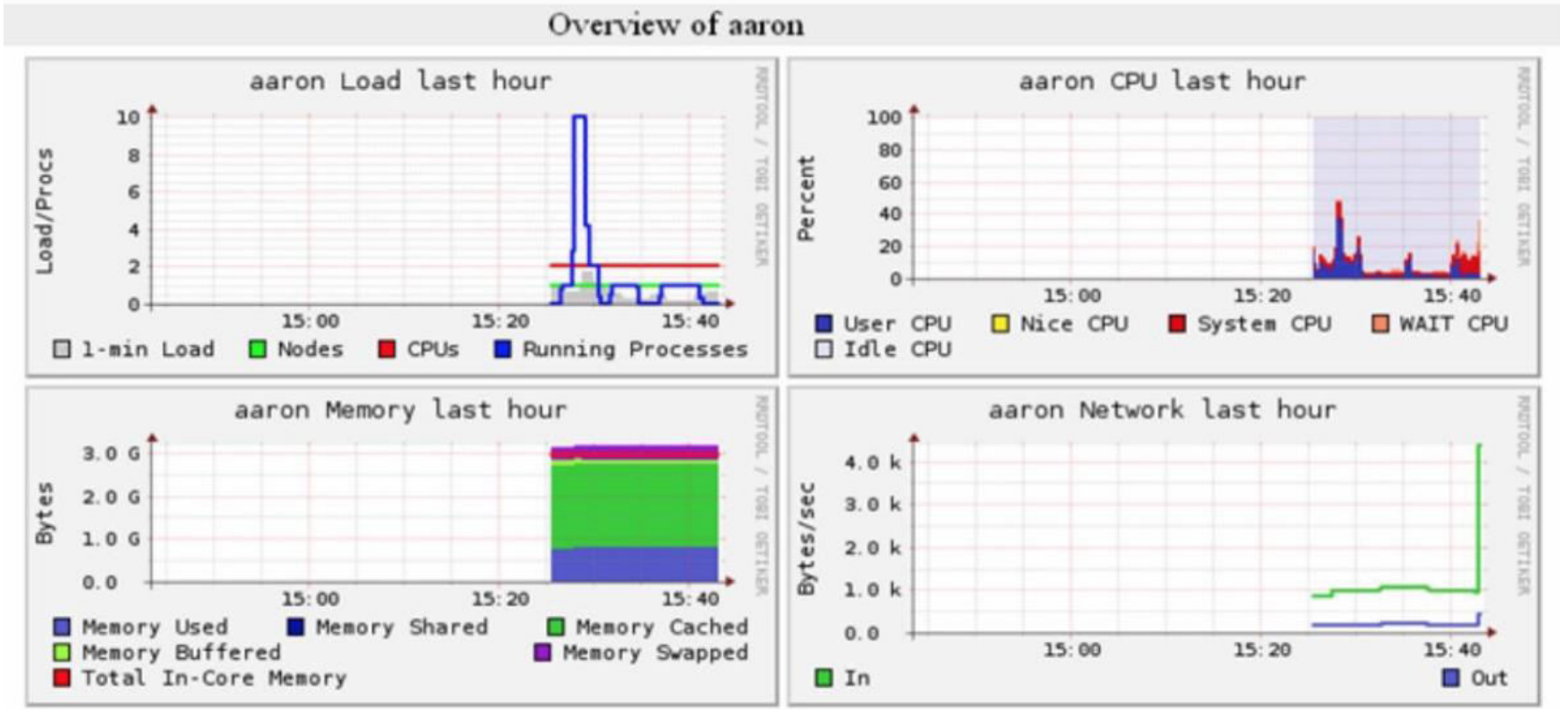
Using logs and stack traces

- Log files are located at /var/log/hadoop-*
  Hdfs
  Yarn
  Mapreduce

- Can change this location by setting HADOOP_LOG_DIR environment variable in hadoop-env.sh

Open-source tools to monitor

- Nagios for Alerts
- Ganglia for Metrics



Job Scheduler

Schedule Jobs on the cluster

- Users share a MapReduce cluster to have all the data in one place and to run queries instead of building a separate Hadoop cluster for each group.

- Sharing requires support from the Hadoop job scheduler to provide guaranteed capacity to production jobs and good response time to interactive jobs while allocating resources fairly between users

Capacity Schedule

- Capacity Scheduler provides concept of queues. These queues are setup by administrators.
- The Capacity Scheduler has a pre-defined queue called root. All other queues are children of the root queue.
- New queues can be setup by configuring yarn.scheduler.capacity.root.queues with a list of comma-separated child queues.
- The configuration for Capacity Scheduler uses a concept called queue path to configure the hierarchy of queues. The queue path is the full path of the queue hierarchy, starting at root, with . (dot) as the delimiter.

Capacity Schedule

Here is an example with three top-level child-queues a, b and c and some sub-queues for a and b:

```
<property>
 <name>yarn.scheduler.capacity.root.queues</name>
 <value>a,b,c</value>
 </description>
</property>

<property>
 <name>yarn.scheduler.capacity.root.a.queues</name>
 <value>a1,a2</value>
 </description>
</property>

<property>
 <name>yarn.scheduler.capacity.root.b.queues</name>
 <value>b1,b2,b3</value>
 </description>
</property>
```

Fair Scheduler

The Fair Scheduler is based on three concepts:

- Jobs are placed into named “pools” based on a configurable attribute such as user name, Unix group, etc ...
- Each pool can have a “guaranteed capacity” that is specified through a config file. When there are pending jobs in the pool, it gets at least this many slots, but if it has no jobs, the slots can be used by other pools.
- Shorter jobs will finish quickly, while longer jobs are guaranteed not to get starved.

Fair Scheduler

```
<property>
  <name>yarn.scheduler.fair.allow-undeclared-pools</name>
  <value>true</value>
 </property>
 <property>
  <name>yarn.scheduler.fair.user-as-default-queue</name>
  <value>true</value>
 </property>
 <property>
  <name>yarn.scheduler.fair.preemption</name>
  <value>true</value>
 </property>
```

Thank You