HDFS Node Labels

(HDFS-9411)

Problem statement

HDFS currently stores data blocks on different datanodes chosen by BlockPlacementPolicy. These Datanodes are randomly chosen within the required scope (i.e. local/rack-local/remote-rack/node-group/domain, etc) of network topology.

Some use-cases require, blocks to be stored on datanodes which have special characteristics or on special location. (Such as hardware, OS, network speed, processor, underlying filesystem, etc). HDFS doesn't have any mechanism to specify these special characteristics to consider while choosing datanode for block placement.

NodeLabels

YARN-796, introduced a concept of Node-labels to logically specify which nodes user wish to run their applications on. As part of node-labels in Yarn, there is also a provision to have partition labels to divide the yarn-cluster logically to different partitions.

Similarly, in HDFS there is a need of specifying the datanodes' special characteristics as Node-labels and these node-labels can be considered while choosing nodes to allocate the block.

HDFS already supports the Heterogeneous Storage by specifying the StorageType for the disk configured for datanode. This helps in improving read/write throughput to/from disk. This characteristic will not be part of Node-labels.

But there might be some other characteristics or attributes of datanode machine, which user wants it to be considered while choosing the datanode to store the block.

What is a Node Label?

- Node label is a non-tangible entity mapped with the datanode to either specify the special characteristics of the node or for the logical grouping. These labels don't carry any value.

Type of NodeLabel

- **PARTITION:** These are labels assigned by admin to each of datanode to logically divide the cluster.
- **ATTRIBUTE:** These are the labels which specify the datanode's special characteristics

Usecases

- *Multitenant cases:*
  Logical partition of cluster to provide independent nodes for each tenant. So independent throughput could be guaranteed.
  Ex: Each tenant may be assigned with one or more of the PARTITIONs to use the nodes from.
- *Performance cases:*

Sometime clients' also need processing power to be present in the same machine to write the data effectively.
Ex: **Erasure coding**. Erasure coding recovery of blocks will be faster if it's done on the powerful machines. So nodelabels can be beneficial in this case.

- *Data locality:*
  Currently data locality solution provided is based on the writer. Not on the processor of the data. There could be a use-case, where data is generated at one point and needs to be processed at another set of nodes. So, in this kind of cases, data locality will not be available for the processor nodes. This can be solved using the node labels.

High-level Requirements

- Label should be created in NameNode first by admin.
- Label-to-Node(s) mapping can be done using any of the following 3-ways or all ways.
  - **Centralized**:
    Mapping will be done using the dfsadmin command, an RPC to NameNode, which will then distribute the same to corresponding nodes using a *DatanodeCommand* via heartbeat response.
  - **Distributed**:
    Datanodes will have a *NodeLabelProvider* configured. Each Datanode will fetch the nodelabels from the NodeLabel provider for the node and sync with the namenode.
  - **Delegated-Centralized:**
    Node-to-labels mapping will be set by configured NodeLabelProvider in the Namenode. NodeLabelProvider will fetch mapping for each of Datanodes and updates accordingly.
- One node can have multiple admin specified **PARTITION** labels and ATTRIBUTE labels provided by Admin/provider
  - There should be a limit on total number of labels per node to ease the management.
- Labels should be persistent across restarts/failover and upgrades
- Admin can specify **List of PARTITION Labels** to store data under a specified path.
  - Ex: **/user/mapred** is allowed to use only **PART_A** or **PART_B** partitions.
- Users should be able to request to place the block on specific nodes by specifying the label expression on files/directories.
  - Label expression can be set on files.
  - Change in Label expression set on directories will be inherited for new blocks of new/old files. Does not reflect on already existing blocks of existing files.
  - Label expression can use only **allowed PARTITION** Labels, if any set on parent directory by admin
  - If Label expression doesn't explicitly use any **PARTITION** labels, by default union of allowed PARTITION labels will be **AND**ed with the specified label expression.
- StorageType (SSD/DISK/ARCHIVE) should not be part of nodelabel, should be specified explicitly via dfs.datanode.dirs

High Level Design

NodeLabel:

Each Node label will have a simple name. If any additional attributes to be stored along with name, can be added later.

```
NodeLabel {
  String name;
  Type; // PARTITION/ATTRIBUTE
}
```

Creation/modify/deletion of NodeLabels:

Admin can perform operations of create/delete/modify/list on NodeLabels. These operations will be directly communicating to Namenode, via RPC.

CLI

```
hdfs nodelabels -createNodeLabel -name <labelname> -type <type>
hdfs nodelabels -removeNodeLabel -name <labelName>
hdfs nodelabels -listNodeLabels [-node <datanode-name>] [-type <type>]
hdfs nodelabels -listNodesForLabel -name <labelName>
hdfs nodelabels -listAllowedLabelsForPath -path <path>
```

NodeLabel->Datanode(s) Mappping

- **Centralized mode**
  CLI

  hdfs nodelabels *-addDataNodeLabel -name <labelname> <dn_addess(es)>*

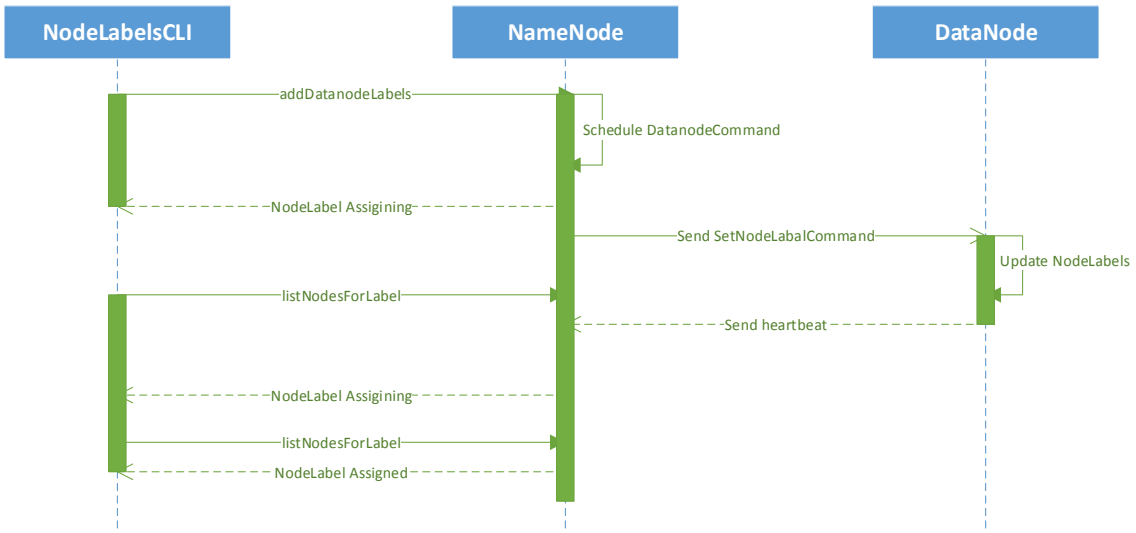
  hdfs nodelabels *-removeDataNodeLabel -name <labelName> <dn_addess(es)>*

  NodeLabel->DataNode mapping will be done by admin in centralized mode. This will be persisted in datanode's data dirs.

  This assignment happens in 2-way communication, because NameNode, by design, will not persist anything related to Datanodes in Namenode itself.

  a. Admin sends RPC to NameNode for assignment of Label to DataNode(s).
  b. NameNode issues a *DatanodeCommand* (SetNodeLabelCommand/RemoveNodeLabelCommand) to specified DataNode(s).
  c. DataNode receives the command, and assigns/removes the NodeLabels to itself, and persists in Datadir's VERSION file to read during restart.
  d. DataNode includes new NodeLabels in its registration and send it via next heartbeat to NameNode.

e. NameNode verifies the next heartbeat for the updates in NodeLabels, and once found DataNode will be updated to NodeLabel->DataNodes mapping. Until the NodeLabel is completely assigned/removed at DataNode, state of assignment can be monitored using *"hdfs nodelabels -listNodesForLabel -name <labelName>"*



- **Distributed mode**

  In Distributed mode, each Datanode will be configured with a ***NodeLabelProvider***. On startup/refresh Datanode will refetch the nodelabels and propogate to NameNode via heartbeat. NameNode will find the change in labels and verifies about the pre-created labels and schedules the DataNodeCommand based on changes and follows the 2-way assignment mentioned in earlier figure.

  In this case also, labels given by the NodeLabelProvider should be pre-created at the NameNode side.

- **Delegated-Centralized mode**

  In this mode, ***NodeLabelProvider*** will be configured at the Namenode itself, which fetches the Label->Datanode mapping on Datanode registrations and assigns to each datanode following the 2-way procedure followed in Centralized mode.

Note:

1. **All modes together**: Since all modes of Label mapping modes are possible at a time, Source of mapping for each label to datanode will be maintained. There could be overlaps, i.e. Already admin specified label could be again added by LabelProvider. These overlaps will be ignored.

i. Changes related to removal of labels should be in sync in providers as well. Ex: If admin removes the mapping via RPC, same should be removed from the provider by admin itself. Otherwise during restart/refresh label could be re-added.

ii. For the Mapping provided by admin RPC, if there are any overlaps with provider, removing the mapping only from provider, will not remove the mapping from node. Only admin-RPC can remove this mapping again. But vice-versa is possible. i.e. Admin-RPC can remove the label mapping provided by the provider immediately. Though refresh/restart can re-add the mapping, if provider still have the same mapping.

2. **Federation case**: Since labels should be pre-created in Namenode before using them, each datanode may face conflicts if its reporting to multiple nameservice. To avoid the conflicts, label mapping will be stored in blockpool specific VERSION file i.e. It's possible to have different labels per nameservice.
   i. Datanode-side label providers can be configured per nameservice.

Persisting of Labels and Label->Nodes mapping

- **Labels**
  - Node Labels will be persisted in the Namenode in edits/fsimage. Each operation related to NodeLabel directly such as create/delete etc are logged in edits as a transaction and later will be persisted in the fsimage.
- **Label(s)<->Node(s) mapping**
  - Label<->Datanode mapping will be persisted in the datanode itself upon registration/setNodeLabelCommand.
  - In Distributed Mode/Delegated-Centralized mode/on refresh/Startup, labels will be refetched and sent via register/heartbeat to be confirmed with the namenode. Once Namenode confirms and sends back setNodeLabelCommand, labels will be persisted in the Datanode's VERSION file.
  - All data dirs, should be having same set of Labels in VERSION file(s).

Allowed PARTITION labels on a path

Admin can specify list of PARTITION labels to be used for a path. All the files' blocks should be stored in any of the allowed PARTITION labels' nodes only.

- This is allowed only for directory
- List could be stored as an XAttr.

CLI

hdfs nodelabels –setAllowedPartitionLabels <listOfLabels> <path>

hdfs nodelabels –getAllowedPartitionLabels <path>

Label expression on file/directory

A label expression, containing simple operands ||, && and !(not) along with nodelabels can be set on a file/directory. This expression will be considered while choosing the target datanodes for the block. If the datanode satisfy this node label expression, then that datanode will be chosen as a target, otherwise datanode will be skipped.

Simple Label expression can be like below.

(LABEL_1 || LABEL_2) && !LABEL_3 [fallback=NONE]

In this case, nodes not having label *LABEL_3*, and having either of *LABEL_1* or *LABEL_2* as Node labels will be chosen.

[fallback=NONE] is optional here.

- **NONE**: Fallback option of NONE, represents that if not enough nodes are able to find using the specified label expression then no other measure taken, just return existing chosen nodes, if any. **NONE is default**, if fallback not specified explicitly.
- **GLOBAL**: This will allow to choose nodes without using label expression for remaining nodes, but within the specified allowed PARTITION, if any.

i. If any ALLOWED PARTITION labels set on a parent directory, then
   - If the user specified label expression contains PARTITION labels, then those should be subset of ALLOWED partition labels set on parent directory.
   - If the user specified label expression DOES NOT contain any PARTITION labels, i.e. contains only ATTRIBUTE labels then union of ALLOWED partition labels will be considered along with label expression to choose nodes.

ii. Label expression can be set on a directory/file.
   - Label expression set on a directory will be inherited for only new files created after setting the expression.
   - Any change in the label expression will only reflect for the further block allocations and replications. Not for already chosen/written blocks. Already chosen blocks should be moved using Mover/Balancer (or StoragePolicySatisfier in future).
   - During rename of file/directory, for which parent have the label expression, label expression of src parent will NOT be copied, destination parent's label expression will be enforced, if Mover/Balancer is used.

Removal of NodeLabels from Cluster:

1. During the deletion of NodeLabels from cluster, if datanodes are not available, then also removal will be successful. Once the datanode restarts and registers to NameNode, if NodeLabels which are not available in NameNode are reported from DataNode, NameNode will ask Datanodes to remove those Labels.

2. NodeLabels could be removed from cluster, even when they are referenced by LabelExpressions. So, there could be label-expressions with NodeLabels which doesn't exist after deletion of NodeLabels. For such files, NodeLabels based selection would ignore those labels.

CLI

hdfs nodelabels –setLabelExpression <expression> <path>

hdfs nodelabels –getLabelExpression <path>

hdfs nodelabels –clearLabelExpression <path>

BlockPlacement Policy support

- All existing BlockPlacement policies need to support selection of nodes based on Label Expression for the file.
- *BlockPlacementPolicy#chooseTarget(..)* need to be modified as below.

```
public DatanodeStorageInfo[] chooseTarget(String srcPath,
                                  int numOfReplicas,
                                  Node writer,
                                  List<DatanodeStorageInfo> chosenNodes,
                                  boolean returnChosenNodes,
                                  Set<Node> excludedNodes,
                                  long blocksize,
                                  final BlockStoragePolicy storagePolicy,
                                  EnumSet<AddBlockFlag> flags,
                                  LabelExpression labelExpression) {
```

- This change is also required for WebHDFS request redirections and re-constructions.

NodeLabel statistics

NodeLabel statistics such as below could be exposed per NodeLabel

- CapacityTotal // disk capacity for the NodeLabel
- CapacityUsed // disk usage for the NodeLabel
- TotalBlocks // Total number of blocks in NodeLabel.
- NumLiveDatanodes // # Live Datanodes with this NodeLabel
- NumDeadDatanodes // # Dead Datanodes with this NodeLabel
- NumDecommissionedNodes // # Decommissioned Nodes with this NodeLabel
- NumDecommissioningNodes // # Decommissioning Nodes with this NodeLabel

WebUI Improvements for NodeLabels.

- WebUI needs to be improved to show the information about NodeLabels

Mover and Balancer support for NodeLabels.

- Mover and Balancers should support movement of blocks from previous LabelExpression to new LabelExpression.

Design considerations for YARN and HDFS NodeLabels.

- YARN's NodeLabel feature is considered for the above design and most of the design (configuration modes,Types, etc) are similar to YARN. But have to use HDFS specific configuration names as YARN's configurations are already part of many releases.
- Code-wise common areas may be very little as of now, based on the development and need we revisit and refactor during the development.

Future scope

- StoragePolicySatisfier (SPS) could be integrated to support change in Label expression as well.
- Support more complex Label expressions.
  - Support different label expressions for different replicas of same block.

References

YARN-796: https://issues.apache.org/jira/secure/attachment/12662291/Node-labels-Requirements-Design-doc-V2.pdf