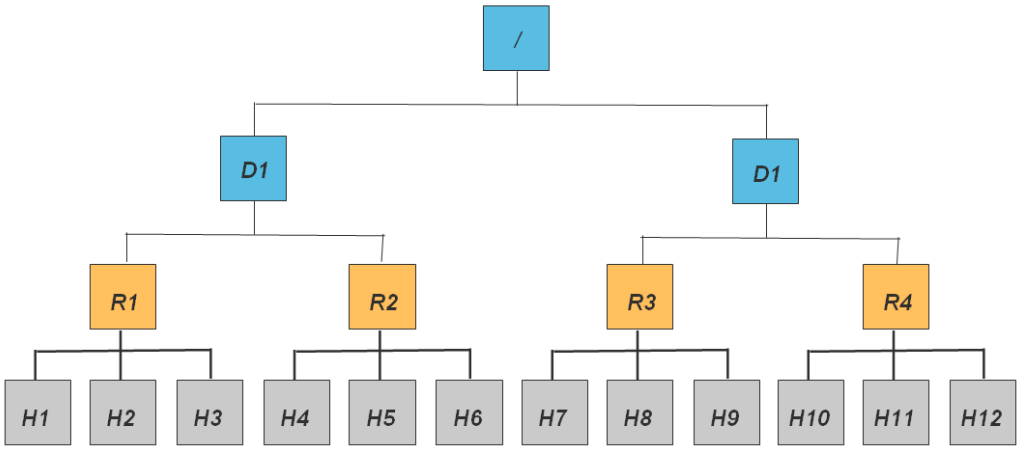
Hadoop Data Locality Change for Virtualization Environment

Problem Statement

The original proposal https://issues.apache.org/jira/browse/HADOOP-692 for hadoop rack awareness proposes a three-layer network topology as following:



And in most cases, the third layer of "data center" is not taken into consideration when placing replica, scheduling tasks, etc. And the user's topology script is required to provide rack info only.

This network topology is designed and work well for hadoop cluster running on physical server farms. However, for hadoop running on virtualized platform, we have additional "hypervisor" layer, and its characteristics include:

1. The communication price between VMs within the same hypervisor is lower than across hypervisor (physical host) which will have higher throughput, lower latency, and not generating physical network traffic.
2. VMs on the same physical box are mostly affected by the same hardware failure.

Due to above characteristics in performance and reliability, this layer is not transparent for hadoop. So we propose to induce an additional layer in hadoop network topology to reflect the characteristics on virtualized platform.

Principle

Our design is to change hadoop network topology related code so that hadoop can perform well in virtualization environment through some advanced configurations. However, these changes should be transparent to hadoop cluster running in physical server farms in both configuration and performing. Also, these changes are independent of specific virtualization platform, which means it can naturally apply to any virtualization platform like: vSphere, Xen, Hyper-V, KVM, etc.

Assumption

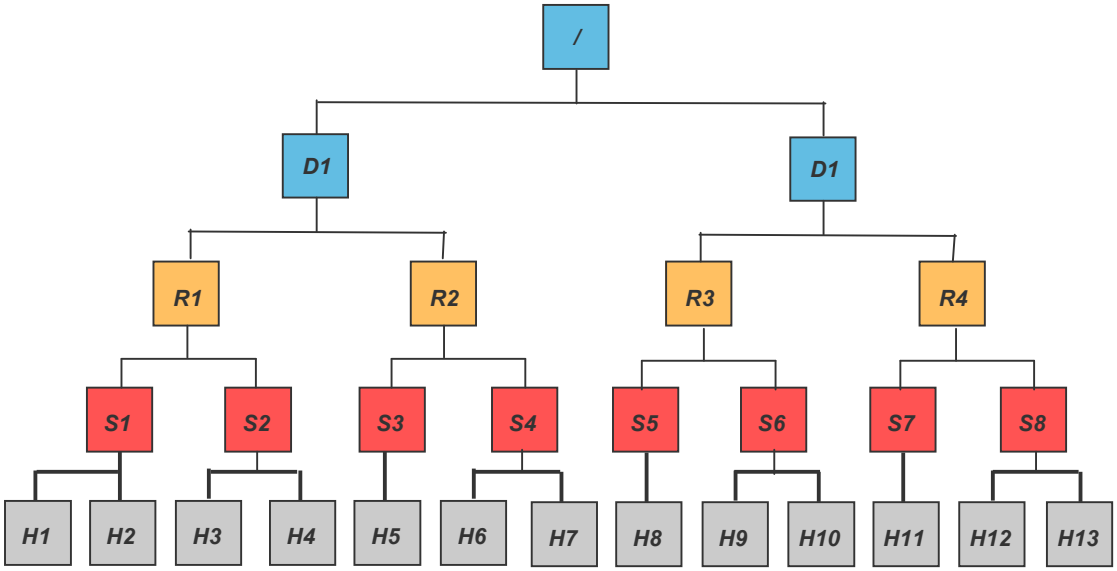
1. Physical network is still hierarchical: rack switch, data center switch, etc.
2. Rack-awareness is still important, for:
   To get rid of data loss when rack failure
   Optimize traffic flow (limit cross-rack traffic)
3. In most cases, communication between virtual machines that on the same physical host is faster than vms across hosts.

4. In general, Virtual Machines living on the same physical host have the same failure zone for hardware failure (although some hardware failures like nic may only affect some VMs).
5. On virtualized platform, the separation of data nodes and compute nodes can contribute to scale in and out the computing resource for hadoop cluster so that to enhance the elasticity of hadoop.

Data locality for hadoop in virtualization environment

N: Rack/Host → V: Rack/ServerGroup/Host

For efficiency and robust reasons for hadoop cluster running in virtualization environment, the hierarchy layers we want to build in hadoop is as following:



In high level, the proposed changes include:

Add new property in hadoop configuration to mark the cluster is running in virtualized environment. Then, the topology script for resolving host should include hypervisor info, i.e: input: 10.1.1.1, output: /rack1/servergroup1.

The replica placement policy in HDFS write flow mostly conform the original pipeline: local node of writer →off rack node of 1st replica → local rack node of 2nd replica. Some tiny changes are: 3rd replica will be off server group (not in the same physical host) of 2nd replica and if there is no local node available, the 1st replica will be local server group node.

The policy of choosing replica of block in HDFS read flow, the sequence is becoming: local-node, local-servergroup, local-rack, off-rack.

When ApplicationMaster negotiate resource with scheduler of RM, their protocol will become from *<priority, (host, rack, *), memory, #containers>* to *<priority, (host, **servergroup**, rack, *), memory, #containers>*. After receiving the resource request, RM scheduler will assign containers for requests in the sequence of data-local, servergroup-local, rack-local and off-switch. Then, ApplicationMaster schedule tasks on allocated container in sequence of data-local, servergroup-local, rack-local and off-switch.

The policy of choosing target and source node for balancing is following the sequence of local-servergroup, local-rack, off-rack.

The detail of changes

1. In configuration file core-site.xml, we add a property "topology.environment.type" to mark the hadoop cluster is running in virtualization environment.

2. In customer configured script (still specified in "topology.script.file.name") for rack awareness, it should provide the additional layer info of server group.

3. NetworkTopology.java is a key component of hadoop data locality and we will extend current NeworkTopology class to VirtualizationNetworkTopology by overriding some methods like pseudoSortByDistance(), adding some new APIs related to additional layer of servergroup and changing some properties' visibility to use them in subclass. Distance calculation between nodes for network topology is updated from: 0(local), 2 (rack-local), 4 (rack-off) to: 0(local), 2(servergroup-local), 4 (rack-local), 6 (rack-off). In initialization of Namenode, clusterMap within DatanodeManager is constructed as NetworkTopology or its subclass VirtualizationNetworkTopology according to the value of new adding configuration property of "topology.environment.type".

4. In BlockPlacementPolicy, we will have BlockPlacementPolicyVirtualization which extends the class of BlockPlacementPolicyDefault to aware of servergroup layer in choosing target. The Replica placement strategy on virtualization is almost the same as original one, and differences are: 3rd replica will be off server group of 2nd replica and if there is no local node available, the 1st replica will be local server group node. The new policy class will be created by BlockManager after user set the new class name in hdfs-site.xml configuration of "dfs.block.replicator.classname".

6. For yarn task scheduling, we will update class of ContainerRequestEvent so that application can request container on specific servergroups (current it can only be host, rack or *). In runtime container scheduling, we will update current RM scheduler FifoScheduler and CapacityScheduler. For FifoScheduler, the update is happened on method of assignContainers(); for CapacityScheduler, it is on assignContainersOnNode() in LeafQueue. Both changes are adding assignServerGroupLocalContainers() between data-local and rack-local.
For scheduling map tasks on allocated containers, we need to update assignToMap() method in RMContainerAllocator to assign map task in sequence of node-local, servergroup-local, rack-local and rack-off, and maintain a MapsServerGroupMapping besides existing MapsHostMapping and MapsRackMapping for tracking map tasks request on specific severgroup. To get servergroup info for a given host, we need to update RackResolver class accordingly also.

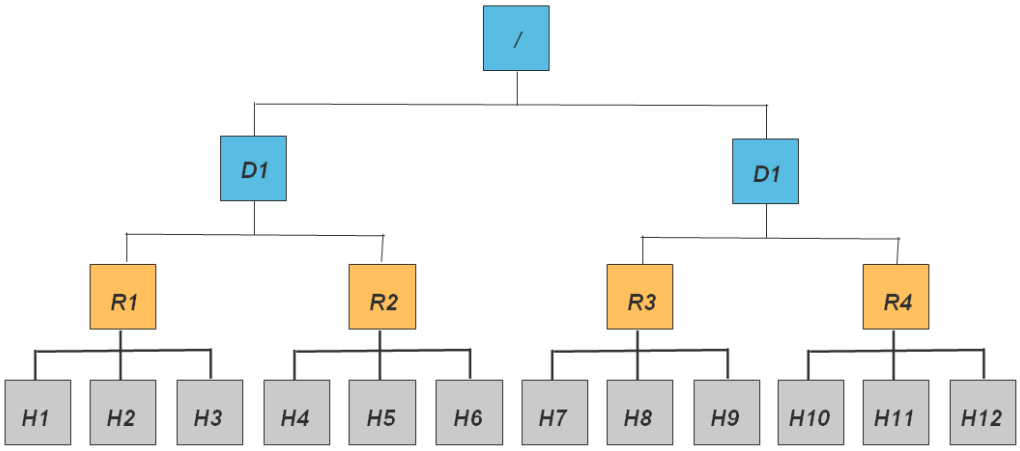
7. For HDFS balancer, we need to add some methods that can choose target and source node on the same server group for balancing. Then our balancing policy is, first doing balancing between nodes within the same servergroup then the same rack and off rack at last.

Reference

Appendix I – Data locality mechanism for hadoop in native environment

Two layers: Rack/Node

The original network topology proposal https://issues.apache.org/jira/browse/HADOOP-692 proposes three-layer network topology as following:



In short, data locality awareness in hadoop affects: ***replica placement in HDFS write flow, choosing replica of block in HDFS read flow, task scheduling*** by ApplicationMaster and ResourceManager, ***choosing target node for balancing***.

The detail mechanism is as follows:

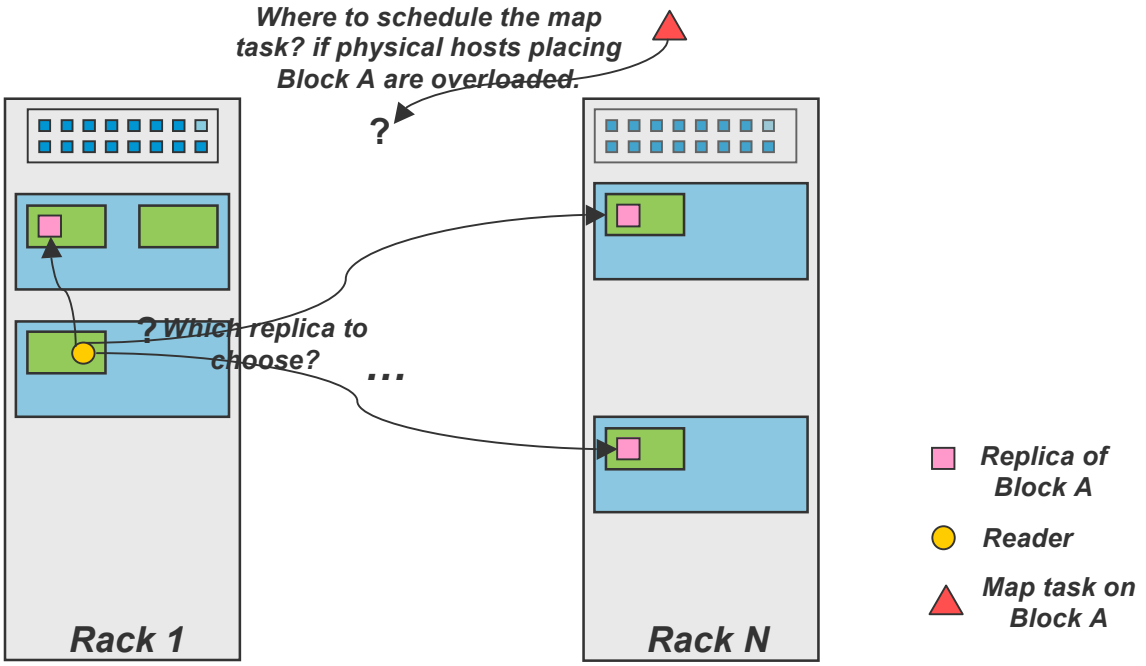
1. ***Customer input topology info***: a script (specified in "topology.script.file.name" of core-site.xml) describe the network topology mapping for each nodes. This script can return rack mappings for a list of IPs. ←→ rackname/nodename ←→ rackname r1/n1.
2. ***Network topology is setup*** when HDFS is started and updated when data node join/removed from cluster:
Based on the topology script, a tree structure of clusterMap is built (defined in NetworkTopology.java) during HDFS FSNamesystem initialization (with creation of BlockManager, DatanodeManager) and registration of new data nodes. Also, dnsToSwitchMapping is setup for resolving and caching network location for given nodes.
3. ***Replica placement in HDFS write flow***: When a client want to write blocks to HDFS, it can get a pipeline of nodes for each block from HDFS block management service (BlockManager)
With clusterMap (NetworkTopology), BlockPlacementPolicy included in BlockManager has intelligence to select nodes for data replication (first on local node, 2nd on different rack, 3rd on same rack with 2nd, others are random) and form them to a pipeline for DataStreamer (specified in DFSOutputStream) to consume.
4. ***Choosing replica of block in HDFS read flow***: When DFSClient want to read (fetchBlockAt) block, it get sorted block locations based on network topology from Namenode.
5. ***Task/container scheduling***: ApplicationMaster take the input splits and translate to requests of ContainerRequestEvent with data locality info (host, rack) and send to RM scheduler (capacity or FIFO). Receiving resource requests from

applications, the RM scheduler will assign containers to NodeManager when NodeManager's heartbeat comes in and with available capacity. Then the scheduler will pick out some applications in some order (Capacity or FIFO) and try to meet their resource requirement which record in cached resource request. For specific application, the sequence of container assignment is: first to meet requests asked on local node, then on local rack, last off rack. After getting containers, AM then schedule map tasks (in sequence of node-local, rack-local, rack-off also) into these containers based on MapsHostMapping and MapsRackMapping.

6. ***Balancer***: With maintaining a ClusterMap, Balancer of HDFS will move some blocks from overloaded nodes to other nodes in sequence of within the same rack, off rack.

Appendix II — The dilemma of current network topology for hadoop cluster running in virtualization environment

If we do nothing on hadoop code base which means we keep two layer hierarchies, then we will have to omit one layer for hadoop running on virtualization environment.
The possible choices could be:
1. Omit Rack layer – "Make host as rack, make vm as node" (Rack(N) → Host(V), Node(N)→VM(V))
Cons: no physical rack awareness
- No reliability of recovery from rack failure
- No rack traffic optimization in HDFS read and scheduling tasks (see below graph)



2. Omit host layer – "Keep rack info there, but treat VMs as different nodes"
Cons: no physical host awareness
- No differentiation for the communication cost between VMs on and off the same host

- No differentiation for failure zone (2nd and 3rd replica could be on same physical host), even worse if cannot find node off-rack

3. Omit VM layer – "Keep rack info there, treat VMs in the same host as the same node" (Rack(N) → Rack(V), Node(N) → Host(V))
Cons: break a lot of assumptions, i.e. remove/update one data node will affect all data nodes on the same physical host.

Thus, there is no tricky configuration that can make current network topology works perfect for hadoop running in virtualization environment.