HDFS 集群的机架感知

目录

1、背景

Hadoop 的设计目的：解决海量大文件的处理问题，主要指大数据的存储和计算问题，其中，HDFS 解决数据的存储问题；MapReduce 解决数据的计算问题

Hadoop 的设计考虑：设计分布式的存储和计算解决方案架构在廉价的集群之上，所以，服务器节点出现宕机的情况是常态。数据的安全是重要考虑点。HDFS 的核心设计思路就是对用户存进 HDFS 里的所有数据都做冗余备份，以此保证数据的安全

那么 Hadoop 在设计时考虑到数据的安全，数据文件默认在 HDFS 上存放三份。显然，这三份副本肯定不能存储在同一个服务器节点。那怎么样的存储策略能保证数据既安全也能保证数据的存取高效呢？

HDFS 分布式文件系统的内部有一个副本存放策略：以默认的副本数=3 为例：
1、第一个副本块存本机
2、第二个副本块存跟本机同机架内的其他服务器节点
3、第三个副本块存不同机架的一个服务器节点上

好处：
1、如果本机数据损坏或者丢失，那么客户端可以从同机架的相邻节点获取数据，速度肯定要比跨机架获取数据要快。
2、如果本机所在的机架出现问题，那么之前在存储的时候没有把所有副本都放在一个机架内，这就能保证数据的安全性，此种情况出现，就能保证客户端也能取到数据

HDFS 为了降低整体的网络带宽消耗和数据读取延时，HDFS 集群一定会让客户端尽量去读取近的副本，那么按照以上头解释的副本存放策略的结果：
1、如果在本机有数据，那么直接读取
2、如果在跟本机同机架的服务器节点中有该数据块，则直接读取

3、如果该 HDFS 集群跨多个数据中心，那么客户端也一定会优先读取本数据中心的数据

但是 HDFS 是如何确定两个节点是否是统一节点，如何确定的不同服务器跟客户端的远近呢？答案就是机架感知。!!!!

在默认情况下，HDFS 集群是没有机架感知的，也就是说所有服务器节点在同一个默认机架中。那也就意味着客户端在上传数据的时候，HDFS 集群是随机挑选服务器节点来存储数据块的三个副本的。

那么假如，datanode1 和 datanode3 在同一个机架 rack1，而 datanode2 在第二个机架 rack2，那么客户端上传一个数据块 block_001，HDFS 将第一个副本存放在 dfatanode1，第二个副本存放在 datanode2，那么数据的传输已经跨机架一次（从 rack1 到 rack2），然后 HDFS 把第三个副本存 datanode3，此时数据的传输再跨机架一次（从 rack2 到 rack1）。显然，当 HDFS 需要处理的数据量比较大的时候，那么没有配置机架感知就会造成整个集群的网络带宽的消耗非常严重。

下图是没有配置机架感知的 HDFS 集群拓扑：

```
[hadoop@hadoop02 hadoop]$ hadoop dfsadmin -printTopology
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

Rack: /default-rack
   192.168.123.102:50010 (hadoop02)
   192.168.123.103:50010 (hadoop03)
   192.168.123.104:50010 (hadoop04)
   192.168.123.105:50010 (hadoop05)
```

2、配置机架感知

2.1、修改配置文件 core-site.xml

给 NameNode 节点的 core-site.xml 配置文件增加一项配置：

```
<property>
    <name>topology.script.file.name</name>
    <value>/home/hadoop/apps/hadoop-2.6.5/etc/hadoop/topology.sh</value>
</property>
```

这个配置项的 value 通常是一个执行文件，该执行文件是一个 shell 脚本 topology.sh，该脚本接收一个参数，输出一个值。
接收的参数：datanode 节点的 IP 地址，比如：192.168.123.102
输出值：datanode 节点所在的机架配置信息，比如：/switch1/rack1

Namenode 启动时，会判断该配置选项是否为空，如果非空，则表示已经启用机架感知的配置，此时 namenode 会根据配置寻找该脚本，并在接收到每一个 datanode 的 heartbeat 时，

将该 datanode 的 ip 地址作为参数传给该脚本运行，并将得到的输出作为该 datanode 所属的机架 ID，保存到内存的一个 map 中.

至于脚本的编写，就需要将真实的网络拓朴和机架信息了解清楚后，通过该脚本能够将机器的 ip 地址和机器名正确的映射到相应的机架上去。一个简单的实现如下：

```
#!/bin/bash
HADOOP_CONF=/home/hadoop/apps/hadoop-2.6.5/etc/hadoop
while [ $# -gt 0 ] ;
do
        nodeArg=$1
        exec<${HADOOP_CONF}/topology.data
        result=""
        while read line
        do
                ar=( $line )
                if [ "${ar[0]}" = "$nodeArg" ]||[ "${ar[1]}" = "$nodeArg" ]
                then
                        result="${ar[2]}"
                fi
        done
        shift
        if [ -z "$result" ]
        then
                echo -n "/default-rack"
        else
                echo -n "$result"
        fi
done
```

那么通过阅读脚本内容知道，我们需要准备一个 topology.data 的文件。topology.data 的内容如下：

```
192.168.123.102   hadoop02        /switch1/rack1
192.168.123.103   hadoop03        /switch1/rack1
192.168.123.104   hadoop04        /switch2/rack2
192.168.123.105   hadoop05        /switch2/rack2
```

其中 switch 表示交换机，rack 表示机架

需要注意的是，在 Namenode 上，该文件中的节点必须使用 IP，使用主机名无效，而 ResourceManager 上，该文件中的节点必须使用主机名，使用 IP 无效，所以，最好 IP 和主机名都配上。

注意：以上两个文件都需要添加可执行权限

chmod 777 topology.data topology.sh

2.2、验证机架感知

以上配置做好之后，启动集群，启动完集群之后，在使用命令：

hdfs dfsadmin -printTopology

查看整个集群的拓扑图：

```
[hadoop@hadoop02 hadoop]$ hadoop dfsadmin -printTopology
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

Rack: /switch1/rack1
   192.168.123.102:50010 (hadoop02)
   192.168.123.103:50010 (hadoop03)

Rack: /switch2/rack2
   192.168.123.104:50010 (hadoop04)
   192.168.123.105:50010 (hadoop05)
```

3、补充

3.1、增加 datanode 节点

增加 datanode 节点，不需要重启 namenode

非常简单的做法：在 topology.data 文件中加入新加 datanode 的信息，然后启动起来就 OK

3.2、节点间距离计算

有了机架感知，NameNode 就可以画出下图所示的 datanode 网络拓扑图。D1,R1 都是交换机，最底层是 datanode。则 H1 的 rackid=/D1/R1/H1，H1 的 parent 是 R1，R1 的是 D1。这些 rackid 信息可以通过 topology.script.file.name 配置。有了这些 rackid 信息就可以计算出任意两台 datanode 之间的距离，得到最优的存放策略，优化整个集群的网络带宽均衡以及数据最优分配。

distance(/D1/R1/H1，/D1/R1/H1)=0	相同的 datanode
distance(/D1/R1/H1，/D1/R1/H2)=2	同一 rack 下的不同 datanode
distance(/D1/R1/H1，/D1/R2/H4)=4	同一 IDC 下的不同 datanode
distance(/D1/R1/H1，/D2/R3/H7)=6	不同 IDC 下的 datanode