Week 2

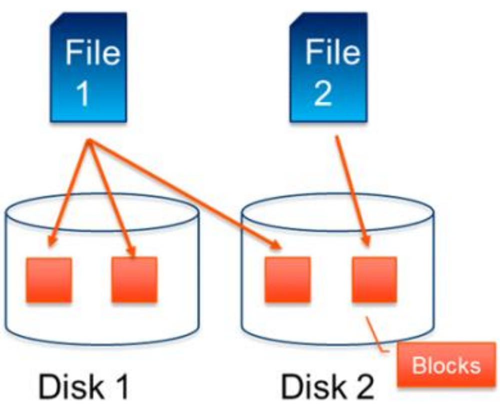
Monday, September 21, 2020 8:13 AM

Lecture Notes

Hadoop Recap:

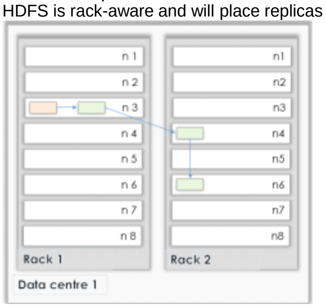
- Hadoop is a framework that allows for distributed storage & processing of large amounts of data
- Allows horizontal scaling by simply adding more servers to a cluster
- Each server add more computing power and storage space
- Provides resiliency (durability) and high availability
- Core components:
    - *MapReduce*: "programming paradigm" that processes large amounts of data
    - *YARN* (yet another resource negotiator) for resource management
    - *HDFS* (Hadoop distributed file system) allows storage of big data
        - Namenodes: contain metadata
        - Datanodes: contain data
- Hadoop is not the right solution for everything. It's not good for quick reading (not the purpose), having a lot of small files (more info to store/manage on namenodes), and lots of people writing to a DB (not supported because writes are always appends)
- Hadoop is good for taking in a large amount of big files, streaming data access, and then read multiple times.
    - "Streaming data access" – High throughput (quick) across most or all the data set. Not just individual transactions. Not like looking at a single bank transaction for a date (that's individual). Get a lot or most (if not all) of the data on our system. Think of something like an airplane and the constant change in temp/elevation, etc
        - Streaming data access is a write once, read many times pattern

HDFS Blocks:

- Data is stored in files, files are stored in directories. Similar to file structure in Windows/Linux
- To avoid fragmentation, HDFS has "blocks", which are fixed sizes (128 MB), which is the minimum amount of data that be read or written
    - Seems large, but this is big data after all
    - It also minimizes the costs of seeks--if there's less items to search through, the read will be fast
    - Having the blocks all the same size helps simplify the system which is good because reasons
    - Having them all the fixed sizes helps for file restoration
- Because of block abstraction, files can be stored in one or more blocks, and those blocks can be spread over multiple disks
- 

- Can run `hdfs fsck / -files -blocks` to see the list of blocks in a file system

More about Namenodes and Datanodes

- Namenodes:
    - "Master node"/momnode
    - 1 per HDFS cluster (originally, and for the purposes of our class)
    - Stores namespace info in memory, such as filesystem tree, files & directory info, block location,
    - Data is stored persistently in two files: namespace image & edit log files
        - Persist is needed because if lost, the system has no idea where files are stored on datanodes. Relates to durability
        - Knows which datanodes have the blocks for a file, but doesn't store where the blocks themselves are located as block locations are dynamically reconstructed on system start
        - *Namespace image:* 'capture' of the current data, although not updated for every write
        - *Edit log:* running list of changes since last namespace image
    - Namenodes can be seen as a kind of bottleneck, as they are responsible for handling all of the cluster's metadata. To resolve this, you can use federation
        - ***Federation*****:** adding more namenodes, each of which is responsible for metadata for part of the cluster. Basically, MapReduce-ing it.
        - Allows us to bypass the limitation of the hardware for the namenode system.
        - Each namenode manages a namespace volume which has metadata for the namespace plus a "block pool" containing all the blocks for the files in the namespace
            - Namespace volumes are separate from each other, meaning namenodes won't talk to each other. If one goes down, others are not affected
- Datanodes:
    - Worker nodes; do all the heavy lifting. They stores and retrieve blocks on demand, plus occasionally update the namenode with the list of blocks they are storing
    - Usually multiple in HDFS cluster
    - Stores and retrieves blocks and notifies namenode of blocks stored
        - Blocks are replicated on different datanodes for fault tolerance (default factor is 3)
        - HDFS is rack-aware and will place replicas on nodes in different racks
        - 

Resiliency and High Availability

- By default, HDFS has high resiliency--the namenode has data written/stored so that in the case of failure, it's still saved. However, the recovery can take a long time which can lead to issues if you need high availability as well. If you wanted to do that, here is what you would do/how it happens:

    1. Write metadata to multiple places (local and network file system, writes are done at the same time and atomic--they're both done at the same time and finish or its not done)
    2. Secondary namenode (usually separate machine)
        1. Warning: doesn't work like a normal name node
        2. Separate machine that does "checkpointing": merges namespace data with the edit logs (which has been growing to keep a record of what's happening on the cluster) to keep edit logs small/from getting to big -- checkpoint works as a starting point if a restart is needed
    3. When the main namenode fails: one of the secondary namenodes (that was on standby) takes over. This is controlled by the failover controller
    4. Datanodes send block reports to the namenode, and then it's ready to go. This can take about tens of seconds vs a normal reboot which can take 30 mins
- In the case of an ungraceful failover (system failure), sometimes the failed namenode doesn't realize it has failed (ie, network issues). To prevent any corruption because of this, fencing is done to prevent that namenode from doing any work
- You can also choose to switch/shutdown a namenode for something like routine maintenance, which is called graceful failover
- All of the above being said, the likelihood of the namenode going down is low so this is rarely implemented
- Depends on your business needs and normally up for discussion when deciding how to set up your Hadoop

Textbook Notes

Block Caching:
- Normally, datanodes read blocks from disks, but for frequently access files the block might be cached in the datanode's memory in an off-heap "block cache"
- Blocks are normally cached in only one datanode, but can be configured differently
- This is done to improve read performance for jobs such as MapReduce, Spark, etc
- Example for when this is useful: small lookup table used for a join
- You can specify which files to cache by adding a cache directive of cache pool to the namenode (NOT the datanode!)

Other additions made to original notes section

Hadoop Config Lab

Hadoop files that need to be configured:
- hadoop-env.sh
    - Sets hadoop environment variables, such as java home, hadoop home, configure and log directory locations
- hdfs-site.xml
    - Settings for the HDFS daemons, configuration for namenode & datanode directories, replication level (default is 3) and security info. Also configures permissions on HDFS and IO resources
- core-site.xml
    - Informs hadoop daemon of the namenode's location
- yarn-site.xml
    - Configures the resource management
- mapred-site.xml
    - Sets MapReduce settings
- workers
    - Defines the datanodes.
    - NOT an xml file, just a list of IP addresses separated by a line break

hadoop-env.sh

- Sets Hadoop environment variables

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export HADOOP_HOME=/opt/hadoop
export HADOOP_CONF_DIR=/opt/hadoop/etc/hadoop
export HADOOP_LOG_DIR=/opt/hadoop/logs
```


hdfs-site.xml

Settings for the HDFS daemons. Here you will configure:

- Directories where the namenode and datanode store their files
- Replication level
- Security

Also configures permissions on HDFS and IO resources

```
<configuration>
<property>
<name>dfs.namenode.name.dir</name>
<value>file:///opt/hdfs/namenode</value>
<description>NameNode directory for namespace and transaction logs storage.</description>
</property>

<property>
<name>dfs.datanode.data.dir</name>
<value>file:///opt/hdfs/datanode</value>
<description>DataNode directory</description>
</property>

<property>
<name>dfs.replication</name>
<value>3</value>
</property>
```


core-site.xml

- Informs Hadoop daemon of the location of the namenode

```
<configuration>
<property>
<name>fs.defaultFS</name>
<value>hdfs://hadoop1:9820/</value>
<description>NameNode URI</description>
</property>
</configuration>
```


yarn-site.xml

- Configuration of the resource manager

```
<configuration>
<property>
<name>yarn.nodemanager.local-dirs</name>
<value>file:///opt/yarn/local</value>
</property>
<property>
<name>yarn.nodemanager.log-dirs</name>
<value>file:///opt/yarn/logs</value>
</property>
</configuration>
```

yarn-site.xml

- Configuration of the resource manager

```
<configuration>
<property>
<name>yarn.nodemanager.local-dirs</name>
<value>file:///opt/yarn/local</value>
</property>
<property>
<name>yarn.nodemanager.log-dirs</name>
<value>file:///opt/yarn/logs</value>
</property>
</configuration>
```


mapred-site.xml

- Sets the MapReduce properties

```
<configuration>
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
<description>MapReduce framework name</description>
</property>
<property>
<name>mapreduce.jobhistory.address</name>
<value>hadoop1:10020</value>
<description>Default port is 10020</description>
</property>
<property>
<name>mapreduce.jobhistory.webapp.address</name>
<value> hadoop1:19888</value>
<description>Default port is 19888</description>
</property>
<property>
<name>mapreduce.jobhistory.intermediate-done-dir</name>
<value>/mapredhistory/tmp</value>
<description>Directory where history files are written by MapReduce jobs.</description>
</property>
<property>
<name>mapreduce.jobhistory.done-dir</name>
<value>/mapredhistory/done</value>
<description>Directory where history files are managed by the MR JobHistory Server.</description>
</property>
</configuration>
```


workers

- Defines the datanodes

```
192.168.56.5
192.168.56.6
192.168.56.7
```