Hadoop
History and Introduction

Explained By
Vaibhav Agarwal
Agenda

• Architecture
• HDFS Data Flow
• Map Reduce Data Flow
• Hadoop Versions History
• Hadoop version 2
Hadoop Architecture

HDFS & MapReduce

- **Hadoop Distributed File System**
  - A scalable, Fault tolerant, High performance distributed file system
  - Asynchronous replication
  - Write-once and read-many (WORM)
  - Hadoop cluster with 3 DataNodes minimum
  - Data divided into 64MB (default) or 128MB blocks, each block replicated 3 times (default)
  - No RAID required for DataNode
  - Interfaces: Java, Thrift, C Library, FUSE, WebDAV, HTTP, FTP
  - **NameNode** holds filesystem metadata
  - Files are broken up and spread over the **DataNodes**

- **Hadoop Map Reduce**
  - Software framework for distributed computation
  - Input | Map() | Copy/Sort | Reduce() | Output
  - **JobTracker** schedules and manages jobs
  - **TaskTracker** executes individual map() and reduce() tasks on each cluster node
HADOOP (HDFS) Data Flow

**The Cast**

- **Client**: People sit in front of me and ask me to read/write data.
- **NameNode**: There is only ONE of me... and I coordinate everything around here.
- **DataNodes**: We store data, there are MANY of us sometimes even thousands!

**Writing Data in HDFS Cluster**

**Request From User**

Let's start with writing some data.

Mr. Client, please write 200 MB data for me.

It'll be my pleasure. But...

**Block and Replication**

--are you not forgetting something?

Ah yes, please: a) divide the data in 128MB blocks b) copy each block in three places

A good client always knows these two things:

- **BlockSize**: large file is divided in blocks (usually 64 or 128MB)
- **Replication Factor**: each block is stored in multiple locations (usually 3)

**Divide File into Blocks**

Here you go buddy. Addresses of three datanodes. I have also sorted them in increasing distance from you

Thank you!

Datnode 1, Datnode 2, Datnode 3

**Ask NameNode**

Let's work on the first block first

Mr. NameNode: please help me write a 128MB block with replication of 3

**NameNode Assigns DataNodes**

Replication 3... Hmm... need to find 3 datanodes for this client

How do I do that? Will tell you some other time

**Client Starts Writing Data**

I send my data (and the list) to first datanode only

I store the data in hard drive, and--

**While I am receiving data**

I forward the same data to the next datanode

© Maneesh Varshney, mvarshney@gmail.com
Reading from HDFS
Writing to HDFS
Secondary Name Node

- Not a hot standby for the Name Node
- Connects to Name Node every hour*
- Housekeeping, backup of Name Node metadata
- Saved metadata can rebuild a failed Name Node

* This is an additional note provided in the image.
HADOOP (Map Reduce) Data Flow
External Client i.e. user

- Clients get all the files prepared
  - Training/Input Data,
  - input parameters,
  - config file,
  - jar file i.e. code which specifies InputFormat
    - Defines InputSplits i.e. how to split the input
    - Provides factory for RecordReader i.e. how to convert InputSplit to Record, Value pairs.
  - output directory
- Copy these files to HDFS
  - Training/Input Data is to be copied dfs.replication (Default 3) times.
  - Other data i.e. User Code, config file, input parameters, etc are to be copied once as of now. JobClient copies it to HDFS later.
- Requests to start job and specifies HDFS address where files are stored.
- Compressed input files are not splitable (without patches)
Ask JobTracker for a new Job Id.

Checks if input and output specification of job is correct ie directories are correctly specified, etc.

Splits the input into “logical” `InputSplit` instances.
   - Sorts the InputSplits in descending order of size.

Creates a new directory in HDFS named after job ID and copies with replication factor `mapred.submit.replication`
   - job JAR file
   - configuration file
   - computed input splits

Informs JobTracker that Job with Job Id is ready.

Optionally polls JobTracker to get the status.
Job Tracker

- Only one JobTracker process run on any Hadoop cluster. It runs in its own JVM process.
- Puts job in an internal queue from where JobScheduler picks and initializes.
- Once a job is selected, Task Assignment is done by the JobTracker.
- JobTracker talks to NameNode to determine the location of data.
- It locates the TaskTracker nodes with available slots at or near the data.
- It submits work to the chosen task tracker nodes.
- If a task is completed
  - Updates its status. Client polls for status.
  - Makes a note of map output and task tracker in the Job object
  - Identifies the next task to be executed on the TaskTracker.
- Takes care of speculative execution.
- Replies Reducer Task with map output locations it noted earlier.
Task Tracker (Generic)

• First localizes the JAR, config file and one copy of input split.
  – Job jar and config file are copied from shared filesystem to task tracker’s filesystem.
  – Copies any file needed from distributed cache to the local disk.
• Creates local working directory for task and unjars the contents of JAR into this directory.
• Executes TaskRunner() by launching a new JVM
  – To run each task separately
  – Any bugs in mapper/reducer doesn’t affect the TaskTracker
  – Older launched JVMs may be reused
  – Updates parent process its progress every few seconds until task is complete
• If a task reports progress it sets a flag which is tested every 3 seconds by a separate thread. If flag is set, status is reported to task tracker.
• The task tracker heartbeat is a different loop which is sent every 5 seconds and is directly proportional and dependent on the size of cluster.
• The heartbeat comprises of:
  – Number of available mapper and reducer slots
  – status of mapper/reducer jobs already running
  – Status of RAM and Virtual Memory on the node
  – ids of blocks possessed (HDFS heartbeat)
  – detail of I/O threads currently working (HDFS heartbeat)
Task Tracker (Mapper)

- Fetches Input split if not locally present
- Fetches JAR code and config files if not locally present
- RecordReader converts InputSplit into key, value pairs
- Once all the input available has been converted, executes user defined map() method.
- On task completion
  - notify tasktracker of status update
  - goes to commit pending state ie renames output file and performs clean up.
  - Task Tracker writes the outputs on HDFS and gets them replicated.
  - The mapper outputs are deleted when Job Tracker instructs to do so
- Status for map task is identified by the proportion of input that has been processed.
- A flag is set if the task progresses. Every 3 seconds a separate thread runs to test if the flag is set. On it being set, task reports status to task tracker.
Task Tracker (Reducer)

- Fetches Outputs from ALL the mappers even if they have not completed. Keeps on updating the input as mapper execution proceeds.
- Periodically asks Job Tracker for map output locations until it has retrieved them all.
- Starts execution only when all the input from all the mappers has been received.
- Status of reduce phase being more complex is identified as (1/3 each for copy, sort and reduction).
- A flag is set if the task progresses. Every 3 seconds a separate thread runs to test if the flag is set. On it being set, task reports status to task tracker.
- Once all input has been received and merged, user defined reduce() method executes.
## Brief History of Hadoop Versions

<table>
<thead>
<tr>
<th>Year</th>
<th>Release</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>0.20.2</td>
<td>The last time one release had all the usable features committed to Apache Hadoop</td>
</tr>
<tr>
<td></td>
<td>0.21</td>
<td>Has append, RAID and symlinks but does not have security</td>
</tr>
<tr>
<td>2011</td>
<td>0.20.203</td>
<td>Has security but does not have append (HBase will lose data) or RAID</td>
</tr>
<tr>
<td></td>
<td>0.20.205(now 1.0)</td>
<td>Has append and security but does not have RAID, symlinks or new MapReduce (aka MR2)</td>
</tr>
<tr>
<td></td>
<td>0.22</td>
<td>Has append, HDFS security, RAID and symlinks but does not have MapReduce security and some performance improvements</td>
</tr>
<tr>
<td></td>
<td>0.23</td>
<td>The first time in 18 months that one release has all the usable features committed to Apache Hadoop</td>
</tr>
<tr>
<td>2012</td>
<td>??</td>
<td>Back to normal (hopefully)</td>
</tr>
</tbody>
</table>
Hadoop 0.23 features

• Released on November 15, 2011
• Federated HDFS
• Introduction of YARN or MRv2 or NextGen Map Reduce
• Same as Hadoop v2.0 alpha
Hadoop 2.x features

1. Name Node federation
   (Multiple Namespaces are permitted as opposed to earlier 1 NS)
2. A big change in architecture

• Why?
  – Under utilization
    • Hard partition of resources into map and reduce slots
    • Many a times either only mapper or reducer tasks run
  – Scalability
    • Max Cluster size - 4000 nodes
    • Max Concurrent tasks – 40000
    • Coarse Synchronization in Job Tracker
  – Single Point of Failure
    • Failure killed all queued and running jobs
  – Lacks support for alternate paradigms
  – Necessity of client and cluster to be of same version
Requirements

• Reliability
• Availability
• Cluster Utilization
• Wire Compatibility
• Agility and Evolution
  – Ability for customers to control Hadoop software stack trace.
• Scalability - Clusters of 6000-10,000 (16 core) m/c
  – 100,000+ concurrent tasks
  – 10,000+ concurrent jobs
• Support for other Programming Paradigms
  – OpenMPI, Spark is Hive-on-Spark
  – Storm (Real time data processing by Twitter), Apache S4
  – Graph Processing – Apache Giraph
• Support for short lived services
• Predictable Latency
• Map Reduce Reduce – will be done in future
Next Gen Map-Reduce Design Center

- Split up the two major functions of Job Tracker
  - Cluster resource management
  - Application life-cycle management
- Map-Reduce becomes user-land library
Components

• Resource Manager
  – Global resource scheduler
  – Hierarchical queues

• Node Manager
  – Per-machine agent
  – Manages the life-cycle of container
  – Container resource monitoring

• Application Master
  – Per-application
  – Manages application scheduling and task execution
  – Eg. Map-Reduce Application Master
RM Architecture
RM Architecture

• ASM
  – Responsible for launching/managing Application Master
  – Monitors the AM for liveness

• Scheduler
  – Responsible for scheduling containers to applications
  – Schedules on heartbeats from NM’s
  – Returns scheduled containers to AM’s on heartbeats

• Resource Tracker
  – Responsible for managing Node Managers
  – Monitors liveness of NM’s
  – Maintains data structures for containers running on particular NM
Map Reduce Application

• JobClient
  – Uses ClientRMProtocol to submit to RM

• ApplicationMaster
  – Uses AMRMProtocol to get containers
  – Uses ContainerManager to launch containers

• Task
  – Same task as in JT world
  – Talk to AM over TaskUmbilicalProtocol
Application Master: Component Event Flow
Thank you

This is my thank you dance!
Input Split

- Represents data to be processed by an individual Mapper
- Comprises of
  - total split length (in bytes)
  - Locations ie list of hostnames where the input split is located.
- Upper bound: dfs.block.size, Lower bound: mapred.min.split.size
- In HDFS terms:
  - Split = physical data block
  - Location = set of machines where block is located (set size = replication factor)
- In HBase terms:
  - Split = set of table keys belonging to a table region
  - Location = machine where a region server is running
- Data Locality can be defined by implementing custom InputFormat by overriding getSplits() method.
Default Scenario

– FileInputFormat is TextInputFormat i.e. input is Text File
– InputSplit is FileSplit which sets map.input.file to the path of the input file for the logical split.
– RecordReader is LineRecordReader i.e. key is offset and value is every line
– Size of input split = size of one HDFS block = mapred.min.split.size
Hadoop 0.19 features

• Released on 24 Nov, 2008
• Hadoop Core NOW needs Java 6
• Filesystem checksum changed
• Thrift interface was added to include languages other than Java
Technology Challenges Limiting Enterprise Adoption

1. Dedicated Storage Infrastructure
   - One-off for Hadoop only

2. Single Point of Failure
   - Namenode

3. Lacking Enterprise Data Protection
   - No Snapshots, replication, backup

4. Poor Storage Efficiency
   - 3X mirroring

5. Fixed Scalability
   - Rigid compute to storage ratio

6. Manual Import/Export
   - No protocol support

Hadoop DAS Environment

© Copyright 2011 EMC Corporation. All rights reserved.