PART 1.
LARGE SCALE DATA ANALYSIS USING MAPREDUCE

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs435

Today’s topics
- FAQs
- MapReduce

FAQs
- Reed-Solomon algorithm?

Reed-Solomon Codes
- Block-based error correcting codes
- Digital communication and storage
- Storage devices (including tape, CD, DVD, barcodes, etc)
- Wireless or mobile communications
- Satellite communications
- Digital TV
- High-speed modems

A Quick Example of the R-S encoding
- 4+2 coding
- Original files are broken into 4 pieces
  - 2 parity pieces are added
- First piece of data “ABCD”, second piece of data “EFGH”...
A Quick Example of the R-S encoding

• Applying coding matrix

\[
\begin{pmatrix}
01 & 00 & 00 & 00 \\
00 & 01 & 00 & 00 \\
00 & 00 & 01 & 00 \\
1b & 1c & 12 & 14 \\
1c & 1b & 14 & 12 \\
\end{pmatrix}
\times
\begin{pmatrix}
A & B & C & D \\
E & F & G & H \\
I & J & K & L \\
M & N & O & P \\
\end{pmatrix}
= 
\begin{pmatrix}
A & B & C & D \\
E & F & G & H \\
I & J & K & L \\
M & N & O & P \\
\end{pmatrix}
\]

• Data loss

2 of 6 rows are lost

\[
\begin{pmatrix}
01 & 00 & 00 & 00 \\
00 & 01 & 00 & 00 \\
00 & 00 & 01 & 00 \\
1b & 1c & 12 & 14 \\
1c & 1b & 14 & 12 \\
\end{pmatrix}
\times
\begin{pmatrix}
A & B & C & D \\
E & F & G & H \\
I & J & K & L \\
M & N & O & P \\
\end{pmatrix}
= 
\begin{pmatrix}
A & B & C & D \\
E & F & G & H \\
I & J & K & L \\
M & N & O & P \\
\end{pmatrix}
\]

• Without 2 rows

\[
\begin{pmatrix}
01 & 00 & 00 & 00 \\
00 & 01 & 00 & 00 \\
1b & 1c & 12 & 14 \\
1c & 1b & 14 & 12 \\
\end{pmatrix}
\times
\begin{pmatrix}
A & B & C & D \\
E & F & G & H \\
I & J & K & L \\
M & N & O & P \\
\end{pmatrix}
= 
\begin{pmatrix}
A & B & C & D \\
E & F & G & H \\
I & J & K & L \\
M & N & O & P \\
\end{pmatrix}
\]

• Multiplying each side with the inverted matrix

\[
\begin{pmatrix}
01 & 00 & 00 & 00 \\
00 & 01 & 00 & 00 \\
1b & 1c & 12 & 14 \\
1c & 1b & 14 & 12 \\
\end{pmatrix}
\times
\begin{pmatrix}
01 & 00 & 00 & 00 \\
00 & 01 & 00 & 00 \\
1b & 1c & 12 & 14 \\
1c & 1b & 14 & 12 \\
\end{pmatrix}
= 
\begin{pmatrix}
A & B & C & D \\
E & F & G & H \\
I & J & K & L \\
M & N & O & P \\
\end{pmatrix}
\]

• The Inverse Matrix and the Coding Matrix Cancel Out

\[
\begin{pmatrix}
01 & 00 & 00 & 00 \\
00 & 01 & 00 & 00 \\
1b & 1c & 12 & 14 \\
1c & 1b & 14 & 12 \\
\end{pmatrix}
\times
\begin{pmatrix}
01 & 00 & 00 & 00 \\
00 & 01 & 00 & 00 \\
1b & 1c & 12 & 14 \\
1c & 1b & 14 & 12 \\
\end{pmatrix}
= 
\begin{pmatrix}
A & B & C & D \\
E & F & G & H \\
I & J & K & L \\
M & N & O & P \\
\end{pmatrix}
\]

• Reconstructing the Original Data

\[
\begin{pmatrix}
01 & 00 & 00 & 00 \\
00 & 01 & 00 & 00 \\
1b & 1c & 12 & 14 \\
1c & 1b & 14 & 12 \\
\end{pmatrix}
\times
\begin{pmatrix}
01 & 00 & 00 & 00 \\
00 & 01 & 00 & 00 \\
1b & 1c & 12 & 14 \\
1c & 1b & 14 & 12 \\
\end{pmatrix}
= 
\begin{pmatrix}
A & B & C & D \\
E & F & G & H \\
I & J & K & L \\
M & N & O & P \\
\end{pmatrix}
\]
Example

- RS(255,223) with 8 bit symbols
  - Each code word contains 255 code word bytes
  - 223 bytes are data and 32 bytes are parity
  - \( n=255, k=223, s=8, 2t = 32, t=16 \)

- The decoder can correct any 16 symbol errors in the code word

Errors in up to 16 bytes anywhere in the codeword can be automatically corrected.

YARN Framework

YARN (MapReduce 2)

- Resource manager
  - Manages the use of resources across the cluster

- Node manager
  - Launches and monitors the compute containers on machines in the cluster

- Application master
  - Manages the lifecycle of applications running on the cluster
  - Application master negotiates with the resource manager for cluster resources
    - Number of container and certain memory limit
    - Node managers oversee containers not to use more resources than allocated

A MapReduce job using YARN

- Reed Solomon Error correction code
- R&D white paper

YARN (MapReduce 2)

- To provide the scalability to MapReduce
  - Splitting responsibility of the job tracker
    - Scheduling
    - Task progress monitoring

- MapReduce is one type of YARN application
Job Initialization

• Application master
  - The application master for MapReduce jobs
    - RRAppMaster
    - Plan job execution
    - If the job is small, runs in the same JVM as itself
    - Uber task
    - Overhead of allocating and running tasks in new container outweighs the gain from running them in parallel, compared to running them sequentially on one node

What is a small job?
A small job is one that has less than 10 mappers, only one reducer, and an input size that is less than the size of one HDFS block

Task assignment

• Application Master requests container for all map and reduce tasks in the job
  - From the resource manager (Step 8)

• All the requests include information about each task's locality
  - Host and corresponding racks the input split resides on
  - Scheduler attempts to place tasks on data-local nodes in the ideal case
    - If not possible, scheduler prefers rack-local placement
    - Job is running on a node in the same rack
Task assignment [2/2]

- Requests specify required memory
  - 1024MB (by default)
  - This is configurable
    - mapreduce.map.memory.mb
    - mapreduce.reduce.memory.mb
- In YARN, resources are managed more fine-grained
  - Applications may request a memory capability that is anywhere between the minimum allocation and a maximum allocation
    - yarn.scheduler.capacity.minimum-allocation-mb
    - Default minimum: 1024MB
    - yarn.scheduler.capacity.maximum-allocation-mb
    - Default maximum: 10240 MB
  - Tasks can request any memory allocation between 1 and 10GB (default) in multiple of 1GB
    - mapreduce.map.memory.mb and mapreduce.reduce.memory.mb

Task execution

- Application master starts the container by contacting node manager
  - The task is executed by YarnChild
    - YarnChild runs in a dedicated JVM

Progress and status updates

- Task reports its progress and status back to its application master
  - Every 3 seconds over the umbilical interface

  - The client polls the application master every second
    - mapreduce.client.progressmonitor.pollinterval

Failures

- The child task fails
  - Runtime exception from the user code
    - The child JVM reports the error back to its parent before it exits
    - Written in the user logs
    - Tasktracker marks the task attempt as failed
    - Frees a slot to run another task
    - Sudden exit of the child JVM
      - Tasktracker notices that the process has exited and marks the attempt as failed
  - Hanging tasks
    - If there is no progress update for a while
      - Mark the task as failed
      - Timeout period is normally 10 minutes
        - mapred.childtask.timeout
Tasktracker failure in Classic MR

- Tasktracker stops sending heartbeats
  - Jobtracker will notice if it hasn’t received one for 10 minutes (configurable)
  - Remove it from the pool of tasktracker
- Jobtracker arranges tasks including the completed jobs
  - Because the output may not be accessible
- Tasktracker can also be blacklisted if more than four tasks from the same job fail (set by mapred.max.tracker.failures)
  - Blocklisted tasktrackers are not assigned tasks.
  - Until faults expire

Jobtracker failure in Classic MR

- The most serious failure mode
- Hadoop has no mechanism for dealing with jobtracker failure
  - Single point of failure
  - All running jobs fail
- After restarting a jobtracker
  - Job should be resubmitted
- This is improved with YARN

Task failure in YARN

- Failure of the running task is similar to the classic case
  - Runtime exception and sudden exit of the JVM are propagated back to the application master
  - The task attempt is marked as failed
  - Hanging tasks are noticed by the application manager by the absence of a ping over the umbilical channel

Application master failure in YARN

- No heartbeats to the resource manager from the application master
  - The resource manager will detect the failure and start a new instance of the master running in a new container
  - All tasks will be rerun (default)
  - Recovery can be enabled
  - Client will access resource manager to get the new address of the application master

Node manager failure in YARN

- Resource manager will stop getting heartbeats
  - Remove the failed node manager from the pool of available nodes
- Any task or application master running on the failed node manager will be recovered

Resource manager failure

- After a crash and a new resource manager instance is brought up (by administrator)
  - It recovers from the saved state
  - Checkpoints saved in the persistent storage
  - Non-completed jobs are included
Hadoop Combiner

- Minimizes the data transferred between map and reduce tasks
- Users can specify a *combiner function*
  - To be run on the map output
  - To replace the map output with the combiner output
- Hadoop does not guarantee how many times it will call combiner for a particular map output record

### Example: Find the maximum temperature

- First map produces
  - (1950, 0)
  - (1950, 20)
  - (1950, 10)

- Second map produces
  - (1950, 25)
  - (1950, 15)

- Input to the reduce function
  - (1950, [0, 20, 10, 25, 15])

- Output
  - (1950, 25)

If a combiner finds the maximum temperature for each map output:

- First map produces
  - (1950, 0)
  - (1950, 20)
  - (1950, 10)
  
  → (1950, 20)

- Second map produces
  - (1950, 25)
  - (1950, 15)
  
  → (1950, 25)

- Input to the reduce function
  - (1950, [0, 20, 10, 25, 15])
  
  → (1950, [20, 25])

- Output

- Can we use a combiner function for finding mean value?

  - mean (0, 20, 10, 25, 15) = 14
  - mean (mean(0,20,10), mean (25,15)) = mean(10, 20) = 15
Can combiners replace reducers?

- No. Combiner is run over only on the map output.