PART 0. INTRODUCTION
3. DATA MODEL FOR BIG DATA: APACHE THRIFT

Sangmi Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs535

Today’s topics
- Apache Thrift continued
- Introduction to MapReduce

Protocols
- Describes “what” is transmitted
  - Thrift supports both text and binary protocols
    - The binary protocol outperforms text protocol
    - The text protocol may be useful (such as in debugging)
  - TBinaryProtocol
    - A straightforward binary format encoding numeric values as binary, rather than converting to text.
  - TCompactProtocol
    - Very efficient, dense encoding of data

FAQs
- Programming Assignment 1
  - Due Sept. 28
  - Submission via Canvas
  - Please check the course Web Page at least twice a week

Thrift Architecture
You can change the protocol and transport without regenerating code

Protocols (1/2)

- TDenseProtocol
  - Similar to TCompactProtocol but strips off the meta information from what is transmitted, and adds it back in at the receiver.

- TJSONProtocol
  - Uses JSON for encoding of data.

- TSimpleJSONProtocol
  - A write-only protocol using JSON. It cannot be parsed by Thrift. Suitable for parsing by scripting languages

- TDebugProtocol
  - Uses a human-readable text format to aid in debugging.

TCompactProtocol (1/2)

- For integers, the TCompactProtocol performs compression
  - Variable-Length Quantity (VLQ) encoding
  - VLQ uses 7 of 8 bits out of each byte for information
    - the 8th bit used as a continuation bit

TCompactProtocol (2/2)

- Decimal value 106903 (0x1A197) saving 1 byte if it was stored in 32 bits:
  - Big-Endian Versioning
    - Applications evolve over time
      - You added an extra field to your message format
    - The system must be able to support reading of old data from log files
      - Process requests from out-of-date clients to new servers and vice versa
Versioning Field Identifiers

- Any new fields should be optional
- Any messages serialized by code using the "old" message format can be parsed by the new code
- Without any missing required fields (Patching error)
- Messages created by new code can be parsed by old code
- Old binary will ignore the new field when parsing
- Unknown fields are not discarded; it will be serialized along with other fields

- Non-required fields can be removed
- Changing a default value is allowed

- Changing a default value is allowed

This material is built based on

- Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters" in Proceeding OSDI’04 Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation - Vol. 6
- MapReduce Design Patterns, Donald Miner and Adam Shook, O’Reilly, 2013

Why MapReduce?

What is MapReduce?

- MapReduce is inspired by the concepts of map and reduce in Lisp.
What is MapReduce?

- Developed within Google as a mechanism for processing large amounts of raw data.
- Crawled documents or web request logs
- Distributes these data across thousands of machines
- Same computations are performed on each CPU with a different portion of the dataset

Why MapReduce?

- MapReduce provides an abstraction that allows engineers to perform simple computations while hiding the details of:
  - Parallelization
  - Data distribution
  - Load balancing
  - Fault tolerance

Example

- Climate data from National Climatic Data Center (NCDC)

```
0057 332130 # USAF weather station identifier
99999 # NNam weather station identifier
19500101 # observation date
0300 # observation time 4
+ 51317 # latitude (degrees x 1000)
+ 028783 # longitude (degrees x 1000)
FM-12
+ 0171 # elevation (meters)
99999
V020
320 # wind direction (degrees)
1 # quality code
```

Analyzing the data with Unix Tools (1/2)

- A program for finding the *maximum recorded temperature by year* from NCDC weather records
  - e.g. Weather change for a century

```
#!/usr/bin/env bash
for year in all/*
do
echo -ne `basename $ year .gz`\t
gunzip -c $ year | \
  awk '{
    temp = substr( $ 0, 88, 5) + 0;
    q = substr( $ 0, 93, 1);
    if (temp == 9999 && q ~ /01459/) && temp > max
      max = temp;
} END { print max }'
```

The first entries for 1990

```
# ls raw/ 1990 | head
010010-99999-1990.gz
010014-99999-1990.gz
010015-99999-1990.gz
010016-99999-1990.gz
010017-99999-1990.gz
010030-99999-1990.gz
010040-99999-1990.gz
010080-99999-1990.gz
010100-99999-1990.gz
010150-99999-1990.gz
```

Analyzing the data with Unix Tools (2/2)

- The script loops through the compressed year files
  - Printing the year
  - Processing each file using awk
    - Extracts two fields
    - Air temperature and the quality code
  - Check if it is greater than the maximum value seen so far

```
% ./max_temperature.sh
1901 317
1902 244
1903 289
1904 256
1905 283
```

Results?

- The complete run for the century took 42 minutes
- To speed up the processing
  - We need to run parts of the program in parallel
  - Process different years in different processes
  - What will be the problems?

Challenges

- Dividing the work into equal-size pieces
  - Data size per year?
- Combining the results from independent processes
  - Combining results and sorting by year?

Map and Reduce

- MapReduce works by breaking the processing into two phases
  - The map phase
  - The reduce phase
- Each phase has key-value pairs as input and output
- Programmers should specify
  - Types of input/output key-values
  - The map function
  - The reduce function

Visualizing the way MapReduce works      (1/4)

Sample lines of input data

```
(0, 006711990999991950001507004... 999999999 + 00001 +99999999999...) 
(106, 00430119909999919500051518004... 99999999999-00111 +99999999999...)
(1950, 0)
```

These lines are presented to the map function as key-value pairs

```
(0, 006711990999991950001507004... 999999999 + 00001+ 99999999999...)
(0, 006711990999991950001507004... 999999999 + 00001+ 99999999999...)
```

The keys are the line offsets within the file

Visualizing the way MapReduce works      (2/4)

The map function extracts the year and the air temperature and emits them as its output

```
(1950, 0)
(1950, 22)
(1950, -11)
(1949, 111)
(1949, 78)
```

This output key-value pairs will be sorted and grouped by key.
Our reduce function will see the following input:

```
(1949, [111, 78])
(1950, [0, 22, -11])
```

Visualizing the way MapReduce works      (3/4)

Reduce function iterates through the list and picks the maximum reading:

```
(1949, 111)
(1950, 22)
```

This is the final output

```
(1949, 111)
(1950, 22)
```
Visualizing the way MapReduce works (4/4)

Map function
- Takes an input pair
- Produces a set of intermediate key/value pairs
  - The MapReduce library groups together all intermediate values associated with the same intermediate key and passes them to the reduce function

Reduce function
- Accepts an intermediate key and a set of values for that key
- Merges together these values
  - Forms a possibly smaller set of values

Example: counting words
- Count the number of occurrences of each word in a large collection of documents:

```java
map(String key, String value):
// key: document name
// value: document contents
for each word w in value:
EmitIntermediate(w,"1");

reduce(String key, Iterator values):
//key: a word
//values: a list of counts
int result = 0;
for each v in values:
result += ParseInt(v)
Emit(AsString(result));
```

Comparison with other systems
- MPI vs. MapReduce
  - MapReduce tries to colocate the data with the compute node
  - Data access is fast
    - Data is local
- Volunteer computing vs. MapReduce
  - SETI@home
    - Uses donated (volunteered) CPU time
MapReduce Data Flow

Job execution framework
- mapred.job.tracker
  - If the configuration property is set to local
  - Local job runner is used
  - Designed for testing and running with small datasets
- Using classic
  - For the classic MapReduce framework (MapReduce 1)
- Using yarn
  - For the YARN framework

Job submission
- Single method call
  - submit() on a Job object
    - You can also call waitForCompletion()
- Creates an internal JobSubmitter instance
  - Calls submitJobInternal() on it
- waitForCompletion()
  - Polls the job's progress once per second
  - Reports the progress to the console if it has changed since the last report
- Job completes successfully
  - The job counters are displayed
  - Otherwise, the error is logged to the console

MapReduce job in the Classic framework

Job submission process in Classic framework
- Asks the jobtracker for a new jobId
  - Step 2
- Checks the output specification of the job
- Computes the input splits for the job
- Copies the resources to the jobtracker's filesystem in a directory named after the jobId
  - The job JAR file, the configuration file, and the computed input splits
- Tells the jobtracker that the job is ready for execution by calling submitJob() on JobTracker
  - Step 4
Job Initialization in the Classic mode

- JobTracker puts the job into an internal queue
  - Creating an object to represent the job being run
  - Bookkeeping information to keep track of the status and progress of its tasks

- Creating the list of tasks
  - The job scheduler retrieves the input splits
  - The job scheduler creates a map task for each split
  - The number of reduce tasks to create
    - mapred.reduce.tasks property in the Job
    - setNumReduceTasks()

- A job setup task and a job cleanup task are created

Task assignment

- Jobtracker select a job to run
  - Default algorithm
    - Based on the priority list of jobs

- Tasktrackers have a fixed number of slots for map tasks and reduce tasks
  - These are set independently
  - These are selected based on the number of cores and the memory

- The default scheduler fills empty map task slots first
  - Before it fills the reduce task slots
  - Scheduling the reduce task does not need to consider data locality

- Tasktrackers run a simple loop that periodically sends heartbeat method calls to the jobtracker

Task execution

- Tasktracker
  - Copies the job Jar to the tasktracker’s file system
  - Any files needed from the distributed cache to the local file system
  - Creates a local working directory
  - Creates instance of TaskRunner

- TaskRunner
  - Launches a new Java Virtual Machine to run each task
  - Any failed map or reduce does not affect the tasktracker

Streaming and pipes

- Runs special map and reduce tasks
  - To launch the user supplied executable
  - To communicate with it

- Streaming task communicates with the process using standard input and output streams

- Pipes task listens on a socket and passes the C++ process a port number in its environment
  - On startup, the C++ process establish a persistent socket connection back to the parent Java Pipes task

Progress and status updates [1/3]

- A Job and each of its tasks have a status
  - State of the job or task
    - E.g. running, successfully complete, failed
  - The progress of maps and reduces
  - The values of the job’s counters
  - A status message or description

- Progress of a task
  - The proportion of the task completed
    - Map task
      - The proportion of the input that has been processed
      - Reduce task
        - Divides the total progress into 3 parts (copy/sort/reduce)
        - If the task has run the reducer on half its input
          - 1/3 (copy) + 1/3 (sort) + a half of 1/3 (reduce phase) = 5/6

Progress and status updates [2/3]

- Tasks
  - have a set of counters
  - Count various events at the task run
    - E.g. the number of map output records written
  - If a task reports progress
    - It sets a flag to indicate that the status change should be sent to the tasktracker
    - Checked every 3 seconds

- Tasktracker
  - Tasks notify the current task status to the tasktracker
    - If the flag is set
  - Tasktracker sends heartbeats to the jobtracker every 5 seconds (minimum)
Progress and status updates [2/3]

- Jobtracker
  - Combines updates to produce a global view

- Job
  - Receives the latest status by polling the jobtracker every second
  - Prints job statistics and counters to the console

YARN (MapReduce 2)

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

- MapReduce is one type of YARN application