Pig, a high level data processing system on Hadoop

Gang Luo
Nov. 1, 2010
Agenda

- Recap on MapReduce
- Introduction to Pig
- View of Pig from outside
  - Pig Latin
- View of Pig from inside
  - Plan generation
- Future work
Recap on MapReduce
MapReduce

- Large scale data processing framework
- Map phase + Reduce phase
- Proposed at 2004 by Google
- Variance and extension in open source community (Hadoop, Pig, Hive, etc.)
MapReduce Programming Model

- Programmers think in a data-centric fashion
  - Apply transformations on data sets
- The MR framework handles the Hard Stuff
  - Fault tolerance
  - Distributed execution, scheduling, concurrency
  - Coordination
  - Network communication
MapReduce System Model

- Designed for batch-oriented computations over large data sets
  - Each operator runs to completion before producing any output
  - Operator output is written to stable storage
  - Map output to local disk, reduce output to HDFS
- Simple, elegant fault tolerance model: operator restart
  - Critical for large clusters
'Hello World'

- Word Count

  - map(doc_id, text)
    - → list(word, count)
  
  - reduce(word, list(count))
    - → list(sum_count)

- Combiner is optional
'Hello World'

```
input
index.html
about.html
index.html

map
(index.html, 1)
(about.html, 1)
(index.html, 1)

shuffle
(index.html, [1,1])
(about.html, [1])

reduce
(index.html, 2)
(about.html, 1)

output
index.html 2
about.html 1
```
Hadoop

- Hadoop MapReduce Execution Engine
  - Single master node, many worker nodes
  - Client submits a job to master node
  - Master splits each job into tasks (map/reduce), and assigns tasks to worker nodes

- Hadoop Distributed File System (HDFS)
  - Single name node, many data nodes
  - Files stored as large, fixed-size (e.g. 64MB) blocks
  - HDFS typically holds map input and reduce output
Introduction to Pig
MapReduce not Good Enough?

- Restrict programming model
  - Only two phases
  - Job chain for long data flow
- Put the logic at the right phase
  - Programmers are responsible for this
- Too many lines of code even for simple logic
  - How many lines do you have for word count?
Pig to Rescue

- High level dataflow language (Pig Latin)
  - Much simpler than Java
  - Simplify the data processing
- Put the operations at the appropriate phases
- Chains multiple MR jobs
Motivation by Example

- Suppose we have user data in one file, website data in another file.
- We need to find the top 5 most visited pages by users aged 18-25.
In Pig Latin

Users = load 'users' as (name, age);
Fltrd = filter Users by
    age >= 18 and age <= 25;
Pages = load 'pages' as (user, url);
Jnd = joinFltrdby name, Pages by user;
Grpd = groupJndbyurl;
Smmnd = foreachGrpdgenerate group,
    COUNT(Jnd) as clicks;
Srtld = orderSmmndby clicks desc;
Top5 = limitSrtld 5;
store Top5 into 'top5sites';
Pig runs over Hadoop

Job executes on cluster

Pig resides on user machine

User machine

Hadoop Cluster

No need to install anything extra on your Hadoop cluster.
How Pig is used in Industry

- At Yahoo, 70% MapReduce jobs are written in Pig
- Used to
  - Process web log
  - Build user behavior models
  - Process images
  - Data mining
- Also used by Twitter, Linkin, Ebay, AOL, etc.
View of Pig from outside
MapReduce vs. Pig

- MaxTemperature

| Year  | Temperature | Air Quality | ...
|-------|-------------|-------------|------
| 1998  | 87          | 2           | ...  |
| 1983  | 93          | 4           | ..   |
| 2008  | 90          | 3           | ...  |
| 2001  | 89          | 5           | ...  |
| 1965  | 97          | 4           | ...  |

SELECT Year, MAX(Temperature)
FROM Table1
WHERE AirQuality = 0|1|4|5|9
GROUPBY Year
In MapReduce
In Pig

-- max_temp.pig: Finds the maximum temperature by year
records = LOAD 'input/ncdc/micro-tab/sample.txt'
  AS (year:chararray, temperature:int, quality:int);
filtered_records = FILTER records BY temperature != 9999 AND
  (quality == 0 OR quality == 1 OR quality == 4 OR quality == 5 OR quality == 9);
grouped_records = GROUP filtered_records BY year;
max_temp = FOREACH grouped_records GENERATE group,
  MAX(filtered_records.temperature);
DUMP max_temp;
Wait a minute

- How to map the data to records
  - By default, one line → one record
  - User can customize the loading process
- How to identify attributes and map them to schema
  - Delimiter to separate different attributes
  - By default, delimiter is tab. Customizable.
MapReduce vs. Pig cont.

- Join in MapReduce
  - Various algorithms. None of them are easy to implement in MapReduce
  - Multi-way join more complicated
  - Hard to integrate into SPJA workflow
MapReduce vs. Pig cont.

- **Join in Pig**
  - Various algorithms already available.
  - Some of them are generic to support multi-way join
  - No need to consider integration into SPJA workflow. Pig does that for you!

```
A = LOAD 'input/join/A';
B = LOAD 'input/join/B';
C = JOIN A BY $0, B BY $1;
DUMP C;
```
Pig Latin

- Data flow language
  - User specify a sequence of operations to process data
  - More control on the process, compared with declarative language
- Various data types supports
- Schema supports
- User defined functions supports
Statement

- A statement represents an operation, or a stage in the data flow.
- Usually a variable is used to represent the result of the statement
- Not limited to data processing operations, but also contains filesystem operations
Schema

- User can optionally define the schema of the input data
- Once the schema of the source data is given, all the schema of the intermediate relation will be induced by Pig
Why schema?
- Scripts are more readable (by alias)
- Help system validate the input

Similar to Database?
- Yes. But schema here is optional
- Schema is not fixed for a particular dataset, but changable
Schema cont.

- **Schema 1**
  
  A = LOAD 'input/A' as (name:chararray, age:int);
  B = FILTER A BY age != 20;

- **Schema 2**
  
  A = LOAD 'input/A' as (name:chararray, age:chararray);
  B = FILTER A BY age != '20';

- **No Schema**
  
  A = LOAD 'input/A' ;
  B = FILTER A BY A.$1 != '20';
Date Types

- Every attribute can always interpreted as bytearray, without further type definition
- Simple data types
  - For each attribute
  - Defined by user in the schema
  - Int, double, chararray ...
- Complex data types
  - Usually contracted by relational operations
  - Tuple, bag, map
Date Types cont.

- **Type casting**
  - Pig will try to cast data types when meets type inconsistency.
  - Warning will be thrown if casting fails. Process still goes on

- **Validation**
  - Null will replace the inconvertable data type in type casting
  - User can tell a corrupted record by detecting whether a particular attribute is null
Date Types cont.

```
1950  0  1
1950  22 1
1950  e 1
1949  111 1
1949  78 1

grunt> records = LOAD 'input/ncdc/micro-tab/sample_corrupt.txt'
>> AS (year:chararray, temperature:int, quality:int);
grunt> DUMP records;
(1950,0,1)
(1950,22,1)
(1950,,1)
(1949,111,1)
(1949,78,1)

grunt> corrupt_records = FILTER records BY temperature is null;
grunt> DUMP corrupt_records;
(1950,,1)
```
Operators

- Relational Operators
  - Represent an operation that will be added to the logical plan
  - LOAD, STORE, FILTER, JOIN, FOREACH...GENERATE

```
grunt> DUMP A;
  (2,Tie)
  (4,Coat)
  (3,Hat)
  (1,Scarf)
grunt> DUMP B;
  (Joe,2)
  (Hank,4)
  (Ali,0)
  (Eve,3)
  (Hank,2)

grunt> C = JOIN A BY $0, B BY $1;
grunt> DUMP C;
  (2,Tie,Joe,2)
  (2,Tie,Hank,2)
  (3,Hat,Eve,3)
  (4,Coat,Hank,4)
```
Operators

- Diagnostic Operators
  - Show the status/metadata of the relations
  - Used for debugging
  - Will not be integrated into execution plan
  - DESCRIBE, EXPLAIN, ILLUSTRATE.

```
grunt> records = LOAD 'input/ncdc/micro-tab/sample.txt'
>> AS (year, temperature:int, quality:int);
grunt> DESCRIBE records;
records: {year: bytearray,temperature: int,quality: int}
```
Functions

- Eval Functions
  - Record transformation

- Filter Functions
  - Test whether a record satisfy particular predicate

- Comparison Functions
  - Impose ordering between two records. Used by ORDER operation

- Load Functions
  - Specify how to load data into relations

- Store Functions
  - Specify how to store relations to external storage
Functions

- **Built in Functions**
  - Hard-coded routines offered by Pig.

- **User Defined Function (UDF)**
  - Supports customized functionalities
  - Piggy Bank, a warehouse for UDFs.
  - Re-think about Word Count in Pig
View of Pig from inside
Pig Execution Modes

- Local mode
  - Launch single JVM
  - Access local file system
  - No MR job running

- Hadoop mode
  - Execute a sequence of MR jobs
  - Pig interacts with Hadoop master node
Compilation

Pig Latin Programs → Query Parser → Logical Plan

Semantic Checking → Logical Plan

Logical Optimizer → Optimized Logical Plan

Logical to Physical Translator → Physical Plan

Physical To M/R Translator → MapReduce Plan

Map Reduce Launcher

Create a job jar to be submitted to hadoop cluster
Parsing

- Type checking with schema
- References verification
- Logic plan generating
  - One-to-one fashion
  - Independent of execution platform
  - Limited optimization
  - No execution until DUMP or STORE
A=LOAD 'file1' AS (x, y, z);
B=LOAD 'file2' AS (t, u, v);
C=FILTER A by y > 0;
D=JOIN C BY x, B BY u;
E=GROUP D BY z;
F=FOREACH E GENERATE group, COUNT(D);
STORE F INTO 'output'
Physical Plan

- 1:1 correspondence with most logical operators
- Except for:
  - DISTINCT
  - (CO)GROUP
  - JOIN
  - ORDER
Physical Optimizations

- Always use combiner for pre-aggregation
- Insert SPLIT to re-use intermediate result
- Early projection
MapReduce Plan

- Determine MapReduce boundaries
  - GLOBAL REARRANGE
  - STORE/LOAD
- Some operations are done by MapReduce framework
- Coalesce other operators into Map & Reduce stages
- Generate job jar file
Execution in Hadoop Mode

- The MR jobs depending on nothing in the MR plan will be submitted for execution
- MR jobs will be removed from MR plan after completion
  - Depending jobs are now available for execution
- Currently, no supports for inter-job fault-tolerance
Performance and future improvement
Pig Performance

Pig Performance vs Map-Reduce

Images from http://wiki.apache.org/pig/PigTalksPapers
Future Improvements

- Query optimization
  - Currently rule-based optimizer for plan rearrangement and join selection
  - Cost-based in the future

- Non-Java UDFs

- Grouping and joining on pre-partitioned/sorted data
  - Avoid data shuffling for grouping and joining
  - Building metadata facilities to keep track of data layout

- Skew handling
  - For load balancing
- Get more information at Pig website
- You can touch the source code to implement something new in Pig
- Also take a look at Hive, a similar system from Facebook
Some of the content come from the following presentations:

- Introduction to data processing using Hadoop and Pig, by Ricardo Varela
- Pig, Making Hadoop Easy, by Alan F. Gates
- Large-scale social media analysis with Hadoop, by Jake Hofman
- Getting Started on Hadoop, by Paco Nathan
- MapReduce Online, by Tyson Condie and Neil Conway