Other M/R Programming Interfaces

EN 600.423

Instructor: Randal Burns

12 March 2014

Department of Computer Science, *Johns Hopkins University*

PIG

- An alternative way to program analysis of large data sets that is declarative and ad-hoc

- Declarative programming is a programming paradigm that expresses the logic of a computation without describing its control flow.
  - in contrast to imperative programming which expresses an algorithm
- Ad-hoc: (adj)
  - Formed for or concerned with one specific purpose: an ad hoc compensation committee.
  - Improvised and often impromptu: “On an ad hoc basis, Congress has . . .

SQL (an aside)

- SQL is declarative
  - Users express queries using first order logic
  - System evaluates queries in different fashions depending upon the size and cardinality of the data, types, constraints, etc.
  - The program places no constraints on execution
- SQL is ad hoc
  - Allows arbitrary inspection, reuse of data, based on logical queries. The design of the database places no restriction on its future use.
  - This is not true of most data structures, e.g. priority heaps support only insert and retrieve greatest

So why not SQL?

- Well, there is SQL for Map/Reduce. That's HIVEQL.
- But, we'll look at PIG first.

C. Olston *et al.* Pig Latin: A Not-So-Foreign Language for Data Processing. SIGMOD, 2008.

- Databases too expensive and don't conform to how programmers think
- Map/reduce is too low level

pensive at this scale. Besides, many of the people who analyze this data are entrenched procedural programmers, who find the declarative, SQL style to be unnatural. The success of the more procedural *map-reduce* programming model, and its associated scalable implementations on commodity hardware, is evidence of the above. However, the map-reduce paradigm is too low-level and rigid, and leads to a great deal of custom user code that is hard to maintain, and reuse.

We describe a new language called *Pig Latin* that we have designed to fit in a sweet spot between the declarative style of SQL, and the low-level, procedural style of map-reduce.

More on what's wrong with M/R

- Practical efforts at MR programs end up with multi-phase confusing programs.
  - Need concepts such as joins

Unfortunately, the map-reduce model has its own set of limitations. Its one-input, two-stage data flow is extremely rigid. To perform tasks having a different data flow, e.g., joins or n stages, inelegant workarounds have to be devised. Also, custom code has to be written for even the most common operations, e.g., projection and filtering. These factors lead to code that is difficult to reuse and maintain, and in which the semantics of the analysis task are obscured. Moreover, the opaque nature of the map and reduce functions impedes the ability of the system to perform optimizations.

SQL versus PIG

- Example: find the average page rank of highly ranked pages in a big categories
- SQL is declarative

```
SELECT category, AVG(pagerank)
FROM urls WHERE pagerank > 0.2
GROUP BY category HAVING COUNT(*) > 10^6
```

- PIG describes a computation as a sequence of steps
  - Imperative style programming

```
good_urls = FILTER urls BY pagerank > 0.2;
groups = GROUP good_urls BY category;
big_groups = FILTER groups BY COUNT(good_urls)>10^6;
output = FOREACH big_groups GENERATE
            category, AVG(good_urls.pagerank);
```

Dataflow

- PIG programs are akin to describing a dataflow execution plan
  - And the people they likey:

"I much prefer writing in Pig [Latin] versus SQL. The step-by-step method of creating a program in Pig [Latin] is much cleaner and simpler to use than the single block method of SQL. It is easier to keep track of what your variables are, and where you are in the process of analyzing your data." – Jasmine Novak, Engineer, Yahoo!

```
CREATE TEMPORARY TABLE rbAUtmp select rbm.Matter_Identifier,
gpm.GP_PAM_Matter_Identifier, mbs.GP_PAM_Attorney_Identifier,
count(mbs.GP_PAM_Attorney_Identifier) as cnt, mbs.GP_PAM_Country_Identifer FROM
dbo_Matter_Billing_Statistics AS mbs JOIN dbo_Matter_GP_PAM_Matter_Identifier AS gpm ON
(mbs.GP_PAM_Matter_Identifier=gpm.GP_PAM_Matter_Identifier AND
mbs.GP_PAM_Country_Identifer=gpm.GP_PAM_Country_Identifier) JOIN rb_Matter as rbm ON
(rbm.Matter_Identifier=gpm.Matter_Identifier) GROUP BY mbs.GP_PAM_Attorney_Identifier
ORDER BY Matter_Identifier, cnt DESC;
  set @num := 0, @mid := '';
  CREATE TABLE rbMatAtt SELECT t1.Matter_Identifier, t2.Attorney_Identifier, t1.cnt,
@num:=if(@mid=Matter_Identifier, @num + 1, 1) AS row_number, @mid:=Matter_Identifier AS
dummy FROM rbAUtmp AS t1 LEFT OUTER JOIN dbo_Attorney_GP_PAM_Attorney_Identifier AS
t2 ON (t1.GP_PAM_Attorney_Identifier=t2.GP_PAM_Attorney_Identifier AND
t1.GP_PAM_Country_Identifer=t2.GP_PAM_Country_Identifier) ORDER BY t1.Matter_Identifier,
t1.cnt DESC;
  ALTER TABLE rbMatAtt ADD PRIMARY KEY (Matter_Identifier,row_number);
  CREATE TEMPORARY TABLE rbATT1 SELECT Matter_Identifier, Attorney_Identifier AS
ATT1_ID, Attorney_AVP_Salary_USD as ATT1_Salary, Attorney_Seniority AS ATT1_Seniority,
Attorney_PLS as ATT1_Specialty, Attorney_Location as ATT1_Location FROM rbMatAtt LEFT
OUTER JOIN rbdp_Attorney USING (Attorney_Identifier) WHERE row_number=1;
 .....
  CREATE TEMPORARY TABLE rbATT1j SELECT m.Matter_Identifier, ATT1_ID, ATT1_Salary,
ATT1_Seniority, ATT1_Specialty, ATT1_Location FROM rb_Matter AS m LEFT OUTER JOIN
rbATT1 AS a1 USING (Matter_Identifier);
  .........
  CREATE TABLE rbATTs SELECT * FROM rbATT1j JOIN rbATT2j USING (Matter_Identifier)
JOIN rbATT3j USING (Matter_Identifier);
```

PIG Language Constructs

- **FOREACH:** allows parallel processing (in a mapper) for all inputs in a data set
- **FILTER:** discard unwanted data (in either mapper or reducer)
- **GROUP/CO-GROUP:** put related data together using the shuffle process.

- **These constructs allow for database-style query optimization.**

PIG Data Model

- Atom: simple value
- Tuple: sequence of values
- Bag: multiset with duplicates
  - flexible schema for elements

$$\left\{ \begin{array}{c} (\texttt{`alice'}, \texttt{`lakers'}) \\ \big(\texttt{`alice'}, (\texttt{`iPod'}, \texttt{`apple'})\big) \end{array} \right\}$$

- Map: key/value data structure
  - Keys must be atoms for efficiency

$$\left[\begin{array}{c} \texttt{`fan of'} \to \left\{ \begin{array}{c} (\texttt{`lakers'}) \\ (\texttt{`iPod'}) \end{array} \right\} \\ \texttt{`age'} \to 20 \end{array} \right]$$

PIG Expressions

- Simple set that have to be parallelizable

$$\texttt{t} = \left(\text{'alice'}, \left\{ \begin{array}{c} (\text{'lakers'}, 1) \\ (\text{'iPod'}, 2) \end{array} \right\}, [\text{'age'} \rightarrow 20] \right)$$

Let fields of tuple `t` be called `f1, f2, f3`

Expression Type	Example	Value for t
Constant	'bob'	Independent of t
Field by position	$0	'alice'
Field by name	f3	['age' → 20]
Projection	f2.$0	{ ('lakers') ('iPod') }
Map Lookup	f3#'age'	20
Function Evaluation	SUM(f2.$1)	1 + 2 = 3
Conditional Expression	f3#'age'>18? 'adult':'minor'	'adult'
Flattening	FLATTEN(f2)	'lakers', 1 'iPod', 2

Table 1: Expressions in Pig Latin.

Bags and Co-Groupings

- Pig programming uses the patten of co-grouping data, applying aggregates, and then flattening the results
  - Allows SQL like functionality in sequenced programming
  - It's not super-intuitive (see the paper)

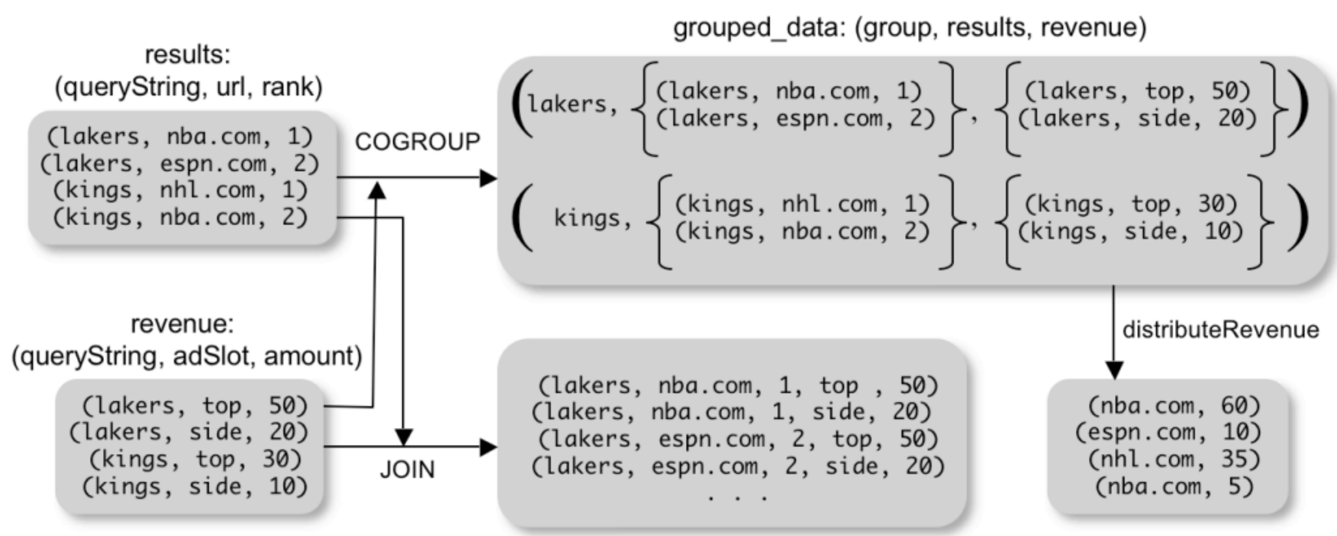


Figure 2: COGROUP **versus** JOIN**.**

Compiliing to MR

- Each PIG program compiles to several MR programs and is run in Hadoop!

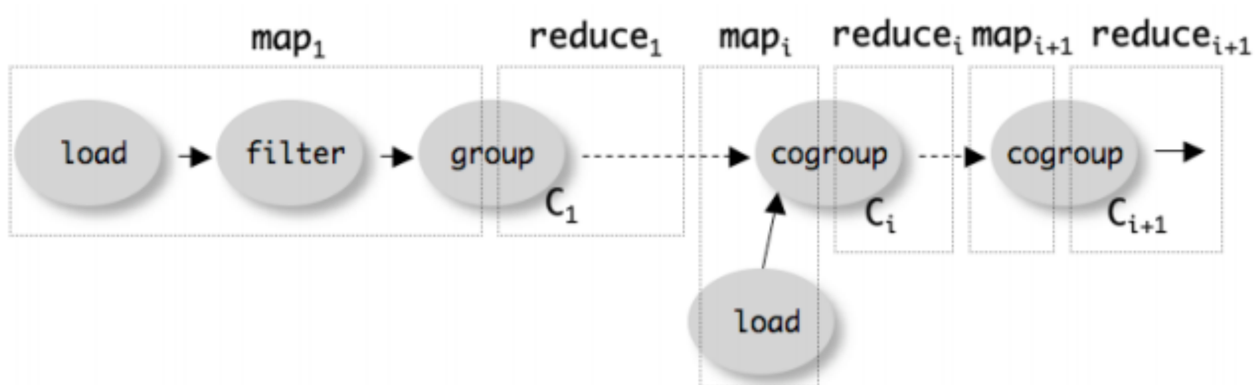


Figure 3: Map-reduce compilation of Pig Latin.

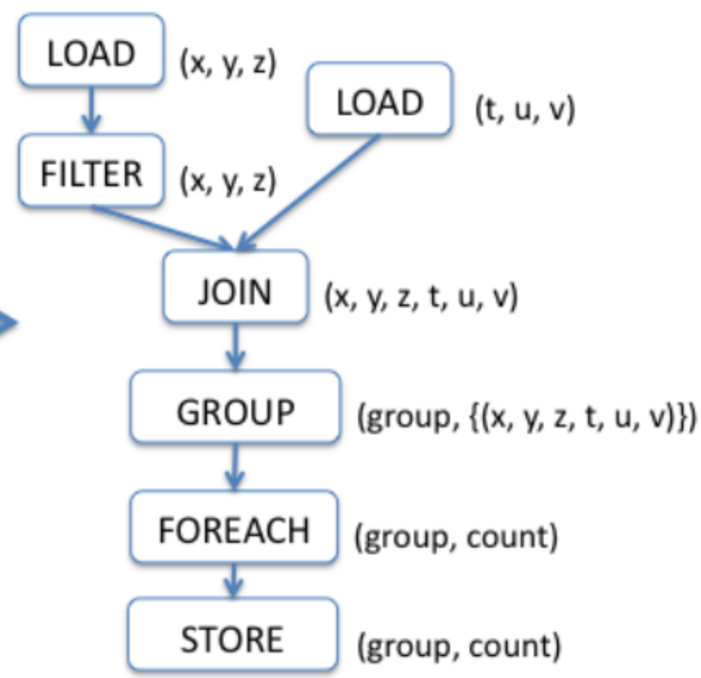
Compiling to MR (ii)

- From: Gates *et al. Building a High-Level Dataflow System on top of Map-Reduce: The Pig Experience, VLDB 2009.*

Pig Latin

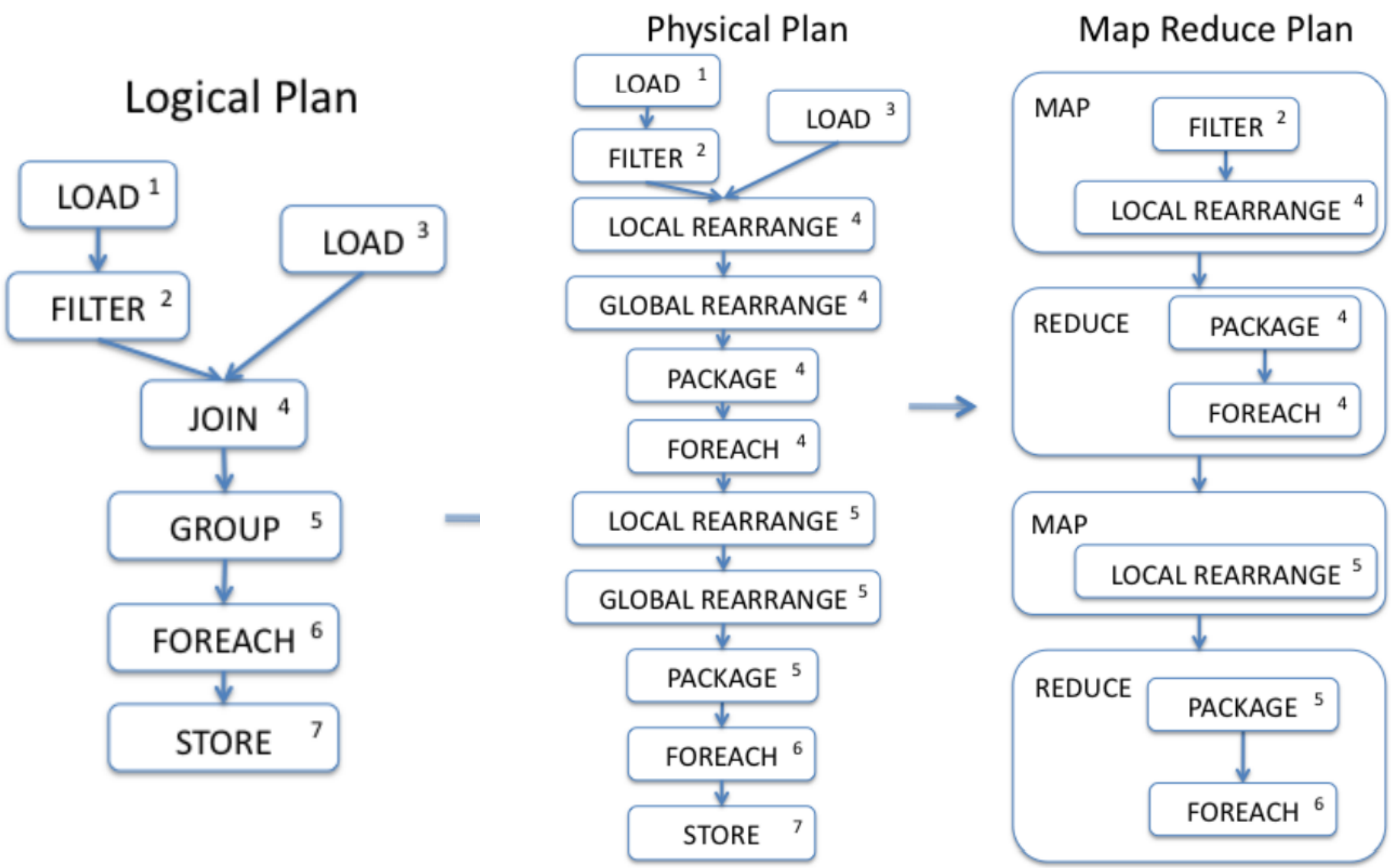
```
A = LOAD 'file1' AS (x, y, z);
B = LOAD 'file2' AS (t, u, v);
C = FILTER A by y > 0;
D = JOIN C BY x, B BY u;
E = GROUP D BY z;
F = FOREACH E GENERATE
      group, COUNT(D);
STORE F INTO 'output';
```

Logical Plan



Compiling to MR (ii)

- From: Gates *et al. Building a High-Level Dataflow System on top of Map-Reduce: The Pig Experience, VLDB 2009.*



JOHNS HOPKINS UNIVERSITY

Hive

- Hive: data model and system for data warehousing in map/reduce systems.
- HiveQL: SQL programming for Map/Reduce
  - Not SQL 92 complete
  - No transactions, no materialized views, limited subquery support
- The definitive Hive paper
  - Thusoo *et al.* Hive - A Warehousing Solution Over a Map-Reduce Framework. PVLDB, 2009.

Hive Example: Status Meme

- Table schema:

```
status_updates(userid int,status string,ds string)
```

- Load log files daily:

```
LOAD DATA LOCAL INPATH '/logs/status_updates'
INTO TABLE status_updates PARTITION (ds='2009-03-20')
```

Daily Statistics

- Join logs with profiles and figure out the number of tweets from men/women and by school

```
FROM (SELECT a.status, b.school, b.gender
      FROM status_updates a JOIN profiles b
           ON (a.userid = b.userid and
               a.ds='2009-03-20' )
      ) subq1
INSERT OVERWRITE TABLE gender_summary
                       PARTITION(ds='2009-03-20')
SELECT subq1.gender, COUNT(1) GROUP BY subq1.gender
INSERT OVERWRITE TABLE school_summary
                       PARTITION(ds='2009-03-20')
SELECT subq1.school, COUNT(1) GROUP BY subq1.school
```

How do it go?

- Hive puts tables on HDFS as files and runs queries as Hadoop! jobs

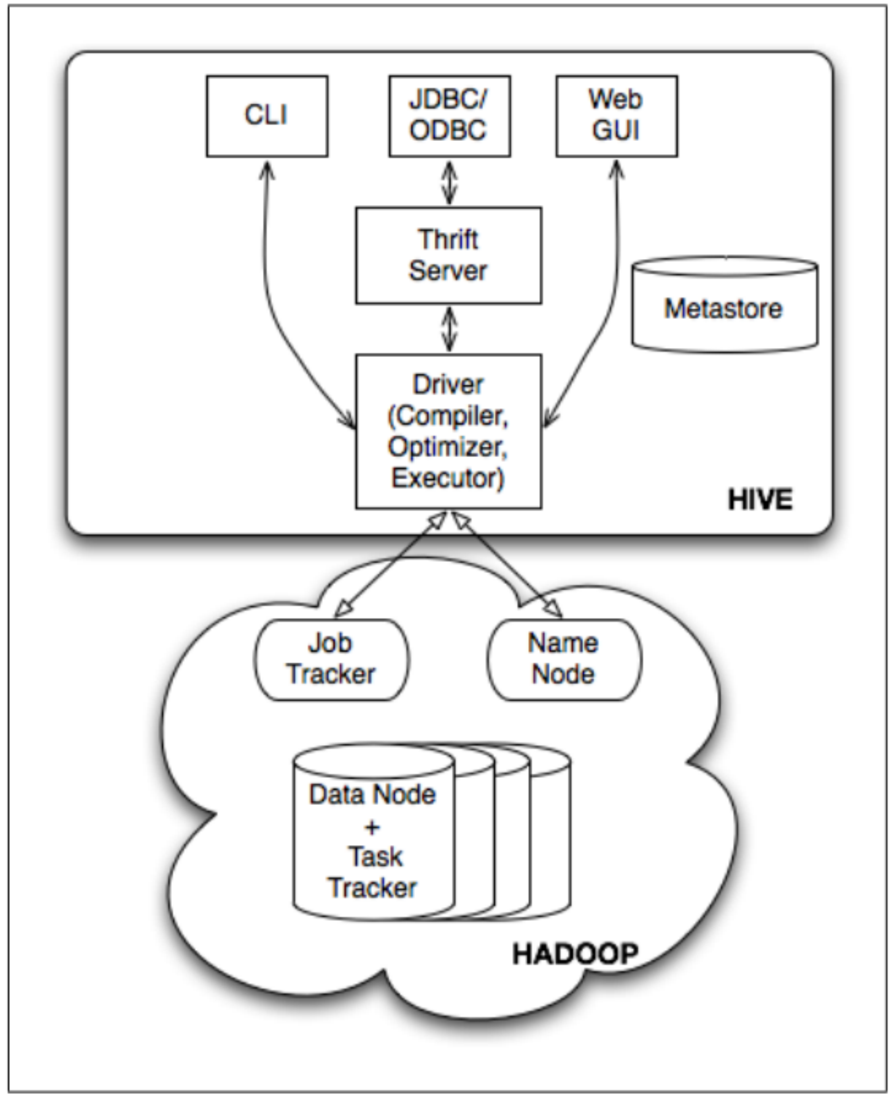
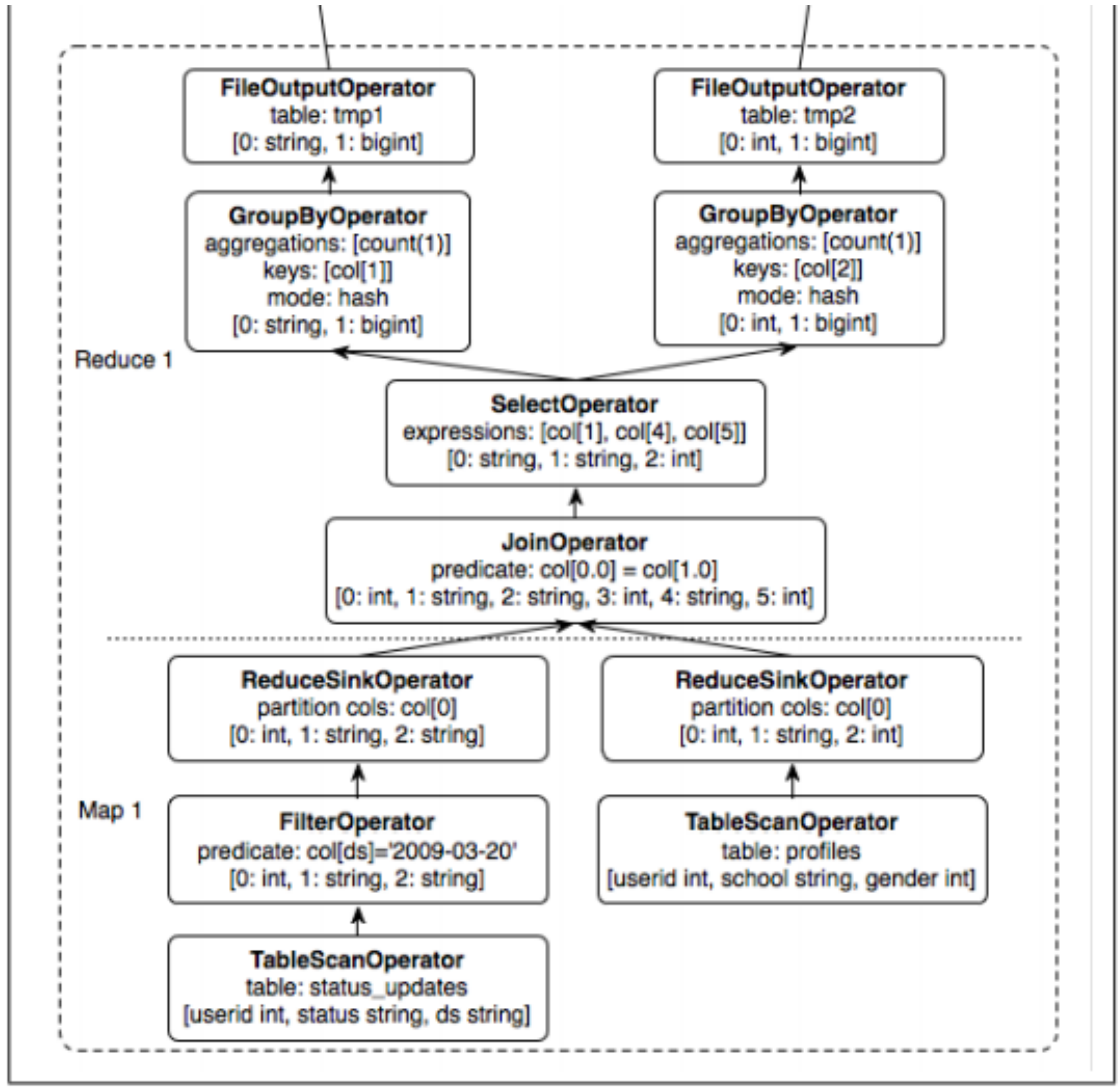


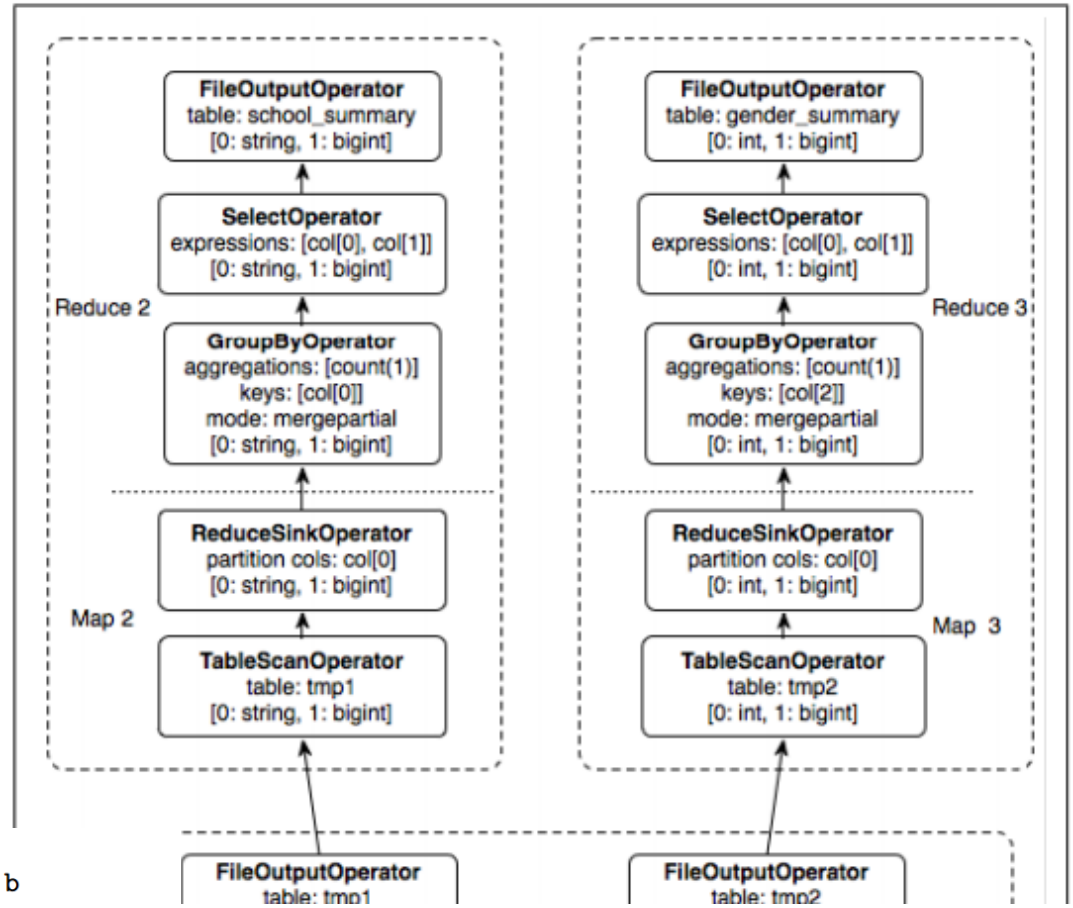
Figure 1: Hive Architecture

Resulting Query Plan (part 1)

- You don't need to understand. These are the MR jobs generated by the example



Resulting Query Plan (part 2)




```
FROM (SELECT a.status, b.school, b.gender
      FROM status_updates a JOIN profiles b
           ON (a.userid = b.userid and
               a.ds='2009-03-20' )
      ) subq1
INSERT OVERWRITE TABLE gender_summary
                       PARTITION(ds='2009-03-20')
SELECT subq1.gender, COUNT(1) GROUP BY subq1.gender
INSERT OVERWRITE TABLE school_summary
                       PARTITION(ds='2009-03-20')
SELECT subq1.school, COUNT(1) GROUP BY subq1.school
```

Take Aways

- Other ways to program M/R
  - More concise, easier to maintain
  - Particularly for data processing tasks that result in mutli-stage map/reduce programs
- Ethos: take the best from DBs
  - Declarative languages and optimization
  - Ad-hoc queries
- Ethos: and leave behind the stuff that's not parallel
  - Indexes, nested sub-queries

The (Un)Reasonable Debate

- Imperative programming
  - How humans think, step by step
  - Program encodes execution instructions
- Declarative programing
  - What! (Not how.)
  - Allows system to optimize execution
  - Non-intuitive (for many)
  - SQL != declarative programming. It is a specific instance that some love and some hate.
- PIG notable for trying to strike a happy balance
  - DB guys don't see the upside here