RAMP: A System for Capturing and Tracing Provenance in MapReduce Workflows

Hyunjung Park, Robert Ikeda, Jennifer Widom
Stanford University
MapReduce Workflow

- Directed acyclic graph composed of MapReduce jobs
- Popular for large-scale data processing
  - Hadoop
  - Higher-level platforms: Pig, Hive, Jaql, ...
- Debugging can be difficult
Provenance

- Where data came from, How it was processed, ...
  - Uses: drill-down, verification, debugging, ...

- RAMP: fine-grained provenance of data elements

- Backward tracing
  - Find the input subsets that contributed to a given output element

- Forward tracing
  - Determine which output elements were derived from a particular input element
Outline of Talk

- MapReduce Provenance: definition and examples
- RAMP System: overview and some details
- RAMP Experimental Evaluation
- Provenance-enabled Pig using RAMP
- Conclusion
MapReduce Provenance

- **Mapper**
  - $\mathbf{M}(I) = \bigcup_{i \in I} \mathbf{M}(\{i\})$
  - Provenance of an output element $o \in \mathbf{M}(I)$ is the input element $i \in I$ that produced $o$, i.e., $o \in \mathbf{M}(\{i\})$

- **Reducer (and Combiner)**
  - $\mathbf{R}(I) = \bigcup_{1 \leq j \leq n} \mathbf{R}(l_j)$ where $I_1, ..., I_n$ partition of $I$ on reduce key
  - Provenance of an output element $o \in \mathbf{R}(I)$ is the group $I_k \subseteq I$ that produced $o$, i.e., $o \in \mathbf{R}(I_k)$
MapReduce Provenance

- **MapReduce workflows**
  - Intuitive recursive definition

- “Replay” property
  - Replay the entire workflow with the provenance of an output element $o$
  - Does the result include the element $o$?
    
    Usually YES, but not always
MapReduce Provenance: Wordcount

Hyunjung Park

Mapper
Tokenizer

Combiner
IntSum

Reducer
IntSum

Apache Hadoop
Hadoop MapReduce
Apache Pig

Hadoop Summit
MapReduce Tutorial

(Hadoop, 1)
(Hadoop, 1)
(MapReduce, 1)
(Apache, 1)
(Pig, 1)

(Hadoop, 2)
(MapReduce, 1)
(Pig, 1)

(Hadoop, 2)
(Pig, 1)
(MapReduce, 2)
(Hadoop, 3)
(Summit, 1)
(Tutorial, 1)

(intSum)
(intSum)

Tokenizer
Mapper
Combiner
Reducer

(intSum)
(intSum)
Replay Property: Wordcount

Apache Hadoop
Hadoop MapReduce
Apache Pig

Hadoop Summit
MapReduce Tutorial

Mapper
Tokenizer

Combiner
IntSum

Reducer
IntSum

(Hadoop, 1)
(Hadoop, 1)
(MapReduce, 1)
(Apache, 1)
(Pig, 1)

(Hadoop, 1)
(Summit, 1)
(MapReduce, 1)
(Tutorial, 1)

(Hadoop, 1)
(Summit, 1)
(MapReduce, 1)
(Tutorial, 1)

(Apache, 1)
(Hadoop, 2)
(MapReduce, 1)
(Pig, 1)

(Apache, 1)
(Hadoop, 3)
(MapReduce, 1)
(Pig, 1)
(Summit, 1)
(Tutorial, 1)
System Overview

- Built as an extension (library) to Hadoop 0.21
  - Some changes in the core part as well

- Generic wrapper for provenance capture
  - Capture provenance for one MapReduce job at a time
  - Transparent
    - Provenance stored separately from the input/output data
    - Retains Hadoop’s parallel execution and fault tolerance
  - Wrapped components
    - RecordReader
    - Mapper, Combiner, Reducer
    - RecordWriter
System Overview

- Pluggable schemes for provenance capture
  - Element ID scheme
  - Provenance storage scheme (OutputFormat)
  - Applicable to any MapReduce workflow

- Provenance tracing program
  - Stand-alone
  - Depends on the pluggable schemes
Provenance Capture

Input -> RecordReader

(k^i, v^i)

Mapper

(k^m, v^m)

Map Output

Wrapper

(k^i, v^i)

Mapper

(k^m, (v^m, p))

Map Output
Provenance Capture

Map Output

Reducer

(k^o, v^o)

RecordWriter

(k^m, [v^m_1,...,v^m_n])

Output

Reducer

(k^m, [v^m_1,...,v^m_n])

Wrapper

Reducer

(k^o, v^o)

(k^o, ⟨v^o, k^m_ID⟩)

Wrapper

RecordWriter

(k^m_ID, p_j)

Output

Provenance

(q, k^m_ID, p_j)
Default Scheme for File Input/Output

- **Element ID**
  - (filename, offset)
  - Element ID increases as elements are appended
    - Reduce provenance stored in ascending key order
    - Efficient backward tracing without special indexes

- **Provenance storage**
  - Reduce provenance: offset $\rightarrow$ reduce group ID
  - Map provenance: reduce group ID $\rightarrow$ (filename, offset)

- **Compaction**
  - Filename: replaced by file ID
  - Integer ID, offset: variable-length encoded
Experimental Evaluation

- 51 large EC2 instances *(Thank you, Amazon!)*

- Two MapReduce “workflows”
  - Wordcount
    - Many-one with large fan-in
    - Input sizes: 100, 300, 500 GB
  - Terasort
    - One-one
    - Input sizes: 93, 279, 466 GB
Experimental Results: Wordcount

![Graph showing execution time, map finish time, and average reduce task time.]

- **Execution time**
  - 100G: 1200 seconds
  - 300G: 2400 seconds
  - 500G: 3600 seconds

- **Map finish time**
  - 100G: 1000 seconds
  - 300G: 2000 seconds
  - 500G: 3000 seconds

- **Avg. reduce task time**
  - 100G: 800 seconds
  - 300G: 1600 seconds
  - 500G: 2400 seconds

![Graph showing map output data size, intermediate data size, and output data size.]

- **Map output data size**
  - 100G: 100GB
  - 300G: 200GB
  - 500G: 300GB

- **Intermediate data size**
  - 100G: 50GB
  - 300G: 100GB
  - 500G: 150GB

- **Output data size**
  - 100G: 150GB
  - 300G: 300GB
  - 500G: 450GB
Experimental Results: Terasort

- **Execution time**
  - 93G: 0, 200, 400, 600, 800, 1000 seconds
  - 279G: 0, 200, 400, 600, 800, 1000 seconds
  - 466G: 0, 200, 400, 600, 800, 1000 seconds

- **Map finish time**
  - 93G: 0, 200, 400, 600, 800, 1000 seconds
  - 279G: 0, 200, 400, 600, 800, 1000 seconds
  - 466G: 0, 200, 400, 600, 800, 1000 seconds

- **Avg. reduce task time**
  - 93G: 0, 200, 400, 600, 800, 1000 seconds
  - 279G: 0, 200, 400, 600, 800, 1000 seconds
  - 466G: 0, 200, 400, 600, 800, 1000 seconds

- **Map output data size**
  - 93G: 93G
  - 279G: 279G
  - 466G: 466G

- **Intermediate data size**
  - 93G: 93G
  - 279G: 279G
  - 466G: 466G

- **Output data size**
  - 93G: 93G
  - 279G: 279G
  - 466G: 466G
Experimental Results: Summary

- **Provenance capture**
  - **Wordcount**
    - 76% time overhead, space overhead depends directly on fan-in
  - **Terasort**
    - 20% time overhead, 21% space overhead

- **Backward tracing**
  - **Wordcount**
    - 1, 3, 5 minutes (for 100, 300, 500 GB input sizes)
  - **Terasort**
    - 1.5 seconds
Instrumenting Pig for Provenance

- Can we run real-world MapReduce workflows on top of RAMP/Hadoop?

Pig 0.8
- Added (file, offset) based element ID scheme: ~100 LOC
- Default provenance storage scheme
- Default provenance tracing program
Input data sets
- Tweets collected in 2009
- 478 highest-grossing movie titles from IMDb

For each tweet:
- Infer a 1-5 overall sentiment rating
- Generate all $n$-grams and join them with movie titles

Output data set
- (Movie title, Rating, #Tweets in November, #Tweets in December)
Pig Script: Sentiment Analysis

```
raw_movie = LOAD 'movies.txt' USING PigStorage('\t') AS (title: chararray, year: int);
movie = FOREACH raw_movie GENERATE LOWER(title) as title;
raw_tweet = LOAD 'tweets.txt' USING PigStorage('\t') AS (datetime: chararray, url, tweet: chararray);
tweet = FOREACH raw_tweet GENERATE datetime, url, LOWER(tweet) as tweet;
rated = FOREACH tweet GENERATE datetime, url, tweet, InferRating(tweet) as rating;

MR #1
Movies → Lower

MR #2
Tweets → Lower → Infer Rating → Generate Ngram → Distinct

MR #3
Extract Month → GroupBy → Count

MR #4
σ → σ → Results

title_rating_month = FOREACH title_rating GENERATE title, rating, SUBSTRING(datetime, 5, 7) as month,
grouped = GROUP title_rating_month BY (title, rating, month);
title_rating_month_count = FOREACH grouped GENERATE flatten($0), COUNT($1);
november_count = FILTER title_rating_month_count BY month eq '11';
december_count = FILTER title_rating_month_count BY month eq '12';
outer_joined = JOIN november_count BY (title, rating) FULL OUTER, december_count BY (title, rating);
result = FOREACH outer_joined GENERATE (($0 is null) ? $4 : $0) as title, (($1 is null) ? $5 : $1) as rating, (($3 is null) ? 0 : $3) as november, (($7 is null) ? 0 : $7) as december;
STORE result INTO '/sentiment-analysis-result' USING PigStorage();
```
Backward Tracing: Sentiment Analysis

there been a lot of anticipation for #avatar? i just saw a trailer and was... disappointed clare that Avatar looks like the most generic sci fi flick ever. Coworker renamed it, "Dan a sneaking suspicion avatar is going to be great special effects and weak plot. boring Avatar was pretty osm. it had a really bad and overused plot but it looked osm & had e I've considered seeing Avatar before the school year starts, but I am not that impress ima2504 Avatar was really good... but u have to see it in 3D or else it will just be too la can't disguise bad story (and i mean, REALLY BAD) with pretty pictures. #avatar I want to go see #NewMoon and #Avatar again REALLY REALLY BAD! tar" is the dumbest, most boring, anti-American movie of 2009. Skip it and save the $12 an
Limitations

- Definition of MapReduce provenance
  - Reducer treated as a black-box
  - Many-many reducers
    - Provenance may contain more input elements than desired
    - E.g., identity reducer for sorting records with duplicates

- Wrapper-based approach for provenance capture
  - “Pure” mapper, combiner, and reducer
  - “Standard” input/output channels
Conclusion

- RAMP transparently captures provenance with reasonable time and space overhead
- RAMP provides convenient and efficient means of drilling-down and verifying output elements