Programming and Debugging Large-Scale Data Processing Workflows

Christopher Olston
Google Research

(work done at Yahoo! Research, with many colleagues)
Context

• Elaborate processing of large data sets
  
e.g.:
  • web search pre-processing
  • cross-dataset linkage
  • web information extraction
Context

Storage & Processing
- Workflow manager: e.g., Nova
- Dataflow programming framework: e.g., Pig
- Distributed sorting & hashing: e.g., Map-Reduce
- Scalable file system: e.g., GFS

Debugging Aides:
- Before: Example data generator
- During: Instrumentation framework
- After: Provenance metadata manager

Overview

Detail:
- Inspector Gadget, RubySky
Visits = load ‘/data/visits’ as (user, url, time);
Visits = foreach Visits generate user, Canonicalize(url), time;

Pages = load ‘/data/pages’ as (url, pagerank);

VP = join Visits by url, Pages by url;
UserVisits = group VP by user;
Sessions = foreach UserVisits generate flatten(FindSessions(*));
HappyEndings = filter Sessions by BestIsLast(*);

store HappyEndings into ‘/data/happy_endings’;
vs. map-reduce: less code!

"The [Hofmann PLSA E/M] algorithm was implemented in pig in 30-35 lines of pig-latin statements. Took a lot less compared to what it took in implementing the algorithm in Map-Reduce Java. Exactly that's the reason I wanted to try it out in Pig. It took 3-4 days for me to write it, starting from learning pig."

-- Prasenjit Mukherjee, Mahout project

![Bar chart showing 1/20 the lines of code and 1/16 the development time for Pig compared to Hadoop.](chart.png)

performs on par with raw Hadoop
vs. SQL:
step-by-step style;
lower-level control

"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!

"PIG seems to give the necessary parallel programming construct (FOREACH, FLATTEN, COGROUP .. etc) and also give sufficient control back to the programmer (which purely declarative approach like [SQL on Map-Reduce] doesn’t)."

-- Ricky Ho, Adobe Software
Conceptually:
A Graph of Data Transformations

Find users who tend to visit “good” pages.

Load Visits(user, url, time) → Transform to (user, Canonicalize(url), time) → Join url = url → Group by user → Transform to (user, Average(pagerank) as avgPR) → Filter avgPR > 0.5
“ILLUSTRATE lets me check the output of my lengthy batch jobs and their custom functions without having to do a lengthy run of a long pipeline. [This feature] enables me to be productive.”

-- Russell Jurney, LinkedIn
(Naïve Algorithm)

**Load**
Visits(user, url, time)

- (Amy, cnn.com, 8am)
- (Amy, http://www.snails.com, 9am)
- (Fred, www.snails.com/index.html, 11am)

**Transform**
to (user, Canonicalize(url), time)

- (Amy, www.cnn.com, 8am)
- (Amy, www.snails.com, 9am)
- (Fred, www.snails.com, 11am)

**Join**
url = url

**Group**
by user

**Transform**
to (user, Average(pagerank) as avgPR)

**Filter**
avgPR > 0.5

**Load**
Pages(url, pagerank)

- (www.youtube.com, 0.9)
- (www.frogs.com, 0.4)
Original UI Prototype (2008)

visits = LOAD 'visits.txt' AS (user, url, time);

pages = LOAD 'pages.txt' AS (url, pagerank);

v_p = JOIN visits BY url, pages BY url;

users = GROUP v_p BY user;

useravg = FOREACH users GENERATE group, AVG(v_p.pagerank) AS avgpr;

answer = FILTER useravg BY avgpr > '0.5';

visits: (Amy, cnn.com, 8am)
         (Amy, frogs.com, 9am)
         (Fred, snails.com, 11am)

pages: (cnn.com, 0.8)
        (frogs.com, 0.8)
        (snails.com, 0.3)

v_p: (Amy, cnn.com, 8am, cnn.com, 0.8)
     (Amy, frogs.com, 9am, frogs.com, 0.8)
     (Fred, snails.com, 11am, snails.com, 0.3)

users: (Amy, { (Amy, cnn.com, 8am, cnn.com, 0.8),
                  (Amy, frogs.com, 9am, frogs.com, 0.8) })
       (Fred, { (Fred, snails.com, 11am, snails.com, 0.3) })

useravg: (Amy, 0.8)
         (Fred, 0.3)

answer: (Amy, 0.8)
“Creators need an immediate connection to what they create … If you make a change … you need to see the effect of that immediately.”

-- Bret Victor, Apple
Pig Today

- Open-source (the Apache Pig Project)
  - Dev./support/training by Cloudera, Hortonworks
  - Offered on Amazon Elastic Map-Reduce
- Used by LinkedIn, Netflix, Salesforce, Twitter, Yahoo ...
- At Yahoo, as of early 2011:
  - 1000s of jobs/day
  - 75%+ of Hadoop jobs
- Has an interactive example-generator command, but no side-by-side UI 😞
Next: NOVA

Debugging aides:

- Before: example data generator
- During: instrumentation framework
- After: provenance metadata manager
Why a Workflow Manager?

- Continuous data processing (simulated on top of Pig/Hadoop static processing layer)

- Independent scheduling of workflow modules
Example Workflow

- RSS feed
  - NEW
- news articles
  - NEW
- template tagging
  - NEW
- shingling
  - NEW
- shingle hashes seen
  - ALL
  - NEW
- de-duping
  - NEW
- unique articles
  - NEW
- news site templates
  - ALL
  - NEW
- template detection
  - ALL
  - NEW

All seen

NEW
Data Passes Through Many Sub-Systems

- ingestion
- GFS
- Map-Reduce
- Pig
- Nova
- low-latency processor
- serving

datum X

datum Y

provenance of X?

metadata queries
Ibis Project

• Benefits:
  – Provide uniform view to users
  – Factor out metadata management code
  – Decouple metadata lifetime from data/subsystem lifetime

• Challenges:
  – Overhead of shipping metadata
  – Disparate data/processing granularities
Example Data and Process Granularities

**Data Granularities**
- Web page
- Cell
- Row
- Column
- Group
- Table

**Process Granularities**
- Workflow
- Pig script
- Pig logical operation
- Pig physical operation
- MR program
- Pig job
- MR job
- MR job phase
- MR task
- Task attempt
What’s Hard About Multi-Granularity Provenance?

- **Inference:** Given relationships expressed at one granularity, answer queries about other granularities (*the semantics are tricky here!*)

- **Efficiency:** Implement inference without resorting to materializing everything in terms of finest granularity (e.g. cells)
Inferring Transitive Links

map phase

IMDB web page
IMDB extracted table
<table>
<thead>
<tr>
<th>title</th>
<th>year</th>
<th>lead actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avatar</td>
<td>2009</td>
<td>Worthington</td>
</tr>
<tr>
<td>Inception</td>
<td>2010</td>
<td>DiCaprio</td>
</tr>
</tbody>
</table>

Yahoo! Movies web page
Yahoo extracted table
<table>
<thead>
<tr>
<th>title</th>
<th>year</th>
<th>lead actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avatar</td>
<td>2009</td>
<td>Saldana</td>
</tr>
<tr>
<td>Inception</td>
<td>2010</td>
<td>DiCaprio</td>
</tr>
</tbody>
</table>

map output 1
<table>
<thead>
<tr>
<th>title</th>
<th>year</th>
<th>lead actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avatar</td>
<td>2009</td>
<td>A1: Worthington</td>
</tr>
<tr>
<td>Inception</td>
<td>2010</td>
<td>DiCaprio</td>
</tr>
</tbody>
</table>

map output 2
<table>
<thead>
<tr>
<th>title</th>
<th>year</th>
<th>lead actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avatar</td>
<td>2009</td>
<td>A2: Saldana</td>
</tr>
<tr>
<td>Inception</td>
<td>2010</td>
<td>DiCaprio</td>
</tr>
</tbody>
</table>

combined table
<table>
<thead>
<tr>
<th>title</th>
<th>year</th>
<th>lead actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avatar</td>
<td>2009</td>
<td>A1: Worthington</td>
</tr>
<tr>
<td>Inception</td>
<td>2010</td>
<td>DiCaprio</td>
</tr>
</tbody>
</table>

Q: Which web page said “Saldana”?
Next: INSPECTOR GADGET

Debugging aides:

- Before: example data generator
- During: instrumentation framework
- After: provenance metadata manager

- workflow manager
  - *e.g.* Nova

- dataflow programming framework
  - *e.g.* Pig

- distributed sorting & hashing
  - *e.g.* Map-Reduce

- scalable file system
  - *e.g.* GFS
Motivated by User Interviews

• Interviewed 10 Yahoo dataflow programmers (mostly Pig users; some users of other dataflow environments)
• Asked them how they (wish they could) debug
<table>
<thead>
<tr>
<th># of requests</th>
<th>feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>crash culprit determination</td>
</tr>
<tr>
<td>5</td>
<td>row-level integrity alerts</td>
</tr>
<tr>
<td>4</td>
<td>table-level integrity alerts</td>
</tr>
<tr>
<td>4</td>
<td>data samples</td>
</tr>
<tr>
<td>3</td>
<td>data summaries</td>
</tr>
<tr>
<td>3</td>
<td>memory use monitoring</td>
</tr>
<tr>
<td>3</td>
<td>backward tracing (provenance)</td>
</tr>
<tr>
<td>2</td>
<td>forward tracing</td>
</tr>
<tr>
<td>2</td>
<td>golden data/logic testing</td>
</tr>
<tr>
<td>2</td>
<td>step-through debugging</td>
</tr>
<tr>
<td>2</td>
<td>latency alerts</td>
</tr>
<tr>
<td>1</td>
<td>latency profiling</td>
</tr>
<tr>
<td>1</td>
<td>overhead profiling</td>
</tr>
<tr>
<td>1</td>
<td>trial runs</td>
</tr>
</tbody>
</table>
Our Approach

• **Goal:** a programming framework for adding these behaviors, and others, to Pig

• **Precept:** avoid modifying Pig or tampering with data flowing through Pig

• **Approach:** perform Pig script rewriting – insert special UDFs that look like no-ops to Pig
Pig w/ Inspector Gadget
Example:

**Integrity Alerts**

- IG coordinator
- propagate alert to user
- alert!
- IG agent
- filter
- IG agent
- load
- IG agent
- join
- IG agent
- group
- IG agent
- count
- IG agent
- store
Example: Crash Culprit Determination

Phases 1 to n-1: record counts
Phase n: records

Phases 1 to n-1: maintain count lower bounds
Phase n: maintain last-seen records
Example:
Forward Tracing

- IG coordinator
  - report traced records to user
- IG agent
  - traced records
  - count
  - group
  - join
  - filter
  - load
  - IG agent
    - traced records
    - load
  - IG agent
    - traced records
    - load
  - IG agent
    - traced records
    - load
  - IG agent
    - traced records
    - load
## Agent & Coordinator APIs

<table>
<thead>
<tr>
<th>Agent Class</th>
<th>Agent Messaging</th>
<th>Coordinator Class</th>
<th>Coordinator Messaging</th>
</tr>
</thead>
<tbody>
<tr>
<td>init(args)</td>
<td>sendToCoordinator(message)</td>
<td>init(args)</td>
<td>sendToAgent(agentId, message)</td>
</tr>
<tr>
<td>tags = observeRecord(record, tags)</td>
<td>sendToAgent(agentId, message)</td>
<td>receiveMessage(source, message)</td>
<td>sendDownstream(message)</td>
</tr>
<tr>
<td>receiveMessage(source, message)</td>
<td>sendUpstream(message)</td>
<td>output = finish()</td>
<td>sendToAgent(agentId, message)</td>
</tr>
</tbody>
</table>
# Applications Developed For IG

<table>
<thead>
<tr>
<th># of requests</th>
<th>feature</th>
<th>lines of code (Java)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>crash culprit determination</td>
<td>141</td>
</tr>
<tr>
<td>5</td>
<td>row-level integrity alerts</td>
<td>89</td>
</tr>
<tr>
<td>4</td>
<td>table-level integrity alerts</td>
<td>99</td>
</tr>
<tr>
<td>4</td>
<td>data samples</td>
<td>97</td>
</tr>
<tr>
<td>3</td>
<td>data summaries</td>
<td>130</td>
</tr>
<tr>
<td>3</td>
<td>memory use monitoring</td>
<td>N/A</td>
</tr>
<tr>
<td>3</td>
<td>backward tracing (provenance)</td>
<td>237</td>
</tr>
<tr>
<td>2</td>
<td>forward tracing</td>
<td>114</td>
</tr>
<tr>
<td>2</td>
<td>golden data/logic testing</td>
<td>200</td>
</tr>
<tr>
<td>2</td>
<td>step-through debugging</td>
<td>N/A</td>
</tr>
<tr>
<td>2</td>
<td>latency alerts</td>
<td>168</td>
</tr>
<tr>
<td>1</td>
<td>latency profiling</td>
<td>136</td>
</tr>
<tr>
<td>1</td>
<td>overhead profiling</td>
<td>124</td>
</tr>
<tr>
<td>1</td>
<td>trial runs</td>
<td>93</td>
</tr>
</tbody>
</table>
RubySky: Scripting Framework with Built-in Debugging Support

• Scenario: *ad-hoc* data analysis
  – e.g. we just did a crawl; is it too biased toward Yahoo content?

• In this scenario, sometimes using new data; *always* running new code
  → Bugs are the norm
  → Debugging should be a first class citizen!
RubySky Example

:Crawl_data <= load(context, "'/crawl_data'", schema {string : url; ...})

# quick-and-dirty determination of "domain" from "url"
# (e.g. http://www.yahoo.com/index.html => yahoo.com)
:With_domains <= Crawl_data.foreach() do |r|
  url = r[:url]
  prefix = "http://www."
  if (url[0, prefix.len] == prefix)
    domain = url[prefix.len, url.index('/') - prefix.len]
  else
    throw "Don't know how to parse this url: " + url
  end
  Tuple.new([url, domain])
end

:Yahoo_only <= With_domains.filter do |r|
  r[:domain] =~ /=^yahoo/= 
end

:Grouped <= Yahoo_only.group

:Grouped.each { |r| puts "NUM. YAHOO URLs: " + r[:Yahoo_only].size }
RubySky Example: w/debug stmts

... With_domains <= Crawl_data.foreach() do |r|
  ...
  else
    domain = PUNT url
  end
  Tuple.new([url, domain])
end

# examine a few extracted domains, as sanity check
:Domain_sample <= With_domains.foreach() do |r|
  $count += 1
  ($count < SAMPLE_SIZE)? [Tuple.new([r[:domain]])] : []
end
SHOW Domain_sample

# alert if any extracted domain is empty or null
:Domain_missing <= With_domains.foreach() do |r|
  (r[:domain] == "" or r[:domain] == nil)?
    [Tuple.new(["ALERT! Missing domain from: " + r[:url]])] : []
end
SHOW Domain_missing

...
RubySky Execution

client

<table>
<thead>
<tr>
<th>launch</th>
</tr>
</thead>
<tbody>
<tr>
<td>load</td>
</tr>
<tr>
<td>extract domain</td>
</tr>
<tr>
<td>spy agent</td>
</tr>
<tr>
<td>filter</td>
</tr>
<tr>
<td>group</td>
</tr>
<tr>
<td>count</td>
</tr>
</tbody>
</table>

punt request

response (code and/or data)

domain samples

empty domain alerts

answer (# yahoo URLs)
Debugging aides:

- Before: example data generator
- During: instrumentation framework
- After: provenance metadata manager

Diagram:
- workflow manager: *e.g.*, Nova
- dataflow programming framework: *e.g.*, Pig
- distributed sorting & hashing: *e.g.*, Map-Reduce
- scalable file system: *e.g.*, GFS

Tools:
- Dataflow Illustrator
- Inspector Gadget, RubySky
- Ibis
Related Work

• **Pig**: DryadLINQ, Hive, Jaql, Scope, *relational query languages*

• **Nova**: BigTable, CBP, Oozie, Percolator, *scientific workflow, incremental view maintenance*

• **Dataflow illustrator**: [Mannila/Raiha, PODS’86], *reverse query processing, constraint databases, hardware verification & model checking*

• **Inspector gadget, RubySky**: XTrace, *taint tracking, aspect-oriented programming*

• **Ibis**: Kepler COMAD, ZOOM user views, *provenance management for databases & scientific workflows*