Data-Intensive Information Processing Applications — Session #7

MapReduce and databases

Jimmy Lin
University of Maryland

Tuesday, March 23, 2010
Today’s Agenda

- Role of relational databases in today’s organizations
  - Where does MapReduce fit in?
- MapReduce algorithms for processing relational data
  - How do I perform a join, etc.?
- Evolving roles of relational databases and MapReduce
  - What’s in store for the future?
Big Data Analysis

- Peta-scale datasets are everywhere:
  - Facebook has 2.5 PB of user data + 15 TB/day (4/2009)
  - eBay has 6.5 PB of user data + 50 TB/day (5/2009)
  - ...

- A lot of these datasets are (mostly) structured
  - Query logs
  - Point-of-sale records
  - User data (e.g., demographics)
  - ...

- How do we perform data analysis at scale?
  - Relational databases and SQL
  - MapReduce (Hadoop)
Relational Databases vs. MapReduce

- Relational databases:
  - Multipurpose: analysis and transactions; batch and interactive
  - Data integrity via ACID transactions
  - Lots of tools in software ecosystem (for ingesting, reporting, etc.)
  - Supports SQL (and SQL integration, e.g., JDBC)
  - Automatic SQL query optimization

- MapReduce (Hadoop):
  - Designed for large clusters, fault tolerant
  - Data is accessed in “native format”
  - Supports many query languages
  - Programmers retain control over performance
  - Open source

Source: O’Reilly Blog post by Joseph Hellerstein (11/19/2008)
Database Workloads

- **OLTP (online transaction processing)**
  - Typical applications: e-commerce, banking, airline reservations
  - User facing: real-time, low latency, highly-concurrent
  - Tasks: relatively small set of “standard” transactional queries
  - Data access pattern: random reads, updates, writes (involving relatively small amounts of data)

- **OLAP (online analytical processing)**
  - Typical applications: business intelligence, data mining
  - Back-end processing: batch workloads, less concurrency
  - Tasks: complex analytical queries, often ad hoc
  - Data access pattern: table scans, large amounts of data involved per query
One Database or Two?

- Downsides of co-existing OLTP and OLAP workloads
  - Poor memory management
  - Conflicting data access patterns
  - Variable latency

- Solution: separate databases
  - User-facing OLTP database for high-volume transactions
  - Data warehouse for OLAP workloads
  - How do we connect the two?
OLTP/OLAP Architecture

OLTP → ETL (Extract, Transform, and Load) → OLAP
OLTP/OLAP Integration

- **OLTP database for user-facing transactions**
  - Retain records of all activity
  - Periodic ETL (e.g., nightly)

- **Extract-Transform-Load (ETL)**
  - Extract records from source
  - Transform: clean data, check integrity, aggregate, etc.
  - Load into OLAP database

- **OLAP database for data warehousing**
  - Business intelligence: reporting, ad hoc queries, data mining, etc.
  - Feedback to improve OLTP services
Business Intelligence

- Premise: more data leads to better business decisions
  - Periodic reporting as well as ad hoc queries
  - Analysts, not programmers (importance of tools and dashboards)

- Examples:
  - Slicing-and-dicing activity by different dimensions to better understand the marketplace
  - Analyzing log data to improve OLTP experience
  - Analyzing log data to better optimize ad placement
  - Analyzing purchasing trends for better supply-chain management
  - Mining for correlations between otherwise unrelated activities
OLTP/OLAP Architecture: Hadoop?

ETL (Extract, Transform, and Load)

What about here?

Hadoop here?
ETL Bottleneck

- Reporting is often a nightly task:
  - ETL is often slow: why?
  - What happens if processing 24 hours of data takes longer than 24 hours?

- Hadoop is perfect:
  - Most likely, you already have some data warehousing solution
  - Ingest is limited by speed of HDFS
  - Scales out with more nodes
  - Massively parallel
  - Ability to use any processing tool
  - Much cheaper than parallel databases
  - ETL is a batch process anyway!
MapReduce algorithms for processing relational data
Design Pattern: Secondary Sorting

- MapReduce sorts input to reducers by key
  - Values are arbitrarily ordered
- What if want to sort value also?
  - E.g., k → (v₁, r), (v₃, r), (v₄, r), (v₈, r)…
Secondary Sorting: Solutions

- **Solution 1:**
  - Buffer values in memory, then sort
  - Why is this a bad idea?

- **Solution 2:**
  - “Value-to-key conversion” design pattern: form composite intermediate key, \((k, v_1)\)
  - Let execution framework do the sorting
  - Preserve state across multiple key-value pairs to handle processing
  - Anything else we need to do?
Value-to-Key Conversion

Before

\[ k \rightarrow (v_1, r), (v_4, r), (v_8, r), (v_3, r) \ldots \]

Values arrive in arbitrary order...

After

\[ (k, v_1) \rightarrow (v_1, r) \]
\[ (k, v_3) \rightarrow (v_3, r) \]
\[ (k, v_4) \rightarrow (v_4, r) \]
\[ (k, v_8) \rightarrow (v_8, r) \]

Values arrive in sorted order...

Process by preserving state across multiple keys

Remember to partition correctly!
Working Scenario

- Two tables:
  - User demographics (gender, age, income, etc.)
  - User page visits (URL, time spent, etc.)
- Analyses we might want to perform:
  - Statistics on demographic characteristics
  - Statistics on page visits
  - Statistics on page visits by URL
  - Statistics on page visits by demographic characteristic
  - ...
Relational Algebra

- Primitives
  - Projection ($\pi$)
  - Selection ($\sigma$)
  - Cartesian product ($\times$)
  - Set union ($\cup$)
  - Set difference ($-$)
  - Rename ($\rho$)

- Other operations
  - Join ($\Join$)
  - Group by… aggregation
  - …
Projection

\[
\begin{align*}
R_1 & \quad \text{Rectangle} & \quad \text{Rectangle} & \quad \text{Rectangle} & \quad \text{Circle} \\
R_2 & \quad \text{Rectangle} & \quad \text{Rectangle} & \quad \text{Circle} \\
R_3 & \quad \text{Rectangle} & \quad \text{Square} & \quad \text{Circle} \\
R_4 & \quad \text{Rectangle} & \quad \text{Rectangle} & \quad \text{Circle} \\
R_5 & \quad \text{Rectangle} & \quad \text{Rectangle} & \quad \text{Circle}
\end{align*}
\]

\[\pi \quad \square \quad \bigcirc \]

\[
\begin{align*}
R_1 & \quad \text{Rectangle} & \quad \text{Circle} \\
R_2 & \quad \text{Square} & \quad \text{Circle} \\
R_3 & \quad \text{Rectangle} & \quad \text{Circle} \\
R_4 & \quad \text{Rectangle} & \quad \text{Circle} \\
R_5 & \quad \text{Rectangle} & \quad \text{Circle}
\end{align*}
\]
Projection in MapReduce

- Easy!
  - Map over tuples, emit new tuples with appropriate attributes
  - No reducers, unless for regrouping or resorting tuples
  - Alternatively: perform in reducer, after some other processing

- Basically limited by HDFS streaming speeds
  - Speed of encoding/decoding tuples becomes important
  - Relational databases take advantage of compression
  - Semistructured data? No problem!
Selection

\[ R_1, R_2, R_3, R_4, R_5 \] \[ \sigma \] \[ R_1, R_3 \]
Selection in MapReduce

- Easy!
  - Map over tuples, emit only tuples that meet criteria
  - No reducers, unless for regrouping or resorting tuples
  - Alternatively: perform in reducer, after some other processing

- Basically limited by HDFS streaming speeds
  - Speed of encoding/decoding tuples becomes important
  - Relational databases take advantage of compression
  - Semistructured data? No problem!
**Group by... Aggregation**

- Example: What is the average time spent per URL?
- In SQL:
  - `SELECT url, AVG(time) FROM visits GROUP BY url`
- In MapReduce:
  - Map over tuples, emit time, keyed by url
  - Framework automatically groups values by keys
  - Compute average in reducer
  - Optimize with combiners
Relational Joins

Source: Microsoft Office Clip Art
Relational Joins
Types of Relationships

Many-to-Many

One-to-Many

One-to-One
Join Algorithms in MapReduce

- Reduce-side join
- Map-side join
- In-memory join
  - Striped variant
  - Memcached variant
Reduce-side Join

- Basic idea: group by join key
  - Map over both sets of tuples
  - Emit tuple as value with join key as the intermediate key
  - Execution framework brings together tuples sharing the same key
  - Perform actual join in reducer
  - Similar to a “sort-merge join” in database terminology

- Two variants
  - 1-to-1 joins
  - 1-to-many and many-to-many joins
Reduce-side Join: 1-to-1

Map

<table>
<thead>
<tr>
<th>R1</th>
<th>R4</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2</td>
<td>S3</td>
</tr>
</tbody>
</table>

Reduce

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>S2</td>
</tr>
<tr>
<td>S3</td>
<td>R4</td>
</tr>
</tbody>
</table>

Note: no guarantee if R is going to come first or S
Reduce-side Join: 1-to-many

Map

Reduce

What’s the problem?
Reduce-side Join: V-to-K Conversion

In reducer...

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₁</td>
<td></td>
</tr>
<tr>
<td>S₂</td>
<td></td>
</tr>
<tr>
<td>S₃</td>
<td></td>
</tr>
<tr>
<td>S₉</td>
<td></td>
</tr>
<tr>
<td>R₄</td>
<td></td>
</tr>
<tr>
<td>S₃</td>
<td></td>
</tr>
<tr>
<td>S₇</td>
<td></td>
</tr>
</tbody>
</table>
Reduce-side Join: many-to-many

In reducer...

What’s the problem?
Map-side Join: Basic Idea

Assume two datasets are sorted by the join key:

A sequential scan through both datasets to join (called a “merge join” in database terminology)
Map-side Join: Parallel Scans

- If datasets are sorted by join key, join can be accomplished by a scan over both datasets

- How can we accomplish this in parallel?
  - Partition and sort both datasets in the same manner

- In MapReduce:
  - Map over one dataset, read from other corresponding partition
  - No reducers necessary (unless to repartition or resort)

- Consistently partitioned datasets: realistic to expect?
In-Memory Join

- Basic idea: load one dataset into memory, stream over other dataset
  - Works if $R << S$ and $R$ fits into memory
  - Called a “hash join” in database terminology

- MapReduce implementation
  - Distribute $R$ to all nodes
  - Map over $S$, each mapper loads $R$ in memory, hashed by join key
  - For every tuple in $S$, look up join key in $R$
  - No reducers, unless for regrouping or resorting tuples
In-Memory Join: Variants

- Striped variant:
  - R too big to fit into memory?
  - Divide R into R₁, R₂, R₃, ... s.t. each Rₙ fits into memory
  - Perform in-memory join: ∀n, Rₙ ⋈ S
  - Take the union of all join results

- Memcached join:
  - Load R into memcached
  - Replace in-memory hash lookup with memcached lookup
Memcached

**Caching servers:** 15 million requests per second, 95% handled by memcache (15 TB of RAM)

**Database layer:** 800 eight-core Linux servers running MySQL (40 TB user data)

Source: Technology Review (July/August, 2008)
Memcached Join

Memcached join:
- Load R into memcached
- Replace in-memory hash lookup with memcached lookup

Capacity and scalability?
- Memcached capacity >> RAM of individual node
- Memcached scales out with cluster

Latency?
- Memcached is fast (basically, speed of network)
- Batch requests to amortize latency costs

Source: See tech report by Lin et al. (2009)
Which join to use?

- In-memory join > map-side join > reduce-side join
  - Why?
- Limitations of each?
  - In-memory join: memory
  - Map-side join: sort order and partitioning
  - Reduce-side join: general purpose
Processing Relational Data: Summary

- MapReduce algorithms for processing relational data:
  - Group by, sorting, partitioning are handled automatically by shuffle/sort in MapReduce
  - Selection, projection, and other computations (e.g., aggregation), are performed either in mapper or reducer
  - Multiple strategies for relational joins
- Complex operations require multiple MapReduce jobs
  - Example: top ten URLs in terms of average time spent
  - Opportunities for automatic optimization
Evolving roles for relational database and MapReduce
OLTP/OLAP/Hadoop Architecture

OLTP

ETL
(Extract, Transform, and Load)

Hadoop

OLAP

Why does this make sense?
Need for High-Level Languages

○ Hadoop is great for large-data processing!
  ● But writing Java programs for everything is verbose and slow
  ● Analysts don’t want to (or can’t) write Java

○ Solution: develop higher-level data processing languages
  ● Hive: HQL is like SQL
  ● Pig: Pig Latin is a bit like Perl
Hive and Pig

- **Hive:** data warehousing application in Hadoop
  - Query language is HQL, variant of SQL
  - Tables stored on HDFS as flat files
  - Developed by Facebook, now open source

- **Pig:** large-scale data processing system
  - Scripts are written in Pig Latin, a dataflow language
  - Developed by Yahoo!, now open source
  - Roughly 1/3 of all Yahoo! internal jobs

- **Common idea:**
  - Provide higher-level language to facilitate large-data processing
  - Higher-level language “compiles down” to Hadoop jobs
Hive: Example

- Hive looks similar to an SQL database
- Relational join on two tables:
  - Table of word counts from Shakespeare collection
  - Table of word counts from the bible

```sql
SELECT s.word, s.freq, k.freq FROM shakespeare s
JOIN bible k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1
ORDER BY s.freq DESC LIMIT 10;
```

<table>
<thead>
<tr>
<th>Word</th>
<th>Shakespeare Frequency</th>
<th>Bible Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>25848</td>
<td>62394</td>
</tr>
<tr>
<td>I</td>
<td>23031</td>
<td>8854</td>
</tr>
<tr>
<td>and</td>
<td>19671</td>
<td>38985</td>
</tr>
<tr>
<td>to</td>
<td>18038</td>
<td>13526</td>
</tr>
<tr>
<td>of</td>
<td>16700</td>
<td>34654</td>
</tr>
<tr>
<td>a</td>
<td>14170</td>
<td>8057</td>
</tr>
<tr>
<td>you</td>
<td>12702</td>
<td>2720</td>
</tr>
<tr>
<td>my</td>
<td>11297</td>
<td>4135</td>
</tr>
<tr>
<td>in</td>
<td>10797</td>
<td>12445</td>
</tr>
<tr>
<td>is</td>
<td>8882</td>
<td>6884</td>
</tr>
</tbody>
</table>

Source: Material drawn from Cloudera training VM
Hive: Behind the Scenes

SELECT s.word, s.freq, k.freq FROM shakespeare s
JOIN bible k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1
ORDER BY s.freq DESC LIMIT 10;

(Abstract Syntax Tree)

(one or more of MapReduce jobs)
Hive: Behind the Scenes

STAGE DEPENDENCIES:
Stage-1 is a root stage
Stage-2 depends on stages: Stage-1
Stage-0 is a root stage

STAGE PLANS:
Stage: Stage-1
Map Reduce
Alias -> Map Operator Tree:
  s
  TableScan
  alias: s
  Filter Operator
  predicate:
    expr: (freq >= 1)
    type: boolean
  Reduce Output Operator
  key expressions:
    expr: word
    type: string
    sort order: +
  Map-reduce partition columns:
    expr: word
    type: string
    tag: 0
  value expressions:
    expr: freq
    type: int
    expr: word
    type: string
  Reduce Operator Tree:
    Join Operator
    condition map:
    Inner Join 0 to 1
    condition expressions:
    0 (VALUE._col0) {VALUE._col1}
    1 (VALUE._col0)
    outputColumnNames: _col0, _col1, _col2
    Filter Operator
    predicate:
      expr: ((_col0 >= 1) and (_col2 >= 1))
      type: boolean
    Select Operator
    expressions:
      expr: _col1
      type: string
      expr: _col0
      type: int
      expr: _col2
      type: int
    outputColumnNames: _col0, _col1, _col2
    File Output Operator
    compressed: false
    GlobalTableId: 0
    table:
      input format: org.apache.hadoop.mapred.SequenceFileInputFormat
      output format: org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat

Stage: Stage-2
Map Reduce
Alias -> Map Operator Tree:
  hdfs://localhost:8022/tmp/hive-training/364214370/10002
  Reduce Output Operator
  key expressions:
    expr: _col1
    type: int
    sort order: -
    tag: -1
  value expressions:
    expr: _col0
    type: string
    expr: _col1
    type: int
    expr: _col2
    type: int
  Reduce Operator Tree:
  Extract
  Limit
  File Output Operator
  compressed: false
  GlobalTableId: 0
  table:
    input format: org.apache.hadoop.mapred.TextInputFormat
    output format: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat

Stage: Stage-0
Fetch Operator
limit: 10
Pig: Example

Task: Find the top 10 most visited pages in each category

<table>
<thead>
<tr>
<th>User</th>
<th>Url</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amy</td>
<td>cnn.com</td>
<td>8:00</td>
</tr>
<tr>
<td>Amy</td>
<td>bbc.com</td>
<td>10:00</td>
</tr>
<tr>
<td>Amy</td>
<td>flickr.com</td>
<td>10:05</td>
</tr>
<tr>
<td>Fred</td>
<td>cnn.com</td>
<td>12:00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Url</th>
<th>Category</th>
<th>PageRank</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnn.com</td>
<td>News</td>
<td>0.9</td>
</tr>
<tr>
<td>bbc.com</td>
<td>News</td>
<td>0.8</td>
</tr>
<tr>
<td>flickr.com</td>
<td>Photos</td>
<td>0.7</td>
</tr>
<tr>
<td>espn.com</td>
<td>Sports</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Pig Slides adapted from Olston et al. (SIGMOD 2008)
Pig Query Plan

Load Visits

Group by url

Foreach url
generate count

Load Url Info

Join on url

Group by category

Foreach category
generate top10(urls)

Pig Slides adapted from Olston et al. (SIGMOD 2008)
visits = load ‘/data/visits’ as (user, url, time);
gVisits = group visits by url;
visitCounts = foreach gVisits generate url, count(visits);
urlInfo = load ‘/data/urlInfo’ as (url, category, pRank);
visitCounts = join visitCounts by url, urlInfo by url;
gCategories = group visitCounts by category;
topUrls = foreach gCategories generate top(visitCounts,10);

store topUrls into ‘/data/topUrls’;

Pig Slides adapted from Olston et al. (SIGMOD 2008)
Pig Script in Hadoop

Map₁

Reduce₁

Join on url

Map₂

Reduce₂

Map₃

Reduce₃

Pig Slides adapted from Olston et al. (SIGMOD 2008)
Parallel Databases ↔ MapReduce

- Lots of synergy between parallel databases and MapReduce
- Communities have much to learn from each other
- Bottom line: use the right tool for the job!
Questions?