Large-scale Data Mining: 
MapReduce and beyond
Part 1: Basics

Spiros Papadimitriou
Rutgers University

original material developed jointly with Jimeng Sun (IBM) and Rong Yan (Facebook)

Data everywhere

- Flickr (8 billion photos)
- YouTube (83M videos, 16 hrs/min)
- Web (10B videos watched / mo.)
- Digital photos (500 billion / year)
- All broadcast (70,000TB / year)
- Yahoo! Webmap (9 trillion links, 300TB compressed, 5PB disk)
- Human genome (2-30TB uncomp.)

So what ??

Data everywhere

- Real-time access to content
- Richer context from users and hyperlinks
- Abundant training examples
- "Brute-force" methods may suffice

Challenges
- "Dirtier" data
- Efficient algorithms
- Scalability (with reasonable cost)
"The Google Way"

“All models are wrong, but some are useful”
– George Box

“All models are wrong, and increasingly you can succeed without them.” – Peter Norvig

- Google PageRank
- Shotgun gene sequencing
- Language translation
- ...

Getting over the marketing hype...

Cloud Computing = Internet + Commoditization/

“It’s what I and many others have worked towards our entire careers. It’s just happening now.”
– Eric Schmidt

This tutorial

- Is not about cloud computing
- But about large scale data processing

Data + Algorithms
Tutorial overview

- Part 1: Basic concepts & tools
  - MapReduce & distributed storage
  - Hadoop / HBase / Pig / Cascading / Hive

- Part 2: Applications & algorithms
  - Introduction
  - Information retrieval
  - Data warehousing
  - Graph algorithms
  - Clustering (k-means, co-clustering)
  - Classification (k-NN, naïve Bayes)

- Summary & resources

Outline

- Introduction
- MapReduce & distributed storage
  - Hadoop
    - HBase
    - Pig
    - Cascading
    - Hive
- Summary

What is MapReduce?

- Programming model?
  - “MapReduce” (this talk)
- Execution
  - Distributed computation + distributed storage + scheduling / fault tolerance
- Software package?

It’s all of those things, depending who you ask...
Q: "What is the frequency of each first name?"

```
employees.txt

<table>
<thead>
<tr>
<th>NAME</th>
<th>FIRST</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>John</td>
<td>$90,000</td>
</tr>
<tr>
<td>Brown</td>
<td>David</td>
<td>$70,000</td>
</tr>
<tr>
<td>Johnson</td>
<td>George</td>
<td>$90,000</td>
</tr>
<tr>
<td>Yates</td>
<td>John</td>
<td>$80,000</td>
</tr>
<tr>
<td>Miller</td>
<td>Bill</td>
<td>$60,000</td>
</tr>
<tr>
<td>Moore</td>
<td>Jack</td>
<td>$85,000</td>
</tr>
<tr>
<td>Taylor</td>
<td>Fred</td>
<td>$75,000</td>
</tr>
<tr>
<td>Smith</td>
<td>David</td>
<td>$80,000</td>
</tr>
<tr>
<td>Harris</td>
<td>John</td>
<td>$90,000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
```

Example – Programming model

```
def getName (line):
    return line.split(\t)[1]
def addCounts (hist, name):
    hist[name] = hist.get(name,default=0) + 1
    return hist

input = open(employees.txt, 'r')
intermediate = map(getName, input)
result = reduce(addCounts, intermediate, {})
```

```
def getName (line):
    return line.split(\t)[1], 1
def addCounts (hist, (name, c)):
    hist[name] = hist.get(name,default=0) + c
    return hist

input = open(employees.txt, 'r')
intermediate = map(getName, input)
result = reduce(addCounts, intermediate, {})
```

Example – Programming model

```
Hadoop / Java
```

```
public class HistogramJob extends Configured implements Tool {
    public static class FieldMapper extends MapReduceBase implements
      Mapper<LongWritable, Text, Text, LongWritable> {
        private static LongWritable ONE = new LongWritable(1);
        private static Text firstname = new Text();

        @Override
        public void map (LongWritable key, Text value,
                          OutputCollector<Text,LongWritable> out, Reporter r) {
            firstname.set(value.toString().split(\t)[1]);
            out.collect(firstname, ONE);
        } // class FieldMapper
    } // class FieldMapper
```
Example – Programming model
Hadoop / Java

```java
public static class LongSumReducer extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, LongWritable> {

    private static LongWritable sum = new LongWritable();

    @Override
    public void reduce (Text key, Iterator<LongWritable> vals,
                 OutputCollector<Text, LongWritable> out, Reporter r) {
        long s = 0;
        while (vals.hasNext())
            s += vals.next().get();
        sum.set(s);
        output.collect(key, sum);
    }
} // class LongSumReducer
```

Example – Programming model
Hadoop / Java

```java
public int run (String[] args) throws Exception {
    JobConf job = new JobConf(getConf(), HistogramJob.class);
    job.setJobName("Histogram");
    FileInputFormat.setInputPaths(job, args[0]);
    job.setMapperClass(FieldMapper.class);
    job.setCombinerClass(LongSumReducer.class);
    job.setReducerClass(LongSumReducer.class);
    JobClient.runJob(job);
    return 0;
} // run()

public static main (String[] args) throws Exception {
    ToolRunner.run(new Configuration(), new HistogramJob(), args);
} // main()
} // class HistogramJob
```

MapReduce for...

- Distributed clusters
  - Google's original
  - Hadoop (Apache Software Foundation)
- Hardware
  - SMP/CMP: Phoenix (Stanford)
- Other
  - Skynet (in Ruby/DRB)
  - QtConcurrent
  - BashReduce
  - … many more
Recap

Quick-n-dirty script xxx Hadoop
~5 lines of (non-boilerplate) code
Single machine, local drive Up to thousands of machines and drives

What is hidden to achieve this:
- Data partitioning, placement and replication
- Computation placement (and replication)
- Number of nodes (mappers / reducers)

As a programmer, you don't need to know what I'm about to show you next...

Execution model: Flow

Input file
SPLIT 0
SPLIT 1
SPLIT 2
SPLIT 3

Mapper
Reducer
Mapper
Reducer
Mapper
Reducer

Key/value iterators
Sequential scan
All-to-all, hash partitioning
Sort-merge

Output file
PART 0
PART 1

Execution model: Placement

HOST 0
SPLIT 0
SPLIT 1
SPLIT 2
SPLIT 3

HOST 1
SPLIT 0
SPLIT 1
SPLIT 2
SPLIT 3

HOST 2
SPLIT 0
SPLIT 1
SPLIT 2
SPLIT 3

HOST 3
SPLIT 0
SPLIT 1
SPLIT 2
SPLIT 3

HOST 4
SPLIT 0
SPLIT 1
SPLIT 2
SPLIT 3

HOST 5
SPLIT 0
SPLIT 1
SPLIT 2
SPLIT 3

HOST 6
SPLIT 0
SPLIT 1
SPLIT 2
SPLIT 3

Computation co-located with data (as much as possible)
Execution model: Placement

MapReduce Summary

- Simple programming model
- Scalable, fault-tolerant
- Ideal for (pre-)processing large volumes of data

"However, if the data center is the computer, it leads to the even more intriguing question “What is the equivalent of the ADD instruction for a data center?” [...] If MapReduce is the first instruction of the “data center computer”, I can’t wait to see the rest of the instruction set, as well as the data center programming language, the data center operating system, the data center storage systems, and more."


Outline

- Introduction
- MapReduce & distributed storage
  - Hadoop
    - HBase
    - Pig
    - Cascading
    - Hive
  - Summary
Hadoop

Hadoop's stated mission (Doug Cutting interview):
Commoditize infrastructure for web-scale,
data-intensive applications

Who uses Hadoop?

- Yahoo!
- Facebook
- Last.fm
- Rackspace
- Digg
- Apache Nutch

... more in part 3

Hadoop

Filesystems and I/O:
- Abstraction APIs
- RPC / Persistence

Core
Avro
Hadoop

- HBase
- Pig
- Hive
- ... 
- Distributed data warehouse
- SQL-like query language
- Data mgmt / query execution
- ZooKeeper
- Core
- Avro

Hadoop

- HBase
- Pig
- Hive
- ... 
- More
- MapReduce
- HDFS
- ZooKeeper
- Core
- Avro

MapReduce

- **Mapper:** \((k1, v1) \rightarrow (k2, v2)[]\)
  - E.g., \((\text{void}, \text{textline} : \text{string}) \rightarrow (\text{first} : \text{string}, \text{count} : \text{int})\)
- **Reducer:** \((k2, v2[]) \rightarrow (k3, v3)[]\)
  - E.g., \((\text{first} : \text{string}, \text{counts} : \text{int[]})) \rightarrow (\text{first} : \text{string}, \text{total} : \text{int})\)
- **Combiner:** \((k2, v2[]) \rightarrow (k2, v2)[]\)
- **Partition:** \((k2, v2) \rightarrow \text{int}\)
Mapper interface

```java
interface Mapper<K1, V1, K2, V2> {
    void configure (JobConf conf);
    void map (K1 key, V1 value,
              OutputCollector<K2, V2> out,
              Reporter reporter);
    void close();
}
```

- Initialize in `configure()`
- Clean-up in `close()`
- Emit via `out.collect(key,val)` any time

Reducer interface

```java
interface Reducer<K2, V2, K3, V3> {
    void configure (JobConf conf);
    void reduce (K2 key, Iterator<V2> values,
                 OutputCollector<K3, V3> out,
                 Reporter reporter);
    void close();
}
```

- Initialize in `configure()`
- Clean-up in `close()`
- Emit via `out.collect(key,val)` any time

Some canonical examples

- Histogram-type jobs:
  - Graph construction (bucket = edge)
  - K-means et al. (bucket = cluster center)
- Inverted index:
  - Text indices
  - Matrix transpose
- Sorting
- Equi-join
- More details in part 2
Equi-joins
"Reduce-side"

| [Smith, 7] | MAP | 7: n, (Smith) |
| [Jones, 7] | -OR- | 7: n, (Jones) |
| [Brown, 7] | -OR- | 7: n, (Brown) |
| [Davis, 3] | -OR- | 7: n, (Davis) |
| [Dukes, 5] | -OR- | 7: n, (Dukes) |
| [Black, 3] | -OR- | 7: n, (Black) |
| [Sales, 3] | -OR- | 7: n, (Sales) |
| [Devel, 7] | -OR- | 7: n, (Devel) |
| [Acct., 5] | -OR- | 7: n, (Acct.) |

Chaining
Typically more than one round, e.g.,

- Assign new labels
- Compute new centroids
- Assign new labels
- Compute new centroids
- SHUF.
- RED.
**HDFS & MapReduce processes**

**Hadoop Streaming & Pipes**
- Don’t have to use Java for MapReduce
- Hadoop Streaming:
  - Use stdin/stdout & text format
  - Any language (C/C++, Perl, Python, shell, etc)
- Hadoop Pipes:
  - Use sockets & binary format (more efficient)
  - C++ library required

**Outline**
- Introduction
- MapReduce & distributed storage
- Hadoop
  - HBase
  - Pig
  - Cascading
  - Hive
- Summary
HBase introduction

- MapReduce canonical example:
  - Inverted index (more in Part 2)

- Batch computations on large datasets:
  - Build static index on crawl snapshot

- However, in reality crawled pages are:
  - Updated by crawler
  - Augmented by other parsers/analytics
  - Retrieved by cache search
  - Etc...

HBase introduction

- MapReduce & HDFS:
  - Distributed storage + computation
  - Good for batch processing
  - But: no facilities for accessing or updating individual items

- HBase:
  - Adds random-access read / write operations
  - Originally developed at Powerset
  - Based on Google’s Bigtable

HBase data model

- Keys and cell values are arbitrary byte arrays
- Can use any underlying data store (local, HDFS, S3, etc)
- Partitioned over many nodes (thousands)
Data model example

```
profile: family

empId  profile: last  profile: first  profile: salary
Smith    John     $90,000
```

Always access via primary key

HBase cluster

```
Master

- Assign regions to servers
- Recover failures
- Manage assigned regions
- Client reads/writes
- Handle splits
```

Can use any underlying store (local, HDFS, KFS, S3, etc.)
HBase reads / writes

- **Writes:**
  - Append to commit log (on HDFS)
  - Also add to in-memory cache
  - Flush cache to filesystem when full
- **Reads:**
  - First check in-memory cache
  - Then check flush files
- **Background tasks:**
  - Flush file compaction
  - Region split

HBase vs. RDBMS

- Different solution, similar problems
- **RDBMSes:**
  - Row-oriented
  - Fixed-schema
  - ACID
- **HBase et al.:**
  - Designed from ground-up to scale out, by adding commodity machines
  - Simple consistency scheme: atomic row writes
  - Fault tolerance
  - Batch processing
  - No (real) indexes

Outline

- Introduction
- MapReduce & distributed storage
- Hadoop
  - HBase
  - Pig
  - Cascading
  - Hive
- Summary
Pig introduction

- "~5 lines of non-boilerplate code"
- Writing a single MapReduce job requires significant gruntwork
  - Boilerplates (mapper/reducer, create job, etc)
  - Input/output formats
- Many tasks require more than one MapReduce job

Pig main features

- Data structures (multi-valued, nested)
- Pig-latin: data flow language
  - SQL-inspired, but imperative (not declarative)

Pig example

records = LOAD filename 
  AS (last: chararray, first: chararray, salary: int);
grouped = GROUP records BY first;
counts = FOREACH grouped 
  GENERATE group, COUNT(records.first);
DUMP counts;

<table>
<thead>
<tr>
<th>employee</th>
<th>last</th>
<th>first</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Jane</td>
<td>$60,000</td>
<td></td>
</tr>
<tr>
<td>Brown</td>
<td>David</td>
<td>$50,000</td>
<td></td>
</tr>
<tr>
<td>Johnson</td>
<td>George</td>
<td>$90,000</td>
<td></td>
</tr>
<tr>
<td>Jones</td>
<td>John</td>
<td>$80,000</td>
<td></td>
</tr>
<tr>
<td>Miller</td>
<td>Bill</td>
<td>$60,000</td>
<td></td>
</tr>
<tr>
<td>Smith</td>
<td>David</td>
<td>$60,000</td>
<td></td>
</tr>
<tr>
<td>Taylor</td>
<td>Fred</td>
<td>$75,000</td>
<td></td>
</tr>
<tr>
<td>Smith</td>
<td>Harry</td>
<td>$80,000</td>
<td></td>
</tr>
<tr>
<td>Harris</td>
<td>John</td>
<td>$90,000</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Q: "What is the frequency of each first name?"
Pig schemas

- Schema = tuple data type
- Schemas are optional!
  - Data-loading step is not required
  - "Unknown" schema: similar to AWK ($0, $1, ..)
- Support for most common datatypes
- Support for nesting

Pig Latin feature summary

- Data loading / storing
  - LOAD / STORE / DUMP
- Filtering
  - FILTER / DISTINCT / FOREACH / STREAM
- Group-by
  - GROUP
- Join & co-group
  - JOIN / COGROUP / CROSS
- Sorting
  - ORDER / LIMIT
- Combining / splitting
  - UNION / SPLIT

Outline

- Introduction
- MapReduce & distributed storage
- Hadoop
  - HBase
  - Pig
  - Cascading
  - Hive
- Summary
Cascading introduction

- Provides higher-level abstraction
  - Fields, Tuples
  - Pipes
  - Operations
  - Taps, Schemes, Flows
- Ease composition of multi-job flows

Library, not a new language

Cascading example

```java
Scheme srcScheme = new TextLine();
Tap source = new Hfs(srcScheme, filename);
Scheme dstScheme = new TextLine();
Tap sink = new Hfs(dstScheme, filename, REPLACE);
Pipe assembly = new Pipe("lastnames");
Function splitter = new RegexSplitter(
    new Fields("last", "first", "salary"), 	"");
assembly = new Each(assembly, new Fields("line"), splitter);
assembly = new GroupBy(assembly, new Fields("first"));
Aggregator count = new Count(new Fields("count"));
assembly = new Every(assembly, count);
FlowConnector flowConnector = new FlowConnector();
Flow flow = flowConnector.connect("last-names",
    source, sink, assembly);
flow.complete();
```

<table>
<thead>
<tr>
<th>LAST</th>
<th>FIRST</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>John</td>
<td>$90,000</td>
</tr>
<tr>
<td>Brown</td>
<td>David</td>
<td>$70,000</td>
</tr>
</tbody>
</table>

Q: What is the frequency of each first name?

Cascading feature summary

- Pipes: transform streams of tuples
  - Each
  - GroupBy / CoGroup
  - Every
  - SubAssembly
- Operations: what is done to tuples
  - Function
  - Filter
  - Aggregator / Buffer
Outline

- Introduction
- MapReduce & distributed storage
- Hadoop
  - HBase
  - Pig
  - Cascading
  - Hive
- Summary

Hive introduction

- Originally developed at Facebook
  - Now a Hadoop sub-project
- Data warehouse infrastructure
  - Execution: MapReduce
  - Storage: HDFS files
- Large datasets, e.g. Facebook daily logs
  - 30GB (Jan'08), 200GB (Mar'08), 15+TB (2009)
- Hive QL: SQL-like query language

Hive example

```
CREATE EXTERNAL TABLE records
  (last STRING, first STRING, salary INT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE
LOCATION filename;

SELECT records.first, COUNT(1)
FROM records
GROUP BY records.first;
```

Q: "What is the frequency of each first name?"
Hive schemas

- Data should belong to tables
  - But can also use pre-existing data
  - Data loading optional (like Pig) but encouraged

- Partitioning columns:
  - Mapped to HDFS directories
  - E.g., (date, time) \(\rightarrow\) datadir/2009-03-12/18_30_00

- Data columns (the rest):
  - Stored in HDFS files

- Support for most common data types
- Support for pluggable serialization

Hive QL feature summary

- Basic SQL
  - FROM subqueries
  - JOIN (only equi-joins)
  - Multi GROUP BY
  - Multi-table insert
  - Sampling

- Extensibility
  - Pluggable MapReduce scripts
  - User Defined Functions
  - User Defined Types
  - SerDe (serializer / deserializer)

Outline

- Introduction
- MapReduce & distributed storage
- Hadoop
  - HBase
  - Pig
  - Cascading
  - Hive
- Summary
Recap

- Scalable: all
- High(-er) level: all except MR
- Existing language: MR, Cascading
- "Schemas": HBase, Pig, Hive, (Casc.)
  - Pluggable data types: all
- Easy transition: Hive, (Pig)

Related projects

Higher level—computation:
- Dryad & DryadLINQ (Microsoft) [EuroSys 2007]
- Sawzall (Google) [Sci Prog Journal 2005]

Higher level—storage:
- Bigtable [OSDI 2006] / Hypertable

Lower level:
- Kosmos Filesystem (Kosmix)
- VSN (Parascale)
- EC2 / S3 (Amazon)
- Ceph / Lustre / PanFS
- Sector / Sphere (http://sector.sf.net/)
- ...

Summary

MapReduce:
- Simplified parallel programming model

Hadoop:
- Built from ground-up for:
  - Scalability
  - Fault-tolerance
  - Clusters of commodity hardware
- Growing collection of components and extensions (HBase, Pig, Hive, etc)
Tutorial overview

- Part 1: Basic concepts & tools
  - MapReduce & distributed storage
  - Hadoop / HBase / Pig / Cascading / Hive
- Part 2: Applications & algorithms
  - Introduction
  - Information retrieval
  - Data warehousing
  - Graph algorithms
  - Clustering (k-means, co-clustering)
  - Classification (k-NN, naïve Bayes)
- Summary & resources

Large-scale Data Mining: MapReduce and Beyond
Part 1: Basics

Spiros Papadimitriou
Rutgers University

original material developed jointly with Jimeng Sun (IBM) and Rong Yan (Facebook)

Large-scale Data Mining: MapReduce and Beyond
Part 2: Applications & Algorithms

Spiros Papadimitriou
Rutgers University

original material developed jointly with Jimeng Sun (IBM) and Rong Yan (Facebook)
Part 2: Applications & Algorithms

- Introduction
- Applications & Algorithms
  - Data warehousing
  - Information retrieval: Basic tasks
  - Graphs: PageRank, Connect components
  - Clustering: k-means, Co-clustering
  - Classification: kNN, Naïve Bayes
- Summary & resources

MapReduce Applications in the Real World

<table>
<thead>
<tr>
<th>Organizations</th>
<th>Application of MapReduce</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google</td>
<td>Wide-range applications, grep / sorting, machine learning, clustering, report extraction, graph computation</td>
</tr>
<tr>
<td>Yahoo</td>
<td>Data model training, Web map construction, Web log processing using Pig, and much, much more</td>
</tr>
<tr>
<td>Amazon</td>
<td>Build product search index</td>
</tr>
<tr>
<td>Facebook</td>
<td>Web log processing via both MapReduce and Hive</td>
</tr>
<tr>
<td>PowerSet (Microsoft)</td>
<td>HBase for natural language search</td>
</tr>
<tr>
<td>Twitter</td>
<td>Web log processing using Pig</td>
</tr>
<tr>
<td>New York Times</td>
<td>Large-scale image conversion</td>
</tr>
<tr>
<td>Others (&gt;74)</td>
<td>Details in <a href="http://wiki.apache.org/hadoop/PoweredBy">http://wiki.apache.org/hadoop/PoweredBy</a> (so far, the longest list of applications for MapReduce)</td>
</tr>
</tbody>
</table>

Growth of MapReduce Applications in Google

- Example Use
  - Distributed grep
  - Term-vector per host
  - Document clustering
  - Statistical index
  - Web link reversal
  - Inverted index
  - Statistical translation

Growth of MapReduce Programs in Google Source Tree (2003 – 2006) (Implemented as C++ library)
MapReduce Goes Big: More Examples

- **Google**: >100,000 jobs submitted, 20PB data processed per day
  - Anyone can process tera-bytes of data w/o difficulties
- **Yahoo**: >100,000 CPUs in >25,000 computers running Hadoop
  - Biggest cluster: 4000 nodes (2*4 CPUs with 4*1TB disk)
  - Support research for Ad system and web search
- **Facebook**: 600 nodes with 4800 cores and ~2PB storage
  - Store internal logs and dimension user data

User Experience on MapReduce
Simplicity, Fault-Tolerance and Scalability

- **Google**: “completely rewrote the production indexing system using MapReduce in 2004” [Dean, OSDI’04]
  - Simpler code (Reduce 3800 C++ lines to 700)
  - MapReduce handles failures and slow machines
  - Easy to speedup indexing by adding more machines

- **Nutch**: “convert major algorithms to MapReduce implementation in 2 weeks” [Cutting, Yahoo!, 2005]
  - Before: several undistributed scalability bottlenecks, impractical to manage collections >100M pages
  - After: the system becomes scalable, distributed, easy to operate; it permits multi-billion page collections

MapReduce in Academic Papers

- 981 papers cite the first MapReduce paper [Dean & Ghemawat, OSDI’04]
  - Category: Algorithmic, cloud overview, infrastructure, future work
  - Company: Internet (Google, Microsoft, Yahoo…), IT (HP, IBM, Intel)
  - University: CMU, U. Penn, UC. Berkeley, UCF, U. of Missouri, …

- >10 research areas covered by algorithmic papers
  - Indexing & Parsing, Machine Translation
  - Information Extraction, Spam & Malware Detection
  - Ads analysis, Search Query Analysis
  - Image & Video Processing, Networking
  - Simulation, Graphs, Statistics, …

- 3 categories for MapReduce applications
  - Text processing: tokenization and indexing
  - Data warehousing: managing and querying structured data
  - Machine learning: learning and predicting data patterns
Machine Learning Applications: Examples

- Multimedia concept detection
- Machine translation
- Distributed co-clustering
- Social network analysis
- DNA sequence alignment
- Image / video clustering
- Spam & Malware Detection
- Advertisement Analysis

Application I: Multimedia Concept Detection

- Automatically categorize image / video into a list of semantic concepts using statistical learning methods
  - Foundations for several downstream use cases
- Apply MapReduce for multimedia concept detection
  - Learning methods: Random subspace bagging with SVMs

First Results: MapReduce-RSBag Scalability

- Results: speedup in mapping phase on 1, 2, 4, 8 and 16 nodes when learning 10 semantic concepts (>100GB features)
- Linear scalability on 1 – 4 nodes, but sub-linear on > 8 nodes
  - Hypothesis: Because of higher communication cost using more nodes? No.
  - Fact: The running time of our tasks varies a lot, but MapReduce assumes each map task takes similar time, Hadoop’s task scheduler is too simple.
Improve Scheduling Methods for Heterogeneous Tasks

- Goal: develop more effective offline task scheduling algorithms in presence of task heterogeneity
- Task Scheduling Approaches
  - Runtime modeling: predict the running time of task based on historical data
    \[ T(X,F) = t_0 + X^T P t_1 + e \]
  - Formulate the task scheduling problem as a multi-processor scheduling problem
  - Apply the Multi-Fit algorithm with First-Fit-Decreasing bin packing to find the shortest time to run all the tasks using a fixed number of nodes
- Results: significantly improve the balance between multiple tasks

Scalability Results w. Improved Task Scheduling

- Results: speedup in mapping phase on 1, 2, 4, 8 and 16 nodes when learning 10 semantic concepts (>100GB)
- Achieve considerably better scalability than Hadoop baseline results

Application 2: Machine Translation

- Formulation: translate foreign \( f \) into English \( e \)
  \[ \hat{e} = \arg\max P(f|e)P(e) \]
- MT Architecture [Lin & Dryer, Tutorial at NAACL-HLT 2005]
  - Two main components: word alignment & phrase extraction
Word Alignment Results
[Lin & Dryer, Tutorial at NAACL/HLT 2009]

Phrase Table Construction
[Lin & Dryer, Tutorial at NAACL/HLT 2009]

Data warehousing
Why use MapReduce for Data Warehouse?

- The amount of data you need to store, manage, and analyze is growing relentlessly
  - Facebook: >1PB raw data managed in database today
- Traditional data warehouses struggle to keep pace with this data explosion, also analytic depth and performance.
  - Difficult to scale to more than PB of data and thousands of nodes
  - Data mining can involve very high-dimensional problems with super-sparse tables, inverted indexes and graphs
- MapReduce: highly parallel data warehousing solution
  - AsterData SQL-MapReduce: up to 1PB on commodity hardware
  - Increases query performance by >9x over SQL-only systems

Status quo: Data Warehouse + MapReduce

Available MapReduce Software for Data Warehouse

- **Open Source:** Hive (http://wiki.apache.org/hadoop/Hive)
- **Commercial:** AsterData (SQL-MR), Greenplum, Teradata, Netezza, omr.sql (Oracle)

Huge Data Warehouses using MapReduce

- Facebook: multiple PBs using Hive in production
- eBay: 6.5PB database running on Greenplum
- Yahoo: >PB web/network events database using Hadoop
- MySpace: multi-hundred terabyte databases running on Greenplum and AsterData inCluster

HIVE: A Hadoop Data Warehouse Platform

Motivations
- Manage and query structured data using MapReduce
- Improve programmability of MapReduce
- Allow to publish data in well known schemas

Key building principles:
- MapReduce for execution, HDFS for storage
- SQL on structured data as a familiar data warehousing tool
- Extensibility – Types, Functions, Formats, Scripts
- Scalability, interoperability, and performance
Simplifying Hadoop based on SQL
[Thusoo, Hive ApacheCon 2008]

hive> select key, count(*) from kv1 where key > 100
     group by key;

VS.

$ cat > /tmp/reducer.sh
uniq -c | awk '($2"\t"$1)'
$ cat > /tmp/map.sh
awk -F " \001" '{if($1 > 100) print $1}'
$ bin/hadoop jar contrib/hadoop-0.19.2-dev-
    streaming.jar -input /user/hive/warehouse/kv1 -
    mapper map.sh -file /tmp/reducer.sh -file /tmp/
    map.sh -reducer reducer.sh -output /tmp/largekey -
    numReduceTasks 1
$ bin/hadoop dfs -cat /tmp/largekey/part*
Application 1: Centralized Reporting Tool

- Top-level site health metrics
- Bird’s-eye view of user growth by countries
- Comparing user activities by groups

Application 2
Lexicon: What are users talking about?

- Extract popular words from user generated content (UGC)
- Slice by age and region
- Sentiment analysis
- Keyword association
- Hadoop/Hive used

Lexicon: “Party” vs. “Hangover”
Taxes!

Hadoop Usage @ Facebook

- Data statistics (Jun. 2009):
  - Total Data: ~1.7PB
  - Cluster Capacity ~2.4PB
  - Net Data added/day: ~15TB
    - 6TB of uncompressed source logs
    - 4TB of uncompressed dimension data reloaded daily
  - Compression Factor ~5x (gzip, more with bzip)

- Usage statistics:
  - 3200 jobs/day with 800K tasks (map-reduce tasks)/day
  - 55TB of compressed data scanned daily
  - 15TB of compressed output data written to hdfs
  - 80 MM compute minutes/day

Thoughts: MapReduce for Database

- The strength of MapReduce is simplicity and scalability
  - No database system can come close to the performance of MapReduce infrastructure
  - RDBMSs cannot scale to that degree, not as fault-tolerant, ...
- Abstract ideas have been known before
  - "Mapreduce: A Major Step Backwards?", DeWitt and Stonebraker
  - Implement-able using user-defined aggregates in PostgreSQL
- MapReduce is very good at what it was designed for, but it may not be the one-fits-all solution
  - E.g. joins are tricky to do: MapReduce assumes a single input
Information retrieval using MapReduce

MapReduce Interface and Data Flow
Recap
- **Map**: (K1, V1) → list(K2, V2)
- **Combine**: (K2, list(V2)) → list(K2, V2)
- **Partition**: (K2, V2) → reducer_id
- **Reduce**: (K2, list(V2)) → list(K3, V3)

IR: Distributed Grep
- **Map**: (id, doc) → list(id, line#)
- **Reduce**: None
IR: URL Access Frequency

- Map: (null, log) \(\rightarrow\) list(URL, 1)
- Reduce: (URL, list(1)) \(\rightarrow\) (URL, total_count)

Also described in Part 1

IR: Reverse Web-Link Graph

- Map: (null, page) \(\rightarrow\) list(target, source)
- Reduce: (target, list(source)) \(\rightarrow\) (target, list(source))

It is the same as matrix transpose

IR: Inverted Index

- Map: (id, doc) \(\rightarrow\) list(word, id)
- Reduce: (word, list(id)) \(\rightarrow\) (word, list(id))
Text Indexing and Retrieval: Overview
[Lin & Dryer, Tutorial at NAACL/HLT 2009]

- Two stages: offline indexing and online retrieval
- Retrieval: sort documents by likelihood of documents
  - Estimate relevance between docs and queries
  - Sort and display documents by relevance
- Standard model: vector space model with TF.IDF weighting
  - Indexing: represent docs and queries as weight vectors

Similarity w. Inner Products
\[ \text{sim}(q, d) = \sum w_d \cdot w_q \]

TF.IDF indexing
\[ w_d = \frac{tf_d}{\log N} \]

MapReduce for Text Retrieval?

- Stage 1: Indexing problem
  - No requirement for real-time processing
  - Scalability and incremental updates important
- Stage 2: Retrieval problem
  - Require sub-second response
  - Only few retrieval results

MapReduce is not ideal for MapReduce application.

Nutch: MapReduce-based Web-scale search engine
Official site: http://lucene.apache.org/nutch/

- Doug Cutting, the creator of Hadoop, and Mike Cattarella founded in 2003
  - Map-Reduce / DFS → Hadoop
  - Content type detection → Tika
- Many installations in operation
  - >48 sites listed in Nutch wiki
  - Mostly vertical search
- Scalable to the entire web
  - Collections can contain 1M – 200M documents, webpages on millions of different servers, billions of pages
  - Complete crawl takes weeks
  - State-of-the-art search quality
  - Thousands of searches per second
Nutch Building Blocks: MapReduce Foundation
[Bialecki, ApacheCon 2009]

- **MapReduce**: central to the Nutch algorithms
  - Processing tasks are executed as one or more MapReduce jobs
- Data maintained as Hadoop SequenceFiles
  - Massive updates very efficient, small updates costly

Nutch in Practice

- Convert major algorithms to MapReduce in 2 weeks
- Scale from tens-million pages to multi-billion pages

Doug Cutting, Founder of Hadoop / Nutch

A scale-out system, e.g., Nutch/Lucene, could achieve a performance level on a cluster of blades that was not achievable on any scale-up computers, e.g., the Power5

Michael et al., IBM Research, IPDPS’07

---

Graph mining using MapReduce
PageRank

PageRank vector $q$ is defined as

$$ q = cA^Tq + \frac{1-c}{N}e $$

Where $A$ is the column normalized source-by-destination adjacency matrix,

$$ A = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} $$

$e$ is all one vector.

$N$ is the number of nodes

$c$ is the weight between 0 and 1 (e.g. 0.85)

PageRank indicates the importance of a page.

Algorithm: Iterative powering for finding the first eigen-vector

MapReduce: PageRank

Map: distribute PageRank $q_i$

Reduce: update new PageRank

Pegasus graph mining system

(G Kang et al. ICDM09)

GIM-V

Generalized Iterative Matrix-Vector Multiplication

Extension of plain matrix-vector multiplication includes as special cases

- Connected Components
- PageRank
- Random Walk With Restart (RWR)
- Diameter Estimation
Pegasus GIM-V: Intuition

- Plain M-V multiplication
  - Weighted Combination of Colors
  - Message Passing

\[ M \times \text{v} = \text{v}' \]

\[ v'_i = \sum m_{ij} v_j \]

Pegasus GIM-V: Main Idea

- Plain M-V multiplication

Three Implicit Operations here:
- multiply \( m_{ij} \) and \( v_j \)
- sum n multiplication results
- update \( v'_i \)

\[ M \times \text{v} = \text{v}' \]

\[ v'_i = \sum m_{ij} v_j \]

Pegasus GIM-V framework

- GIM-V
  - Adjacency Matrix
  - Vector represents some value of nodes
  - Customizing the three operations leads to many algorithms

<table>
<thead>
<tr>
<th>operations</th>
<th>Standard MV</th>
<th>Con. Cmpt.</th>
<th>PageRank</th>
<th>RWR</th>
<th>Diameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>map</td>
<td>Multiply</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>reduce</td>
<td>Sum</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>assign</td>
<td>Assign</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Pegasus GIM-V framework

**How to define the matrix vector multiplication operation?**

<table>
<thead>
<tr>
<th>Operations</th>
<th>Standard MV</th>
<th>Con. Cmpt.</th>
<th>PageRank</th>
<th>RWR</th>
<th>Diameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>map</td>
<td>Multiply</td>
<td>Multiply</td>
<td>Distribute pageRank</td>
<td>Multiply with c</td>
<td>Multiply with c</td>
</tr>
<tr>
<td>reduce</td>
<td>Sum</td>
<td>MIN</td>
<td>Aggregate pageRank</td>
<td>Sum with restart prob</td>
<td>BIT-OR()</td>
</tr>
<tr>
<td>assign</td>
<td>Assign</td>
<td>MIN</td>
<td>Assign</td>
<td>Assign</td>
<td>BIT-OR()</td>
</tr>
</tbody>
</table>

### Connected Components

**How many connected components?**

**Which node belong to which component?**

```
node id | component id
---|---
1 | 1
2 | 1
3 | 1
4 | 1
5 | 2
6 | 2
7 | 3
8 | 3
```

Input Graph

Output
Connected Components

- Design a proper matrix vector multiplication

Connected Components algorithm

- Naïve-Method (GIM-V BASE)

Input: Edge(src,dst) and Vector(id,val)
Map: create messages
Reduce: combine messages

Connected Components

- GIM-V for Connected Components

Map: \( x = m_j \times v_j \)
Reduce: \( v \rightarrow \text{MIN}(\text{all } x) \)
Assign: \( v_j' = \text{MIN}(v_j, v') \)

"Sending potential comp ID"
"Accept the Smallest"

Worst case # of iterations: diameter
Fast Algorithms For GIM-V

Block-Method (GIM-V BL)

Group matrix, vectors into blocks
Do the multiplication on blocks

Decrease Sorting Time
Decrease File Size

Performance

Running Time (Seconds)

Data: Kronecker 300 m edges

GiM-V BL-CL is best
Clustering using MapReduce

KMeans: Multi-pass clustering

**Traditional**

- **AssignClusters()**:
  - For each point p, Assign p to the closest c

- **UpdateCentroids()**:
  - For each cluster

**KMeans**

- While not converged:
  - AssignClusters()
  - UpdateCentroids()
MapReduce – KMeans

KmeansIter():
  Map(p) // Assign Cluster
  For c in clusters:
    If dist(p,c) < minDist,
      then minC = c, minDist = dist(p,c)
  Emit(minC.id, (p, 1))

Reduce() // Update Centroids
  For all values (p, c):
    total += p, count += c;
  Emit(key, (total, count))

Map and Reduce stages:
- **Map**: assigns each p to the closest centroids.
- **Reduce**: updates each centroid with its new location (total, count).

Distributed Co-Clustering
[Papadimitriou & Sun, ICDM’08]

Split:
  Increase k or f
Shuffle:
  Rearrange rows and columns

Splitting:
- k = 1, f = 1
- k = 2, f = 2
- k = 3, f = 3
- k = 4, f = 4
- k = 5, f = 5
(Co-)clustering with MapReduce

Key

Val

5 7 13
5 9 11 19 27
5 6 12

p1 p2 p3

3

2

1

2

3

4

5

(p) 1

(p) 2

(p) 3

R(1)

R(2)

R(3)

R(m)

Scalability of MapReduce Co-Clustering

Scales up to ~10-15 nodes.

But, at the moment, Hadoop implementation is sub-optimal for short jobs...

Aggregate bandwidth (Mbps)

Job length ≈ 20 ± 2 sec.

Sleep overhead ≥ 5 sec.

Scales with data volume.

But, at the moment, Hadoop implementation is sub-optimal for short jobs...
Classification using MapReduce

MapReduce kNN

Naive Bayes
MapReduce: Naïve Bayes

Goals:
1. Total number of docs N
2. Number of docs in class c: N_c
3. Word count histogram in class T_cw
4. Total word count in class: \( \sum T_cw \)

Naïve Bayes can be implemented using MapReduce jobs of histogram computation.

ClassPrior();
Map(doc);
Emit(class_id, (doc_id, doc_length))
Combine/Reduce();
N_c = 0, S_Tcw = 0
For each doc_id:
N_c += 1, S_Tcw += doc_length
Emit(N_c, S_Tcw)

ConditionalProbability();
Map(doc);
For each word w in doc:
Emit(pair(c, w), 1)
Combine/Reduce();
S_Tcw = 0
For each value v:
S_Tcw += v
Emit(pair(c, w), S_Tcw)

Summary & Resources

Practical Experience on MapReduce
[Dean, PACT’06 Keynote]

- Fine granularity tasks: map tasks (200K) >> nodes (2K)
  - Minimizes time for fault recovery
  - Can pipeline shuffling with map execution
  - Better dynamic load balancing
- Fault Tolerance: handled by re-execution
  - Lost 1600/1800 machines once → finished ok
- Speculative execution: spawn tasks when near to end
  - Avoid slow workers which significantly delay completion time
- Locality optimization: move the code to “data”
  - Thousands of machines read at local speed
- Multi-core: more effective than multi-processors
**Machine Learning w. MapReduce: Remarks**

- MapReduce is applicable to many scenarios
  - Convertible to MapReduce for summation-form algorithms
  - Suitable for algorithms with less iterations and large computational cost inside the loop
- No universally optimal parallelized methods
  - Tradeoff: Hadoop overhead and parallelization
  - Need algorithm design and parameter tuning for specific tasks
  - Goldilocks argument: it's all about the right-level abstraction
- Useful resources:
  - MR toolboxes: Apache Mahout
  - ICDM’09, Workshop on “Large-scale data mining”
  - ACM MM’09, Workshop on “Large-scale multimedia mining”
  - NIPS’09, Workshop on “Large-scale machine learning”

**Machine learning algorithms using MapReduce**

- Linear regression
- Logistic Regression
- Neural Networks
- PCA
- ICA
- EM for Gaussian Mixture Model

“Map-Reduce for Machine Learning on Multicore” [NIPS’06]

**Taxonomy of MapReduce mining algorithms**

```
<table>
<thead>
<tr>
<th>Method</th>
<th>Single Iteration</th>
<th>Multiple Iterations</th>
<th>Not good for MapReduce</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clustering</td>
<td>Canopy, KMeans</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Classification</td>
<td>Naive Bayes, KNN</td>
<td>Gaussian Mixture, SVM</td>
<td></td>
</tr>
<tr>
<td>Graphs</td>
<td>PageRank, Connected Components</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IR</td>
<td>Inverted Index, Topic modeling (PLSI, LDA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- Single iteration algorithms are perfect fit
- Multiple iteration algorithms are good fit, if
  - Small shared info has to be synchronized across iterations (typically through filesystem - HDFS)
  - Some algorithms are not good for MapReduce framework
  - Typically require large shared info with a lot of synchronization
  - Traditional parallel frameworks like MPI are better suited
MapReduce for machine learning algorithms

- The key is to convert into summation form (Statistical Query model [Kearns'94])
  - $y = \sum f(x)$ where $f(x)$ corresponds to map(), $\sum$ corresponds to reduce().

- Naïve Bayes
  - MR job: $P(c)$
  - MR job: $P(w|c)$

- KMeans
  - MR job: split data into subgroups and compute partial sum in Map(), then sum up in Reduce()

Summary: algorithms

- Best for MapReduce:
  - Single pass, keys are uniformly distributed

- Good for MapReduce:
  - Multiple pass, intermediate/shared state is small

- Bad for MapReduce:
  - Key distribution is skewed
  - Fine-grained synchronization is required

Summary: algorithms

<table>
<thead>
<tr>
<th>Favored Algorithms</th>
<th>Unfavored Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve Bayes</td>
<td>Perceptron</td>
</tr>
<tr>
<td>k Nearest Neighbor</td>
<td>AdaBoost</td>
</tr>
<tr>
<td>kMeans / EM</td>
<td>Support Vector Machine</td>
</tr>
<tr>
<td>Random Bagging</td>
<td>Logistic Regression</td>
</tr>
<tr>
<td>Gaussian Mixture</td>
<td>Spectral Clustering</td>
</tr>
<tr>
<td>Linear Regression</td>
<td></td>
</tr>
</tbody>
</table>

Bold: discussed

Few iterations & Long Inner-Loop Cycle

Many iterations & Short Inner-Loop Cycle
Future Research Opportunities
MapReduce for Data Mining

- **Algorithm** perspective
  - Convert known algorithms to their MapReduce version
  - Design descriptive language for MapReduce mining
  - Extend MapReduce primitives for data mining, such as multi-iteration MapReduce with data sharing

- **System** perspective
  - Improve MapReduce scalability for mining algorithms

---

Mahout: Hadoop data mining library

  - Scalable data mining libraries: mostly implemented in Hadoop
  - Data structures for vectors and matrices
    - Vectors
      - Dense vector as double[]
      - Sparse vectors as HashMap<Integer, Double>
      - Operations: assign, cardinality, copy, divide, dot, get, haveSharedCells, like, minus, normalize, plus, set, size, times, toArray, viewPart, zSum and cross
    - Matrices
      - Dense matrix as a double[][]
      - SparseRowMatrix or SparseColumnMatrix as Vector[], holding the rows or columns of the matrix in a SparseVector
      - SparseMatrix as a HashMap<Integer, Vector>
      - Operations: assign, assignColumn, assignRow, cardinality, copy, divide, get, haveSharedCells, like, minus, plus, set, size, times, transpose, toArray, viewPart and zSum

---

MapReduce Mining Papers

- Chu et al. NIPS'06: Map-Reduce for Machine Learning on Multicore
  - General framework under MapReduce
- Papadimitriou et al ICDM'08: DisCo: Distributed Co-clustering with MapReduce
  - Co-clustering
- Kang et al. ICDM'09: PEGASUS: A Peta-Scale Graph Mining System - Implementation and Observations
  - Graph algorithms
- [Das et al. WWW'07](http://www.citeulike.org/user/ktorg/react/article/1189890): Google news personalization: scalable online collaborative filtering
  - PLSI EM
  - Alternative to Hadoop which supports wide-area data collection and distribution
MapReduce Books

- “Pro Hadoop” by Jason Venner
  - Hadoop Version: 0.20
  - Publisher: Apress (2009)

- “Hadoop: The Definitive Guide” by Tom White
  - Hadoop Version: 0.20
  - Publisher: O’Reilly (2009)

Conclusions

- MapReduce: simplified parallel programming model
  - Build ground-up from scalability, simplicity, fault-tolerance
  - Hadoop: open-source platform on commodity machines
  - Growing collections of components & extensions

- Data Mining Algorithms with MapReduce
  - MapReduce-compatible for summation-form algorithms
  - Need task-specific algorithm design and tuning

- MapReduce has been widely used in a broad range of applications and by many organizations
  - Growing traction from both academia and industry
  - Text processing, data warehousing, machine learning, …

Large-scale Data Mining: MapReduce and Beyond
Part 2: Applications & Algorithms

Spiros Papadimitriou
Rutgers University

original material developed jointly with Jimeng Sun (IBM) and Rong Yan (Facebook)
Example: Simple Indexing Benchmark

- Node configuration: 1, 24 and 39 nodes
  - 347.5GB raw log indexing input
  - ~30KB total combiner output
  - Dual-CPU, dual-core machines
  - Variety of local drives (ATA-100 to SAS)

- Hadoop configuration
  - 64MB HDFS block size (default)
  - 64-256MB MapReduce chunk size
  - 6 (= # cores + 2) tasks per task-tracker
  - Increased buffer and thread pool sizes

Scalability: Aggregate Bandwidth

- Cluster is running a single job
### Inverted Index for Text Retrieval

[Lin & Dryer, Tutorial at NAACL/HLT 2009]

<table>
<thead>
<tr>
<th>Term</th>
<th>Doc 1</th>
<th>Doc 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>tdf</td>
<td>1.2</td>
<td>0.361</td>
</tr>
<tr>
<td>emotion</td>
<td>1.3</td>
<td>0.125</td>
</tr>
<tr>
<td>talked</td>
<td>5.4</td>
<td>0.125</td>
</tr>
<tr>
<td>information</td>
<td>0.1</td>
<td>0.00</td>
</tr>
<tr>
<td>interesting</td>
<td>1.4</td>
<td>0.002</td>
</tr>
<tr>
<td>number</td>
<td>3.7</td>
<td>0.301</td>
</tr>
<tr>
<td>retrieval</td>
<td>1.1</td>
<td>0.125</td>
</tr>
<tr>
<td>theta</td>
<td>2.1</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**Indexing Construction using MapReduce**

More details in Part 1 & 2

- Map over documents on each node to collect statistics
  - Emit term as keys, (docid, tf) as values
  - Emit other meta-data as necessary (e.g., term position)

- Reduce to aggregate doc. statistics across nodes
  - Each value represents a posting for a given key
  - Sort the posting at the end (e.g., based on docid)

- MapReduce will do all the heavy lifting
  - Typically postings cannot fit in memory of a single node

### Thread pool size

![Graph showing thread pool size](image-url)
Single-core performance

- EPIA (VIA Nehemiah 1GHz)
- Desktop (Intel Pentium 3GHz)
- Laptop (Intel Pentium M 2GHz)
- Blade (Intel Xeon 3GHz / SAS drive)

Throughput (Mbps)

Out-of-the-box configuration(s):