Data-Intensive Text Processing with MapReduce

Jimmy Lin
The iSchool
University of Maryland

JHU Summer School on Human Language Technology
Wednesday, June 17, 2009

This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States
See http://creativecommons.org/licenses/by-nc-sa/3.0/us/ for details
No data like more data!
s/knowledge/data/g;

How do we get here if we’re not Google?

(Banko and Brill, ACL 2001)
(Brants et al., EMNLP 2007)
cheap commodity clusters (or utility computing) + simple, distributed programming models = data-intensive computing for the masses!
Why is this different?
Divide and Conquer

“Work”

 Partition

 $w_1$ $w_2$ $w_3$

 “worker” “worker” “worker”

 $r_1$ $r_2$ $r_3$

 “Result”

 Combine
It’s a bit more complex...

Fundamental issues
- scheduling, data distribution, synchronization,
- inter-process communication, robustness, fault tolerance, …

Architectural issues
- Flynn’s taxonomy (SIMD, MIMD, etc.), network typology, bisection bandwidth
- UMA vs. NUMA, cache coherence

Common problems
- livelock, deadlock, data starvation, priority inversion…
- dining philosophers, sleeping barbers, cigarette smokers, …

Different programming models
- Message Passing
- Shared Memory

Different programming constructs
- mutexes, conditional variables, barriers, …
- masters/slaves, producers/consumers, work queues, …

The reality: programmer shoulders the burden of managing concurrency…
Typical Problem

- Iterate over a large number of records
- Extract something of interest from each
- Shuffle and sort intermediate results
- Aggregate intermediate results
- Generate final output

**Key idea:** provide a functional abstraction for these two operations

(Dean and Ghemawat, OSDI 2004)
MapReduce

- Programmers specify two functions:
  - \texttt{map} \( (k, v) \rightarrow <k', v'>* \)
  - \texttt{reduce} \( (k', v') \rightarrow <k', v'>* \)
    - All values with the same key are reduced together

- Usually, programmers also specify:
  - \texttt{partition} \( (k', \text{number of partitions}) \rightarrow \text{partition for } k' \)
    - Often a simple hash of the key, e.g. \( \text{hash}(k') \mod n \)
    - Allows reduce operations for different keys in parallel
  - \texttt{combine} \( (k', v') \rightarrow <k', v'>* \)
    - Mini-reducers that run in memory after the map phase
    - Used as an optimization to reducer network traffic

- Implementations:
  - Google has a proprietary implementation in C++
  - Hadoop is an open source implementation in Java
Shuffle and Sort: aggregate values by keys

Reduce

r₁ s₁
r₂ s₂
r₃ s₃
MapReduce Runtime

- Handles scheduling
  - Assigns workers to map and reduce tasks
- Handles “data distribution”
  - Moves the process to the data
- Handles synchronization
  - Gathers, sorts, and shuffles intermediate data
- Handles faults
  - Detects worker failures and restarts
- Everything happens on top of a distributed FS (later)
"Hello World": Word Count

Map(String input_key, String input_value):
   // input_key: document name
   // input_value: document contents
   for each word w in input_values:
      EmitIntermediate(w, "1");

Reduce(String key, Iterator intermediate_values):
   // key: a word, same for input and output
   // intermediate_values: a list of counts
   int result = 0;
   for each v in intermediate_values:
      result += ParseInt(v);
   Emit(AsString(result));
Redrawn from (Dean and Ghemawat, OSDI 2004)
How do we get data to the workers?

What’s the problem here?
Distributed File System

- Don’t move data to workers… Move workers to the data!
  - Store data on the local disks for nodes in the cluster
  - Start up the workers on the node that has the data local

- Why?
  - Not enough RAM to hold all the data in memory
  - Disk access is slow, disk throughput is good

- A distributed file system is the answer
  - GFS (Google File System)
  - HDFS for Hadoop (= GFS clone)
GFS: Assumptions

- Commodity hardware over “exotic” hardware
- High component failure rates
  - Inexpensive commodity components fail all the time
- “Modest” number of HUGE files
- Files are write-once, mostly appended to
  - Perhaps concurrently
- Large streaming reads over random access
- High sustained throughput over low latency

GFS slides adapted from material by (Ghemawat et al., SOSP 2003)
GFS: Design Decisions

- Files stored as chunks
  - Fixed size (64MB)
- Reliability through replication
  - Each chunk replicated across 3+ chunkservers
- Single master to coordinate access, keep metadata
  - Simple centralized management
- No data caching
  - Little benefit due to large data sets, streaming reads
- Simplify the API
  - Push some of the issues onto the client
Application

GSF Client

(file name, chunk index)

(chunk handle, chunk location)

(chunk handle, byte range)

chunk data

GFS master

File namespace

/foo/bar

chunk 2ef0

Instructions to chunkserver

Chunkserver state

GFS chunkserver

Linux file system

...
Master’s Responsibilities

- Metadata storage
- Namespace management/locking
- Periodic communication with chunkservers
- Chunk creation, re-replication, rebalancing
- Garbage Collection
Questions?
Case study: graph algorithms
Graph Algorithms: Topics

- Introduction to graph algorithms and graph representations
- Single Source Shortest Path (SSSP) problem
- PageRank
What’s a graph?

- $G = (V,E)$, where
  - $V$ represents the set of vertices (nodes)
  - $E$ represents the set of edges (links)
  - Both vertices and edges may contain additional information

- Different types of graphs:
  - Directed vs. undirected edges
  - Presence or absence of cycles
  - ...

Some Graph Problems

- Finding shortest paths
  - Routing Internet traffic and UPS trucks
- Finding minimum spanning trees
  - Telco laying down fiber
- Finding Max Flow
  - Airline scheduling
- Identify “special” nodes and communities
  - Breaking up terrorist cells, spread of avian flu
- Bipartite matching
  - Monster.com, Match.com
- And of course... PageRank
Representing Graphs

- $G = (V, E)$
- Two common representations
  - Adjacency matrix
  - Adjacency list
Adjacency Matrices

Represent a graph as an $n \times n$ square matrix $M$

- $n = |V|$
- $M_{ij} = 1$ means a link from node $i$ to $j$

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & 0 & 1 & 0 & 1 \\
2 & 1 & 0 & 1 & 1 \\
3 & 1 & 0 & 0 & 0 \\
4 & 1 & 0 & 1 & 0 \\
\end{array}
\]
Adjacency Lists

Take adjacency matrices… and throw away all the zeros

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

1: 2, 4
2: 1, 3, 4
3: 1
4: 1, 3
Single Source Shortest Path

- **Problem**: find shortest path from a source node to one or more target nodes
- Single processor machine: Dijkstra’s Algorithm
- MapReduce: parallel Breadth-First Search (BFS)
Finding the Shortest Path

- First, consider equal edge weights
- Solution to the problem can be defined inductively
- Here’s the intuition:
  - \( \text{DistanceTo}(\text{startNode}) = 0 \)
  - For all nodes \( n \) directly reachable from \( \text{startNode} \), \( \text{DistanceTo}(n) = 1 \)
  - For all nodes \( n \) reachable from some other set of nodes \( S \), \( \text{DistanceTo}(n) = 1 + \min(\text{DistanceTo}(m), m \in S) \)
From Intuition to Algorithm

- A map task receives
  - Key: node $n$
  - Value: $D$ (distance from start), points-to (list of nodes reachable from $n$)

- $\forall p \in \text{points-to}: \text{emit } (p, D+1)$

- The reduce task gathers possible distances to a given $p$ and selects the minimum one
Multiple Iterations Needed

- This MapReduce task advances the “known frontier” by one hop
  - Subsequent iterations include more reachable nodes as frontier advances
  - Multiple iterations are needed to explore entire graph
  - Feed output back into the same MapReduce task

- Preserving graph structure:
  - Problem: Where did the points-to list go?
  - Solution: Mapper emits \((n, \text{points-to})\) as well
Visualizing Parallel BFS
Weighted Edges

- Now add positive weights to the edges
- Simple change: points-to list in map task includes a weight $w$ for each pointed-to node
  - emit $(p, D+w_p)$ instead of $(p, D+1)$ for each node $p$
Comparison to Dijkstra

- Dijkstra’s algorithm is more efficient
  - At any step it only pursues edges from the minimum-cost path inside the frontier
- MapReduce explores all paths in parallel
Random Walks Over the Web

- **Model:**
  - User starts at a random Web page
  - User randomly clicks on links, surfing from page to page

- PageRank = the amount of time that will be spent on any given page
PageRank: Defined

Given page $x$ with in-bound links $t_1 \ldots t_n$, where

- $C(t)$ is the out-degree of $t$
- $\alpha$ is probability of random jump
- $N$ is the total number of nodes in the graph

$$PR(x) = \alpha \frac{1}{N} + (1 - \alpha) \sum_{i=1}^{n} \frac{PR(t_i)}{C(t_i)}$$
Computing PageRank

- Properties of PageRank
  - Can be computed iteratively
  - Effects at each iteration is local

- Sketch of algorithm:
  - Start with seed $PR_i$ values
  - Each page distributes $PR_i$ “credit” to all pages it links to
  - Each target page adds up “credit” from multiple in-bound links to compute $PR_{i+1}$
  - Iterate until values converge
PageRank in MapReduce

**Map:** distribute PageRank “credit” to link targets

**Reduce:** gather up PageRank “credit” from multiple sources to compute new PageRank value

Iterate until convergence
PageRank: Issues

- Is PageRank guaranteed to converge? How quickly?
- What is the “correct” value of $\alpha$, and how sensitive is the algorithm to it?
- What about dangling links?
- How do you know when to stop?
Graph Algorithms in MapReduce

- General approach:
  - Store graphs as adjacency lists
  - Each map task receives a node and its outlinks (adjacency list)
  - Map task compute some function of the link structure, emits value with target as the key
  - Reduce task collects keys (target nodes) and aggregates

- Iterate multiple MapReduce cycles until some termination condition
  - Remember to “pass” graph structure from one iteration to next
Questions?
MapReduce Algorithm Design

Adapted from work reported in (Lin, EMNLP 2008)
Managing Dependencies

- Remember: Mappers run in isolation
  - You have no idea in what order the mappers run
  - You have no idea on what node the mappers run
  - You have no idea when each mapper finishes

- Tools for synchronization:
  - Ability to hold state in reducer across multiple key-value pairs
  - Sorting function for keys
  - Partitioner
  - Cleverly-constructed data structures
Motivating Example

- Term co-occurrence matrix for a text collection
  - \( M = N \times N \) matrix (\( N \) = vocabulary size)
  - \( M_{ij} \): number of times \( i \) and \( j \) co-occur in some context (for concreteness, let’s say context = sentence)

- Why?
  - Distributional profiles as a way of measuring semantic distance
  - Semantic distance useful for many language processing tasks
**MapReduce: Large Counting Problems**

- Term co-occurrence matrix for a text collection
  = specific instance of a large counting problem
  - A large event space (number of terms)
  - A large number of observations (the collection itself)
  - Goal: keep track of interesting statistics about the events

- Basic approach
  - Mappers generate partial counts
  - Reducers aggregate partial counts

**How do we aggregate partial counts efficiently?**
First Try: “Pairs”

- Each mapper takes a sentence:
  - Generate all co-occurring term pairs
  - For all pairs, emit \((a, b) \rightarrow \text{count}\)
- Reducers sums up counts associated with these pairs
- Use combiners!
“Pairs” Analysis

- Advantages
  - Easy to implement, easy to understand

- Disadvantages
  - Lots of pairs to sort and shuffle around (upper bound?)
Another Try: “Stripes”

- Idea: group together pairs into an associative array
  
  \[
  (a, b) \rightarrow 1 \\
  (a, c) \rightarrow 2 \\
  (a, d) \rightarrow 5 \\
  (a, e) \rightarrow 3 \\
  (a, f) \rightarrow 2 \\
  \]

- Each mapper takes a sentence:
  - Generate all co-occurring term pairs
  - For each term, emit \( a \rightarrow \{ \text{b: count}_b, \text{c: count}_c, \text{d: count}_d \ldots \} \)

- Reducers perform element-wise sum of associative arrays

\[
\begin{align*}
  a & \rightarrow \{ \text{b: 1, d: 5, e: 3} \} \\
  + & \quad a \rightarrow \{ \text{b: 1, c: 2, d: 2, f: 2} \} \\
  a & \rightarrow \{ \text{b: 2, c: 2, d: 7, e: 3, f: 2} \}
\end{align*}
\]
“Stripes” Analysis

- Advantages
  - Far less sorting and shuffling of key-value pairs
  - Can make better use of combiners

- Disadvantages
  - More difficult to implement
  - Underlying object is more heavyweight
  - Fundamental limitation in terms of size of event space
Cluster size: 38 cores
Data Source: Associated Press Worldstream (APW) of the English Gigaword Corpus (v3), which contains 2.27 million documents (1.8 GB compressed, 5.7 GB uncompressed)
Conditional Probabilities

- How do we estimate conditional probabilities from counts?

\[
P(B | A) = \frac{\text{count}(A, B)}{\text{count}(A)} = \frac{\text{count}(A, B)}{\sum_{B'} \text{count}(A, B')}
\]

- Why do we want to do this?

- How do we do this with MapReduce?
P(B|A): “Stripes”

\[ a \rightarrow \{ b_1 : 3, b_2 : 12, b_3 : 7, b_4 : 1, \ldots \} \]

- Easy!
  - One pass to compute \((a, *)\)
  - Another pass to directly compute P(B|A)
P(B|A): “Pairs”

Reducer holds this value in memory

(a, *) → 32

(a, b₁) → 3
(a, b₂) → 12
(a, b₃) → 7
(a, b₄) → 1

For this to work:

- Must emit extra (a, *) for every bₙ in mapper
- Must make sure all a’s get sent to same reducer (use partitioner)
- Must make sure (a, *) comes first (define sort order)
- Must hold state in reducer across different key-value pairs
Synchronization in Hadoop

- Approach 1: turn synchronization into an ordering problem
  - Sort keys into correct order of computation
  - Partition key space so that each reducer gets the appropriate set of partial results
  - Hold state in reducer across multiple key-value pairs to perform computation
  - Illustrated by the “pairs” approach

- Approach 2: construct data structures that “bring the pieces together”
  - Each reducer receives all the data it needs to complete the computation
  - Illustrated by the “stripes” approach
Issues and Tradeoffs

- Number of key-value pairs
  - Object creation overhead
  - Time for sorting and shuffling pairs across the network
- Size of each key-value pair
  - De/serialization overhead
- Combiners make a big difference!
  - RAM vs. disk and network
  - Arrange data to maximize opportunities to aggregate partial results
Questions?
Case study: statistical machine translation
Statistical Machine Translation

- Conceptually simple:
  (translation from foreign \( f \) into English \( e \))

\[
\hat{e} = \arg \max_e P(f \mid e)P(e)
\]

- Difficult in practice!

- Phrase-Based Machine Translation (PBMT):
  - Break up source sentence into little pieces (phrases)
  - Translate each phrase individually

Dyer et al. (Third ACL Workshop on MT, 2008)
Translation as a “Tiling” Problem

Example from Koehn (2006)

Maria no dio una bofetada a la bruja verde

Maria did not give a slap to the green witch
MT Architecture

Training Data

i saw the small table
vi la mesa pequeña

Parallel Sentences

he sat at the table
the service was good

Target-Language Text

Word Alignment

Phrase Extraction

Language Model

Translation Model

Decoder

maria no daba una bofetada a la bruja verde
mary did not slap the green witch

Foreign Input Sentence

English Output Sentence
The Data Bottleneck

![Graph showing the relationship between time (in days, hours, minutes) and translation quality (BLEU score) as corpus size increases.]
There are MapReduce Implementations of these two components!

Training Data
- I saw the small table
  - vi la mesa pequeña
- He sat at the table
  - the service was good

Parallel Sentences

Target-Language Text

Word Alignment

Phrase Extraction

Language Model

Translation Model

Decoder

Foreign Input Sentence
- maria no daba una bofetada a la bruja verde

English Output Sentence
- mary did not slap the green witch
HMM Alignment: Giza

HMM alignment (Giza toolkit) ––△––

Single-core commodity server

Time (seconds)

3 hrs
60 min
20 min
3m20s
90 s
30 s
10 s
3 s

Corpus size (sentences)

10000
100000
1e+06
HMM Alignment: MapReduce

- HMM alignment (Giza toolkit)
- HMM alignment (MapReduce implementation)

Time (seconds)

- 3 hrs
- 60 min
- 20 min
- 3m20s
- 90 s
- 30 s
- 10 s
- 3 s

Corpus size (sentences)

- 10000
- 100000
- 1e+06

Single-core commodity server

38 processor cluster
HMM Alignment: MapReduce
What’s the point?

- The optimally-parallelized version doesn’t exist!
- It’s all about the right level of abstraction
  - Goldilocks argument
Questions?
What’s next?

- Web-scale text processing: luxury → necessity
  - Fortunately, the technology is becoming more accessible
- MapReduce is a nice hammer:
  - Whack it on everything in sight!
- MapReduce is only the beginning…
  - Alternative programming models
  - Fundamental breakthroughs in algorithm design
Applications
(NLP, IR, ML, etc.)

Programming Models
(MapReduce…)

Systems
(architecture, network, etc.)
Questions?
Questions?
Comments?
Comments?

Thanks to the organizations who support our work: