CS 6604: Data Mining Large Networks and Time-series

B. Aditya Prakash

Lecture #6: Hadoop and Graph Analysis
(some slides from Xiao Yu)

NO SQL
Why No SQL?

**HOW TO WRITE A CV**

**DO YOU HAVE ANY EXPERTISE IN SQL?**

**NO**

**DIDNT MATTER. WRITE: “EXPERT IN NO SQL”**

Leverage the NoSQL boom
RDBMS

- The predominant choice in storing data
  - Not so true for data miners since we put much in txt files.
- First formulated in 1969 by Codd
  - We are using RDBMS everywhere
Aside: RDBMS performance

Salary List

Majority of Webapps

Social network

Semantic Trading

custom

Data complexity

I DON'T ALWAYS USE RDBMS

BUT WHEN I DO, I DUMP EVERYTHING IN IT

When RDBMS met Web 2.0

Big data

Connectivity

P2P Knowledge

Concurrency

Diversity

Cloud-Grid

Slide from Lorenzo Alberton, "NoSQL Databases: Why, what and when"
What to do if data is really large?

- Peta-bytes (exabytes, zettabytes .....
- Google processed 24 PB of data per day (2009)
- FB adds 0.5 PB per day
BIG data

How Much Data Is Generated Every Minute?

THE MOBILE WEB RECEIVES 217 NEW USERS.

YOUTUBE USERS UPLOAD 48 HOURS OF NEW VIDEO.

EMAIL USERS SEND 204,166,667 MESSAGES.

GOOGLE RECEIVES OVER 2,000,000 SEARCH QUERIES.

WORDPRESS USERS PUBLISH 347 NEW BLOG POSTS.

FACEBOOK USERS SHARE 684,478 PIECES OF CONTENT.

FOURSQUARE USERS PERFORM 2,083 CHECK-INS.

TWITTER USERS SEND OVER 100,000 TWEETS.

Flickr USERS ADD 3,125 NEW PHOTOS.

INSTAGRAM USERS SHARE 3,600 NEW PHOTOS.

APPLE RECEIVES ABOUT 47,000 APP DOWNLOADS.

TUMBLR BLOG OWNERS PUBLISH 27,778 NEW POSTS.

BRANDS & ORGANIZATIONS ON FACEBOOK RECEIVE 34,722 "LIKES."

571 NEW WEBSITES ARE CREATED.

EVERY MINUTE OF THE DAY:

CONSUMERS SPEND $272,070 ON WEB SHOPPING.

EVERY DAY:

WITH NO SIGNS OF SLOWING, THE DATA KEEPS GROWING

Prakash 2015
What’s Wrong with Relational DB?

- Nothing is wrong. You just need to use the right tool.
- Relational is hard to scale.
  - Easy to scale reads
  - Hard to scale writes
What’s NoSQL?

- The misleading term “NoSQL” is short for “Not Only SQL”.
- non-relational, schema-free, non-(quite)-ACID
  - More on ACID transactions later in class
- horizontally scalable, distributed, easy replication support
- simple API
Four (emerging) NoSQL Categories

- **Key-value (K-V) stores**
  - Based on Distributed Hash Tables/ Amazon’s Dynamo paper *
  - Data model: (global) collection of K-V pairs
  - Example: Voldemort

- **Column Families**
  - BigTable clones **
  - Data model: big table, column families
  - Example: HBase, Cassandra, Hypertable

---

*G DeCandia et al, Dynamo: Amazon's Highly Available Key-value Store, SOSP 07
** F Chang et al, Bigtable: A Distributed Storage System for Structured Data, OSDI 06
Four (emerging) NoSQL Categories

- **Document databases**
  - Inspired by Lotus Notes
  - Data model: collections of K-V Collections
  - Example: CouchDB, MongoDB

- **Graph databases**
  - Inspired by Euler & graph theory
  - Data model: nodes, relations, K-V on both
  - Example: AllegroGraph, VertexDB, Neo4j
Focus of Different Data Models

Slide from neo technology, “A NoSQL Overview and the Benefits of Graph Databases"
C-A-P “theorem"

- Consistency
- Availability
- Partition Tolerance

RDBMS
NoSQL (most)
When to use NoSQL?

- Bigness
- Massive write performance
  - Twitter generates 7TB / per day (2010)
- Fast key-value access
- Flexible schema or data types
- Schema migration
- Write availability
  - Writes need to succeed no matter what (CAP, partitioning)
- Easier maintainability, administration and operations
- No single point of failure
- Generally available parallel computing
- Programmer ease of use
- Use the right data model for the right problem
- Avoid hitting the wall
- Distributed systems support
- Tunable CAP tradeoffs

from http://highscalability.com/
Key-Value Stores

<table>
<thead>
<tr>
<th>id</th>
<th>hair_color</th>
<th>age</th>
<th>height</th>
</tr>
</thead>
<tbody>
<tr>
<td>1923</td>
<td>Red</td>
<td>18</td>
<td>6’0”</td>
</tr>
<tr>
<td>3371</td>
<td>Blue</td>
<td>34</td>
<td>NA</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

Table in relational db

<table>
<thead>
<tr>
<th>user1923_color</th>
<th>Red</th>
</tr>
</thead>
<tbody>
<tr>
<td>user1923_age</td>
<td>18</td>
</tr>
<tr>
<td>user3371_color</td>
<td>Blue</td>
</tr>
<tr>
<td>user4344_color</td>
<td>Brackish</td>
</tr>
<tr>
<td>user1923_height</td>
<td>6’ 0”</td>
</tr>
<tr>
<td>user3371_age</td>
<td>34</td>
</tr>
</tbody>
</table>

Store/Domain in Key-Value db

Find users whose age is above 18?
Find all attributes of user 1923?
Find users whose hair color is Red and age is 19?
(Join operation) Calculate average age of all grad students?
Voldemort in LinkedIn

People You May Know

Roshan Sumbaly, Senior Software Engineer at LinkedIn
Alex Feinberg, Senior Software Engineer at LinkedIn
Jay Kreps, Principal Staff Engineer at LinkedIn

Viewers of this profile also viewed

Sam Shah, Principal Engineer at LinkedIn
Igor Perelis, Director of Engineering at LinkedIn
Anmol Bhasin, Recommendations, A/B Testing and more
Jun Rao, Principle Software Engineer at LinkedIn

Related Searches

mapreduce, java, big data, hbase, machine learning, lucene, data mining, data warehouse

Events you may be interested in

Improving Hadoop Performance by up to 1000x - A LinkedIn Te... December 15, 2011 - LinkedIn headquarters - TALK:OPEN TO PUBLIC. Event highlights:

LinkedIn Skills

Hadoop

Jobs you may be interested in

Senior Software Engineer - Applications, Modo Networks - San Francisco Bay Area
Senior Software Engineer, CD++, StumbleUpon - San Francisco, CA
Senior Software Engineer, Data Infrastructure, PolySync - San Francisco Bay Area

Sid Anand, LinkedIn Data Infrastructure (QCon London 2012)
Voldemort vs MySQL

Sid Anand, LinkedIn Data Infrastructure (QCon London 2012)
Column Families – BigTable like

Sparse, distributed, persistent multi-dimensional sorted map indexed by \((row\_key, \ column\_key, \ timestamp)\)
BigTable Data Model

The row name is a reversed URL. The contents column family contains the page contents, and the anchor column family contains the text of any anchors that reference the page.
BigTable Performance

- 4M scans
- 3M random reads (mem)
- 3M random writes
- 2M sequential reads
- 2M sequential writes
- 1M random reads

Values read/written per second vs Number of tablet servers
Document Database - mongoDB

<table>
<thead>
<tr>
<th>Last Name</th>
<th>First Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>DUMONT</td>
<td>Jean</td>
<td>43</td>
</tr>
<tr>
<td>PELLERIN</td>
<td>Franck</td>
<td>29</td>
</tr>
<tr>
<td>MATTHIEU</td>
<td>Nicolas</td>
<td>51</td>
</tr>
</tbody>
</table>

Table in relational db

```
{
   "_id": ObjectId("4efa8d2b7d284dad101e4bc9"),
   "Last Name": "DUMONT",
   "First Name": "Jean",
   "Age": 43
},
{
   "_id": ObjectId("4efa8d2b7d284dad101e4bc7"),
   "Last Name": "PELLERIN",
   "First Name": "Franck",
   "Age": 29,
   "Address": "1 chemin des Loges",
   "City": "VERSAILLES"
}
```

Documents in a collection

Open source, document db
Json-like document with dynamic schema

Initial release 2009
mongoDB Product Deployment
Graph Database

Data Model Abstraction:
• Nodes
• Relations
• Properties
Neo4j - Build a Graph

```java
NeoService neo = ... // Get factory

// Create Thomas 'Neo' Anderson
Node mrAnderson = neo.createNode();
mrAnderson.setProperty( "name", "Thomas Anderson" );
mrAnderson.setProperty( "age", 29 );

// Create Morpheus
Node morpheus = neo.createNode();
morpheus.setProperty( "name", "Morpheus" );
morpheus.setProperty( "rank", "Captain" );
morpheus.setProperty( "occupation", "Total bad ass" );

// Create a relationship representing that they know each other
mrAnderson.createRelationshipTo( morpheus, RelTypes.KNOWS );
// ...create Trinity, Cypher, Agent Smith, Architect similarly
```

Slide from neo technology, “A NoSQL Overview and the Benefits of Graph Databases"
A Debatable Performance Evaluation
Conclusion

- Use the right data model for the right problem
MAP-REDUCE AND HADOOP
MapReduce

- Much of the course will be devoted to large scale computing for data mining.

- **Challenges:**
  - How to distribute computation?
  - Distributed/parallel programming is hard.

- **Map-reduce** addresses all of the above.
  - Google’s computational/data manipulation model.
  - Elegant way to work with big data.
Single Node Architecture

Machine Learning, Statistics

“Classical” Data Mining
Motivation: Google Example

- 20+ billion web pages x 20KB = 400+ TB
- 1 computer reads 30-35 MB/sec from disk
  - ~4 months to read the web
- ~1,000 hard drives to store the web
- Takes even more to do something useful with the data!
- Today, a standard architecture for such problems is emerging:
  - Cluster of commodity Linux nodes
  - Commodity network (ethernet) to connect them
Cluster Architecture

2-10 Gbps backbone between racks

1 Gbps between any pair of nodes in a rack

Each rack contains 16-64 nodes

In 2011 it was guestimated that Google had 1M machines, [http://bit.ly/Shh0RO](http://bit.ly/Shh0RO)
Large-scale Computing

- Large-scale computing for data mining problems on commodity hardware

- Challenges:
  - How do you distribute computation?
  - How can we make it easy to write distributed programs?
  - Machines fail:
    - One server may stay up 3 years (1,000 days)
    - If you have 1,000 servers, expect to lose 1/day
    - People estimated Google had ~1M machines in 2011
      - 1,000 machines fail every day!
Idea and Solution

- **Issue:** Copying data over a network takes time

- **Idea:**
  - Bring computation close to the data
  - Store files multiple times for reliability

- **Map-reduce addresses these problems**
  - Google’s computational/data manipulation model
  - Elegant way to work with big data
  - **Storage Infrastructure – File system**
    - Google: GFS. Hadoop: HDFS
  - **Programming model**
    - Map-Reduce
Hadoop VS NoSQL

- **Hadoop:** computing framework
  - Supports data-intensive applications
  - Includes MapReduce, HDFS etc.
    (we will study MR mainly next)

- **NoSQL:** Not only SQL databases
  - Can be built ON hadoop. E.g. HBase.
Storage Infrastructure

- **Problem:**
  - If nodes fail, how to store data persistently?

- **Answer:**
  - **Distributed File System:**
    - Provides global file namespace
    - Google GFS; Hadoop HDFS;

- **Typical usage pattern**
  - Huge files (100s of GB to TB)
  - Data is rarely updated in place
  - Reads and appends are common
Distributed File System

- **Chunk servers**
  - File is split into contiguous chunks
  - Typically each chunk is 16-64MB
  - Each chunk replicated (usually 2x or 3x)
  - Try to keep replicas in different racks

- **Master node**
  - a.k.a. Name Node in Hadoop’s HDFS
  - Stores metadata about where files are stored
  - Might be replicated

- **Client library for file access**
  - Talks to master to find chunk servers
  - Connects directly to chunk servers to access data
Distributed File System

- Reliable distributed file system
- Data kept in “chunks” spread across machines
- Each chunk replicated on different machines
  – Seamless recovery from disk or machine failure

Bring computation directly to the data!

Chunk servers also serve as compute servers
Programming Model: MapReduce

Warm-up task:

- We have a huge text document
- Count the number of times each distinct word appears in the file

Sample application:

- Analyze web server logs to find popular URLs
Task: Word Count

Case 1:

– File too large for memory, but all <word, count> pairs fit in memory

Case 2:

- Count occurrences of words:
  - `words(doc.txt) | sort | uniq -c`
    - where `words` takes a file and outputs the words in it, one per a line

- Case 2 captures the essence of MapReduce
  - Great thing is that it is naturally parallelizable
MapReduce: Overview

- Sequentially read a lot of data
- **Map:**
  - Extract something you care about
- **Group by key:** Sort and Shuffle
- **Reduce:**
  - Aggregate, summarize, filter or transform
- Write the result

Outline stays the same, **Map** and **Reduce** change to fit the problem
MapReduce: The **Map** Step

**Input key-value pairs**

```
\[ k \quad v \]
```

```
\[ k \quad v \]
```

```
\[ k \quad v \]
```

```
\[ k \quad v \]
```

```
\[ k \quad v \]
```

**Intermediate key-value pairs**

```
\[ k \quad v \]
```

```
\[ k \quad v \]
```

```
\[ k \quad v \]
```

```
\[ k \quad v \]
```

```
\[ k \quad v \]
```

```
\[ k \quad v \]
```

Prakash 2015
MapReduce: The Reduce Step

Intermediate key-value pairs

Key-value groups

Output key-value pairs

Group by key

reduce

reduce

reduce

Prakash 2015
More Specifically

- **Input**: a set of key-value pairs
- Programmer specifies two methods:
  
  - **Map** \( (k, v) \rightarrow <k’, v’>* \)
    
    - Takes a key-value pair and outputs a set of key-value pairs
    - E.g., key is the filename, value is a single line in the file
    - There is one Map call for every \((k, v)\) pair
  
  - **Reduce** \( (k’, <v’>*) \rightarrow <k’, v”>* \)
    
    - All values \(v’\) with same key \(k’\) are reduced together and processed in \(v’\) order
    - There is one Reduce function call per unique key \(k’\)
The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/machine partnership. "The work we're doing now -- the robotics we're doing -- is what we're going to need .........................

MapReduce: Word Counting

**Big document**

Prakash 2015

**Provided by the programmer**

**MAP:**
Read input and produces a set of key-value pairs

- (The, 1)
- (crew, 1)
- (of, 1)
- (the, 1)
- (space, 1)
- (shuttle, 1)
- (Endeavor, 1)
- (recently, 1)
- ....

**Group by key:**
Collect all pairs with same key

- (crew, 1)
- (crew, 1)
- (space, 1)
- (the, 1)
- (the, 1)
- (the, 1)
- (shuttle, 1)
- (recently, 1)
- ....

**Provided by the programmer**

**Reduce:**
Collect all values belonging to the key and output

- (crew, 2)
- (space, 1)
- (the, 3)
- (shuttle, 1)
- (recently, 1)
- ...

CS 6604: DM Large Networks & Time-Series

Prakash 2015
map(key, value):
  // key: document name; value: text of the document
  for each word w in value:
    emit(w, 1)

reduce(key, values):
  // key: a word; value: an iterator over counts
  result = 0
  for each count v in values:
    result += v
  emit(key, result)
Map-Reduce (MR) as SQL

- select `count(*)` from DOCUMENT
- group by word

Reducer

Mapper
Map-Reduce: Environment

Map-Reduce environment takes care of:

- Partitioning the input data
- Scheduling the program’s execution across a set of machines
- Performing the group by key step
- Handling machine failures
- Managing required inter-machine communication
Map-Reduce: A diagram

**MAP:**
Read input and produces a set of key-value pairs

**Group by key:**
Collect all pairs with same key (Hash merge, Shuffle, Sort, Partition)

**Reduce:**
Collect all values belonging to the key and output

---

**Input:** Big document

**Intermediate:**
- \(k_1:v\) \(k_1:v\) \(k_2:v\)
- \(k_1:v\)
- \(k_3:v\) \(k_4:v\)
- \(k_4:v\) \(k_5:v\)
- \(k_4:v\)
- \(k_1:v\) \(k_3:v\)

**Group by Key**

**Grouped:**
- \(k_1:v,v,v,v\)
- \(k_2:v\)
- \(k_3:v,v\)
- \(k_4:v,v,v\)
- \(k_5:v\)

**Output**

---

Prakash 2015
Map-Reduce: In Parallel

All phases are distributed with many tasks doing the work
Map-Reduce

- **Programmer specifies:**
  - Map and Reduce and input files
- **Workflow:**
  - Read inputs as a set of key-value-pairs
  - **Map** transforms input kv-pairs into a new set of k'v'-pairs
  - Sorts & Shuffles the k'v'-pairs to output nodes
  - All k'v'-pairs with a given k’ are sent to the same **reduce**
  - **Reduce** processes all k'v'-pairs grouped by key into new k''v''-pairs
  - Write the resulting pairs to files

- All phases are distributed with many tasks doing the work

Prakash 2015
Data Flow

- **Input and final output** are stored on a distributed file system (FS):
  - Scheduler tries to schedule map tasks “close” to physical storage location of input data

- **Intermediate results** are stored on local FS of Map and Reduce workers

- **Output is often input to another MapReduce task**
Coordination: Master

- **Master node takes care of coordination:**
  - **Task status:** (idle, in-progress, completed)
  - **Idle tasks** get scheduled as workers become available
  - When a map task completes, it sends the master the location and sizes of its $R$ intermediate files, one for each reducer
  - Master pushes this info to reducers

- Master pings workers periodically to detect failures
Dealing with Failures

- **Map worker failure**
  - Map tasks completed or in-progress at worker are reset to idle
  - Reduce workers are notified when task is rescheduled on another worker

- **Reduce worker failure**
  - Only in-progress tasks are reset to idle
  - Reduce task is restarted

- **Master failure**
  - MapReduce task is aborted and client is notified
GRAPH ANALYSIS ON HADOOP

Based on slides from Prof. U. Kang
Problem Definition

- **Given:** a BIG graph $G(V, E)$
- **Compute:**
  - Connected Components
  - Diameter
  - PageRank
  - ....

- **Solution:** Use Hadoop

Prakash 2015
GIM-V: One unified framework [Kang+, 2008]

YahooWeb:
|V| = 1.4B
|E| = 6.6B

Example: GIM-V At Work

Connected Components

300-size cmpt X 500. Why?
1100-size cmpt X 65. Why?

Giant Connected Component
Main Idea

- **GIM-V**
  - Generalized Iterative Matrix-Vector Multiplication
  - Extension of plain matrix-vector multiplication
  - Includes as special cases
    - Connected Components
    - PageRank
    - RWR
    - Diameters
    - ....
Main Idea: Intuition

- Weighted Combination of Colors
- ~ Message Passing

\[ M \times v = v' \]

\[ v_4' = \sum_{i=1}^{4} m_{i4} v_i \]
Three implicit operations here:

- Multiply $m_{ij}$ and $v_j$
- Sum $n$ multiplication results
- Update $v_i'$

Message sending
Message combination

**Main Idea**

"Plain M-V multiplication"
Main Idea

- **GIM-V**
  - Matrix represents edge(Src, dest)
  - Vector represents ‘node values/labels’
  - 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>combineAll</td>
<td>Multiply</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>combine2</td>
<td></td>
<td>Sum</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>
GIM-V

<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>combine2</td>
<td>Multiply</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>combineAll</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>

- **Bool. X**: MIN
- **MIN**:
GIM-V for CC

- How many connected components?
  -- == which node belongs to which component?

Input Graph

Output

node id  component id
1       1
2       1
3       1
4       1
5       5
6       5
7       7
8       7

G1 G5 G7
GIM-V for CC

Prakash 2015
CS 6604: DM Large Networks & Time-Series
GIM-V for CC

\[ \text{combine}_2(m_{i,j}, v_j) = m_{i,j} \times v_j \]
\[ \text{combine}_\text{All}(x_1, \ldots, x_n) = \min\{x_i \mid i = 1..n\} \]
\[ \text{assign}(v_i, v_{\text{new}}) = \min(v_i, v_{\text{new}}) \]

“Sending Invitations”
“Accept the Smallest”

```
1  2  3  4  5  6  7  8
1  1  1  1  1  1  1  1
2  1  1  1  1  1  1  1
3  1  1  1  1  1  1  1
4  1  1  1  1  1  1  1
5  1  1  1  1  1  1  1
6  1  1  1  1  1  1  1
7  1  1  1  1  1  1  1
8  1  1  1  1  1  1  1

\times G
```

---

Prakash 2015
CS 6604: DM Large Networks & Time-Series
GIM-V for CC

\[
\text{combine}_2(m_{i,j}, v_j) = m_{i,j} \times v_j \\
\text{combine}_\text{All}(x_1, ..., x_n) = \min\{x_i \mid i = 1..n\} \\
\text{assign}(v_i, v_{\text{new}}) = \min(v_i, v_{\text{new}})
\]

“Sending Invitations”

“Accept the Smallest”

1 GIM-V with MIN = find min. node ids within 1 hop
GIM-V for CC

combine\(2(m_{i,j}, v_j) = m_{i,j} \times v_j\)

\(combineAll(x_1, ..., x_n) = \min\{x_i \mid i = 1..n\}\)

\(assign(v_i, v_{\text{new}}) = \min(v_i, v_{\text{new}})\)

"Sending Invitations"

"Accept the Smallest"

\[\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
3 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
4 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
5 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
6 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
7 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
8 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}\]

\[\times G\]

\[\begin{array}{cccc}
1 & 2 & 3 & 4 \\
\min(1, \min(2)) & \min(2, \min(1,3)) & \min(3, \min(2,4)) & \min(4, \min(3)) \\
\min(5, \min(6)) & \min(6, \min(5)) & \min(7, \min(8)) & \min(8, \min(7)) \\
\end{array}\]

\(k\) GIM-V with MIN = find min. node ids within \(k\) hops
**GIM-V for Connected Components**

\[
\text{combine2}(m_{i,j}, v_j) = m_{i,j} \times v_j
\]

\[
\text{combineAll}(x_1, \ldots, x_n) = \min\{x_i \mid i = 1 \ldots n\}
\]

\[
\text{assign}(v_i, v_{new}) = \min(v_i, v_{new})
\]

---

**Main Idea**

```
min(1, min(2) )
min(2, min(1,3) )
min(3, min(2,4) )
min(4, min(3) )
min(5, min(6) )
min(6, min(5) )
min(7, min(8) )
min(8, min(7) )
```

---

“Sending Invitations”

“Accept the Smallest”

---

Max. iterations = diameter
### GIM-V

<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>combine2</td>
<td>Multiply</td>
<td>Multiply</td>
<td>Multiply with c</td>
<td>Multiply with c</td>
<td>Multiply bit-vector</td>
</tr>
<tr>
<td>combineAll</td>
<td>Sum</td>
<td>MIN</td>
<td>Sum with rj prob.</td>
<td>Sum with res 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>

Prakash 2015

CS 6604: DM Large Networks & Time-Series
HDFS restrictions

- [R1] HDFS is location transparent
  - Users don’t know which machine has which file

- [R2] A line is never split
  - A large file is split into pieces of a size (e.g. 256MB)
  - Users don’t know the point of split
Fast GIM-V

- Given R1 and R2, how to design faster algs. for GIM-V?
  - 1. Block Multiplication
  - 2. Clustering
  - 3. Compression
Block Multiplication

1. Group elements together into 1 line
2. Storage for an element: $2\log n$ bits $\rightarrow$ $2\log b$ bits
3. Adjust the MapReduce code (block multiplication)

Log b bits

Log n bits

b: block width

n: # of nodes
Clustering and Compress

Main Idea

I2) Clustering

Preprocess

Faloutsos and Kang (CMU)

SIGMOD'12

I3) Compression

Compress

ZIP

ZIP

Faloutsos and Kang (CMU)

SIGMOD'12
Performance

<table>
<thead>
<tr>
<th></th>
<th>Block Encoding?</th>
<th>Compression?</th>
<th>Clustering?</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>NNB</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>NCB</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>CCB</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

5x smaller, 43x smaller, 9.2x faster, 2x faster, 1.7x faster.
Many extensions and optimizations

- Local queries
- Diagonal block iterations
- ...
- ...

Prakash 2015
Conclusions

- Hadoop is a distributed data-intensive computing framework
- MapReduce
  - Simple programming paradigm
  - Surprisingly powerful (may not be suitable for all tasks though)
- Hadoop has specialized FileSystem, Master-Slave Architecture to scale-up
NoSQL and Hadoop

- Hot area with several new problems
  - Good for academic research
  - Good for industry

= Fun AND Profit 😊
POINTERS AND FURTHER READING
Implementations

- **Google**
  - Not available outside Google

- **Hadoop**
  - An open-source implementation in Java
  - Uses HDFS for stable storage

- **Aster Data**
  - Cluster-optimized SQL Database that also implements MapReduce
Cloud Computing

- Ability to rent computing by the hour
  - Additional services e.g., persistent storage

- Amazon’s “Elastic Compute Cloud” (EC2)

- Aster Data and Hadoop can both be run on EC2
Reading

- Jeffrey Dean and Sanjay Ghemawat: MapReduce: Simplified Data Processing on Large Clusters

- Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung: The Google File System
Resources

- Hadoop Wiki
  - Introduction
  - Getting Started
  - Map/Reduce Overview
    - [http://wiki.apache.org/lucene-hadoop/HadoopMapReduce](http://wiki.apache.org/lucene-hadoop/HadoopMapReduce)
    - [http://wiki.apache.org/lucene-hadoop/HadoopMapRedClasses](http://wiki.apache.org/lucene-hadoop/HadoopMapRedClasses)
  - Eclipse Environment

- Javadoc
  - [http://lucene.apache.org/hadoop/docs/api/](http://lucene.apache.org/hadoop/docs/api/)
Resources

- Releases from Apache download mirrors

- Nightly builds of source

- Source code from subversion
  - [http://lucene.apache.org/hadoop/version_control.html](http://lucene.apache.org/hadoop/version_control.html)
Further Reading

- Programming model inspired by functional language primitives
- Partitioning/shuffling similar to many large-scale sorting systems
  - NOW-Sort ['97]
- Re-execution for fault tolerance
  - BAD-FS ['04] and TACC ['97]
- Locality optimization has parallels with Active Disks/Diamond work
  - Active Disks ['01], Diamond ['04]
- Backup tasks similar to Eager Scheduling in Charlotte system
  - Charlotte ['96]
- Dynamic load balancing solves similar problem as River's distributed queues
  - River ['99]