Distributed Data Management
Summer Semester 2013
TU Kaiserslautern

Dr.-Ing. Sebastian Michel

smichel@mmci.uni-saarland.de
Lecture 3

MAP REDUCE (CONT’D): HADOOP AND APPLICATIONS
Recap: Map Reduce from High Level

Distributed Data Management, SoSe 2013, S. Michel
Recap: Map and Reduce

- Map \((k1,v1) \rightarrow \text{list}(k2,v2)\)
- Reduce\((k2, \text{list}(v2)) \rightarrow \text{list}(k3, v3)\)

- For instance:
  - \(k1=\) document identifier
  - \(v1=\) document content
  - \(k2=\) term
  - \(v2=\) count
  - \(k3=\) term
  - \(v3=\) final count
Example: Co-occurrences

• Given: text file
• Want: for terms a, b, how often does a and b occur close together, e.g., within sentence?
• That is, output = ([a,b], count)

• How can this be computed?
Example: Co-occurrences (Cont’d)

• **Solution 1: pairs approach**
  – mapper for string s:
    • for all term pairs (a,b) in s: emit({a,b}, 1)
  – reducer just aggregates counts

• **Solution 2: “stripes” approach**
  – mapper for string s:
    • collect all \( t_i \) that co-occur with a
    • emit (a,{t_1, t_2, .... t_n})
  – reducer aggregates
### Map Reduce vs. Databases

<table>
<thead>
<tr>
<th></th>
<th>Traditional RDBMS</th>
<th>Map Reduce</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Size</strong></td>
<td>Gigabytes</td>
<td>Petabytes</td>
</tr>
<tr>
<td><strong>Access</strong></td>
<td>Interactive and batch</td>
<td>Batch</td>
</tr>
<tr>
<td><strong>Updates</strong></td>
<td>Read and write many times</td>
<td>Write once, read many times</td>
</tr>
<tr>
<td><strong>Structure</strong></td>
<td>Static schema</td>
<td>Dynamic schema</td>
</tr>
<tr>
<td><strong>Integrity</strong></td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td><strong>Scaling</strong></td>
<td>Non linear</td>
<td>Linear</td>
</tr>
</tbody>
</table>

*source: T. White, Hadoop, The Definitive Guide, 3rd edition*
Objectives/Benefits

• Simple model (see also criticisms) ;)
• Scalable (depends also on problem of course)

• Aims at high throughput

• Tolerant against node failures
Limitations

• Very low level routines
• Can have quite slow response time for individual, small tasks
• Writing complex queries can be a hassle
  – Think: declarative languages like SQL

SELECT * FROM
WHERE
GROUP BY
...

Distributed Data Management, SoSe 2013, S. Michel
Criticism

• Some people claim MR is a **major step backward**

• Why?
  – Too low level
  – No indices
  – No updates
  – No transactions

• But: was it really made to replace a DB?

SQL expressed in MR?

• SELECT name, salary
  FROM employees
  WHERE age < 40

How?
  – map filters based on age
  – reduce doesn’t have to do anything
    
    (no reducer=> map output just written do
    distributed file system, no sorting,
    no partitioning).

  In Hadoop: Diff between “no” and identity reducer?
SQL expressed in MR? (Cont’d)

• SELECT name, avg(contacts)  
  FROM facebookTable  
  GROUP by name

How?
  map emits data with key=name  
  reduce computes average number of contacts
Hands on MapReduce (with Hadoop)

- Apache Hadoop. Open Source MR

- Wide acceptance:
  - http://wiki.apache.org/hadoop/PoweredBy
  - Amazon.com, Apple, AOL, eBay, IBM, Google, LinkedIn, Last.fm, Microsoft, SAP, Twitter, ...
Hadoop Distributed File System (HDFS): Basics

- Given file is cut in big pieces (blocks) (e.g., 64MB)
- Which are then assigned to (different) nodes
HDFS Architecture

Metadata (Name, replicas, ...) /home/foo/data, 3, ...

Client

metadata ops

NameNode

block ops

DataNodes

replication of block

replication

write

read

Client

DataNodes

Rack 1

Rack 2

source: http://hadoop.apache.org
Replication

• Can specify default replication factor (or per directory/file)

• Replication is pipelined
  – if block is full, NameNode is asked for other DataNodes (that can hold replica)
  – DataNode is contacted, receives data
  – Forwards to third replica, etc.
HDFS: reading data

1: open
2: get locations
3: read
4: read
5: read
6: close

NameNode 'master:54310'

**Cluster Summary**

71 files and directories, 538 blocks = 609 total. Heap Size is 290.75 MB / 888.94 MB (32%)

- Configured Capacity: 3.49 TB
- DFS Used: 96.38 GB
- Non DFS Used: 385.03 GB
- DFS Remaining: 3.02 TB
- DFS Used%: 2.7%
- DFS Remaining%: 86.53%
- **Live Nodes**: 4
- **Dead Nodes**: 0
- **Decommissioning Nodes**: 0
- Number of Under-Replicated Blocks: 3

**NameNode Storage:**

<table>
<thead>
<tr>
<th>Storage Directory</th>
<th>Type</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>/hdfs/tmp/dfs/name</td>
<td>IMAGE_AND_EDITS</td>
<td>Active</td>
</tr>
</tbody>
</table>

This is [Apache Hadoop](http://hadoop.apache.org) release 1.1.2
MR job execution in Hadoop

MR job execution in Hadoop

1: run client JVM

2: get new job ID

3: copy job resources

4: submit job

Job

Shared Filesystem (e.g., HDFS)

JobTracker

5: init job

6: retrieve input splits

... tasktracker node ...

Distributed Data Management, SoSe 2013, S. Michel
MR job execution in Hadoop (2)

JobTracker

TaskTracker

Child

Shared Filesystem (e.g., HDFS)

5: init job

6: retrieve input splits

7: heartbeat (returns task)

8: retrieve job resources

9: launch

10: run

Map or Reduce child JVM

Distributed Data Management, SoSe 2013, S. Michel 22
Job Submission, Initialization, Assignment, Execution

• asks for new job id
• checks if input/output directories exist
• computes input splits
• writes everything to HDFS
• submits job to JobTracker (step 4)

• Retrieves splits (chunks) from HDFS
• Creates for each split a Map task
• TaskTracker is responsible for executing a certain assigned task (multiple on one physical machine)
Hadoop job_201304031252_0007 on master

User: hduser
Job Name: tag count
Job File: hdfs://master.54310/hdfs/tmp/mapred/staging/hduser/staging/job_201304031252_0007/job.xml
Submit Host: master
Submit Host Address: 139.19.252.114
Job-ACLs: All users are allowed
Job Setup: Successful
Status: Running
Started at: Thu Apr 11 12:28:30 CEST 2013
Running for: 2mins, 8sec
Job Cleanup: Pending

<table>
<thead>
<tr>
<th>Kind</th>
<th>% Complete</th>
<th>Num Tasks</th>
<th>Pending</th>
<th>Running</th>
<th>Complete</th>
<th>Killed</th>
<th>Failed/Killed Task Attempts</th>
</tr>
</thead>
<tbody>
<tr>
<td>map</td>
<td>49.99%</td>
<td>508</td>
<td>246</td>
<td>8</td>
<td>254</td>
<td>0</td>
<td>0 / 0</td>
</tr>
<tr>
<td>reduce</td>
<td>15.28%</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0 / 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Counter</th>
<th>Map</th>
<th>Reduce</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLOTS_MILLIS_MAPS</td>
<td>0</td>
<td>0</td>
<td>960,420</td>
</tr>
<tr>
<td>Launched reduce tasks</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Launched map tasks</td>
<td>0</td>
<td>0</td>
<td>262</td>
</tr>
<tr>
<td>Data-local map tasks</td>
<td>0</td>
<td>0</td>
<td>262</td>
</tr>
<tr>
<td>Bytes Read</td>
<td>17,049,714,688</td>
<td>0</td>
<td>17,049,714,688</td>
</tr>
<tr>
<td>HDFS_BYTES_READ</td>
<td>17,049,743,898</td>
<td>0</td>
<td>17,049,743,898</td>
</tr>
<tr>
<td>FILE_BYTES_WRITTEN</td>
<td>38,398,954</td>
<td>49,434</td>
<td>38,448,388</td>
</tr>
<tr>
<td>Reduce input groups</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Map output materialized bytes</td>
<td>25,815,904</td>
<td>0</td>
<td>25,815,904</td>
</tr>
<tr>
<td>Combine output records</td>
<td>1,389,752</td>
<td>0</td>
<td>1,389,752</td>
</tr>
<tr>
<td>Map input records</td>
<td>12,797,588</td>
<td>0</td>
<td>12,797,588</td>
</tr>
<tr>
<td>Reduce shuffle bytes</td>
<td>0</td>
<td>15,779,308</td>
<td>15,779,308</td>
</tr>
<tr>
<td>Physical memory (bytes) snapshot</td>
<td>60,081,565,696</td>
<td>128,823,296</td>
<td>60,210,388,992</td>
</tr>
<tr>
<td>Reduce output records</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Stragglers and Speculative Execution

• JobTracker continuously controls progress (see Web user interface)

• Stragglers are slow nodes
  – have to wait for the slowest one (think: only one out of 1000 is slow and delays overall response time)

• Speculative execution
  – run same task on more nodes
  – the first one who finishes wins
  – wasted resources vs. improved performance
Job Scheduling

• FIFO

• + Priorities
  – VERY_HIGH, HIGH, NORMAL, LOW, VERY_LOW

• But once a job is running and a job of higher priority arrives, the lower one is not stopped
Typical Setup

Node 1
Disks

Node 2
Disks

Node 3
Disks

Node 4
Disks

Node 5
Disks

Node 6
Disks

Rack 1

Rack 2

Switch

Locality

- data-local
- rack-local
- off-rack

map tasks

Cost Model + Configuration for Rack Awareness

- **Cost model** applied in Hadoop:
  - Same node: 0
  - Same rack: 2
  - Same data center: 4
  - Different data center: 6

- Hadoop needs help: You have to specify config. (topology)

- Sample configuration:
  '13.2.3.4' : '/datacenter1/rack0',
  '13.2.3.5' : '/datacenter1/rack0',
  '13.2.3.6' : '/datacenter1/rack0',
  '10.2.3.4' : '/datacenter2/rack0',
  '10.2.3.4' : '/datacenter2/rack0'

....
Shuffle and Sort

- **Output** of map is partitioned by key as standard
- Reducer is guaranteed to get **entire partition**
- Output of each reducer is sorted also by this key
- Selecting which key to use, hence, affects partitions and sort order (see few slides later how to customize)
Shuffle and Sort

- **map task**
  - Input split
  - Map
  - Buffer in memory
  - Partitions

- **Copy phase**
  - Merge on disk
  - Fetch

- **reduce task**
  - Merge
  - Other maps
  - Other reducers
Shuffle and Sort (Cont’d)

“Sort” phase

Reduce phase

map task

fetch

merge on disk

other maps

other reducers

reduce task

merge

merge

merge

reduce

output

mixture of in-memory and on-disk data
Partitioning, Grouping, Sorting

• Consider weather data, temperature for each day. Want: maximum temp per year
• So, want data per year sorted by temp:

<table>
<thead>
<tr>
<th>Year</th>
<th>Temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1900</td>
<td>35°C</td>
</tr>
<tr>
<td>1900</td>
<td>34°C</td>
</tr>
<tr>
<td>1900</td>
<td>34°C</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>1901</td>
<td>36°C</td>
</tr>
<tr>
<td>1901</td>
<td>35°C</td>
</tr>
</tbody>
</table>

• Idea: **composite key**: (year, temp)

*example source: T. White, Hadoop, The Definitive Guide, 3rd edition*
Partitioning, Grouping, Sorting (Cont’d)

• Obviously, doesn’t work: (1900, 35°C) and (1900, 34°C) end up at different partitions

• Solution(?): Write a custom partitioner that considers year as partition and sort comparator for sorting by temperature
Need for Custom Grouping

• With that custom partitioner (by year) and temp as key we get

<table>
<thead>
<tr>
<th>Year</th>
<th>Temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1900</td>
<td>35°C</td>
</tr>
<tr>
<td>1900</td>
<td>34°C</td>
</tr>
<tr>
<td>1900</td>
<td>34°C</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• Problem: reducer still consumes groups by key (within correct partitions)
Custom Grouping

• Solution: Define custom grouping method (class) that considers temperature for grouping

<table>
<thead>
<tr>
<th>Partition</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>1900</td>
<td>35°C</td>
</tr>
<tr>
<td>1900</td>
<td>34°C</td>
</tr>
<tr>
<td>1900</td>
<td>34°C</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>1901</td>
<td>36°C</td>
</tr>
<tr>
<td>1901</td>
<td>35°C</td>
</tr>
</tbody>
</table>
Additional Combiner

• Map phase might output large amounts of data that could be reduced already locally
• As network bandwidth is often limiting factor
• Works for functions like: $\text{max}(1,2,6,2,1,9) = \text{max}(\text{max}(1,2,6), \text{max}(2,1,9))$
• Add combiner to be run on map output.
• Usually, same as reducer (code)

• Not a replacement of reducer (as it sees only local information!)
Combiner Caveats

• Note that some aggregates can’t be done locally.
  – like: output if $\text{sum(value)} > \text{threshold}$. Why? Can’t decide that threshold crossing because it sees only local info.
    • Note: this application makes still a good case for the combiner, but it should just sum up the local values and not “prune” based on threshold. So, it is different from the final reducer.
  – if aggregation function is not associative “$((x*y)*z = x*(y*z))$” and commutative “$(x*y = y*x)$”
  – also problematic: average (but can be fixed: reducer need to know also the number of items then)
Flexible: Hadoop Streaming

$HADOOP_HOME/bin/hadoop jar
$HADOOP_HOME/hadoop-streaming.jar
  -input myInputDirs
  -output myOutputDir
    -mapper /bin/cat
    -reducer /bin/wc
Compression

• Massive amounts of data is sent over network
• And read from disk

• Opens optimization in form of data compression.
  – Data can be read already from compressed files
  – Map output can be compressed
Failure/Recovery in MR

• Tasktracker failure:
  – detected by master through periodic heartbeats
  – can also be black listed if too many failures occur
  – just restart if dead.
  – Jobtracker re-schedules tasks

• Master failure:
  – unlikely to happen (only one machine) but if: all running jobs failed
  – improved in Hadoop 2.x (YARN)
And Specifically in HDFS

- NameNode marks DataNodes without recent Heartbeats as dead
- Replication factor of some blocks can fall below their specified value
- The NameNode constantly tracks which blocks need to be replicated and initiates replication whenever necessary.
- If NameNode crashed: Manual restart/recovery.
Hadoop: Read on


- ...
APPLICATION OF MAPREDUCE TO DATA MANAGEMENT
Data Management with MapReduce

• Now after the intro to MapReduce/Hadoop, we can perform computations (beyond simple wordcount)

• How to implement common SQL Joins
• How to compute n-gram statistics
• How to compute PageRank
(Equi) Joins in Map Reduce

• Two relations R(A,B) and S(B,C):

  SELECT *
  FROM R, S
  WHERE R.b = S.b

• Task: Join partners have to end up at same node.
Example

<table>
<thead>
<tr>
<th>Station ID</th>
<th>Station Name</th>
<th>Timestamp</th>
<th>Temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>12434343434300</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>12434343434500</td>
<td>27</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>12434343434700</td>
<td>31</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>12434343434900</td>
<td>28</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>12434343435200</td>
<td>29</td>
</tr>
</tbody>
</table>
Reduce Side Join

• Two relations $R(A,B)$ and $S(B,C)$.

• Map:
  – Send tuple $t$ to reducer of key $t.b$
  – And information where $t$ is from (R or S)

• Reduce:
  – Join tuples $t_1$, $t_2$ with $t_1.b = t_2.b$ and $t_1$ in $R$ and $t_2$ in $S$
Map Side Join with one entirely known Relation

• Two relations $R(A,B)$ and $S(B,C)$.
• One relation is small, say $R$

• Map:
  – each map process knows entire relation $R$
  – can perform join on subset of $S$
    • output joined tuple
• Reduce:
  – no reducer needed
Reduce Side Join with “Semi Join” Optimization (Filtering) in Map Phase

- Two relations $R(A,B)$ and $S(B,C)$.
- Unique values in $R.B$ are small in number

- Map:
  - knows unique ids of $R.B$
  - send tuples $t$ in $R$ by key $t.b$
  - send tuples $t$ in $S$ only if $t.b$ in $R.B$

- Reduce:
  - perform the actual join
Global Sharing of Information

• Implemented as “Distributed Cache”
• For small data
• E.g., dictionary or “stopwords” file for text processing, or “small” relation for joins

• Read at initialization of Mapper
Reduce Side Join with Map side filtering, but now with Bloom Filters

- Reduce-side join with Map-side filtering

- Compact representation of join attributes

- Using Bloom Filter*
  
  – very generic data structure with wide applications to distributed data management / systems

- Will see them later again (so worth introducing)

Bloom Filter

• Bit array of size m (all bits=0 initially)
• Encode elements of a set in that array
  – set is for instance the distinct attributes of table column or a set of words. How to hash non-numbers? E.g., use byte representation of string

• How is the bit array constructed?
  – Hash element to bucket no. and set this bit to 1
    (If the bit is already 1, ok, keep it 1)
  – Use multiple (=k) hash functions \( h_i \)
Bloom Filter: Insert + Query

h1(x) = 3*x mod 8
h2(x) = 5*x mod 8

• **Query:** is \( x \) contained in the set (=filter)?
  
  – Check if bits at both \( h1(x) \) and \( h2(x) \) are set to 1. Yes?
    Then \( x \) ”might be” in the set. No? Then \( x \) is **for sure** not in!

---

Distributed Data Management, SoSe 2013, S. Michel
Bloom Filter: False Positives

• In case all bits at hash positions are 1, the element might be in, but maybe it’s a mistake.

• Is $x=45$ contained? $h_1(45)=7$ $h_2(45)=1$

• Looks like, but actually it is not! (i.e., we didn’t insert it on the slide before)

• It is a **false positive**!
Bloom Filter: Probability of False Positives

• Bloom Filter of size $m$ (bits)
• $k$ hash functions
• $n$ inserted elements

$$pfp = \left( 1 - \left[ 1 - \frac{1}{m} \right]^{kn} \right)^k \approx (1 - e^{-kn/m})^k.$$

• Thus, can be controlled: tradeoff between compression and “failures“
Implications of False Positives on Join

• Reconsider the reduce-side join with map-side filtering of relations R(A,B) and S(B,C).
• We have a Bloom filter for R.B, etc (see slide before)

• What do false positives cause?
  – additional (and useless network) traffic and also more work for reducer
  – but no erroneous results as reducer will check if the join can in fact be done
Literature

• Jeffrey Dean und Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters“. Google Labs.
• Foto N. Afrati, Jeffrey D. Ullman: Optimizing joins in a map-reduce environment. EDBT 2010: 99-110
• Alper Okcan, Mirek Riedewald: Processing theta-joins using MapReduce. SIGMOD Conference 2011: 949-960
• Rakesh Agrawal, Tomasz Imielinski, Arun N. Swami: Mining Association Rules between Sets of Items in Large Databases. SIGMOD Conference 1993: 207-216
Literature (2)


