“Hadoop”: A Distributed Architecture, FileSystem, & MapReduce

H. Andrew Schwartz

CSE545
Spring 2020
Big Data Analytics, The Class

**Goal:** Generalizations
*A model or summarization* of the data.

**Data Frameworks**
- Hadoop File System
- Spark
- MapReduce
- Streaming
- Tensorflow

**Algorithms and Analyses**
- Similarity Search
- Hypothesis Testing
- Graph Analysis
- Recommendation Systems
- Deep Learning
Big Data Analytics, The Class

- Workflow Systems
- Algorithms
- Statistical Methods
- Distributed Tools

Big Data Analytics
Classical Data Mining
IO Bounded

Reading a word from disk versus main memory: $10^5$ slower!

Reading many contiguously stored words is faster per word, but fast modern disks still only reach 150MB/s for sequential reads.
IO Bounded

Reading a word from disk versus main memory: $10^5$ slower!

Reading many contiguously stored words is faster per word, but fast modern disks still only reach 150MB/s for sequential reads.

IO Bound: biggest performance bottleneck is reading / writing to disk.

starts around 100 GBs: $\sim$10 minutes just to read

200 TBs: $\sim$20,000 minutes = 13 days
Classical Big Data

**Classical focus:** efficient use of disk. e.g. Apache Lucene / Solr

**Classical limitation:** Still bounded when needing to process all of a large file.
Classical Big Data

How to solve?

Classical limitation: Still bounded when needing to process all of a large file.
Distributed Architecture

Switch ~1Gbps

Rack 1

Switch ~1Gbps

Rack 2

Switch ~10Gbps
In reality, modern setups often have multiple CPUs and disks per server, but we will model as if one machine per CPU-disk pair.
Distributed Architecture (Cluster)
Challenges for IO Cluster Computing

1. Nodes fail
   1 in 1000 nodes fail a day

2. Network is a bottleneck
   Typically 1-10 Gb/s throughput

3. Traditional distributed programming is often ad-hoc and complicated
Distributed Architecture (Cluster)

Challenges for IO Cluster Computing

1. Nodes fail
   - 1 in 1000 nodes fail a day
     Duplicate Data

2. Network is a bottleneck
   - Typically 1-10 Gb/s throughput
     Bring computation to nodes, rather than data to nodes.

3. Traditional distributed programming is often ad-hoc and complicated
   Stipulate a programming system that can easily be distributed
Distributed Architecture (Cluster)

Challenges for IO Cluster Computing

1. Nodes fail
   - 1 in 1000 nodes fail a day
   - **Duplicate Data**

2. Network is a bottleneck
   - Typically 1-10 Gb/s throughput
   - **Bring computation to nodes, rather than data to nodes.**

3. Traditional distributed programming is often ad-hoc and complicated
   - **Stipulate a programming system that can easily be distributed**

**HDFS/MapReduce Accomplishes**
The effectiveness of MapReduce is in part simply due to use of a distributed filesystem!
Distributed Filesystem

Characteristics for Big Data Tasks

Large files (i.e. >100 GB to TBs)
Reads are most common
No need to update in place
(append preferred)
Distributed Filesystem

(e.g. Apache Hadoop DFS, GoogleFS, EMRFS)

C, D: Two different files

(chunk server 1) (chunk server 2) (chunk server 3) ...

(chunk server n)

(Leskovec at al., 2014; http://www.mmds.org/)
Distributed File System

(e.g. Apache HadoopDFS, GoogleFS, EMFS)

C, D: Two different files

chunk server 1  chunk server 2  chunk server 3  ...  chunk server n

“Hadoop” was named after a toy elephant belonging to Doug Cutting’s son. Cutting was one of Hadoop’s creators.

(Leskovec at al., 2014; http://www.mmds.org/)
Distributed Filesystem

(e.g. Apache HadoopDFS, GoogleFS, EMRFS)

C, D: Two different files

(chunk server 1)  (chunk server 2)  (chunk server 3)  (chunk server n)

(Leskovec et al., 2014; http://www.mmds.org/)
Distributed Filesystem

(e.g. Apache HadoopDFS, GoogleFS, EMRFS)

C, D: Two different files

(chunk server 1)  (chunk server 2)  (chunk server 3)  (chunk server n)

(Leskovec at al., 2014; http://www.mmds.org/)
Distributed Filesystem

Chunk servers (on Data Nodes)

- 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

(Leskovec at al., 2014; http://www.mmds.org/)
Components of a Distributed Filesystem

Chunk servers (on Data Nodes)

- 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

Name node (aka master node)

- Stores metadata about where files are stored
- Might be replicated or distributed across data nodes.

(Leskovec at al., 2014; http://www.mmds.org/)
Components of a Distributed Filesystem

Chunk servers (on Data Nodes)
- 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

Name node (aka master node)
- Stores metadata about where files are stored
- Might be replicated or distributed across data nodes.

Client library for file access
- Talks to master to find chunk servers
- Connects directly to chunk servers to access data

(Leskovec at al., 2014; http://www.mmds.org/)
Distributed Architecture (Cluster)

Challenges for IO Cluster Computing

1. Nodes fail
   1 in 1000 nodes fail a day
   Duplicate Data (Distributed FS)

2. Network is a bottleneck
   Typically 1-10 Gb/s throughput
   Bring computation to nodes, rather than data to nodes.

3. Traditional distributed programming is often ad-hoc and complicated
   Stipulate a programming system that can easily be distributed
What is MapReduce

*noun.* 1 - A *style of programming*

input chunks => **map tasks** | **group_by keys** | **reduce tasks** => output

“|” is the linux “pipe” symbol: passes stdout from first process to stdin of next.
What is MapReduce

**noun.1 - A style of programming**

input chunks => map tasks | group_by keys | reduce tasks => output

“|” is the linux “pipe” symbol: passes stdout from first process to stdin of next.

E.g. counting words:

```bash
tokenize(document) | sort | uniq -c
```
What is MapReduce

*noun.1* - A *style of programming*

input chunks => map tasks | group_by keys | reduce tasks => output

“|” is the linux “pipe” symbol: passes stdout from first process to stdin of next.

E.g. counting words:

```
  tokenize(document) | sort | uniq -c
```

*noun.2* - A *system* that distributes MapReduce style programs across a distributed file-system.

(e.g. Google’s internal “MapReduce” or apache.hadoop.mapreduce with hdfs)
What is MapReduce
What is MapReduce

- Input chunks
- Key-value pairs (k, v)
- Group by keys
- Keys with all their values (k, [v, w, ...])
- Reduce tasks
- Combined output

Map

extract what you care about.
line => (k, v)
What is MapReduce

- **Map**: sort and shuffle many \((k, v) \Rightarrow (k, [v_1, v_2], \ldots)\)
- **Group by keys**
- **Reduce tasks**
- **Combined output**

- **Input chunks**

- **Keys with all their values** \((k, [v, w, \ldots])\)

---

extract what you care about.
Map

Reduce

Summarize aggregate

What is MapReduce
What is MapReduce

Easy as 1, 2, 3!

Step 1: Map  Step 2: Sort / Group by  Step 3: Reduce
What is MapReduce

Easy as 1, 2, 3!

Step 1: Map

Step 2: Sort / Group by

Step 3: Reduce

(Leskovec at al., 2014; http://www.mmds.org/)
(1) The Map Step

(Leskovec et al., 2014; http://www.mmds.org)
(2) The Sort / Group-by Step

Intermediate key-value pairs

Key-value groups

Group by key

...
(3) The *Reduce* Step

(Leskovec at al., 2014; http://www.mmds.org/)
What is MapReduce

Easy as 1, 2, 3!

Step 1: Map

Input key-value pairs

\[ \begin{align*}
  k &\quad v \\
  k &\quad v \\
  \vdots \\
  k &\quad v
\end{align*} \]

\text{map}

Step 2: Sort / Group by

Intermediate key-value pairs

\[ \begin{align*}
  k &\quad v \\
  k &\quad v \\
  \vdots \\
  k &\quad v
\end{align*} \]

Group by key

\text{reduce}

Step 3: Reduce

Key-value groups

\[ \begin{align*}
  k &\quad v \\
  k &\quad v \\
  \vdots \\
  k &\quad v
\end{align*} \]

\text{reduce}

Output key-value pairs

\[ \begin{align*}
  k &\quad v \\
  k &\quad v \\
  \vdots \\
  k &\quad v
\end{align*} \]

(Leskovec at al., 2014; http://www.mmds.org/)
What is MapReduce

Map: \( (k, v) \rightarrow (k', v') * \)
(Written by programmer)

Group by key: \((k_1', v_1'), (k_2', v_2'), \ldots \rightarrow (k_1', (v_1', v', \ldots)), \)
(system handles)

\((k_2', (v_1', v', \ldots)), \ldots \)

Reduce: \((k', (v_1', v', \ldots)) \rightarrow (k', v'') * \)
(Written by programmer)
Example: Word Count

tokenize(document) | sort | uniq -c
Example: Word Count

tokenize(document) | sort | uniq -c

Map: extract what you care about.

sort and shuffle

Reduce: aggregate, summarize
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.........................
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 .................."
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 .......
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.
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 ..."
Example: Word Count

@abstractmethod
def map(k, v):
    pass

@abstractmethod
def reduce(k, vs):
    pass
Example: Word Count (v1)

```python
def map(k, v):
    for w in tokenize(v):
        yield (w, 1)

def reduce(k, vs):
    return len(vs)
```
def map(k, v):
    for w in tokenize(v):
        yield (w, 1)

def tokenize(s):
    # simple version
    return s.split(‘ ‘)

def reduce(k, vs):
    return len(vs)
Example: Word Count (v2)

```python
def map(k, v):
    counts = dict()
    for w in tokenize(v):
        counts[w] += 1  # counts each word within the chunk
                      # (try/except is faster than
                      #  "if w in counts")
```

def map(k, v):
    counts = dict()
    for w in tokenize(v):
        try:
            counts[w] += 1
        except KeyError:
            counts[w] = 1
    for item in counts.items():
        yield item

counts each word within the chunk (try/except is faster than “if w in counts”)

Example: Word Count (v2)

```python
def map(k, v):
    counts = dict()
    for w in tokenize(v):
        try:
            counts[w] += 1
        except KeyError:
            counts[w] = 1
    for item in counts.items():
        yield item

def reduce(k, vs):
    return sum(vs)
```

- `counts = dict()` initializes an empty dictionary.
- `for w in tokenize(v):` iterates over each word in the input string.
- `try:` block increments the count of the word in the dictionary `counts` if the word already exists.
- `except KeyError:` block initializes the count to 1 if the word is not found.
- `for item in counts.items():` iterates over the dictionary items and yields each item.

This code snippet counts each word within the chunk and returns the sum of counts from different chunks.
Challenges for IO Cluster Computing

1. Nodes fail
   1 in 1000 nodes fail a day
   **Duplicate Data (Distributed FS)**

2. Network is a bottleneck
   Typically 1-10 Gb/s throughput
   Bring computation to nodes, rather than data to nodes.

3. Traditional distributed programming is often ad-hoc and complicated
   Stipulate a programming system that can easily be distributed
Challenges for IO Cluster Computing

1. Nodes fail
   1 in 1000 nodes fail a day
   Duplicate Data (Distributed FS)

2. Network is a bottleneck
   Typically 1-10 Gb/s throughput
   Bring computation to nodes, rather than data to nodes. (Sort and Shuffle)

3. Traditional distributed programming is often ad-hoc and complicated
   Stipulate a programming system that can easily be distributed
Distributed Architecture (Cluster)

Challenges for IO Cluster Computing

1. Nodes fail
   - 1 in 1000 nodes fail a day
     - **Duplicate Data (Distributed FS)**

2. Network is a bottleneck
   - Typically 1-10 Gb/s throughput
     - Bring computation to nodes, rather than data to nodes. **(Sort and Shuffle)**

3. Traditional distributed programming is often ad-hoc and complicated
   - **(Simply define a map and reduce)**
     - Stipulate a programming system that can easily be distributed
Example: Relational Algebra

Select  Project  Union, Intersection, Difference  Natural Join  Grouping
Example: Relational Algebra

Select

Project

Union, Intersection, Difference

Natural Join

Grouping
Example: Relational Algebra

Select

$R(A_1, A_2, A_3, \ldots)$, Relation $R$, Attributes $A_*$

return only those attribute tuples where condition $C$ is true
Example: Relational Algebra

**Select**

$R(A_1, A_2, A_3, ...)$, Relation $R$, Attributes $A_*$

return only those attribute tuples where condition $C$ is true

def map(k, v):
    # v is list of attribute tuples
    for t in v:
        if t satisfies $C$:
            yield (t, t)

def reduce(k, vs):
    For each v in vs:
        yield (k, v)
Example: Relational Algebra

Natural Join

Given $R_1$ and $R_2$ return $R_{join}$ -- union of all pairs of tuples that match given attributes.
Example: Relational Algebra

Natural Join

Given $R_1$ and $R_2$ return $R_{\text{join}}$ -- union of all pairs of tuples that match given attributes.

```python
def map(k, v):
    # k \in \{R1, R2\}, v is (R_1=(A, B), R_2=(B, C)); B are matched attributes
    if k=="R1":
        (a, b) = v
        yield (b,(R_1,a))
    if k=="R2":
        (b,c) = v
        yield (b,(R_2,c))
```
Example: Relational Algebra

Natural Join

Given $R_1$ and $R_2$ return $R_{join}$ -- union of all pairs of tuples that match given attributes.

def map(k, v):
    # k \in \{R1, R2\}, v is (R_1=(A, B), R_2=(B, C)); B are matched attributes
    if k==”R1”:
        (a, b) = v
        yield (b,(R_1,a))
    if k==”R2”:
        (b,c) = v
        yield (b,(R_2,c))

def reduce(k, vs):
    r1, r2 = [], []
    for (S, x) in vs: #separate rs
        if S == r1: r1.append(x)
        else: r2.append(x)
    for a in r1: #join as tuple
        for each c in r2:
            yield (R_{join}, (a, k, c)) #k is
Data Flow

Input: Big document

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

Intermediate:
M M M M M

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

Grouped:
M M M M M

Reduce:
Collect all values belonging to the key and output

Output:

Data Flow

(Map Task 1)

Partitioning Function

(k1:v, k1:v, k2:v, k1:v)

Sort and Group

(k2:v, k4:v, v, v, k5:v)

Reduce Task 1

(R, R, R)

(Map Task 2)

Partitioning Function

(k3:v, k4:v, k4:v, k5:v)

Sort and Group

(k4:v, v, v, v, k3:v, v, k1:v)

Reduce Task 2

(R, R)

(Leskovec et al., 2014; http://www.mmds.org/)
Data Flow

(Map Task 1) k1:v k1:v k2:v k1:v

Partitioning Function

(Map Task 2) k3:v k4:v k4:v k5:v

Partitioning Function

(Map Task 3) k4:v k1:v k3:v

Partitioning Function

Sort and Group

k2:v k4:v,v,v k5:v

Reduce Task 1

R R R

Reduce Task 2

R R

Programmed

hash

(Leskovec et al., 2014; http://www.mmds.org/)
Data Flow

DFS ➞ Map ➞ Map’s Local FS ➞ Reduce ➞ DFS
Data Flow

MapReduce system handles:

- Partitioning
- Scheduling map / reducer execution
- Group by key

- Restarts from node failures
- Inter-machine communication
Data Flow

- Schedule map tasks near physical storage of chunk
- Intermediate results stored locally
- Master / Name Node coordinates
Data Flow

- Schedule map tasks near physical storage of chunk
- Intermediate results stored locally
- Master / Name Node coordinates
  - Task status: idle, in-progress, complete
  - Receives location of intermediate results and schedules with reducer
  - Checks nodes for failures and restarts when necessary
    - All map tasks on nodes must be completely restarted
    - Reduce tasks can pickup with reduce task failed
Data Flow

- Schedule map tasks near physical storage of chunk
- Intermediate results stored locally
- Master / Name Node coordinates
  - Task status: idle, in-progress, complete
  - Receives location of intermediate results and schedules with reducer
  - Checks nodes for failures and restarts when necessary
    - All map tasks on nodes must be completely restarted
    - Reduce tasks can pickup with reduce task failed
Skew: The degree to which certain tasks end up taking much longer than others.

Handled with:

- More reducers than reduce tasks
- More reduce tasks than nodes
Key Question: How many Map and Reduce jobs?
Key Question: *How many Map and Reduce jobs?*

**M:** map tasks, **R:** reducer tasks

**A:** If possible, one chunk per map task

\[ M \gg |\text{nodes}| \approx |\text{cores}| \]

(better handling of node failures, better load balancing)

**R \,<\, M**

(reduces number of parts stored in DFS)
Data Flow

version 1: few reduce tasks
(same number of reduce tasks as nodes)

Reduce tasks represented by
**time to complete task**
(some tasks take much longer)
Data Flow

version 1: few reduce tasks
(same number of reduce tasks as nodes)

Version 2: more reduce tasks
(more reduce tasks than nodes)

Reduce tasks represented by
time to complete task
(some tasks take much longer)
Data Flow

version 1: few reduce tasks (same number of reduce tasks as nodes)

Reduce tasks represented by time to complete task (some tasks take much longer)

version 2: more reduce tasks (more reduce tasks than nodes)

Reduce tasks represented by time to complete task (some tasks take much longer)

Last task completed

Can redistribute these tasks to other nodes

(time the last task now completes much earlier)
Communication Cost Model

How to assess performance?

(1) Computation: Map + Reduce + System Tasks

(2) Communication: Moving (key, value) pairs
Communication Cost Model

How to assess performance?

1. Computation: Map + Reduce + System Tasks

2. Communication: Moving (key, value) pairs

Ultimate Goal: wall-clock Time.
Communication Cost Model

How to assess performance?

(1) Computation: Map + Reduce + System Tasks

- Mappers and reducers often single pass $O(n)$ within node
- System: sort the keys is usually most expensive
- Even if map executes on same node, disk read usually dominates
- In any case, can add more nodes

Ultimate Goal: wall-clock Time.
Communication Cost Model

How to assess performance?

1. Computation: Map + Reduce + System Tasks

2. Communication: Moving key, value pairs
   - Often dominates computation.
   - Connection speeds: 1-10 gigabits per sec;
     HD read: 50-150 gigabytes per sec
   - Even reading from disk to memory typically takes longer than operating on the data.
Communication Cost Model

How to assess performance?

\[
\text{Communication Cost} = \text{input size} + \\
\text{(sum of size of all map-to-reducer files)}
\]

(2) Communication: Moving key, value pairs

Often dominates computation.
- Connection speeds: 1-10 gigabits per sec;
- HD read: 50-150 gigabytes per sec
- Even reading from disk to memory typically takes longer than operating on the data.
Communication Cost Model

How to assess performance?

Communication Cost = input size +
(sum of size of all map-to-reducer files)

(2) Communication: Moving key, value pairs

Often dominates computation.
- Connection speeds: 1-10 gigabits per sec;
  HD read: 50-150 gigabytes per sec
- Even reading from disk to memory typically takes longer than operating on the data.
- Output from reducer ignored because it’s either small (finished summarizing data) or being passed to another mapreduce job.
Communication Cost: Natural Join

R, S: Relations (Tables) \quad R(A, B) \bowtie S(B, C)

Communication Cost = input size +
(sum of size of all map-to-reducer files)

DFS → Map → LocalFS → Network → Reduce → DFS → ?
Communication Cost: Natural Join

\[ \text{R, S: Relations (Tables)} \quad \text{R}(A, B) \bowtie \text{S}(B, C) \]

**Communication Cost =**

input size +

(sum of size of all map-to-reducer files)

```python
def map(k, v):
    if k=="R1":
        (a, b) = v
        yield (b,(R_1,a))
    if k=="R2":
        (b,c) = v
        yield (b,(R_2,c))

def reduce(k, vs):
    r1, r2 = [], []
    for (rel, x) in vs: #separate rs
        if rel == 'R': r1.append(x)
        else: r2.append(x)
    for a in r1: #join as tuple
        for each c in r2:
            yield (R_join', (a, k, c)) #k is
Communicate Cost: Natural Join

\[ R, S: \text{Relations (Tables)} \quad R(A, B) \bowtie S(B, C) \]

**Communication Cost =** input size + (sum of size of all map-to-reducer files)

\[ = |R_1| + |R_2| + (|R_1| + |R_2|) \]

\[ = O(|R_1| + |R_2|) \]

```python
def map(k, v):
    if k==’R1’:
        (a, b) = v
        yield (b,(R_1,a))
    if k==’R2’:
        (b,c) = v
        yield (b,(R_2,c))

def reduce(k, vs):
    r1, r2 = [], []
    for (rel, x) in vs: #separate rs
        if rel == ‘R’: r1.append(x)
        else: r2.append(x)
    for a in r1: #join as tuple
        for each c in r2:
            yield (R_join,, (a, k, c)) #k is
Exercise:

Calculate Communication Cost for “Matrix Multiplication with One MapReduce Step” (see MMDS section 2.3.10)
Performance Refinements:

- Combiners (like word count version 2 but done via reduce)
  - Run reduce right after map from same node before passing to reduce (MapTask can execute)
  - Reduces communication cost

- Backup tasks (aka speculative tasks)
  - Schedule multiple copies of tasks when close to the end to mitigate certain nodes running slow.

- Override partition hash function to organize data
  - E.g. instead of hash(url) use hash(hostname(url))