A Distributed Architecture, FileSystem, & MapReduce

Stony Brook University
CSE545, Fall 2017
Classical Data Mining

- CPU
- Memory
- Disk
Classical Data Mining

- CPU
- Memory (64 GB)
- Disk
Classical Data Mining

- CPU
- Memory (64 GB)
- Disk
Classical Data Mining

- CPU
- Memory (64 GB)
- Disk
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; ~10 minutes just to read).
Classical Big Data Analysis

Often focused on efficiently utilizing the disk.

e.g. Apache Lucene / Solr

Still bounded when needing to process all of a large file.
IO Bound

How to solve?
Distributed Architecture (Cluster)
Distributed Architecture (Cluster)

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
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
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 File System

Before we understand MapReduce, we need to understand the type of file system it is meant to run on.

The filesystem itself is largely responsible for much of the speed up MapReduce provides!
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 File System

(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 Hadoop DFS, GoogleFS, EMRFS)

C, D: Two different files

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

(C₀, C₂)  

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

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

C, D: Two different files

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

**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 File System

**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 et al., 2014; http://www.mmds.org/)
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?

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
What is MapReduce?

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

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?

Map

- Extract what you care about.
- Line => (k, v)

Combined output

- Keys with all their values (k, [v, w, ...])
- Group by keys
- Reduce tasks
What is MapReduce?

Map

- Extract what you care about.

Reduce

- Sort and shuffle
- Many \((k, v) \Rightarrow (k, [v_1, v_2]), \ldots\)
- Group by keys
- Reduce tasks
- Combined output
What is MapReduce?

Map: extract what you care about.
Reduce: aggregate, summarize
What is MapReduce?

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

Input key-value pairs

Intermediate key-value pairs

(Leskovec at al., 2014; http://www.mmds.org/)
The Sort / Group By Step

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

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

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

Map: (k,v) -> (k’, v’)*
(Written by programmer)

Group by key: (k_1’, v_1’), (k_2’, v_2’), ... -> (k_1’, (v_1’, v’, ...)),
(system handles)
(k_2’, (v_1’, v’, ...)), ... 

Reduce: (k’, (v_1’, v’, ...)) -> (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.

Reduce: aggregate, summarize

sort and shuffle
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

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

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

def reduce(k, vs):
    return len(vs)
```
Example: Word Count (version 2)

```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.iteritems():
        yield item

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

- `counts` each word within the chunk (try/except is faster than “if w in counts”)
- 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  **(Sort & Shuffle)**
   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 (Sort & Shuffle)
   Bring computation to nodes, rather than data to nodes.

3. Traditional distributed programming is often ad-hoc and complicated (Simply requires Mapper and Reducer)
   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

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

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

Select

\[ R(A_1, A_2, A_3, \ldots), \text{Relation } R, \text{ Attributes } A_\ast \]

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 $R_{\text{join}}$ -- union of all pairs of tuples that match given attributes.
Example: Relational Algebra

Natural Join

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

```python
def map(k, v):
    # v is $(R_1=(A, B), R_2=(B, C))$; B are matched attributes
    for (a, b) in $R_1$:
        yield (b, $(R_1, a)$)
    for (b, c) in $R_2$:
        yield (b, $(R_2, c)$)
```
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.

def map(k, v): #v is $(R_1=(A, B), R_2=(B, C)); B$ are matched attributes
    for (a, b) in $R_1$:
        yield (b,($R_1$,a))
    for (b, c) in $R_2$:
        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 ($R_{\text{join}}$, (a, k, c)) #k is
Data Flow

**Input**

**Big document**

**Intermediate**

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

**Output**

Data Flow: In Parallel

(Map Task 1)

Map Task 2

Map Task 3

Sort and Group

Reduce Task 1

Reduce Task 2

Partitioning Function

Partitioning Function

Partitioning Function

(R)

(R)

(R)

(R)

(R)

(R)

(R)

(R)

(k1:v, k2:v, k3:v)

(k4:v, k5:v)

(k1:v, k3:v)

(k2:v, k4:v, v)

(k5:v)

(k4:v)

(k1:v, k3:v)

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

(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

DFS → MapReduce → DFS → MapReduce → DFS → DFS
Data Flow

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
Data Flow

**Key Question:** How many Map and Reduce jobs?
Data Flow

Key Question: How many Map and Reduce jobs?

\( M: \) map tasks, \( R: \) reducer tasks

\( R < M \)

(\( R \) < \( M \) reduces number of parts stored in DFS)

\( A: \) If possible, one chunk per map task

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

(better handling of node failures, better load balancing)
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)

Can redistribute these tasks to other nodes

Last task completed

(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

(2) Communication: Moving key-value pairs

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?

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.
Communication Cost Model

How to assess performance?

![Communication Cost Model](image)

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.
Example: Natural Join

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

**Communication Cost** = \quad \text{input size} +
\quad \text{(sum of size of all map-to-reducer files)}
Example: Natural Join

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

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

\[
= |R| + |S| + (|R| + |S|) \\
= O(|R| + |S|)
\]

```python
def map(k, v):
    for (a, b) in R:
        yield (b, ('R', a))
    for (b, c) in S:
        yield (b, ('S', c))

def reduce(k, vs):
    r1, r2 = [], []
    for (rel, x) in vs:
        if rel == 'R': r1.append(x)
        else: r2.append(x)
    for a in r1:
        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)
Last Notes: Further Considerations for MapReduce

- Performance Refinements:
  - Backup tasks (aka speculative tasks)
    - Schedule multiple copies of tasks when close to the end to mitigate certain nodes running slow.
  - Combiners (like word count version 2)
    - Do some reducing from within map before passing to reduce
    - Reduces communication cost
  - Override partition hash function
    E.g. instead of hash(url) use hash(hostname(url))