Indian Institute of Science
Bangalore, India
भारतीय विज्ञान संस्थान
बंगलौर, भारत

SE256:Jan16 (2:1)

L2-4:Programming for Large Datasets
MapReduce

Yogesh Simmhan

13/20/27 Jan, 2016

Learning Objectives

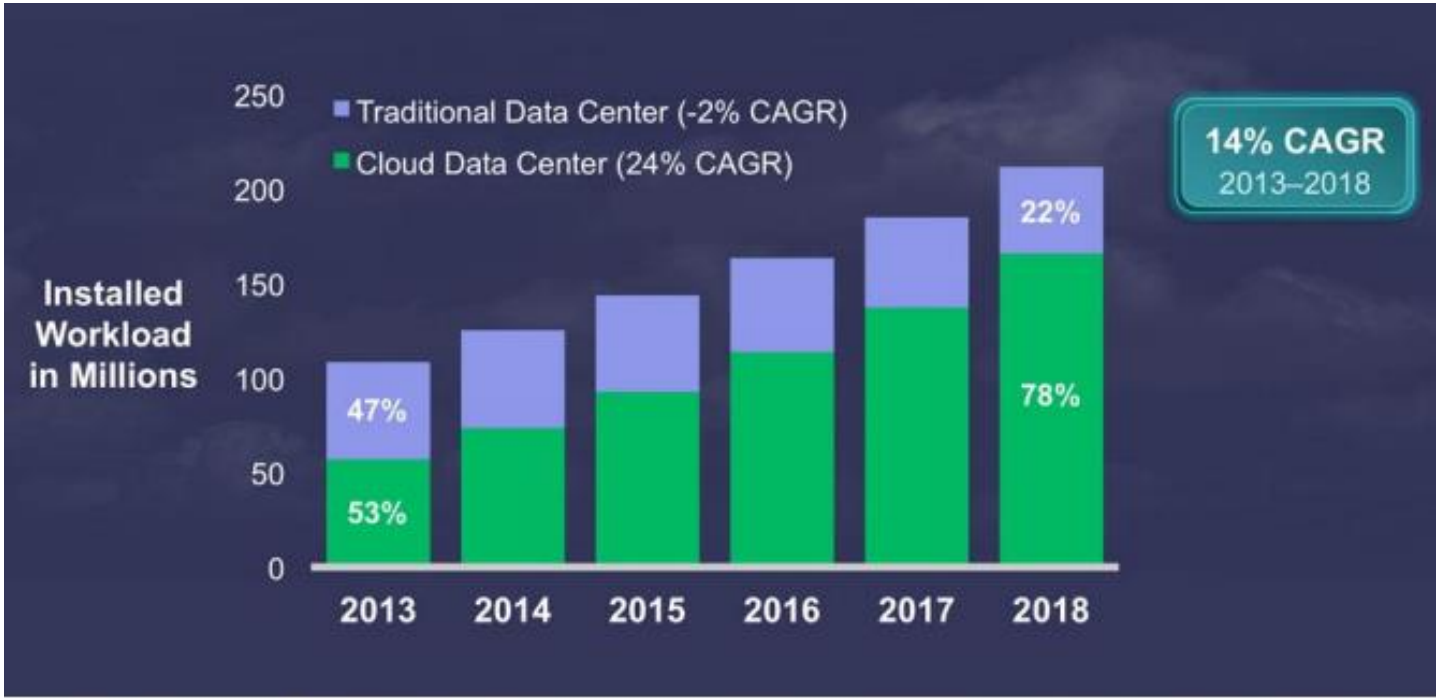
1. Why is MapReduce is useful?
2. How does the MapReduce programming model work?
3. How can you design and write simple MR applications?
4. How can you design and write more advanced MR applications? [L4]

Motivation

Distributed Systems

- Distributed Computing
  - Clusters of machines
  - Connected over network
- Distributed Storage
  - Disks attached to clusters of machines
  - Network Attached Storage
- *How can we make effective use of multiple machines?*
- **Commodity** clusters vs. HPC clusters
  - Commodity: Available off the shelf at large volumes
  - Lower Cost of Acquisition
  - Cost vs. Performance
    - Low disk bandwidth, and high network latency
    - CPU typically comparable (Ceon vs. i3/5/7)
- *How can we use many machines of modest capability?*

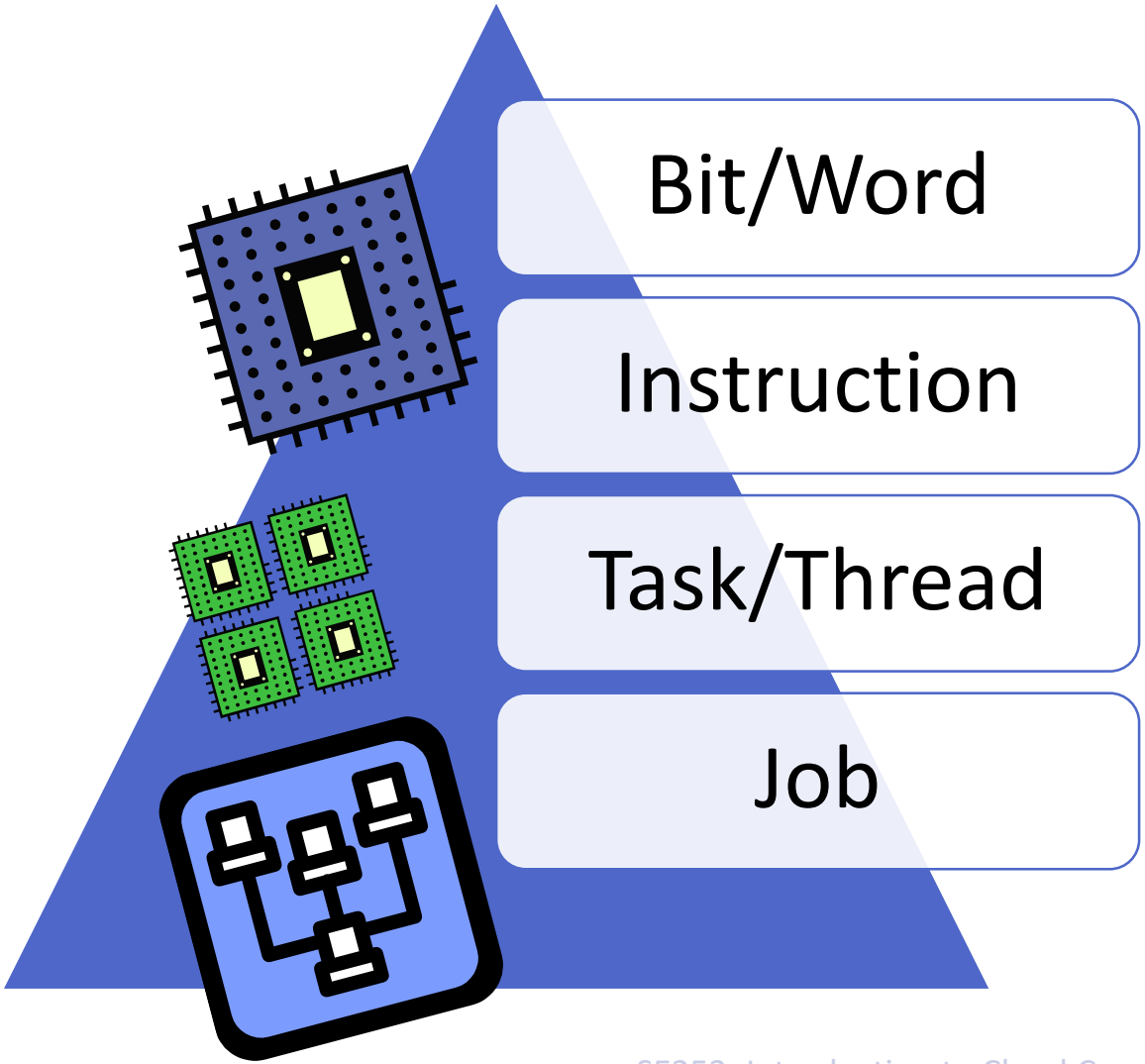
Growth of Commodity Data Centres



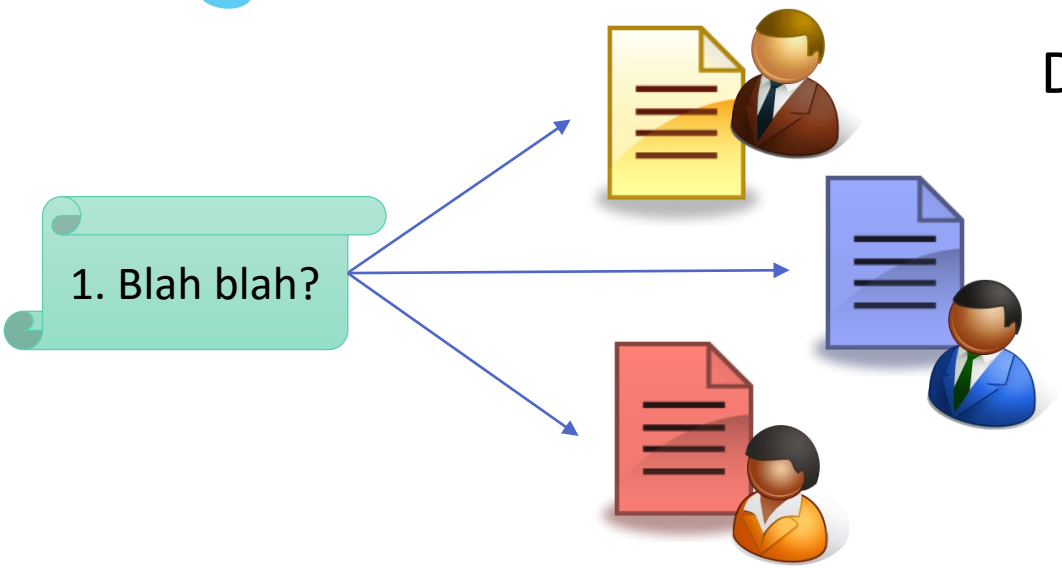
Source: Cisco Global Cloud Index, 2013–2018

http://www.datacenterdynamics.com/app-cloud/cloud-to-eat-traditional-data-centers-cisco/91432.article

Degrees of parallelism

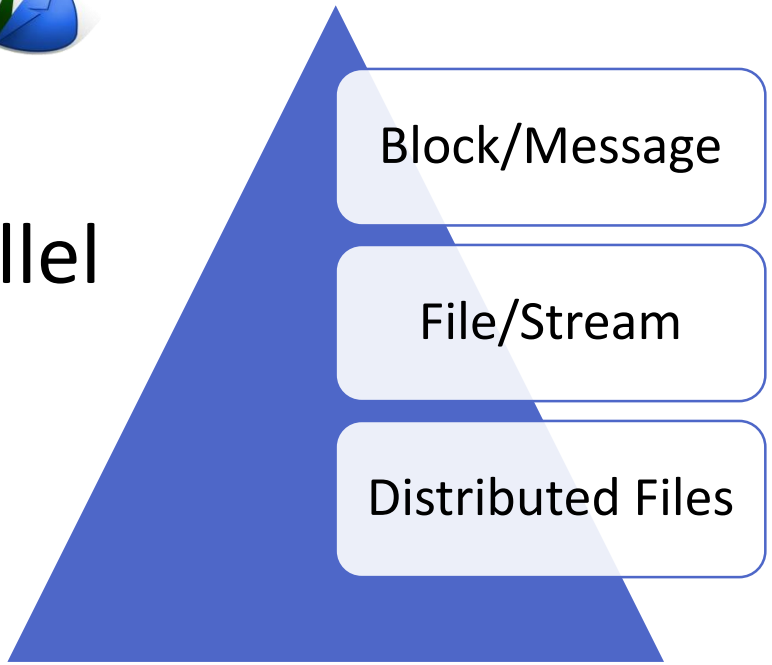


Degrees of Parallelism



Do your review…*collectively?*

- Data parallel vs. Task Parallel
  - Independent processes
  - Independent data dependency
- Tight vs. Loose Coupling



Scalability

- **System Size**: Higher performance when adding more machines
- **Software**: Can framework and middleware work with larger systems?
- **Technology:** Impact of scaling on time, space and diversity
- **Application**: As problem size grows (compute, data), can the system keep up?
- **Vertical vs Horizontal:** ?
- …

Scalability Metric

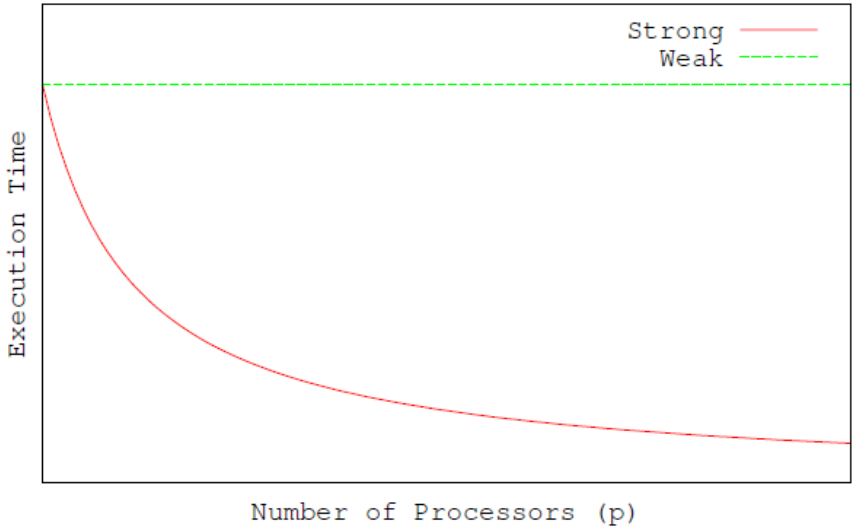
- If the *problem size* is fixed as x and the number of processors available is p

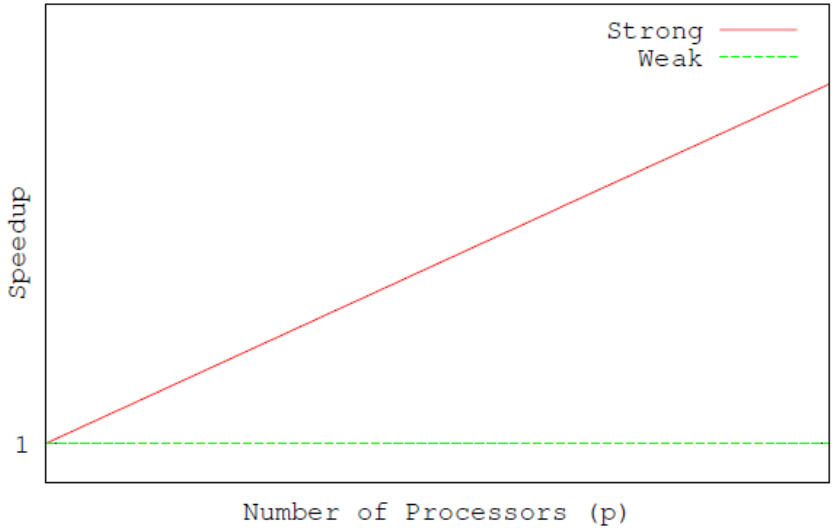
$$Speedup(p, x) = \frac{time(1, x)}{time(p, x)}$$

- If the *problem size per processor* is fixed as x and the number of processors available is p

$$Speedup(p, x.p) = \frac{time(1, x)}{time(p, x.p)}$$

Ideal Strong/Weak Scaling



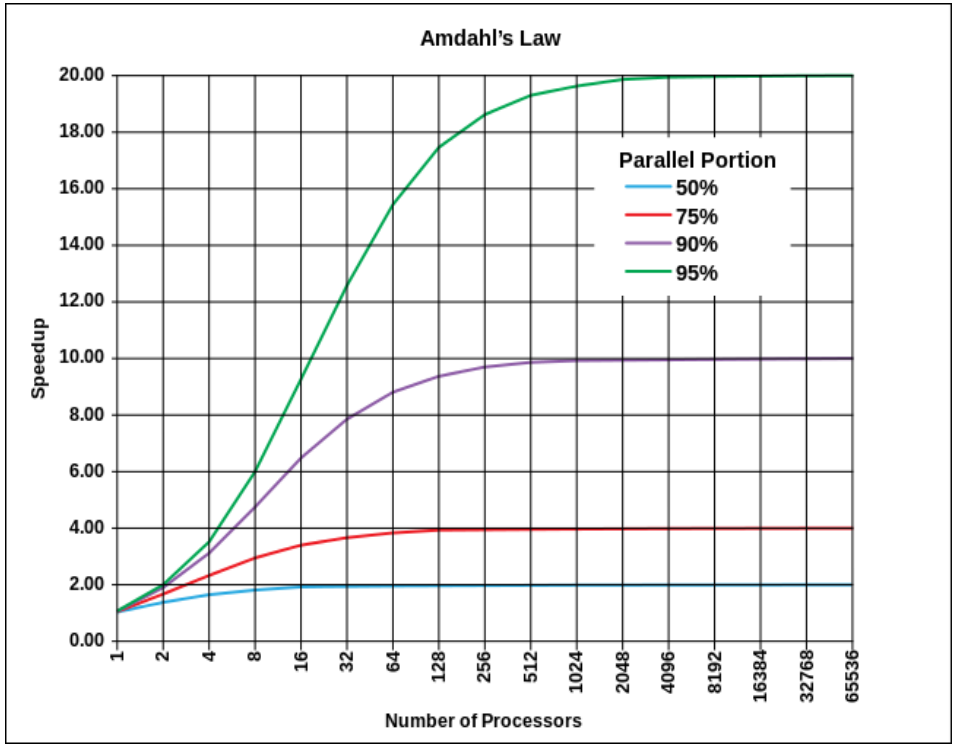


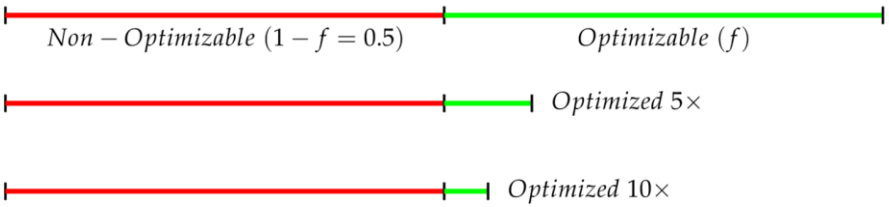
Strong Scaling

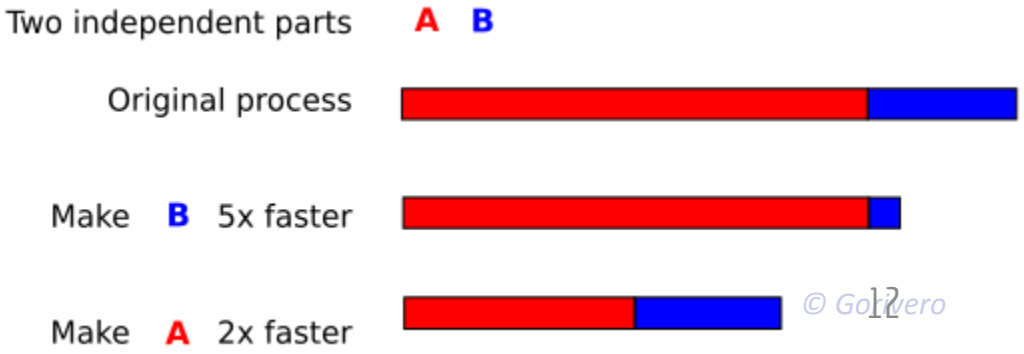
- Amdahl's Law for Application Scalability
  - Total problem size is fixed
  - *Speedup limited by sequential bottleneck*
- f_s is *serial* fraction of application
- f_p is fraction of application that can be *parallelized*
- p is number of processors

$$Speedup(p,x) = \frac{time(1,x)}{time(p,x)} = \frac{1}{f_s + \frac{f_p}{p}}$$

Amdahl's Law







Weak Scaling

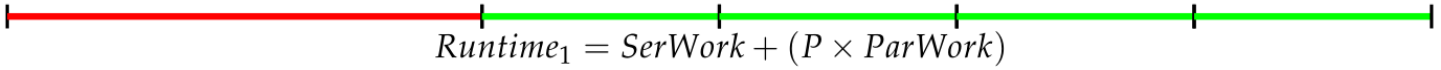
- Gustafson's Law of weak scaling
  - Problem size increases with # of processors
  - "Scaled speedup"
- α is fraction of application that is sequential, *when parallel work per processor is fixed* (**different** from f_s)
- p is number of processors

$$Speedup(p, x.p) = \frac{time(1, x)}{time(p, x.p)}$$

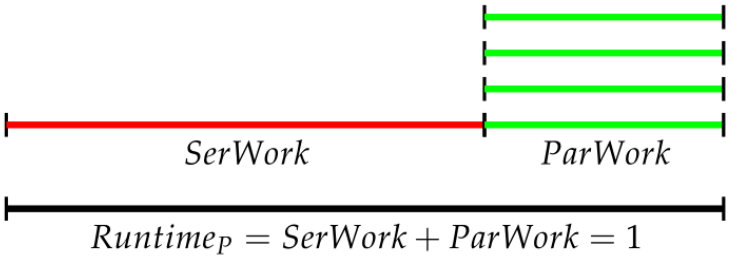
$$= \frac{\alpha + p.(1-\alpha)}{\alpha + \frac{p.(1-\alpha)}{p}}$$

$$= p - \alpha.(p-1)$$

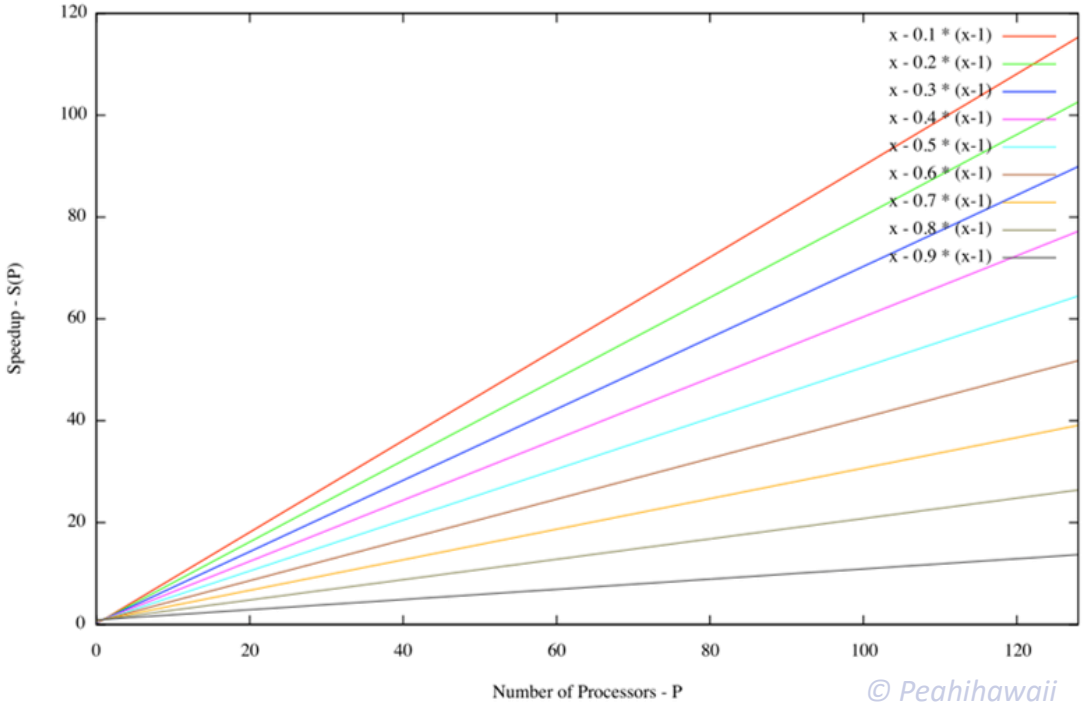
Weak Scaling



$$Runtime_1 = SerWork + (P \times ParWork)$$



$$Runtime_P = SerWork + ParWork = 1$$



Scalability

- Strong vs. Weak Scaling
- **Strong Scaling**: How the performance varies with the # of processors for a *fixed total problem size*
- **Weak Scaling**: How the performance varies with the # of processors for a *fixed problem size per processor*
  - MapReduce is intended for "Weak Scaling"

Ease of Programming

- Programming distributed systems is difficult
  - Divide a job into multiple tasks
  - Understand dependencies between tasks: Control, Data
  - Coordinate and synchronize execution of tasks
  - Pass information between tasks
  - Avoid race conditions, deadlocks
- Parallel and distributed programming models/languages/abstractions/platforms try to make these easy
  - E.g. Assembly programming vs. C++ programming
  - E.g. C++ programming vs. Matlab programming

Availability, Failure

- Commodity clusters have lower reliability
  - Mass-produced
  - Cheaper materials
  - Smaller lifetime (~3 years)
- *How can applications easily deal with failures?*
- *How can we ensure availability in the presence of faults?*

Map Reduce

Patterns & Technologies

- **MapReduce** is a distributed data-parallel programming model from Google
- MapReduce works best with a distributed file system, called **Google File System (GFS)**
- **Hadoop** is the open source framework implementation from Apache that can execute the MapReduce programming model
- **Hadoop Distributed File System (HDFS)** is the open source implementation of the GFS design
- **Elastic MapReduce (EMR)** is Amazon's PaaS

MapReduce

"A simple and powerful interface that enables automatic parallelization and distribution of large-scale computations, combined with an implementation of this interface that achieves high performance on large clusters of commodity PCs."

Dean and Ghermawat, "MapReduce: Simplified Data Processing on Large Clusters", OSDI, 2004

MapReduce Design Pattern

- Clean abstraction for programmers
- Automatic parallelization & distribution

- Fault-tolerance
- A batch data processing system
- Provides status and monitoring tools

MapReduce: Data-parallel Programming Model

- Process data using map & reduce functions
- **map(k_i, v_i) → List<k_m, v_m>[]**
  - ***map*** is called on every input item
  - Emits a series of intermediate key/value pairs
- All values with a given key are **grouped** together
- **reduce(k_m, List<v_m>[]) → List<k_r, v_r>[]**
  - ***reduce*** is called on every unique key & all its values
  - Emits a value that is added to the output

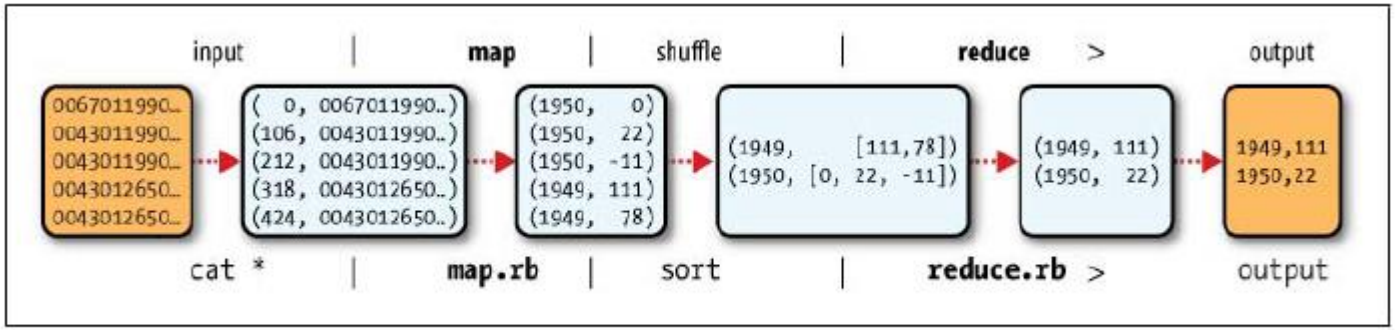


Figure 2-1. MapReduce logical data flow

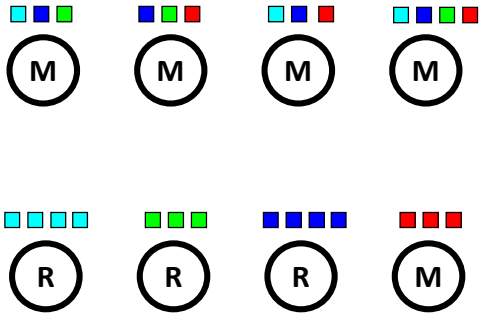
MR Borrows from Functional Programming

- Functional operations do not modify data structures
  - *They always create new ones*
  - Original data still exists in unmodified form (read only)
- Data flows are implicit in program design
- Order of operations does not matter
  - *Commutative*: **a ◊ b ◊ c = b ◊ a ◊ c = c ◊ b ◊ a**

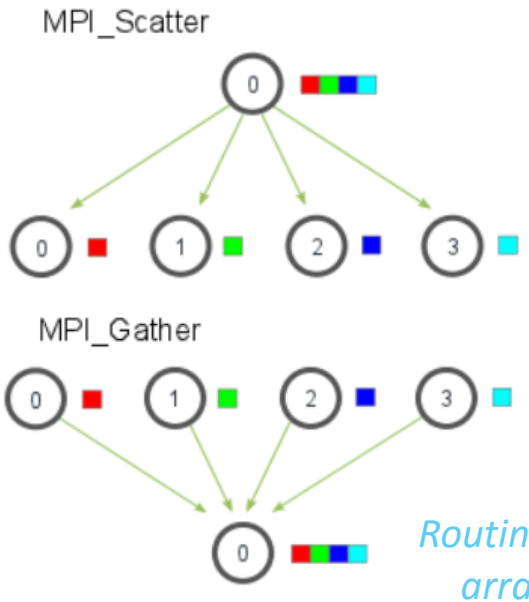
MR Borrows from Functional Programming

- In a purely functional setting
  - Elements computed by **map** cannot see the effects of map on other elements
  - Order of applying **reduce** is *commutative*
    - **a ◊ b = b ◊ a**
    - Allowing parallel/reordered execution
  - More optimizations possible if **reduce** is also *associative*
    - **(a ◊ b) ◊ c = a ◊ (b ◊ c)**

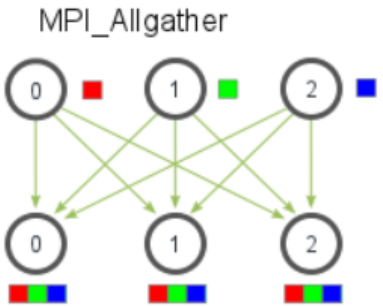
MapReduce & MPI Scatter-Gather



Routing determined by key



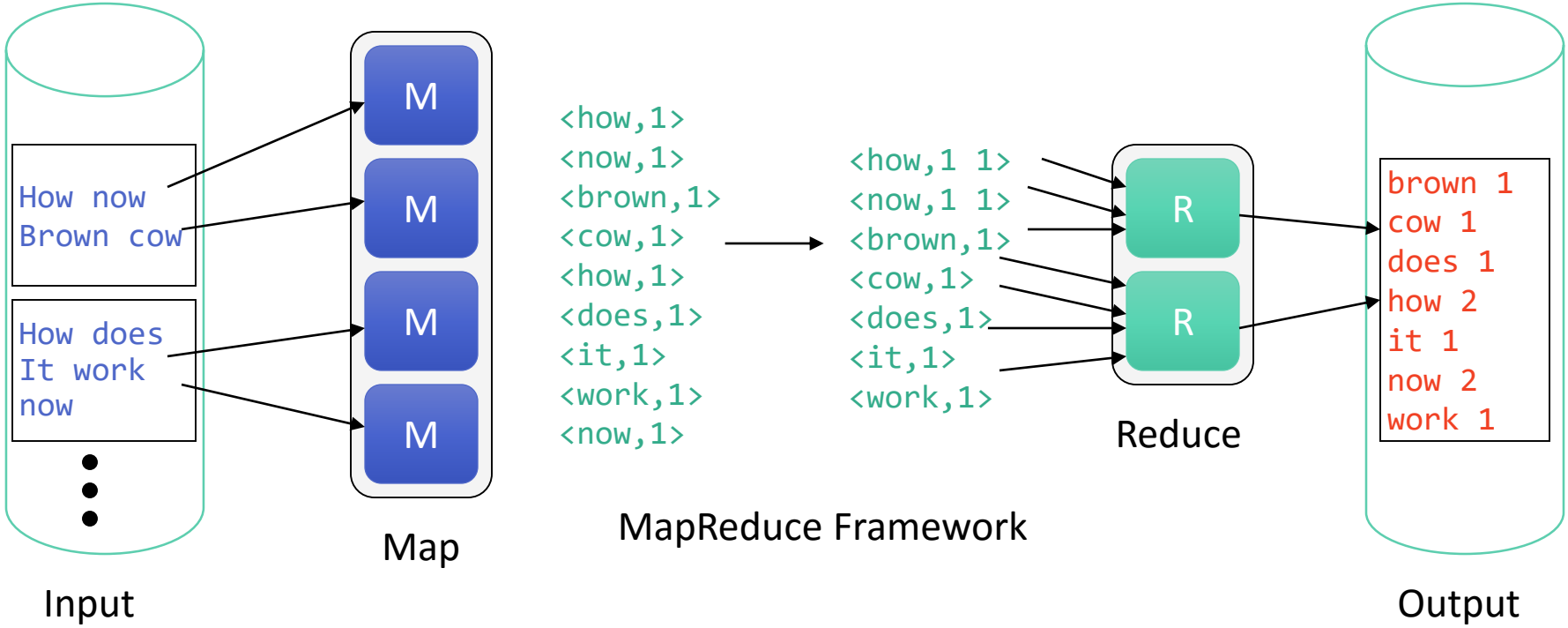
Routing determined by array index/element position



http://mpitutorial.com/mpi-scatter-gather-and-allgather/

MapReduce: Word Count

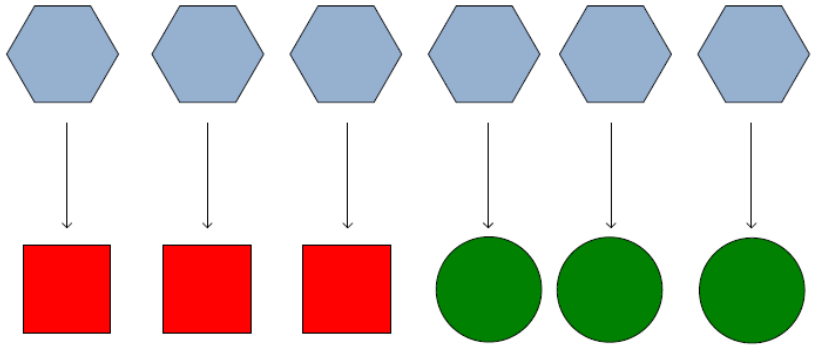
Map(k1,v1) → list(k2,v2)
Reduce(k2, list(v2)) → list(v2)



Distributed Wordcount

Map

- Input records from the data source
  - lines out of files, rows of a database, etc.
- Passed to map function as key-value pairs
  - Line number, line value
- map() produces *zero or more intermediate values*, each associated with an output key



Map

- **Example Wordcount**

```
map(String input_key, String input_value):
  // input_key: line number
  // input_value: line of text
  for each Word w in input_value.tokenize()
    EmitIntermediate(w, "1");
```

(0, “How now brown cow”) →
[(“How”, 1), (“now”, 1), (“brown”, 1), (“cow”, 1)]

- **Example: Upper-case Mapper**

```
map(k, v) { emit(k.toUpper(), v.toUpper()); }

("foo", "bar") → ("FOO", "BAR")
("Foo", "other") → ("FOO", "OTHER")
("key2", "data") → ("KEY2", "DATA")
```

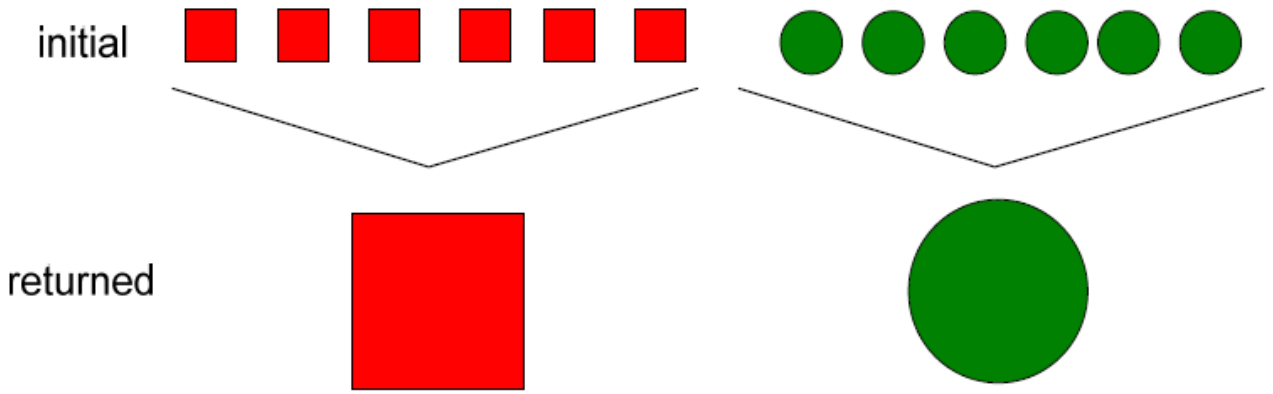
- **Example: Filter Mapper**

```
map(k, v) { if (isPrime(v)) then emit(k, v); }

("foo", 7) → ("foo", 7)
("test", 10) → ()        //nothing emitted
```

Reduce

- All the intermediate values from map for a given output key are combined together into a list
- reduce() combines these intermediate values into one or more final values for that same output key … Usually one final value per key
- One output "file" per reducer



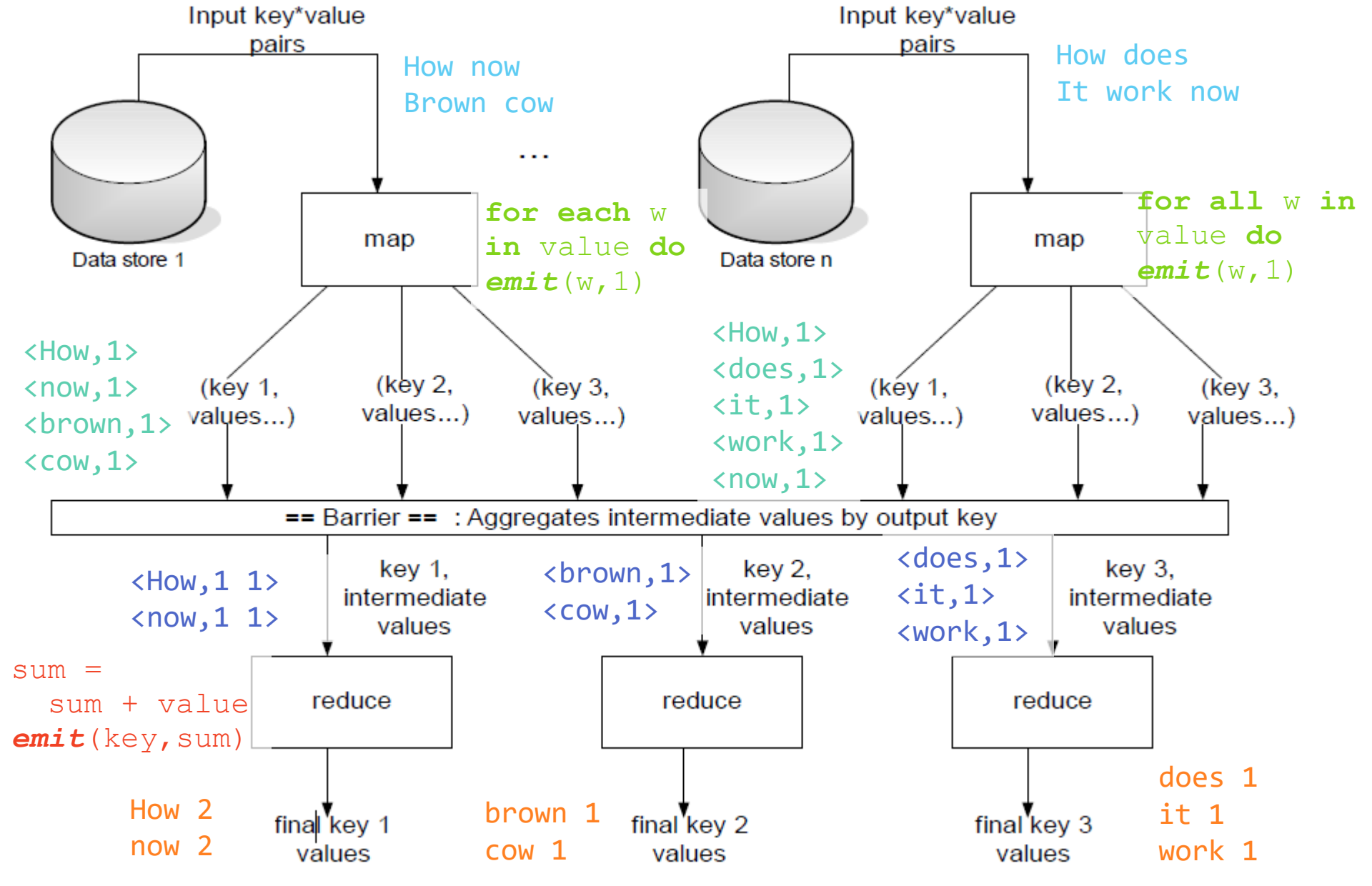
Reduce

- **Example Wordcount**

```
reduce(String output_key, Iterator intermediate_values)
  // output_key: a word
  // output_values: a list of counts
  int sum = 0;
  for each v in intermediate_values
    sum += ParseInt(v);
  Emit(output_key, AsString(sum));
```

("A", [1, 1, 1]) → ("A", 3)
("B", [1, 1]) → ("B", 2)

MapReduce: Word Count Drilldown

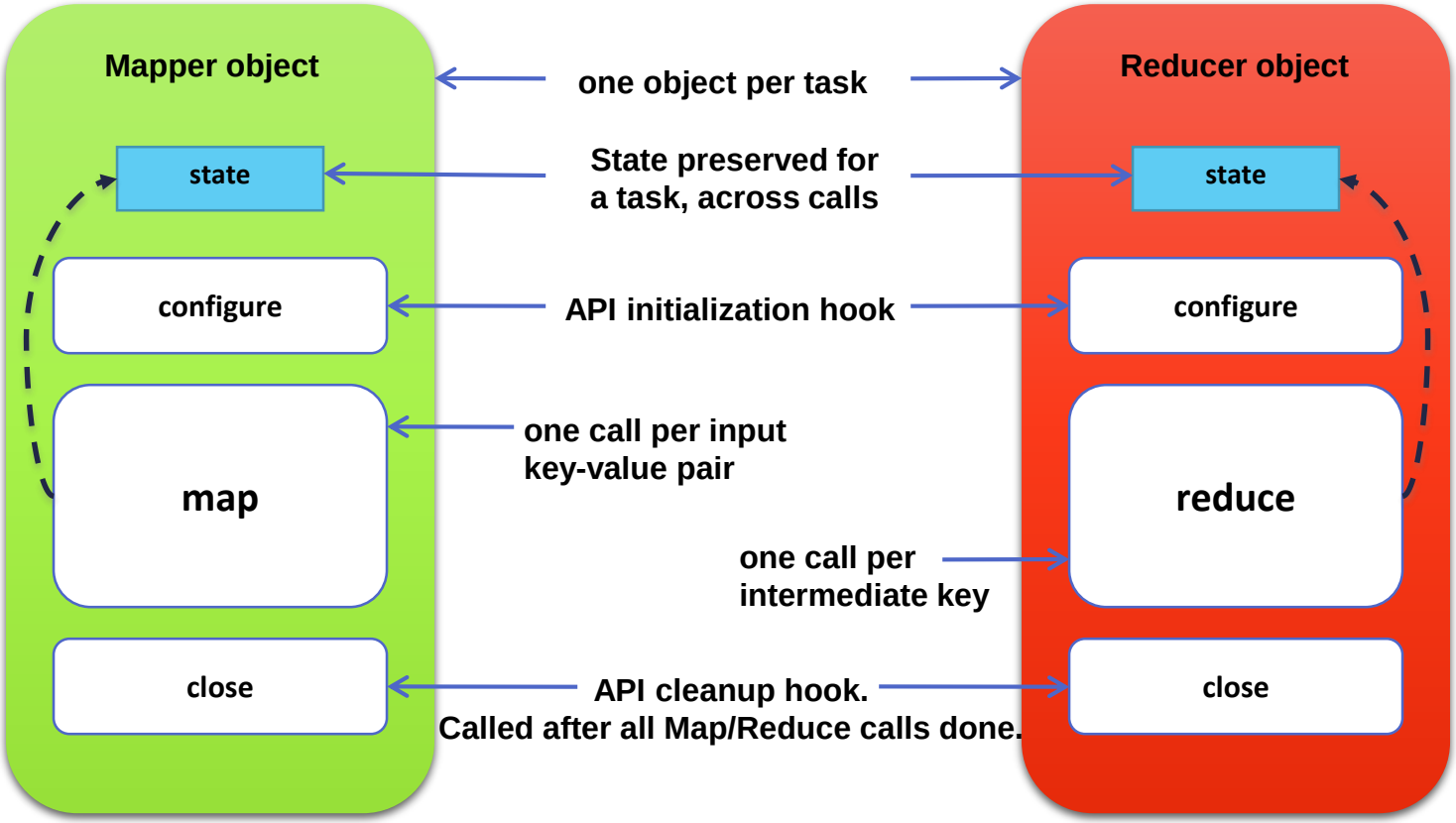


Mapper/Reducer Tasks *vs.* Map/Reduce Methods

- Number of Mapper and Reducer tasks is specified by user
- Each **Mapper/Reducer task** can make multiple calls to **Map/Reduce method**, sequentially
- Mapper and Reducer tasks may run on different machines
- Implementation framework decides
  - Placement of Mapper and Reducer tasks on machines
  - Keys assigned to mapper and reducer tasks
  - But can be controlled by user…

Maintainer State in Tasks

- Capture state & dependencies across multiple keys and values



Improve Word Count using State?

```
map(String k, String v)
  foreach w in v.tokenize()
    emit(w, "1")
```

```
mapperInit()
  H = new HashMap<String,int>()
map(String k, String v)
  foreach w in v.tokenize()
    H[w] = H[w] + 1
mapperClose()
 foreach w in H.keys()
  emit(w, H[w])
```

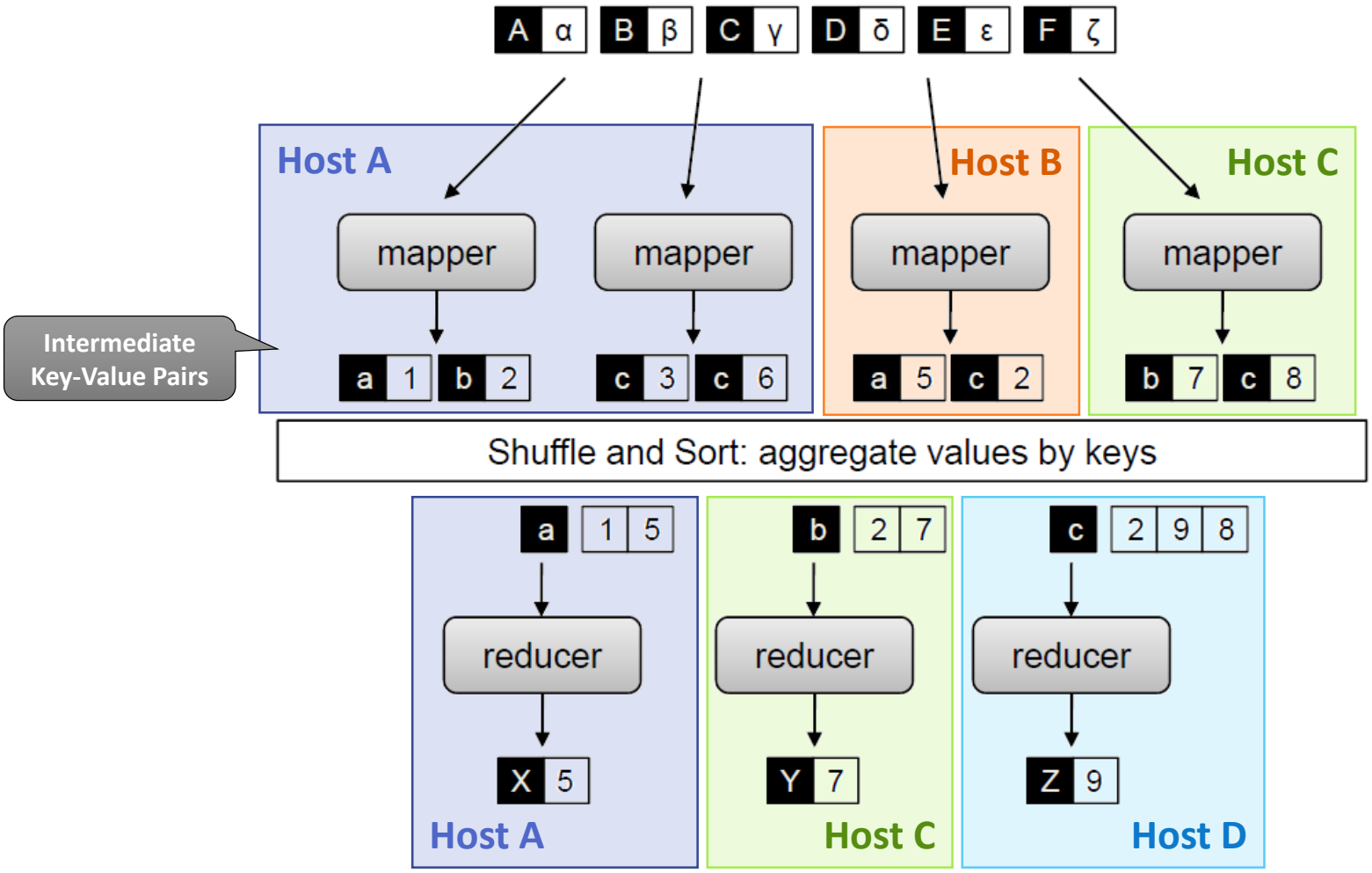
```
reduce(String k, int[] v)
  int sum = 0
  foreach n in v[]  sum += v
  emit(k, sum)
```

```
reduce(String k, int[] v)
  int sum = 0;
  foreach n in v[]  sum += v
  emit(k, sum)
```

Shuffle & Sort
The Magic happens here!

- **Shuffle** does a "group by" of keys from all mappers
  - Similar to SQL goupBy operation
- **Sort** of *local keys* to *Reducer task* performed
  - Keys arriving at each reducer are sorted
  - No sorting guarantee of keys across reducer tasks
- No ordering guarantees of values for a key
  - Implementation dependent
- Shuffle and Sort *implemented efficiently* by framework

Map-*Shuffle*-*Sort*-Reduce



Anagram Example

- *"An **anagram** is a type of word play, the result of rearranging the letters of a word or phrase to produce a new word or phrase, using all the original letters exactly once; for example **orchestra** can be rearranged into **carthorse**." … Wikipedia*
  - thickens = kitchens, reserved = reversed,
    - cheating = teaching, cause = sauce
  - Tom Marvolo Riddle = I am Lord Voldemort
- Problem: Find ALL anagrams in the English dictionary of ~1M words (10^6)
- *1M X 1M comparisons?*

Anagram Example

```
public class AnagramMapper extends MapReduceBase implements
                    Mapper<LongWritable, Text, Text, Text> {
  private Text sortedText = new Text();
  private Text orginalText = new Text();
  public void map(LongWritable key, Text value,
    OutputCollector<Text, Text> outputCollector, Reporter reporter) {
    String word = value.toString();
    char[] wordChars = word.toCharArray();
    Arrays.sort(wordChars);
    String sortedWord = new String(wordChars);
    sortedText.set(sortedWord);
    orginalText.set(word);
    // Sort word and emit <sorted word, word>
    outputCollector.collect(sortedText, orginalText);
  }
}
```

http://code.google.com/p/hadoop-map-reduce-examples/

Anagram Example…

```
public void reduce(Text anagramKey, Iterator<Text> anagramValues,
        OutputCollector<Text, Text> results, Reporter reporter) {
    String output = "";
    while(anagramValues.hasNext()) {
        Text anagram = anagramValues.next();
        output = output + anagram.toString() + "~";
    }
    StringTokenizer outputTokenizer =
        new StringTokenizer(output,"~");
    // if the values contain more than one word
    // we have spotted a anagram.
    if(outputTokenizer.countTokens()>=2) {
        output = output.replace("~", ",");
        outputKey.set(anagramKey.toString());
        outputValue.set(output);
        results.collect(outputKey, outputValue);
    }
}
```

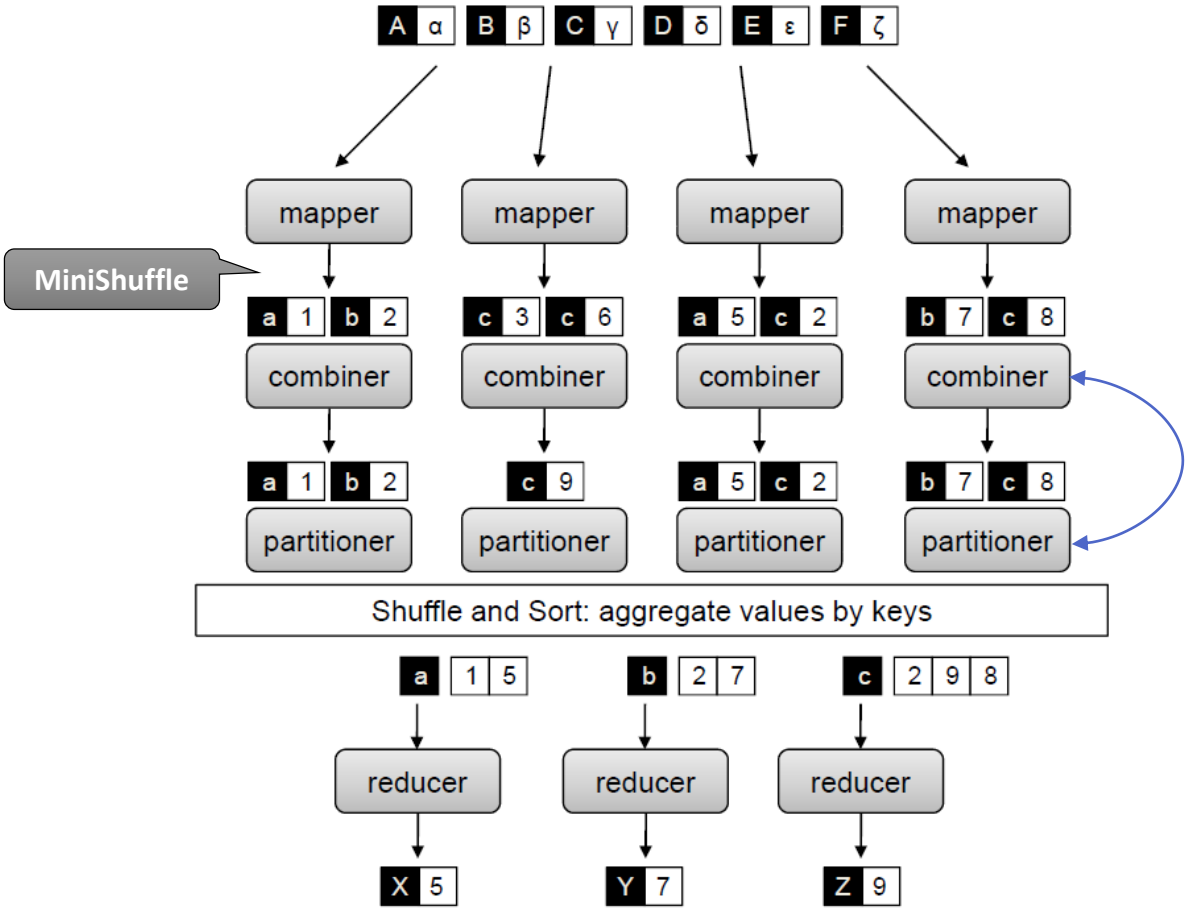
Optimization: Combiner

- Logic runs on output of Map tasks, on the map machines
  - "Mini-Reduce," only on local Map output
- Output of Combiner sent to shuffle
  - Saves bandwidth before sending data to Reducers
- Same *input* and *output types* as Map's *output* type
  - `Map(k,v) → (k',v')`
  - `Combine(k',v'[]) → (k',v')`
  - `Reduce(k',v'[]) → (k'',v'')`
- Reduce task logic can be used as combiner **if** commutative & associative. Usually for trivial ops.
- Combiner may be called 0, 1 or more times

Optimization: Partitioner

- Decides assignment of intermediate keys grouped to specific Reducer tasks
  - Affects the load on each reducer task
- Sorting of local keys for Reducer task done after partitioning
- Default is hash partitioning
  - **`HashPartitioner(key, nParts)` → `part`**
  - Number of Reducer (**nParts**) tasks known in advance
  - Returns a partition number [0, nParts)
  - Default partitioner balances number of keys per Reducer … *assuming uniform key distribution*
  - May not balance the number of values processed by a Reducer

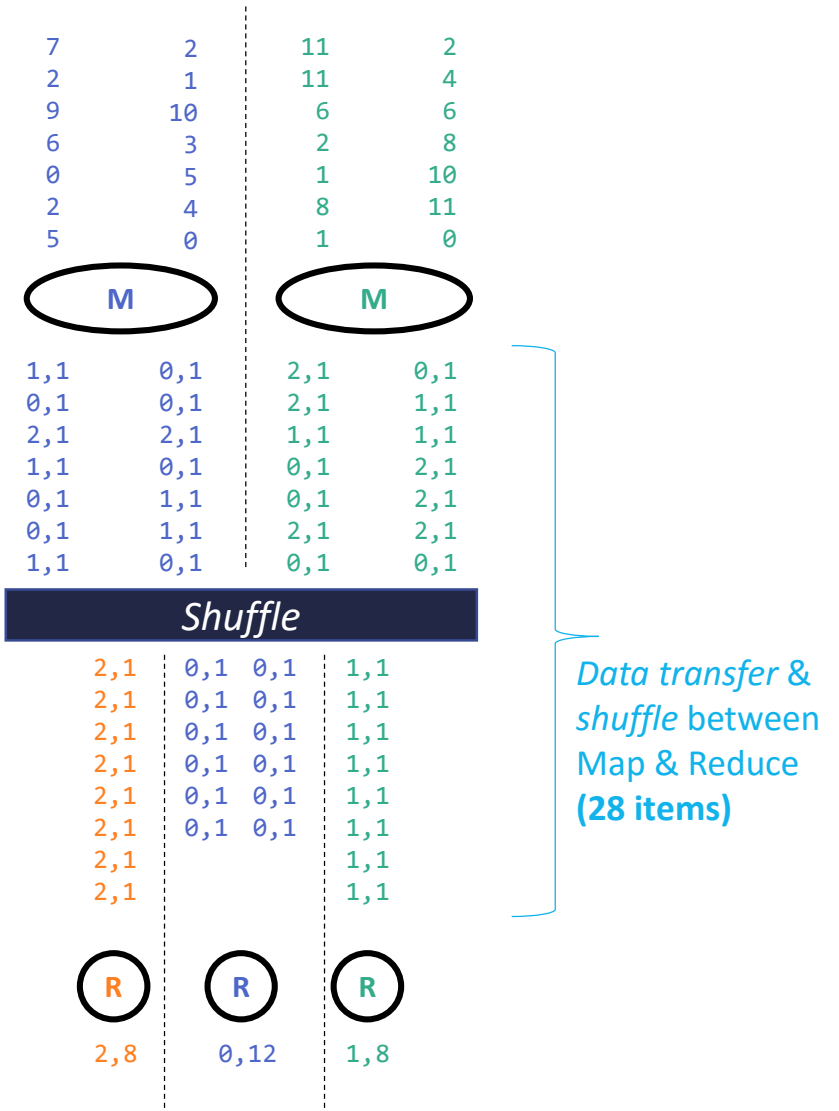
Map-*MiniShuffle*-Combine-*Partition*-Shuffle-Sort-**Reduce**



Combine & Partition phases could be interchanged, based on implementation

Data-Intensive Text Processing with MapReduce, Jimmy Lin, 2010

MapReduce for Histogram

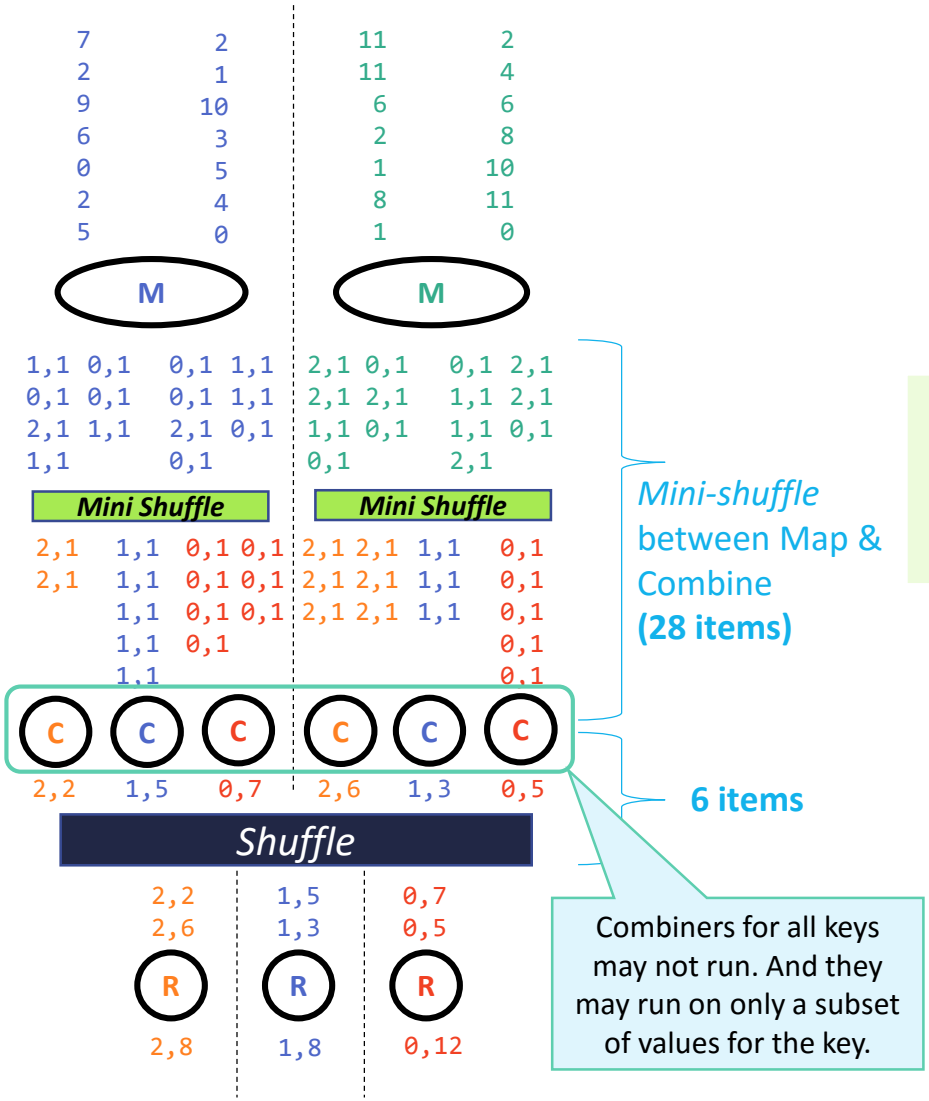



```
int bucketWidth = 4 // input

Map(k, v) {
    emit(floor(v/bucketWidth), 1)
    // <bucketID, 1>
}


// one reduce per bucketID
Reduce(k, v[]){
    sum=0;
    foreach(n in v[])  sum++;
    emit(k, sum)
    // <bucketID, frequency>
}
```

MapReduce for Histogram




```
int bucketWidth = 4 // input

Map(k, v) {
    emit(floor(v/bucketWidth), 1)
    // <bucketID, 1>
}

Combine(k, v[]){
    // same code as Reduce()
}

// one reduce per bucketID
Reduce(k, v[]){
    sum=0;
    foreach(n in v[])  sum++;
    emit(k, sum)
    // <bucketID, frequency>
}
```

Since Reducer is *commutative* and *associative*, its logic can be used as a Combiner

Combiner Advantage

- Mini-Shuffle lowers the overall cost for Shuffle
- E.g. *n* total items emitted from *m* mappers
- NW Transfer and Disk IO costs
  - In ideal case, m items vs. n items *written and read from disk*, *transferred over network* (m<<n)
- Shuffle, less of an impact
  - If more mapper tasks are present than reducers, higher parallelism for doing groupby and mapper-side partial sort.
  - Local Sort on reducer is based on number of unique keys, which does not change due to combiner.

TeraSort: Sorting Large Files

- Input is a list of positive numbers (or words)
  - Say a **terabyte** of data, 10^{11} entries
- Output is the list of numbers or words in sorted order

- *How can we use MapReduce for this?*

TeraSort: Sorting Large Files

- Approach 1
  - Have ***n*** mappers and ***1*** reducer tasks
  - **`Map(key, num) → (num, 0)`**
  - Shuffle & Local Sort: All numbers (intermediate keys) to the single reducer is sorted by framework
  - **`Reduce(num, [0]) → (num+)`**
  - Output from the reducer task is in sorted order
  - NOTE: repeat printing of num if there are duplicate '0'
  - *Do we have any scaling?*

TeraSort: Sorting Large Files

- Approach 2
  - ***n*** mapper, ***m*** reducer tasks
  - **`Map(key, num) → (num, 0)`**
  - Shuffle & Local Sort: All numbers (intermediate keys) to a single reducer are sorted
  - **`Reduce(num, [0]) → (num+)`**
  - Local output of numbers from each reducer is sorted, e.g. *m* sorted files
  - *Merge Sort separately? $O(n.k^2)$*
  - *What is the scaling?*

TeraSort: Sorting Large Files

- Approach 3
  - ***n*** mapper, ***m*** reducer tasks
  - **`Map(key, num) → (num/(MAX/m), num)`**
  - Map does a histogram distribution of *num* into reduce method buckets
  - **`Reduce(bucketID, num[]) → sort(num[])`**
  - Reduce performs a local sort of all local numbers
    - Sort managed by us, needs to fit in memory, etc.
  - Concatenate output of *m* sorted files
  - *What is the scaling?*

TeraSort: Sorting Large Files

- Approach 4
  - ***n*** mapper, ***m*** reducer tasks
  - **`Map(key, num) → (num, 0)`**
  - **`Partition(num, m) → floor(num/(MAX/m))`**
  - Partitioner causes numbers to be *range-partitioned* to each reducer
    - Range of values required, 0..MAX
    - Words (string) requires a trie for efficiency
  - Shuffle & Sort: Local range of numbers to a reducer is sorted
  - **`Reduce(num, 0) → (num)`**
  - Concatenate sorted output from each reducer
  - *What is the scaling?*

Quick Assignment

Find the Mean of a set of numbers

Map Input: ×, int e.g., <×,8>,<×,32>,<×,20>,<×,4>

Reduce Output: ×, int e.g. <×,16>

Computing the Mean: *Simple Approach*

```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, integer r)

1: class Reducer
2:     method Reduce(string t, integers [r_1, r_2, ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all integer r ∈ integers [r_1, r_2, ...] do
6:             sum ← sum + r
7:             cnt ← cnt + 1
8:         r_avg ← sum/cnt
9:         Emit(string t, integer r_avg)
```

All work performed by single Reducer!

Optimization: Can we use Reducer as Combiner?

Computing the Mean: *Using a Combiner*

```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, integer r)
```

```
1: class Combiner
2:     method Combine(string t, integers [r_1, r_2, ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all integer r ∈ integers [r_1, r_2, ...] do
6:             sum ← sum + r
7:             cnt ← cnt + 1
8:         Emit(string t, pair (sum, cnt))          ▷ Separate sum and count
```

```
1: class Reducer
2:     method Reduce(string t, pairs [(s_1, c_1), (s_2, c_2) ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s_1, c_1), (s_2, c_2) ...] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         r_avg ← sum/cnt
9:         Emit(string t, integer r_avg)
```

Is this correct?

Computing the Mean: Fixed?

```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, pair (r, 1))
```

```
1: class Combiner
2:     method Combine(string t, pairs [(s_1, c_1), (s_2, c_2) ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s_1, c_1), (s_2, c_2) ...] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         Emit(string t, pair (sum, cnt))
```

```
1: class Reducer
2:     method Reduce(string t, pairs [(s_1, c_1), (s_2, c_2) ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s_1, c_1), (s_2, c_2) ...] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         r_avg ← sum/cnt
9:         Emit(string t, pair (r_avg, cnt))
```

MapReduce: Recap

- Programmers must specify:
  map (k, v) → <k', v'>*
  reduce (k', v'[]) → <k'', v''>*
  - All values with the same key are reduced together
- Optionally, also:
  partition (k', number of partitions) → partition for k'
  - Often a simple hash of the key, e.g., hash(k') mod n
  - Divides up key space for parallel reduce operations
  combine (k', v') → <k', v'>*
  - Mini-reducers that run in memory after the map phase
  - Used as an optimization to reduce network traffic
- The execution framework handles *everything else*…

"Everything Else"

- The execution framework handles everything else…
  - Scheduling: assigns workers to map and reduce tasks
  - "Data distribution": moves processes to data
  - Synchronization: gathers, sorts, and shuffles intermediate data
  - Errors and faults: detects worker failures and restarts
- Limited control over data and execution flow
  - All algorithms must expressed in m, r, c, p
- You don't know:
  - Where mappers and reducers run
  - When a mapper or reducer begins or finishes
  - Which input a particular mapper is processing
  - Which intermediate key a particular reducer is processing

MR Algorithm Design

Map-Only Design
Filtering: Distributed Grep

- Input
  - Lines of text from HDFS
  - “Search String” (e.g. regex), input parameter to job
- Mapper
  - Search line for string/pattern
  - Output matching lines
- Reducer
  - Identity function (output = input), or none at all

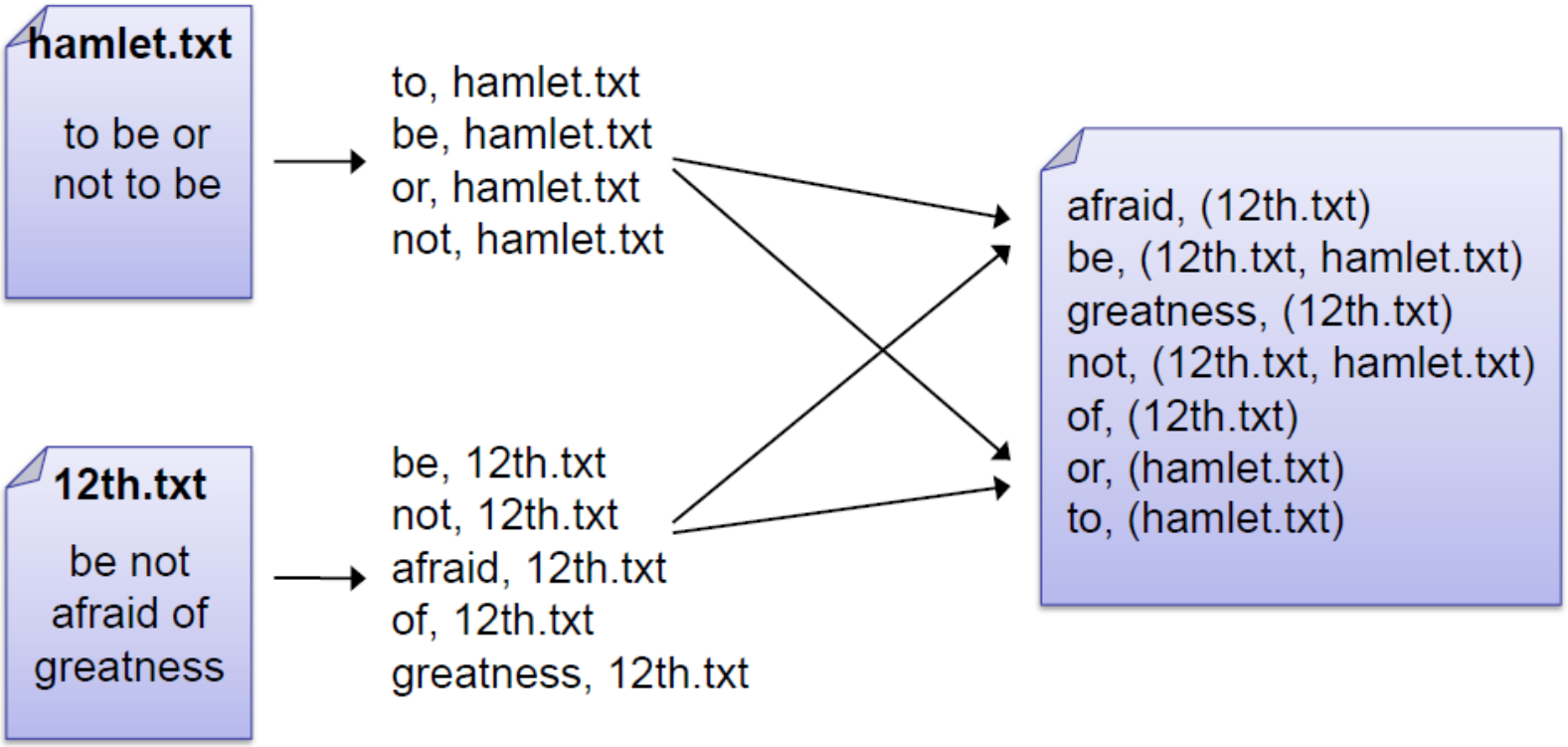
Accumulation

- List of courses with number of students enrolled in each (GoI scheme with citizens enrolled in each)
- Input
  - <StudentID, CourseID>
  - <2482, SE256> <6427, SE252> <1635, E0 259>
- Mapper
  - Emit <CourseID, 1>
  - <SE256, 1>, <SE252, 1>, <E0 259, 1>
- Partition
  - By Course ID
- Sort <E0 259, 1>, <SE252, 1>, <SE256, 1>
- Reduce <E0 259, [1,1]>, <SE252, [1]>, <SE256, [1,1,1]>
  - Count number of students per Course.
  - Output <Course ID, Count>
  - <SE256, 2>, <SE252, 1>, <E0 259, 3>

Inverted Index

- Convert from Key:Values to Value:Keys form
  - E.g. <URL, Lines> => <Word:URL[]>
  - Useful for building search index
- Input: <URL, Line>
- Map: `foreach(Word in Line) emit(Word, URL)`
- Combiner: Combine URLs for same Word
- Reduce: `emit(Word, sort(URL[ ]))`

Inverted Index Example



Join

Customers

cfirstname	clastname	cphone	cstreet	czipcode
Tom	Jewett	714-555-1212	10200 Slater	92708
Alvaro	Monge	562-333-4141	2145 Main	90840
Wayne	Dick	562-777-3030	1250 Bellflower	90840

Orders

cfirstname	clastname	cphone	orderdate	soldby
Alvaro	Monge	562-333-4141	2003-07-14	Patrick
Wayne	Dick	562-777-3030	2003-07-14	Patrick
Alvaro	Monge	562-333-4141	2003-07-18	Kathleen
Alvaro	Monge	562-333-4141	2003-07-20	Kathleen

Customers joined to Orders

cfirstname	clastname	cphone	cstreet	czipcode	orderdate	soldby
Alvaro	Monge	562-333-4141	2145 Main	90840	2003-07-14	Patrick
Wayne	Dick	562-777-3030	1250 Bellflower	90840	2003-07-14	Patrick
Alvaro	Monge	562-333-4141	2145 Main	90840	2003-07-18	Kathleen
Alvaro	Monge	562-333-4141	2145 Main	90840	2003-07-20	Kathleen

http://www.tomjewett.com/dbdesign/dbdesign.php?page=join.php

Join

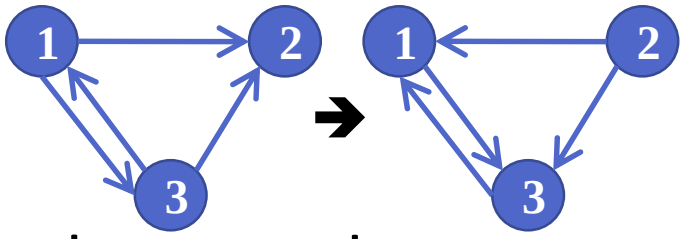
- Given two sets of files, combine the lines having the same key in each file
- Input:
  - <customer_data>, <order_data>
- Mapper:
  - emit <cell, customer_data>, <cell, order_data>
- Reduce:
  - If both keys are *unique*,
    - <cell, [customer_data, order_data]>
    - Just concatenate and emit the pair, if there are two items
    - If only one item (only customer or order value present), skip
  - If either or both keys are *not unique*,
    - <cell, [customer_data*, order_data*]>
    - Emit cross product of customer_data* and order_data* values, i.e., local join for each cell key

Reverse graph edge directions & output in node order

- Input: adjacency list of graph (e.g. 3 nodes and 4 edges)

(3, [1, 2])
(1, [2, 3]) ➔

(1, [3])
(2, [1, 3])
(3, [1])



- node_ids in the output values are also sorted. But Hadoop only sorts on keys!
- MapReduce format
  - Input: (3, [1, 2]), (1, [2, 3]).
  - Intermediate: (1, [3]), (2, [3]), (2, [1]), (3, [1]). (reverse edge direction)
  - Out: (1,[3]) (2, [1, 3]) (3, [[1]).

More Algorithms

- Numerical Integration
- PageRank
- Regression Tree
- *Try these yourselves!*

Scalable Hadoop Algorithms: Themes

- Avoid object creation
  - Inherently costly operation
  - Garbage collection
- Avoid buffering
  - Limited heap size
  - Works for small datasets, but won't scale!

Importance of Local Aggregation

- Ideal scaling characteristics:
  - Twice the data, twice the running time
  - Twice the resources, half the running time
- Why can't we achieve this?
  - Synchronization requires communication
  - Communication kills performance
- Thus… avoid communication!
  - Reduce intermediate data via local aggregation
  - Combiners can help

Design Pattern for Local Aggregation

- "In-mapper combining"
  - Fold the functionality of the combiner into the mapper by preserving state across multiple map calls
- Advantages
  - Speed
  - Why is this faster than actual combiners?
- Disadvantages
  - Explicit memory management required
  - Potential for order-dependent bugs

Combiner Design

- Combiners and reducers share same method signature
  - Sometimes, reducers can serve as combiners
  - Often, not…
- Remember: combiner are optional optimizations
  - Should not affect algorithm correctness
  - May be run 0, 1, or multiple times
- Example: find average of all integers associated with the same key

Assignment A

Due on ~~**Wed Feb 3, 2016**~~ **Sun Feb 7, 2016** by **midnight IST**

~~**10% weightage**~~ **15% weightage**

Objectives

- Defining simple analytics over large text and graph datasets, and translating them into MapReduce algos.
- Writing new MR applications & improving performance existing ones using combiners and partitioners.
- Generating large synthetic datasets for evaluation.
- Awareness of memory/CPU used by Mappers/Reducer tasks.
- Analysis of MapReduce application logs to evaluate performance and scalability.
- Learning to start assignments early on a shared cluster ☺

Simple Analytics over Text Data

- Source: **Common Crawl**, Nov 2015*
  - *Internet Archive: WayBack Machine*, www.archive.org
- Total data: 1.8 Billion pages (1,800,000,000,000), 150TB (150,000,000,000,000)
- Course data: *0.5% of WWW*
  - 750GB raw data, **180GB** compressed data in 180 files
  - `HDFS://SE256/CC`
- Web ARChive (WARC) file format
  - HTTP headers and responses
  - Content body
  - Hadoop input format readers provided (1 split per file)

*http://blog.commoncrawl.org/2015/12/november-2015-crawl-archive-now-available/

Simple Analytics over Text Data

```
[simmhan@sslcluster data]$ hfs -ls /SE256/CC
16/01/27 07:35:35 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where app
Found 36 items
-rw-r--r--   2 simmhan supergroup  919726379 2016-01-24 20:58 /SE256/CC/CC-MAIN-20151124205404-00000-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  923524350 2016-01-24 20:59 /SE256/CC/CC-MAIN-20151124205404-00001-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  919912183 2016-01-25 14:30 /SE256/CC/CC-MAIN-20151124205404-00002-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  925106185 2016-01-25 14:31 /SE256/CC/CC-MAIN-20151124205404-00003-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  911197924 2016-01-25 14:36 /SE256/CC/CC-MAIN-20151124205404-00004-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  913033680 2016-01-25 14:36 /SE256/CC/CC-MAIN-20151124205404-00005-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  917006052 2016-01-25 14:36 /SE256/CC/CC-MAIN-20151124205404-00006-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  907718559 2016-01-25 14:36 /SE256/CC/CC-MAIN-20151124205404-00007-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  911919205 2016-01-25 14:36 /SE256/CC/CC-MAIN-20151124205404-00008-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  916227193 2016-01-25 14:36 /SE256/CC/CC-MAIN-20151124205404-00009-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  902707407 2016-01-25 14:53 /SE256/CC/CC-MAIN-20151124205404-00010-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  918098236 2016-01-25 14:54 /SE256/CC/CC-MAIN-20151124205404-00011-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  911963709 2016-01-25 14:54 /SE256/CC/CC-MAIN-20151124205404-00012-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  917302853 2016-01-25 14:54 /SE256/CC/CC-MAIN-20151124205404-00013-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  906261716 2016-01-25 14:54 /SE256/CC/CC-MAIN-20151124205404-00014-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  907338115 2016-01-25 14:54 /SE256/CC/CC-MAIN-20151124205404-00015-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  922153165 2016-01-25 14:54 /SE256/CC/CC-MAIN-20151124205404-00016-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  908269738 2016-01-25 14:54 /SE256/CC/CC-MAIN-20151124205404-00017-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  902650433 2016-01-25 14:55 /SE256/CC/CC-MAIN-20151124205404-00018-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  927320721 2016-01-25 14:55 /SE256/CC/CC-MAIN-20151124205404-00019-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  900701903 2016-01-25 15:36 /SE256/CC/CC-MAIN-20151124205404-00020-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  915062896 2016-01-25 15:36 /SE256/CC/CC-MAIN-20151124205404-00021-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  916899286 2016-01-25 15:36 /SE256/CC/CC-MAIN-20151124205404-00022-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  911241910 2016-01-25 15:36 /SE256/CC/CC-MAIN-20151124205404-00023-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  919615856 2016-01-25 15:36 /SE256/CC/CC-MAIN-20151124205404-00024-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  905403359 2016-01-25 15:36 /SE256/CC/CC-MAIN-20151124205404-00025-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  915832672 2016-01-25 15:37 /SE256/CC/CC-MAIN-20151124205404-00026-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  909857671 2016-01-25 15:37 /SE256/CC/CC-MAIN-20151124205404-00027-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  916954086 2016-01-25 15:37 /SE256/CC/CC-MAIN-20151124205404-00028-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  928646442 2016-01-25 15:37 /SE256/CC/CC-MAIN-20151124205404-00029-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  910799081 2016-01-25 15:37 /SE256/CC/CC-MAIN-20151124205404-00030-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  925704947 2016-01-25 15:37 /SE256/CC/CC-MAIN-20151124205404-00031-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  922179298 2016-01-25 15:37 /SE256/CC/CC-MAIN-20151124205404-00032-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  916101863 2016-01-25 15:38 /SE256/CC/CC-MAIN-20151124205404-00033-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  917994920 2016-01-25 15:38 /SE256/CC/CC-MAIN-20151124205404-00034-ip-10-71-132-137.ec2.internal.warc.gz
-rw-r--r--   2 simmhan supergroup  921907054 2016-01-25 15:38 /SE256/CC/CC-MAIN-20151124205404-00035-ip-10-71-132-137.ec2.internal.warc.gz
[simmhan@sslcluster data]$
```


More files will be added gradually to total 180

Peek inside an uncompressed WARC file …

```
WARC/1.0
WARC-Type: metadata
WARC-Date: 2015-11-24T22:07:43Z
WARC-Record-ID: <urn:uuid:a5a74571-bd61-4ff2-bef4-7e8320840cee>
Content-Length: 20
Content-Type: application/warc-fields
WARC-Warcinfo-ID: <urn:uuid:a24fb23b-945f-4f56-81f0-a323837335e3>
WARC-Concurrent-To: <urn:uuid:9da609ec-88ec-479c-8ae3-e096acdf8ecb>
WARC-Target-URI: http://040231.hospitaltambon.com/

fetchTimeMs: 572



WARC/1.0
WARC-Type: request
WARC-Date: 2015-11-24T22:04:24Z
WARC-Record-ID: <urn:uuid:d2b177e2-96f7-4b00-9fe0-1fb8e2405417>
Content-Length: 278
Content-Type: application/http; msgtype=request
WARC-Warcinfo-ID: <urn:uuid:a24fb23b-945f-4f56-81f0-a323837335e3>
WARC-IP-Address: 173.194.205.132
WARC-Target-URI: http://10000thingsthatmakemehappy.blogspot.com/

GET / HTTP/1.0
Host: 10000thingsthatmakemehappy.blogspot.com
Accept-Encoding: x-gzip, gzip, deflate
User-Agent: CCBot/2.0 (http://commoncrawl.org/faq/)
Accept-Language: en-us,en-gb,en;q=0.7,*;q=0.3
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8



WARC/1.0
WARC-Type: response
WARC-Date: 2015-11-24T22:04:24Z
WARC-Record-ID: <urn:uuid:770febae-9cfb-4752-8d1b-2fda3c110904>
Content-Length: 134388
Content-Type: application/http; msgtype=response
WARC-Warcinfo-ID: <urn:uuid:a24fb23b-945f-4f56-81f0-a323837335e3>
WARC-Concurrent-To: <urn:uuid:d2b177e2-96f7-4b00-9fe0-1fb8e2405417>
WARC-IP-Address: 173.194.205.132
WARC-Target-URI: http://10000thingsthatmakemehappy.blogspot.com/
WARC-Payload-Digest: sha1:S4FKLMYY7KGPD43ZCAMWHHA2ORQPYR4B
WARC-Block-Digest: sha1:B6H4ETNWZADF36TYIOUN62IAGUCKASSJ

HTTP/1.0 200 OK
Content-Type: text/html; charset=UTF-8
Expires: Tue, 24 Nov 2015 22:04:24 GMT
Date: Tue, 24 Nov 2015 22:04:24 GMT
Cache-Control: private, max-age=0
Last-Modified: Mon, 23 Nov 2015 02:23:17 GMT
ETag: "c9570a59-8f6c-4678-8e06-6ce8f68ec888"
Content-Encoding: gzip
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Content-Length: 21947
Server: GSE

<!DOCTYPE html>
<html class='v2' dir='ltr'>
```

HTTP Request/ Response/Metadata

```
<!DOCTYPE html>
<html class='v2' dir='ltr'>
<head>
<meta content='width=1100' name='viewport'/>
<meta content='text/html; charset=UTF-8' http-equiv='Content-Type'/>
<script type="text/javascript">(function() { (function(){function c(a){this.t={};this.tick=function(a,c,b){var d=void
 0!=b?b:(new Date).getTime();this.t[a]=[d,c];if(void 0==b)try{window.console.timeStamp("CSI/"+a)}catch(e){}};this.tic
k("start",null,a)}var a;window.performance&&(a=window.performance.timing);var h=a?new c(a.responseStart):new c;window
.jstiming={Timer:c,load:h};if(a){var b=a.navigationStart,e=a.responseStart;0<b&&e>=b&&(window.jstiming.srt=e-b)}if(a)
{var d=window.jstiming.load;0<b&&e>=b&&(d.tick("_wtsrt",void 0,b),d.tick("wtsrt_",
"_wtsrt",e),d.tick("tbsd_","wtsrt_"))}try{a=null,window.chrome&&window.chrome.csi&&(a=Math.floor(window.chrome.csi().
pageT),d&&0<b&&(d.tick("_tbnd",void 0,window.chrome.csi().startE),d.tick("tbnd_","_tbnd",b))),null==a&&window.gtbExte
rnal&&(a=window.gtbExternal.pageT()),null==a&&window.external&&(a=window.external.pageT,d&&0<b&&(d.tick("_tbnd",void
0,window.external.startE),d.tick("tbnd_","_tbnd",b))),a&&(window.jstiming.pt=a)}catch(k){}})();window.tickAboveFold=f
unction(c){var a=0;if(c.offsetParent){do a+=c.offsetTop;while(c=c.offsetParent)}c=a;750>=c&&window.jstiming.load.tick
("aft")};var f=!1;function g(){f||(f=!0,window.jstiming.load.tick("firstScrollTime"))}window.addEventListener?window.
addEventListener("scroll",g,!1):window.attachEvent("onscroll",g);
 })();</script>
<meta content='blogger' name='generator'/>
<link href='http://10000thingsthatmakemehappy.blogspot.com/favicon.ico' rel='icon' type='image/x-icon'/>
<link href='http://10000thingsthatmakemehappy.blogspot.com/' rel='canonical'/>
<link rel="alternate" type="application/atom+xml" title="10,000 things that make me happy - Atom" href="http://10000t
hingsthatmakemehappy.blogspot.com/feeds/posts/default" />
<link rel="alternate" type="application/rss+xml" title="10,000 things that make me happy - RSS" href="http://10000thi
ngsthatmakemehappy.blogspot.com/feeds/posts/default?alt=rss" />
<link rel="service.post" type="application/atom+xml" title="10,000 things that make me happy - Atom" href="https://ww
w.blogger.com/feeds/7712223741997599193/posts/default" />
<link rel="me" href="https://www.blogger.com/profile/13784768297915294889" />
<link rel="openid.server" href="https://www.blogger.com/openid-server.g" />
<link rel="openid.delegate" href="http://10000thingsthatmakemehappy.blogspot.com/" />
<!--[if IE]><script type="text/javascript" src="https://www.blogger.com/static/v1/jsbin/1494333581-ieretrofit.js"></s
cript>
<![endif]-->
<!--[if IE]> <script> (function() { var html5 = ("abbr,article,aside,audio,canvas,datalist,details," + "figure,footer
,header,hgroup,mark,menu,meter,nav,output," + "progress,section,time,video").split(','); for (var i = 0; i < html5.le
ngth; i++) { document.createElement(html5[i]); } try { document.execCommand('BackgroundImageCache', false, true); } c
atch(e) {} })(); </script> <![endif]-->
<title>10,000 things that make me happy</title>
<link type='text/css' rel='stylesheet' href='https://www.blogger.com/static/v1/widgets/3726630547-css_bundle_v2.css'
/>
<link type='text/css' rel='stylesheet' href='https://www.blogger.com/dyn-css/authorization.css?targetBlogID=771222374
1997599193&zx=c9570a59-8f6c-4678-8e06-6ce8f68ec888' />
<style id='page-skin-1' type='text/css'><!--
/*
-----------------------------------------------
Blogger Template Style
Name:     Simple
Designer: Josh Peterson
URL:      www.noaesthetic.com
----------------------------------------------- */
/* Variable definitions
====================
<Variable name="keycolor" description="Main Color" type="color" default="#66bbdd"/>
<Group description="Page Text" selector="body">
<Variable name="body.font" description="Font" type="font"
default="normal normal 12px Arial, Tahoma, Helvetica, FreeSans, sans-serif"/>
<Variable name="body.text.color" description="Text Color" type="color" default="#222222"/>
</Group>
<Group description="Backgrounds" selector
<Variable name="body.background.color" de                    ="color" default="#66bbdd"/>
<Variable name="content.background.color"                    pe="color" default="#ffffff"/>
```

Content Body

Simple Analytics over Text Data

- Sample program provided
  - Uses `WARCFileInputFormat` to read the files and return `ArchiveReader`
  - Extracts HTTP header's *content-length* attribute – *Summation of size of all content*
  - Checks if the *content-type* is HTML, and if so, extracts the *content of the body* – *Summation of size of HTML content*
- Demo…

Simple Analytics over Text Data

1. **Country's Digital Footprint & Correlation**

MR job to return the *number of unique webpages* in which each country is mentioned, and *total number of times* a country is mentioned.

Is there correlation between the *GDP* and *mentions*?

2. **Partnerships & Feuds between Countries**

MR Job to find *Top 20 pairs of countries* frequently mentioned together in the same webpage.

Is there a reason why?

Simple Analytics over Text Data

3. Popularity Contests

Define a problem to find which of *two entities are popular*, e.g., "Martian" vs. "The Big Short", "Swacch Bharath" vs. "Jan Dhan". Write an MR job to find it.

Is the answer as expected? Can this help *predict future*? (Oscars!)

4. Pre-processing & Transforming Data

MR job to generate the *graph of this crawl dataset* as an adjacency list (SourceURL→LinkedURL1, LinkedURL2, …)

Can you *run the default MR PageRank algorithm sample* on this? What does it show?

Simple Analytics over Text Data

5. Analyze Weak Scaling

Experiments on one of the above to analyse weak scaling. Plots and analysis.

Simple Analytics over Graph Data

- Source: **Twitter**, July 2009*
  - *What is Twitter, a Social Network or a News Media?, Haewoon Kwak, et al, WWW, 2010*
- Total data:
  - *41.7 million users, 1.47 billion social relations,*
  - *26GB raw size*
  - `HDFS://SE256/TWITTER`
- Data Format
  - Directed graph as an edge list
  - `UserID \t FollowerID \n`

*http://an.kaist.ac.kr/traces/WWW2010.html

Simple Analytics over Graph Data

```
[simmhan@sslcluster ~]$ hfs -ls /SE256/TWITTER
16/01/27 07:56:54 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin
-java classes where applicable
Found 1 items
-rw-r--r--   2 se256 supergroup 26172280241 2016-01-23 00:12 /SE256/TWITTER/twitter_rv.net
[simmhan@sslcluster ~]$ hfs -tail /SE256/TWITTER/twitter_rv.net
16/01/27 07:57:04 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin
-java classes where applicable
577587  61147436
61577588      61147436
61577595      61147436
61577601      61147436
61577610      61147436
61577611      61147436
61577614      61147436
61577616      61147436
61577617      61147436
61577618      61147436
61577625      61147436
61577640      61147436
61577648      61147436
61577649      61147436
61577656      61147436
61577666      61147436
61577683      61147436
61577684      61147436
61577693      61147436
61577697      60246206
61577697      61147436
61577714      61147436
61577737      61147436
61577741      61147436
61577743      61147436
61577744      61147436
```

Simple Analytics over Graph Data

- Sample program provided
  - Counts the number of *unique* vertices and edges
  - Uses partitioner to sends vertices to one reducer, edges to another
  - *Inefficient…takes ~2hrs. Do not run on original data without optimization. Smaller test graph will be provided.*
- ~~Demo…~~

Simple Analytics over Graph Data

1. **Improving efficiency of MR**

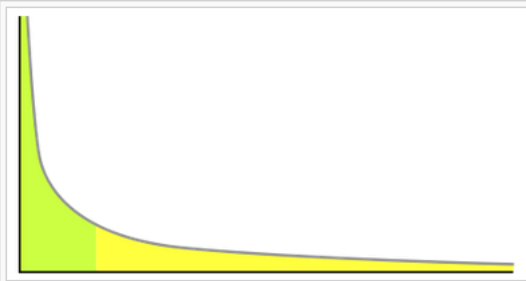
Write a *combiner* such that the MR job is much faster ~10mins.

2. **PowerLaw Distribution of Graph**

MR Job to test if Twitter follows a *Powerlaw distribution* for the number *incoming* and the number of *outgoing edges*.

Plot the frequency distribution graphs.

Simple Analytics over Graph Data



An example power-law graph, being used to demonstrate ranking of popularity. To the right is the long tail, and to the left are the few that dominate (also known as the 80–20 rule).

$$y = ax^k + \varepsilon.$$

en.wikipedia.org/wiki/Power_law

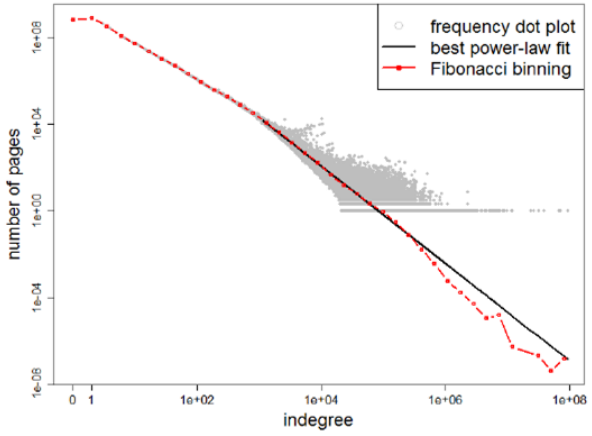


Figure 1: Frequency plot of the indegree distribution

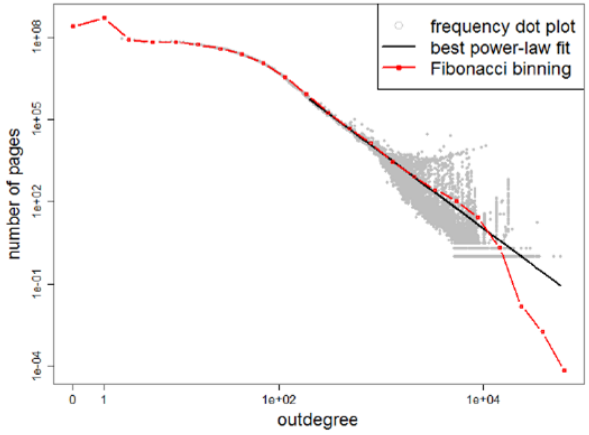


Figure 2: Frequency plot of the outdegree distribution

Graph Structure in the Web — Revisited or A Trick of the Heavy Tail, Robert Meusel, et al, ACM WWW, 2014

Simple Analytics over Graph Data

3. Recommender Service (Extra Credit)

Write one or more MR job(s) that recommends people to follow for a *given user* U_A.

Say the given user U_A follows users $\{U_1, U_2, \ldots, U_i\}$.

Identify other users $\{U_B, U_C, U_D, \ldots\}$ who follow the same set of users $\{U_1, U_2, \ldots, U_i\}$ as U_A, plus additional ones, $\{U_k, \ldots, U_m\}$.

From these additional followees $\{U_k, \ldots, U_m\}$, find the ones who are most frequently followed by $\{U_B, U_C, U_D, \ldots\}$ and recommend them to user U_A to follow.

WTF: The Who to Follow Service at Twitter, Pankaj Gupta, et al, ACM WWW, 2013

Generating and Sorting Datasets

1. **Generating Aadhaar numbers for Benchmarks**

MR Job to generate *12 digit positive random long numbers* in the range of 100,000,000,000 to 999,999,999,999.

Given *probability distribution* for 90 equi-width intervals, e.g. $(10\times10^{10},11\times10^{10}]$: 0.012211111

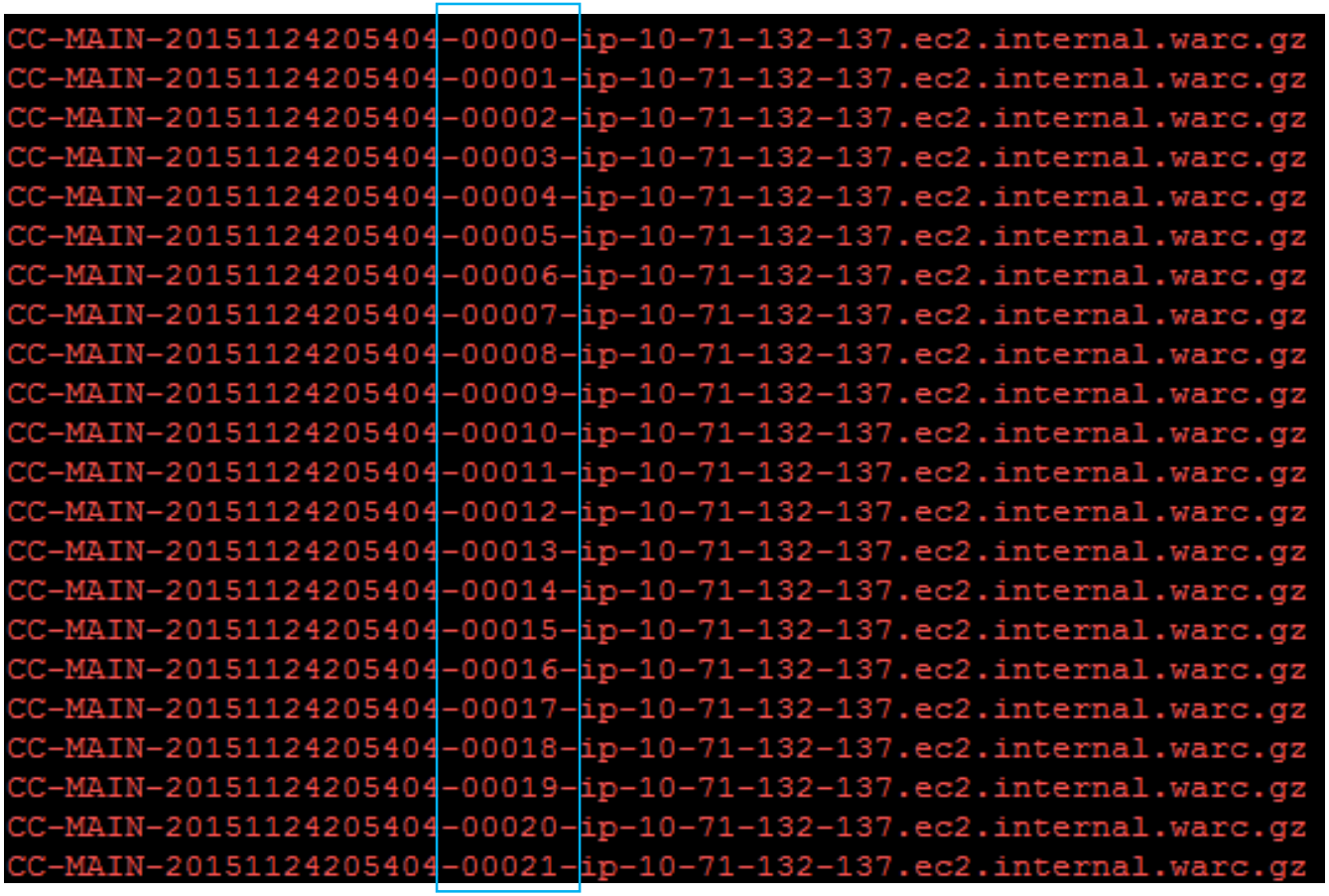
MR job to *validate and print* the actual distribution.

2. **Sorting Large Numbers**

MR application to *sort these numbers* such that the *load on each reducer task is balanced*.

3. **Analyze Weak Scaling**

Experiments on the sort MR job to analyse weak scaling. Plots and analysis.



- Try small inputs first before going to larger ones
- Use wildcards to use a subset of inputs
  - 1 file: `*-00001*.gz` 10 files: `*-0002*.gz` 100 files: `*-000*.gz`

Pravega Data Science

http://www.iisc.dsschack.com/

Assignment Preparation (by 20/Jan)

- *Wordcount*, *Distributed Grep*, *Random Int64 generator* on Course Cluster
- Monitoring, Logging and Performance measurement
- How long does *grep* and *sort* Linux commands take?
  - 1MB, 10MB, 100MB, 1GB integer files

Assignment A posted on Jan 25, due Feb 10

Reading

- Textbook (Leskovec) Chapters 1.3, 2.1-2.3, 2.5-2.6
- Lin & Dryer, Chapters 2, 3

Additional Resources

- Hadoop, HDFS & YARN
  - Data-Intensive Text Processing with MapReduce, Jimmy Lin and Chris Dyer, 2010
  - Hadoop: The Definitive Guide, **4th Edition**, 2015
  - Apache Hadoop YARN: Moving Beyond MapReduce and Batch Processing with Apache Hadoop, 2015