Introduction to Map-Reduce

Do The Math

Chicago, IL
Bangalore, India
www.mu-sigma.com

January 25, 2012

Proprietary Information

"This document and its attachments are confidential. Any unauthorized copying, disclosure or distribution of the material is strictly forbidden"

Knowledge sessions schedule

Date	Session
Wed Jan 18	Introduction to Hadoop
Wed Jan 25	Introduction to Map-Reduce
Fri Jan 27	Introduction to RHadoop
Mon Jan 30	Introduction to Java and Mahout
Wed Feb 1	Introduction to Hive
Fri Feb 3	Introduction to Pig
Mon Feb 6	Lab session

* Sessions will include hands on exercises on need basis

Agenda

- Introduction to Distributed computing
- Example: Word count using map-reduce
- Key-value pairs – The core of map-reduce
- The map-reduce paradigm – The step-by-step approach
- The map phase – Extreme parallelism
- Shuffling, Sorting – The invisible middle operations
- The reduce phase – Bringing it all together
- Examples and exercises

Distributed computation using multiple machines has been around since the 60s

- The concept of distributed computing arose in the 1960s as an offshoot of studies into operating system architecture
- The power of this method of computation derives from the virtues of parallelization – think of the incremental advantage offered by every additional driving lane added to a busy road
- A powerful idea ahead of it's time, it took an explosion in data for it to be revisited seriously; as data volumes grew, processing large data in a sequential manner became infeasible within any appreciable amount of time
- Example 1: Consider a very simple use-case: summing numbers (Reference: Slide 25)
  - Assume it takes k seconds to add two numbers
  - From large historic dataset containing a trillion rows and "number of store visits" as one of it's column, if we wished to compute total store visits to date, that's $k \times 10^{12}$ seconds to sum the entire "number of store visits" column
  - But if we were to spread this data and this computation across 1000 machines, each summing across (only) a billion rows of data, the time required to compute the sum over our column would look more like $k \times 10^{9}$ seconds
  - A direct drop in computation time of **three orders of magnitude**

Overview of Map-Reduce in Hadoop

- Hadoop is an open-source programming framework for performing massively parallel processing
- It implements the map-reduce programming model – inspired from the map and fold programming concepts in functional programming, map-reduce is a functional programming framework for parallelized computation on large distributed data
- The model provides the user with an abstraction over parallel computing with the majority of computation being performed simply by defining the **map** and **reduce** functions
- The user is buffered from implementation details like assigning data and computations to nodes, failover implementations and other internal management tasks
- Ironically enough, the Hadoop implementation of map-reduce is in Java, a decidedly un-functional programming language
- **Map-reduce programs can be written and used in Hadoop in languages apart from Java – R, Perl, Python, Ruby, PHP are few examples**

Agenda

- Introduction to Distributed computing
- Example: Word count using map-reduce
- Key-value pairs – The core of map-reduce
- The map-reduce paradigm – The step-by-step approach
- The map phase – Extreme parallelism
- Shuffling, Sorting – The invisible middle operations
- The reduce phase – Bringing it all together
- Examples and exercises

Example 2: Word count using map-reduce

	Input	Output
map	<K1,V1>	<K2,V2>
reduce	<K2, List(V2)>	<K3,V3>

The over all map-reduce word-count process

Input	Read (Key)	Read (Value)	Map (Key Value)	Combine (Key Value)	Sort and Shuffle (Key Value)	Reduce (Key Value)	Final output (Key Value)
Bear Deer Bear Car Car River Car Bear River Car River Deer Deer River Car Deer	0x0001b	Bear Deer Bear Car	Bear , 1 Deer , 1 Bear , 1 Car , 1	Bear , 2 Deer , 1 Car , 1	Bear , 2 Bear , 1	Bear , 3	Bear , 3 Deer , 2 Car , 4 River , 3
	0x0001b	Car River Car Bear	Car , 1 River , 1 Car , 1 Bear , 1	Car , 2 River , 1 Bear , 1	Deer , 1 Deer , 1	Car , 4	
	0x0001b	River Car River Deer	River , 1 Car , 1 River , 1 Deer , 1	River , 2 Car , 1 Deer , 1	Car , 1 Car , 2 Car , 1	Deer , 2	
					River , 1 River , 2	River , 3	

1. By default, the line address of every line is they key and value is the contents of the line. These Key-Value pairs will be the input to the map function

2. Each Mapper will generate new Key-Value pairs based on the map function, In this case new key is the word and value is 1.

3. Key-Value pairs will be merged on the same key before sending it to the reducer. In this case, since word is the key, so all the key-value pairs associated with same word are merged

4. Finally, the reducer function run on the values associated to same key, and produces the result

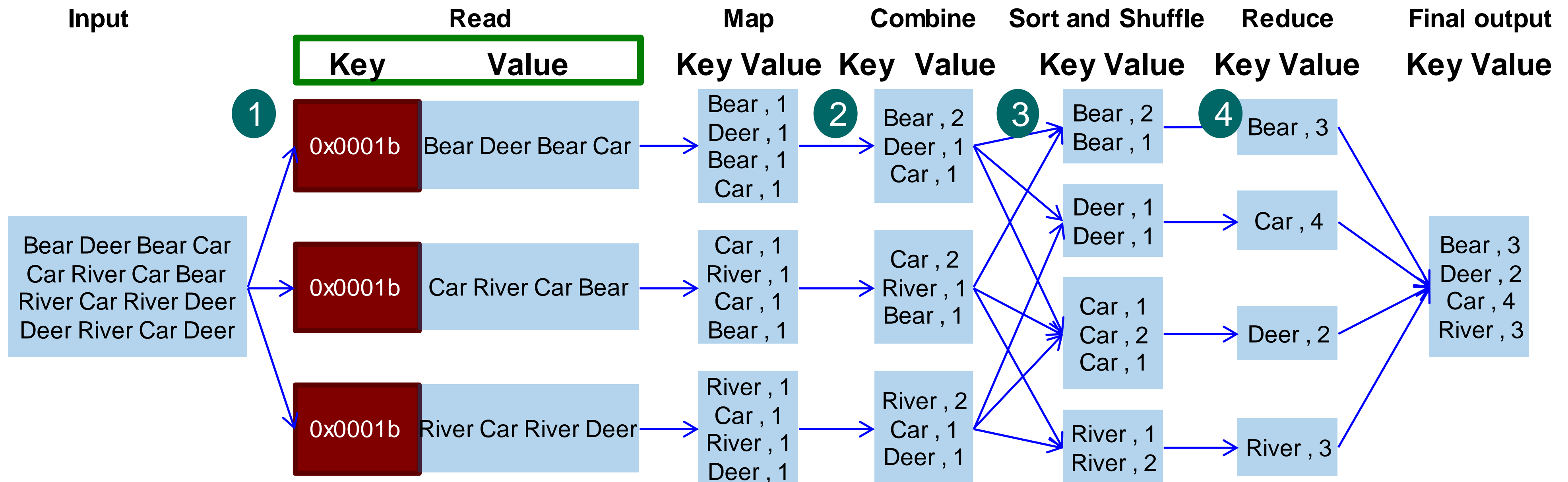
Agenda

- Introduction to Distributed computing
- Example: Word count using map-reduce
- Key-value pairs – The core of map-reduce
- The map-reduce paradigm – The step-by-step approach
- The map phase – Extreme parallelism
- Shuffling, Sorting – The invisible middle operations
- The reduce phase – Bringing it all together
- Examples and exercises

Example 2 continued: Key-value pairs

	Input	Output
map	<K1,V1>	<K2,V2>
reduce	<K2, List(V2)>	<K3,V3>

The over all map-reduce word-count process



1. By default, the line address of every line is they key and value is the contents of the line. These Key-Value pairs will be the input to the map function

2. Each Mapper will generate new Key-Value pairs based on the map function, In this case new key is the word and value is 1.

3. Key-Value pairs will be merged on the same key before sending it to the reducer. In this case, since word is the key, so all the key-value pairs associated with same word are merged

4. Finally, the reducer function run on the values associated to same key, and produces the result

Notes: Key-value pairs are the core of the Map-Reduce paradigm

- The idea behind key-value pairs has been around for about as long as the concept of distributed computing
- Just as the primary key in any normalized table is used to uniquely identify its associated row, the key in a key value pair is used to uniquely identify its associated value
- Within the context of Hadoop and map-reduce, however, keys needn't be unique; duplication of keys is in fact a “key” factor in the versatility of the map-reduce framework
- In the Hadoop implementation of map-reduce, the data types of both key and value must be serializable by the framework (serialization is the process of converting a data structure into a storable format)

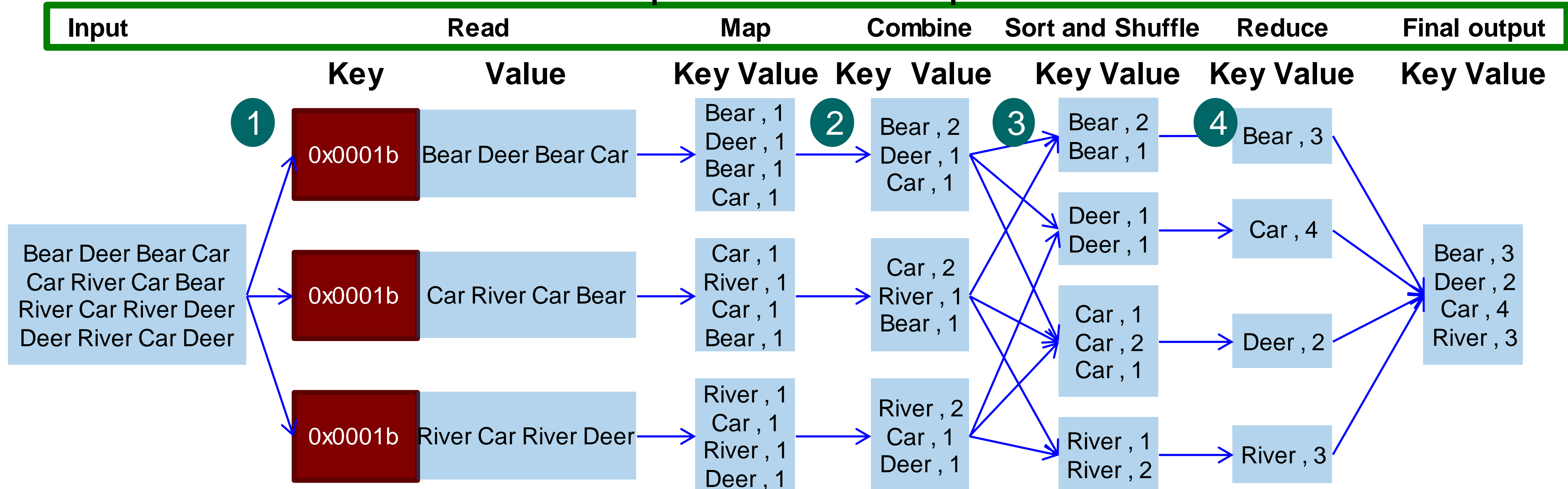
Mu Sigma Confidential

Agenda

- Introduction to Distributed computing
- Example: Word count using map-reduce
- Key-value pairs – The core of map-reduce
- The map-reduce paradigm – The step-by-step approach
- The map phase – Extreme parallelism
- Shuffling, Sorting – The invisible middle operations
- The reduce phase – Bringing it all together
- Examples and exercises

Example 2 continued: Map-reduce workflow

	Input	Output
map	<K1,V1>	<K2,V2>
reduce	<K2, List(V2)>	<K3,V3>



1. By default, the line address of every line is they key and value is the contents of the line. These Key-Value pairs will be the input to the map function

2. Each Mapper will generate new Key-Value pairs based on the map function, In this case new key is the word and value is 1.

3. Key-Value pairs will be merged on the same key before sending it to the reducer. In this case, since word is the key, so all the key-value pairs associated with same word are merged

4. Finally, the reducer function run on the values associated to same key, and produces the result

Notes: The map-reduce workflow

- **The map-reduce (MR) workflow consists of the eight stages of read, map, partition, combine, shuffle, compare, reduce and write**
- In the reading phase, data is read from disk and converted into key-value pairs $(k_{m,i}, v_{m,i})$, so for input data in a tabular format, each record could be converted into a key-value pair
- Mapping logic is applied to each input key-value pair; it takes as input a key-value pair $(k_{m,i}, v_{m,i})$; it outputs zero, one or more key-value pairs $(k_{m,o}, v_{m,o})$
- Partitioning assigns mapper output key-value pairs to reducers – the default partitioner provided by Hadoop implements a hash function for load balancing
- Combining, if used, acts as “local reducing”; it takes as input a pair of key and list of values $(k_{m,o}, list(v_{m,o}))$; it outputs zero, one or more key-value pairs $(k_{c,o}, v_{c,o})$
- In shuffling and sorting all mapper/combiner output key-value pairs and transmitted to the reducers designated by the partitioner and sorted on the basis of the input key
- Reducing logic is applied to each input key; it takes as input mapping output keys and lists of all their associated values $(k_{m,o}, list(v_{m,o}))$; it outputs zero, one or more key-value pairs $(k_{r,o}, v_{r,o})$ which are finally written back to disk

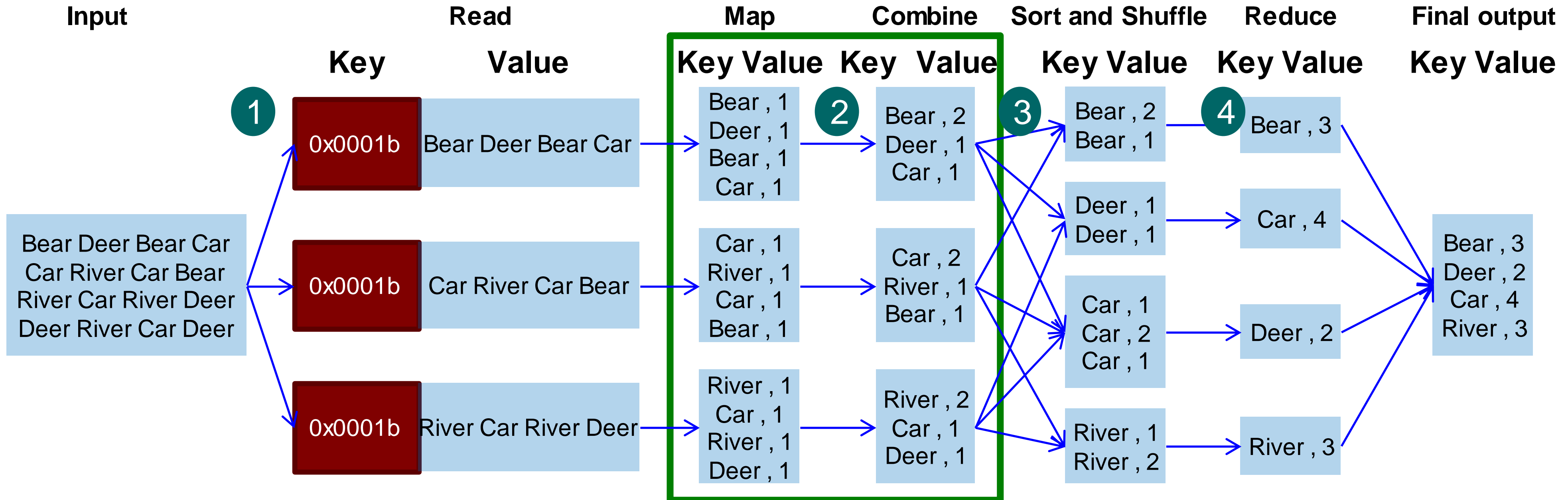
Agenda

- Introduction to Distributed computing
- Example: Word count using map-reduce
- Key-value pairs – The core of map-reduce
- The map-reduce paradigm – The step-by-step approach
- The map phase – Extreme parallelism
- Shuffling, Sorting – The invisible middle operations
- The reduce phase – Bringing it all together
- Examples and exercises

Example 2 continued: Mapping & combining

	Input	Output
map	<K1,V1>	<K2,V2>
reduce	<K2, List(V2)>	<K3,V3>

The over all map-reduce word-count process



1. By default, the line address of every line is they key and value is the contents of the line. These Key-Value pairs will be the input to the map function

2. Each Mapper will generate new Key-Value pairs based on the map function, In this case new key is the word and value is 1.

3. Key-Value pairs will be merged on the same key before sending it to the reducer. In this case, since word is the key, so all the key-value pairs associated with same word are merged

4. Finally, the reducer function run on the values associated to same key, and produces the result

Notes: Achieving efficiency with mapping and combining

- Many mappers run **independently** of each other on each machine in **parallel**
- Each mapper reads in **one** key-value pair and writes out **zero, one or more** key-value pairs; in **most** cases, the value of the mapper input key is irrelevant since it doesn't get used
- Mappers **run on the same machines** as the input data – each mapper explicitly runs on local data, Hadoop has no provision for mappers running in parallel to communicate with each other
- Output from mappers get collected in an in-memory buffer where all values corresponding to the same key are collected into lists; each time this buffer is close to full, the sorted output is written to disk OR passed on to a combiner, freeing up the buffer again
- A lot of data transformation tasks can be achieved using only mappers in conjunction with **identity reducers**, reducers that simply pass on the key-value pairs that they receive as input such that the output is another dataset consisting of some transformed form of the rows from the input dataset

Notes: Achieving efficiency with mapping and combining continued..

- In cases where both mappers and reducers are required, a combiner can be used to locally reduce the data before being passed on to the reducers,
- This can potentially **significantly reduce** the amount of data transmitted in the shuffling phase
- However, since it is called every time the mapper output memory buffer is close to full, it is **not always guaranteed** that the combiner, even if specified, will be used, since the buffer may never fill up sufficiently
- In cases when the reducing logic is both commutative $(a + b = b + a)$ and associative $((a + b) + c = a + (b + c))$, the combiner used can be the same as the reducer
- However consider the simple example where the reducer sums over the list of all input values and **adds a constant value** to this sum
- A reducer implementing such logic will not be usable as a combiner

Agenda

- Introduction to Distributed computing
- Example: Word count using map-reduce
- Key-value pairs – The core of map-reduce
- The map-reduce paradigm – The step-by-step approach
- The map phase – Extreme parallelism
- Shuffling, sorting – The invisible middle operations
- The reduce phase – Bringing it all together
- Examples and exercises

Example 2 continued: Shuffling and sorting

	Input	Output
map	<K1,V1>	<K2,V2>
reduce	<K2, List(V2)>	<K3,V3>

The over all map-reduce word-count process

1. By default, the line address of every line is they key and value is the contents of the line. These Key-Value pairs will be the input to the map function

2. Each Mapper will generate new Key-Value pairs based on the map function, In this case new key is the word and value is 1.

3. Key-Value pairs will be merged on the same key before sending it to the reducer. In this case, since word is the key, so all the key-value pairs associated with same word are merged

4. Finally, the reducer function run on the values associated to same key, and produces the result

Notes: Shuffling and sorting – transferring data from mappers to reducers

- Key-value pairs that mappers output are transmitted within the cluster to the machines running reducers the reducers they've been assigned to by the partitioner

- While it is "invisible" to the end-user, from the perspective of the time taken to run an MR job, this phase is a major component since this is where data is being communicated between different machines

- Example 1: Consider again the summing example
  - Without a combiner, the shuffling phase would effectively end up transmitting **all** 999 billion rows worth of data on 999 machines onto one machine and the reducer running on this machine would then be responsible for summing a trillion numbers
  - Assume it takes k seconds to transmit a single key-value pair
  - That then implies a total processing time of around $(k+n) \times 10^{12}$ seconds to run the process in its entirety

- Sorting of the mapper output keys occurs **before** reducing and **after** shuffling on the machines running reducers

- Because of this sorting, mapper output keys must be **comparable**

Agenda

- Introduction to Distributed computing
- Example: Word count using map-reduce
- Key-value pairs – The core of map-reduce
- The map-reduce paradigm – The step-by-step approach
- The map phase – Extreme parallelism
- Shuffling, Sorting – The invisible middle operations
- The reduce phase – Bringing it all together
- Examples and exercises

Example 2 continued: Reducing & writing out

	Input	Output
map	<K1,V1>	<K2,V2>
reduce	<K2, List(V2)>	<K3,V3>

The over all map-reduce word-count process

1. **By default, the line address of every line is they key and value is the contents of the line. These Key-Value pairs will be the input to the map function**

2. **Each Mapper will generate new Key-Value pairs based on the map function, In this case new key is the word and value is 1.**

3. **Key-Value pairs will be merged on the same key before sending it to the reducer. In this case, since word is the key, so all the key-value pairs associated with same word are merged**

4. **Finally, the reducer function run on the values associated to same key, and produces the result**

Notes: The last phase in the MR process depends on how the input has been mapped

- Each reducer reads in a pair of **one** key and list of **all** it's associated values and writes out **zero, one or more** key-value pair
- Since each reducer takes as input all the values associated with a mapper output key, reducers can also run in **parallel**, but they cannot run before all mappers have completed running
- One or more reducers can be run in a MR job, often depending on run-time considerations, but the idea of "more reducer the merrier" should be approached with care
- Example 1 continued: In the summing example
  - All the mappers write out the same key, so it doesn't make sense to run more than one reducer
  - However if we now wish to take subtotals of the "number of store visits" column across levels of another column, we can pass the rows of that column as the mapper output keys and the rows of "number of store visits" to be summed as the mapper output values
- A lot of data summarization tasks can be achieved using only reducers in conjunction with **identity mappers**, mappers that simply pass on the key-value pairs that they receive as input such that the computed output is some aggregation over the input rows

Agenda

- Introduction to Distributed computing
- Example: Word count using map-reduce
- Key-value pairs – The core of map-reduce
- The map-reduce paradigm – The step-by-step approach
- The map phase – Extreme parallelism
- Shuffling, Sorting – The invisible middle operations
- The reduce phase – Bringing it all together
- Examples and exercises

Example 1: Problem statement

- Consider data that tracks store daily visits at all outlets of a retail chain. Sample in Table A:
  - How would you compute total store visits across all stores?
  - How would you compute total store visits for all stores after 31 January 2011?
  - How would you compute store-wise total visits after 31 January 2011 for each store?
  - How would you sort this dataset on store?
- Consider another dataset that tracks total number of sales made each day for each store. Sample in Table B:
  - how would you merge the two tables so that store visits and total number of sales made can be viewed side-by-side?

Table A

Visit Date	Store	Store visits
2011-11-30	A	128
2011-11-23	A	34
2012-01-11	B	435
2010-09-15	A	786
2012-01-19	B	93
2012-01-14	B	847
2010-11-30	A	297
2010-07-25	B	218
2011-07-04	B	454
2010-10-19	A	23
2010-02-01	B	634
2011-11-26	A	9
2011-08-18	A	12
…	…	…

Table B

Visit Date	Store	# Sales
2011-11-30	A	46
2011-11-23	A	15
2012-01-11	B	223
2010-09-15	A	497

Example 1: Pseudo-code for filtering and summation using Map-Reduce

- In the example summing a trillion rows, let us now add one more constraint
  - We wish to now sum the values of “number of store visits” for only those rows that correspond to a visit date after 31 January, 2011 on the basis of the “visit date” column
  - Assume that “visit date” is the first column and “number of store visits” is the third column
- In pseudo-code, the mapper, combiner and reducer would look something like:

```
let map(key, value) =
        if (yyyymmdd(value[1]) > d"2011-01-31") emit(const, value[3])

let combine(key, values) =
        foreach v in values:
                sum += v
        emit(key, sum)

let reduce(key, values) = combine
```

- Notice that the reducer is implementing the same logic as the combiner, only that the values that it is summing across are **all** the locally computed subtotals that are output from each combiner

Example 1: Pseudo-code for filtering and computing monthly store-wise subtotals using Map-Reduce

- In the example summing a trillion rows, let us now add a further constraint
  - we wish to now compute monthly subtotals of store-wise store visits for all visits after 31 January 2011
  - The “store” column contains string identifiers for stores and is the second column
- in pseudo-code the mapper, combiner and reducer would look something like:

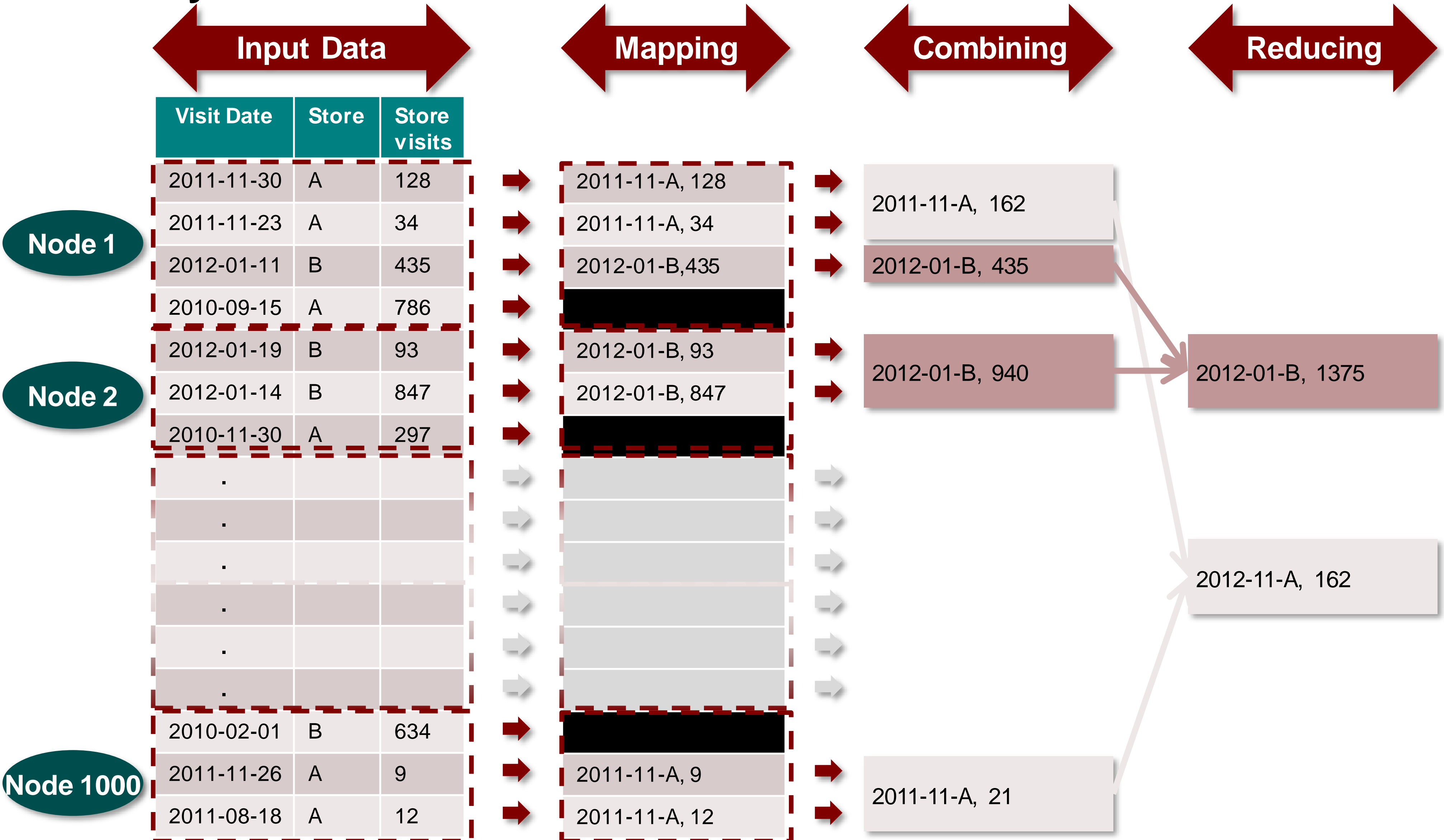
```
let map(key, value) =
        if (yyyymmdd(value[1]) > d"2011-01-31")
                emit(concatenate(year(value[1]), "-", month(value[1]), "-", value[2]), value[3])

let combine(key, values) =
        foreach v in values:
                sum += v
        emit(key, sum)

let reduce(key, values) = combine(key, values)
```

- Note that the reducer is still the same as the combiner, only that the values that it is summing across are **all** the locally computed subtotals that are output from each combiner **corresponding to the same combiner output key**

Example 1: Filtering and computing monthly subtotals explained visually



Example 1: Pseudo-code for sorting the table using Map-Reduce

- For sorting a dataset on the values of the first column, the MR pseudo-code would look something like:

```
let map(key, value) =
        emit(value[2]), value)

let reduce(key, values) =
        foreach v in values:
                emit(key, v)
```

- Since Hadoop sorts the reducer input keys before running the reducer, we leverage this property by simply setting the mapper output key to the value of the column we wish to sort on

Example 1: Pseudo-code for joining two tables using Map-Reduce

- For joining two datasets, A and B on the second column in both, the MR pseudo-code would look something like:

```
let map(key, value) =
        emit(value[2]), concatenate(name(input), "-", as.character(value)))

let reduce(key, values) =
        foreach v in values:
                if (splitString(v, "-")[1] == "A")
                        aData = splitString(v, "-")[-1]
                        exit foreach
        foreach v in values:
                if (splitString(v, "-")[1] != "A")
                        emit(key, concatenate(splitString(v, "-")[-1], “,", aData))
```

- We use the mapper to group all rows from tables A and B corresponding to a store by setting the mapper output key as the store and since all values corresponding to a mapper output key are sent to one reducer
- We then locally find the value corresponding to dataset A and concatenate that value with all other values (which will then be from dataset B)

Exercises

1) The trickiness of parallelization – Averaging

- Given the following data

Node	Store	Week	No of customers
Node 1	A	1	300
	A	2	150
	B	1	200
	B	2	100
	C	1	250
	C	2	50
Node 2	A	3	500
	A	4	400
	B	3	350
	C	3	450

- Using the map-reduce framework, how would you calculate the average weekly customers by store for available data

2) Linear-algebra in map-reduce (hint: getting started with OLS)

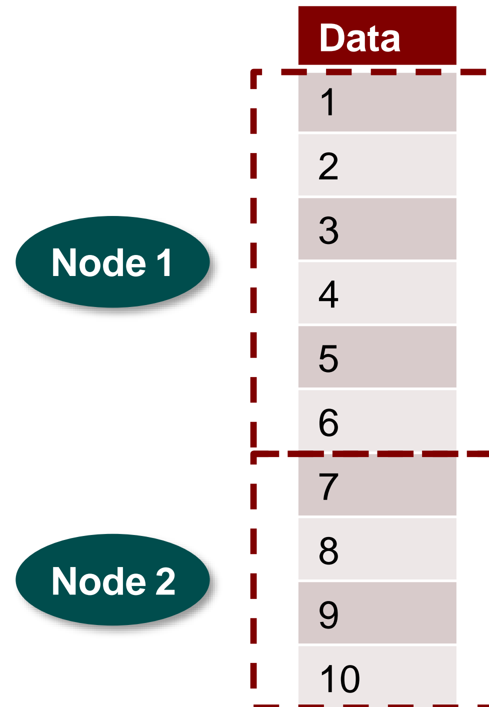
- Given the following matrices

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \\ 9 & 10 \end{bmatrix} \qquad \boldsymbol{B} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \qquad \boldsymbol{C} = \begin{bmatrix} 7 & 8 \\ 9 & 10 \end{bmatrix}$$

- What are the dimensions and values of $\boldsymbol{A}^T\boldsymbol{A}$, $\boldsymbol{B}^T\boldsymbol{B}$ and $\boldsymbol{C}^T\boldsymbol{C}$?
- Is $\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{B}^T\boldsymbol{B} + \boldsymbol{C}^T\boldsymbol{C}$?
- Is there any information lost when transforming from $\boldsymbol{A}$ to $\boldsymbol{A}^T\boldsymbol{A}$?
- What if $\boldsymbol{A}$ had a **billion** rows and a **hundred** columns? Would this sort of approach perhaps be helpful when trying to perform OLS? (Hint: $\boldsymbol{X}^T\boldsymbol{X}\vec{\boldsymbol{\beta}} = \boldsymbol{X}^T\vec{\boldsymbol{y}}$)

Closing thoughts – limitations of Map-Reduce

- Given the data shown to the right, how can you compute a running total of the rows in map-reduce?
- Not all computations are easily parallelized
- As a thumb-rule, computations for which a divide and conquer scheme can be devised are amenable to map-reduce
- As another thumb-rule, computations that inherently serial in nature and not of complexity type NC are not amenable to map-reduce
  - which is not to say they **cannot** be performed in a map-reduce framework,
  - it just means that the reductions in processing time that arise out of parallel computation in a map-reduce framework might not always be realizable in these cases
- Problems like linear-programming, inversion of large dense matrices, agent-based methods are not easily parallelizable



Reference material

- On the Internet
  - http://wiki.apache.org/hadoop/HadoopMapReduce
  - http://code.google.com/edu/parallel/mapreduce-tutorial.html
  - http://en.wikipedia.org/wiki/MapReduce
  - http://developer.yahoo.com/hadoop/tutorial/module4.html
  - http://hadoop.apache.org/common/docs/current/mapred_tutorial.html
  - http://vimeo.com/3584536
  - http://www.cloudera.com/videos/programming_with_hadoop
  - http://www.cloudera.com/videos/mapreduce_algorithms
  - Google videos on Cluster Computing and Map Reduce
    - Lecture 1 - http://www.youtube.com/watch?v=yjPBkvYh-ss&feature=relmfu
    - Lecture 2 - http://www.youtube.com/watch?v=-vD6PUdf3Js&feature=relmfu
    - Lecture 3 - http://www.youtube.com/watch?v=5Eib_H_zCEY&feature=relmfu
    - Lecture 4 - http://www.youtube.com/watch?v=1ZDybXl212Q&feature=relmfu
    - Lecture 5 - http://www.youtube.com/watch?v=BT-piFBP4fE&feature=relmfu

- Textbooks
  - **Data-Intensive Text Processing with Map Reduce** by Jimmy Lin
  - **Hadoop: The Definitive Guide** by Tom White.

Chicago, IL
Bangalore, India
January 25, 2012
www.mu-sigma.com

Proprietary Information

"This document and its attachments are confidential. Any unauthorized copying, disclosure or distribution of the material is strictly prohibited"