BIG DATA (CS5620)

Introduction to MapReduce

Instructor: Dr. Yui Man Lui

Why MapReduce

Big Data

- Big data is everywhere
  - There are 2,161,530,000,000 searches in 2013
  - 92,100,000 pages mentioning Albert Einstein
  - Only 38,400 pages mentioning Yui Man Lui
- Gather as much data as you need and run machine learning / data mining / analytics algorithms to generate insights
- Challenges (3Vs)
  - Big Volume – the quantity of generated and stored data
  - Big Velocity – the speed at which the data is generated and processed
  - Big variety – the type and nature of the data

Big Data

- Need a fast way to process large amount of data
- MapReduce provides a framework for parallel computations using a large number of nodes
- MapReduce produces a scale-out, distributed, and fault-tolerant solution
- Hadoop / GFS is designed to use MapReduce as the primary method of interaction

What is MapReduce

MapReduce

- MapReduce is inspired by the concept of map and reduced in *Lisp*
  - A map function takes parameters as a function and a set of values
  - (map ‘length ‘(() (a) (ab) (abc))) ➔ (0 1 2 3)
  - The reduce function is a binary function and a set of values as parameters. It combines all values together using the given function
  - (reduce #'+ '(0 1 2 3)) ➔ 6

MapReduce

- MapReduce is a programming model
- MapReduce provides an abstraction to perform simple computations while hiding the details of parallelization, data distribution, load balancing, and fault tolerance
- Developed in Google as a mechanism for processing large amounts of data
  - Distributed data across thousands of machines
  - Same computations are performed on each CPU with different subsets of data

MapReduce

- The world has changed

Google Stanford Hardware (1998)

Google Data Center, Iowa

Need to think parallel

How MapReduce Works

Mapper

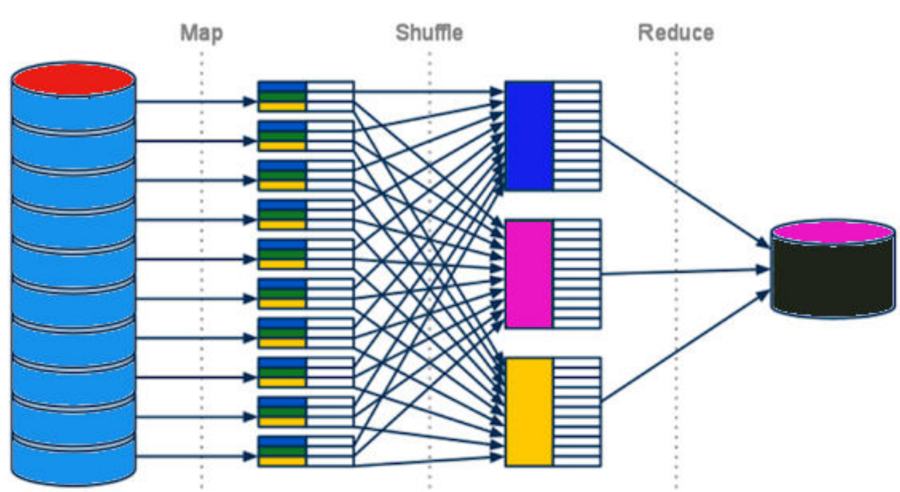
- Mapper maps input (key, value) pairs to a set of intermediate (key, value) pairs
  - Maps are individual tasks that transform input data to intermediate records
  - A given input may map to zero or many output pairs

Reducer

- Reducer reduces a set of intermediate values which share a key (hash key) to a smaller set of values

- Reducer has three primary phases – Shuffle, Sort, Reduce
  - Shuffle – Take similar data and group them together
  - Sort – Groups the data to the reducer by keys
  - Reduce – Aggregate the data and summarize it in some way

Schematic of MapReduce Computation

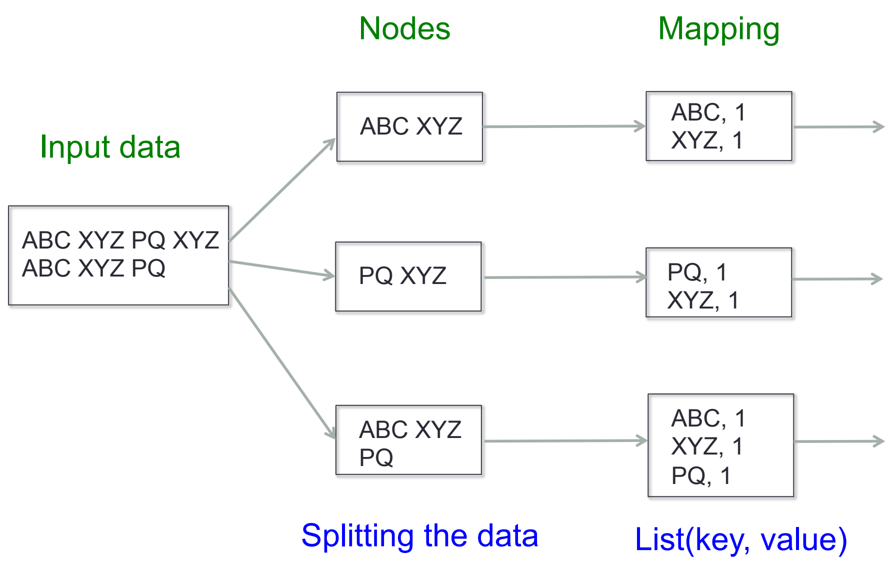


Schematic of MapReduce Computation

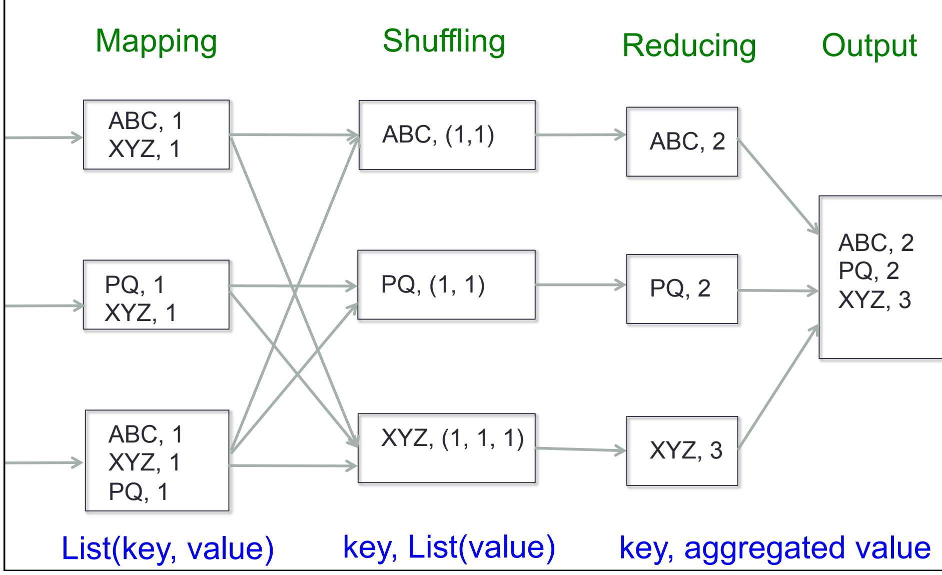
- Split phase – partitions the data across different mappers
  - Think they are different machines
- Each mapper executes *user defined* map code on the partitions in parallel
- Data is shuffled such that there is one reducer per key
  - Think they are different machines
- Each reducer executes the *user defined* reduce code in parallel

Word Count Example

MapReduce – Word Count Example



MapReduce – Word Count Example



Summary

- Input data distributed to nodes (servers)
- Each map works on a split data
- Data exchange between nodes in a “shuffle” process
- Intermediate data of the same key goes to the same reducer
- Reducer summarizes the values
- Reducer output is stored

- Need to provide a functional abstraction for mapper and reducer

Python - Word Count Example

MapReduce – Word Count Example

```
from multiprocessing import Pool

if __name__ == '__main__':
    text = loadtext('simple.txt')
    numProcessors = 3
    pool = Pool(processes=numProcessors)
    partitioned_text = chunks(text, 1 + len(text) / numProcessors)

    single_count_tuples = pool.map(Map, partitioned_text)
    token_to_tuples = Partition(single_count_tuples)
    term_frequencies = pool.map(Reduce, token_to_tuples.items())

    term_frequencies.sort(tuple_sort)

    for pair in term_frequencies:
        print('%20s: %5s' % (pair[0], pair[1]))
```

MapReduce – Word Count Example

```
def loadtext(path):
    word_list = []
    f = open(path, "r")
    for line in f:
        line = line.replace('.','')
        word_list.append(line)

 ## " ".join(["a", "b", "c"]).split()   ->  ['a', 'b', 'c']
        text = (''.join(word_list)).split()
        return text

def chunks(text, n):
    data_block = []
    for i in xrange(0, len(text), n):
        data_block.append(text[i:i+n])

    return list(data_block)
```

MapReduce – Word Count Example

```
def Partition(lst):
    term_freq = {} # dictionary
    for sublst in lst:
        for w in sublst:
        ## if not a key in dictionary, then create a new key
        if (term_freq.get(w[0]) == None):
            term_freq[w[0]] = [w]
        else:
        ## otherwise append to the end
            term_freq[w[0]].append(w)

        return term_freq
```

MapReduce – Word Count Example

```
def Map(lst):
    results = []
    for word in lst:
        results.append((word, 1))

    return results

def Reduce(mapping):
    return (mapping[0], sum(pair[1] for pair in mapping[1]))

def tuple_sort (a, b):
    if (a[1] < b[1]):
      return 1
    elif (a[1] > b[1]):
      return -1
    else:
      return cmp(a[0], b[0])
```

MapReduce – Word Count Example

```
SINGLE COUNT TUPLES

[[('ABC', 1), ('XYZ', 1), ('PQ', 1)], [('XYZ', 1), ('ABC', 1), ('XYZ', 1)], [('PQ', 1)]]
------------------------------
PARTITION DATA

{'XYZ': [('XYZ', 1), ('XYZ', 1), ('XYZ', 1)], 'ABC': [('ABC', 1), ('ABC', 1)], 'PQ':
[('PQ', 1), ('PQ', 1)]}
------------------------------
TERM FREQUENCIES

[('XYZ', 3), ('ABC', 2), ('PQ', 2)]
------------------------------
      XYZ:    3
      ABC:    2
       PQ:    2
```

Python – MRJOB
Word Count Example

MapReduce – Word Count Example

```
from mrjob.job import MRJob
import re
WORD_RE = re.compile(r"[\w']+")

class MRWordFreqCount(MRJob):
    def mapper(self, _, line):
        for word in WORD_RE.findall(line):
        ## we see this word one time
            yield (word, 1)

    def reducer(self, word, counts):
        yield (word, sum(counts))

if __name__ == '__main__':
    MRWordFreqCount.run()
```

MapReduce – Word Count Example

- python mr_wordcount-2.py simple.txt > output.txt
- cat output.txt

```
"ABC"   2
"PQ"    2
"XYZ"   3
```

Python – MRJOB
Another Word Count Example

MapReduce – Word Count Example

```
from mrjob.job import MRJob

class MRWordFrequencyCount(MRJob):

    def mapper(self, _, line):
        yield "chars", len(line)
        yield "words", len(line.split())
        ## we see one line
        yield "lines", 1

    def reducer(self, key, values):
        yield key, sum(values)


if __name__ == '__main__':
    MRWordFrequencyCount.run()
```

MapReduce – Word Count Example

- python mr_wordcount_example.py test.txt > output.txt
- cat output.txt

```
"chars"  464
"lines"  2
"words"  78
```

Natural Join using MapReduce

Natural Join

- Joining database A and database B
  - Database A has attributes A1, A2, and A3
  - Database B has attributes A3, A4, and A5
- Find the records that agree on their attribute A3
  - i.e, the third attribute in database A and the first attribute in database B

Natural Join Example

sid	bid	day
22	101	10/10/96
58	103	11/12/96

R1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1

$R1 \bowtie S1$ =

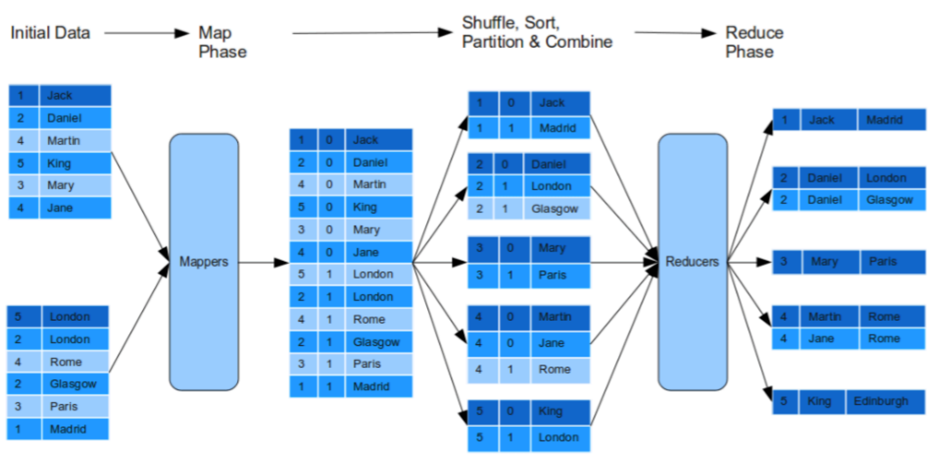
sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

Natural Join using MapReduce

- Mapper
  - For each record (A1, A2, A3) in database A
  - Produce a key-value pair (A3, (“A”, (A1,A2)))
  - For each record (A3, A4, A5) in database B
  - Produce a key-value pair (A3, (“B”, (A4,A5)))
- Reducer
  - For all of the pairs with same key
  - Construct all pairs comprising one with “A” and other with the first component “B”
  - e.g. , (“A”, (A1,A2)) and (“B”, (A4,A5))
  - Produce a record with (A1, A2, A3, A4, A5)

Natural Join using MapReduce

- Two-way joins (can be extended to multi-way joins)
  - The join operation actually happens on the reduce side



MapReduce Implementation

- Google has a proprietary implementation in C++
  - Bindings in Java, Python
- Hadoop is an open-source implementation in Java

Supporting Casts for Hadoop

- High Level languages for describing MapReduce applications
  - Pig (data exploration) - for data processing written in Pig Latin (dataflow language) developed by Yahoo
  - Hive (data warehousing) – Query language HQL developed by Facebook
  - MRJOB – supports Amazon's Elastic MapReduce service developed by Yelp
- HBase
  - Column-oriented distributed DBMS
- ZooKeeper
  - Coordination service for distributed applications