大数据分析

Large-scale computing

刘盛华

Large-scale Computing

- Large-scale computing for data mining problems on commodity hardware
- Challenges:
  - How do you distribute computation?
  - How can we make it easy to write distributed programs?
  - Machines fail:
    - One server may stay up 3 years (1,000 days)
    - If you have 1,000 servers, expect to loose 1/day
    - People estimated Google had ~1M machines in 2011
      - 1,000 machines fail every day!

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Idea and Solution

- Issue: Copying data over a network takes time
- Idea:
  - Bring computation close to the data
  - Store files multiple times for reliability
- Map-reduce addresses these problems
  - Google's computational/data manipulation model
  - Elegant way to work with big data
  - Storage Infrastructure – File system
    - Google: GFS. Hadoop: HDFS
  - Programming model
    - Map-Reduce

4

Storage Infrastructure

- **Problem:**
  - If nodes fail, how to store data persistently?
- **Answer:**
  - **Distributed File System:**
    - Provides global file namespace
    - Google GFS; Hadoop HDFS;
- **Typical usage pattern**
  - Huge files (100s of GB to TB)
  - Data is rarely updated in place
  - Reads and appends are common

5

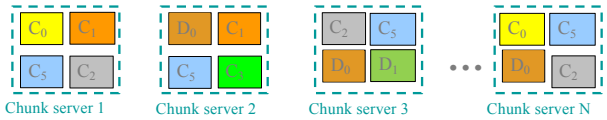
Distributed File System

- **Chunk servers**
  - 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
- **Master node**
  - a.k.a. Name Node in Hadoop's HDFS
  - Stores metadata about where files are stored
  - Might be replicated
- **Client library for file access**
  - Talks to master to find chunk servers
  - Connects directly to chunk servers to access data

6

Distributed File System

- Reliable distributed file system
- Data kept in "chunks" spread across machines
- Each chunk replicated on different machines
  - Seamless recovery from disk or machine failure



Bring computation directly to the data!

Chunk servers also serve as compute servers

7

Example: World Counting

Example MapReduce task:

- We have a huge text document
- Count the number of times each distinct word appears in the file
- **Sample application:**
  - Analyze web server logs to find popular URLs
  - Statistical machine translation:
    - Need to count number of times every 5-word sequence occurs in a large corpus of documents

8

MapReduce: Overview

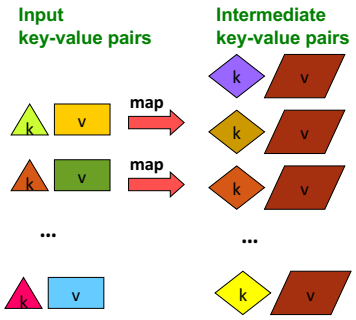
3 steps of MapReduce

- Sequentially read a lot of data
- Map:
  - user-written Map function
  - Extract something you care about
- Group by key: Sort and Shuffle
  - System sorts all the key-value pairs by key
  - outputs key-(list of values) pairs
- Reduce:
  - User-written Reduce function
  - Aggregate, summarize, filter or transform
- Write the result

Outline stays the same, **Map** and **Reduce** change to fit the problem

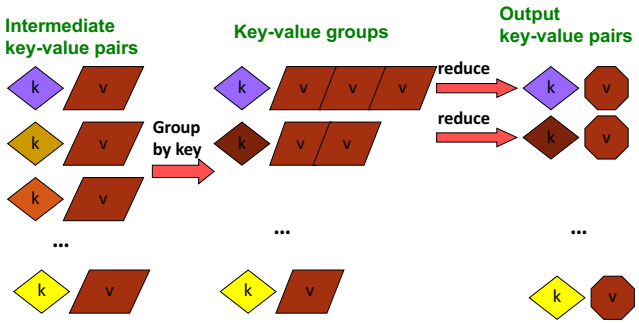
9

MapReduce: The Map Step



10

MapReduce: The Reduce Step



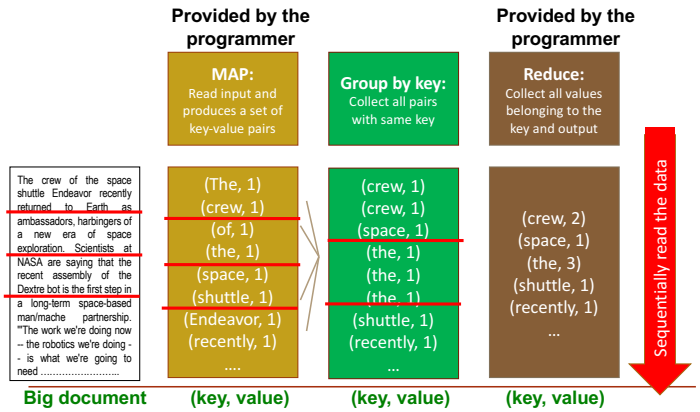
11

More Specifically

- Input: a set of key-value pairs
- Programmer specifies two methods:
  - Map(k, v) → <k', v'>*
    - Takes a key-value pair and outputs a set of key-value pairs
      - E.g., key is the filename, value is a single line in the file
    - There is one Map call for every *(k,v)* pair
  - Reduce(k', <v'>*) → <k', v''>*
    - All values *v'* with same key *k'* are reduced together and processed in *v'* order
    - There is one Reduce function call per unique key *k'*

12

MapReduce: Word Counting



Word Count Using MapReduce

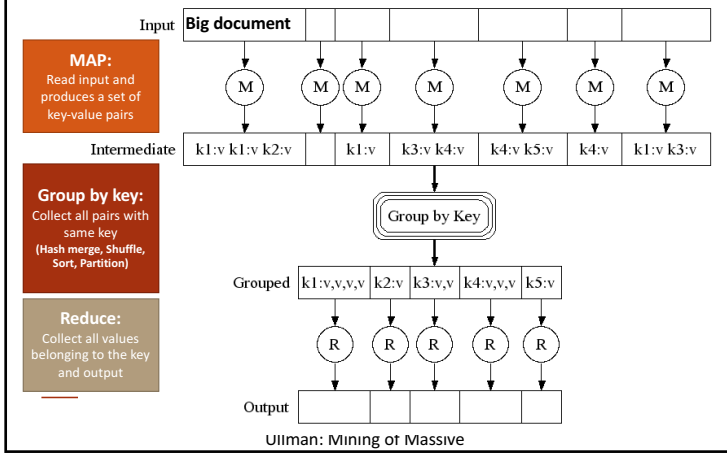
```
map(key, value):
// key: document name; value: text of the document
  for each word w in value:
      emit(w, 1)


reduce(key, values):
// key: a word; value: an iterator over counts
      result = 0
      for each count v in values:
            result += v
      emit(key, result)
```


Map-Reduce: Environment

Map-Reduce environment takes care of:

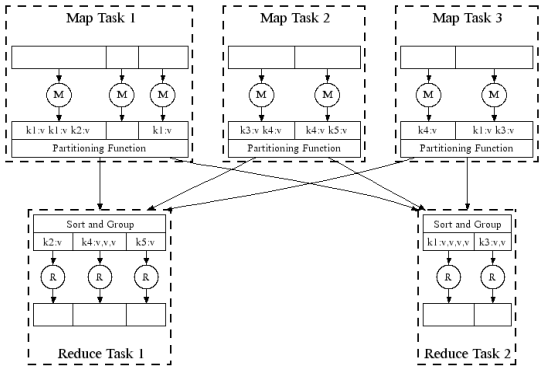
- **Partitioning the input data**
- **Scheduling the program's execution across a set of machines**
- **Performing the group by key step**
- **Handling machine failures**
- **Managing required inter-machine communication**

Map-Reduce: A diagram



Ullman: Mining of Massive

Map-Reduce: In Parallel

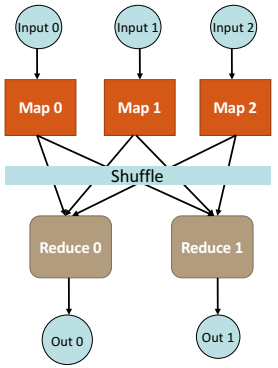


All phases are distributed with many tasks doing the work

17

Map-Reduce

- **Programmer specifies:**
  - Map and Reduce and input files
- **Workflow:**
  - Read inputs as a set of key-value-pairs
  - Map transforms input kv-pairs into a new set of k'v'-pairs
  - Sorts & Shuffles the k'v'-pairs to output nodes
  - All k'v'-pairs with a given k' are sent to the same reduce
  - Reduce processes all k'v'-pairs grouped by key into new k''v''-pairs
  - Write the resulting pairs to files
- **All phases are distributed with many tasks doing the work**



18

Data Flow

- **Input and final output** are stored on a **distributed file system (FS):**
  - Scheduler tries to schedule map tasks "close" to physical storage location of input data
- **Intermediate results** are stored on **local FS** of Map and Reduce workers
- Output is often input to another MapReduce task

19

Coordination: Master

- **Master node takes care of coordination:**
  - Task status: (idle, in-progress, completed)
  - Idle tasks get scheduled as workers become available
  - When a map task completes, it sends the master the location and sizes of its *R* intermediate files, one for each reducer
  - Master pushes this info to reducers
- Master pings workers periodically to detect failures

20

Dealing with Failures

- **Map worker failure**
  - Map tasks completed or in-progress at worker are reset to idle
  - Reduce workers are notified when task is rescheduled on another worker
- **Reduce worker failure**
  - Only in-progress tasks are reset to idle
  - Reduce task is restarted
- **Master failure**
  - MapReduce task is aborted and client is notified

21

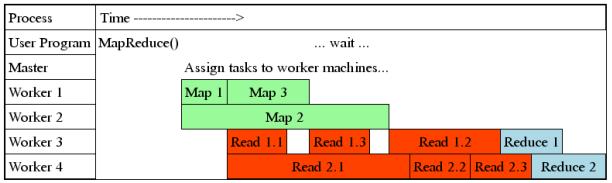
How many Map and Reduce jobs?

- *M* map tasks, *R* reduce tasks
- **Rule of a thumb:**
  - Make *M* much larger than the number of nodes in the cluster
  - One DFS chunk per map is common
  - Improves dynamic load balancing and speeds up recovery from worker failures
- Usually *R* is smaller than *M*
  - Because output is spread across *R* files

22

Task Granularity & Pipelining

- **Fine granularity tasks:** map tasks >> machines
  - Minimizes time for fault recovery
  - Can do pipeline shuffling with map execution
  - Better dynamic load balancing



23

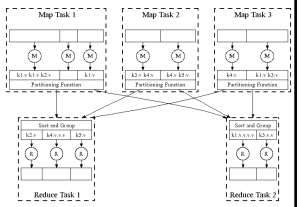
Refinements: Backup Tasks

- **Problem**
  - Slow workers significantly lengthen the job completion time:
    - Other jobs on the machine
    - Bad disks
    - Weird things
- **Solution**
  - Near end of phase, spawn backup copies of tasks
    - Whichever one finishes first "wins"
- **Effect**
  - Dramatically shortens job completion time

24

Refinement: Combiners

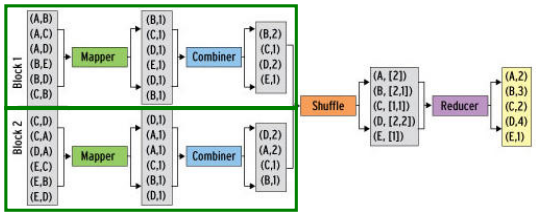
- Often a Map task will produce many pairs of the form $(k,v_1), (k,v_2), \ldots$ for the same key k
  - E.g., popular words in the word count example
- Can save network time by pre-aggregating values in the mapper:
  - combine(k, list(v_1)) → v_2
  - Combiner is usually same as the reduce function
- Works only if reduce function is commutative and associative



25

Refinement: Combiners

- Back to our word counting example:
  - Combiner combines the values of all keys of a single mapper (single machine):



  - Much less data needs to be copied and shuffled!

26

Refinement: Partition Function

- Want to control how keys get partitioned
  - Inputs to map tasks are created by contiguous splits of input file
  - Reduce needs to ensure that records with the same intermediate key end up at the same worker
- System uses a default partition function:
  - hash(key) mod *R*
- Sometimes useful to override the hash function:
  - E.g., hash(hostname(URL)) mod *R* ensures URLs from a host end up in the same output file

27

Problems Suited for Map-Reduce

Example: Host size

- Suppose we have a large web corpus
- Look at the metadata file
  - Lines of the form: (URL, size, date, …)
- For each host, find the total number of bytes
  - That is, the sum of the page sizes for all URLs from that particular host
- Other examples:
  - Link analysis and graph processing
  - Machine Learning algorithms

29

Example: Language Model

- Statistical machine translation:
  - Need to count number of times every 5-word sequence occurs in a large corpus of documents
- Very easy with MapReduce:
  - Map:
    - Extract (5-word sequence, count) from document
  - Reduce:
    - Combine the counts

30

Example: Join By Map-Reduce

- Compute the natural join $R(A,B) \bowtie S(B,C)$
- *R* and *S* are each stored in files
- Tuples are pairs *(a,b)* or *(b,c)*

A	B
a_1	b_1
a_2	b_1
a_3	b_2
a_4	b_3

R

$\bowtie$

B	C
b_2	c_1
b_2	c_2
b_3	c_3

S

=

A	C
a_3	c_1
a_3	c_2
a_4	c_3

31

Map-Reduce Join

- Use a hash function *h* from B-values to *1...k*
- A Map process turns:
  - Each input tuple *R(a,b)* into key-value pair *(b,(a,R))*
  - Each input tuple *S(b,c)* into *(b,(c,S))*
- Map processes send each key-value pair with key *b* to Reduce process *h(b)*
  - Hadoop does this automatically; just tell it what *k* is.
- Each Reduce process matches all the pairs *(b,(a,R))* with all *(b,(c,S))* and outputs *(a,b,c)*.

32

Cost Measures for Algorithms

- In MapReduce we quantify the cost of an algorithm using

1. *Communication cost* = total I/O of all processes
2. *Elapsed communication cost* = max of I/O along any path
3. (*Elapsed*) *computation cost* analogous, but count only running time of processes

Note that here the big-O notation is not the most useful (adding more machines is always an option)

33

Example: Cost Measures

- For a map-reduce algorithm:
  - (Total) communication cost = input file size + 2 × (sum of the sizes of all files passed from Map processes to Reduce processes) + the sum of the output sizes of the Reduce processes.
  - Elapsed communication cost is the sum of the largest input + output for any map process, plus the same for any reduce process

34

What Cost Measures Mean

- Either the I/O (communication) or processing (computation) cost dominates
  - Ignore one or the other
- Total communication cost tells what you pay in rent from your friendly neighborhood cloud
- Elapsed cost is wall-clock time using parallelism

35

Cost of Map-Reduce Join

- Total communication cost
  = $O(|R|+|S|+|R \bowtie S|)$
- Elapsed communication cost = $O(s)$
  - We're going to pick k and the number of Map processes so that the I/O limit s is respected
  - We put a limit s on the amount of input or output that any one process can have. s could be:
    - What fits in main memory
    - What fits on local disk
- With proper indexes, computation cost is linear in the input + output size
  - So computation cost is like comm. cost

36

Further Reading

Reading

- Jeffrey Dean and Sanjay Ghemawat: MapReduce: Simplified Data Processing on Large Clusters
  - http://labs.google.com/papers/mapreduce.html
- Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung: The Google File System
  - http://labs.google.com/papers/gfs.html

38

Resources

- Hadoop Wiki
  - Introduction
    - http://wiki.apache.org/lucene-hadoop/
  - Getting Started
    - http://wiki.apache.org/lucene-hadoop/GettingStartedWithHadoop
  - Map/Reduce Overview
    - http://wiki.apache.org/lucene-hadoop/HadoopMapReduce
    - http://wiki.apache.org/lucene-hadoop/HadoopMapRedClasses
  - Eclipse Environment
    - http://wiki.apache.org/lucene-hadoop/EclipseEnvironment
- Javadoc
  - http://lucene.apache.org/hadoop/docs/api/

39

Resources

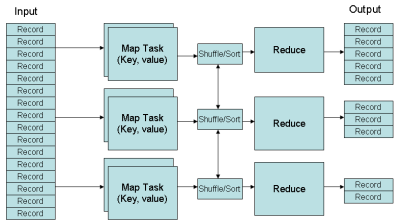
- Releases from Apache download mirrors
  - http://www.apache.org/dyn/closer.cgi/lucene/hadoop/
- Nightly builds of source
  - http://people.apache.org/dist/lucene/hadoop/nightly/
- Source code from subversion
  - http://lucene.apache.org/hadoop/version_control.html

40

Popular Systems

Hadoop Map-Reduce

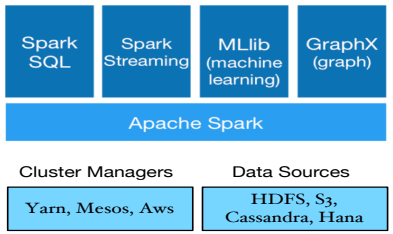
- **Born in 2006 (Doug Cutting → Yahoo!)**
- **Disk-based (HDFS) map-reduce**
- **HDFS ≈ Google GFS**



https://medium.com/@markobonaci/the-history-of-hadoop-68984a11704

Spark

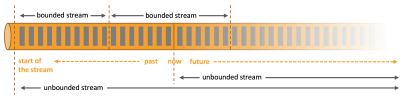
- **Born in UC Berkeley 2009**
  - **improve Map-Reduce**
  - in-memory computing : RDD
    - RDD copy between iterations (Hadoop passes results by HDFS)
    - Transformations: RDD →RDD
    - Actions: RDD → other data structure
- **Spark Software Stack**
  - **SQL**
  - **Streaming**
  - **Mllib**
  - **GraphX**



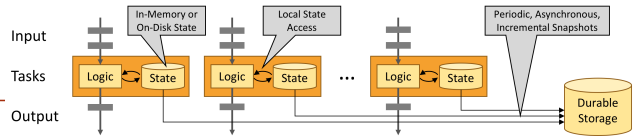
43

Flink

- scalable and fault-tolerant processing framework for streams of data
  - bounded stream
  - unbounded stream



- **states**
  - **local states** leverage in-memory performance
  - durable storage guarantees exactly-once state consistency in case of failures



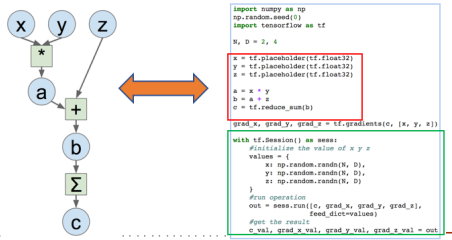
TensorFlow/Pytorch

- **Popular deep learning frameworks**
- **Make highly use of GPU**
- **Point#1**
  - **TF is based on Theano, developed by Google**
  - **Pytorch is based on Torch, developed by Google**
- **Point #2:**
  - **define the computation graphs**
  - **TF creates a static graph**
  - **PyTorch define/manipulate your graph on-the-go**
    - variable length inputs in RNNs
- Point #3:
  - TensorBoard

45

Tensorflow computation graph

- **Example**
  - Graph：represent the computation as a graph
  - Tensor：data is tensors in the graph
  - Flow：the process of tensors computed in the graph



Other systems

- **GraphLab/GraphX**
  - **for iterative graph computing**
- **Pegasus/PegasusN**
  - **graph-mining system with massive scalability**
  - **matrix and vector product**
- **Caffe**
  - **most used for CNN, not friendly to RNN**