第二章
Map-Reduce计算范式及其软件栈

主讲：陈爱国

大数据分析与挖掘

outline

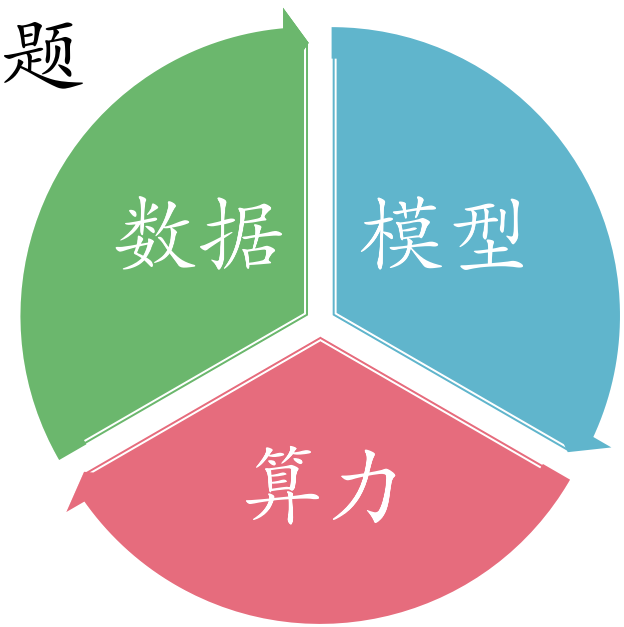
Lecture 2.1 large scale computing
Lecture 2.2 Distributed File Systems
Lecture 2.3 Programming Model
Lecture 2.4 Problems Suited for Map-Reduce

新基建的概念及背景

- 2018年12月，中央经济工作会议，把5G、人工智能、工业互联网、物联网定义为“新型基础设施建设”。
- 随后“加强新一代信息基础设施建设”被列入2019年政府工作报告 。2019年7月，中共中央政治局召开会议，提出“加快推进信息网络等新型基础设施建设”。
- 2020年4月，国家发展改革委首次明确“新基建” 包括三大领域：信息基础设施、融合基础设施和创新基础设施。在信息基础设施领域，人工智能与云计算、区块链一起被视为一种新技术基础设施；在融合基础设施中，人工智能则被视为支撑传统基础设施转型升级的重要工具。
- 2020年中央经济工作会议上，国家提出了“加快实施创新驱动发展战略，推动形成国内大循环”。

- “基建”是生产力的生产力
- “新基建”是大数据、人工智能时代，提高生产力的基础（平行世界的基建）
- 解决大规模数据处理的算力问题，可以说是“新基建”的一项重要内容，也是我们长期以来被卡脖子的问题

智能化的关键要素，也是数据分析挖掘的关键要素

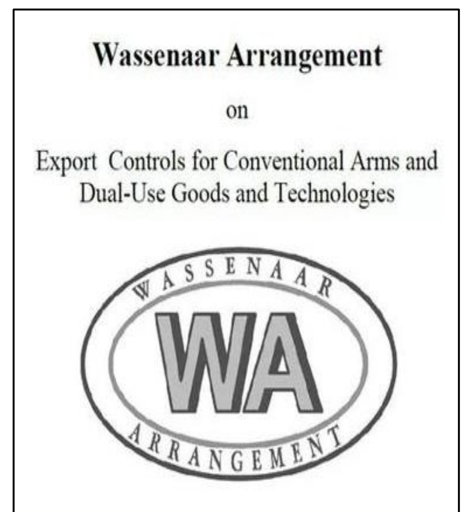


解决算力问题面临的情况

1979年中美正式建交，《中美科技合作协定》签署，逐步放宽计算机技术和通信设备的出口管制

但是，以下的政策，华裔工程师进不到欧美半导体公司核心部门，国内买不到近两代的关键设备

《巴黎统筹委员会》



《瓦森纳协定》

国内算力发展？

- 计算力的第一个里程碑：90年代引进西方的芯片、PC和操作系统。（造vs 买？）
  - 高端计算，依赖服务器，IBM 的服务器（大/小型机）
  - 架构特殊的封闭性，物以稀为贵
- 计算力的第二座里程碑：“云计算”的创世。（例如成本更低的 x86 服务器，去IOE）
- 计算力的第三座里程碑：服务器的遍地开花。
  - 从“通用计算”变成“异构计算”
  - 芯片开源，RISC-V 开源架构

最后问题的关键落到了芯片！

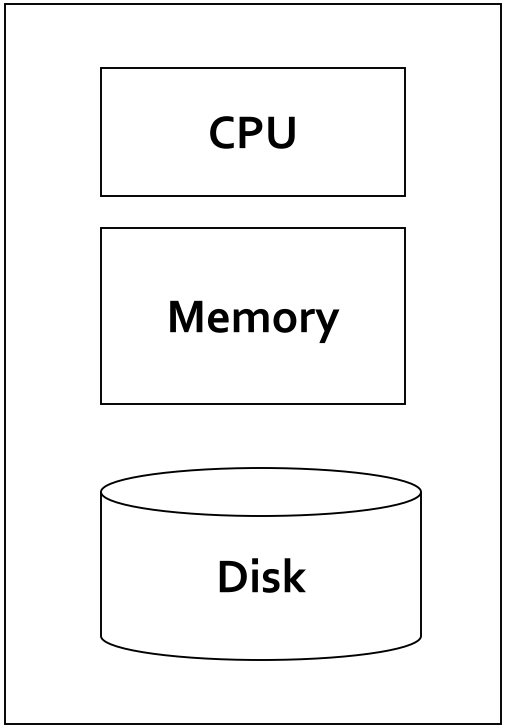
业界趋势: Google Example

- 20+ billion web pages x 20KB = 400+ TB
- 1 computer reads 30-35 MB/sec from disk
  - ~4 months to read the web
- ~1,000 hard drives to store the web
- Takes even more to **do** something useful with the data!
- **Today, a standard architecture for such problems is emerging:**
  - Cluster of commodity Linux nodes
  - Commodity network (ethernet) to connect them

在分布式算力架构上的挑战

- Much of the course will be devoted to **large scale computing** for **data mining**
- **Challenges:**
  - How to distribute computation?
  - Distributed/parallel programming is hard

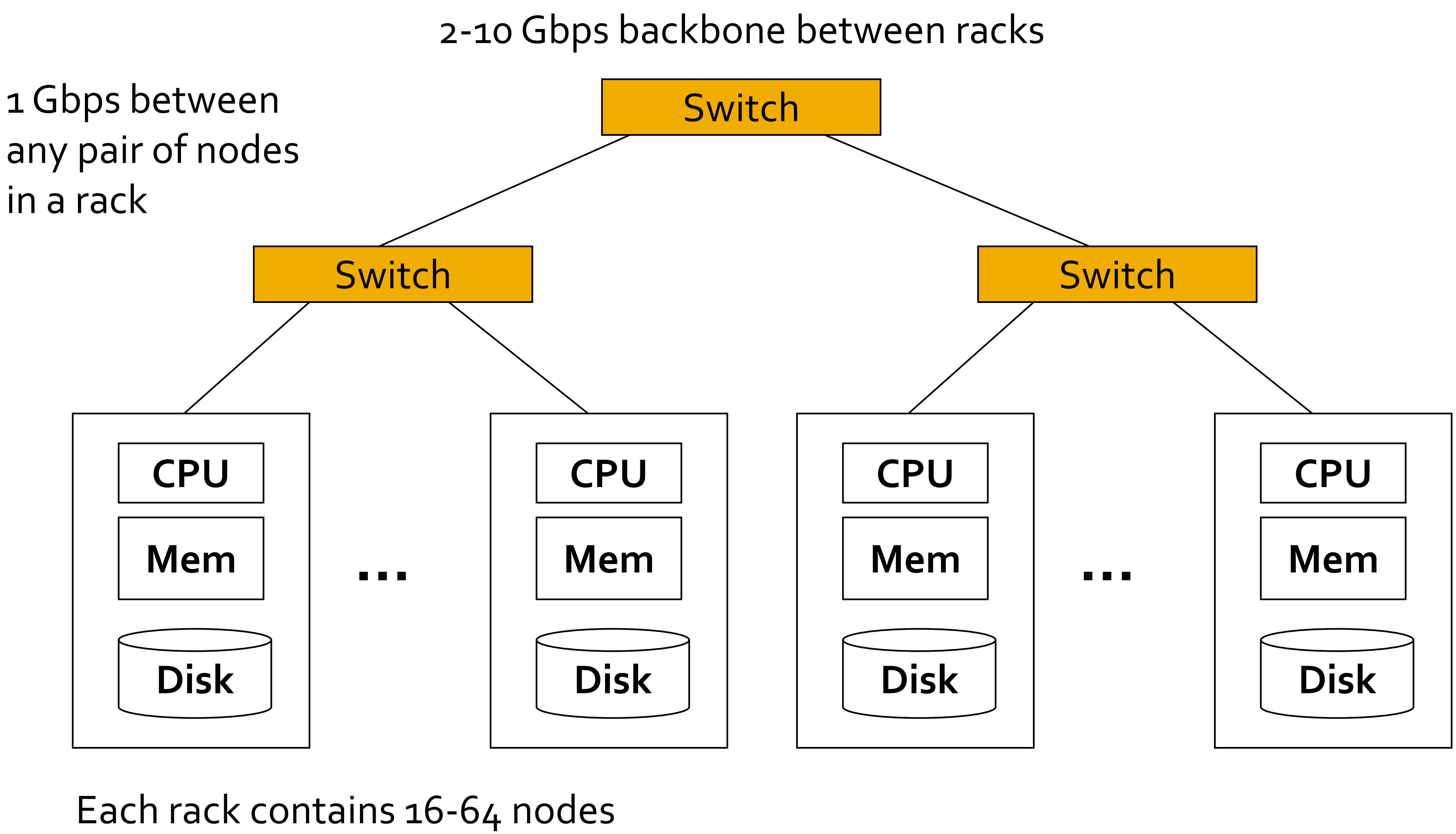
Single Node Architecture



Machine Learning, Statistics

“Classical” Data Mining

Cluster Architecture



In 2011 it was guestimated that Google had 1M machines, http://bit.ly/Shh0RO

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!

Map-reduce : Google's internal implementation

- **Map-reduce** addresses all of the above
  - Google's computational/data manipulation model
  - Elegant way to work with big data

Hadoop 开源生态系统

HDFS：Hadoop Distributed File System，分布式文件系统。有着高容错性（fault-tolerant）的特点，并且设计用来部署在低廉的（low-cost）硬件上。而且它提供高吞吐量（high throughput）来访问应用程序的数据，适合那些有着超大数据集（large data set）的应用程序

MapReduce：一种编程模型，用于大规模数据集（大于1TB）并行运算。分为两部分，"Map（映射）"和"Reduce（化简）"

Oozie：开源工作流引擎。用于管理和协调运行在Hadoop平台上（包括：HDFS、Pig和MapReduce）的Jobs

Hadoop 开源生态系统

Mahout：提供可扩展的机器学习领域经典算法实现，包括聚类、分类、推荐过滤、频繁子项挖掘，旨在帮助开发人员更加方便快捷地创建智能应用程序

HBase：是一个基于HDFS的分布式的、面向列的开源数据库

Zookeeper：针对大型分布式系统的可靠调度系统，功能包括：配置维护、名字服务、分布式同步、组服务等

Hive：基于Hadoop的数据仓库工具，可以将结构化的数据文件映射为一张数据库表，并提供完整的sql查询功能，可以将sql语句转换为MapReduce任务进行运行

Sqoop：一个用来将Hadoop和关系型数据库中的数据相互转移的工具，可以将关系型数据库（例如 MySQL 等）中的数据导进到HDFS中，也可以将HDFS的数据导进到关系型数据库

Flume：一个高可用的，高可靠的，分布式的海量日志采集、聚合和传输的系统，支持在日志系统中定制各类数据发送方，用于收集数据；同时，提供对数据进行简单处理，

outline

Lecture 2.1 large scale computing

Lecture 2.2 Distributed File Systems

Lecture 2.3 Programming Model

Lecture 2.4 Problems Suited for Map-Reduce

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

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

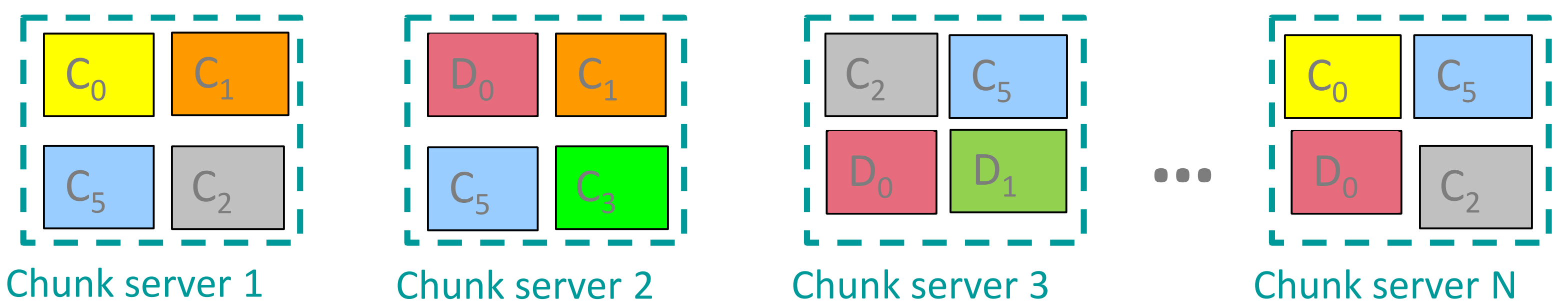
Distributed File System

- Chunk servers
  - File is split into contiguous chunks
  - Typically each chunk is 64-256MB
  - 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

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

大数据存储-HDFS

NameNode

- 负责管理文件系统名称空间和控制外部客户机的访问
- 决定是否将文件映射到 DataNode 上的哪个复制块上

DataNode

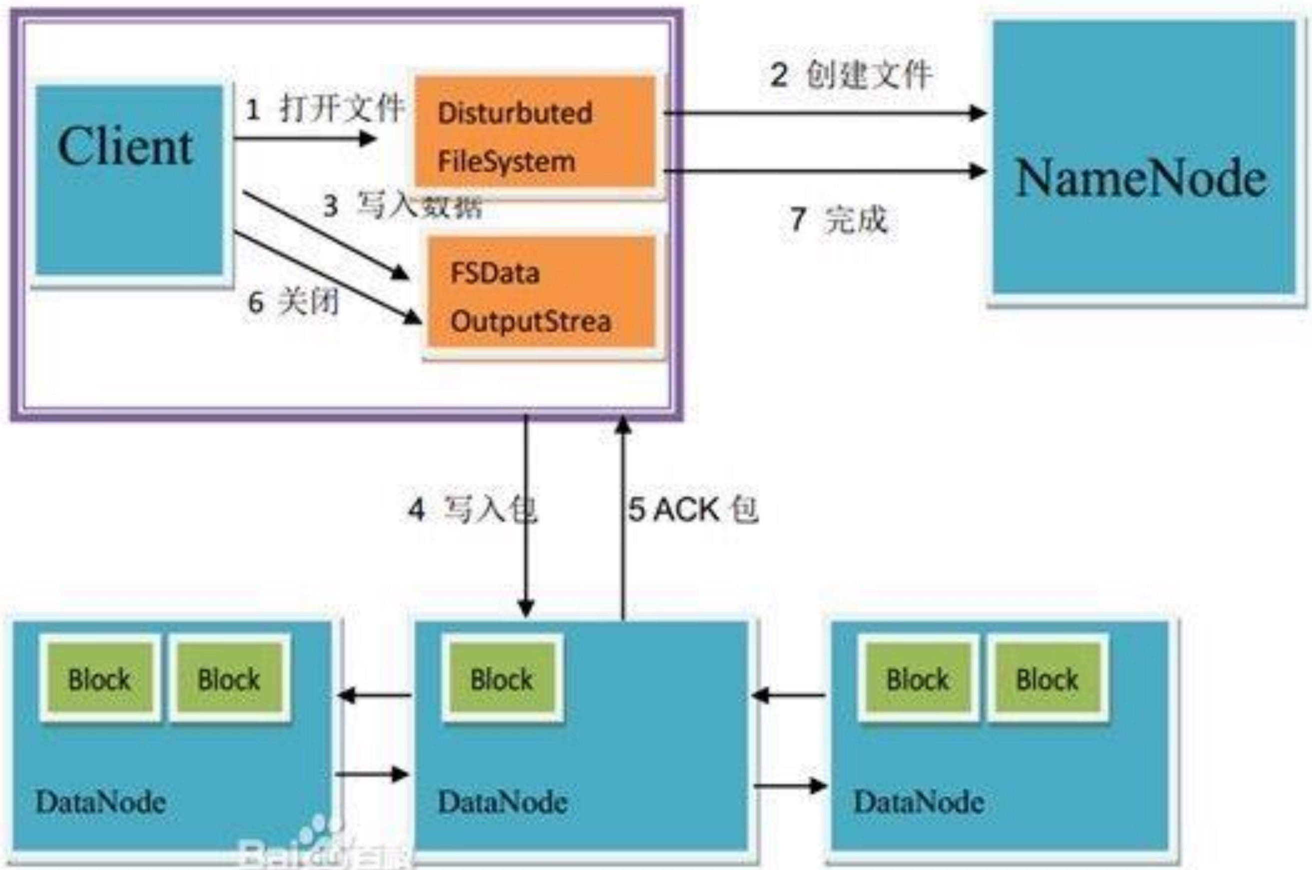
- 响应来自 HDFS 客户机的读写请求
- 响应来自NameNode的创建、删除和复制块的命令
- 依赖来自每个 DataNode 的定期心跳（heartbeat）消息验证块映射文件系统和元数据

可扩展：HDFS可以扩展到几百台甚至几千台的集群规模

低成本：自动容错、自动负载均衡机制使其可以构建在普通PC机之上。另外，线性扩展能力也使得增加、减少节点非常方便，可以实现自动运维

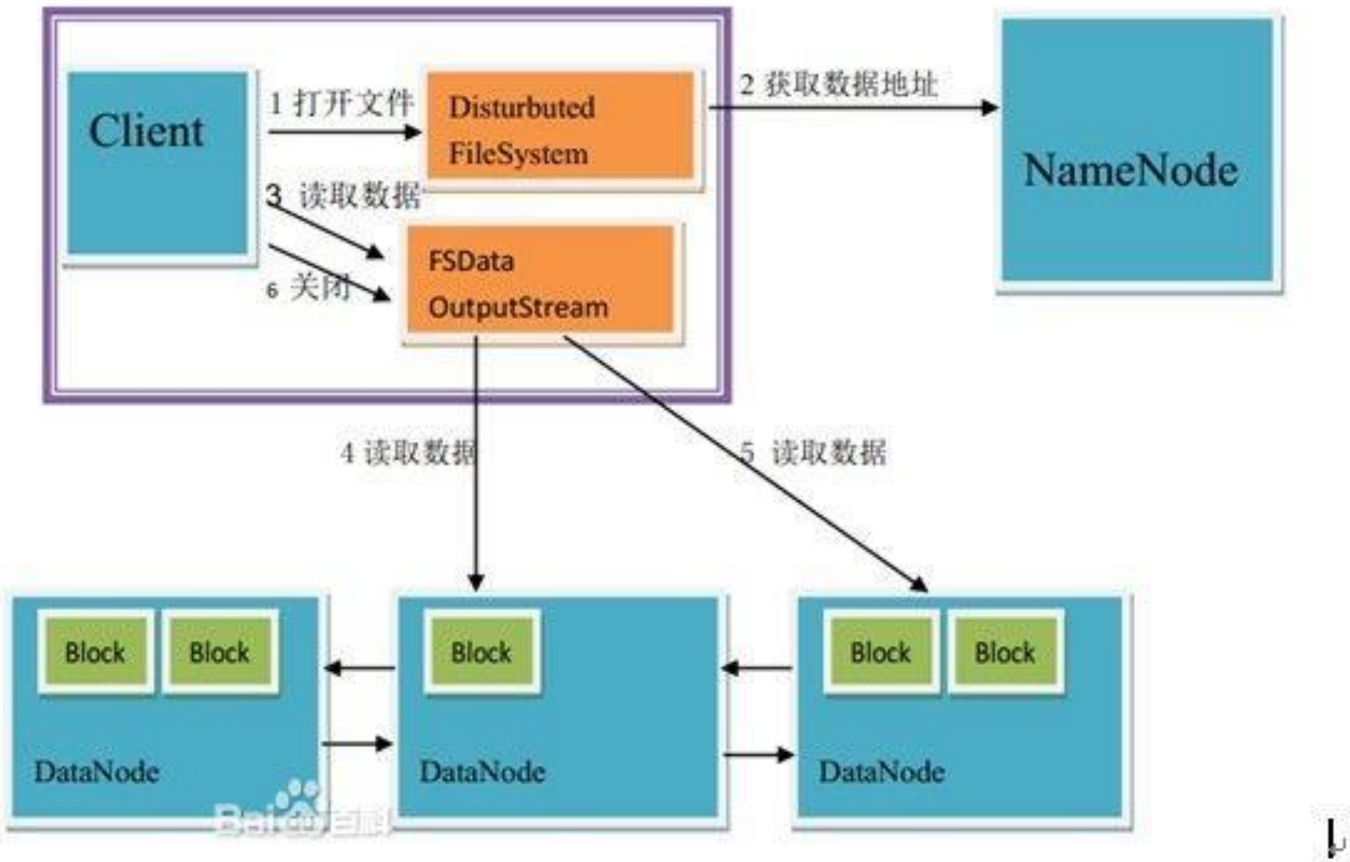
高容错：文件可以通过不同策略备份到不同数据节点、机架

大数据存储-HDFS.写入



HDFS 写入数据流程图

大数据存储-HDFS.读取



HDFS 读取数据流程图

outline

Lecture 2.1 large scale computing

Lecture 2.2 Distributed File Systems

Lecture 2.3 Programming Model

Lecture 2.4 Problems Suited for Map-Reduce

Programming Model: MapReduce

Warm-up 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

Task: Word Count

Case 1:

- File too large for memory, but all <word, count> pairs fit in memory

Case 2:

- Count occurrences of words:
  - **`words(doc.txt) | sort | uniq -c`**
    - where **`words`** takes a file and outputs the words in it, one per a line
- Case 2 captures the essence of **MapReduce**
  - Great thing is that it is naturally parallelizable

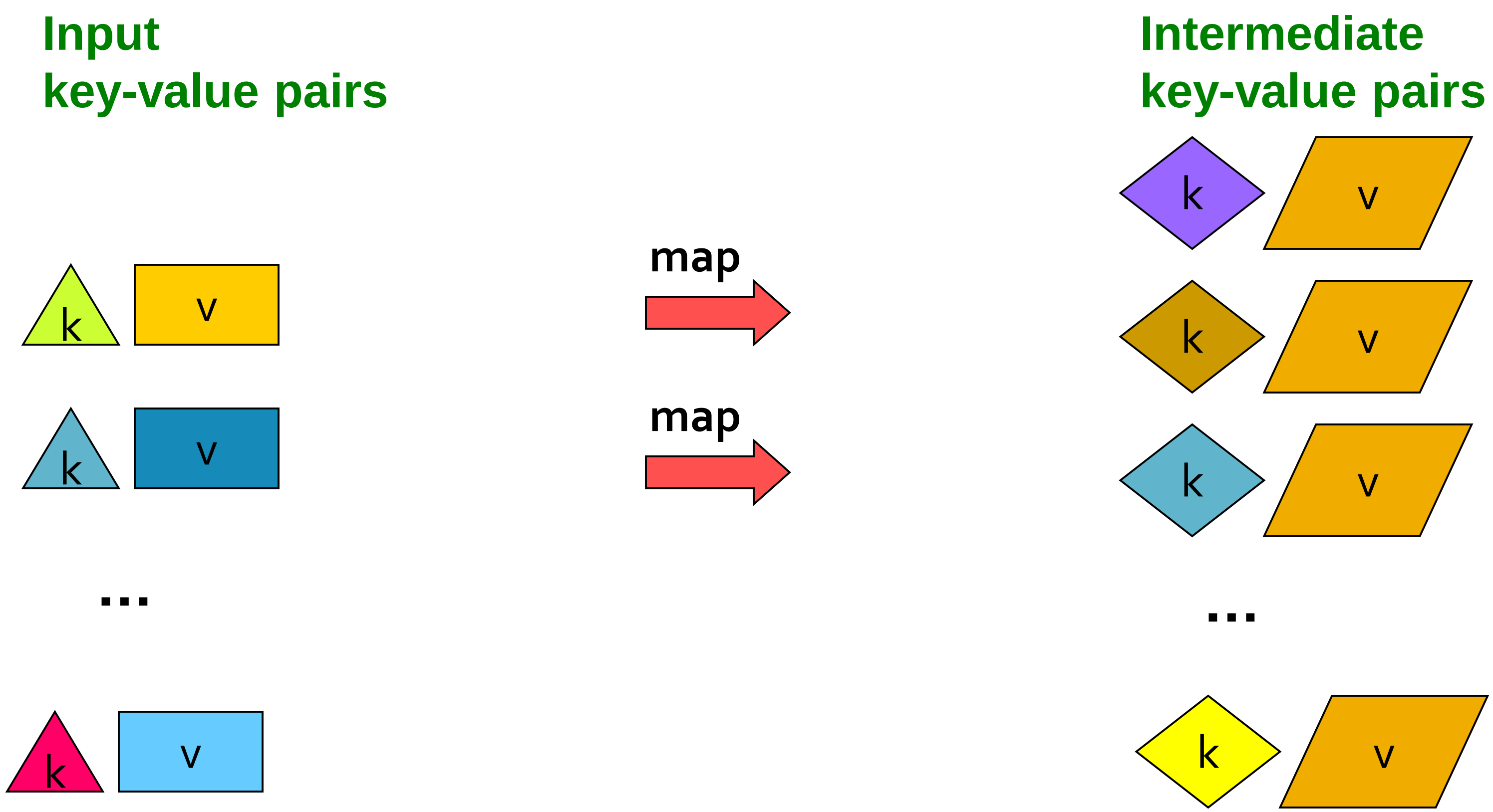
MapReduce: Overview

- Sequentially read a lot of data
- **Map:**
  - Extract something you care about
- **Group by key:** Sort and Shuffle

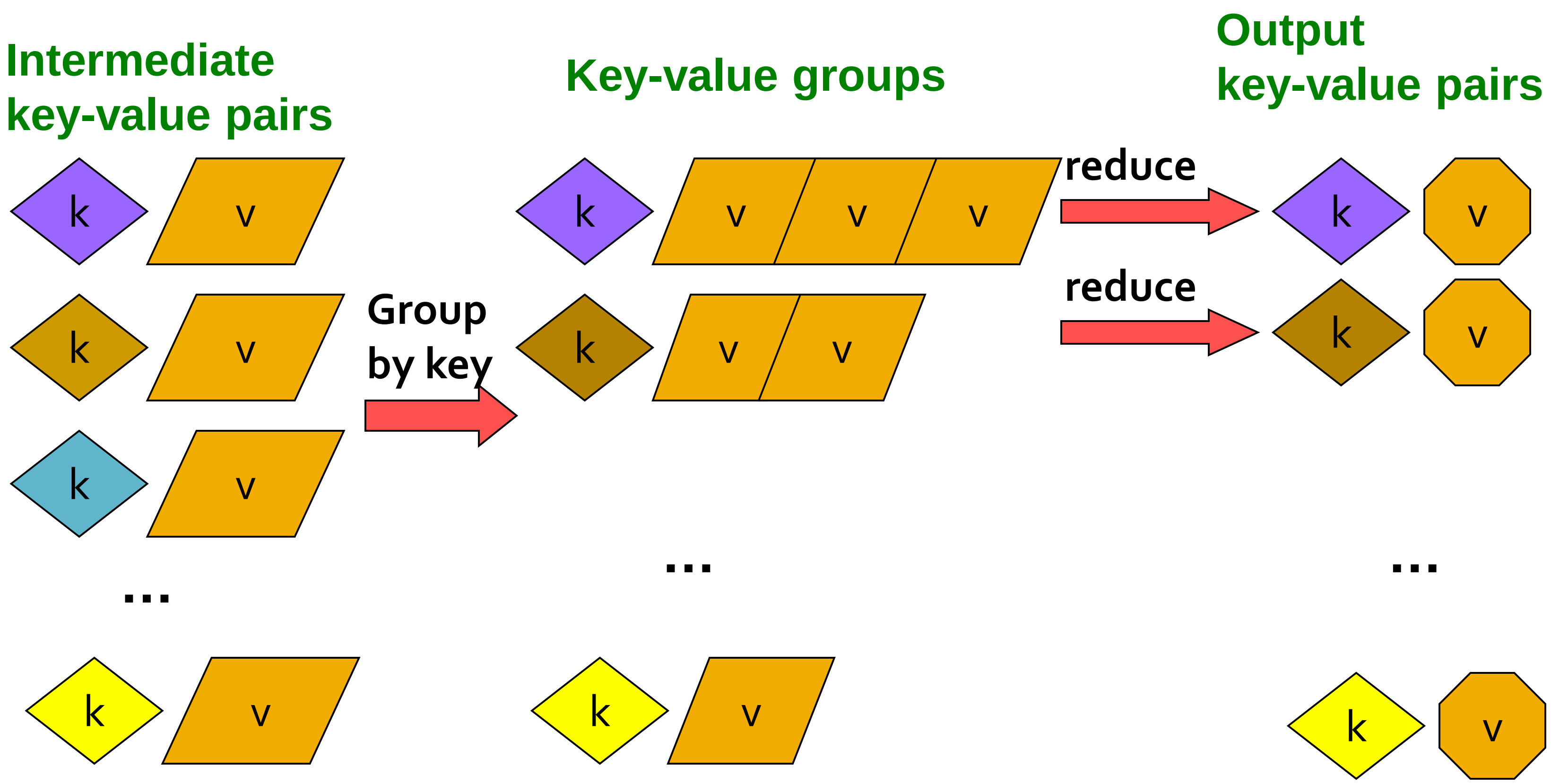
- **Reduce:**
  - Aggregate, summarize, filter or transform
- Write the result

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

MapReduce: The Map Step



MapReduce: The Reduce Step



More Specifically

- **Input: a set of key-value pairs**
- Programmer specifies two methods:
  - **Map(k, v)** $\rightarrow$ <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’>*)** $\rightarrow$ <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’*

MapReduce: Word Counting

Provided by the programmer

MAP:
Read input and produces a set of key-value pairs

Group by key:
Collect all pairs with same key

Provided by the programmer

Reduce:
Collect all values belonging to the key and output

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/mache partnership. '"The work we're doing now -- the robotics we're doing -- is what we're going to need ……………………..

Big document

(The, 1)
(crew, 1)
(of, 1)
(the, 1)
(space, 1)
(shuttle, 1)
(Endeavor, 1)
(recently, 1)
….

(key, value)

(crew, 1)
(crew, 1)
(space, 1)
(the, 1)
(the, 1)
(the, 1)
(shuttle, 1)
(recently, 1)
…

(key, value)

(crew, 2)
(space, 1)
(the, 3)
(shuttle, 1)
(recently, 1)
…

(key, value)

Only sequential reads

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:

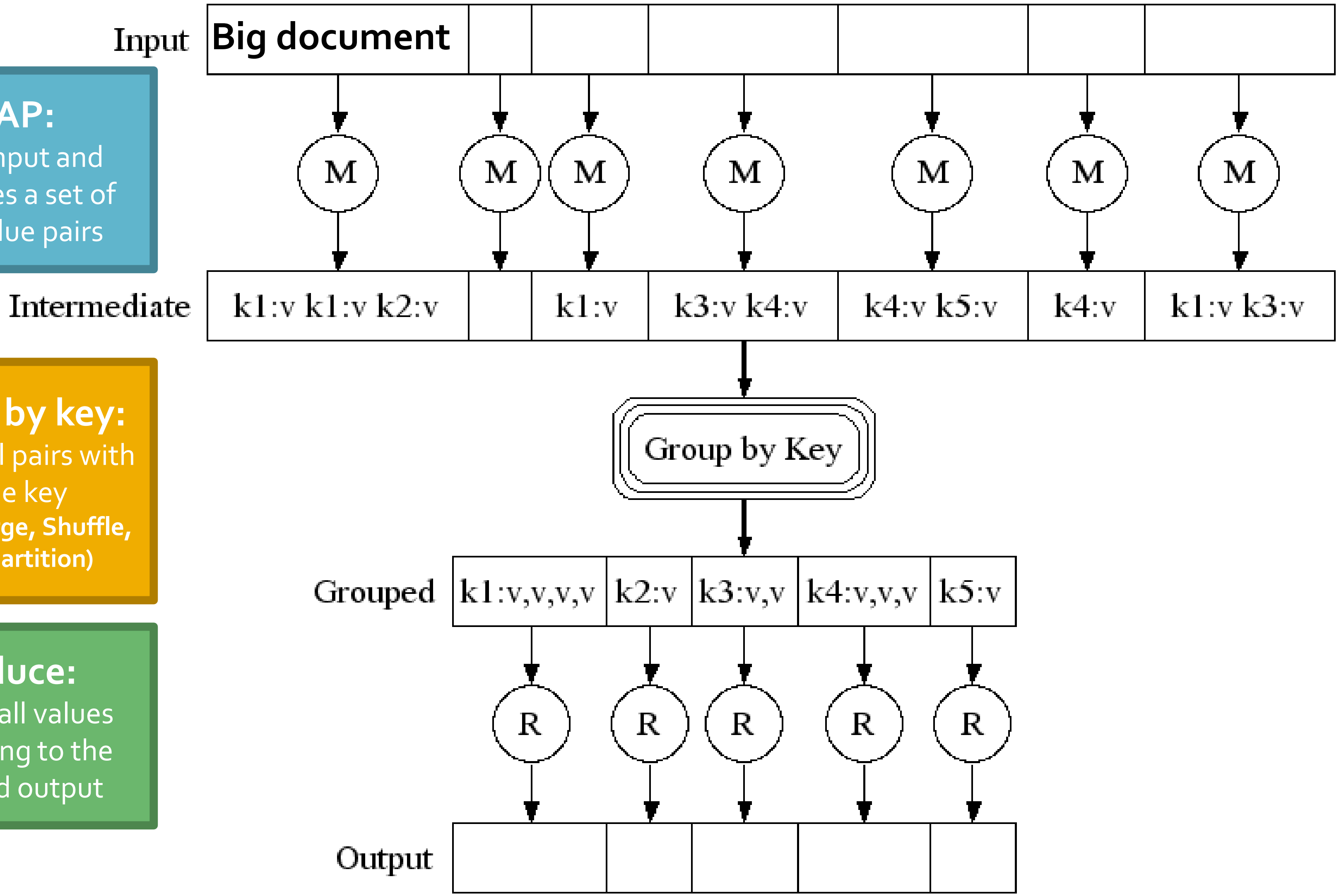
- Partitioning the input data
- Scheduling the program's execution across a set of machines
- Performing the **group by key** step
- Handling machine failures
- Managing required inter-machine communication

Map-Reduce: A diagram

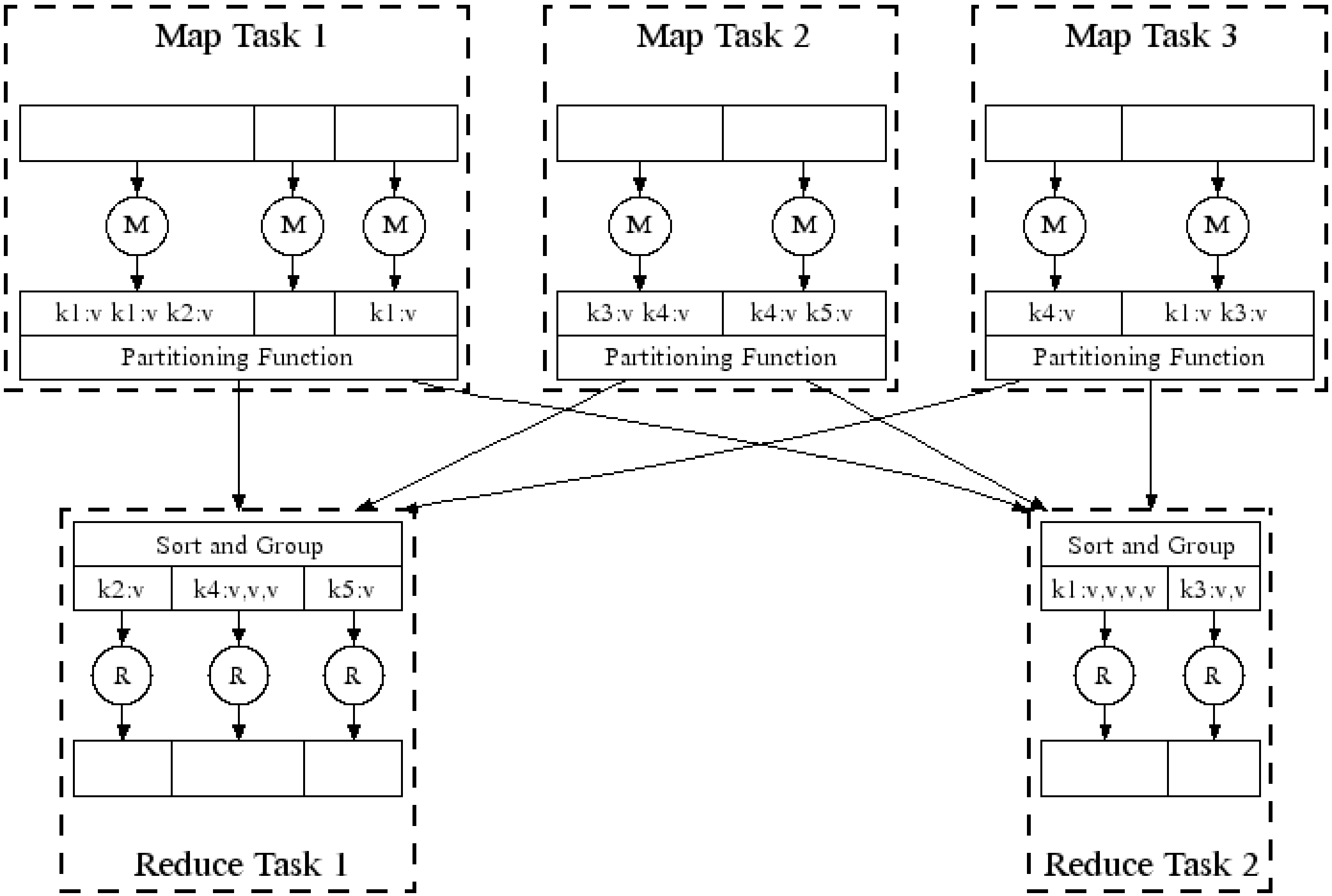
MAP:
Read input and produces a set of key-value pairs

Group by key:
Collect all pairs with same key
(Hash merge, Shuffle, Sort, Partition)

Reduce:
Collect all values belonging to the key and output



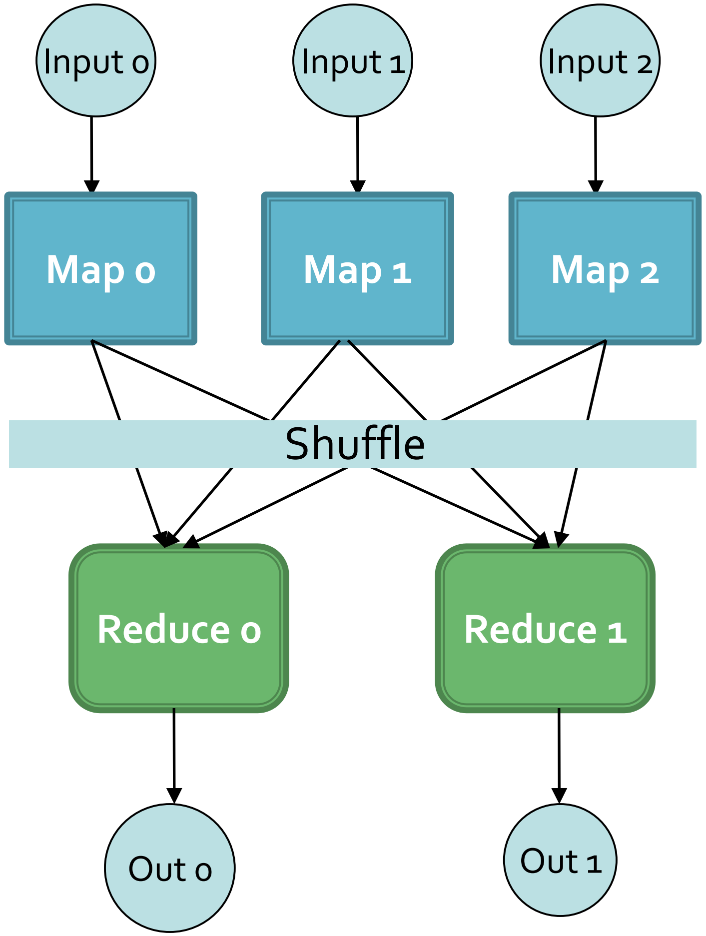
Map-Reduce: In Parallel



All phases are distributed with many tasks doing the work

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

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**

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

Dealing with Failures

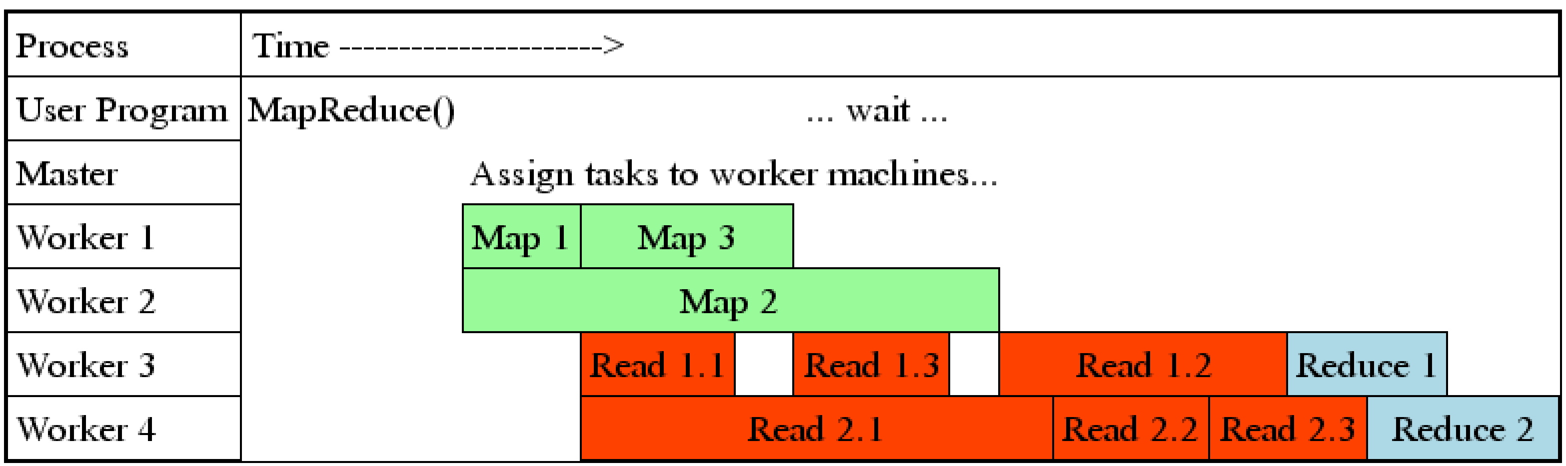
- **Map worker failure**
  - Map tasks completed or in-progress at worker are reset to idle
  - Reduce workers are notified when task is rescheduled on another worker
- **Reduce worker failure**
  - Only in-progress tasks are reset to idle
  - Reduce task is restarted
- **Master failure**
  - MapReduce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

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

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

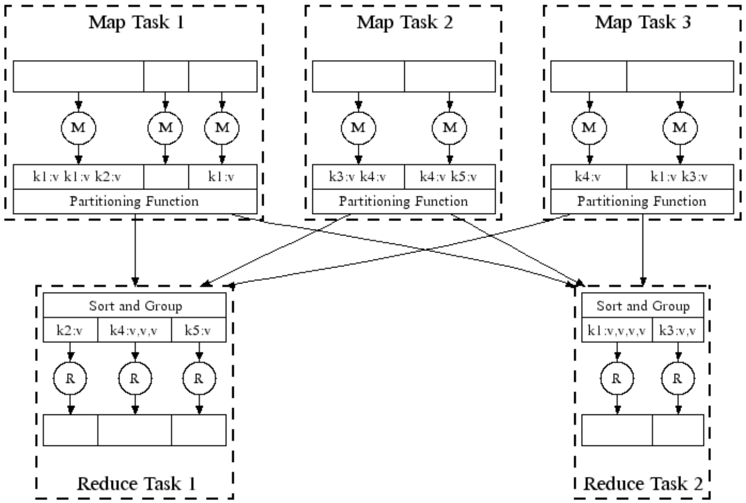


Refinements: Backup Tasks

- 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

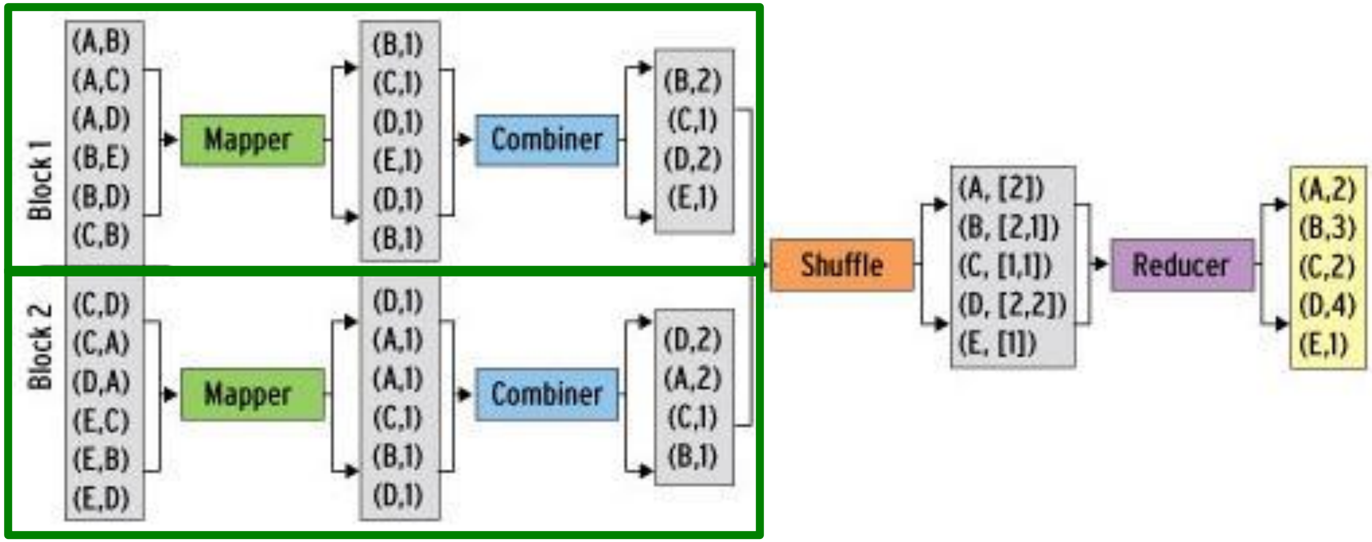
Refinement: Combiners

- Often a Map task will produce many pairs of the form *(k,v_1), (k,v_2), …* 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



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!

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

outline

Lecture 2.1 large scale computing

Lecture 2.2 Distributed File Systems

Lecture 2.3 Programming Model

Lecture 2.4 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

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

Example: Join By Map-Reduce

- **Compute the natural join *R(A,B)* ⋈ *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

⋈

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

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)*.

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)

Example: Cost Measures

- **For a map-reduce algorithm:**
  - **Communication cost =** input file size + 2 $\times$ (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

What Cost Measures Mean

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

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

第2章作业

- 编程分析题：英文书2.2节练习题，Exercise 2.2.1
  - 要求：自己找数据进行实践，写出实验的过程报告，篇幅不限，记录过程和代码。并对题目问题进行分析
  - 时间：9.14日前

Pointers and Further Reading

Implementations

- Google
  - Not available outside Google
- **Hadoop**
  - An open-source implementation in Java
  - Uses HDFS for stable storage
  - Download: http://lucene.apache.org/hadoop/
- Aster Data
  - Cluster-optimized SQL Database that also implements MapReduce

Cloud Computing

- Ability to rent computing by the hour
  - Additional services e.g., persistent storage
- Amazon's "Elastic Compute Cloud" (EC2)
- Aster Data and Hadoop can both be run on EC2
- **For CS341 (offered next quarter) Amazon will provide free access for the class**

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

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/

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

Further Reading

- Programming model inspired by functional language primitives
- Partitioning/shuffling similar to many large-scale sorting systems
  - NOW-Sort ['97]
- Re-execution for fault tolerance
  - BAD-FS ['04] and TACC ['97]
- Locality optimization has parallels with Active Disks/Diamond work
  - Active Disks ['01], Diamond ['04]
- Backup tasks similar to Eager Scheduling in Charlotte system
  - Charlotte ['96]
- Dynamic load balancing solves similar problem as River's distributed queues
  - River ['99]