大数据并行编程模型：MapReduce

MapReduce思想来源于LISP（Scheme）

- (map ***f list [list$_2$ list$_3$ …]***)
- (map square '(1 2 3 4))
  - (1 4 9 16)
- (reduce ***f id list***)
- (reduce + 0 '(1 4 9 16))
  - (+ 16 (+ 9 (+ 4 (+ 1 0))))
  - 30
- (reduce + 0 (map square (map – l1 l2))))

大数据并行编程模型：MapReduce

- MapReduce是一种高度可扩展的编程模型
  - 通过在大量廉价计算结点上并行执行来处理大规模数据（>1TB）
- 1994年从Google提出并流行，今天已经在许多如Apache Hadoop开源项目中实现
- MapReduce成为大数据处理主要手段
  - 高度可扩展性
  - 容错性
  - 简单性
  - 独立于编程语言以及数据存储系统

MapReduce

- 这部分内容:
  - 基本概念
  - 近观MapReduce 数据流
  - 辅助功能
  - MapReduce的调度及容错
  - 与现有技术及模型比较

4

Problem Scope

- MapReduce 数据处理用的编程模型
  - 适合于数据密集型计算
- MapReduce的能力表现在可扩展至成百上千的计算机，而且每个都有几个核
- 计算量有多大?
  - TB或PB级Web数据
  - 输入数据在单个计算机的硬盘放不下
  - 因此，需要分布式文件系统(如GFS, HDFS

5

常规集群

- MapReduce 被设计为通过连在一起的许多常规计算机以并行方式处理大规模数据
- 理论上 1000-CPU 机器价值非常昂贵，远大于 1000个单CPU 或 250个四核机器
- MapReduce 将更小且合理价格机器绑在一起成为一个成本有效的常规集群

6

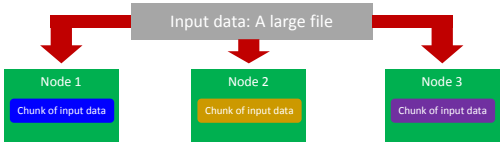
Isolated Tasks

- MapReduce 划分工作负载为多个独立任务并调度到集群节点
- 以隔离方式做完每一个任务从而最终完成计算工作
- 任务完成的通信量主要受限于可扩展因素
  - The communication overhead required to keep the data on the nodes synchronized at all times would prevent the model from performing reliably and efficiently at large scale

7

数据分布

- In a MapReduce cluster, data is distributed to all the nodes of the cluster as it is being loaded in
- An underlying distributed file systems (e.g., GFS) splits large data files into chunks which are managed by different nodes in the cluster

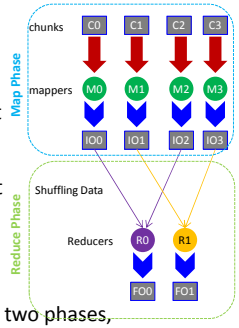


- Even though the file chunks are distributed across several machines, they form a single namesapce

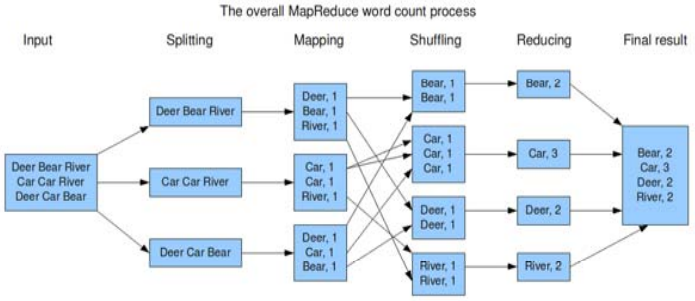
8

MapReduce概览

- In MapReduce, chunks are processed in isolation by tasks called Mappers
- The outputs from the mappers are denoted as intermediate outputs (IOs) and are brought into a second set of tasks called Reducers
- The process of bringing together IOs into a set of Reducers is known as shuffling process
- The Reducers produce the final outputs (FOs)
- Overall, MapReduce breaks the data flow into two phases, map phase and reduce phase



例：统计词频



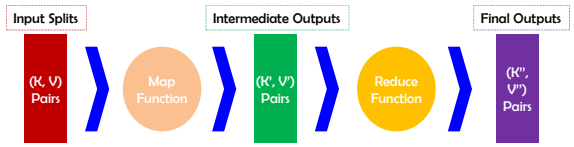
function map(name, document)
for each word **in** document
emit (word, 1)

function reduce (word, List partialCounts)
sum = 0
for each pc **in** partialCounts
sum += pc
emit (word, sum)

11

键与值

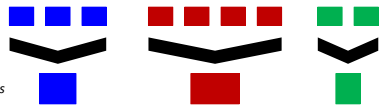
- The programmer in MapReduce has to specify two functions, the map function and the reduce function that implement the Mapper and the Reducer in a MapReduce program
- In MapReduce data elements are always structured as key-value (i.e., (K, V)) pairs
- The map and reduce functions receive and emit (K, V) pairs



Partitions

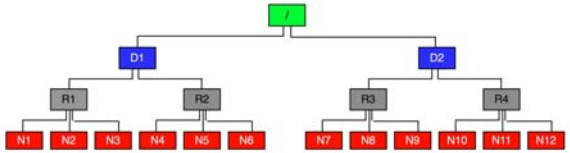
- In MapReduce, intermediate output values are not usually reduced together
- *All values with the same key are presented to a single Reducer together*
- More specifically, a different subset of intermediate key space is assigned to each Reducer : hash(k') mod R
- These subsets are known as *partitions*

Different colors represent different keys (potentially) from different Mappers



Partitions are the input to Reducers

MapReduce的网络拓扑



- MapReduce assumes a tree style network topology
- Nodes are spread over different racks embraced in one or many data centers
- 突出的一点是结点间带宽与它们在网络拓扑中的相对位置有关
  - For example, nodes that are on the same rack will have higher bandwidth between them as opposed to nodes that are off-rack

Hadoop

- Since its debut on the computing stage, MapReduce has frequently been associated with *Hadoop*
- Hadoop is an open source implementation of MapReduce and is currently enjoying wide popularity
- Hadoop presents MapReduce as an analytics engine and under the hood uses a distributed storage layer referred to as Hadoop Distributed File System (*HDFS*)
- HDFS mimics Google File System (*GFS*)

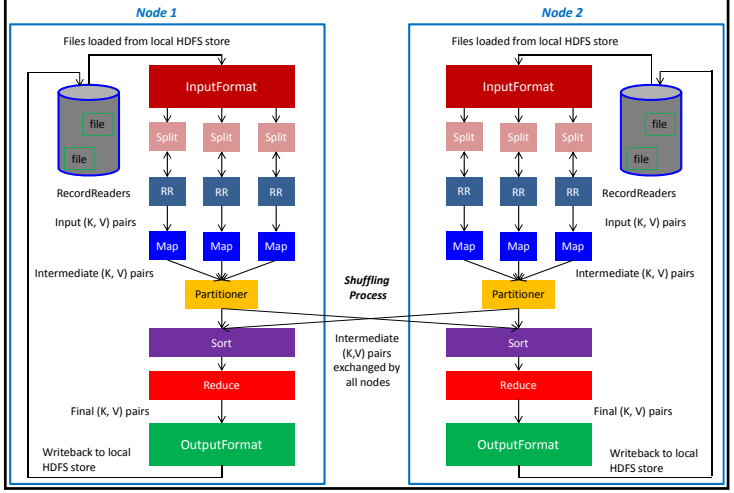
15

MapReduce

- In this part, the following concepts of MapReduce will be described:
  - Basics
  - MapReduce 数据流
  - Additional functionality
  - Scheduling and fault-tolerance in MapReduce
  - Comparison with existing techniques and models

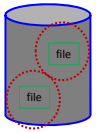
14

Hadoop MapReduce: A Closer Look



Input Files

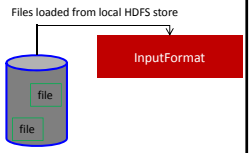
- *Input files* are where the data for a MapReduce task is initially stored
- The input files typically reside in a distributed file system (e.g. HDFS)
- The format of input files is arbitrary
  - Line-based log files
  - Binary files
  - Multi-line input records
  - Or something else entirely



17

InputFormat

- How the input files are split up and read is defined by the *InputFormat*
- InputFormat is a class that does the following:
- Selects the files that should be used for input
- Defines the *InputSplits* that break a file
- Provides a factory for *RecordReader* objects that read the file



18

InputFormat Types

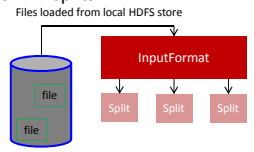
- Several InputFormats are provided with Hadoop:

InputFormat	Description	Key	Value
TextInputFormat	**Default format; reads lines of text files**	**The byte offset of the line**	**The line contents**
KeyValueInputFormat	**Parses lines into (K, V) pairs**	**Everything up to the first tab character**	**The remainder of the line**
SequenceFileInputFormat	A Hadoop-specific high-performance binary format	user-defined	user-defined

19

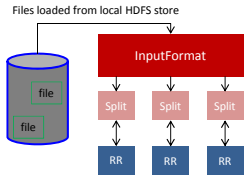
Input Splits

- An *input split* describes a unit of work that comprises a single map task in a MapReduce program
- By default, the InputFormat breaks a file up into 64MB splits
- By dividing the file into splits, we allow several map tasks to operate on a single file in parallel
- If the file is very large, this can improve performance significantly through parallelism
- 每个 map 任务对应于单个输入分块（split）



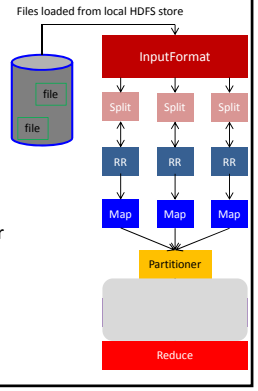
RecordReader

- The input split defines a slice of work but does not describe how to access it
- The *RecordReader* class actually loads data from its source and converts it into (K, V) pairs suitable for reading by Mappers
- The RecordReader is invoked repeatedly on the input until the entire split is consumed
- Each invocation of the RecordReader leads to another call of the map function defined by the programmer



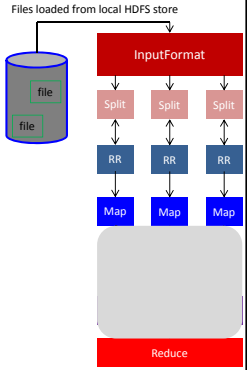
Partitioner

- Each mapper may emit (K, V) pairs to *any* partition
- Therefore, the map nodes must all agree on where to send different pieces of intermediate data
- The *partitioner* class determines which partition a given (K,V) pair will go to
- The default partitioner computes *a hash value* for a given key and assigns it to a partition based on this result



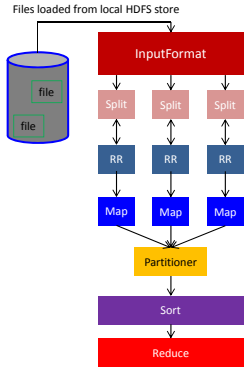
Mapper and Reducer

- The *Mapper* performs the user-defined work of the first phase of the MapReduce program
- A new instance of Mapper is created for each split
- The *Reducer* performs the user-defined work of the second phase of the MapReduce program
- 为每个partition都创建一个新的Reducer实例
- *对于赋给Reducer的partition中的每一个key, Reducer都被调用一次*



Sort

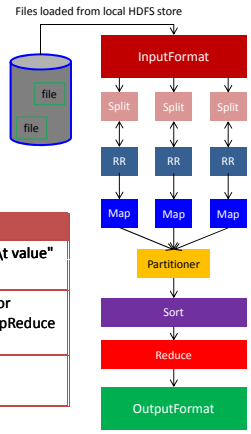
- 每个 Reducer 负责归结与中间键（可多个）关联的那些值
- 单个结点上的中间键（intermediate keys）集合会被自动地排序，然后呈现给 Reducer



OutputFormat

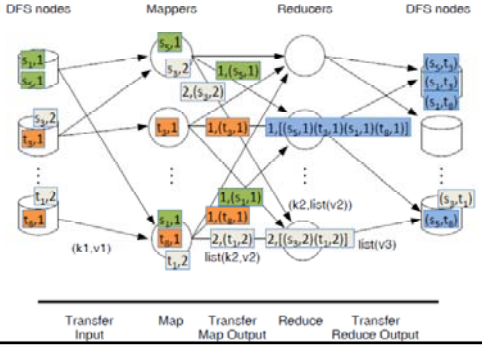
- The *OutputFormat* class defines the way (K,V) pairs produced by Reducers are written to output files
- The instances of OutputFormat provided by Hadoop write to files on the local disk or in HDFS
- Several OutputFormats are provided by Hadoop:

OutputFormat	Description
TextOutputFormat	Default; writes lines in "key \t value" format
SequenceFileOutputFormat	Writes binary files suitable for reading into subsequent MapReduce jobs
NullOutputFormat	Generates no output files



Equi-Join in MapReduce

- Join condition: S.A=T.A
- Map(s) = (s.A, s); Map(t) = (t.A, t)
- Reduce computes Cartesian product of set of S-tuples and set of T-tuples with same key



Example: Equi-Join

- Given two data sets $S=(s_1,s_2,\ldots)$ and $T=(t_1,t_2,\ldots)$ of integers, find all pairs (s_i,t_j) where $s_i.A=t_j.A$
- Can only combine the s_i and t_j in Reduce
  - To ensure that the right tuples end up in the same Reduce invocation, use join attribute A as intermediate key (k2)
  - Intermediate value is actual tuple to be joined
- Map needs to output (s.A, s) for each S-tuple s (similar for T-tuples)

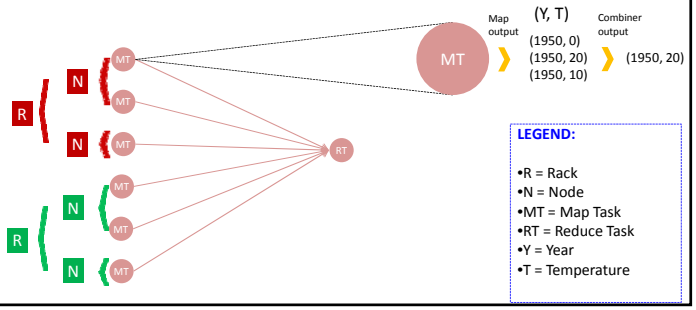
MapReduce

- In this part, the following concepts of MapReduce will be described:
  - Basics
  - A close look at MapReduce data flow
  - 附加功能
  - Scheduling and fault-tolerance in MapReduce
  - Comparison with existing techniques and models

28

Combiner Functions

- MapReduce applications are limited by the bandwidth available on the cluster
- 值得最小化进行洗牌（shuffle）的数据（map和reduce任务间）
- Hadoop allows the user to specify a combiner function (just like the reduce function) to be run on a map output



MapReduce

- In this part, the following concepts of MapReduce will be described:
  - Basics
  - A close look at MapReduce data flow
  - Additional functionality
  - MapReduce中的调度及容错
  - Comparison with existing techniques and models

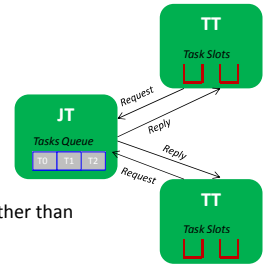
31

Careful With Combiners

- Consider Word Count, but assume we only want words with count > 10
  - Reducer computes total word count, only outputs if greater than 10
  - Combiner = Reducer? No. Combiner should not filter based on its local count!
- Consider computing average of a set of numbers
  - Reducer should output average
  - Combiner has to output (sum, count) pairs to allow correct computation in reducer

Task Scheduling in MapReduce

- MapReduce adopts a *master-slave architecture*
- The master node in MapReduce is referred to as *Job Tracker* (JT)
- Each slave node in MapReduce is referred to as *Task Tracker* (TT)
- MapReduce adopts a *pull scheduling* strategy rather than a *push one*



  - I.e., JT does not push map and reduce tasks to TTs but rather TTs pull them by making pertaining requests

32

Map and Reduce Task Scheduling

- Every TT sends a *heartbeat message* periodically to JT encompassing a request for a map or a reduce task to run

I. Map Task Scheduling:

- JT satisfies requests for map tasks via attempting to schedule mappers in the 邻近 of their input splits (i.e., it considers locality)

II. Reduce Task Scheduling:

- However, JT simply assigns the next yet-to-run reduce task to a requesting TT regardless of TT's network location and its implied effect on the reducer's shuffle time (i.e., it does not consider locality)

33

Job Scheduling in MapReduce

- In MapReduce, an application is represented as a *job*
- A job encompasses multiple map and reduce tasks
- MapReduce in Hadoop comes with a choice of schedulers:
  - The default is the *FIFO scheduler* which schedules jobs in order of submission
  - There is also a multi-user scheduler called the *Fair scheduler* which aims to give every user a fair share of the cluster capacity over time

34

Fault Tolerance in Hadoop

- MapReduce can guide jobs toward a successful completion even when jobs are run on a large cluster where probability of failures increases
- The primary way that MapReduce achieves fault tolerance is through *restarting tasks*
- If a TT fails to communicate with JT for a period of time (by default, 1 minute in Hadoop), JT will assume that TT in question has crashed
  - If the job is still in the map phase, JT asks another TT to re-execute *all Mappers that previously ran at the failed TT*
  - If the job is in the reduce phase, JT asks another TT to re-execute *all Reducers that were in progress on the failed TT*

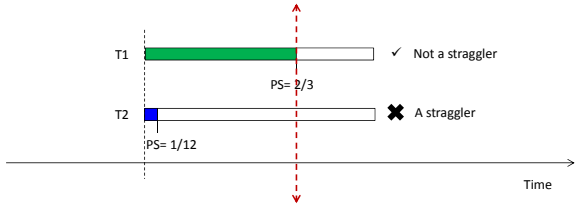
35

Speculative Execution

- A MapReduce job is dominated by the slowest task
- MapReduce attempts to locate slow tasks (*stragglers*) and run redundant (*speculative*) tasks that will optimistically commit before the corresponding stragglers
- This process is known as *speculative execution*
- Only one copy of a straggler is allowed to be speculated
- Whichever copy (among the two copies) of a task commits first, it becomes the definitive copy, and the other copy is killed by JT

Locating Stragglers

- How does Hadoop locate stragglers?
  - Hadoop monitors each task progress using a *progress score* between 0 and 1
  - If a task's progress score ***is less than*** (average – 0.2), and the task has run for at least 1 minute, it is marked as a straggler

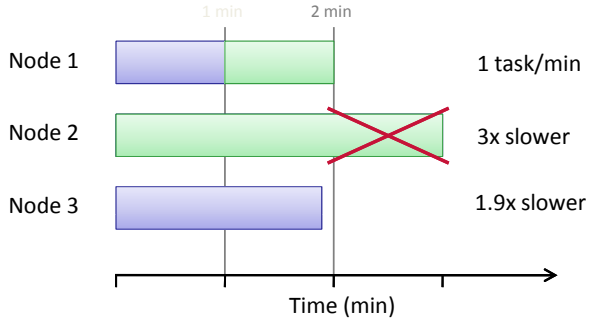


MapReduce

- In this part, the following concepts of MapReduce will be described:
  - Basics
  - A close look at MapReduce data flow
  - Additional functionality
  - Scheduling and fault-tolerance in MapReduce
  - Comparison with existing techniques and models

39

Progress Rate Example



什么使得MapReduce如此独特？

- MapReduce is characterized by:
  1. Its simplified programming model which allows the user to quickly write and test distributed systems
  2. Its efficient and automatic distribution of data and workload across machines
  3. Its flat scalability curve. Specifically, after a Mapreduce program is written and functioning on 10 nodes, very little-if any- work is required for making that same program run on 1000 nodes

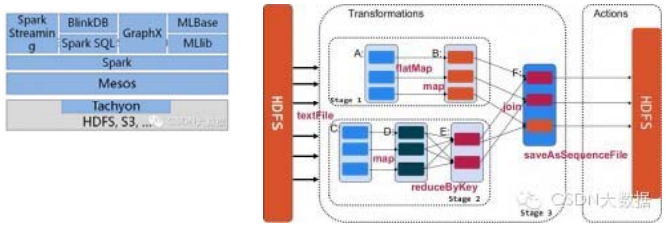
40

与传统编程模型对比

Aspect	Shared Memory	Message Passing	MapReduce
Communication	Implicit (via loads/stores)	Explicit Messages	Limited and Implicit
Synchronization	Explicit	Implicit (via messages)	Immutable (K, V) Pairs
Hardware Support	Typically Required	None	None
Development Effort	Lower	Higher	Lowest
Tuning Effort	Higher	Lower	Lowest

41

内存计算架构Spark



RDD是Spark的最基本抽象这对于迭代运算比较常见的机器学习算法,交互式数据挖掘来说，效率提升比较大转换（Transformation）和操作（Action）这两大类

Spark常见存储数据的格式是Key-Value，也就是Hadoop标准的Sequence File，但同时也听说支持类似Parquet这样的列存格式。Key-Value格式的优点在于灵活，上至数据挖掘算法，明细数据查询，下至复杂SQL处理都能承载，缺点也很明显就是存储空间比较浪费，和类似Parquet列存格式相比更是如此，key-Value格式数据一般是原始数据大小的2倍左右，而列存一般是原始数据的1/3到1/4。

Google内部广泛使用的运算环境

- 用以太网交换机连接、由普通PC机组成的大型集群
  - x86架构、运行Linux操作系统、双处理器、2-4GB内存
  - 普通的网络硬件设备，每个机器的带宽为百兆或者千兆，但是远小于网络的平均带宽的一半
  - 集群中包含成百上千的机器，因此，机器故障是常态
  - 存储为廉价的内置IDE硬盘。一个内部分布式文件系统用来管理存储在这些磁盘上的数据。文件系统通过数据复制来在不可靠的硬件上保证数据的可靠性和有效性
  - 用户提交作业给调度系统。每个作业都包含一系列的任务，调度系统将这些任务调度到集群中多台可用的机器上执行