Assume 100 Billion Web Pages

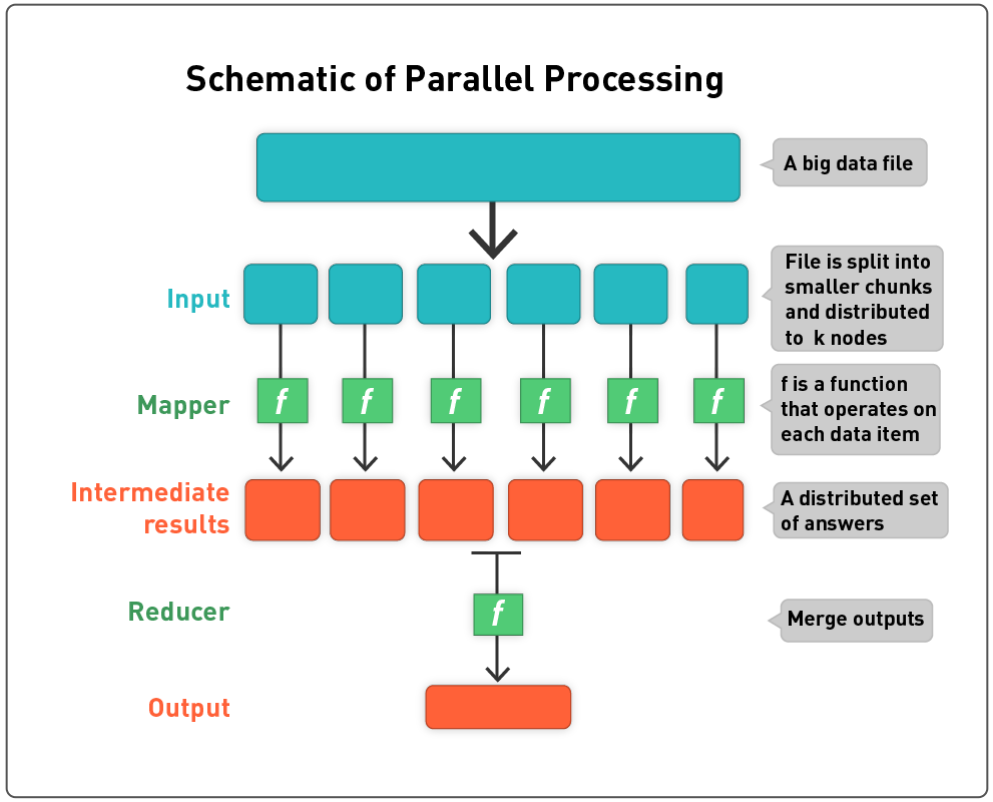
1. How to store them?
    - 10,000 per page $10^{11} = 10^{15}$ bytes (1 petabyte of raw data)
2. How can we scan them?
3. How to do something useful with them?
    - Document frequency for a term?
    - Extract out-links of page and say just count the number of in-links for a Web page

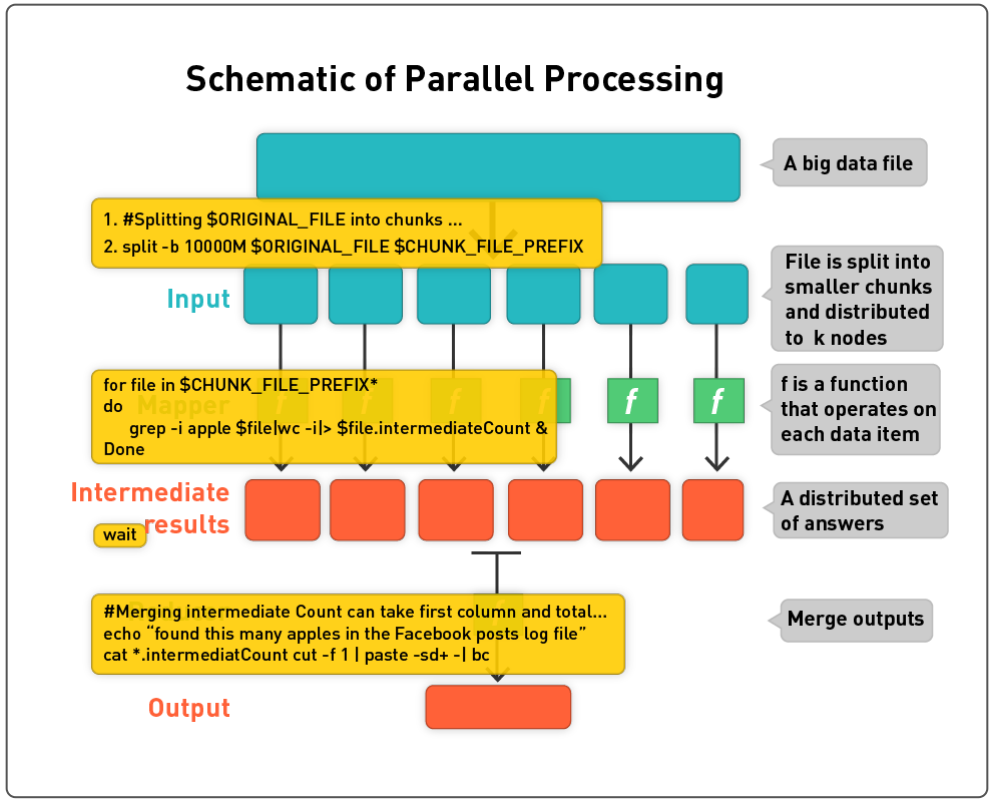
Why Parallelism?

1. Data size is increasing
    - Single-node architecture is reaching its limit
        - Scan 1,000 TB on one node at 100 MB/s = 120 days
    - Store that amount of data?
2. Growing demand to leverage data to do useful tasks
3. Parallelism: Divide and conquer (D&C)
4. Standard or commodity and affordable architecture emerging
    - Cluster of commodity Linux nodes (CPU, memory, and disk)
    - Gigabit Ethernet interconnect

Design Goals for Parallelism

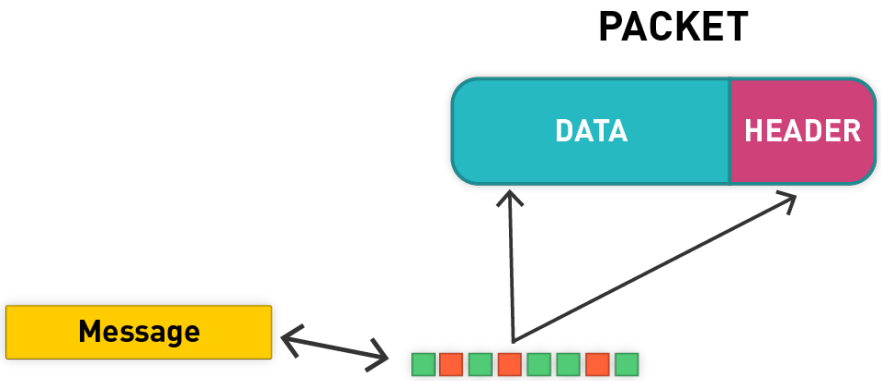
1. Scalability to large data volumes
    - Scan 1,000 TB (1PB) on one node at 50 MB/s = 120 days
    - Scan on 10,000-node cluster = 16 minutes!
2. Cost efficiency:
    - Commodity nodes (cheap but unreliable)
    - Commodity network
    - Automatic fault tolerance (fewer admins)
    - Easy to use (fewer programmers)

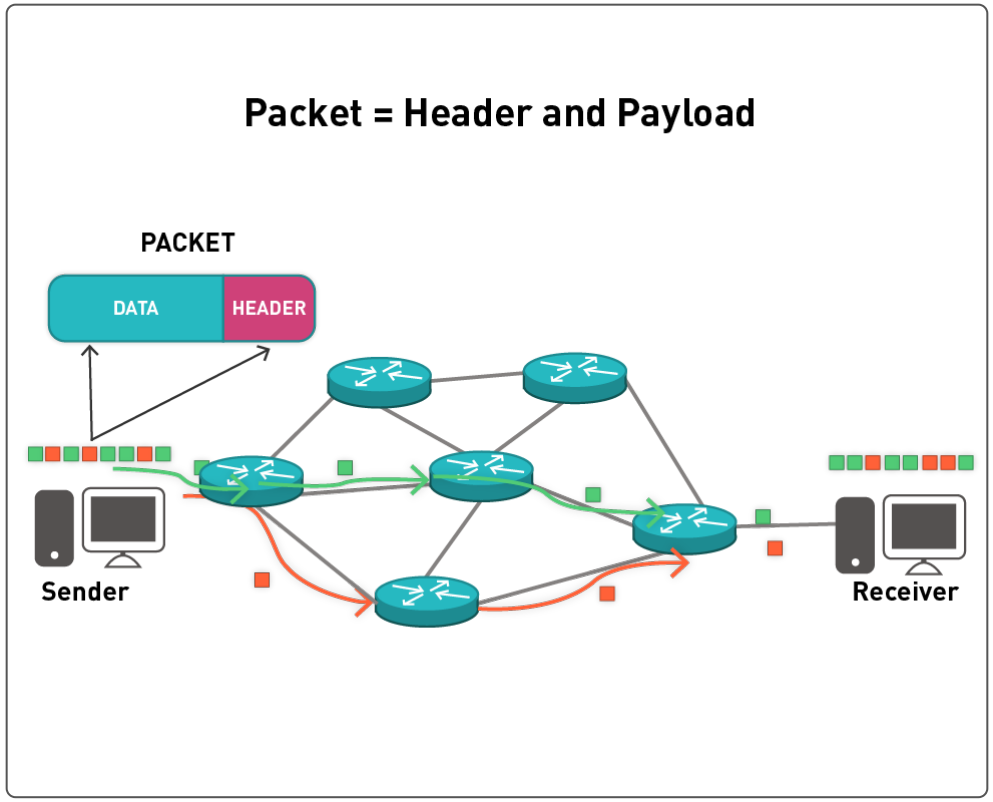

Schematic of Parallel Processing
A big data file
Input
File is split into smaller chunks and distributed to k nodes
Mapper
f
f is a function that operates on each data item
Intermediate results
A distributed set of answers
Reducer
Merge outputs
Output


Schematic of Parallel Processing
A big data file
1. #Splitting $ORIGINAL_FILE into chunks ...
2. split -b 10000M $ORIGINAL_FILE $CHUNK_FILE_PREFIX
Input
File is split into smaller chunks and distributed to k nodes
for file in $CHUNK_FILE_PREFIX*
do
grep -i apple $file|wc -i|> $file.intermediateCount &
Done
Mapper
f
f is a function that operates on each data item
Intermediate results
wait
A distributed set of answers
#Merging intermediate Count can take first column and total...
echo "found this many apples in the Facebook posts log file"
cat *.intermediatCount cut -f 1 | paste -sd+ -| bc
Merge outputs
Output

D&C Is Common: Packet Switching

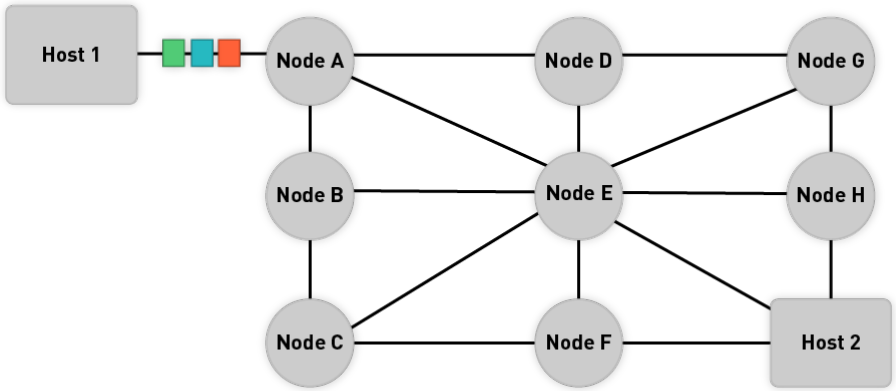
1. Packet switching is a digital networking communications method that transmits messages in chunks or blocks, called **packets**.
2. Packets are transmitted via a medium that may be shared by multiple simultaneous communication sessions.
3. Packet switching increases network efficiency and robustness and enables technological convergence of many applications operating on the same network.




Packet = Header and Payload
PACKET
DATA
HEADER
Sender
Receiver

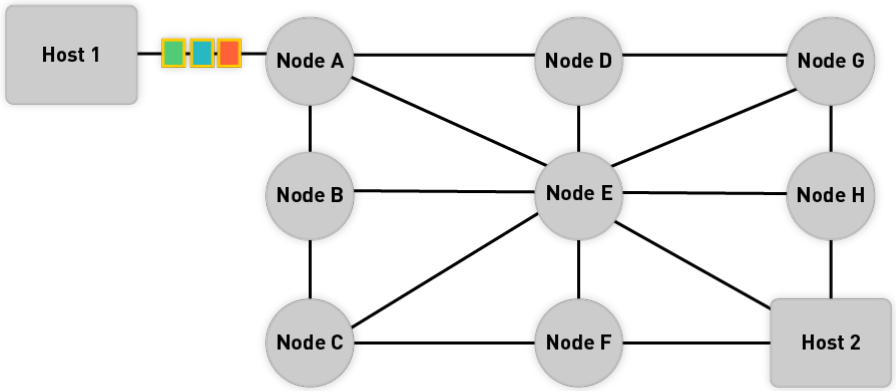
Message Passing: Divide and Conquer/Send

The original message is Green, Blue, Red.



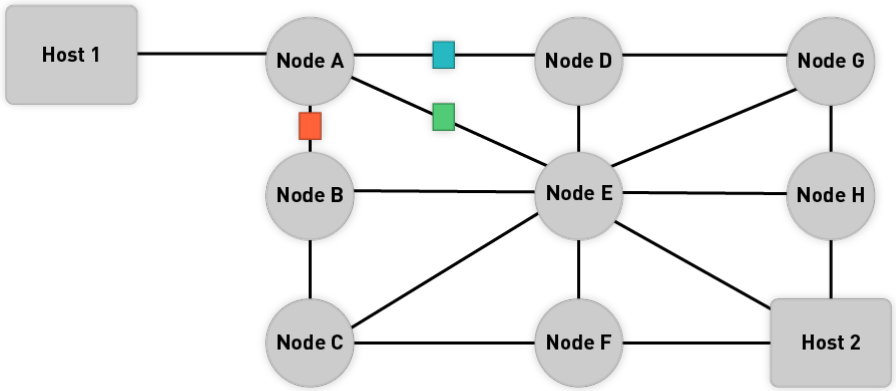
Message Passing: Divide and Conquer/Send

The original message is Green, Blue, Red.



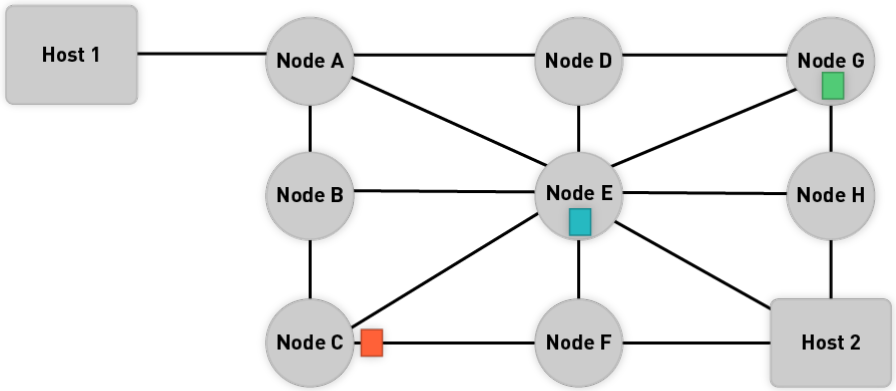
Route Packets in Parallel if Possible

The data packets take different routes to their destinations.



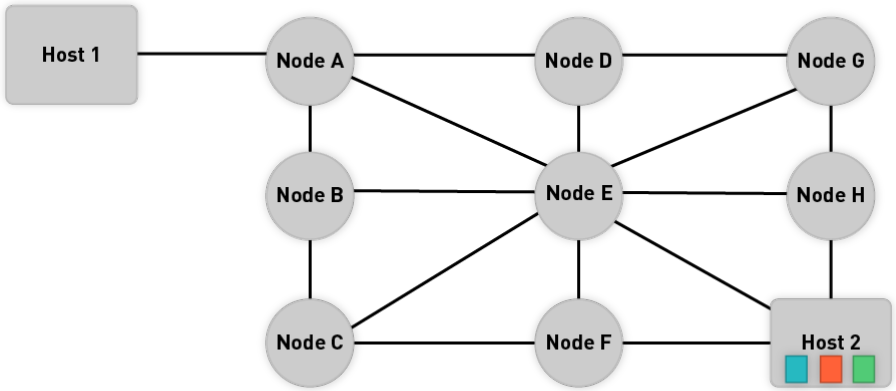
Route Packets in Parallel if Possible

The data packets take different routes to their destinations.



Barrier Sync: Wait Until All Packets Arrive

The data packets take different routes to their destinations.



Individual packets get transmitted in isolation. There is no communication between the packets, and they synchronize only at the destination node; this is known as barrier-based synchronization in parallel computing.

Handle process synchronization through devices such as barriers.

Conclusion

1. The key to success here was divide and conquer
2. Decompose a large task into smaller ones
3. We came up with a very nice framework for parallelizing tasks on the command line!
    - But it is limited
    - Granularity of task is somewhat coarse
    - No fault tolerance
    - No control over file space
4. Divide and conquer does not come for free; there are obligations in terms of communication, synchronization, and fault tolerance

Overview

1. Background on parallel computing
    - General principles and concepts
2. MapReduce
3. Hadoop

Serial Computation vs. Parallel

1. Traditionally, computer software has been written for serial computation.
    - To solve a problem, an algorithm is constructed and implemented as a serial stream or list of instructions.
    - These instructions are executed on a central processing unit on one computer. Only one instruction may execute at a time—after that instruction is finished, the next is executed.
2. Parallel computing, in contrast, uses multiple processing elements simultaneously to solve a problem.
    - This is accomplished by breaking the problem into independent parts so that each processing element can execute its part of the algorithm simultaneously with the others.
    - The processing elements can be diverse and include resources such as a single computer with multiple processors, several networked computers, specialized hardware, or a combination of the above.

Parallel Computing

1. Parallel computing is a form of computation in which many calculations are carried out simultaneously.
2. It operates on the principle that large problems can often be divided into smaller ones, which are then solved concurrently ("in parallel").

Parallel Computers

Parallel computers can be roughly classified according to the level at which the hardware supports parallelism.

1. Single computer
    - With multicore and multi-processor computers having multiple processing elements within a single machine
2. Clusters of computers
    - While clusters, massively parallel processors (MPPs), and grids use multiple computers to work on the same task
3. Specialized parallel computer architectures are sometimes used alongside traditional processors for accelerating specific tasks, e.g., GPUs for graphics processing

Challenges of Group Work

1. Division of work
2. Coordination costs (partition problem, communicate)
    - Coordination costs represent time and energy that group work consumes that individual work does not
    - They include the time it takes to coordinate schedules, arrange meetings, meet, correspond, make decisions collectively
    - Integrate the contributions of group members, etc.
    - The time spent on each of these tasks may be small, but together they are significant
3. Coordination costs can't be eliminated
4. Synergy, motivation are also costs
5. Similar challenges exist in distributed computing

Parallel Computer Programs Are Challenging

Distributed computation is similar to group work

1. Single node
    - Parallel computer programs are more difficult to write than sequential ones, because concurrency introduces several new classes of potential software bugs, of which race conditions are the most common
    - Communication and synchronization between the different subtasks are typically some of the greatest obstacles to getting good parallel program performance
2. Clusters
    - Similar issues arise in clusters of computers

Developer Responsibilities

1. In traditional parallel or distributed programming environments, the developer needs to explicitly address many (and sometimes all) of the above issues:
    - Concurrency, communication synchronization
    - (Just as we did earlier in our command-line framework)
2. In shared memory programming:
    - The developer needs to
        - Explicitly coordinate access to shared data structures through synchronization primitives such as mutexes
        - Explicitly handle process synchronization through devices such as barriers
        - Remain ever vigilant for common problems such as deadlocks and race conditions

Without Synchronization

Shared variable X

1. Thread A
    - Read variable X
    - Compute X + 1
    - Assign to X
2. Thread B
    - Read variable X
    - Compute X + 1
    - Assign to X

Without Synchronization → Race Condition

Depending on order, final value can be different

- Case 1: X = X + 1
  - A and B could both read the initial value of X and then overwrite each other
  - Final result: X = X + 1
- Case 2: X = X + 2
  - A reads and writes, then B reads and writes
  - Final result: X = X + 2
- That's a race condition
  - And you will grow to hate them—if you don't already

Use Synchronization to Avoid Race Conditions

- There are several ways to synchronize and avoid race conditions
  - Depending on the level of parallelism possible for the problem at hand
    - Mutex
    - Barrier

Avoid Race Conditions by Syncing with Mutex

1. Simplest is mutex
    - Mutually exclusive
        - In computer science, mutual exclusion refers to the requirement of ensuring that no two concurrent processes are in their critical section at the same time; it is a basic requirement in concurrency control, to prevent race conditions
    - In computer programming, a mutex is a program object that allows multiple program threads to share the same resource, such as file access, but not simultaneously
    - When a program is started, a mutex is created with a unique name
2. Usually a mutex is associated with a variable or code block
    - `Mutex_t count_lock;`
    - `Mutex_lock(&count_lock);  #If you want to write to a variable, you need to own the lock first`
    - `Modify count variable`
    - `mutex_unlock(&count_lock);  #release lock`

Onus Is on the Programmer

1. If you
    - Alter a variable without a lock
    - Enter a code block without a lock
2. You won't know it's happening until you see the side effects
    - At random intervals in a totally nondeterministic way

Mutexes Can Lead to Deadlock

1. Next issue: Mutex base syncing can cause deadlocks.
2. Threads A and B both need locks for variables X and Y.
    - A acquires mutex_X.
    - B acquires mutex_Y.
    - A waits for mutex_Y to be free.
    - B waits for mutex_X to be free.

Deadlock

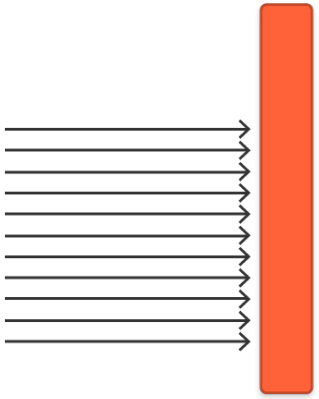
Threads A and B both need locks for variables X and Y.

1. A acquires mutex_X.
2. B acquires mutex_Y.
3. A waits for mutex_Y to be free.
4. B waits for mutex_X to be free.

Join Pattern for Synchronization

Those problems that can be decomposed into independent subtasks, requiring no communication or synchronization between the subtasks except a **join** or **barrier** at the end

Synchronization Through a Barrier



1. Another popular way of syncing is the barrier method
    - In parallel computing, a barrier is a type of synchronization method
2. Explicitly handle process synchronization through devices such as barriers
3. It is pretty effective and very coarse grained and can be great in certain types of problems
    - A barrier for a group of threads or processes in the source code means any thread or process must stop at this point and cannot proceed until all other threads or processes reach this barrier

Summary: Communication and Synchronization

Those problems that can be decomposed into independent subtasks, requiring no communication or synchronization between the subtasks except a join or barrier at the end, are very parallelizable (embarrassingly parallel)

- Mutex pattern
- Join pattern
- Barrier pattern

Categories of Parallel Computation Tasks

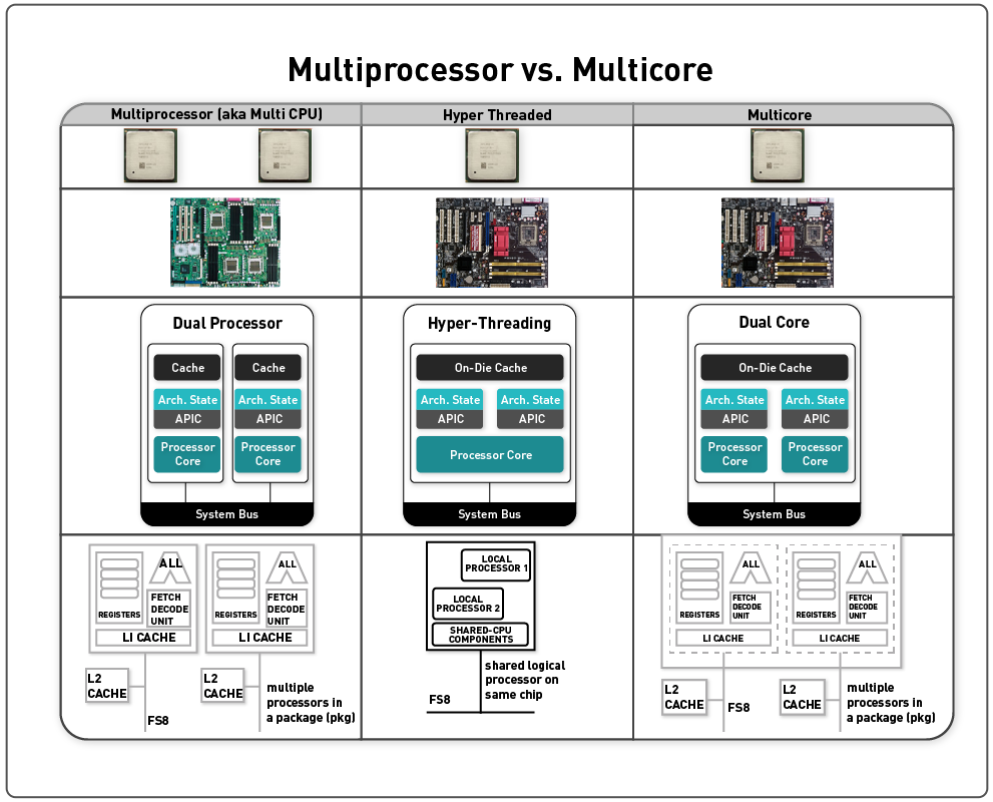
- Applications are often classified according to how often their subtasks need to synchronize or communicate with each other
- Fine-grained parallelism
  - An application exhibits fine-grained parallelism if its subtasks must communicate many times per second (share memory programming)
- Coarse-grained parallelism
  - It exhibits coarse-grained parallelism if they do not communicate many times per second
- Embarrassingly parallel
  - An application is **embarrassingly parallel** if it rarely or never has to communicate: divide and conquer; summing a list of numbers
  - Such applications are considered the easiest to parallelize
  - Can be realized on a shared nothing architecture (see this shortly)

Examples of Embarrassingly Parallel Problems

1. An application is embarrassingly parallel if it rarely or never has to communicate
2. Applications:
    - Summing a list of numbers
    - Matrix multiplication
    - Lots of machine learning algorithms
        - Tree growth step of the random forest machine learning technique
        - Genetic algorithms and other evolutionary computation metaheuristics
    - Serving static files on a Web server to multiple users at once
    - Computer simulations comparing many independent scenarios, such as climate models
    - Ensemble of numerical weather predictions
    - Discrete Fourier transform, where each harmonic is independently calculated
    - Many, many more ...

CPUs: Multiprocessor vs. Multicore

- We think about computation in terms of CPU, memory, and hard disks (and possibly network bandwidth)
- CPU (central processing unit)
  - A computer (motherboard) has multiple components for computation
    - A CPU, or simply "processor," contains many discrete parts, such as one or more memory caches for instructions and data, instruction decoders, and various types of execution units for performing arithmetic or logical operations
    - GPU (graphics processor unit)
- Multicore CPU
  - The CPU is not always busy; e.g., it may be idle while the GPU is performing a task. A CPU can be better utilized by sharing with many tasks. In 2000, IBM first introduced a dual-core CPU.
  - A multicore CPU has multiple execution cores on one CPU; also shares memory; may have its LX caches


Multiprocessor vs. Multicore
Multiprocessor (aka Multi CPU)
Hyper Threaded
Multicore
Dual Processor
Cache
Cache
Arch. State
APIC
Arch. State
APIC
Processor Core
Processor Core
System Bus
Hyper-Threading
On-Die Cache
Arch. State
APIC
Arch. State
APIC
Processor Core
System Bus
Dual Core
On-Die Cache
Arch. State
APIC
Arch. State
APIC
Processor Core
Processor Core
System Bus
REGISTERS
ALL
FETCH DECODE UNIT
LI CACHE
L2 CACHE
FS8
multiple processors in a package (pkg)
LOCAL PROCESSOR 1
LOCAL PROCESSOR 2
SHARED-CPU COMPONENTS
shared logical processor on same chip

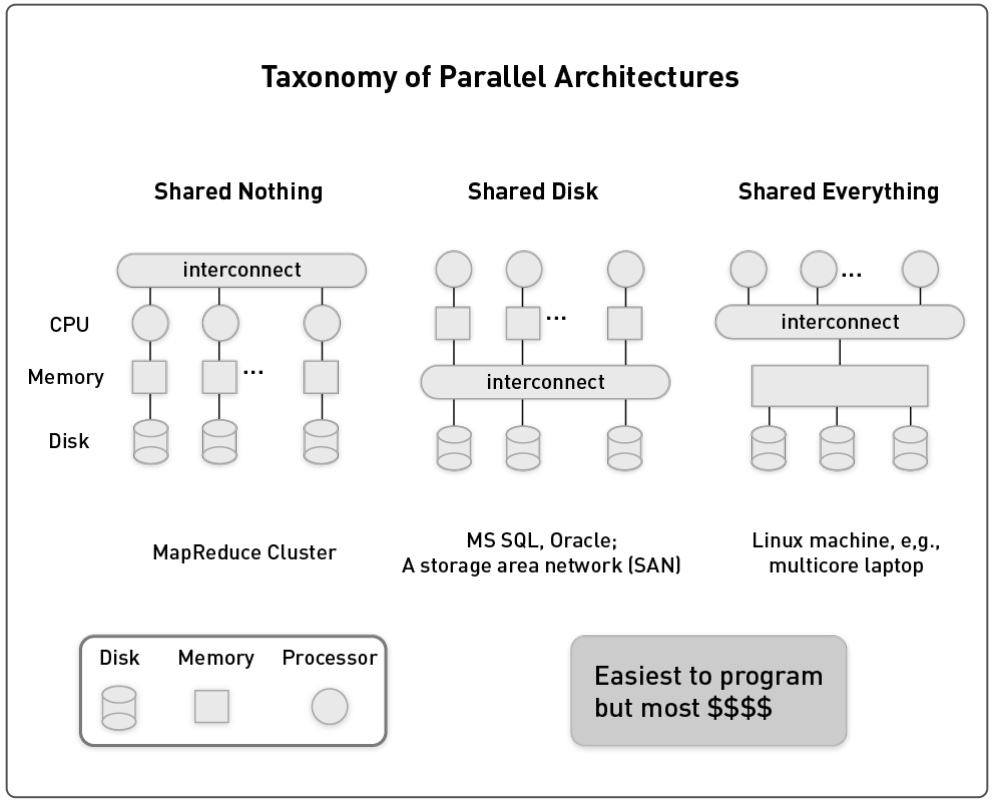
iPhone 6 Single CPU with Dual Cores

Die size	CPU ISA	CPU	CPU cache	GPU	Memory techno
? mm²[87]	ARMv8-A[44]	1.4 GHx dual-core Cyclone 2nd gen.[68]	L1i: 64 KB L1d: 64 KB L2: 1 MB L3: 4 MB[44]	PowerVR GX6450 (quad-core) @ 450 MHz (115.2 GFLOPS)[88][89]	64-bit Single-channel 1600[47] (12.8 GB/sec)[8
6 mm²[73]	ARMv8-A	1.5 GHz triple-core Cyclone 2nd gen.[73]	L1i: 64 KB L1d: 64 KB L2: 2 MB L3:4 MB[73]	PowerVR GXA6850 (octa-core) @ 450 Mhz (230.4 GFLOPS).[73][80]	64-bit Dual-channel L 1600[73] (25.6 GB/sec)[8

- iPhone has single CPU with dual cores
- And a quad-core GPU
- iPad: three-core CPU; eight-core GPU

Parallel Systems

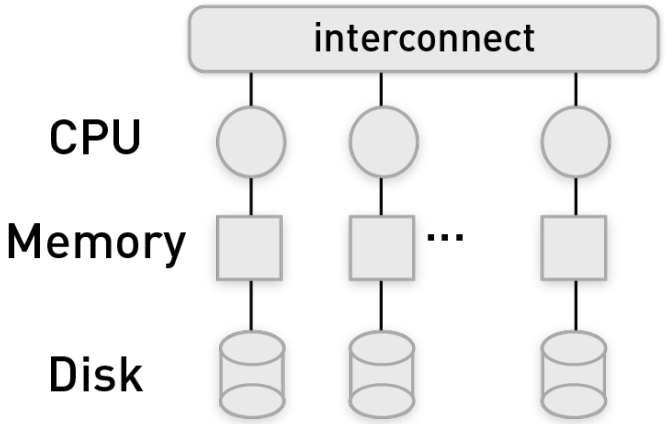
- Resources
  - CPU
  - Memory
  - Disk
  - Network
- There are many parallel architectures that are commonly used in practice:
  - Shared nothing
  - Shared disc
  - Shared all


Taxonomy of Parallel Architectures
Shared Nothing
Shared Disk
Shared Everything
interconnect
CPU
Memory
Disk
interconnect
interconnect
MapReduce Cluster
MS SQL, Oracle;
A storage area network (SAN)
Linux machine, e.g.,
multicore laptop
Disk
Memory
Processor
Easiest to program
but most $$$$

Shared Memory Programming

- Onus on the programmer
- In shared memory programming, the developer needs:
  - To explicitly coordinate access to shared data structures through synchronization primitives such as mutexes
  - To explicitly handle process synchronization through devices such as barriers
  - To remain ever vigilant for common problems such as deadlocks and race conditions

Shared Nothing Architecture



Scale out, linear scaling (cheap) vs. scale up (expensive)

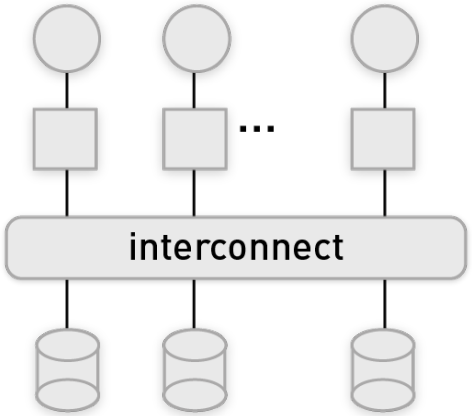
Shared Nothing Architecture (cont.)

- Shared nothing: The machines are connected only by the network (scale out).
  - A shared nothing architecture (SN) is a distributed computing architecture in which each node is independent and self-sufficient, and there is no single point of contention across the system.
  - A "shared nothing" architecture means that each computer system has its own private memory and private disk.
  - The "shared nothing" architecture shares network bandwidth because it must transfer data from machines doing the map tasks to machines doing the reduce tasks over the network.
- In order to achieve a good workload distribution, MPP systems have to use a hash algorithm to distribute (partition) data evenly across available CPU cores.
- This architecture is followed by essentially all high-performance, scalable database management systems (e.g., MemSQL, Teradata, Netezza, Greenplum, Parcel, DB2, Vertica) and most high-end Internet companies, including Amazon, Akamai, Yahoo, Google, and Facebook.

Shared Nothing (SN) Architecture Is Popular

- Shared nothing is popular for Web development because of its scalability
  - As Google has demonstrated, a pure SN system can scale almost infinitely simply by adding nodes in the form of inexpensive computers, since there is no single bottleneck to slow the system down
- Shard or partitioned data (e.g., distribute documents)
  - Google calls this sharding
  - An SN system typically partitions its data among many nodes on different databases (assigning different computers to deal with different users or queries) or may require every node to maintain its own copy of the application's data, using some kind of coordination protocol
  - This is often referred to as database sharing

Shared Everything (Disk or Memory)



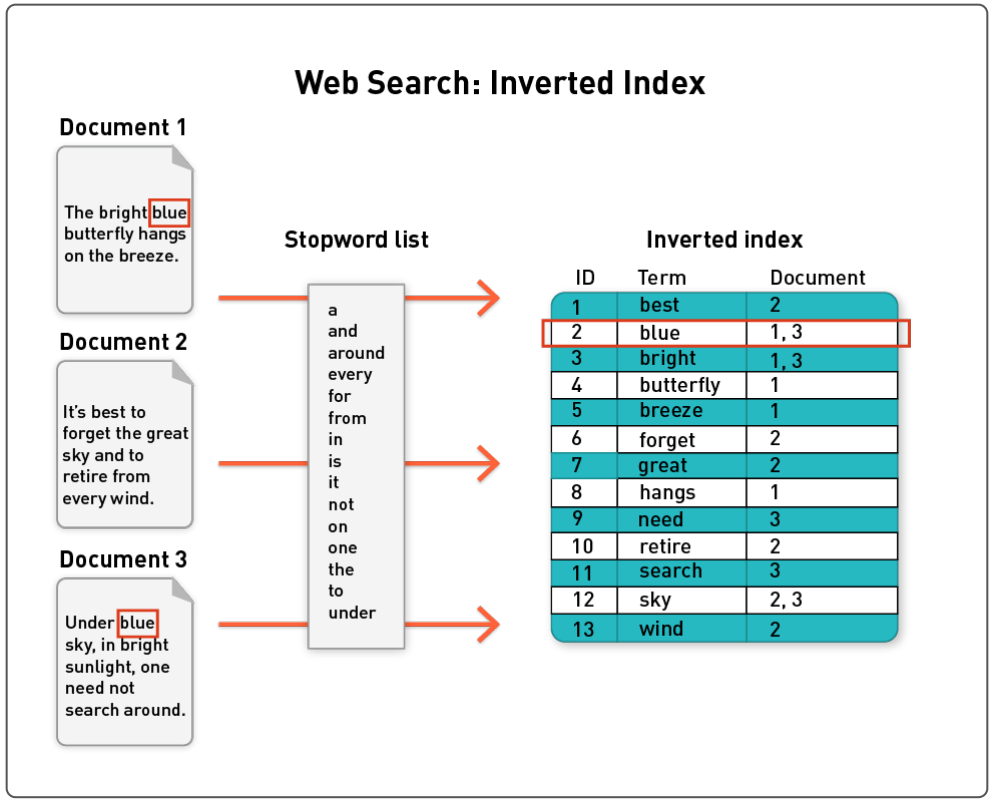
Scale up, vertical scaling (expensive): Add more memory or disks

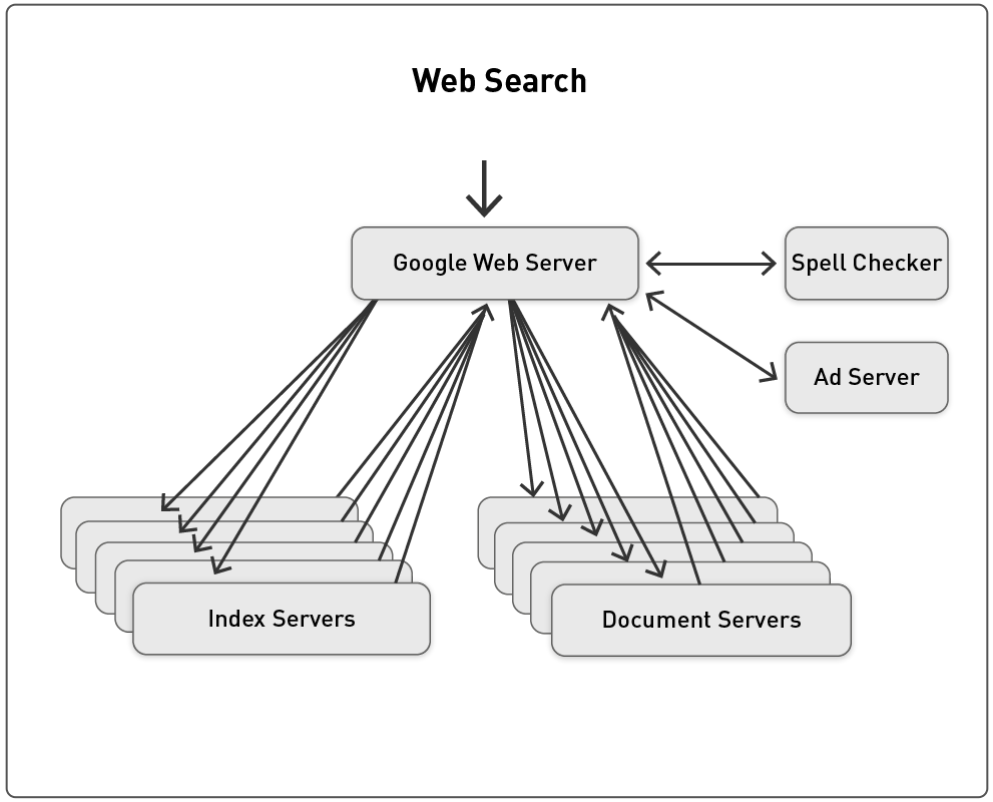
Shared Everything (cont.)

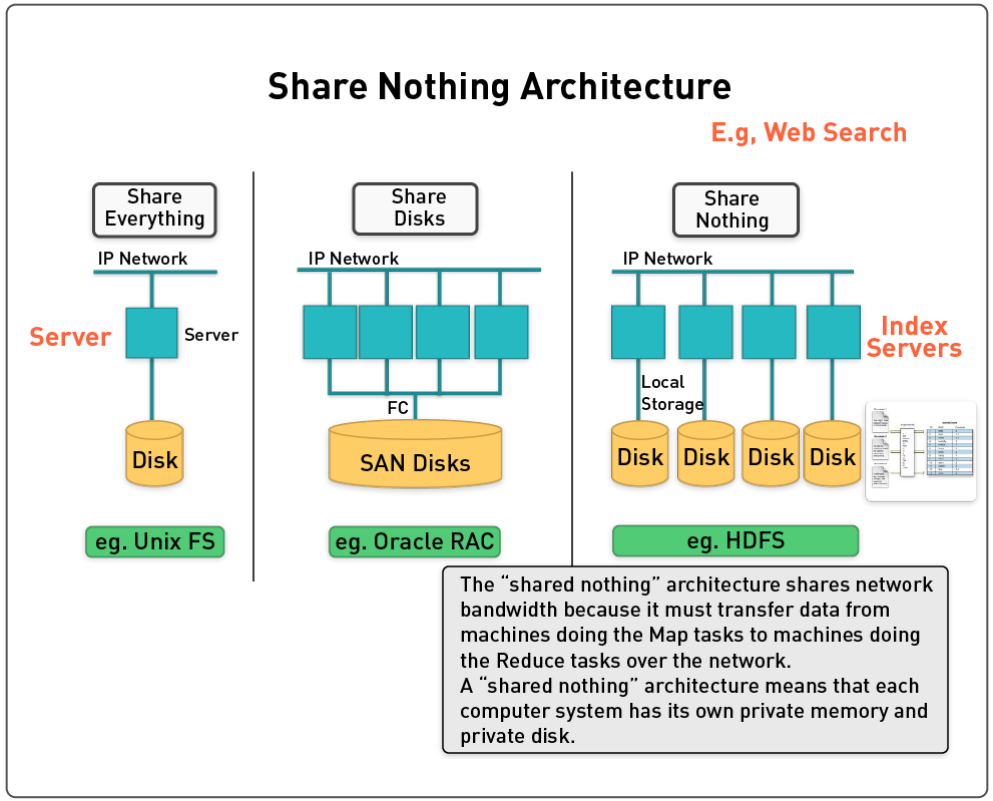
- Oracle RAC does not run on a shared nothing system. It was built many years ago to run on a "shared disk" architecture.
- In this world, a computer system has private memory but shares a disk system with other computer systems.
    - Such a "disk cluster" was popularized in the 1990s by Sun and HP, among others. In the 2000s, this architecture has been replaced by "grid computing," which uses shared nothing.
- Shared disk has well-known scalability problems when applied to database management systems (DBMSs) and is super-expensive.
    - A storage area network (SAN) can cost $500,000 and still struggle with terabytes of data.

Shared Nothing vs. Shared Something

- Shared Nothing Architecture (SN)
  - None of the nodes share memory or disk storage.
  - The "shared nothing" architecture shares network bandwidth.
- Shared Something
  - People typically contrast SN with systems that keep a large amount of centrally stored state information, whether in a database, an application server, or any other similar single point of contention.
- Shared nothing is popular.


Web Search: Inverted Index
Document 1
The bright blue butterfly hangs on the breeze.
Document 2
It's best to forget the great sky and to retire from every wind.
Document 3
Under blue sky, in bright sunlight, one need not search around.
Stopword list
a
and
around
every
for
from
in
is
it
not
on
one
the
to
under
Inverted index
ID Term Document
1 best 2
2 blue 1, 3
3 bright 1, 3
4 butterfly 1
5 breeze 1
6 forget 2
7 great 2
8 hangs 1
9 need 3
10 retire 2
11 search 3
12 sky 2, 3
13 wind 2


Web Search
Google Web Server
Spell Checker
Ad Server
Index Servers
Document Servers


Share Nothing Architecture
E.g, Web Search
Share Everything
IP Network
Server
Server
Disk
eg. Unix FS
Share Disks
IP Network
FC
SAN Disks
eg. Oracle RAC
Share Nothing
IP Network
Index Servers
Local Storage
Disk
Disk
Disk
Disk
eg. HDFS
The "shared nothing" architecture shares network bandwidth because it must transfer data from machines doing the Map tasks to machines doing the Reduce tasks over the network.
A "shared nothing" architecture means that each computer system has its own private memory and private disk.

Overview

- In the previous section, we looked at different architectures for parallel computation, mainly from a hardware perspective
- While in this section we look at parallel developer frameworks from a software perspective

Frameworks for Parallel Computation

- Onus is on the programmer (mutexes)
- Language extensions such as OpenMP (Open Multi-Processing) can be used for shared-memory parallelism
    - OpenMP has predefined locking paradigms
        - Reduce the chance of deadlock
        - Not foolproof (though possibly fool resistant)
- Cluster and grid computing
    - Provide logical abstractions that hide details of operating system synchronization and communications primitives
    - **MPI**: Libraries implementing the Message Passing Interface (MPI) for cluster-level parallelism (same local network)
    - **MapReduce**: MapReduce is a framework processing parallelizable problems across huge data sets, using a large number of computers (nodes); cluster or grid (nodes are shared across geographically or heterogeneous hardware)

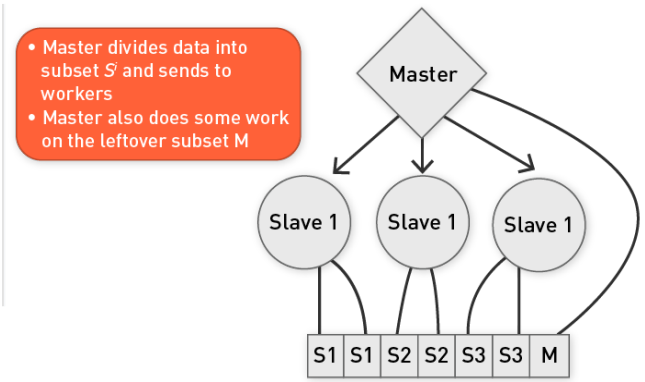
MPI Distributed Computing Frameworks

- Message Passing Interface (MPI) provides a powerful, efficient, and portable way to express parallel programs.
- MPI is a specification for an API that allows many computers to communicate with one another.
    - It is used in computer clusters and supercomputers.
    - MPI was created by William Gropp, Ewing Lusk, and others.
- MPI, in short:
    - Hides the details of architecture.
    - Hides the details of message passing and buffering.
    - Provides message management services:
        - Packaging
        - Send, receive
        - Broadcast, reduce, scatter, gather message modes

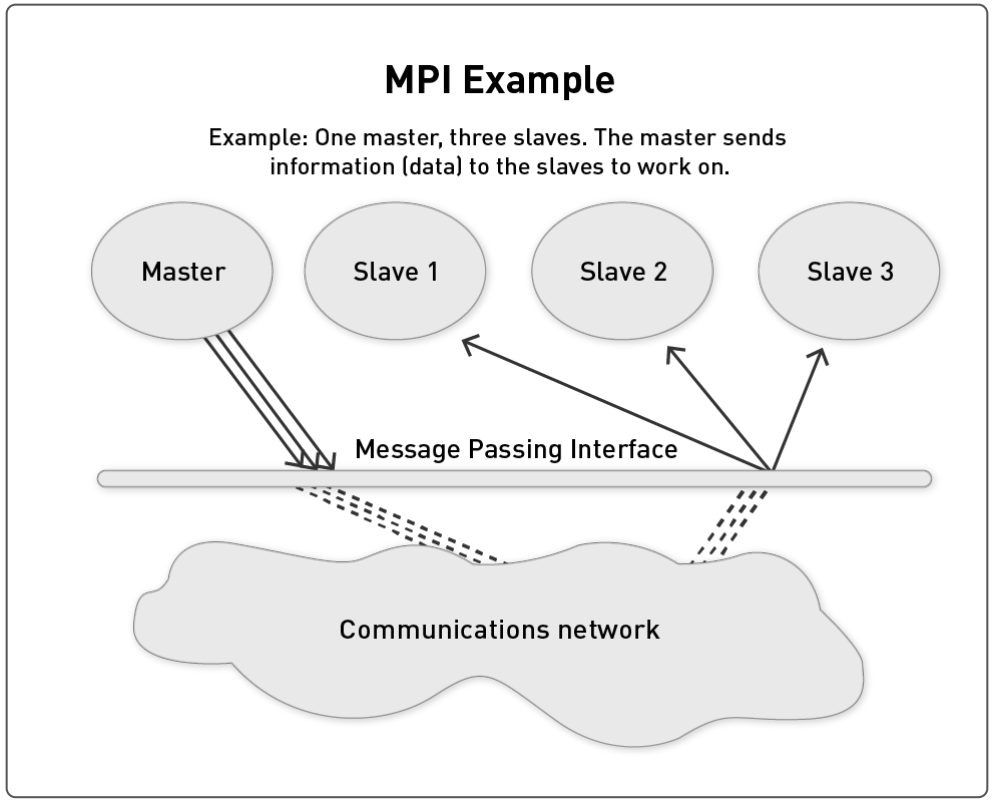
MPI Basics

- MPI defines an environment where programs can run in parallel and communicate with each other by passing messages to each other. There are two major parts to MPI:
  - The environment programs run in.
  - The library calls programs use to communicate.
- MPI is so preferred by users because it is so simple to use. Programs just use *messages* to communicate with each other. Can you think of another successful system that functions using message passing? (The Internet!)
  - Each program has a mailbox (called a queue).
  - Any program can put a message into another program's mailbox.
  - When a program is ready to process a message, it receives a message from its own mailbox and does something with it.

MPI: Master Divides and Sends Data to Workers



- Example task: Sum an array of numbers.
- In this program, the master divides the array into subarrays S^i and sends these to each slave.
- Each slave will then calculate the sum of its subarray. In the case where the size of the array is not an even multiple of the number of slaves, the master will finish the remaining work by calculating the sum of the last clause of the array.


MPI Example
Example: One master, three slaves. The master sends information (data) to the slaves to work on.
Master
Slave 1
Slave 2
Slave 3
Message Passing Interface
Communications network

Why Not MPI for Big Data?

- But ...
  - Data transfer
  - Requires cooperation of sender and receiver
  - Cooperation not always apparent in code
- Hard for programming
  - Explicitly communicate between nodes via message passing
  - Synchronization is hard and error prone
- Not designed for fault tolerance
- Not designed for data locality
  - Move data to computation nodes
  - Network bandwidth is limited
  - Please click here for Wikipedia article on MPI

Data Locality and Isolation

- Various patterns for parallel computation have been proposed over the years.
- These patterns, when composed, can be used to express computations in a wide variety of domains.

Design Patterns for Parallelization

A straightforward way to express both data locality and isolation:

Structured Patterns: An Overview

Parallel Pattern 1: Superscalar Sequences and Task Graphs

Parallel Pattern 2: Selection

Parallel Pattern 3: Map

Parallel Pattern 4: Gather

Parallel Pattern 5: Stencil

Parallel Pattern 6: Partition

Parallel Pattern 7: Reduce

Parallel Pattern 8: Scan

Parallel Pattern 9: Pack

Data Locality and Isolation (cont.)

- Scalable approaches to parallel computation have to take two things into account:
  - Tasks and data.
- Many approaches to parallel computation overemphasize the former (task) at the expense of the latter (e.g., MPI).
- However, in order to achieve scalable parallel performance, algorithms need to be designed in a way that emphasizes data locality and isolation.

Data Locality and Isolation: MapReduce

The MapReduce pattern provides a simple way for the software developer to express both data and locality in isolation.

Extending for Loops to Maps

E.g., a set of numbers

- A common pattern in sequential programming is iteration
- If we **disallow** dependencies between loop iterations, we do get a type of computation that can be parallelized easily: The map pattern
- In the map pattern, a set of loop indices are generated, and independent computations are performed for every unique index
- These computations are not allowed to communicate with one another
- At the end of the map, however, there is an implied barrier, and then other operations can depend on the output of the map operation as a whole

MapReduce Frameworks

- Some parallel programming systems, such as OpenMP and Cilk, have language support for the map pattern in the form of a parallel *for* loop; languages such as OpenCL (Open Computing Language) and CUDA (Complete Unified Device Architecture) support elemental functions (as "kernels") at the language level
- Limitations of these frameworks:
  - Developers are still burdened to keep track of how resources are made available to workers
  - Additionally, these frameworks are mostly designed to tackle processor-intensive problems and have only rudimentary support for dealing with very large amounts of input data
- MapReduce-Hadoop-Spark to the rescue!
  - Distributed data handling, and distributed computation, framework that hides system-level details

MapReduce: Move the Code to the Data

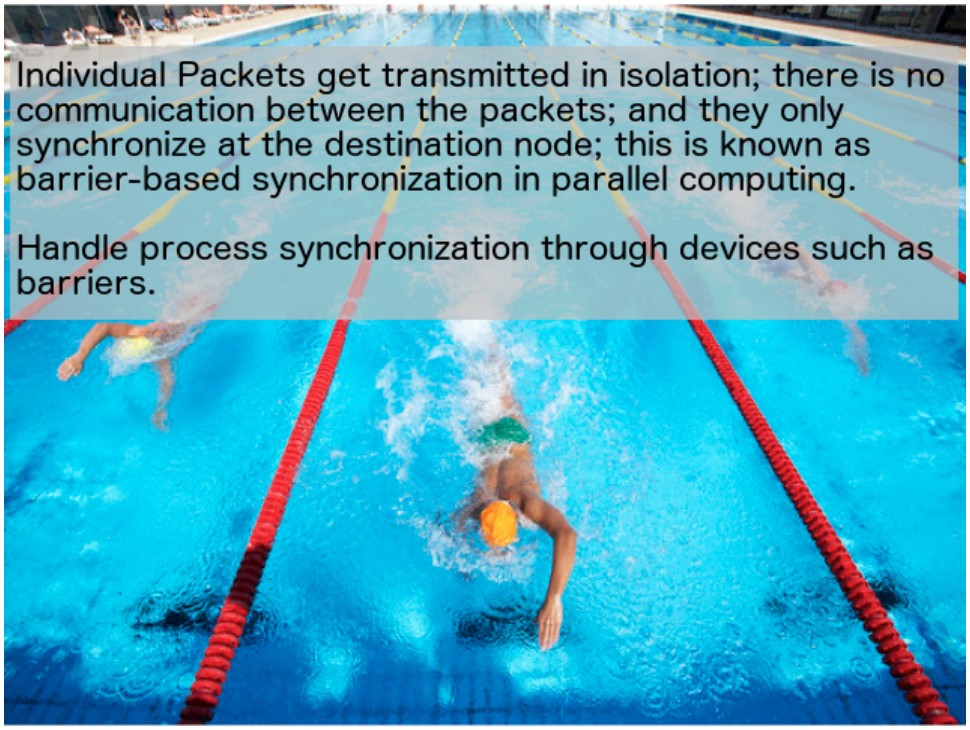
- Distributed data handling and storage
  - Terabytes and petabytes in size
  - Large-data processing by definition requires bringing data and code together for computation to occur; no small feat for huge data sets
  - MapReduce addresses this challenge by providing a simple abstraction for the developer, transparently handling most of the details behind the scenes in a scalable, robust, and efficient manner
- Distributed computation: Move the code to the data
  - Like OpenMP and MPI, MapReduce provides a means to distribute computation without burdening the programmer with the details of distributed computing (but at a different level of granularity)
  - But instead of moving large amounts of data around, it is far more efficient, if possible, to move the code to the data
  - This is operationally realized by spreading data across the local disks of nodes in a cluster and running processes on nodes that hold the data

MapReduce + Scalability and Fault Tolerance

- Inspired by functional programming
  - The model is inspired by the map and reduce functions commonly used in functional programming, although their purpose in the MapReduce framework is not the same as in their original forms
- Scalability and fault tolerance
  - The key contributions of the MapReduce framework are not the actual map and reduce functions but the scalability and fault tolerance achieved for a variety of applications by optimizing the execution engine once
- Optimizing the communication
  - Optimizing the communication cost is essential to a good MapReduce algorithm
- The use of this model is beneficial only when the optimized distributed shuffle operation (which reduces network communication cost) and fault-tolerance features of the MapReduce framework come into play

MapReduce Frameworks (cont.)

- The name MapReduce originally referred to the proprietary Google technology but has since become generic.
- A popular open-source implementation that has support for distributed shuffles is part of Apache Hadoop.
- MapReduce libraries have been written in many programming languages, with different levels of optimization.


Individual Packets get transmitted in isolation; there is no communication between the packets; and they only synchronize at the destination node; this is known as barrier-based synchronization in parallel computing.
Handle process synchronization through devices such as barriers.

Why Use MapReduce?

- Increase your computational power
- Processing large volumes of data
  - Must be able to split up the data in chunks for processing, which are then recombined later
  - Requires a constant flow of data from one simple state to another
- The Hadoop framework handles the processing details, leaving developers free to focus on application logic
- Within the Hadoop Core framework, MapReduce is often referred to as mapred, and HDFS is often referred to as dfs

Not Everything Is an MR

- MRs ideal for "embarrassingly parallel" problems
- Very little communication
- Easily distributed
- Linear computational flow

Hadoop

What Is MapReduce?

- MapReduce is a programming model for processing and generating large data sets with a parallel, distributed algorithm on a cluster
- Pioneered by Google
  - Processes 20 PB of data per day
- Popularized by open-source Hadoop project
  - Used by Yahoo!, Facebook, Amazon, NativeX, ...

Hadoop

- Distributed file system (HDFS)
  - Single namespace for entire cluster
  - Replicates data three times for fault tolerance
- MapReduce implementation
  - Executes user jobs specified as "map" and "reduce" functions
  - Manages work distribution and fault tolerance

What Is MapReduce Used For?

At Google:

-Index building for Google Search

-Article clustering for Google News

-Statistical machine translation

At Yahoo!:

-Index building for Yahoo! Search

-Spam detection for Yahoo! Mail

At Facebook:

-Data mining

-Ad optimization

-Spam detection

NativeX (Mobile Ad Tech):

-Ad optimization

-Building predictive models with 10s of GBs of data

-Process 10s of TBs of data

In research:

-Analyzing Wikipedia conflicts (PARC)

-Natural language processing (CMU)

-Bioinformatics (Maryland)

-Astronomical image analysis (Washington)

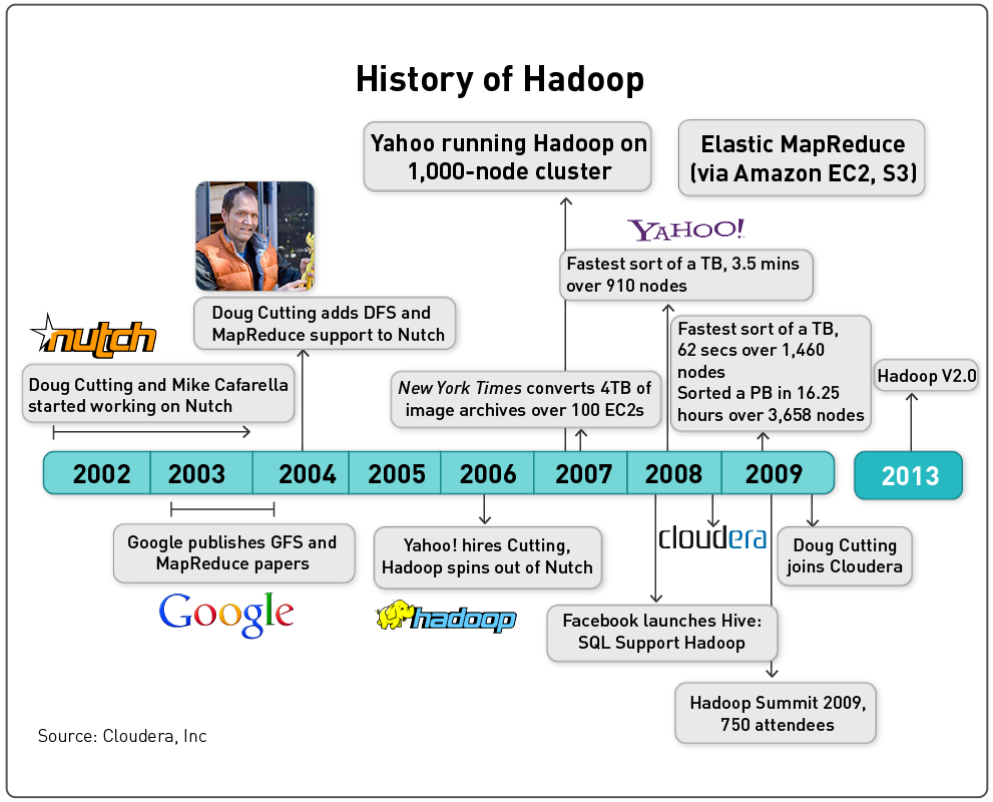
-Ocean climate simulation (Washington)

-Graph OLAP (NUS)

-& hellip;

- [Your application]

And for machine learning.

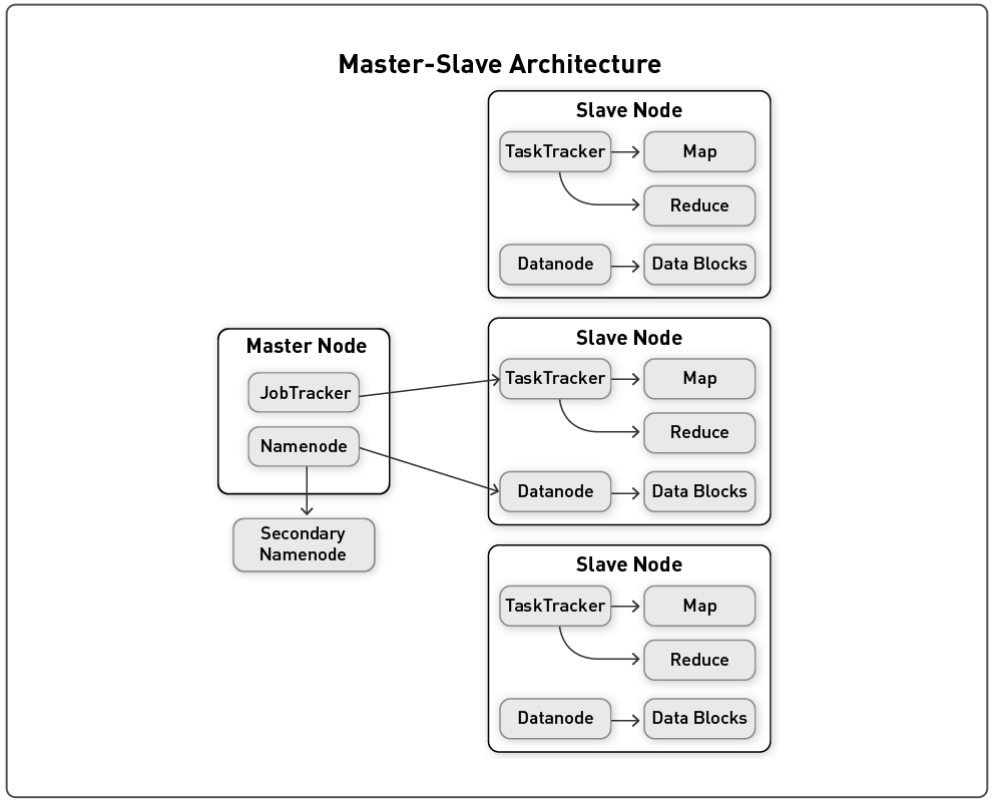

History of Hadoop
Yahoo running Hadoop on 1,000-node cluster
Elastic MapReduce (via Amazon EC2, S3)
YAHOO!
Fastest sort of a TB, 3.5 mins over 910 nodes
Doug Cutting adds DFS and MapReduce support to Nutch
nutch
Doug Cutting and Mike Cafarella started working on Nutch
New York Times converts 4TB of image archives over 100 EC2s
Fastest sort of a TB, 62 secs over 1,460 nodes
Sorted a PB in 16.25 hours over 3,658 nodes
Hadoop V2.0
2002
2003
2004
2005
2006
2007
2008
2009
2013
Google publishes GFS and MapReduce papers
Google
Yahoo! hires Cutting, Hadoop spins out of Nutch
hadoop
cloudera
Doug Cutting joins Cloudera
Facebook launches Hive: SQL Support Hadoop
Hadoop Summit 2009, 750 attendees
Source: Cloudera, Inc

What Is Hadoop?

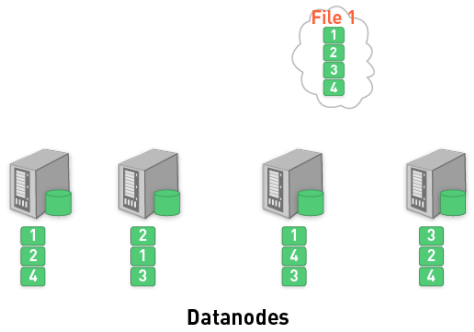
- The Apache Hadoop project develops open-source software for reliable, scalable, distributed computing. Hadoop includes:
  - Hadoop Common utilities.
  - Avro, a data serialization system with scripting languages.
  - HDFS, a distributed file system.
  - Hive, data summarization and ad hoc querying.
  - MapReduce, distributed processing on compute clusters.

Master-Slave Architecture

- Hadoop follows the master-slave architecture described in the original MapReduce paper.
- Since Hadoop consists of two parts—data storage (HDFS) and processing engine (MapReduce)—there are two types of master node.
- Types of node:
  - For HDFS, the master node is namenode, and slave is datanode.
  - For MapReduce in Hadoop, the master node is jobtracker.

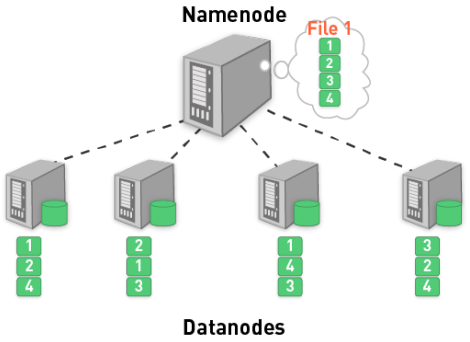

Master-Slave Architecture
Master Node
JobTracker
Namenode
Secondary Namenode
Slave Node
TaskTracker
Map
Reduce
Datanode
Data Blocks
Slave Node
TaskTracker
Map
Reduce
Datanode
Data Blocks
Slave Node
TaskTracker
Map
Reduce
Datanode
Data Blocks

Hadoop Distributed File System

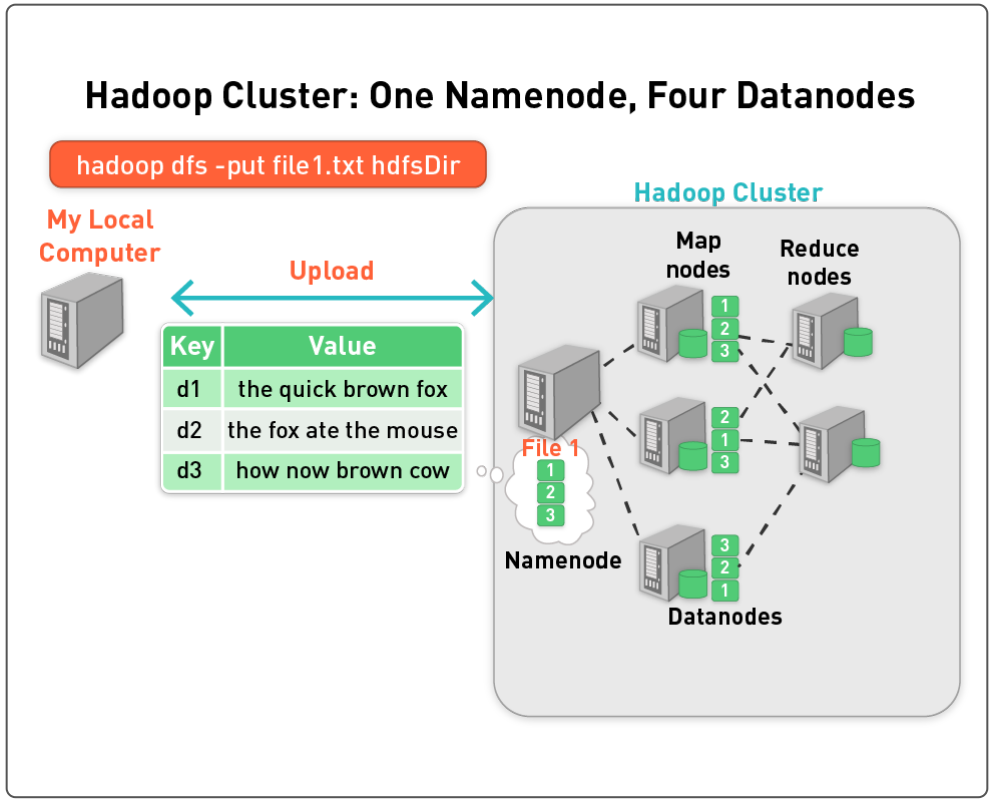


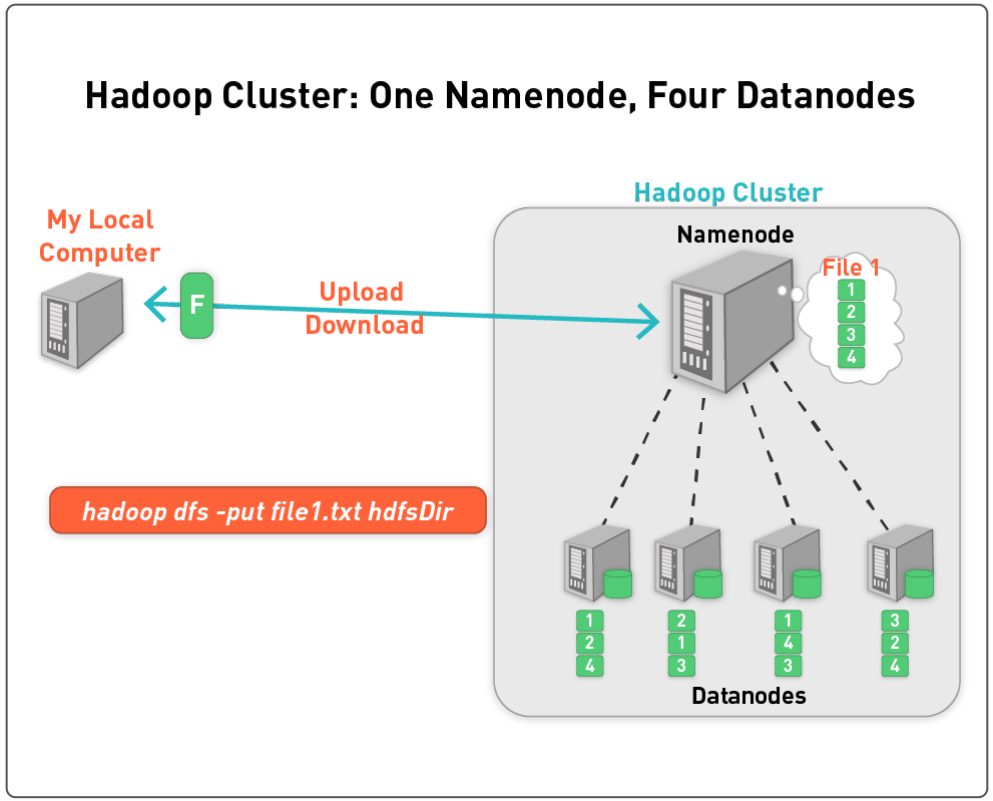
- Big files
  - 100s of GBs, TBs, even PBs
- Typical usage patterns
  - Append only
    - Data are rarely updated in place
  - Reads common
- Optimized for large files, sequential reads
- Files split into blocks (chunks): 64–128 MB, today 1GB–2GB
  - Blocks are replicated (usually three times) across several **datanodes** (called chunk or slave nodes)
  - Chunk nodes are compute nodes, too

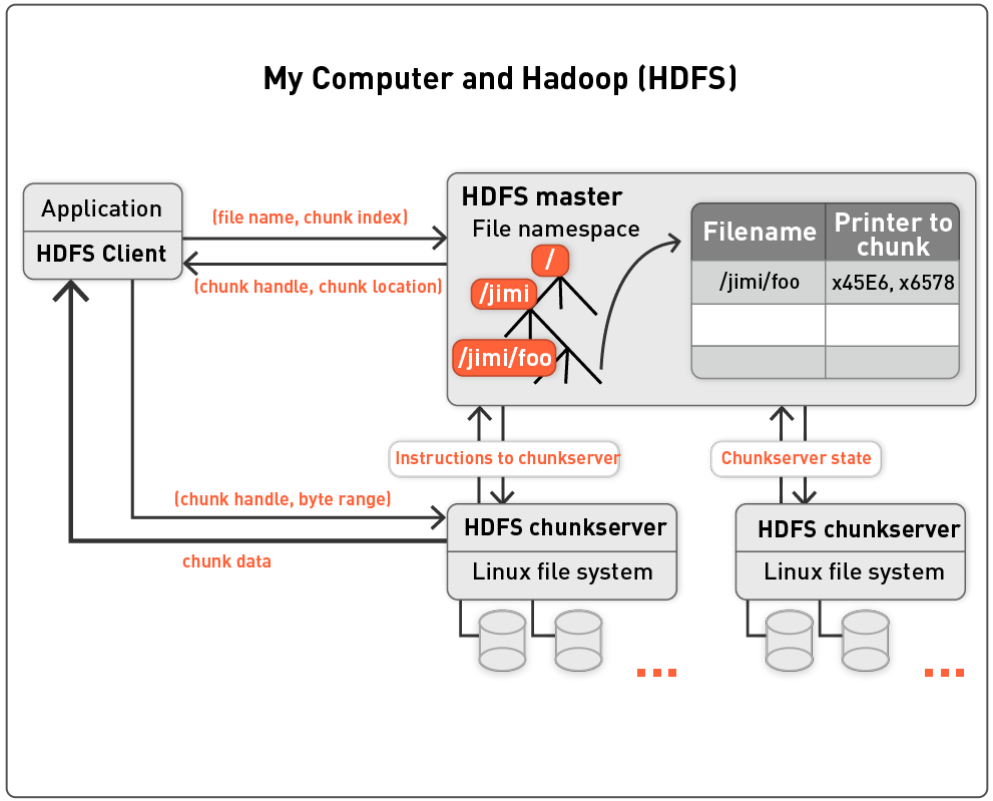
Hadoop Distributed File System (cont.)



- Single **namenode**(master node) stores metadata (file names, block locations, etc.)
  - May be replicated also
- Datanodes provide redundant store for the data blocksn
- Client library provides file access
  - Talks to master to find chunk servers
  - Connects directly to chunk servers to access data
    - Master node is not a bottleneck
    - Computation is done at chunk node (close to data)


Hadoop Cluster: One Namenode, Four Datanodes
hadoop dfs -put file1.txt hdfsDir
My Local Computer
Upload
Hadoop Cluster
Map nodes
Reduce nodes
Key
Value
d1
the quick brown fox
d2
the fox ate the mouse
d3
how now brown cow
File 1
Namenode
Datanodes


Hadoop Cluster: One Namenode, Four Datanodes
My Local Computer
F
Upload
Download
Hadoop Cluster
Namenode
File 1
1
2
3
4
hadoop dfs -put file1.txt hdfsDir
1 2 4
2 1 3
1 4 3
3 2 4
Datanodes


My Computer and Hadoop (HDFS)
Application
HDFS Client
(file name, chunk index)
(chunk handle, chunk location)
HDFS master
File namespace
/
/jimi
/jimi/foo
Filename
Printer to chunk
/jimi/foo
x45E6, x6578
Instructions to chunkserver
Chunkserver state
(chunk handle, byte range)
chunk data
HDFS chunkserver
Linux file system
HDFS chunkserver
Linux file system

HDFS Replication

- Default: 3× replication
- Rack-aware block placement algorithm
- Dynamic control of replication factor
- Balancer application to rebalance cluster in background

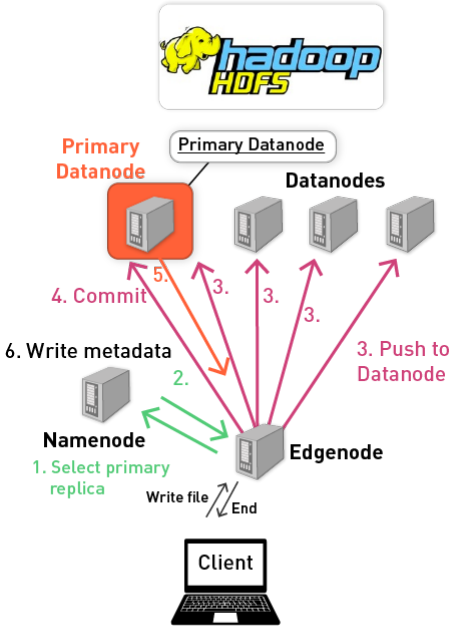
Interacting With HDFS

HDFS supports familiar syntax

```
hadoop fs -cat /root/example/file1.txt
hadoop fs -chmod -R 755 /root/example/file1.txt
hadoop fs -chown phil /root/example/file1.txt
hadoop fs -cp /root/example/file1.txt
   /root/example/file1.new
hadoop fs -ls /root/example/
hadoop fs -mkdir /root/example/new_directory
```

Write File in HDFS

1. Client contacts the namenode, who designates one of the replicas as the primary.
2. The response of the namenode contains who is the primary and who are the secondary replicas.
3. The client pushes its changes to all datanodes in any order, but this change is stored in a buffer of each datanode.
4. The client sends a "commit" request to the primary, which determines an order to update and then pushes this order to all other secondaries.
5. After all secondaries complete the commit, the primary response is sent to the client about the success.
6. All changes of block distribution and metadata changes are written to an operation log file at the namenode.



HDFS Summary

- Hadoop file system
  - Master-slave architecture
  - Organized data
  - How to get data in
  - How to get data out
- HDFS goals
  - Store large data sets
  - Deal with hardware failures
  - Emphasize streaming data access

Overview

The next couple of sections will focus on Hadoop MapReduce

- MapReduce
  - Functional programming
  - MapReduce
  - Animated examples
  - Hadoop in practice

Functional Programming: Map and Reduce

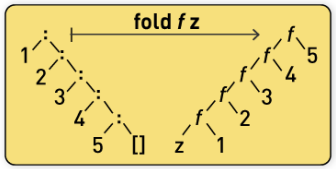
1. The idea of map and reduce is 40-plus years old.
    - Present in all functional programming languages
    - See, e.g., APL, Lisp, and ML
2. A key feature of functional languages is the concept of higher-order functions, or functions that can accept other functions as arguments
    - Map and reduce are higher-order functions
3. Alternate name for map: Apply all

Their purpose in the MapReduce framework is not the same as in their original forms

Map: A Higher-Order Function

1. Let
    - F(x: int) return r: int
    - V be an array of integers
2. W = map(F, V)
    - Apply F to every element of V
    - W[i] = F(V[i]) for all I
3. Map examples in Haskell:
    - map (+1) [1, 2, 3, 4, 5]
      == [2, 3, 4, 5, 6]
    - map (toLower) "abcDEFG12!@#"
      == "abcdefg12!@#"
    - map ('mod' 3) [1 ... 10]
      ==[1, 2, 0, 1, 2, 0, 1, 2, 0, 1]

Reduce: A Higher-Order Function



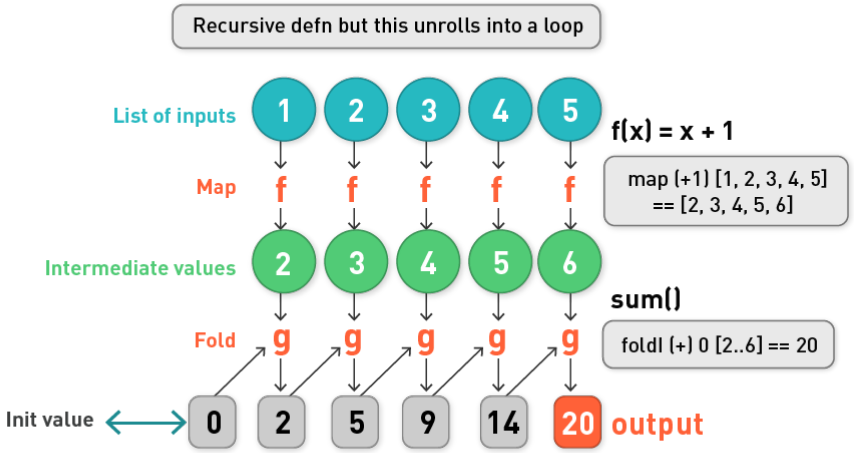
Trace

Recursive defn but this unrolls into a loop

- Reduce is also known as fold, accumulate, compress, or inject.
- Reduce/fold takes in a function and folds it in between the elements of a list.
  - Definition (recursive function):
    - foldl f z [] = z
    - foldl f z (x:xs) = foldl f (f z x) xs
  - Examples:
    - foldl (+) 0 [1 ... 5] == 15
    - foldl (+) 10 [1 ... 5] == 25

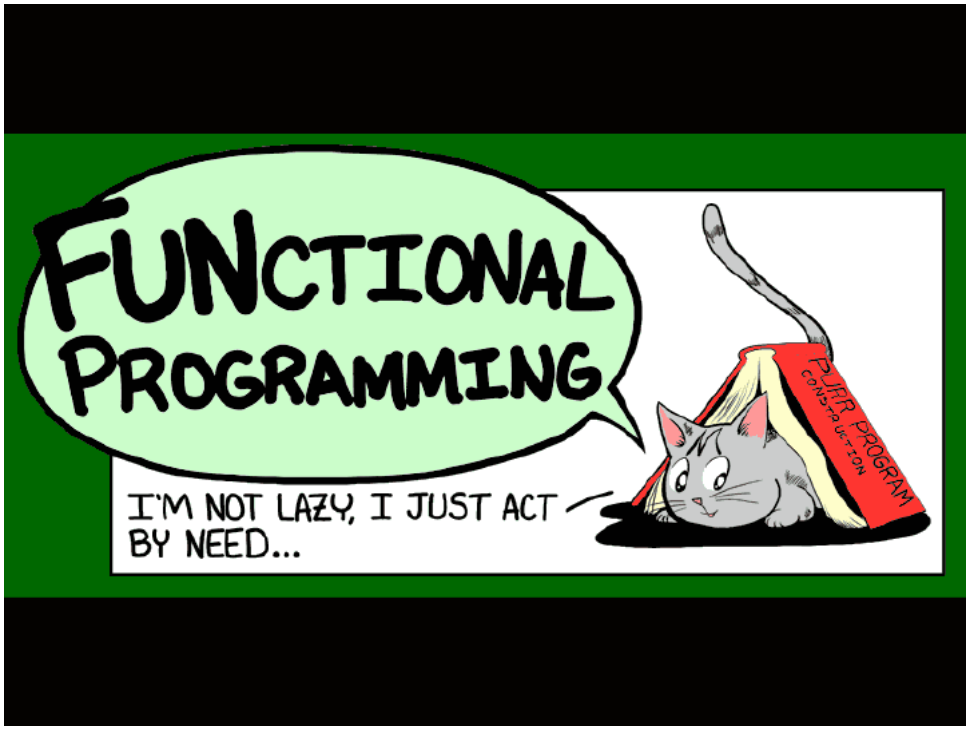
MapReduce Is Derived From EP

MapReduce ~ Map + Fold from functional programming!



FP Is ...

1. FP is all about the evaluation of mathematical functions and avoids changing state and mutable data.
2. It is a declarative programming paradigm, which means programming is done with expressions.
3. In functional code, the output value of a function depends only on the arguments that are input to the function, so calling a function f twice with the same value for an argument x will produce the same result f(x) each time.
4. This is an abstract diagram; despite this, you can see the potential for parallelism, especially in the map phase.

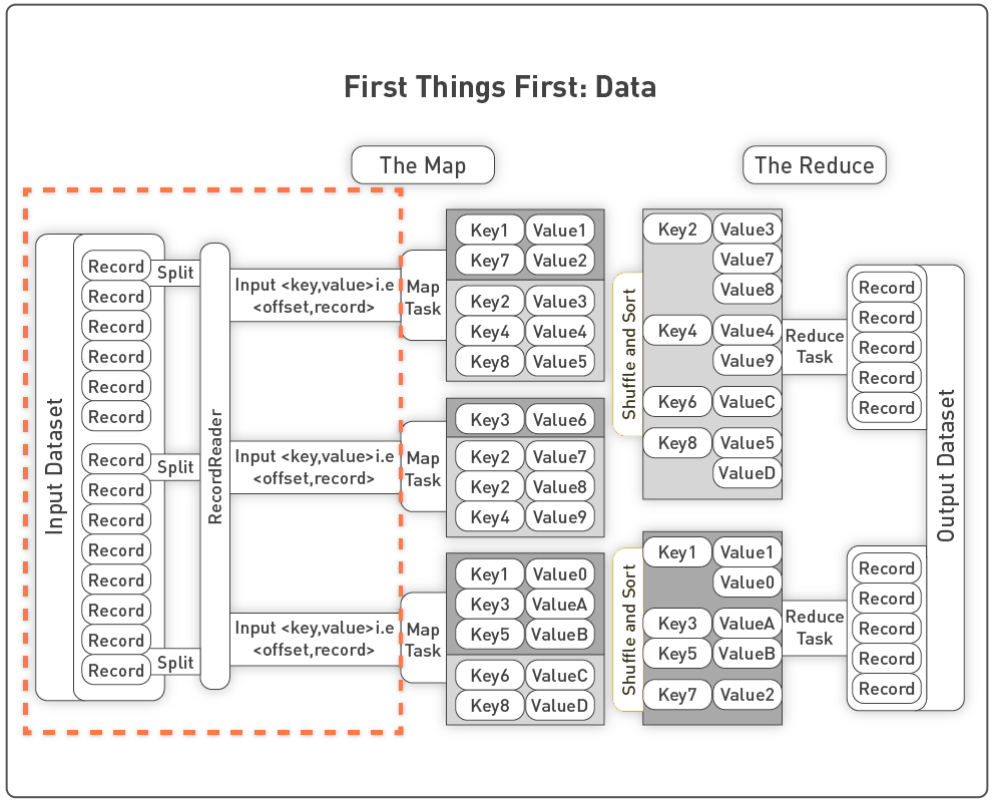

FUNCTIONAL PROGRAMMING
I'M NOT LAZY, I JUST ACT BY NEED...
PURE PROGRAM
CONSTRUCTION

Conclusion

1. In this section, we talked about functional programming and have seen the map and reduce higher-order functions, and we have seen the potential to parallelize at least the map function.
2. Over the next couple of sections, we will see how the map-and-reduce paradigm has been adapted.

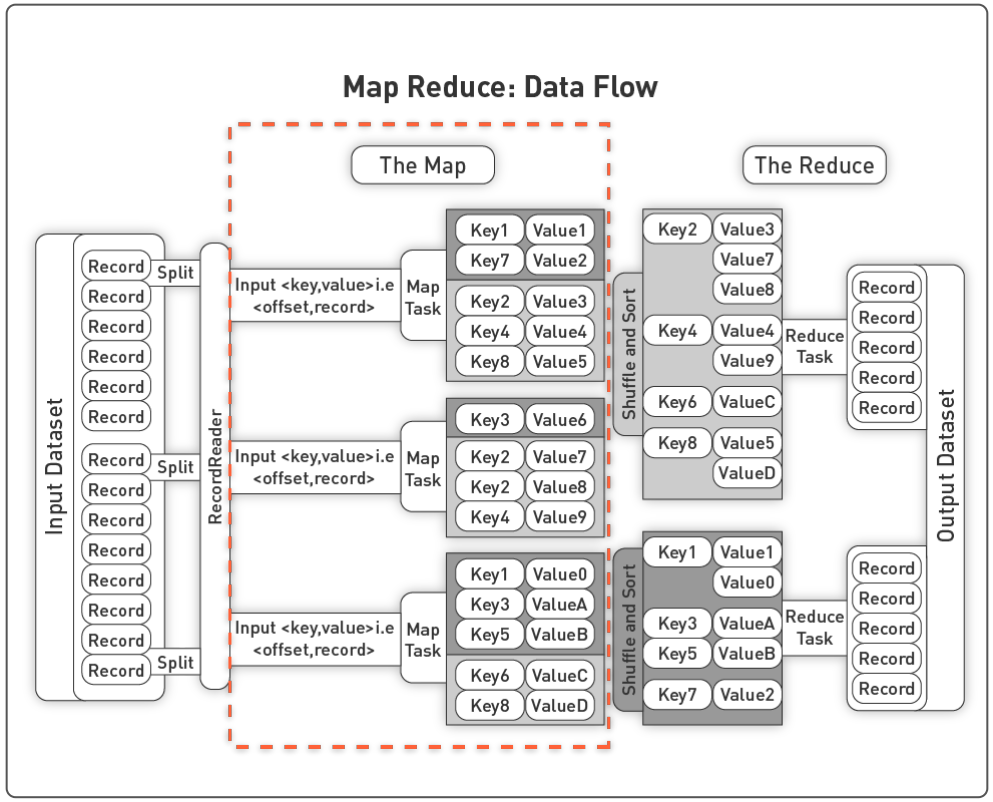
From Functional Programming to MapReduce

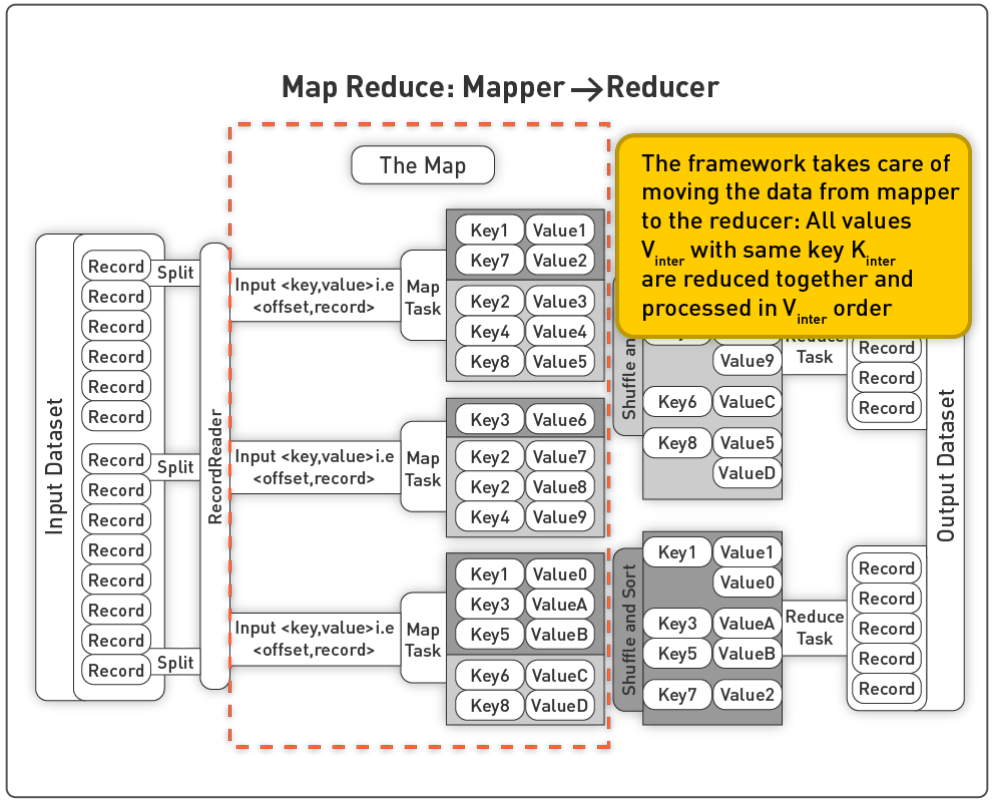
1. Maps correspond and reduce approximately.
    - The map phase in MapReduce roughly corresponds to the map operation in functional programming, whereas the reduce phase in MapReduce roughly corresponds to the fold operation in functional programming.
2. The MapReduce (MR) execution framework does it all.
    - HDFS takes care of the data,
    - While the MR framework takes care of everything else (almost).
    - As we will discuss in detail shortly, the MapReduce execution framework coordinates the map and reduce phases of processing over large amounts of data on large clusters of commodity machines.

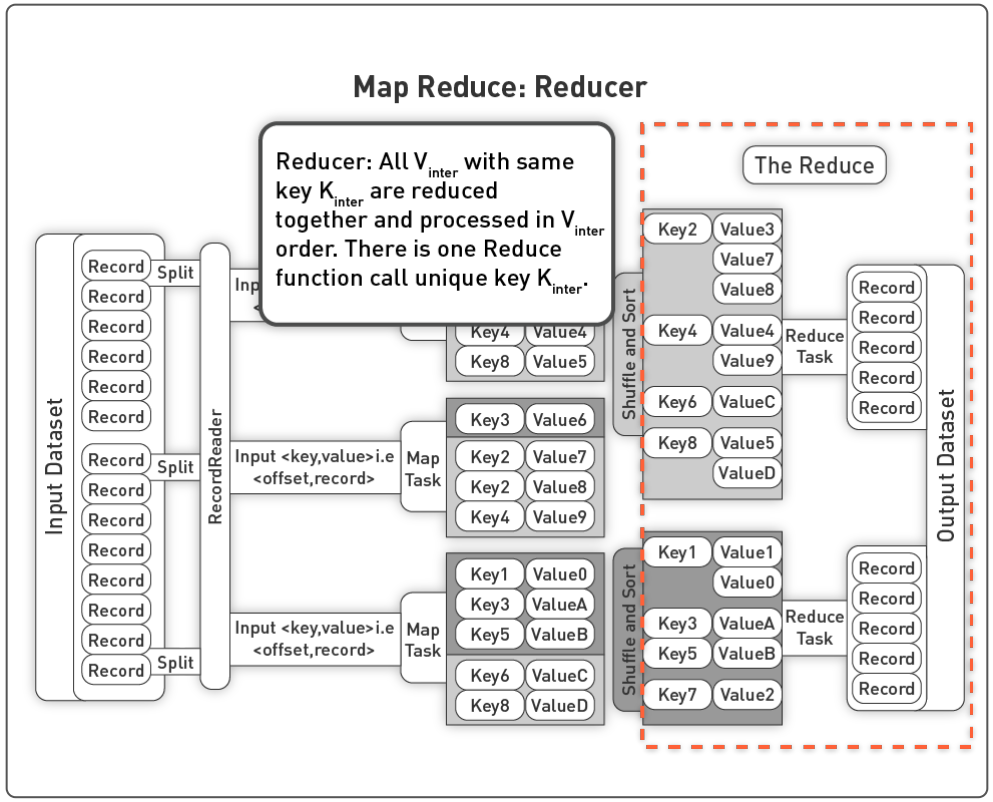

First Things First: Data
The Map
The Reduce
Input Dataset
Record
Split
RecordReader
Input <key,value>i.e <offset,record>
Map Task
Key1 Value1
Key7 Value2
Key2 Value3
Key4 Value4
Key8 Value5
Key3 Value6
Key2 Value7
Key2 Value8
Key4 Value9
Key1 Value0
Key3 ValueA
Key5 ValueB
Key6 ValueC
Key8 ValueD
Shuffle and Sort
Key2 Value3
Value7
Value8
Key4 Value4
Value9
Key6 ValueC
Key8 Value5
ValueD
Key1 Value1
Value0
Key3 ValueA
Key5 ValueB
Key7 Value2
Reduce Task
Output Dataset

Common Question

1. What is the restriction on the size of (key, value) pairs?
2. Each pair must be small enough to fit into one single machine; e.g., a document should be fine, an image should be fine, but 1PB value is probably not.

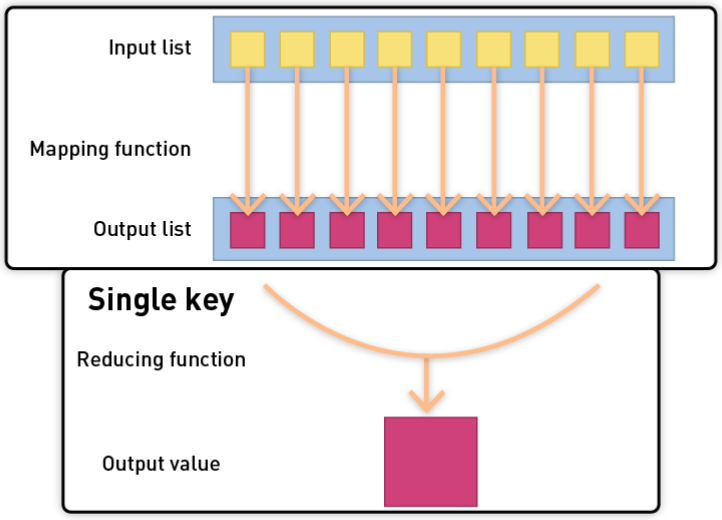

Map Reduce: Data Flow
The Map
The Reduce
Input Dataset
Record
Split
RecordReader
Input <key,value> i.e <offset,record>
Map Task
Key1 Value1
Key7 Value2
Key2 Value3
Key4 Value4
Key8 Value5
Key3 Value6
Key2 Value7
Key2 Value8
Key4 Value9
Key1 Value0
Key3 ValueA
Key5 ValueB
Key6 ValueC
Key8 ValueD
Shuffle and Sort
Key2 Value3
Value7
Value8
Key4 Value4
Value9
Key6 ValueC
Key8 Value5
ValueD
Key1 Value1
Value0
Key3 ValueA
Key5 ValueB
Key7 Value2
Reduce Task
Output Dataset


Map Reduce: Mapper → Reducer
The Map
The framework takes care of moving the data from mapper to the reducer: All values V_{inter} with same key K_{inter} are reduced together and processed in V_{inter} order
Input Dataset
Record
Split
RecordReader
Input <key,value> i.e <offset,record>
Map Task
Key1 Value1
Key7 Value2
Key2 Value3
Key4 Value4
Key8 Value5
Key3 Value6
Key2 Value7
Key2 Value8
Key4 Value9
Key1 Value0
Key3 ValueA
Key5 ValueB
Key6 ValueC
Key8 ValueD
Shuffle and Sort
Key1 Value1
Value0
Key3 ValueA
Key5 ValueB
Key7 Value2
Value9
Key6 ValueC
Key8 Value5
ValueD
Reduce Task
Output Dataset


Map Reduce: Reducer
Reducer: All V_{inter} with same key K_{inter} are reduced together and processed in V_{inter} order. There is one Reduce function call unique key K_{inter}.
The Reduce
Input Dataset
Record
Split
RecordReader
Input <key,value>i.e <offset,record>
Map Task
Key3 Value6
Key2 Value7
Key2 Value8
Key4 Value9
Key1 Value0
Key3 ValueA
Key5 ValueB
Key6 ValueC
Key8 ValueD
Shuffle and Sort
Key2 Value3
Value7
Value8
Key4 Value4
Value9
Key6 ValueC
Key8 Value5
ValueD
Key1 Value1
Value0
Key3 ValueA
Key5 ValueB
Key7 Value2
Reduce Task
Output Dataset

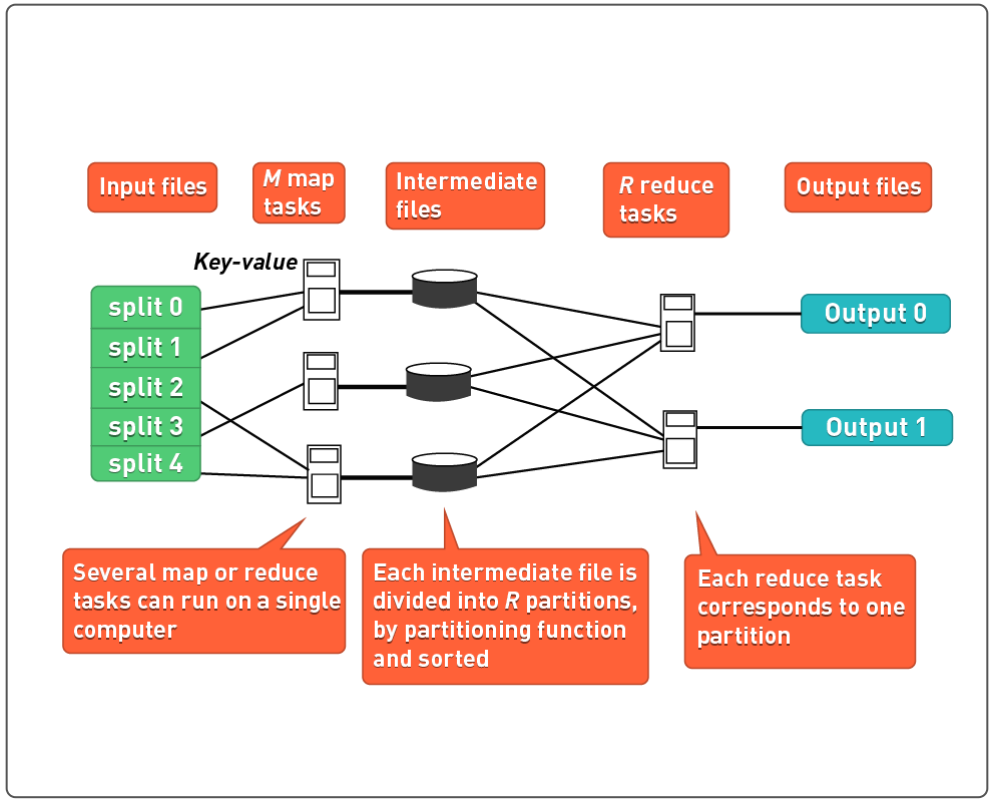
MapReduce: Example Sum(X^2)

1. .



MapReduce a Framework for Big Data

1. MapReduce codifies a generic recipe for processing large data sets that consist of two stages
    - In the first stage, a user-specified computation is applied over all input records in a data set
    - These operations occur in parallel and yield intermediate output that is then aggregated by another user-specified computation
2. Programmer and execution framework synergy
3. Just provide the mapper and reducer functions
    - The programmer defines these two types of computations, and the execution framework coordinates the actual processing (very loosely, MapReduce provides a functional abstraction)
4. Very powerful: Many interesting algorithms can be expressed quite concisely
    - Although such a two-stage processing structure may appear to be very restrictive, many interesting algorithms can be expressed quite concisely, especially if one decomposes complex algorithms into a sequence of MapReduce jobs


Input files
M map tasks
Intermediate files
R reduce tasks
Output files
Key-value
split 0
split 1
split 2
split 3
split 4
Output 0
Output 1
Several map or reduce tasks can run on a single computer
Each intermediate file is divided into R partitions, by partitioning function and sorted
Each reduce task corresponds to one partition

In Summary: Data Records Flow From Map to Reduce

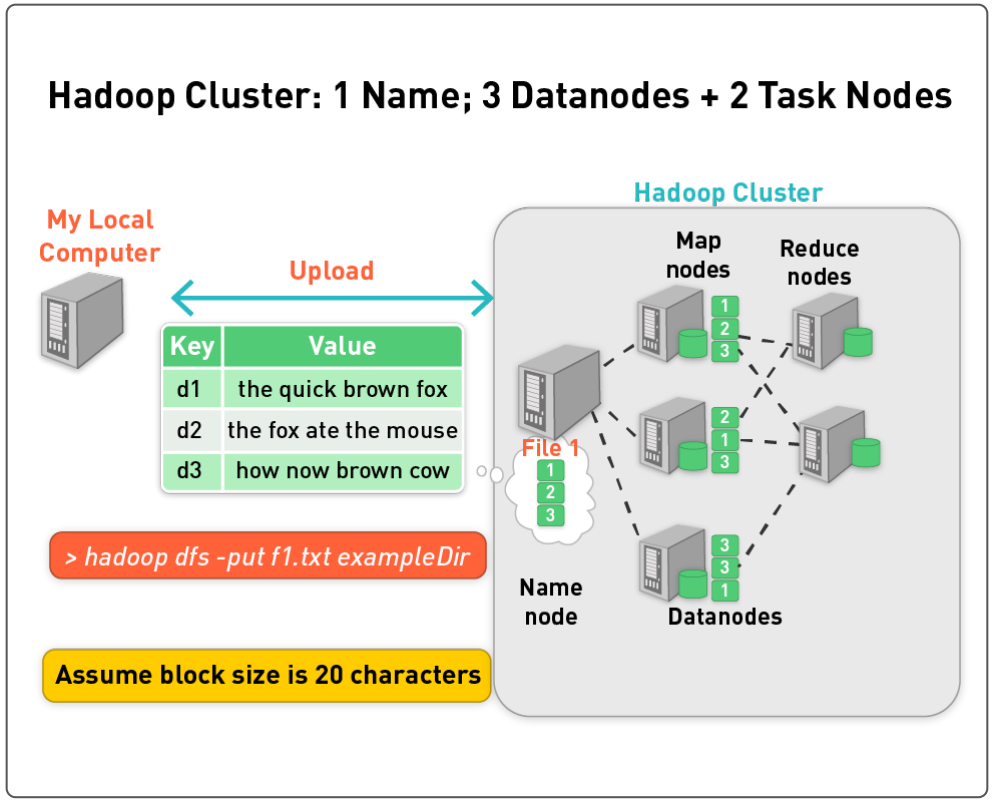
1. The framework will convert each record of input into a key-value pair.
    - The framework will convert each record of input into a key-value pair, and each pair will be input to the map function once.
2. The map output pairs are grouped and sorted by key.
    - The map output is a set of key-value pairs—nominally, one pair that is the transformed input pair, but it is perfectly acceptable to output multiple pairs.
3. The reduce function is called one time for each key, in sort sequence, with the key and the set of values that share that key.
4. Reduce outputs to file:
    - The reduce method may output an arbitrary number of key-value pairs, which are written to the output files in the job output directory.
    - If the reduce output keys are unchanged from the reduce input keys, the final output will be sorted.

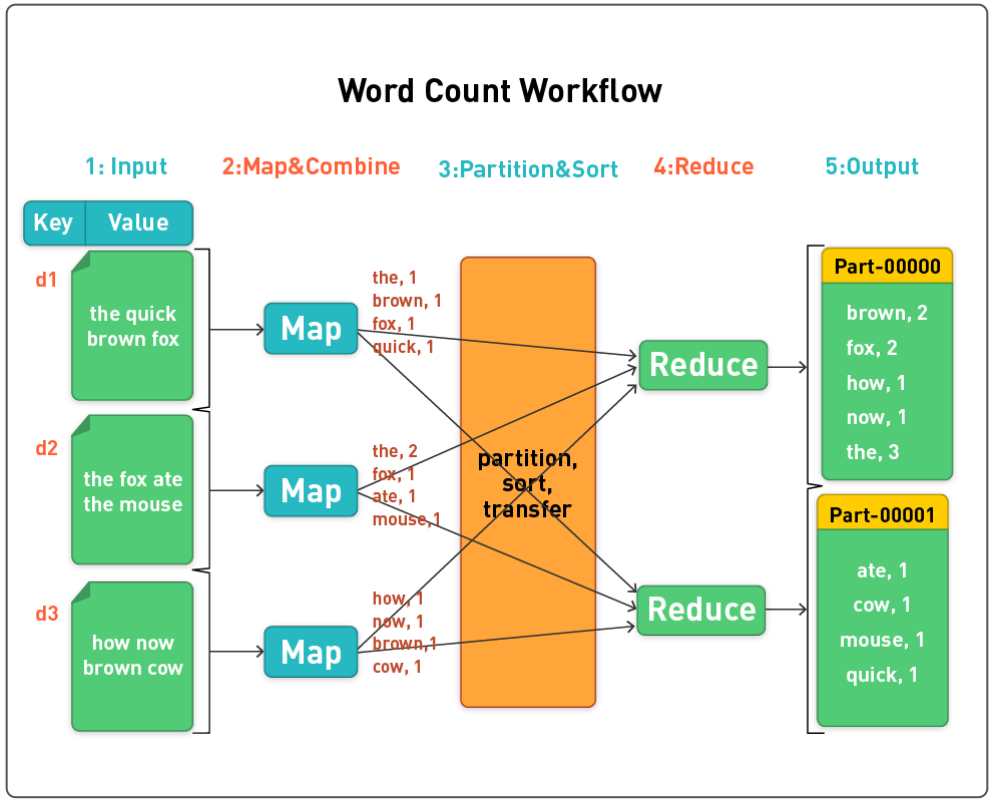
Conclusion

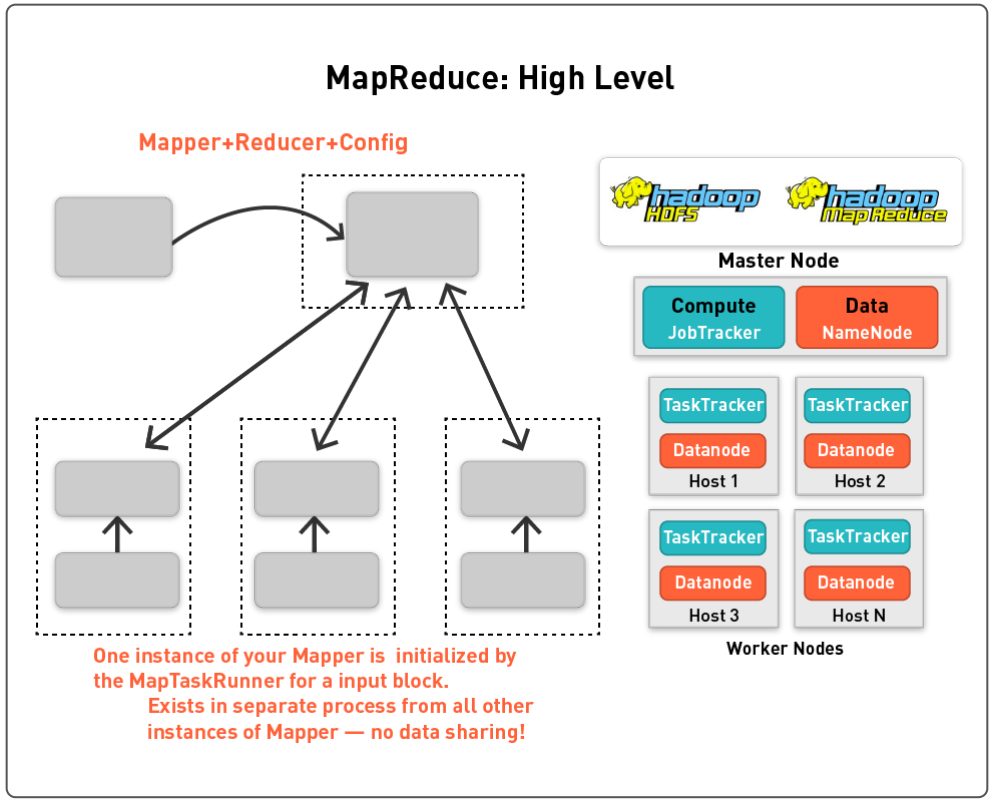
1. So that's MapReduce!
2. Aren't you excited about what you can do with MapReduce?
3. In the next section, we will get concrete and work through a couple of examples of using Hadoop MapReduce to solve problems.

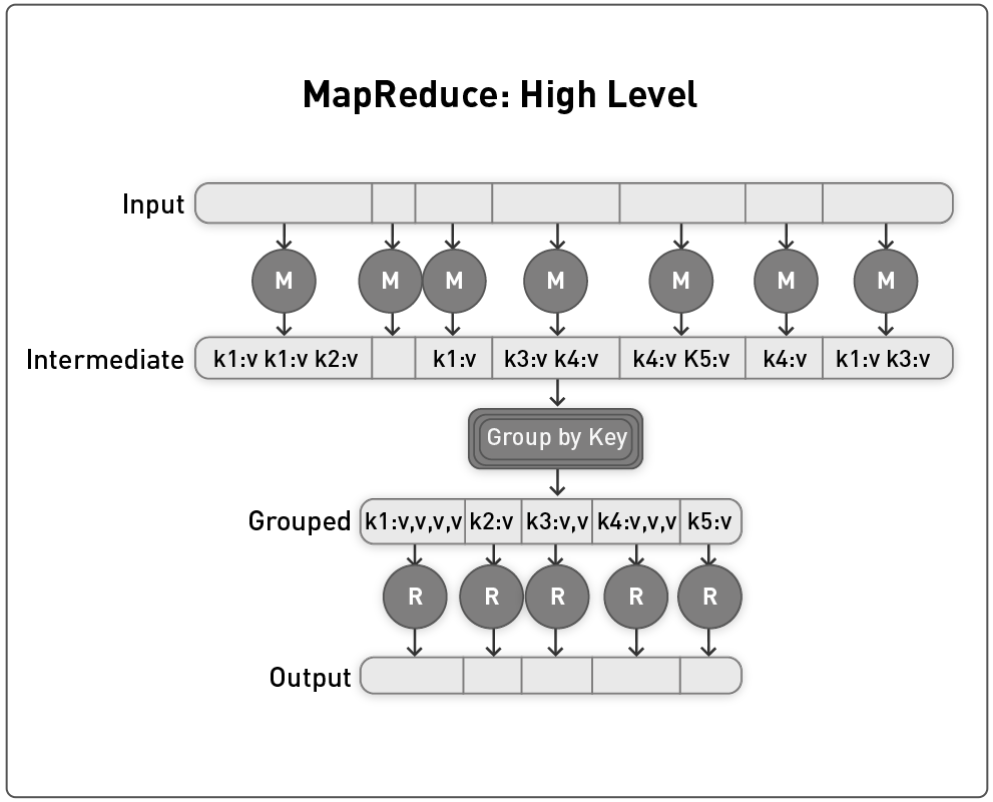
Agenda

- In this section, we will work through a couple of examples using Hadoop MapReduce to solve problems.
- First up is the word count example.
- The goal is to count the number of occurrences of each word in a bunch of input files.

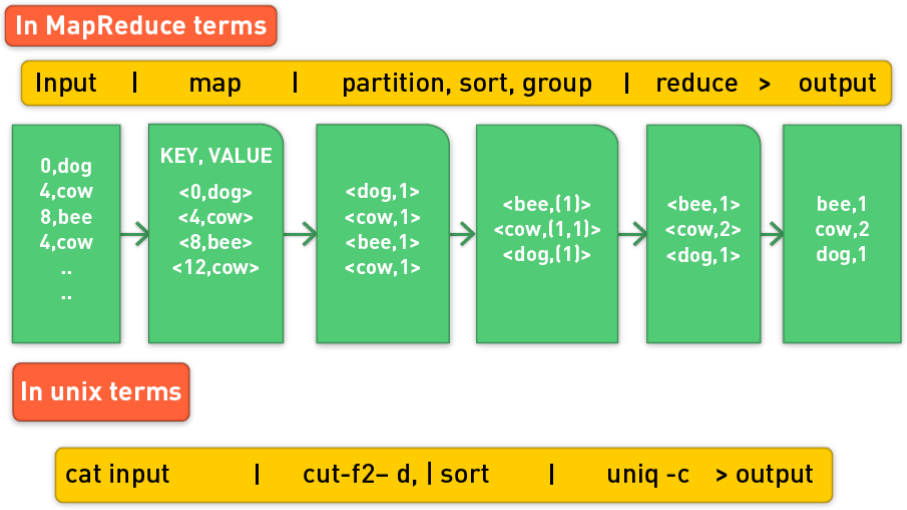

Hadoop Cluster: 1 Name; 3 Datanodes + 2 Task Nodes
My Local Computer
Upload
Key
Value
d1
the quick brown fox
d2
the fox ate the mouse
d3
how now brown cow
> hadoop dfs -put f1.txt exampleDir
Assume block size is 20 characters
Hadoop Cluster
Map nodes
Reduce nodes
File 1
Name node
Datanodes


Word Count Workflow
1: Input
2:Map&Combine
3:Partition&Sort
4:Reduce
5:Output
Key
Value
d1
the quick brown fox
d2
the fox ate the mouse
d3
how now brown cow
Map
Map
Map
the, 1
brown, 1
fox, 1
quick, 1
the, 2
fox, 1
ate, 1
mouse, 1
how, 1
now, 1
brown, 1
cow, 1
partition, sort, transfer
Reduce
Reduce
Part-00000
brown, 2
fox, 2
how, 1
now, 1
the, 3
Part-00001
ate, 1
cow, 1
mouse, 1
quick, 1


MapReduce: High Level
Mapper+Reducer+Config
Master Node
Compute
JobTracker
Data
NameNode
TaskTracker
Datanode
Host 1
TaskTracker
Datanode
Host 2
TaskTracker
Datanode
Host 3
TaskTracker
Datanode
Host N
Worker Nodes
One instance of your Mapper is initialized by the MapTaskRunner for a input block.
Exists in separate process from all other instances of Mapper — no data sharing!


MapReduce: High Level
Input
M
M
M
M
M
M
M
Intermediate
k1:v k1:v k2:v
k1:v
k3:v k4:v
k4:v K5:v
k4:v
k1:v k3:v
Group by Key
Grouped
k1:v,v,v,v
k2:v
k3:v,v
k4:v,v,v
k5:v
R
R
R
R
R
Output

MapReduce: Example 2, Word Count from a Word Stream



Example: Word Count Pseudo-Code

```
def mapper(line):
    foreach word in line.split():
        output(word, 1)
def reducer(key, values):
    output(key, sum(values))
```

Example: Word Count

Key	Value
d1	the quick brown fox
d2	the fox ate the mouse
d3	how now brown cow

```
def mapper(line):
    foreach word in line.split():
        output(word, 1)
```

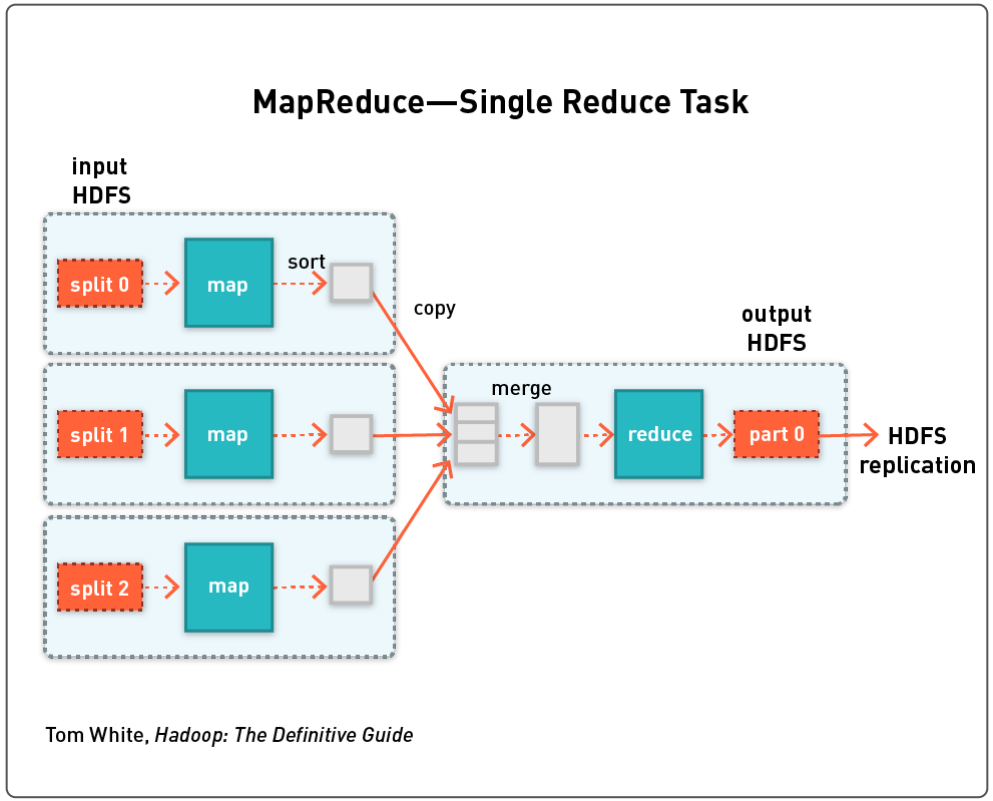
partition, sort, transfer	('brown', 1) ('brown', 1) ('brown', (1, 1))

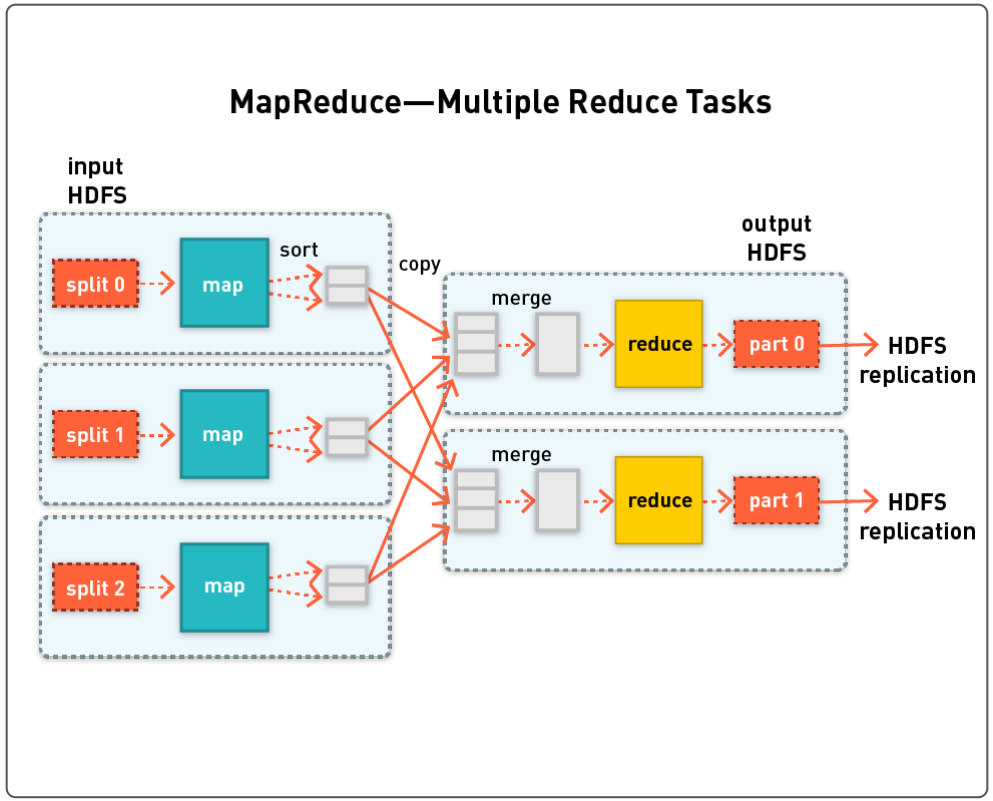
```
def reducer(key, values):
    output(key, sum(values))
```

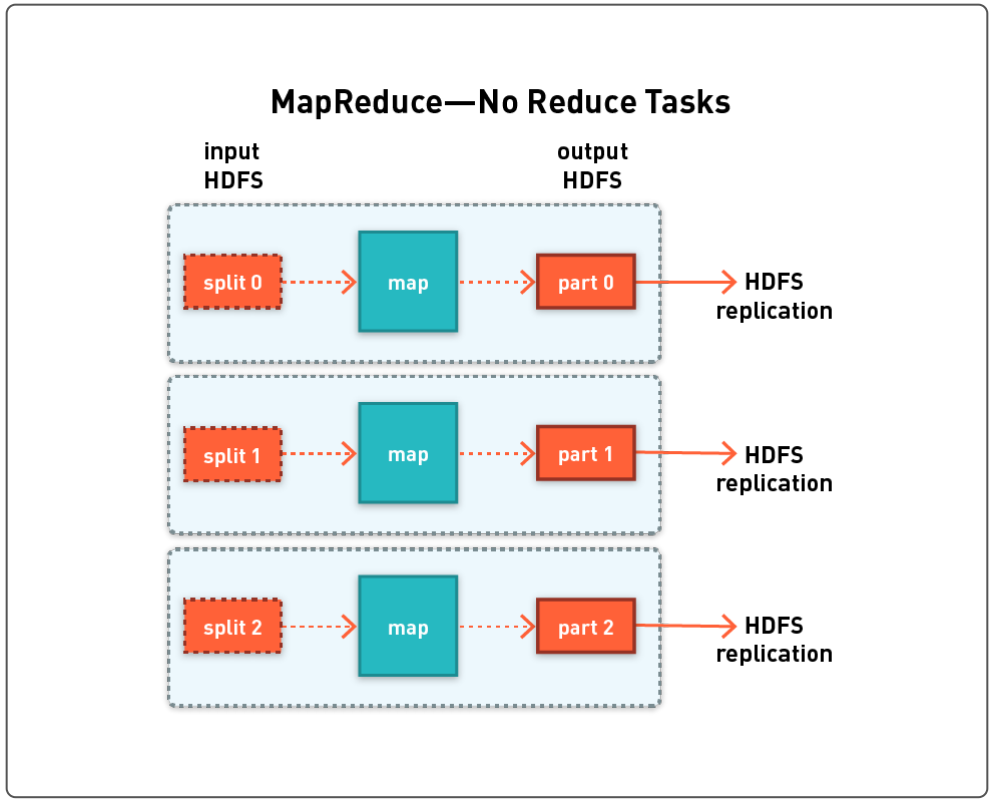
('brown', 2)
.....

```
public static class MapClass extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, IntWritable> {

    private final static inWritable ONE = new  IntWritable(1);

    public void map(LongWritable key, Text value,
                    OutputCollector<Text, InWrtable> output,
                    Reporter reporter) throws   IOException  {
        String line = value.toString();
        StringTokenizer   itr  = new  StringTokenizer(line);
        while (itr .hasMoreTokens())  {
          output.collect(new text(itr .nextToken()), ONE);
        }
    }
 }
```

```
public static class Reduce extends MapReduceBase
      implements Reducer<Text, IntWritable, Text, IntWritable> {

      public void reduce(Text key, Iterator<Inwritable> values,
                         OutputCollector<Text, InWritable> output,
                         Reporter reporter) throws IOExpection {
        int sum = 0;
        while (values, hasNext()) {
            sum += values.next().get();
         }
         output.collect(key, new IntWritable(sum));
      }
  }
```


MapReduce—Single Reduce Task
input
HDFS
split 0
map
sort
split 1
map
split 2
map
copy
merge
output
HDFS
reduce
part 0
HDFS
replication
Tom White, Hadoop: The Definitive Guide


MapReduce—Multiple Reduce Tasks
input
HDFS
split 0
map
sort
split 1
map
split 2
map
copy
output
HDFS
merge
reduce
part 0
HDFS
replication
merge
reduce
part 1
HDFS
replication


MapReduce—No Reduce Tasks
input
HDFS
output
HDFS
split 0
map
part 0
HDFS
replication
split 1
map
part 1
HDFS
replication
split 2
map
part 2
HDFS
replication

Contract With the Programmer

- The frozen part of the MapReduce framework is a large distributed sort. The hot spots, which the application defines, are:

An input reader

A map function*

A partition function

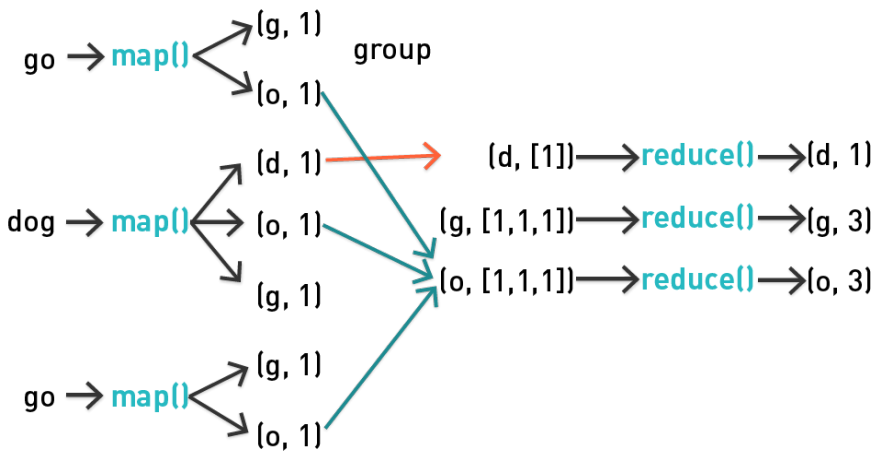
A compare function

A reduce function*

An output writer

**Required from programmer (generally)*

Character Counter



Write the MapReduce code to count the number of times characters appear in the input data.

Hadoop: Native Java or Streaming

- Although the Hadoop framework is implemented in Java™, MapReduce applications need not be written in Java.
- Hadoop provides an API to MapReduce that allows you to write your map and reduce functions in languages other than Java (e.g., Python, Ruby, Perl, etc.).
    - Hadoop Streaming uses Unix standard streams as the interface between Hadoop and your program, so you can use any language that can read standard input and write to standard output to write your MapReduce program.
    - Hadoop Streaming is a utility that allows users to create and run jobs with any executables (e.g., shell utilities) as the mapper or the reducer.

Word Count in Python With Hadoop Streaming

The prototypical MapReduce example counts the appearance of each word in a set of documents.

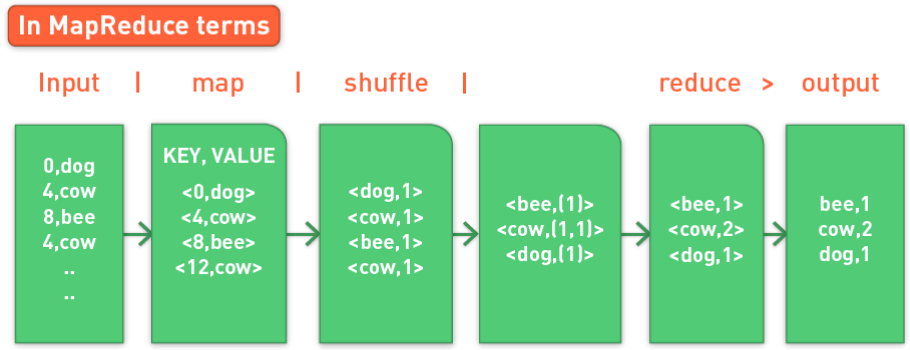
1: Input	Doc 1, the quick brown fox Doc 2, the fox ate the mouse Doc 3, how now brown cow
2: Map	the, 1 quick, 1 brown, 1 fox, 1 the, 1 fox, 1 …

```
function map(String name, String document):
  // name: document name
  // document: document contents
  for each word w in document:
    emit (w, 1)

function reduce (String word, Iterator partialCounts):
  // word: a word
  // partialCounts: a list of aggregated partial counts
  sum = 0
  for each pc in partialCounts:
    sum += ParseInt(pc)
  emit (word, sum)
```

Hadoop Streaming in Python

- Hadoop Streaming is a utility that comes with the Hadoop distribution.
- The utility allows you to create and run MapReduce jobs with any executable or script as the mapper or reducer.
- For example:
    - `$HADOOP_Home/bin/hadoop jar`
    - `$HADOOP_Home/hadoop-streaming.jar\`
        - `input myInputDirs \`
        - `output myOutputDir \`
        - `mapper wordCountMapper.py \`
        - `reducer wordCountReducer.py`

MapReduce: Word Count of Word Stream



Unit Test

1: cat input.txt | mapper.py

2: cat input.txt | mapper.py | sort

3: cat input.txt | mapper.py | sort | reducer.py

```
$HADOOP_HOME/Bin/hadoop jar $HADOOP_HOME/hadoop-streaming.jar
    -input myInputDirs -output myOutputDir \
    -mapper wordCountMapper.py -reducer wordCountReducer.py
```

Parallel Computing

- Motivation for parallel computing
- Parallel computing (PC)
  - Definition, communication and synchronization, types of PC tasks
  - Architectures for parallel computation
  - Developer frameworks for parallel computation

MapReduce and Hadoop

- Hadoop
  - Background and history
  - Hadoop Distributed File System (HDFS)
  - MapReduce
    - Functional programming
    - MapReduce
    - Animated examples
    - Hadoop in practice
- Full-code examples
  - Word count example on local machine
  - Word count example on cluster
- MapReduce: Runtime environment

Hadoop 2.0

- Hadoop 2.0 and what is Hadoop good at?
- Sync time
  - Install Hadoop; run locally; run in the cloud?
  - Ten most popular words
  - What is this? That?
  - Naive Bayes in poor man's versions Python and shell commands

Happy Hadooping!