EE4221
Cloud Computing Systems

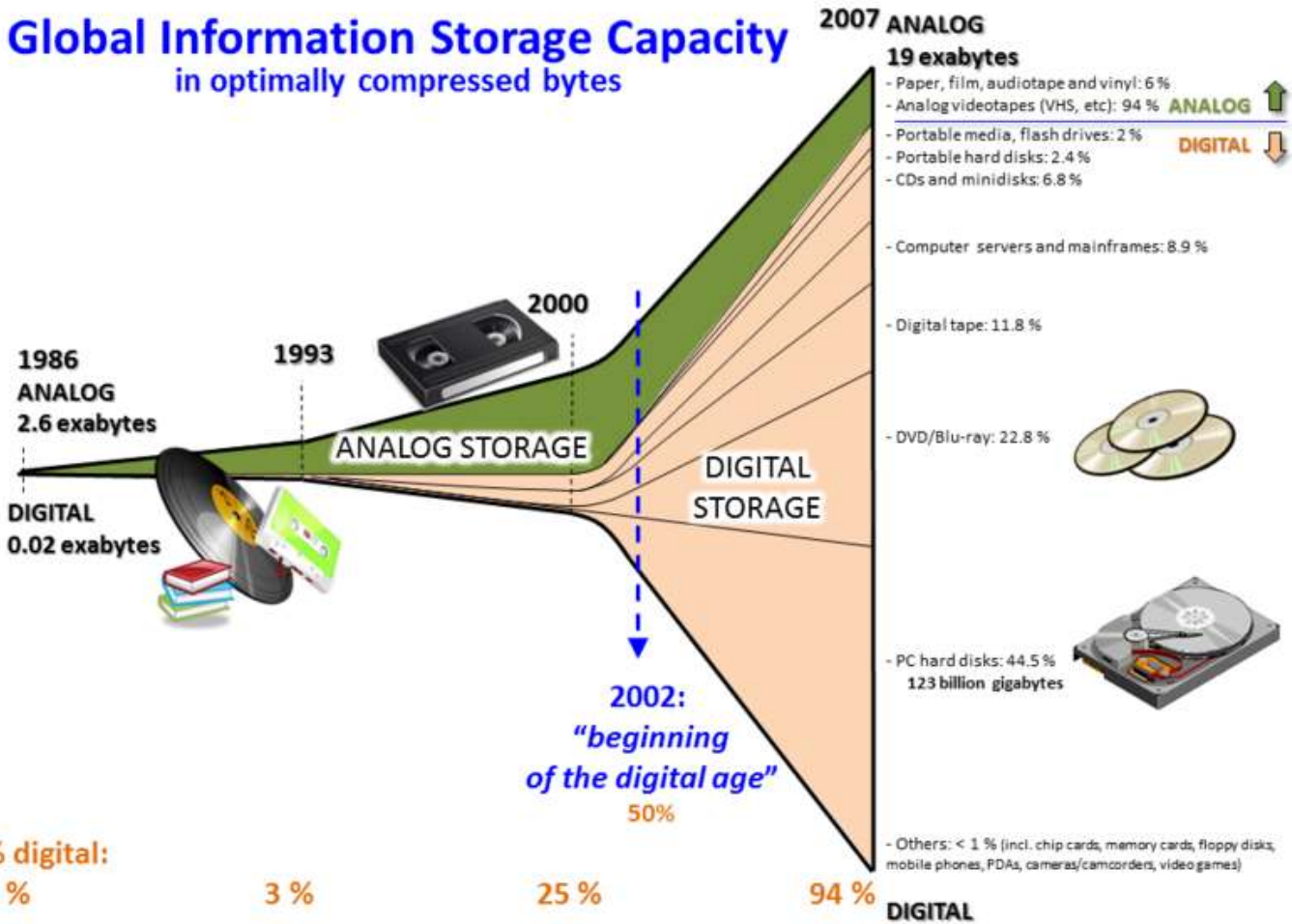
Parallel Computing and MapReduce

Eric Wong
City University of Hong Kong

Outlines

- Big Data
- Distributed Parallel computing
- Apache Hadoop
  - Hadoop Distributed File System (HDFS)
  - HDFS vs NFS (Network File System)
  - Hadoop Cluster
    - Node types: Client Node, Name Node, Data Node
    - Tracker types: Job Tracker, Task Tracker
- MapReduce

Big Data

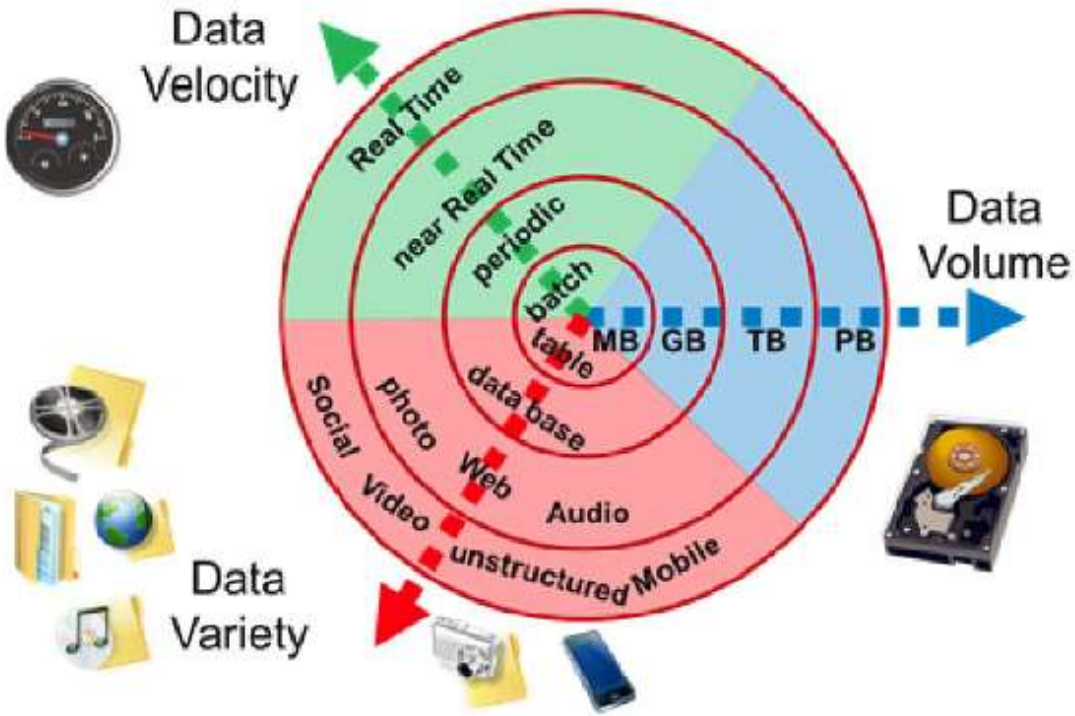


Source: Hilbert, M., & López, P. (2011). The World's Technological Capacity to Store, Communicate, and Compute Information. *Science*, 332(6025), 60 –65. http://www.martinhilbert.net/WorldInfoCapacity.html

Three Vs for Big Data

- Volume: Large amount – “how does a person eat 100 burgers in one day?”
- Velocity: Quickly moving – “how does a person drink water from a running fire hose?”
- Variety: Many forms: structured, unstructured, texts, images, etc. – “how does a person master 100 different languages?”

It is just too much, too fast and too different so we need a new way to handle Big Data!!



The growth of big data's primary characteristics of volume, velocity, and variety.

Source: https://en.wikipedia.org/wiki/Big_data

How is Big Data Different?

Traditional Data

- Large scale
- Highly centralized
- Structured
  - Files
  - Records
  - Databases
- Sequential
- Indexed
- **Processing transactions**

Big Data

- Massive scale
- Highly distributed
- Unstructured
  - Emails
  - Audio/Video
  - Documents
  - Spreadsheets
  - Log files
  - Sensor data
  - Geo-spatial data
  - Books
  - Journals
  - Blogs
  - Text messages
  - Chat sessions
  - Search data
- Random
- **Looking for patterns and relationships**

Source: http://www.slideshare.net/

Dealing with Structured Data

- Very large databases stored in a central location – data warehouse
- Attached to very large, very powerful computers
- Accessed by structured queries
- Continually updated
- Used for "real time" transactional processing
- Reports created by a "batch" process

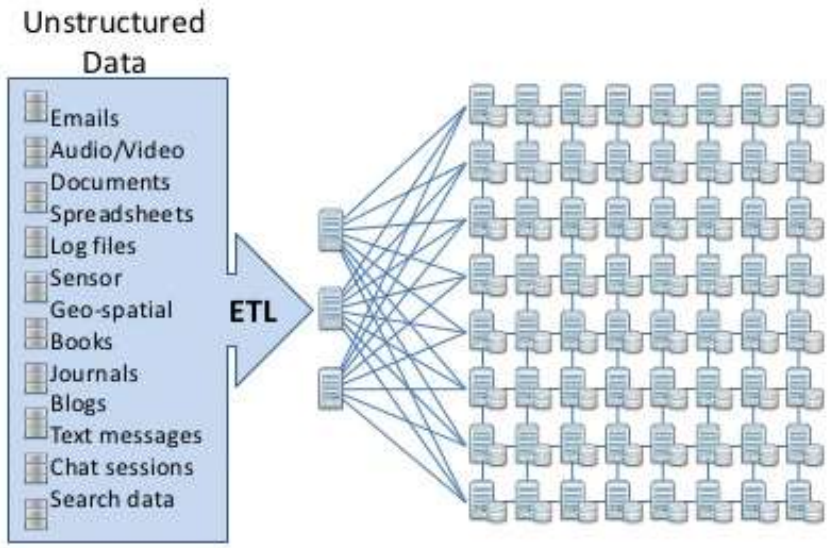
Source: http://www.slideshare.net/

Traditional Centralized approach for Big Data

- Finite resources for computing hardware
  - Processor, memory, storage, network bandwidth
- With web-scale big data e.g. 100TBs
  - No single machine can hold it in RAM to process
  - Intermediate data sets generated need several times more space
  - I/O speed is around 60-100MB/s, takes 3 hours to load 1TB dataset
  - Quickly saturate the switch's bandwidth capacity
  - Quickly generate too many synchronization requests, causing buffer overflow
  - Deadlock or race conditions among nodes of a cluster
- We need a new way approach for Big Data => Distributed computing

Dealing with Big Data

- Unstructured data retrieved from variety of sources
- Data is **E**xtracted, **T**ranslated and **L**oaded into Big Data system
- Small parts of data are distributed by master nodes to hundreds (thousands) of small networked nodes
- Each node processes a part of the data and returns an answer
  - MapReduce
- Process is repeated until all data is analyzed
- Results are then used for further analysis



Source: http://www.slideshare.net/
What is Hadoop?: SQL Comparison https://www.youtube.com/watch?v=MfF750YVDxM

Distributed Parallel computing

Problems in Distributed Systems

- Networks can experience partial or total failure if switches and routers break down.
- Data may not arrive at a particular point in time due to unexpected network congestion.
- Individual compute nodes may overheat, crash, experience hard drive failures, or run out of memory or disk space.
- Data may be corrupted, or maliciously or improperly transmitted.
- Multiple implementations or versions of client software may speak slightly different protocols from one another.
- Clocks may become desynchronized, lock files may not be released, parties involved in distributed atomic transactions may lose their network connections part-way through, etc.

Solutions

- In each of these cases, the rest of the distributed system should be able to recover from the component failure or transient error condition and continue to make progress.
- Actually, providing such resilience is a major software engineering challenge!

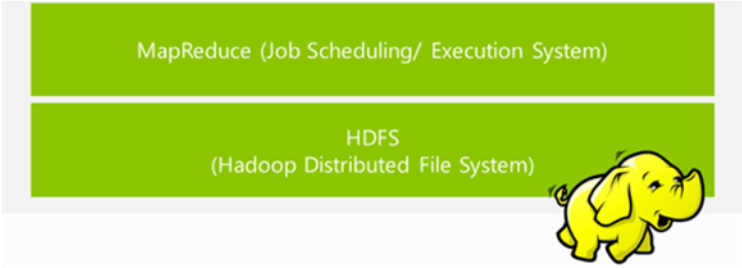
Issues in Distributed Parallel computing

- Issue 1: High Hardware Failure Rate
  - When using many computers for data storage and analysis, the probability that one computer will fail is very high.
  - The rest of the distributed system should be able to recover from the component failure or transient error condition and continue to work.
- Issue 2: High Cost of Backup
  - To avoid data loss or corrupted analysis information loss, using backup computers and memory is needed, which helps the reliability, but is very expensive.
- Issue 3: Difficult to Combine Analyzed Data
  - If one part of the analyzed data is not ready, then the overall combining process has to be delayed.
  - If one part has errors in its analysis, then the overall combined result may be unreliable and useless.

Apache Hadoop

Apache Hadoop

- Hadoop is an Apache open source framework written in Java that allows distributed processing of large datasets (hundreds of GBs to PBs) across clusters of computers using simple programming models.
  - For example, a Hadoop cluster with 20,000 inexpensive commodity servers and 256MB block of data in each, can process around 5TB of data at the same time.
- Hadoop Core = Hadoop Distributed File System (HDFS) + MapReduce
- Data Distribution
  - Split large data files into chunks
  - Replicate each chunk over several machines (data redundancy)
  - High data locality by moving computation to the data
- Isolated Processes
  - No explicit communications by users
  - Implicit communication via Map-Reduce model (more reliable)



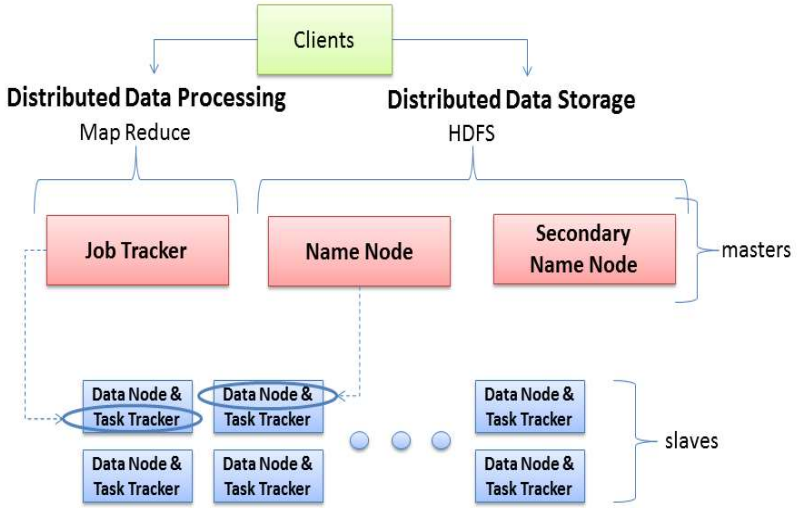
- Scalability
  - Easily scale from 10 to 1000 nodes with minimal re-work required for your applications.
  - Hadoop auto-manages the data and hardware resources and provides dependable performance growth proportionate to the number of machines available.

NFS vs HDFS

- Network File System (NFS) is designed to provide remote access to a single logical volume stored on a single machine
  - Limited storage capacity
  - No reliability guarantees if that machine goes down
  - All clients retrieve data from same source, overloading the server -> low throughput
- Hadoop Distributed File System (HDFS) is designed to:
  - Hold vast amounts of data (terabytes or even petabytes)
    - Spread the data across a large number of machines. It also supports much larger file sizes than NFS.
  - Store data reliably
    - Files are stored redundantly across multiple machines to ensure their durability to failure of individual machine and high availability to every parallel applications.
  - Provide high-throughput and scalability
    - Serve a larger number of clients by simply adding more machines to the cluster.
    - Applications that use HDFS are assumed to perform long sequential streaming reads from files (not general purpose in which random access is more common).
  - Maximize data locality
    - Integrate well with Hadoop MapReduce and allow data to be read and computed upon locally when possible.

Three Node Types in Hadoop Cluster

- Client Node
  - Loads data into the cluster
  - Submits MapReduce jobs describing how that data should be processed
  - Retrieves or views the results of the job when processing is finished

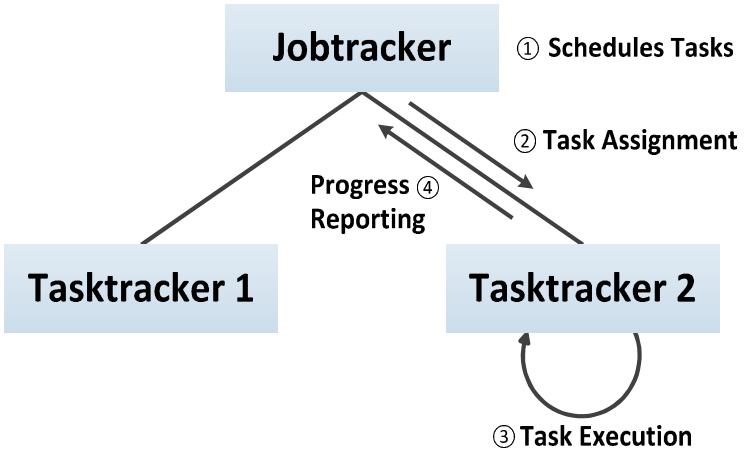


- Name Node (run on a master)
  - Manages the filesystem namespace
  - Build metadata in memory and synchronize with all clients for any updates
  - Track file names, permissions and locations of each block of each file

- Data Node (run on a slave)
  - The workhorse of the filesystem
  - Send heartbeats to the name node every 3 seconds
  - Send block report to the name node every 10th heartbeat
  - Store and retrieve blocks (default data block size is 64Mb) when requested by the client or the name node

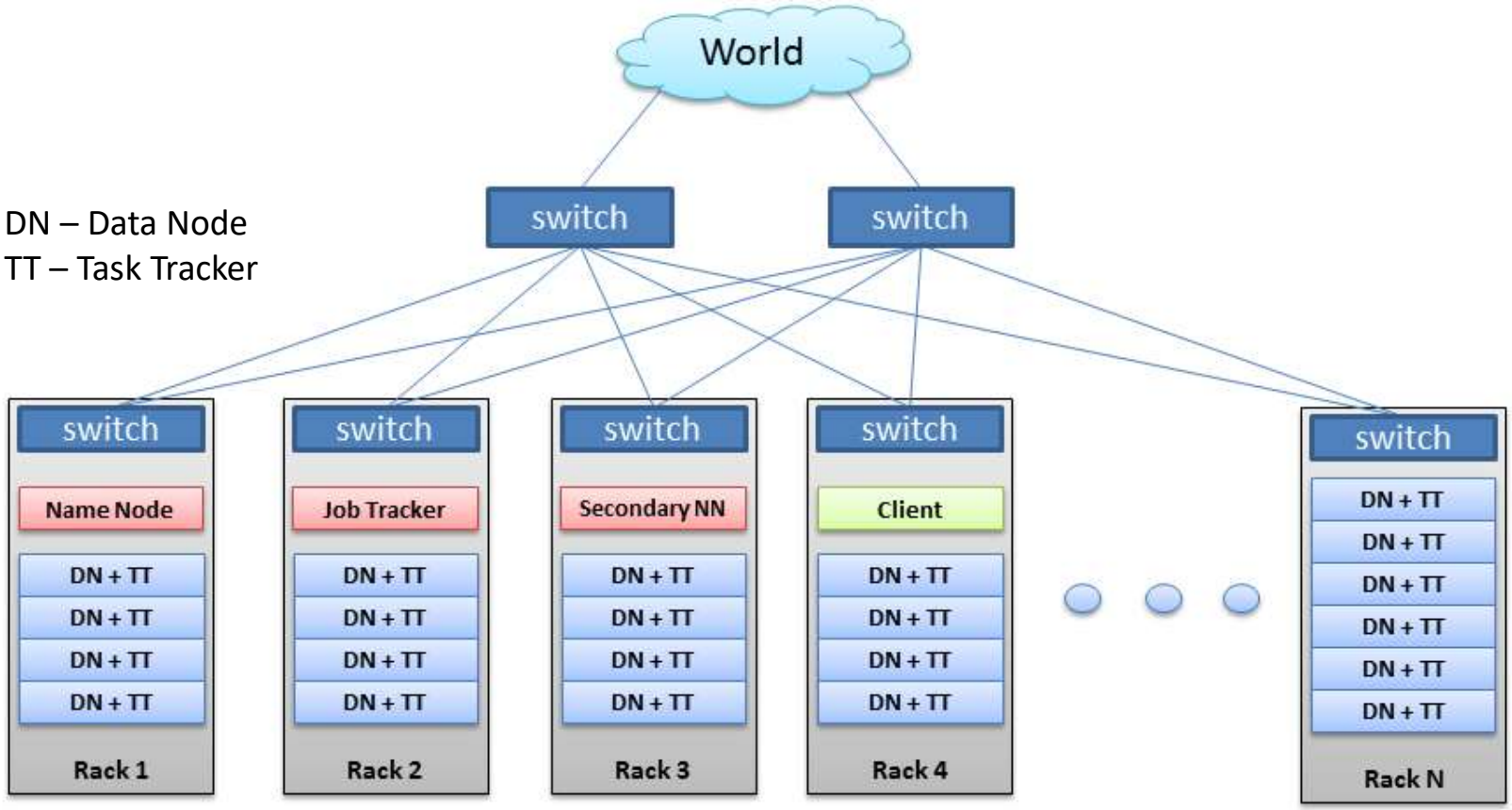
Two Tracker Types in Hadoop Cluster: Job Tracker and Task Tracker

- A Job Tracker is responsible for scheduling execution of Tasks on appropriate nodes, coordinating the execution of Tasks, sending the information for the execution of Tasks, getting the results back after the execution of each task, re-executing the failed Tasks, and monitors/maintains the overall progress of the Job.
  - There is only one JobTracker node per Hadoop Cluster.
- A Task Tracker is responsible for executing a Task assigned to it by the JobTracker. TaskTracker receives the information necessary for execution of a Task from JobTracker, executes the Task, and sends the results back to JobTracker.
  - No restriction on the number of TaskTracker nodes in a Hadoop Cluster.
- They are also works in a master-slave fashion

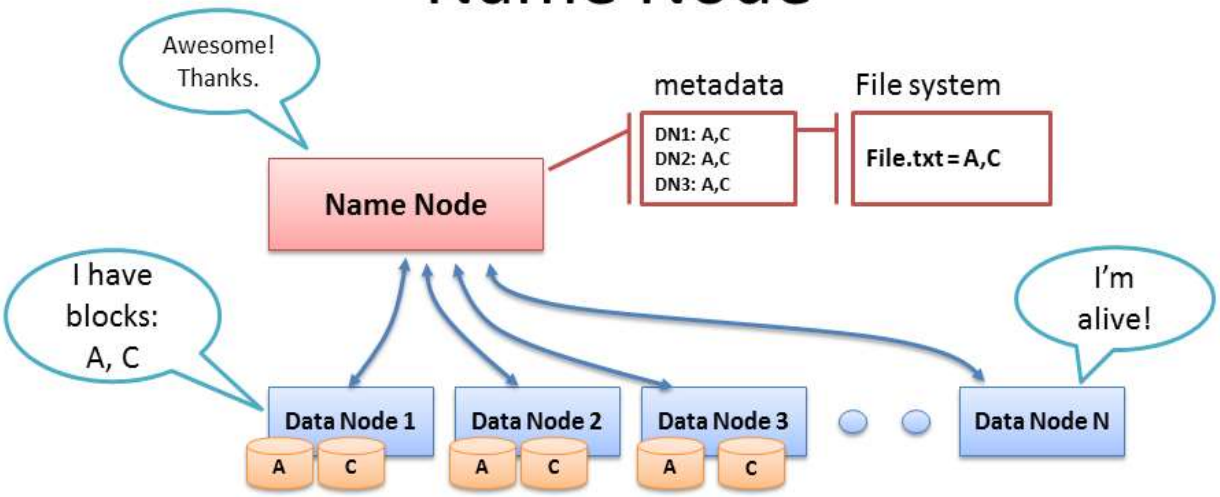


Hadoop Cluster

DN – Data Node
TT – Task Tracker

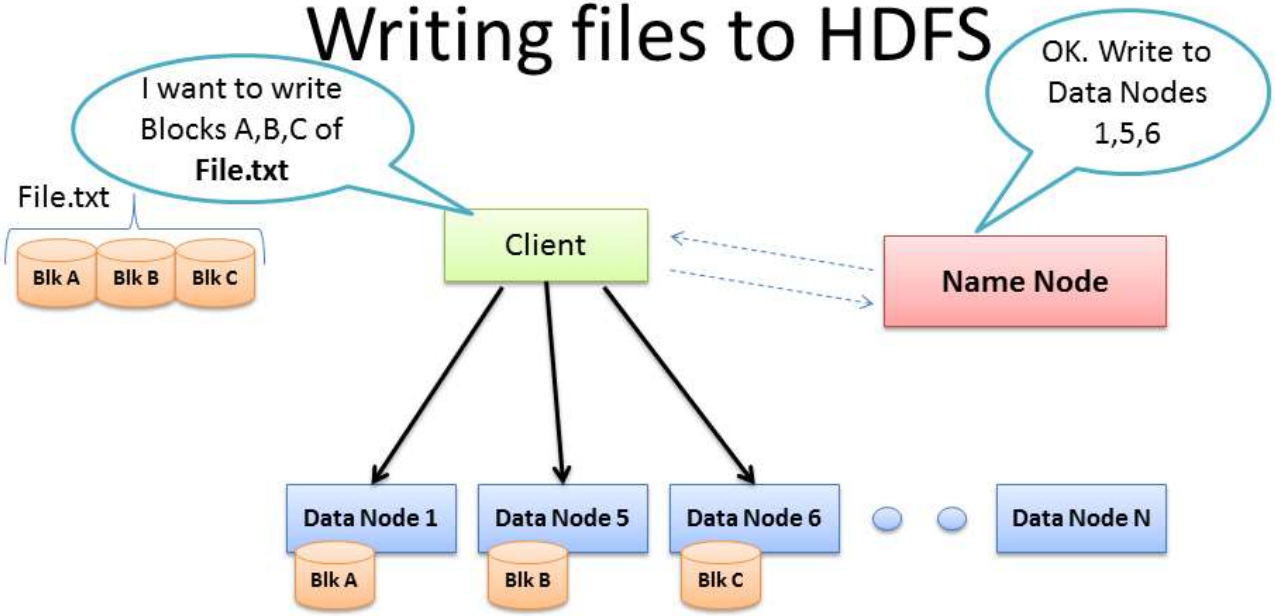


Name Node



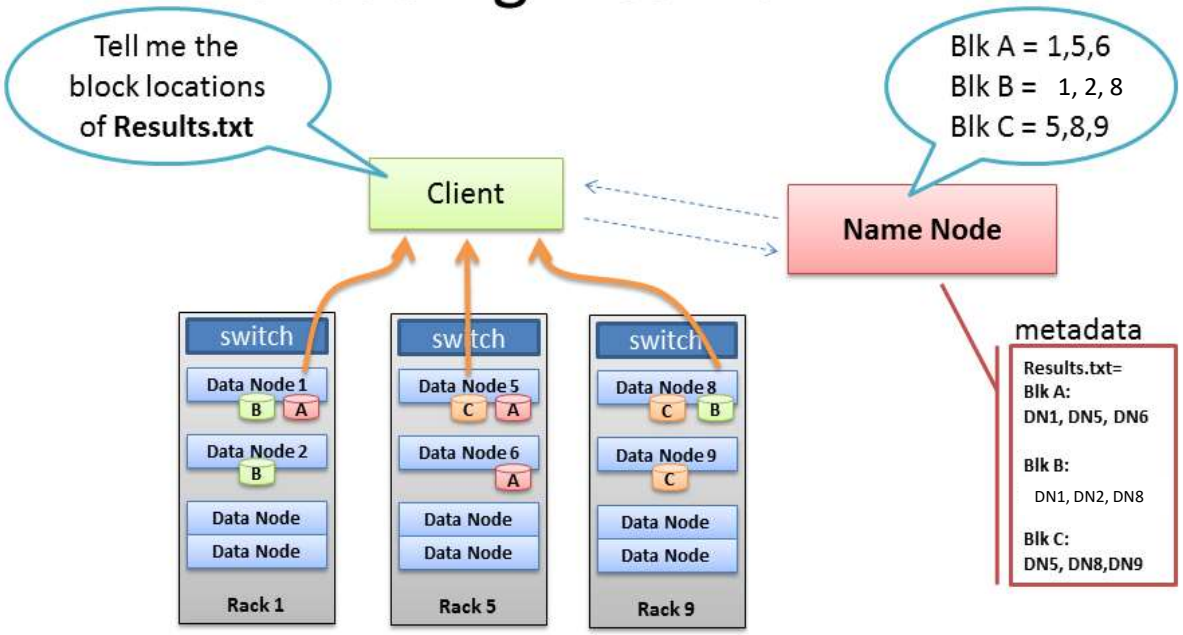
- Data Node sends Heartbeats
- Every 10th heartbeat is a Block report
- Name Node builds metadata from Block reports
- TCP – every 3 seconds
- If Name Node is down, HDFS is down

Writing files to HDFS



- Client consults Name Node
- Client writes block directly to one Data Node
- Data Nodes replicates block
- Cycle repeats for next block

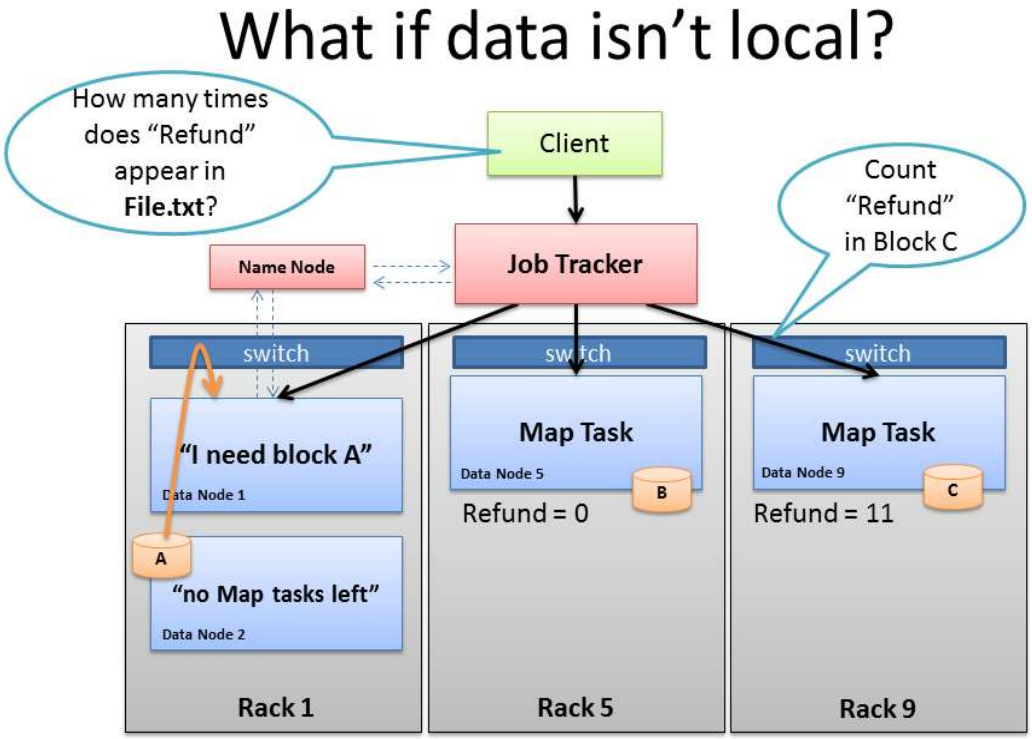
Client reading files from HDFS



- Client receives Data Node list for each block
- Client picks first Data Node for each block
- Client reads blocks sequentially

Data Locality

- MapReduce tries to place the data and the computer as close as possible. Job Tracker will always try to pick nodes with local data for a Map task.
- It may not always be able to do so. One reason for this might be that all of the nodes with local data already have too many other tasks running and cannot accept anymore.
- In this case, the Job Tracker will consult the Name Node whose Rack Awareness knowledge can suggest other nodes in the same rack.
- The Job Tracker will assign the task to a node in the same rack, and when that node goes to find the data it needs, the Name Node will instruct it to grab the data from another node in its rack, leveraging the presumed single hop and high bandwidth of in-rack switching.



- Job Tracker tries to select Node in same rack as data
- Name Node rack awareness

MapReduce

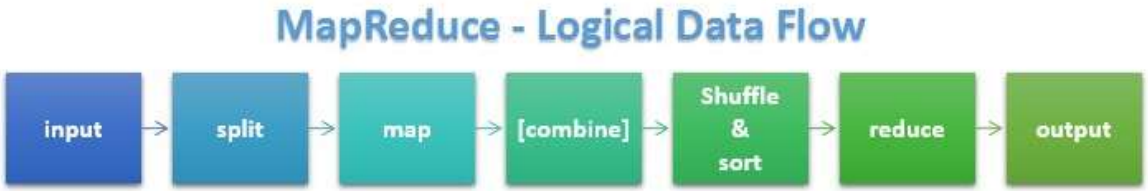
MapReduce

- MapReduce is a pattern of parallel functional programming that has been very successful in enabling "big data" computations.
- The Map and Reduce functions of MapReduce are both defined with respect to data structured in (key, value) pairs.
- The Map function takes input from the disk as key-value pairs in one domain, processes them, and produces a set of intermediate key-value pairs in another domain.
  - `(kA,vA)` → `{(kA1,vA1), (kA2,vA2),....}`
- The Shuffle and Sort function groups/redistributes intermediate key-value pairs with the same key, such that all pairs belonging to one key are located on the same worker node. This function is implicitly done by the MapReduce framework.
  - `(k1,{vA1,vB1,....}) for kA1 = kB1 = … = k1`
- The Reduce function takes a grouped key-value pair *and* reduces all those values associated with the same key to obtain a single output key-value pair.
  - `(k1,{vA1,vB1,....})` → `(k1, v1)`
- MapReduce allows for the distributed processing of the map and reduce operations. Both Map and Reduce functions can be performed in parallel, provided that the operations are independent of the others.

Implicit Communication Model

- When works are divided across a large number of machines, the communication overhead required to keep the data on the nodes synchronized at all times would prevent the system from performing reliably or efficiently at large scale.
- All data elements in MapReduce are **immutable** – cannot be updated during the process.
  - Modifications to data are not reflected in input files
  - Communication occurs only by generating new output (key, value) pairs which are then forwarded by the Hadoop system into the next phase of execution.

MapReduce Word Count Example



- For the purpose of understanding MapReduce, let us consider a simple example. Let us assume that we have a file which contains the following four lines of text.
- In this file, we need to count the number of occurrences of each word. For instance, DW x2, BI x1, SSRS x2, and so on. Let us see how this counting operation is performed when this file is input to MapReduce.

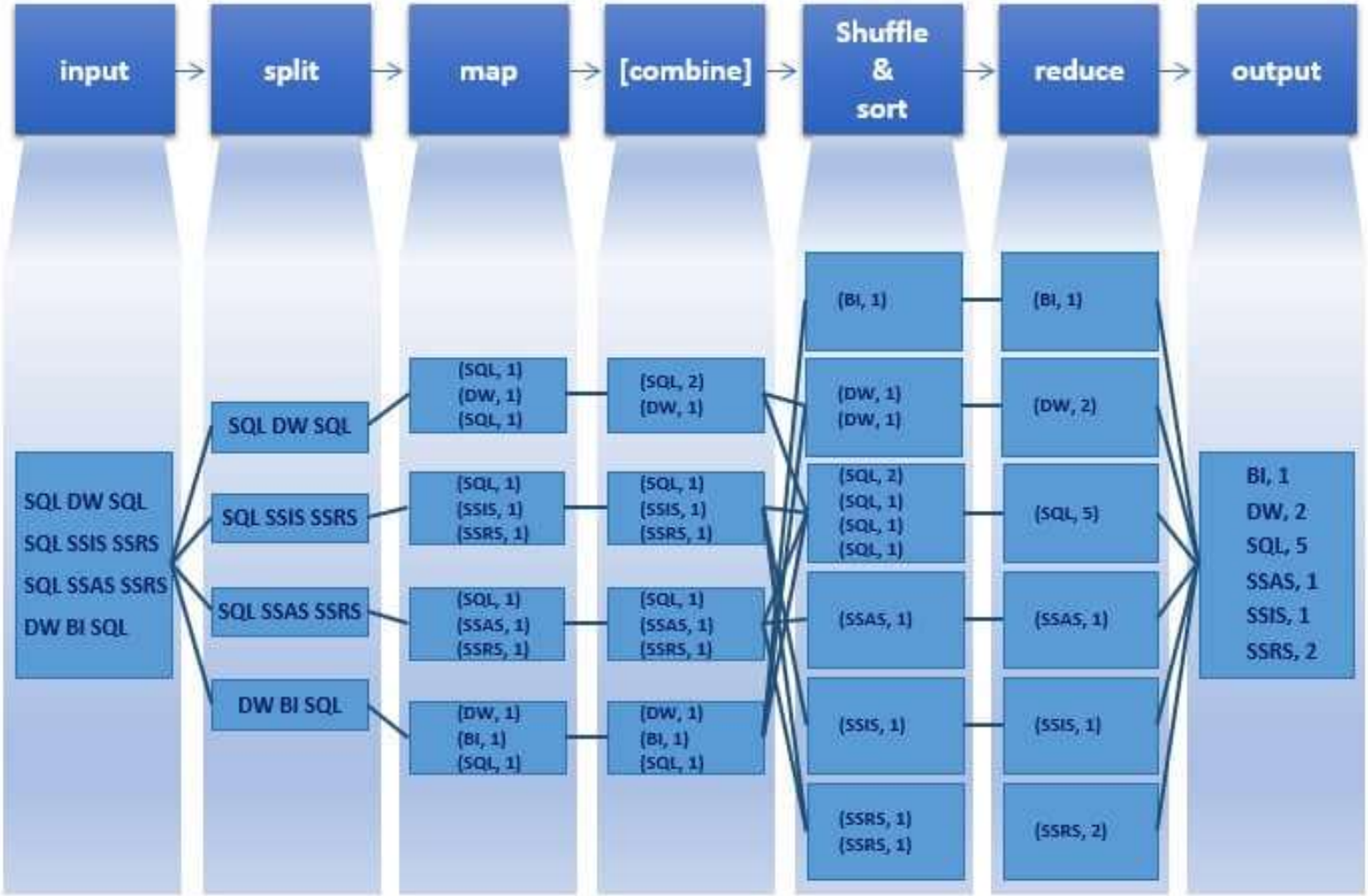
```
SQL DW SQL
SQL SSIS SSRS
SQL SSAS SSRS
DW BI SQL
```

MapReduce Word Count Example

- **Input**: In this step, the sample file is input to MapReduce.
- **Split**: In this step, Hadoop splits / divides our sample input file into four parts, each part made up of one line from the input file. Note that, for the purpose of this example, we are considering one line as each split. However, this is not necessarily true in a real scenario.
- **Map**: In this step, each split is fed to a mapper which is the map() function containing the logic on how to process the input data, which in our case is the line of text present in the split. For our scenario, the map() function would contain the logic to count the occurrence of each word and each occurrence is captured / arranged as a (key, value) pair, which in our case is like (SQL, 1), (DW, 1), (SQL, 1), and so on.
- **Combine**: This is an optional step and is often used to improve the performance by reducing the amount of data transferred across the network. This is essentially the same as the reducer (reduce() function) and acts on output from each mapper. In our example, the key value pairs from first mapper "(SQL, 1), (DW, 1), (SQL, 1)" are combined and the output of the corresponding combiner becomes "(SQL, 2), (DW, 1)".
- **Shuffle and Sort**: In this step, output of all the mappers is collected, shuffled, and sorted and arranged to be sent to reducer.
- **Reduce**: In this step, the collective data from various mappers, after being shuffled and sorted, is combined / aggregated and the word counts are produced as (key, value) pairs like (BI, 1), (DW, 2), (SQL, 5), and so on.
- **Output**: In this step, the output of the reducer is written to a file on HDFS. The following image is the output of our word count example.

Video: Learn MapReduce with Playing Cards in 10 Minutes https://youtu.be/bcjSe0xCHbE & Map Reduce explained with example in 9 Minutes https://www.youtube.com/watch?v=cHGaQz0E7AU&t=489s

MapReduce – Word Count Example Flow



Limitations of Hadoop MapReduce

- MapReduce cluster computing paradigm forces a particular linear dataflow structure on distributed programs:
  - Programs read input data from disk, map a function across the data, reduce the results of the map, and store reduction results on disk.
  - It is not suitable for iterative computations (common for data analysis), where output of one algorithm needs to be passed to another algorithm.
  - To run iterative jobs, we need a sequence of MapReduce jobs where the output of one step needs to be stored in HDFS (slow write-to-disk speed) before it can be passed to the next step. And the next step cannot be invoked until the previous step has completed (no pipelining between phases).