Models and Systems for Big Data

Map Reduce & MongoDB

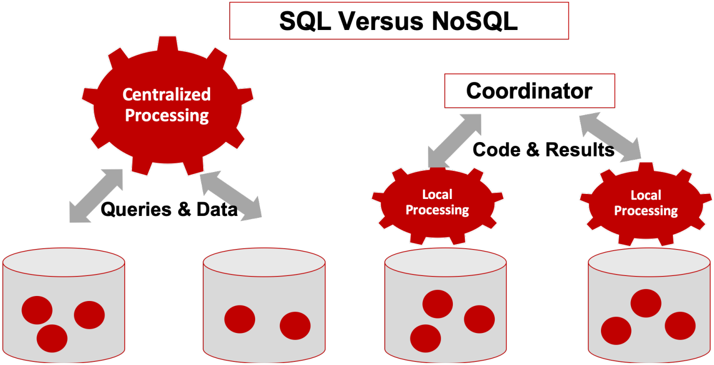
Nacéra Seghouani

Computer Science Department, CentraleSupélec
Laboratoire de Recherche en Informatique, LRI LaHDAK group
nacera.bennacer@centralesupelec

2019-2020

Introduction

☞ Design of parallel algorithms in a distributed environment.



Introduction

- ☞ Design of parallel algorithms
- ☞ Big data processing pipelines
- ☞ Trade the communication cost against the degree of parallelism
- ☞ Processing pipeline based on MapReduce paradigm in a distributed environment.
  - ➜ Google's internal implementation and Hadoop (Apache Foundation) to manage large-scale computations, to be tolerant of hardware faults.
  - ➜ HDFS, Hadoop Distributed File System, splits files into large blocks and distributes them across nodes in a cluster.
  - ➜ MapReduce programming model to manage many large-scale parallel computations

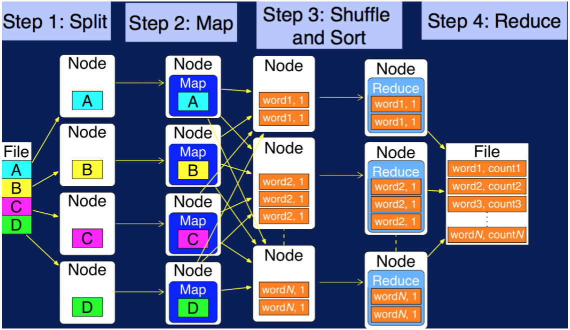
Overview of MapReduce Computation Paradigm

- A style of computing

- All you need: define *Map* and *Reduce* functions, while the system
  - ➜ manages the parallel execution and coordination,
  - ➜ deals with the possibility that one of these tasks will fail to execute.

Example: Word Counter

- A, B, C, D are files distributed across different nodes (machines).
- *Map* task turns each partition into a sequence of pairs $(word, 1)$.
- Shuffle/sort task collects and groups the pairs by key/word $(word, [1, 1, \ldots])$ in order to guarantee that the same key will be processed by the same reduce task.
  Shuffling is a process of redistributing data.
- *Reduce* task takes as input $(word, [1, 1, \ldots])$ and produces pairs by key $(word, countWord)$.



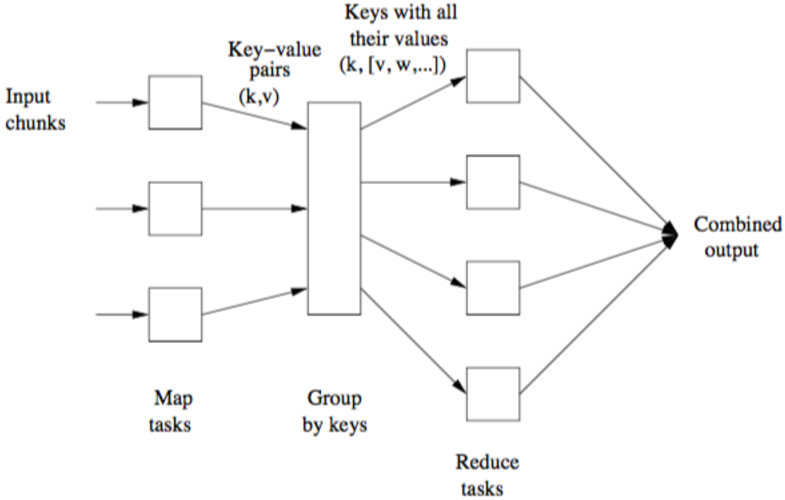
Example: Word Counter

- *Map* task will typically process many words in one or more **chunks** (of about 64MB by default).
  - ➜ if a word w appears m times among all chunks assigned to that process, there will be m key-value pairs $(word, 1)$ among its output.
- To perform the grouping and distribution to the Reduce task, the master controller:
  - ➜ merges the pairs by key/word and produces a sequence of $(word, [1, 1, ..., 1])$.
  - ➜ knows how many reduce tasks there will be, say r, produces from 1 to r lists, puts a list in one of r local files destined for one of the Reduce tasks.
- Each key/word k is assigned as input to one and only one *Reduce* task.
- *Reduce* task executes one or more reducers (one by key). The outputs from all reducers are merged into a single file.

Overview of MapReduce Computation Paradigm

- *Map* task is given one or more chunks from HDFS, turns the chunk into a sequence of key-value pairs (k, v) determined by the Map function F_{map}.
- The key-value pairs (k, v) from each Map task are collected by a master controller and sorted by key.
- The key-value pairs (k, v) are then assigned to the Reduce tasks, all (k, v) with the same k are assigned to the same Reduce task.
- *Reduce* task works on one key k at a time, and combine all the values associated $(k, list(v))$ using the Reduce function F_{red}.
- Inputs to reduce tasks and outputs from map tasks of the key-value pair form allow the composition of MapReduce processes.

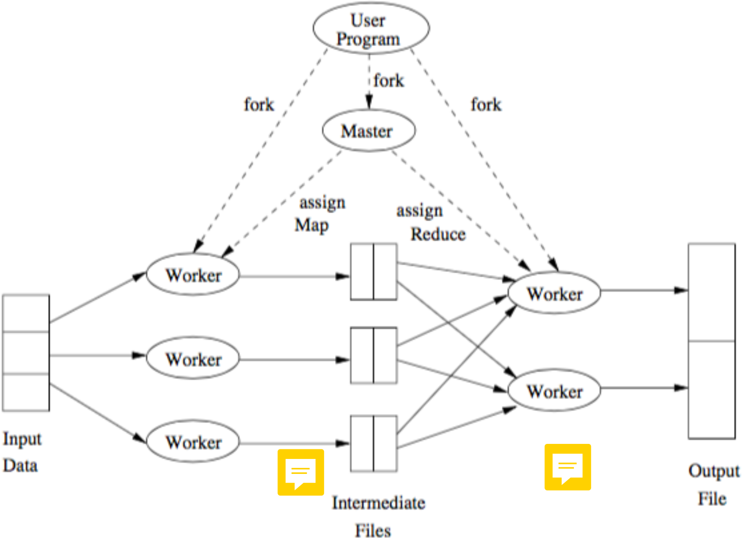
Overview of a MapReduce Computation Paradigm



MapReduce Execution Model

- The user program forks a master controller process and some number of worker processes at different compute nodes.
- The master creates some number of map tasks and some number of reduce tasks. It assigns the tasks to worker processes by taking into account the co-location.
- A worker handles either map tasks (a map worker) or reduce tasks (a reduce worker), but not both.
- A worker process reports to the master when it finishes a task, and a new task is scheduled by the master for that worker process.
- The master keeps track of the status of each map and reduce task (idle, executing at a particular worker, or completed).

MapReduce Execution Model



MapReduce Execution Model: Coping With Node Failures

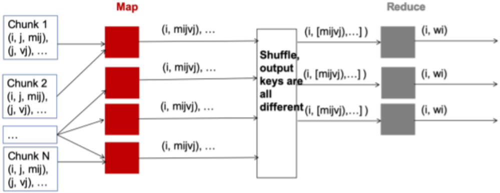
- ☞ If the master node fails the entire MapReduce job must be restarted.
- ☞ Work node failure is detected and managed by the master, because it periodically pings the worker processes.
- ☞ All the map tasks assigned to this worker have to be redone.

Algorithms by MapReduce

- MapReduce is not a solution to every problem
- It makes sense only when files are very large and are rarely updated.
- The original purpose of Google MapReduce implementation is to execute very large matrix-vector multiplications.

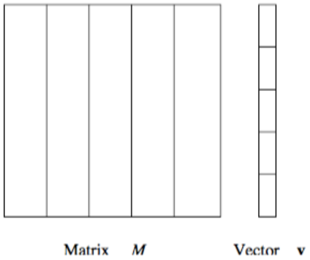
Matrix-Vector Multiplication by MapReduce

- Let $M[m_{ij}]$ be a squared matrix and V a vector of size n, the product $W = MV$ is defined : $w_i = \sum_j^n m_{ij} v_j$
- M and V are stored in a file of the DFS as triples (i, j, m_{ij}) and pairs (j, v_j)
- Map function, applied to each (i, j, m_{ij}) and (j, v_j), produces a key-value pair $(i, m_{ij} v_j)$
- Reduce function simply sums all the values associated with a given key i, produces a pair (i, w_i)



Matrix-Vector Multiplication by MapReduce

- The matrix M and the vector V each will be stored in a file of the DFS. If n is too large V cannot not fit in main memory of a work node, a large number of disk accesses are required.
  - Divide the matrix into vertical stripes of equal width and divide the vector into an equal number of horizontal stripes, of the same height.
  - Each Map task is assigned a chunk from one of the the matrix stripes and gets the entire corresponding stripe of the vector.



Matrix Multiplication by MapReduce

- Let M and N be matrixes of size $l \times r$ and $r \times t$ resp., the product $P = MN$ is a matrix of size $l \times r$, where $p_{ik} = \sum_{j=1}^{r} m_{ij} n_{jk}$
- Fom (M, i, j, m_{ij}) and (N, j, k, n_{jk})) the Map task produces $(j, (M, i, m_{ij}))$ and $(j, (N, k, n_{jk}))$
- The Reduce Function produces for each key j the key-value-pair $((i, k), m_{ij} n_{jk})$.
- Grouping and aggregation achieved by another MapReduce operation.
- The Map Function: just the identity.
- The Reduce Function: For each key (i, k), produce the sum of the list of values associated with this key $p_{ik} = \sum_j m_{ij} n_{jk}$
- M and N could be divided into n vertical and horizontal stripes of resp. (l, r_i) and (r_i, t) sizes, with $\sum_i^n r_i = r$

Relational-Algebra Selection and Projection Operations by MapReduce

- Let $R(A_1, A_2, \ldots A_n)$ be a relation stored as a file in a DFS. The elements of this file are the tuples of R.
- Selection $\sigma_C(R)$
  - Map Function: For each tuple t in R, test if it satisfies C. If so, produce the key-value pair (t, t). That is, both the key and value are t.
  - Reduce Function: It simply passes each key-value pair to the output.
- Projection $\pi_A(R)$
  - Map Function: For each tuple t in R, construct a tuple $t\prime$ by eliminating from t attributes $\notin A$. Output the key-value pair $(t\prime, t\prime)$.
  - Reduce Function: For each key $t\prime$ produced by any of the Map tasks, there will be one or more key-value pairs $(t\prime, t\prime)$. The Reduce function turns $(t\prime, [t\prime, \ldots t\prime])$ into $(t\prime, t\prime)$, so it produces exactly one pair $(t\prime, t\prime)$ for this key $t\prime$.

CentraleSupélec

Relational-Algebra Natural Join Operation by MapReduce

- ☞ $R(A) \bowtie_B S(C)$, with A, B, C sets of attributes, $B \subseteq A$ and $B \subseteq C$
  - ➜ The Map Function: For each tuple (a, b) of R, produce the key-value pair $(b, (R, a))$. For each tuple (b, c) of S, produce the key-value pair $(b, (S, c))$.
  - ➜ The Reduce Function: Each key value b will be associated with a list of pairs $(b, [(R, a), (S, c)])$.

CentraleSupélec

Grouping and Aggregation Operations by MapReduce

- $\gamma_{A,\theta(B)}(R)$, where $A \cup B$ is the set of attributes of R, A is the set of grouping attributes with $A \cap B = \phi$.
  - The Map Function: For each tuple produce the key-value pair (a, b).
  - The Reduce Function: Each key a represents a group. Apply the aggregation operator θ to the list $(a, [b1, b2, \ldots, bn]$. The output is the pair (a,x), where x is the result of applying θ to the list.

CentraleSupélec

Map Reduce & MongoDB