MapReduce and Pregel limits in BigData processing

Mostafa Bamha

Université d'Orléans, INSA Centre Val de Loire, LIFO EA 4022, France
Email. Mostafa.Bamha@univ-orleans.fr

LaMHA meeting, March 27th, 2017

Outline

1. MapReduce model and its limits
   - Parallel and Mapreduce Join processing limits
   - Randomised keys: A solution for data skew in Join queries using MapReduce
   - Tests of performance of Join and GroupBy-Join queries
2. Variants of MapReduce (Pregel, GraphLab, ...)
   - High degree vertices problem in Graph processing
   - Test of performance of high degree vertices partitioning
3. Current research on Graph & Bigdata processing

Data processing using MapReduce

A High-level Parallel Programming model :

➠ Communication, load balancing, fault tolerance, synchronisation, ... issues.

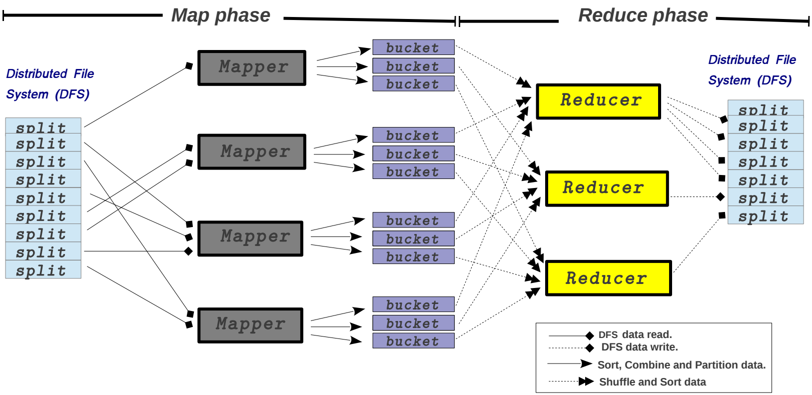
Distributed File Systems: Hadoop DFS, Google's File System, ...

- Build from thousands of commodity machines: Assure scalability, reliability and availability issues
- Files divided into Chunks/Blocks of data and each block is replicated on several nodes for fault tolerance.

MapReduce Model:

Programs easily written : Workflow of Map & Reduce operations.

MapReduce Workflow



Return
Pregel

MapReduce: A programming model for large-scale data-parallel applications

MapReduce is efficient in many applications:

- Hides low level parallel programming details,
- Scalable to Petabytes of data processed on clusters with thousands of commodity machines,
- Suitable for programs that can be decomposed into many independent parallel tasks.

MapReduce model

MapReduce Model - Map-reduce Workflow

map: $(k_1, v_1) \longrightarrow list(k_2, v_2)$,
reduce: $(k_2, list(v_2)) \longrightarrow list(v_3)$.

In Map phase: All emitted pairs (k_2, v_2) with the same value k_2 are sent to the same reducer !!!

MapReduce may be sensitive to data skew:

- Appropriate map keys and communication templates should be generated to avoid the effects of data skew this imbalance can not be directly handled by MapReduce framework,
- Data redistribution must be performed using **User defined MapReduce** *Partition* **function**.

Return

Join of two relations

The *join* of two relations R and S on attribute A of R and attribute B of S is the relation, written $R \bowtie S$, obtained by concatenating the pairs of tuples from R and S for which $R.A = S.B$.

Example -1-

Relation R

Product	*Company*
prod1	*2*
prod2	*2*
prod3	*3*
prod4	*3*
prod5	*3*
prod6	*1*

6 tuples

Relation S

Item	*Company*
item1	*4*
item2	*3*
item3	*3*
item4	*2*
item5	*2*
item6	*3*
item7	*5*

7 tuples

$R \bowtie S$

Product	*Item*	*Company*
prod1	*item4*	*2*
prod1	*item5*	*2*
prod2	*item4*	*2*
prod2	*item5*	*2*
prod3	*item2*	*3*
prod3	*item3*	*3*
prod3	*item6*	*3*
prod4	*item2*	*3*
prod4	*item3*	*3*
prod4	*item6*	*3*
prod5	*item2*	*3*
prod5	*item3*	*3*
prod5	*item6*	*3*

13 tuples

◂ Parallel join

◂ MapReduce join

Parallel evaluation of Join Queries

Parallel Join evaluation proceeds in 2 phases:

1. A redistribution phase where the relations to join are partitioned into distinct buckets. These buckets are generally generated using a hash function of the join attribute and sent to distinct processors.
2. A join phase where each processor computes the join of its local buckets.

Parallel hash join : Example 1.1

→ Number of processors = 3
→ Hashing function : (Company mod 3) +1

Processor 1

Relation R1

Produit	Company	
prod5	3	(1)
prod1	2	(3)
prod6	1	(2)

3 tuples

Processor 2

Relation R2

Produit	Company	
prod3	3	(1)

1 tuples

Processor 3

Relation R3

Produit	Company	
prod2	2	(3)
prod4	3	(1)

2 tuples

Relation S1

Item	Company	
item1	4	(2)
item7	5	(3)
item6	3	(1)

3 tuples

Relation S2

Item	Company	
item2	3	(1)
item5	2	(3)

2 tuples

Relation S3

Item	Company	
item4	2	(3)
item3	3	(1)

2 tuples

◂ Sequential join

Example -1.2-

Processor 1

Relation R1

Product	Company
prod3	3
prod4	3
prod5	3

3 tuples

Relation S1

Item	Company
item2	3
item3	3
item6	3

3 tuples

Processor 2

Relation R2

Product	Company
prod6	1

1 tuples

Relation S2

Item	Company
item1	4

1 tuples

Processor 3

Relation R3

Product	Company
prod1	2
prod2	2

2 tuples

Relation S3

Item	Company
item4	2
item5	2
item7	5

3 tuples

Example -1.3-

Processor 1

$R1 \bowtie S1$

Product	Item	Company
prod3	item2	3
prod3	item3	3
prod3	item6	3
prod4	item2	3
prod4	item3	3
prod4	item6	3
prod5	item2	3
prod5	item3	3
prod5	item6	3

9 tuples

Processor 2

$R2 \bowtie S2$

Product	Item	Company

0 tuples

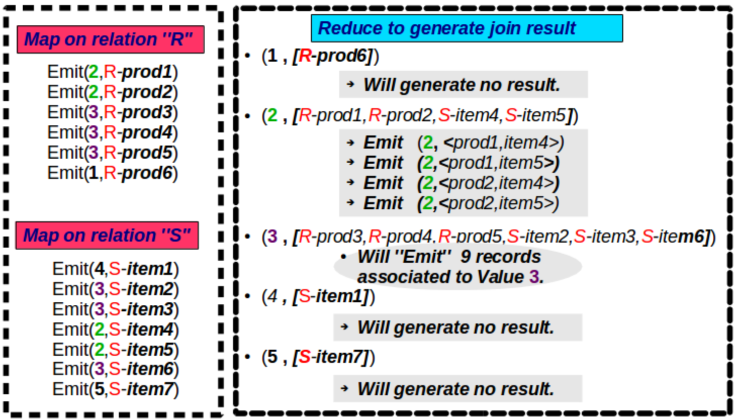
Processor 3

$R3 \bowtie S3$

Product	Item	Company
prod1	item4	2
prod1	item5	2
prod2	item4	2
prod2	item5	2

4 tuples

Join processing using MapReduce: Example



➠ Is very sensitive to data skew

◂ Sequential join

A Skew insensitive MapReduce approach for Join & GroupBy-Join queries

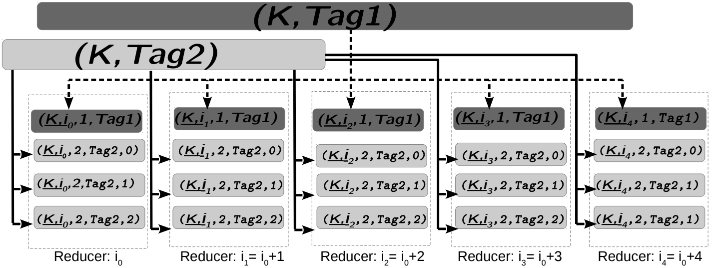
A Skew insensitive MapReduce join algorithm for Distributed File Systems :

MRFA_Join computation steps :

1. Map phase to compute local histograms of join attribute,
2. Reduce phase (global histogram's frequencies, Number of buckets used to partition records of each relevant join attribute value, ...),
3. Map phase for relevant and randomised data redistribution,
4. Reduce phase for join computation.

Randomised communication templates in MRFAG_Join

Example of generated mapper keys used to partition data associated to a join attribute K associated to a high frequency.



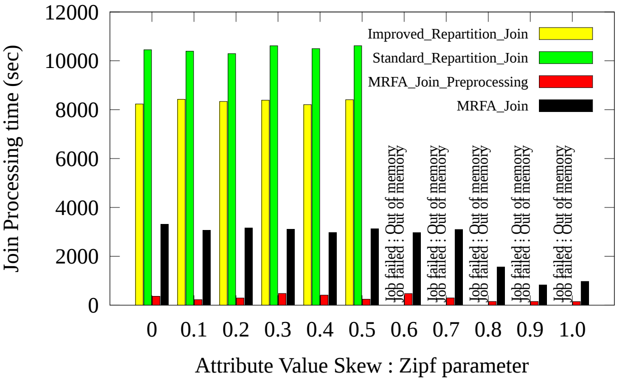
MapReduce model's limit

MapReduce model's limit

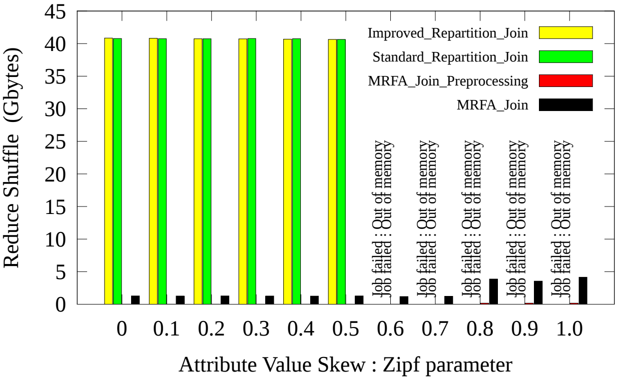
- Is very sensitive to data skew problem,
- Is inappropriate in the case of iterative problems since input data must be read from DFS and output data must rewritten back to DFS for each iteration,
- Do not scale well in the case of dependant tasks or graph processing since this may induce high communication and disk I/O costs for each iteration.

Data skew effect on Hadoop join processing time

* Zipf=0.0-1.0, Input relations ∼400M records (∼40GB of data),
* Join result varied from : ∼35M to ∼17000M records (∼7GB to ∼340GB of data).



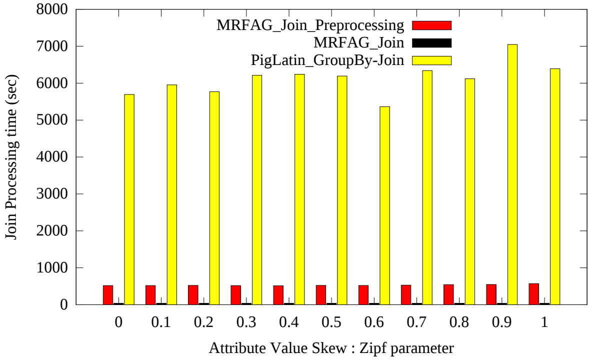
Data skew effect on the amount of data moved across the network during shuffle phase



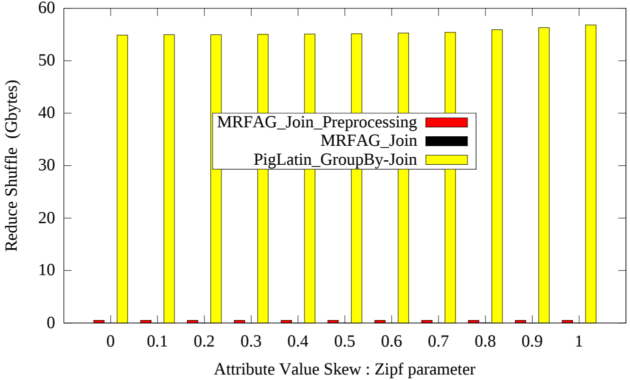
Data skew effect on Hadoop GroupBy join processing time

* Zipf=0.0-1.0, Input relations of ∼1billion and 400M records (resp. ∼ 100GB and 40GB of data),

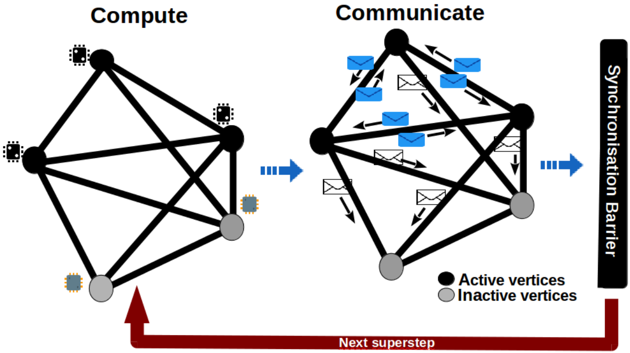
* GroupBy Join result varied from : ∼20M to ∼50M records (∼400MB to ∼1GB of aggregated data).



Data skew effect on the amount of data moved across the network during shuffle phase



Variants of MapReduce for graphs or iterative processing



Bulk Synchronous Parallel Model in Graph processing using Pregel

MapReduce Workflow

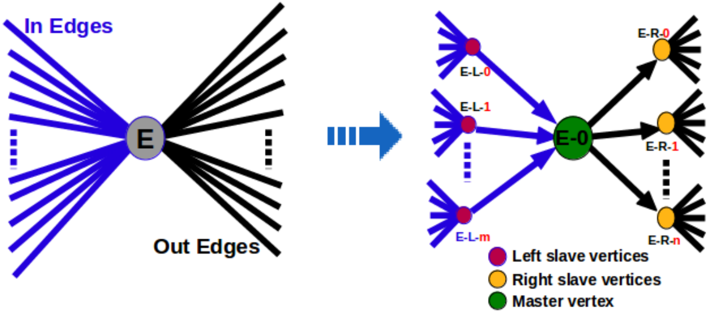
Variants of MadReduce for graphs or iterative processing (Pregel, GraphLab, ...)

➠ Efficient for graphs or iterative processing.
Many challenges are still not solved:

1. Communication and load imbalance can be very high in presence of high degree vertices,
2. Existing solutions, in many problems, are not optimised, for example the "Shortest path" :
   - Each iteration, passes the shortest distance seen from one node to its neighbours : the number of iterations is equal the longest path from source node !!!!

   ➠ This may induce load imbalance since only the neighbours of a node are discovered and activated at each iteration,

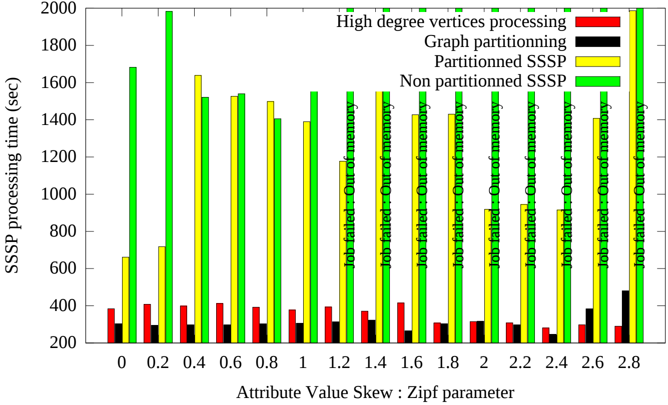
High Degree vertices - Graph topology transformation



High degree vertices partitioning: **Slave vertices are affected to distinct random workers** in a round-robin manner for scalability

Graph skew effects on SSSP processing time and scalability

* 200M vertices and 1B edges (about ∼25GB for each input graph)
* Zipf=0.0-2.8 (Natural graphs : Zipf ∼2.0)



Current research

1. Extend the use of randomised keys to graph processing using of a master/slave approach (using Pregel, GraphLab or other MapReduce variants) to solve the problem of load imbalance due to **high degree Vertices**,
2. Development of optimized and scalable programs in applications such as:
   - Collaborative filtering, Graph mining, PageRank, Shortest Path, etc.

   using a randomized approach for data redistribution related to high degree vertices,
3. Participate to the development of an optimised library for efficient graph processing in the scope of "Girafon" project,
4. Participate to the development of scalable algorithms for BigData Mining (ICVL Action).