Systems for Big data

Authors :

- Randy ZEBAZE DONGMO
- Paul FOTSO KAPTUE

Presentation of the problem

Purpose :

- ❖ Familiarize with use of MapReduce
- ❖ Solve big data problems with the use of Hadoop and Spark

Integer Manipulation with Apache Spark

- ❖ We want to compute the largest integer in the file
- ❖ We use a first map to cast the lines of our RDD to integers
- ❖ After that we use a reduce operation to compute in parallel the different maxima and aggregate them.

Average of all the integers

- ❖ We want to compute the average of all the integers
- ❖ Firstly we compute the sum by using a MapReduce operation:
  - ➢ Map for cast the lines of our RDD to integrer
  - ➢ Reduce to compute the sum
  - ➢ We use the count function in order to find the number of elements

Output all the distinct integers

- We want all the distinct integers
- We use a Map operation to map x to the tuple (x,1)
- We ReduceByKey and we obtain all the distinct integers and their occurrence
- We finally save the keys

Number of distinct integers

- ❖ We want the numbers of distinct integers
- ❖ We use the previous pairRDD (distinct integer, number of occurrence) and we apply the count() function on the RDD which contains its keys.

Integer Manipulation with Spark Streaming

Largest of all integers

- ❖ We want the largest integer
- ❖ We begin by storing our file in a Dstream
- ❖ We apply the MapReduce to this Dstream as we did when using RDD
- ❖ We use list that contain the largest integer per batch of the stream
- ❖ we use the foreach function of RDD to update the list and print the current maximum who is equal to max(old-max, actual-batch-max)

Average of all the integers

- We want the average of all the integers
- We start by storing our file as a Dstream
- We apply a Map x - > (x, 1) followed by a Reduce ((x1, y1), (x2, y2)) -> (x1 + x2, y1 + y2)
- We use 2 lists, one containing the sum of integers per batch, and the other containing the number of integers per batch
- We use foreachRDD to update these 2 lists and the value of the average we output

Number of distinct integers

- ❖ We want the number of distinct integers
- ❖ We are going to use the Flajolet-Martin Algorithm, so we choose a **hash function** *h*
- ❖ We start by storing our file as a Dstream
- ❖ We apply a Map x - > ***trailing zeros*** (*h*(x))
- ❖ We use a list that store the maximum number of trailing zeros per batch
- ❖ We use foreachRDD to update the list and output 2 ^ (The current maximum number of trailing zeros)

Ranking wikipedia web pages

- We have a graph G where :
  - Nodes : wikipedia web pages
  - Edges : hyperlinks between pages
  - PySpark API

Eigenvector centrality computation

With Apache Spark:

- ❖ Two steps matrix multiplication
- ❖ Computation of the norm with one MapReduce

Most important pages in Wikipedia

- Build an RDD that map each pages to it index
- Compute rt by our matrix multiplication algorithm
- This algorithm return rt as a pairRDD<Integer,Double>(index->importance)
- We used the max() method to get the index with the maximum importance value
- Finally we apply the lookup() method to find the corresponding name

Matrix Multiplication

We define a matrix element by three attributes:

- ❖ Row number
- ❖ Column number
- ❖ Corresponding Value

Matrix Format storage :

- ❖ We store the matrix A in RDD where line are in the form (i,j,aij)
- ❖ We store the matrix B in RDD where line are in the form (j,k,bjk)
- ❖ We don’t store null elements

Using Two MapReduce steps

- ❖ We used a first Map to send each matrix element aij respectively(bjk) to the pair (j,(A,i,aij)) respectively (j,(B,k,bjk))
- ❖ For each j, we examined its lits of values, and for each value that comes from A (A,i,aij) and each value that comes from B (B,k,bjk) we produced the tuple (i,k,val= aij*bjk)
- ❖ We used a reduce to output (j, [(i1,k1,val1),......(iq,kq,valq)])
- ❖ We flat the previous paire to produce q pairs ((i1,k1,val1),......(iq,kq,valq))
- ❖ For each key (i,k) we used a reduce operation to sum all its associated value
- ❖ We ended up with a pair ((i,k),v)

Using one MapReduce Step

- We use a Map to, for each aij of A produce pair ((i,k),(A,j,aij)) for k = 1...up to the number of columns of B.
- The same operation for each bjk of B to produce pair ((i,k),((B,j,bjk)) for i = 1..up to the number of columns of A
- For each (i,k) we have a list of value (A,j,aij) and (B,j,bjk) for all possible j
- We associated the two values that have the same j, for each j by:
  - Sort by j the values that begin with A, and do the same for values that begin with B in separate lists
  - We extracted and multiplied the third components aij and bjk of the jth values of each list
  - theses products are summed up and paired with (i,k)

Cost of computing eigenvector centrality

cost = #iterations ×(cost(matrix multiplication) +cost (vector normalization))

Cost with two mapreduce steps

- The first map produces m tuples coming from A because we store only non-zero's elements and(at most) n tuples coming from B. We end up with m+n tuples
- The second Map produce for each j at most degree(i) tuples and for all j we have m(sum of all degree(j)) tuples which are send to the reduce
- With the formula above, we have:
- Cost = k*((2m+n) + O(n))

Cost with one MapReduce step

- The Map produces m tuples coming from A and n^2 tuples coming from B
- We ended up with m+n^2 tuples to send to the reduce
- With the cost generic formula above we have:
- cost = k*((m + n^2) + O(n))