BIG DATA MANAGEMENT

CZ/CE4123

Tutorial 7
Distributed Systems and MapReduce

QUESTION 1

Amazon wants to estimate the Top-K best sold products from S purchase records of L products in the form of a list of (User id, Product id) pairs. Suppose there is a distributed system with 1 master machine and M slave machines. Assume that L is a multiple of M. Design a distributed computation procedure to finish the task. Please describe

(1) how the data is distributed, computed and aggregated?

(2) how much data is sent across machines?

SOLUTION – MAIN IDEA

Let us roughly think…

First, we let each machine handle L/M products. Distribute the records corresponding to Product 1, 2, …, L/M to Machine 1, the records corresponding to Product $L/M+1$, …, $2L/M$ to Machine 2, …

Second, in each machine, we can simply compute the top-K sold products locally.

Third, when there are local Top-K best sold products computed from each machine, aggregate the results to finally form the global top-K results.

SOLUTION – DETAILS

Firstly, the master machine distributes the S purchase records to the slave machines, so that the records of Product 1, 2, …, L/M are sent to Machine 1, the records of Product L/M+1, …, 2L/M are sent to Machine 2, …

Secondly, each machine computes the top-K sold products locally. In particular, for each product, we collect the frequencies of it being sold, and sort the frequencies and find the K products *with* top-K frequencies. Only send the top-K pairs of (Product id, Frequency) to master.

SOLUTION – DETAILS

Finally, the master machine aggregates all the MK pairs of (Product id, Frequency). Output the products with the Top-K frequencies.

SOLUTION – DATA SENT

Data Distribution Phase: S records have been sent out.
Aggregation Phase: MK pairs of (Product id, Frequencies) have been sent out.

In total, the data sent out is in the scale of $O(S+MK)$.

QUESTION 2

(1) In a MapReduce job, the output of Map phase is a list of key-value pairs:

(A, 1) (C, 2), (A, 5), (C, 6), (B, 3), (E, 3), (C, 8). Please list the possible input to the Reduce function.

The input to the *Reducer* aggregates the *Map* output keys.
Hence, the possible input to the reducer is (A, {1, 5}); (B, {3}); (C, {2, 6, 8}); (E, {3}).

(2) Based on the answer to Q2(1), write a Reduce function (in pseudocode) so that the MapReduce output is: (2, A), (3, C), (6, A), (7, C), (4, B), (4, E), (9, C).

```
Reduce(int key, iterator values){
    for(each v in values){
        Emit(v+1, key);
    }
}
```

(3) Consider an employee table containing three columns (EmployeeID, age, monthly-salary) where age and monthly-salary are integers. Use MapReduce to collect the number of employees falling into each of the following two categories:

- Category 1: The age of the employee is between 30 and 40 (including 30 and 40). His/her monthly salary is at most 7000.

- Category 2: The age of the employee is between 40 and 50 (including 40 and 50). His/her monthly salary is more than 7000.

Please use **only one MapReduce Job** to achieve this task and write down the pseudocode of the Map function and Reduce function. The input key and value for Map function are an employee's age and monthly-salary respectively.

```
Map(int age, int salary){
    if(age>=30 and age<=40 and salary<=7000){
        Emit-Intermediate(1, 1);
    }
    if(age>=40 and age<=50 and salary>7000){
        Emit-Intermediate(2, 1);
    }
}

Reduce(int category, iterator values){
    int total_frequency = 0;
    for(int frequency: values)
    {
        total_frequency+=frequency;
    }
    Emit(category, total_frequency);
}
```

QUESTION 3

Design MapReduce algorithms for the multiplication of two matrices *A*, *B* of *n* by *n*. Elements in the matrices are integers. The input key for *Map* function is *MatrixName*; the input value for *Map* function is in the form of *i;j;v*, indicating that the value of the *i*-th row and *j*-th column is *v*.

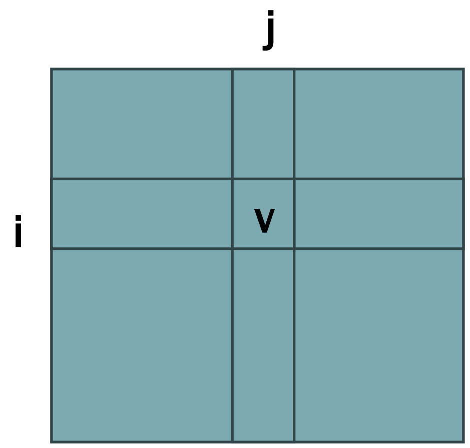
(Matrix Multiplication: Given matrix *A*[*nxn*] and matrix *B*[*nxn*], compute matrix *C* such that $\mathrm{C[i][z]} = \sum_{\mathrm{j=0}}^{\mathrm{n-1}} \mathrm{A[i][j]xB[j][z]}$, for i,z in [0, n-1]).

(1) Use at most two MapReduce jobs to finish the computation.

(2) Furthermore, can the multiplication be finished using one MapReduce job?

SOLUTION – 1ST JOB

```
Map(String MatrixName,  String value){
    int i = get_i(value);// get row
    int j = get_j(value);// get column
    int v = get_v(value);// get value
    if(MatrixName==“A”)
                                Column as key
        Emit-Intermediate(j, ToString(MatrixName, i, v));// combine
    else
        Emit-Intermediate(i, ToString(MatrixName, j, v));
                                Row as key
}
```



For each intermediate key j, the values can contain

(1) A[u][j] for any u;

(2) B[j][v] for any v;

What needs to be done in reduce()?

For any (u, v), send out A[u][j]*B[j][v] for aggregation of C[u][v]

SOLUTION – 1ST JOB

```
Reduce(String key,  Iterator<String> values){
    int A_start[n];  int B_end[n];
    for(String value : values){
       String MatrixName=get_first_element(value);
       if(MatrixName=="A"){
            int i = get_second_element(value);// get row index in matrix A
            A_start[i]=get_third_element(value);// get the matrix element
       }
       else{
            int j = get_second_element(value);// get column index in matrix B
            B_end[j] = get_third_element(value);// get the matrix element
       }
    }

}
```

A[u][j]   B[j][v]

```
for (int u=0;u<n;u++)
  for(int v=0;v<n;v++)
  {
    Emit(ToString(u,v), A_start[u]*B_end[v]);
  }
```

THE 2^{ND} JOB

Need to aggregate all the values for each pair (u,v).

So,

Map() does nothing;

Reduce() conducts the aggregation.

SOLUTION - 2ND JOB

```
Map(String key, String value){
    Emit-Intermediate(key, value);
}
```

SOLUTION - 2ND JOB

```
Reduce(String key, Iterator<String> values){
    int sum=0;
    for (String value in values){
        sum+=ToInteger(value);
    }
    Emit(key, sum);
}
```

Can we use a single job only?

Think…

For any A[i][j], which elements of C would it contribute to?

Can we use a single job only?

Think…

For any A[i][j], which elements of C would it contribute to?

C[i][k] for any k

For any B[i][j], which elements of C would it contribute to?

C[k][j] for any k

CAN WE FINISH THESE TASKS WITH ONE JOB ONLY?

```
Map(String MatrixName,  String value){
    int i = get_i(value);
    int j = get_j(value);
    int v = get_v(value);
    if(MatrixName=="A")
        for(int k=0;k<n;k++)
            Emit-Intermediate(ToString(i, k), ToString(MatrixName, j, v));// each A[i][j] may contribute
                                                                          //to C[i][k] for any k
    else
        for(int k=0;k<n;k++)
            Emit-Intermediate(ToString(k, j), ToString(MatrixName, i, v)); // each B[i][j] may
                                                                           //contribute to C[k][j] for any k
}
```

For each intermediate key (pair (i,j)), it aggregates

(1) The values of A[i][u], for any u

(2) The values of B[u][j], for any u

CAN WE FINISH THESE TASKS WITH ONE JOB ONLY?

```
Reduce(String index_pair_for_C,  Iterator<String> values){
    int A_middle[n]; int B_middle[n];
    for(String value in values){
        String MatrixName=get_first_element(value);
        if(MatrixName=="A"){
            int j = get_second_element(value);
            A_middle[j]=get_third_element(value);
        }
        else{
            int i = get_second_element(value);
            B_middle[i] = get_third_element(value);
        }
    }

}
```

```
int sum=0;
for (int u=0;u<n;u++)
  sum+=A_middle[u]*B_middle[u];

Emit(index_pair_for_C, sum);
```

EXAMPLE

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

EXAMPLE-MAP

A[0][0] → ((0;0), (A; 0; A[0][0]))
→ ((0;1), (A; 0; A[0][0]))
A[0][1] → ((0;0), (A; 1; A[0][1]))
→ ((0;1), (A; 1; A[0][1]))
A[1][0] → ((1;0), (A; 0; A[1][0]))
→ ((1;1), (A; 0; A[1][0]))
A[1][1] → ((1;0), (A; 1; A[1][1]))
→ ((1;1), (A; 1; A[1][1]))

B[0][0] → ((0;0), (B; 0; B[0][0]))
→ ((1;0), (B; 0; B[0][0]))
B[0][1] → ((0;1), (B; 0; B[0][1]))
→ ((1;1), (B; 0; B[0][1]))
B[1][0] → ((0;0), (B; 1; B[1][0]))
→ ((1;0), (B; 1; B[1][0]))
B[1][1] → ((0;1), (B; 1; B[1][1]))
→ ((1;1), (B; 1; B[1][1]))

EXAMPLE-REDUCE

Take (1,0) for example intermediate key

It aggregates {(A; 0; A[1][0]), (B; 0; B[0][0]), (A; 1; A[1][1]), (B; 1; B[1][0])}

Hence, we can compute C[1][0] by A[1][0]*B[0][0]+A[1][1]*B[1][0]