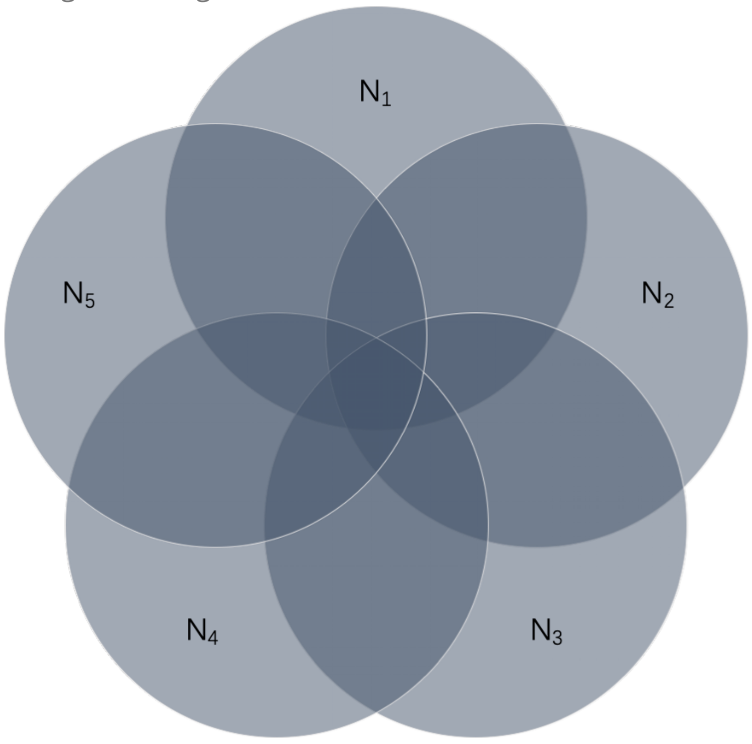
Yuan Gao z5239220 Assignment1

Q1. HDFS

1. Formula of $L_i(k, N)$ for $i \in \{1, \ldots, 5\}$

> - Formula of $L_1(k, N)$:
>
> > Using Venn diagram to derive formula:
> >
> > 
> >
> >
> > According to the diagram,
> >
> > > $N_1 \cap N_2 = L_2(k, N)$
> > > $N_1 \cap N_2 \cap N_3 = L_2(k, N)$
> > > $N_1 \cap N_2 \cap N_3 \cap N_4 = L_4(k, N)$
> > > $N_1 \cap N_2 \cap N_3 \cap N_4 \cap N_5 = L_5(k, N)$
> >
> > Unavailable blocks if one node fails is $\frac{R}{N}$
> > Hence,
> >
> > $$L_1(k, N) = k \cdot \frac{R}{N} - 2L_2(k, N) - 3L_3(k, N) - 4L_4(k, N) - 5L_5(k, N)$$
>
> - Formula of $L_2(k, N)$:
>
> > In $k-1$ period, L_1 has:
> >
> > > There are $L_1(k-1, N)$ blocks have lost one replica. Therefore, these left $4L_1(k-1, N)$ blocks.
> > > In the remained $4L_1(k-1, N)$ blocks, $\frac{4L_1(k-1,N)}{N-(k-1)}$ blocks have lost two replica.
> >
> > In $k-1$ period, L_2 has:
> >
> > > There are $L_2(k-1, N)$ blocks have lost two replica.
> >
> > In k period, L_2 has:
> >
> > > In the remained $3L_2(k-1, N)$ blocks, $\frac{3L_2(k-1,N)}{N-(k-1)}$ blocks have lost three replica. Hence, we need minus these blocks
> >
> > Therefore, the formual of L_2 is
> >
> > $$L_2(k, N) = \frac{4L_1(k-1, N)}{N-(k-1)} + L_2(k-1, N) - \frac{3L_2(k-1, N)}{N-(k-1)}$$
>
> - And by the same logic, $L_3(k, N)$, $L_4(k, N)$, $L_5(k, N)$ can be written.
> - The formula of $L_i(k, N)$ for $i \in \{1, \ldots, 5\}$:
>
> $$\begin{aligned}
> L_1(k, N) =& k \cdot \frac{R}{N} - 2L_2(k, N) - 3L_3(k, N) - 4L_4(k, N) - 5L_5(k, N) \\
> L_2(k, N) =& \frac{4L_1(k-1, N)}{N-(k-1)} + L_2(k-1, N) - \frac{3L_2(k-1, N)}{N-(k-1)} \\
> L_3(k, N) =& \frac{3L_2(k-1, N)}{N-(k-1)} + L_3(k-1, N) - \frac{2L_3(k-1, N)}{N-(k-1)} \\
> L_4(k, N) =& \frac{2L_3(k-1, N)}{N-(k-1)} + L_4(k-1, N) - \frac{1L_4(k-1, N)}{N-(k-1)} \\
> L_5(k, N) =& \frac{L_4(k-1, N)}{N-(k-1)} + L_5(k-1, N)
> \end{aligned}$$

2. Compute the number of blocks that cannot be recovered under this scenario

The result can be calculated by a program:

- Dynamic programming

> According to the formula:
>
> $$L_i(k,N) = \frac{(6-i)\cdot L_{i-1}(k-1,N)}{N-(k-1)} + L_i(k-1,N) - \frac{(5-i)\cdot L_i(k-1,N)}{N-(k-1)}$$
>
> $L_i(k,N)$ is related to $L_{i-1}(k-1,N)$ and $L_i(k-1,N)$
> Therefore, using dynamic programming can be sufficient
>
> - Code
>
> ```
> #include<iostream>
> #include<vector>
>
> using namespace std;
>
> void calculate(vector<vector<double>> & dp, int index, int k, long double R, long double N,
> long double B){
>     if(index == 1){
>         dp[index][k] = k * B - 2 * dp[2][k] - 3 * dp[3][k] - 4 * dp[4][k] - 5 * dp[5][k];
>         return;
>     }
>     dp[index][k] = (6 - index) * dp[index - 1][k - 1] / (N - k + 1)
>                  + dp[index][k - 1]
>                  - (5 - index) * dp[index][k - 1] / (N - k + 1);
> }
>
> int main(){
>     long double R = 20000000;
>     long double N = 500;
>     int K = 200;
>     long double B = R / N;
>
>     vector<vector<double>> dp(5 + 1, vector<double>(K + 1));
>     dp[1][1] = B;
>
>     for(int i = 2; i <= K; ++i){
>         for(int j = 5; j > 0; --j){
>             calculate(dp, j, i, R, N, B);
>         }
>     }
>     cout << dp[5][200] << endl;
> }
> ```

The number of blocks that cannot be recovered under this scenario is approximately 39737.

Q2. Spark

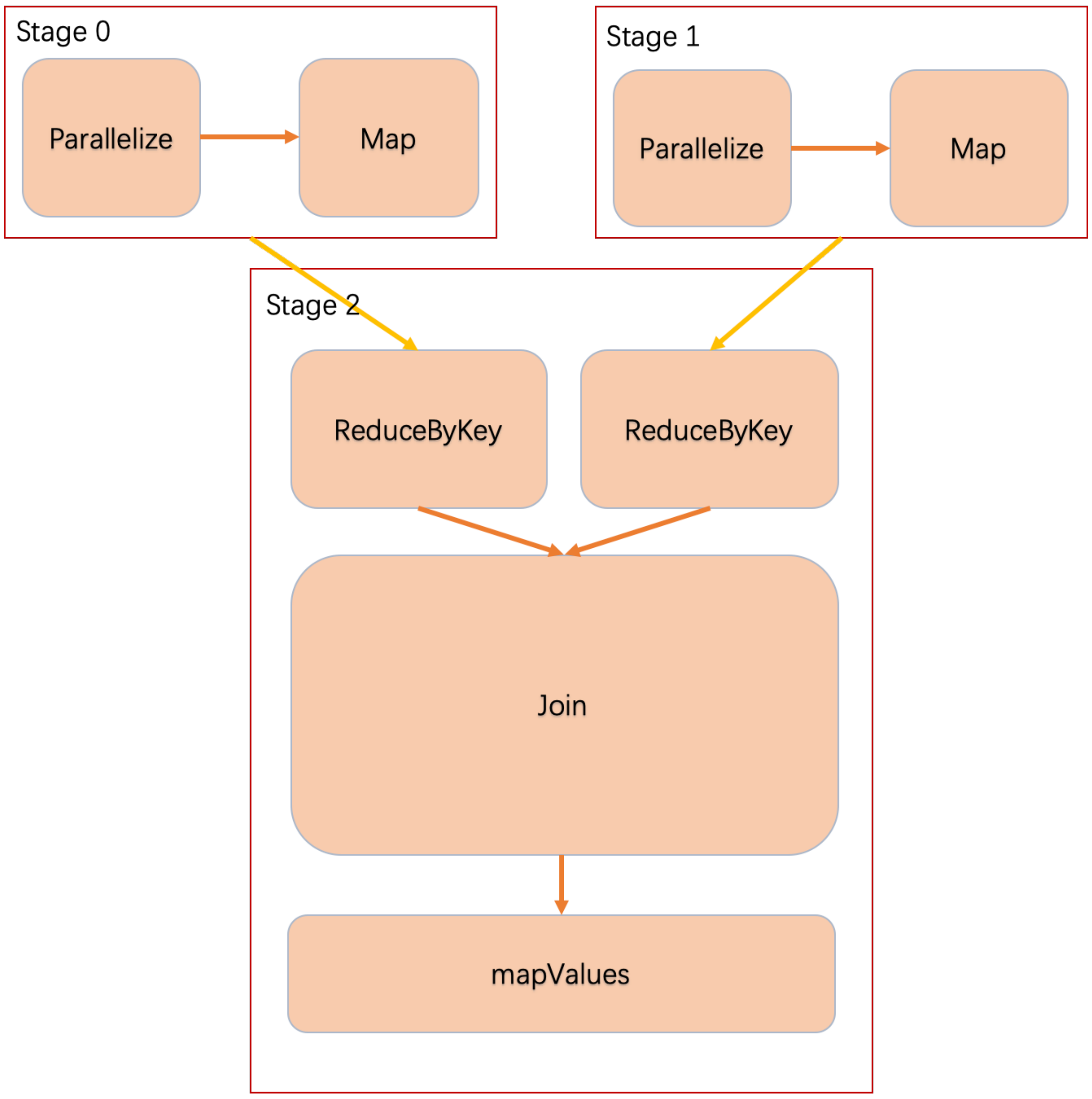
1. Write down the expected output of the above code snippet

- `rdd_1 = sc.parallelize(raw_data)`
  Paralleize python list to pyspark rdd
- `rdd_2 = rdd_1.map(lambda x:(x[0], x[2]))`
  Remove the second column, remain `x[0]` and `x[2]`
- `rdd_3 = rdd_2.reduceByKey(lambda x, y:max(x, y))`
  Get the maximum value `x[2]` for every `x[0]`
- `rdd_4 = rdd_2.reduceByKey(lambda x, y:min(x, y))`
  Get the minimum value `x[2]` for every `x[0]`
- `rdd_5 = rdd_3.join(rdd_4)`
  Join every maximum value (`rdd_3`) and minimum value(`rdd_4`) together
- `rdd_6 = rdd_5.map(lambda x: (x[0], x[1][0]+x[1][1]))`
  Get sum for every maximum value (`rdd_3`) and minimum value(`rdd_4`) together
- `rdd_6.collect()`
  Switch rdd to python list
- Result:

```
rdd_6 = [('Joseph', 165), ('Jimmy', 159), ('Tina', 155), ('Thomas', 167)]
```

2. List all the stages in the above code snippet.

There are 3 stages, because only two `reduceByKey()` have shuffled RDD. The original stage plus two stages are 3 stages.
There are diagram shows these steps:



3. What makes the above implmentation inefficient? How would you modify the code and improve the performance?

- Inefficient step

```
rdd_3 = rdd_2.reduceByKey(lambda x, y:max(x, y))
rdd_3 = rdd_2.reduceByKey(lambda x, y:max(x, y))
```

The above implmentation experience two shuffleds which are inefficient.

- Modify the code
  Reduce shuffle times

```
rdd_1 = sc.parallelize(raw_data)
rdd_2 = rdd_1.map(lambda x:(x[0], x[2]))
rdd_3 = rdd_2.groupByKey() #just experience one shuffleds
rdd_4 = rdd_3.mapValues(lambda x:max(x) + min(x))
rdd_4.collect()
```

Q3: LSH

1. When $k = 5$, the number of tables we can find

> According to the formula $cos(\theta(o,q)) \geq 0.9$, $\theta \leq arccos(0.9) \approx 25.842^{\circ}$
> Hence, the probability is:
> $Pr[h_i(o) = h_i(q)] = 1 - \frac{\theta}{\pi} = 1 - \frac{25.842^{\circ}}{180^{\circ}} \approx 0.856$ which is $p_{q,o}$
> The formula of the probability of find any near duplicate is $1 - (1 - p_{q,o}^k)^L$
> Let $1 - (1 - 0.856^5)^L \geq 0.99$, We can get the reuslt is $L \geq 8$

2. When $cos(\theta(o,q)) < 0.8$, $k = 5$ and $L = 10$, the maximum value

> According to the formula $cos(\theta(o,q)) < 0.8$, $\theta > arccos(0.8) \approx 36.87^{\circ}$
> $Pr[h_i(o) = h_i(q)] = 1 - \frac{\theta}{\pi} = 1 - \frac{36.87^{\circ}}{180^{\circ}} \approx 0.795$ which is $p_{q,o}$
> When $k = 5$ and $L = 10$,
> $1 - (1 - 0.795^5)^{10} < 0.9782$
> Therefore, the probability is 97.82%