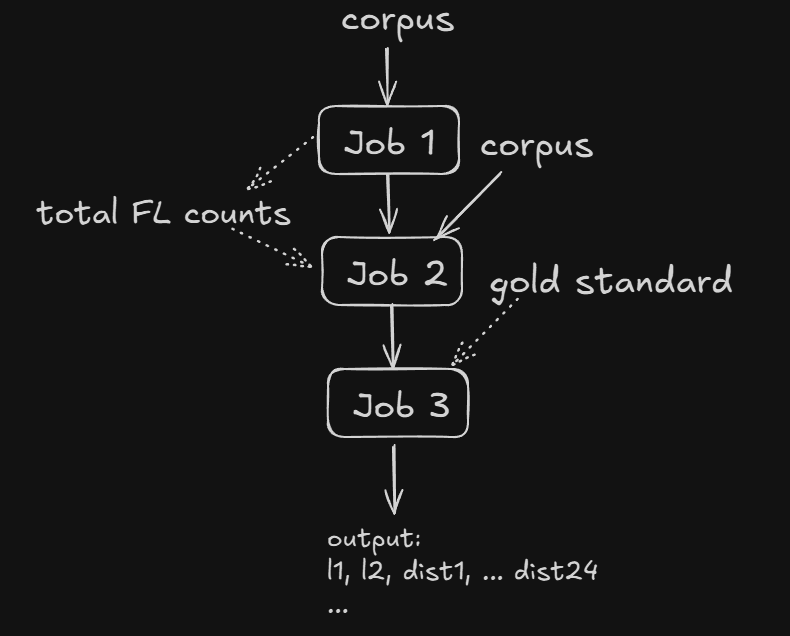
**Assignment 3 – Design Document**

We split the problem into **3 stages** of Map-Reduce:

**Job 1 + Job 2:**

Calculate the **4 association metrics** for each **<Lexeme,Feature>** pair.

**Job1**

We made a **custom key** class **WordAndTagKey** that holds **a** **word and a tag**, to be able to calculate the different count metrics: , in the same job.

Mapper

Input -> corpus

Output -> **<WordAndTagKey,LongWritable>**

There are **4 key options**:  
**<lexeme, 'Lex'>, count\_l  
<lexeme feature, 'Pair'>, count\_lf  
<\*, 'L'>, count\_L**   
**<\*, 'F'>, count\_F**

the sorting: F < L < Lex < Pair

Reducer:

F -> sum all the values and emit to **LFFile**  
L -> sum all the values and emit to **LFFile**  
for each Lexeme the reducer will get the Lex tag for the lexeme and sum all the values to get the count\_l, we will save it as field in the reducer and will get next all the **<lexeme,feature>** Pairs with that specific lexeme and emit the **<lexeme,feature>**,**count\_lf,count\_l**

Out1 line: **<lexeme,feature> count\_lf, count\_l**

outLF file: **L Count(L)**  
 **F Count(F)**

**Job2**

add the count\_f values to each line of out1

Mapper:

Input -> corpus + out1

If corpus line -> emit **<feature,“F”>, count**  
if out1 line -> emit **<feature,“out” >, line**

Reducer:

Startup -> load F and L values from the FLFolder

The sorting will send **same features** in the key to the **same reducer**, with the **“F”** tag keys **first**.

This will enable us to save local field with the total **count\_f** for a feature and immediately afterwards get all the **lexeme feature** pairs and

For each **<lexeme,feature>** pair in the reducer we will have all of the necessary data to calculate the **4 association metrics**.

Out2 line: **<lexeme,feature>, assoc1, assoc2, assoc3, assoc4**

**Job3:**

Mapper

Input -> out2

Startup -> **load gold standard** into a hashmap in each mapper

Map -> for each line in output 2   
 if lexeme in gold standard proceed:  
 go over all paired words with this lexeme in the gold standard and for each of them emit: key: **lexeme** paired with the **other word** in the correct order appeared in the gold standard and the **current feature**, with all the **association metrics as value**, the value will also contain a **tag “First” or “Second”** indicating the position of the sent word values in the pair

Reduce -> we will receive **<lexeme1, lexeme2, feature>** key   
values of type: **assoc1, assoc2, assoc3, assoc4**this will allow us to keep **incremental counters** for each part of the distance and similarity equations.  
the sorting will make all the features of a specific lexeme1,lexeme2 pair come **sequentially** and once the pair is different we can calculate the distance metrics and emit the <l1, l2> with the 24 metrics.

Close -> emit the last <l1,l2> 24 metrics data.

Out3: last **<l1,l2> 24 metrics data**.  
first 4 metrics will be Manhattan distances, with respect to assoc1, assoc2, assoc3, assoc4.  
second 4 will be Euclidian distances and so on up to Jensen-Shannon metric.

**Estimations:**

V1 = set of lexemes, |V1| = n  
V2 = set of features, |V2| = m

**Job1:**

Key-Value pairs: 2 keys for L and F counts, n keys for count\_l, and **O(n\*m)** keys for count\_lf

Memory usage:

Mapper -> **none**   
Reducer -> **O(1),** saving 1 long variable in the reducer for current count\_l

**Job2:**

Key-Value pairs: each feature have 2 possible tags => **2\*m**

Memory usage:

Mapper -> **none**  
Reducer -> **O(1),** saving 1 long variable in the reducer for current count\_f, and 2 longs for count(F) and count(L)

**Job3:**

Let us define: F(l) -> number of features of lexeme l that appear in the corpus

Key-Value pairs:

For each lexeme we will iterate over all lexemes that appear with it in the corpus, and emit 1 record for each one of its features -> **Gold-Standard \* 2 \* F(l)**

Memory usage:

Mapper -> **gold standard**   
Reducer -> **O(1)** for current distances accumulators