Project Report

Data Intensive Computing

CSE – 587

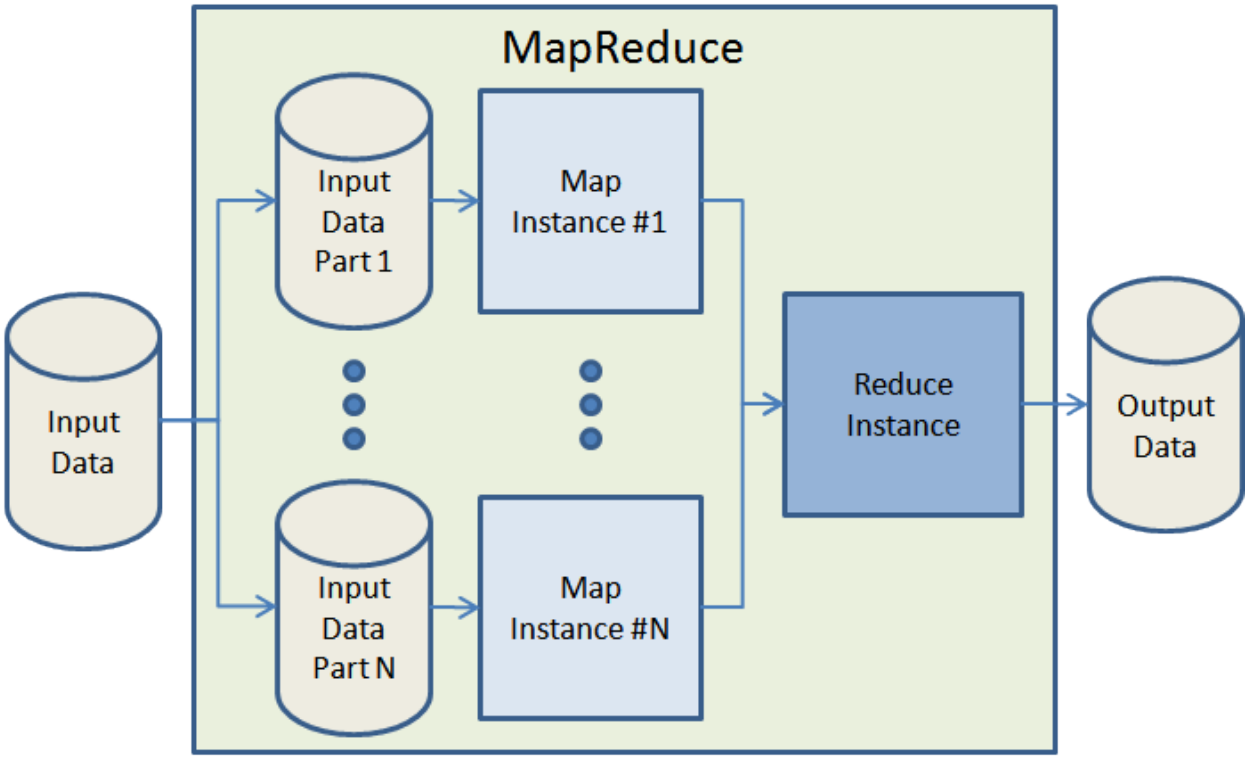
Raman Sonkhla

rsonkhla@buffalo.edu

Person# - 50026724

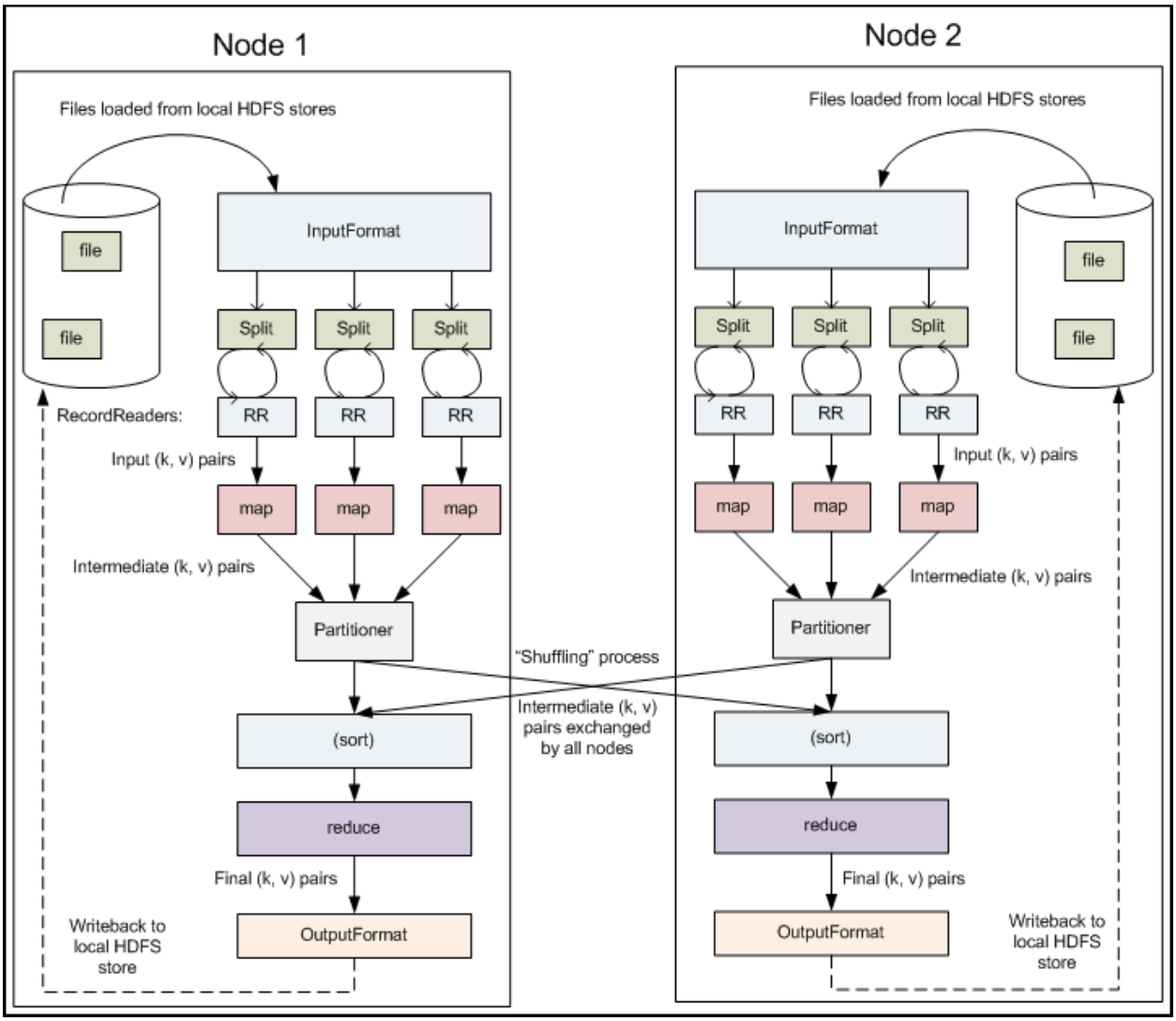
Map-Reduce design details

Map-Reduce programs are designed to compute large volumes of data in a parallel fashion. This requires dividing the workload across a large number of machines. The first phase of a Map-Reduce program is called mapping. A list of data elements are provided, one at a time, to a function called the Mapper, which transforms each element individually to an output data element. Reducing lets you aggregate values together. A reducer function receives an iterator of input values from an input list. It then combines these values together, returning a single output value.



When the mapping phase has completed, the intermediate (key, value) pairs must be exchanged between machines to send all values with the same key to a single reducer. The reduce tasks are spread across the same nodes in the cluster as the mappers. This is the only communication step in Map-Reduce. Individual map tasks do not exchange information with one another, nor are they aware of one another's existence. Similarly, different reduce tasks do not communicate with one another.

Detailed steps describing what happens in MR framework after the input is split, how the input data is processed by the mappers and reducers, how they and the framework communicate with each other, etc. are presented clearly in the following diagram.



Assignment 1 – MR Solution for WORD COUNT

Pseudo Code

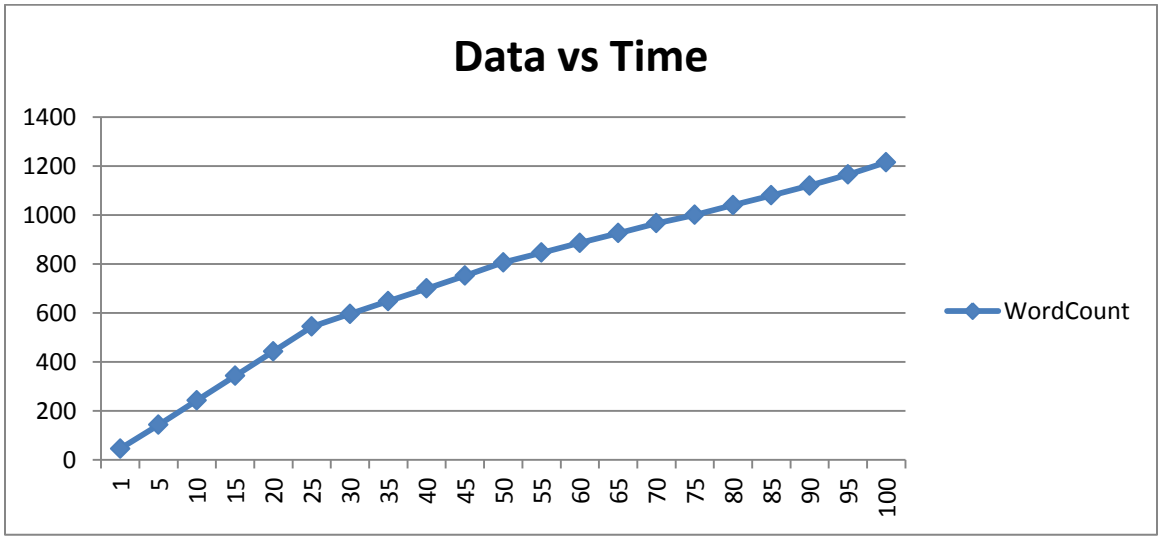
```
1: class Mapper
2: method Map(docid a, doc d)
3: for all term t ∈ doc d do
4: Emit (term t, count 1)

1: class Reducer
2: method Reduce(term t, counts [c1, c2, . . .])
3: sum ← 0
4: for all count c ∈ counts [c1, c2, . . .] do
5: sum ← sum + c
6: Emit (term t, count sum)
```

Performance Table

Data Size (MBs)	Number of words	Time Taken (s)
1	192882	46
5	964410	143
25	4822050	544
50	9644100	806
100	19288200	1215

Performance Chart [X-Axis is data size (in 100KBs) Y-Axis is time (in seconds)]



Assignment 2.1 – MR Solution for WORD CO-OCCURRENCE using PAIRS approach

Pseudo Code

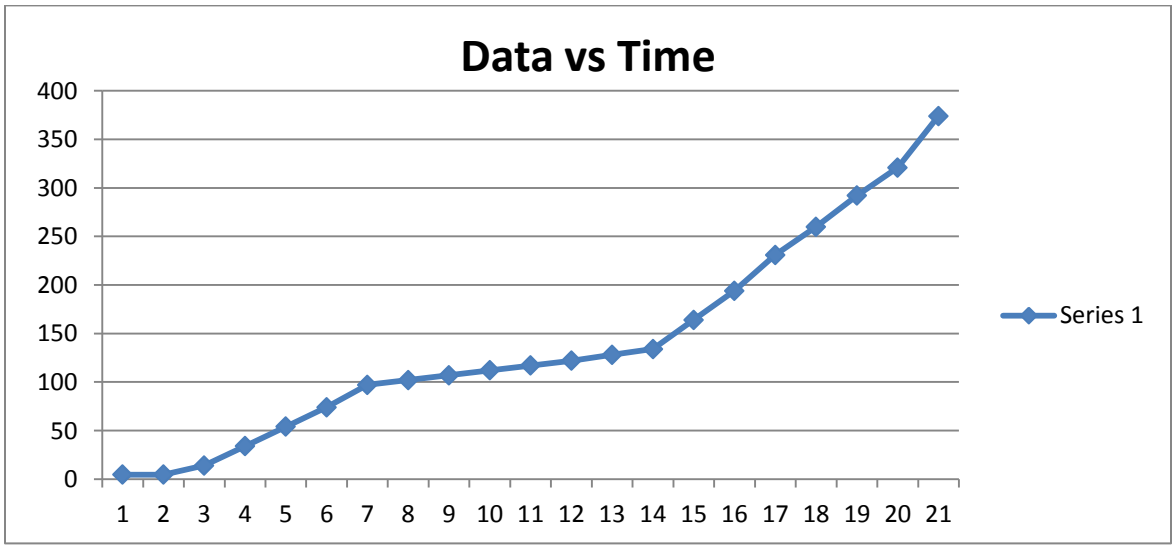
```
1: class Mapper
2: method Map(docid a, doc d)
3: for all term w ∈ doc d do
4: for all term u ∈ Neighbors (w) do
5: Emit (pair (w, u), count 1)

1: class Reducer
2: method Reduce(pair p, counts [c1, c2, . . .])
3: s ← 0
4: for all count c ∈ counts [c1, c2, . . .] do
5: s ← s + c
6: Emit (pair p, count s)
```

Performance Table

Data Size (kBs)	Time Taken (s)
300	14
700	97
1400	134
1800	260
2100	374

Performance Chart [X-Axis is data size (in 100KBs) Y-Axis is time (in seconds)]



Assignment 2.2 – MR Solution for WORD CO-OCCURRENCE using STRIPES approach

Pseudo Code

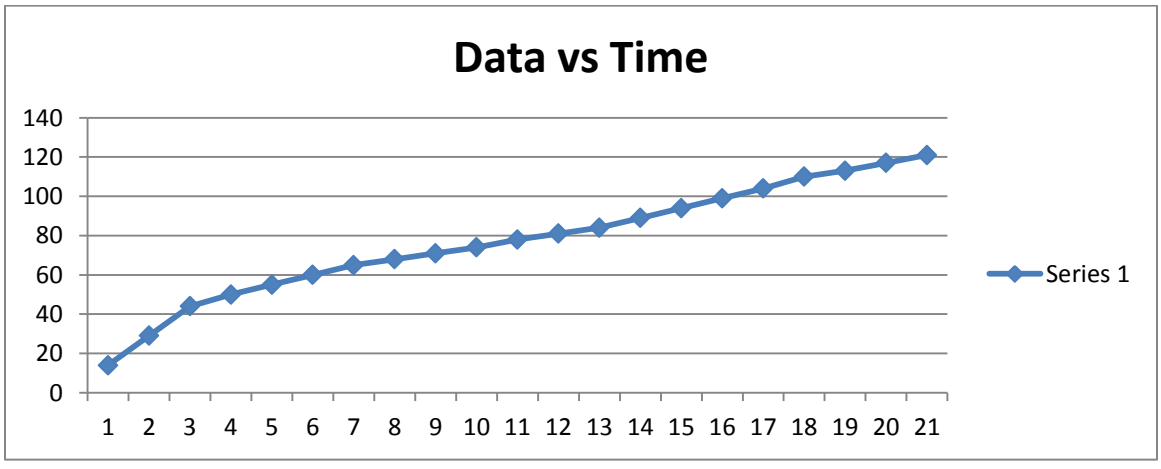
```
1: class Mapper
2: method Map(docid a, doc d)
3: for all term w ∈ doc d do
4: H ← new AssociativeArray
5: for all term u ∈ Neighbors (w) do
6: H{u} ← H{u} + 1
7: Emit (Term w, Stripe H)

1: class Reducer
2: method Reduce(term w, stripes [H1, H2, H3, . . .])
3: Hf ← new AssociativeArray
4: for all stripe H ∈ stripes [H1, H2, H3, . . .] do
5: Sum (Hf, H)
6: Emit (term w, stripe Hf )
```

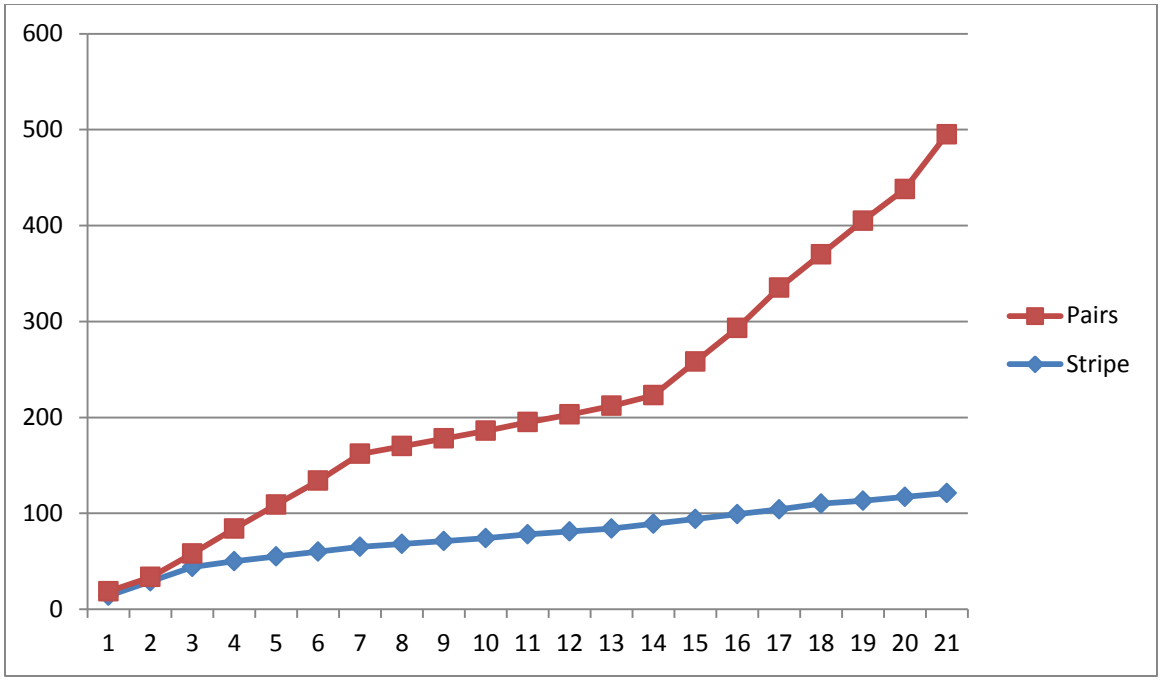
Performance Table

Data Size (kBs)	Time Taken (s)
300	44
700	65
1400	89
1800	104
2100	121

Performance Chart [X-Axis is data size (in 100KBs) Y-Axis is time (in seconds)]



Performance Comparison [X-Axis is data size (in 100KBs) Y-Axis is time (in seconds)]



For calculation of relative frequency of word1 w.r.t. word2, we need marginal occurrence count of word1, i.e. count of occurrence of word1 with all other word pairs. This calculation is straight forward in Stripes approach, as we can easily calculate marginal occurrence count of a word from its associative array. But in Pairs approach this count is not directly available. To get this marginal occurrence count, we emit <key, *> pair for each occurrence of word. Number of these special pairs w.r.t. to each word can be calculated to find the marginal occurrence of that word. But we must ensure that this calculation occurs before any other pair of that word is processed. Hence we have to implement custom partitioner to ensure that these special key-value pairs corresponding to each key reaches the reducer first. These are used to calculate the marginal occurrence count to compute relative frequency.

Please note:

1. I have computed timing data for some points (shown in respective tables) and interpolated for the rest.
2. I am running hadoop on a virtual machine (VMware). I am unable to run stripes approach on it for a large data set due to memory issues.