MapReduce

Divide a large data set into many smaller pieces for independent parallel processing. Combine and process intermediate results to obtain final result. Execution goes as follows:

0) User program breaks the input files into small pieces and starts up many copies of the program on different machines. One copy is the *Master* and assigns work to many *Workers.*
1) **Map Task:** Reads an input file fragment/shard and parses key/value pairs that are passed through the Map function and buffered in memory.
2) **Data Shuffle**: Buffered pairs are written to disk and partitioned. Locations of the partitioned regions are passed to the Master, who forwards these to Reduce workers.
3) **Reduce Task:** Reads buffered data and sorts by key. Key/value pairs are passed through the Reduce function and saved to an output file.
4) When all tasks complete, the algorithm is complete, and control returns to the user program.

Use psuedocode to write the MapReduce functions necessary to solve the below questions:

1. Computing the number of words at each possible length in a series of documents
   Input Key-Value Pairs: {document title, document text}

map(Object key, Object value):	**reduce(Object key, Iterable value):**

for word in value:
  emit(len(word), 1)
	total = 0
	while(value.hasNext):
	  total += value
	  value = value.next
	emit (key, total)

2. For each pair of friends, find the mutual friends between them.
   Input Key-Value Pairs: {Name, (List of Friend's Names)}

map(Object key, Object value):	**reduce (Object key, Iterable value):**
for current in value: ** if (key < current):** ** pair = (key, current)** ** else:** ** pair = (current, key)** ** list = for each friend in value except current** ** emit (pair, list)**	**arrayToCheckMutualFriends = []** **for current in value:** ** append current to arrayToCheckMutualFriends** **mutualFriends =** ** arrayToCheckMutualFriends.intersection()** **emit(key, mutualFriends)**

3. A) Computing the number of every coin in every person's possession (Quarters, dimes, etc.):
Input Key-Value Pairs: {person_name, coin_name}

map(Object key, Object value):	**reduce(Object key, Iterable value):**
emit((key, value), 1)	total = 0 while(value.hasNext) total += value.next emit (key,total)

B) Computing the amount of money every person has:
Input Key-Value Pairs: Output Key-Value Pairs of part A

map(Object key, Object value):	**reduce(Object key, Iterable value):**
emit (key.person, key.coin * value)	total = 0 while (value.hasNext) total += value.next emit(key,total)

Power Usage Effectiveness (PUE)

A measure of how efficiently a computer data center uses its power; specifically, how much of the power is actually used by the computing equipment (in contrast to cooling and other overhead).

PUE = (Total Building Power) / (IT Equipment Power)

- TBP = IT equipment + Power supplies + Networking equipment + Cooling equipment.
- Lower PUE = Most power going to IT equipment = Good power usage ☺

Warehouse Scale Computing (WSC)

Sources speculate Google has over 1 million servers. Assume each of the 1 million servers draws an average of 200W. Assume Google pays an average of 6 cents per kilowatt-hour for datacenter electricity.

a) Estimate Google's annual power bill for its datacenters. Ignore the power cost of networking equipment. Assume 365 days (8760 hours) in a year.

1,000,000 servers x 200 W/server x 6 cent/kW-hr x 8760 hrs/yr = $105.12M/yr

b) Suppose Google reduced the PUE in a 50,000 machine datacenter from 1.5 to 1.25 without significant infrastructure changes or decreasing the total power available for the datacenter. How much more power is available for servers? What's the total cost savings per server?

PUE: Total Building Power = 200W x 50,000 servers x 1.5 = IT Power x 1.5 = 15,000kW
Since Total Building Power is unchanged, new IT Power is 15,000kW / 1.25 = 12,000kW
So the difference is 12,000kW – 10,000kW = 2,000kW
2,000kW x 8760 hours * .06 dollars/KW-hr = $1,051,200