

MapReduce for Large Scale Computing Solutions to Think-Pair-Share Activities



1. MapReduce Exercise 1:

Case Analysis or Capitalization Probability: In a collection of text documents, find the percentage capitalization for each letter of the alphabet. That is, (number of occurrences of a letter that are capitalized/ total number of occurrences for that letter) $\times 100$.

```
file1: The happy Fox
file2: The THE THE
file3: And AND AND
```

```
result:
a: 3/4 * 100 = 75%
t: 4/4 * 100 = 100%
e: 1/4 * 100 = 25%
. . .
```

Solution

The main idea is to output a 2-tuple for both a lower case and upper case character that has the same key: the character in uppercase. The uppercase 2-tuple will have a value of "1" where as the lowercase one would have a value of "0". Then all the 2-tuples for a character will go to one reducer that can add the number of characters as well the number of upper case characters to finally output the percentage.

```
map(String key, String value):
// key: document name
// value: document contents (line by line)
1. for each character ch in value:
2. if (ch is uppercase)
3.     emitIntermediate(ch, 1)
4. else
5.     emitIntermediate(toUppercase(ch), 0)

reduce(String key, Iterable values):
// key: a character
// values: a list of counts
1. int total = 0
2. int sum = 0;
3. for v in values:
4.     sum += parseInt(v)
5.     total += 1
6. emit(key, asString(count * 100.0/total))
```

2. Top-N patents

Let us consider a simpler problem first.

- Find the number of citations for each patent in a patent reference data set. The format of the input is:

`citing_patent, cited_patent`

Assume that each document has a list of such references.

Here is a MapReduce algorithm to solve this problem. We assume that the mappers are fed one line at a time (with the line number as the key)

```
map(String key, String value):
// key: line number in a document (we ignore this)
// value: citing_patent, cited_patent
1. parse into two variables: cited_patent and citing_patent
2. emitIntermediate(cited_patent, 1)

reduce(String key, Iterable values):
// key: cited_patent
// values: a list of count values (each may be more than 1)
1. int sum = 0;
2. for v in values:
3.     sum += parseInt(v)
4. emit(key, asString(sum))
```

- Find the top N most frequently cited patents (assuming that the counts are unique). The format of the input is:

`citing_patent, cited_patent`

Describe a MapReduce algorithm to solve this problem. *Hint*: This will take two passes.

- **Solution 1.** Here is a pure MapReduce solution.

```
map1(String key, String value):
// key: line number in a document (we ignore this)
// value: citing_patent, cited_patent
1. parse into two variables: cited_patent and citing_patent
2. emitIntermediate(cited_patent, 1)

reduce1(String key, Iterable values):
// key: cited_patent
// values: a list of count values (each may be more than 1)
1. int sum = 0;
2. for v in values:
3.     sum += parseInt(v)
4. emit(key, asString(sum))

map2(String key, String value):
// key: line number in a document (we ignore this)
// value: cited_patent, count
1. parse into two variables: cited_patent and count
2. emitIntermediate(count, cited_patent) //flip the key!

// This reduce is basically a pass through so the sort is
// triggered before it
reduce(String key, Iterable values):
// key: cited_patent
// values: a count value (we will only have one at this stage)
1. emit(cited_patent, asString(count))
```

Now the patent counts are sorted across the cluster but we would have to write a sequential program to go find the top N values – so not a complete solution!

- **Solution 2:** Here is second way to get just the top N values. It will also be more efficient even though it isn't a purely functional solution.

```
map1(String key, String value):
// key: line number in a document (we ignore this)
// value: citing_patent, cited_patent
1. parse into two variables: cited_patent and citing_patent
2. emitIntermediate(cited_patent, 1)

setup1(): //for reduce1
1. initialize a global priority queue that will contain
   (patent, count) values, sorted by count
```

```

reduce1(String key, Iterable values):
// key: cited_patent
// values: a list of 1's
1.   int count = 0;
2.   for v in values:
3.       count += parseInt(v)
       // the priority queue is sorted by count and has N entries
       // if the size of the priority queue exceeds N,
       // we drop the lowest count entry
4.   insert (key, count) into a global priority queue of size N

cleanup1(): //for reduce1
1.   for each (patent, count) pair in the global priority queue:
2.       emit(patent, count)

map2(String key, String value):
// key: cited_patent
// value: count
1. Create a 2-tuple (cited_patent, count)
// null key forces all values to go to one reducer!
2. emitIntermediate(null, (cited_patent, count))

setup2(): //for reduce1
1.   initialize a global priority queue that will contain
       (patent, count) values, sorted by count

reduce2(String key, String value)
// key: null
// values: a list of 2-tuples (patent, count)
// the priority queue is sorted by count and has N entries
// if the size of the priority queue exceeds N,
// we drop the lowest count entry
1. insert (patent, count) into a global priority queue

cleanup2(): //for reduce1
1.   for each (patent, count) pair in the global priority queue:
2.       emit(patent, count)

```

- Note that this solution doesn't work if total counts aren't unique. For example, what if we have more N patents with the same count?