Massive data processing
Homework 1
Gaétan Morand

Configuration

Mac OS 10.11
Hadoop 2.7.3 installed with Homebrew
The standard output does not indicate the total CPU time, so I have to report the times with the UNIX « time » command.

a.i

To collect stopwords we use the same map function as a simple Wordcount Mapreduce program. The difference is that we use the reducer to filter only the words which appear more than 4,000 times.

In order to use 10 reducers we must add the line:

```
job.setNumReduceTasks(10);
```

Due to that the results are divided into 10 individual files, which we shall merge with the following command:

```
hdfs dfs -getmerge output ouput.csv
```

The program reports an execution time of 36.816s.

Here are the identified stopwords :

a
about
after
again
all
am
an
and
any
are
as
at
away
be
been
before
but
by
can
come
could
d
day
did
do
down
ever
every
first
for
from
get
go
good
got
great
had
has
have
he
her
here
him
his
how
i
if
in
into
is
it
its
just
king
know
let
like
little
ll
long
made
make
man
may
me
more
most
mr
much
must
my
never
no
not
nothing
now
o
of
old
on
one
only
or
other
our
out
over
own
s
said
say
see
shall
she
should
sir
so
some
such
t
take
than
that
the
their
them
then
there
these
they
thing
think
this
thou
thy
time
to
too
two
up
upon
us
very
was
way
we
well
were
what
when
where
which
who
will
with
would
yet
you
your

They are also in the file a.i.csv.
The code is in the file a.i.java.

a.ii

We add a combiner that does the same job as the reducer, except for the filtering part. The combiner adds up the counts of each word, but we leave it to the reducer to only keep the most used. Indeed this operation must be executed only at the end of the process.

```
public static class Combine extends Reducer<Text, IntWritable, Text, IntWritable> {

    public void combine(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum = sum + val.get();
        }
        context.write(key, new IntWritable(sum));
    }
 }
```

The resulting csv file is the same as the precedent one, but the program reports an execution time of 35.912s.

It is slightly faster this way because the combiner is called right after the map function, which is highly parallelized. In reality my processor only has 2 cores so parallelization does not add much.

The code is in the file a.ii.java.

a.iii

In order to compress the map results, I added the following lines:

```
conf.set("mapreduce.map.output.compress", "true");
conf.set("mapreduce.map.output.compress.codec",
"org.apache.hadoop.io.compress.SnappyCodec");
```

The program reports an execution time of 36.837s. It is longer than before. Indeed the limiting factor on my computer is the processing power, not the transfer time between memory and processor. So we waste more time compressing the data than we save transporting it.

The code is in the file a.iii.java.

a.iv

To use 50 reducers, we must add this line:

```
job.setNumReduceTasks(50);
```

The program reports an execution time of 40.850s. This is because my computer is only able to run 2 threads at a time since it has 2 cores. So there is no point in having more than 2 threads. It does not compute faster with 50 threads than with 10. It is even slower since we waste computing power dividing and merging the data, and managing these threads.

b.

To learn how to skip word, I read this website:
https://www.cloudera.com/documentation/other/tutorial/CDH5/topics/ht_wordcount3.html

There are two major components to build in this program: the part that skips stopwords in the map function, and the part that remove the duplicate filenames, in the reduce function.

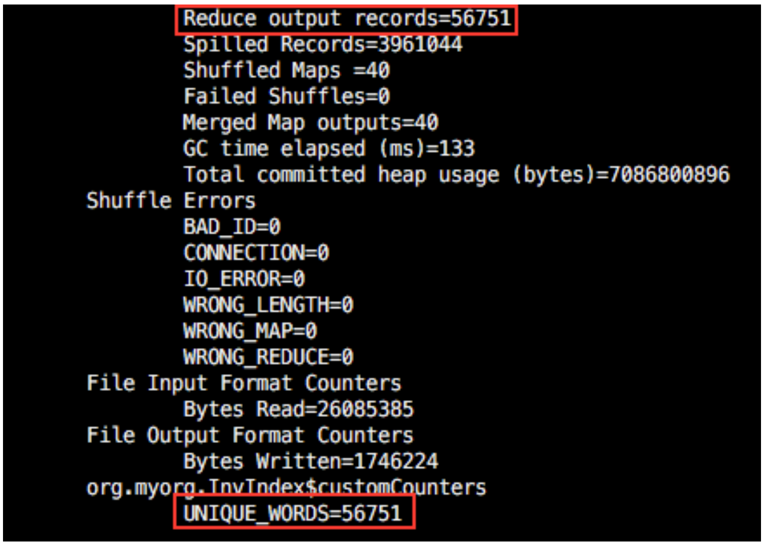
The output file is in the file b.txt
The code is in the file b.java

c.

The total number of words is indicated by the counter “Reduce output records”. In this case there are 56,751 different words.

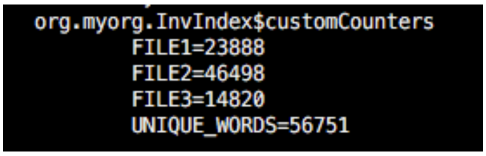
I created a new counter, which increments in each instance of the reduce method. It confirms that there are 56,751 different words.

Here is a screenshot of the output:



```
                Reduce output records=56751
                Spilled Records=3961044
                Shuffled Maps =40
                Failed Shuffles=0
                Merged Map outputs=40
                GC time elapsed (ms)=133
                Total committed heap usage (bytes)=7086800896
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=26085385
        File Output Format Counters
                Bytes Written=1746224
        org.myorg.InvIndex$customCounters
                UNIQUE_WORDS=56751
```

Then we have to create a new counter for each file. We can only count in the Reduce method because we want to avoid duplicates. We test for the name of the file.



```
        org.myorg.InvIndex$customCounters
                FILE1=23888
                FILE2=46498
                FILE3=14820
                UNIQUE_WORDS=56751
```

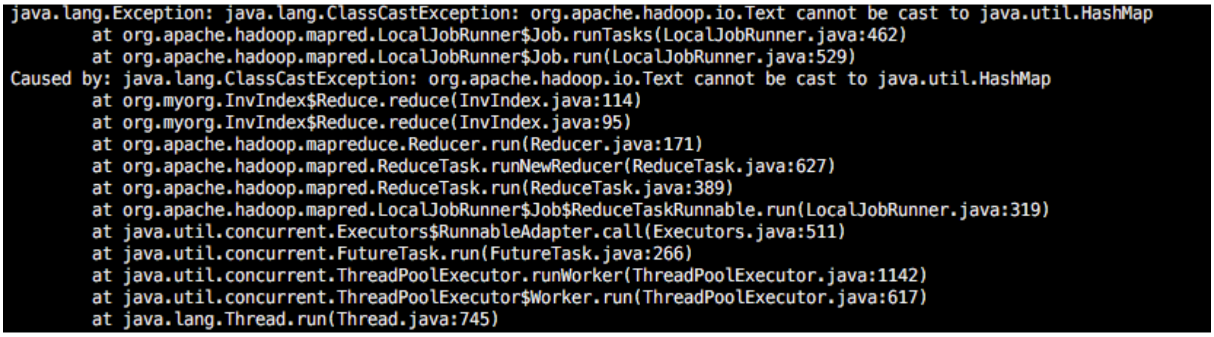
The code is in the file c.java
The output file is counters.txt

d.

Here we have to take back the code from our inverted index. The difficulty is to maintain a different wordcount for each file. To do this I chose to use a Map class, in which the key represents the file name, and the value represents the number of occurrences.

I am sure it can also be done using two mapreduces sequentially but I am not sure how to program this.

I had a bug in the final problem which I uploaded.



```
java.lang.Exception: java.lang.ClassCastException: org.apache.hadoop.io.Text cannot be cast to java.util.HashMap
        at org.apache.hadoop.mapred.LocalJobRunner$Job.runTasks(LocalJobRunner.java:462)
        at org.apache.hadoop.mapred.LocalJobRunner$Job.run(LocalJobRunner.java:529)
Caused by: java.lang.ClassCastException: org.apache.hadoop.io.Text cannot be cast to java.util.HashMap
        at org.myorg.InvIndex$Reduce.reduce(InvIndex.java:114)
        at org.myorg.InvIndex$Reduce.reduce(InvIndex.java:95)
        at org.apache.hadoop.mapreduce.Reducer.run(Reducer.java:171)
        at org.apache.hadoop.mapred.ReduceTask.runNewReducer(ReduceTask.java:627)
        at org.apache.hadoop.mapred.ReduceTask.run(ReduceTask.java:389)
        at org.apache.hadoop.mapred.LocalJobRunner$Job$ReduceTaskRunnable.run(LocalJobRunner.java:319)
        at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
        at java.util.concurrent.FutureTask.run(FutureTask.java:266)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
        at java.lang.Thread.run(Thread.java:745)
```

I was unfortunately not able to debug this.

I uploaded the code in the file d.java. I think it is really close to working, but I have to upload now to meet the deadline.