Testing Your Map-reduce Job Outside of Hadoop

Roy E Lowrance

September 30, 2013

1 Problem

You want to test your map-reduce job before you submit it to Hadoop, perhaps because the debugging tools in the Hadoop streaming interface are lacking.

2 Solution

The solution has three parts:

- Write the mapper so that it can process less than all the input.
- In the mapper, write debugging information into the key-value file by using keys designated for that purpose. Modify the reducer to handle these keys appropriately.
- Write a shell script to run map-reduce locally, without Hadoop.

2.1 Map a subset of the input

Write your map program so that it can run from the command line and as a Hadoop command. When it runs from the command line, you may need to limit the amount of input it reads, if it reads a lot of input or creates a lot of key-value pairs for your reduce job.

To do this, write one map program that has an optional parameter. If the optional parameter is present, its value is used to limit the scope of the map program. One way to do this is to have let the optional parameter be READLIMIT, an integer with a default value of -1. If READLIMIT is non-negative, the map program stops after reading the indicated number of input records. If READLIMIT is negative, the map program reads and processes all the input records. (The READLIMIT idea is common in R for controlling how many records are read and processed from a file.)

2.2 Put debugging information into the key-value pairs

The reducer will read all the key-value pairs after they have been combined and sorted by key. Designate one set of keys as only containing debugging information and have the reducer handle records with these keys specially. The keys and values are strings. One choice is to say that any key beginning with a "#" is for debugging work, not regular processing.

2.3 Shell script

Running the job is work that is best delegated to a script. We propose that the script take the same initial parameters as the script `map-reduce-hadoop.sh` described in the recipe "Run Many Hadoop Jobs". Thus the parameters of the script will be

- INPUT–name of the input file. This is also the first parameter in `map-reduce-hadoop.sh`.
- JOB–name of the job. The name of the job, the same paraemeter as for `map-reduce-hadoop.sh`. The mapper is `JOB-map.lua` and the reducer is `JOB-reduce.lua`.
- READLIMIT–integer. Not a parameter to `map-reduce-hadoop.sh`.

3 Discussion

The sample code is a modification of the code used for the recipe "Using Torch on Hadoop."

The mapper program is modified to accept the optional READLIMIT argument and to write diagnostic information to its output stream using keys that begin with "#".

```
#!/home/rel292/local/bin/torch-lua
-- count number of records and bytes in stdin map function
-- stdin format: records consisting of pairs of strings
--   (keyname --> count)
-- stdout format:
--   (keyname --> sum of counts for that keyname)
--
-- COMMAND LINE ARGUMENTS
-- READLIMIT optional integer default -1
--           if >= 0, read only READLIMIT input records

require "getKeyValue"

local readlimit = -1
if arg[1] ~= nil then
    readlimit = tonumber(arg[1])
```

```
end

print('# readlimit \t' .. tostring(readlimit))

local records = 0
local keyBytes = 0
local valueBytes = 0
local nRead = 0
for line in io.stdin:lines("*l") do
    nRead = nRead + 1
    if readlimit >= 0 and nRead > readlimit then
        break
    end

    if line == nil then break end
    records = records + 1
    local key, value = getKeyValue(line)
    keyBytes = keyBytes + string.len(key)
    valueBytes = valueBytes + string.len(value)
end
print("records" .. "\t" .. tostring(records))
print("keyBytes" .. "\t" .. tostring(keyBytes))
print("valueBytes" .. "\t" .. tostring(valueBytes))
```

The reducer program looks for the diagnostic records and writes them to stderr. The output is written to stdout. In some uses, you may want to write the diagnostic output to stdout as well.

```
#!/home/rel292/local/bin/torch-lua
---- count number of records in standard in reduce function
-- this is the reduce
-- input is in some arbitrary order
--   ("records" --> <number of records>) ...
--   ("keyBytes" --> <number of bytes in keys>) ...
--   ("valueBytes" --> <number of bytes in values>) ...
-- output, written to stdout, is total for each key value
-- NOTE: keys are arbitary strings, not restricted to value named above
-- Comment records (beginning with #) are written to stderr

require "getKeyValue"

local lastKey = nil
local count = 0
for line in io.stdin:lines("*l") do
    if line == nil then break end
```

```
    local key, value = getKeyValue(line)
    if string.sub(key, 1, 1) == '#' then
       io.stderr:write('comment record: ' .. line .. '\n')
    else
       if lastKey == key then
           count = count + tonumber(value)
       else
           if lastKey then
               print(lastKey .. "\t" .. count)
           end
           lastKey = key
           count = tonumber(value)
       end
   end
end

if lastKey then
    print(lastKey .. "\t" .. count)
end
```

The shell script `map-reduce-local.sh` is just below. The only trick is that you need to define the output directory before running the script.

```
# run map-reduce locally

# capture arguments
INPUT=$1
JOB=$2
READLIMIT=$3

# directory for local map-reduce output
LOCAL_MAP_REDUCE_OUTPUT=map-reduce-output-local

# do the work
./$JOB-map.lua $READLIMIT < $INPUT | \
  sort -k 1 | \
  ./$JOB-reduce.lua > $LOCAL_MAP_REDUCE_OUTPUT/$INPUT.$JOB
```

The script `go-map.sh` runs just the mapper. It executes the mapper program as a command.

```
# run just the mapper
./countInput-map.lua 2 < courant-abel-prize-winners.txt
```

The shell script `go-map-reduce.sh` knowns the names of the input file and job and the READLIMIT.

```
# run map-reduce locally
./map-reduce-local.sh courant-abel-prize-winners.txt countInput 2
```

Below is the input file `courant-abel-prize-winners.txt`.

```
2005 Peter Lax
2007 S. R. Srinivasa Varadhan
2009 Mikhail Gromov
```

4 See also

This recipe is free documentation. You can modify it by visiting github for account "rlowrance" and forking the repo "torch-cookbook."