

Read Files Other Than Just stdin in Hadoop

Roy E Lowrance

October 6, 2013

1 Problem

You want to read more than stdin in either your mapper or reducer. We call these extra files the auxillary files. The auxillary files might contain tables needed by the mapper or reducer. One use case is when you are selecting which of several models to use and you need to estimate the generalization error for each of the models. In this case, the input file could be the test data and the auxillary file the training data.

2 Solution

One supposed easy solution—just open the auxillary files in the mapper and reducer and read them—doesn’t work because the auxillary files will not be in the working directory of the mapper or reducer.

Another supposed easy solution—specify multiple input files on the `-file` option when starting map-reduce—doesn’t work because each mapper will see only a portion of the input file on its stdin. In this use case, you want each mapper to see all of the auxillary files.

The solution is to ship the auxillary files to the working directory that is used by the mapper and reducer. The mapper and reducer then open the files using their file names and read them.

3 Discussion

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

The sample scripts in this section copy specific files to the working directory on the execution nodes. This is done to make things as clear as possible. However, you may want to copy an entire directory of files to the local system. Or you may want to copy only certain files in a directory. That can be done by specifying a directory instead of a single file or by specifying a file glob as in `/logs/2013/09/*.log`.

The mapper program is modified to read an auxillary file. To test that the read actually works, the records in the auxillary file are written to the mappers stdout as comments, using our convention that a comment begins with a key starting 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)
--
-- Also echo an auxillary file.
--
-- 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))

-- read the auxillary file
-- echo its records to output as commennts
-- our convention is that comments being with a #

local auxFilename = "auxillary-mapper.txt"
local nAuxRecords = 0
for auxRecord in io.lines(auxFilename) do
    if auxRecord == nil then break end
    nAuxRecords = nAuxRecords + 1
    print("# mapper aux " .. tostring(nAuxRecords) .. "\t" .. auxRecord)
end

local records = 0
local keyBytes = 0
local valueBytes = 0
local nRead = 0
for line in io.stdin:lines("*1") 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 is also modified to read an auxillary file. The debugging records from the mapper and the records from the reducers's auxillary files are written to stderr. Note that writing to stderr in the reducer is OK for debugging when running in the local file system. However, when running under Hadoop, CHECK THIS the stderr is not available as an output of the map-reduce job.

```

#!/home/rel292/local/bin/torch-lua
---- count number of records in standard in reduce function
---- read and echo an auxillary file
-- 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 arbitrary strings, not restricted to value named above
-- Comment records (beginning with #) are written to stderr

require "getKeyValue"

local function writeStderr(line)
    io.stderr:write(line .. "\n")
end

-- read the auxillary file
-- echo its records to output as commennts
-- our convention is that comments being with a #

local auxFilename = "auxillary-reducer.txt"
local nAuxRecords = 0
for auxRecord in io.lines(auxFilename) do
    if auxRecord == nil then break end

```

```

        nAuxRecords = nAuxRecords + 1
        writeStderr("# reducer aux " .. toString(nAuxRecords) .. ": " .. auxRecord)
    end

    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
            writeStderr('comment record from mapper: ' .. line)
        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
end

```

The shell script `map-reduce.sh` is just below. It runs a map reduce job on hadoop. The arguments to the script are the input file name in the Hadoop file system, the job name, the name of the auxillary input file in the local file system for the map command, the name of the auxillary input file in the local file system for the output file. The auxillary files are copied to the execution nodes.

NOTE: WE SHOULD FIGURE OUT HOW TO VIEW THE STDERR OF THE REDUCE STEP WHEN THE MAP-REDUCE JOB IS RUN ON THE HADOOP SYSTEM.

```

# run map reduce job using streaming interface
# USAGE:
# Change to the directory where your source code and scripts live. Then
# enter
# ./map-reduce.sh INPUT JOB AUX_MAP AUX_REDUCE [READLIMIT]

```

```

# where
# INPUT      is the name of your input file in the Hadoop file system
#            If not in the Hadoop file system, must be in the local
#            file system, in which case, it is copied to the Hadoop
#            file system.
# AUX_FILES  is a list of auxillary files to be read by either the
#            map or reduce command. Files in this list are copied to the
#            local execution nodes.
# JOB        is the name of your map reduce job. The mapper command is
#            JOB-map.lua is the mapper command
#            JOB-reduce.lua is the reducer command
# READLIMIT  for compatibility with your local map-reduce job runner.
#
# The script copies the output of the job from the Hadoop file system
# to $HOME/map-reduce-output

set -x # print each command before executing it

# 1: capture command line arguments
INPUT=$1
JOB=$2
AUX_MAP=$3
AUX_REDUCE=$4
# READLIMIT IS NOT USED

# 2: input and output paths
INPUT_PATH=/home/$USER/$INPUT
OUTPUT_DIR_HADOOP=$INPUT.$JOB
MAP_REDUCE_OUTPUT=map-reduce-output
OUTPUT_DIR_LOCAL=$HOME/$MAP_REDUCE_OUTPUT/$OUTPUT_DIR_HADOOP

# 3: where Hadoop lives
HADOOP_HOME=/usr/lib/hadoop
STREAMING="hadoop-streaming-1.0.3.16.jar"

# 4: copy test file to the hadoop file system if it is not already there
hadoop fs -test -e $INPUT
if [ $? -ne 0 ]
then
    echo copying input from local file system to hadoop file system
    hadoop fs -copyFromLocal $INPUT $INPUT
else
    echo input file already in the hadoop file system
fi

```

```

# 5: delete output directory from previous run if it exists
hadoop fs -test -e $OUTPUT_DIR_HADOOP
if [ $? -eq 0 ]
then
    echo deleting output directory: $OUTPUT_DIR_HADOOP
    hadoop fs -rmr $OUTPUT_DIR_HADOOP
else
    echo output directory not in the hadoop file system
fi

# 6: run the streaming job; it will create the output directory
echo starting hadoop streaming job
echo AUX_MAP=$AUX_MAP
echo AUX_REDUCE=$AUX_REDUCE
hadoop jar $HADOOP_HOME/contrib/streaming/$STREAMING \
    -file *.lua \
    -file $AUX_MAP -file $AUX_REDUCE \
    -mapper $PWD/$JOB-map.lua \
    -reducer $PWD/$JOB-reduce.lua \
    -input $INPUT \
    -output $OUTPUT_DIR_HADOOP
echo finished hadoop job

# 7: copy output file to home directory
# delete output directory if it already exists
# We delete in case it has old content that is not overwritten by copyToLocal
echo output dir local=$OUTPUT_DIR_LOCAL
echo output dir hadoop=$OUTPUT_DIR_HADOOP
if [ -a $OUTPUT_DIR_LOCAL ]
then
    # local output directory exists, so delete it
    # run in subshell so cd has only temporary effect
    (cd $OUTPUT_DIR_LOCAL; rm -rf -- $OUTPUT_DIR_LOCAL)
fi

mkdir -p $OUTPUT_DIR_LOCAL # copyToLocal wants the directory to already exist
echo about to copy to local from $OUTPUT_DIR_HADOOP
hadoop fs -copyToLocal $OUTPUT_DIR_HADOOP $HOME/$MAP_REDUCE_OUTPUT/
#hadoop fs -copyToLocal courant-abel-prize-winners.txt.countInput /home/rel292/map-reduce-out

# 8: list output dir
echo LOCAL OUTPUT DIRECTORY
ls $OUTPUT_DIR_LOCAL

```

For ease of testing, I wrote two scripts. `go-map-reduce-hadoop.sh` run the

test on Hadoop. Here it is.

To test on the local file system I wrote `go-map-reduce-local.sh`.

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

The generic scripts to run on the local file system is here.

```
# 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 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` knows 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

Apache provides documentation on map-reduce at wiki.apache.org/hadoop/HadoopStreaming. An example of providing log files as auxillary input is provided.

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