A Guide to Transfer Hundred Million Small Files to S3 Using S3 SnowbailEdge

Date: 2019.12.30 Yongki, Kim (kyongki@)

Background

It is really hassle to move mass small files to S3, however my customer asked me to do it. And so do I. The condition was here. Each file size is under 200K and numbers of files are about a hundred and 30 million, moreover the file name should be changed because he wanted to restructure files and directories architecture. I attempted many of things to find out the suitable solution, but those solutions had its own limitation. At least, I created very useful and efficient python script to move it fast and without consuming bunch of memory and of disk spaces. Final version is here, if you don't want to read whole story and just know the final script, jump to here; Go to Final Section Using this solution, you will transfer hundred millions files in a week!! But, BE AWARE OF that consuming time will depend on the storage's read time, network bandwidth, and memory size and other infra component.

Customer's Request

- A number of files: approximate 130,000,000 files
- Each file size: mostly under 200K, but mixed with mega-bytes files
- Source Storage: all files stored in NAS and this storage was on IDC, not connected with AWS
- File name: every file's name should be changed
    - customer provided a manifest file containing original directory and file name

Snowball Edge Manual

- snowball edge data migration: https://d1.awsstatic.com/whitepapers/snowball-edge-data-migration-guide.pdf?did=wp_card&trk=wp_card

Building The Test Environment

At first, I didn't know that how could I start. I asked all SA in my company to get some insights. They replied eagerly with some useful tools, as like *data sync*, *s3 batch operation*, and *tar's transform* option and so on. Based on my research and replies from my colleagues,I started to make test environment. Test environment consists of 100 directories with 100 files.

```
#!/bin/bash
# file name: create_dummy_files.sh
s3_org="your-own-org-bucket"
s3_dest="your-own-dest-bucket"

dir_count=100
file_count_per_dir=100

function create_dir_structure {
    for i in $(seq 1 $dir_count);do
        mkdir dir$i
    done
}

function create_files {
    for i in $(seq 1 $dir_count);do
        for j in $(seq 1 $file_count_per_dir);do
            dd if=/dev/urandom of=dir$i/dir$i_file_$j.raw bs=1k count=200
        done
    done
}

function create_s3_bucket {
    aws s3 mb s3://[$s3_org $s3_dest]
```

```
}

function file_sync {
    aws s3 sync . s3://$s3_org
}

function shuffle_files_on_s3 {
    for i in {1..100};do
        for j in {1..100};do
#            aws s3 mv s3://$s3_org/dir$i/dir$i_file_$j.raw s3://$s3_dest/dir$((i+10))/dir$((i+10))_file_$
           aws s3 mv s3://$s3_org/dir$i/file_$j.raw s3://$s3_dest/dir$((i+10))/dir$((i+10))_file_$((j+10)
        done
    done
}

## main
create_dir_structure
echo "dir structures are created"

create_files
echo "files are created"
```

The 1st Try: S3 Batch Operation

After building the environment, I run just simple python scripts to measure the consuming time. To shorten the time, I tested it with 100 files.

initial test result

```
- in single bucket, sync (keeping filenames)
  - cmd: time aws s3 sync s3://alex-s3-mv-org/dir99/ s3://alex-s3-mv-org/dest-dir99/
  - result: real    0m6.473s


- in single bucket, copy (changing filename with python)
  - cmd: copy_object(src_bucket_name, src_object_name, dest_bucket_name, dest_object_name)
  - result: real    1m36.594s


- in single bucket, upload (changing filename with python)
  - cmd: S3.upload_file(SOURCE_FILENAME, BUCKET_NAME, DEST_FILENAME)
  - result: real    0m58.813s
```

As you notice that, **aws sync** showed the best performance but I couldn't do it because **aws sync** don't provide renaming function. So I decided to upload it with own name and change the name with **s3 batch operation**. S3 Batch Operation's performance is very good, only took 40 sec to change the names of 10,000 file.

S3 Batch Operation

Test result

Viewing 1 to 17 of 17 jobs

Job ID	Description	Operation	Date created	Status	Total objects	% Complete	Total failed (rate)	Priority
f66035df-f3d6-458d-8472-1d7b9163be86	2019-12-12 Lambda	Invoke AWS Lambda function	Dec 12, 2019 6:21:01 PM GMT+0900	Complete	10,000	100%	0 (0%)	10

References to configure S3 Batch Operation

1. Official Documents: https://docs.aws.amazon.com/AmazonS3/latest/dev/batch-ops-invoke-lambda.html
2. S3 Batch Operation Blog: https://aws.amazon.com/blogs/aws/new-amazon-s3-batch-operations/
3. Lambda Code to rename file: https://docs.aws.amazon.com/AmazonS3/latest/dev/batch-ops-invoke-lambda.html#batch-ops-invoke-lambda-create-job
    - search keyword: *rename_key* in above document

The Lesson Learn from The 1st Try

The customer ordered the 1st snowballEdge to migrate files and run the file moving script. But it took too long time whether changing the file name or not, so I couldn't propose s3 batch operation. Moving fast to snowball is the urgent issue. It took 5 day to move only 10% of the entire files. To solve this problem, I concluded that the only solution is to archive the files and send it snowball to eliminate the connection time of all each snowball connection per all each file.

The 2nd Try: smart_open

I could make a python script to archive with tar, but customer didn't want to save the tar file on disk. I modified the script to archive files on memory and send it seamlessly. Sadly soon, I found out this solution caused memory error easily. So I sought to find a way to send a chunk as soon as it archived a part of them. **smart_open** package gave me a light to solve this problem. Using it, the script could support on-fly tar archiving and sending to snowball(or s3). Below is the that script.

references

- batching small files: https://docs.aws.amazon.com/snowball/latest/developer-guide/batching-small-files.html
- smart_open: https://github.com/RaRe-Technologies/smart_open

```
#!/bin/python

import boto3
import smart_open
import tarfile
import io
import os.path
from datetime import datetime

## 'smart_open' package is needed to run this program
# pip install smart_open

bucket_name = "alex-s3-mv-dest-seoul"
s3 = boto3.client('s3', 'ap-northeast-2')
transport_params = {'session': boto3.Session(profile_name='snowball'), 'resource_kwargs': { 'endpoint_url'
tarfiles_one_time = 2000
source_file = 'filelist_dir1_10000.txt'

#org_file = ["Logs/a.txt", "Logs/b.txt"]
def rename_file(org_file):
    return org_file.replace('\n','') + "_new_buffer"
org_files = open(source_file).readlines()
target_files = list(map(rename_file, org_files))

#### don't need to modify from here
current_time = datetime.now().strftime("%Y%m%d_%H%M%S")
error_file = ('error_tar_%s.log' % current_time)
successlog_file = ('success_tar_%s.log' % current_time)
batch_tar = ('snowball_batch_%s.tar' % current_time)
key_name = batch_tar

#s3_session = boto3.Session( aws_access_key_id='XXXXXXXXXXXXXXXXXXXX', aws_secret_access_key='XXXXXXXXXXX
#transport_params = {'session': s3_session , 'resource_kwargs': { 'endpoint_url': 'https://s3.ap-northeast
out = io.BytesIO()
line_break = len(org_files) / tarfiles_one_time + 1
final_line_list = [ i*tarfiles_one_time-1 for i in range(1,line_break)]
final_line_list.append(len(org_files)-1)
s3_location = "s3://" + bucket_name + "/" + batch_tar

def add_metadata_to_s3(bucket_name, batch_tar):
    s3.copy_object(Key=key_name, Bucket=bucket_name,
                CopySource={"Bucket": bucket_name, "Key": key_name},
                Metadata={"snowball-auto-extract": "true"},
                MetadataDirective="REPLACE")
def log_non_exist(org_file):
    with open(error_file,'a+') as err:
        err.write(org_file + " does not exist\r\n")
def log_success(target_file):
    with open(successlog_file,'a+') as success:
        success.write(target_file + " is archived successfully\r\n")
```

```
with smart_open.open(s3_location, 'wb', transport_params=transport_params) as fout:
    with tarfile.open(fileobj=out, mode="w") as tar:
        for index in range(len(org_files)):
            org_file = org_files[index].replace('\n','')
            target_file = target_files[index]
            if os.path.isfile(org_file):
                tar.add(org_file, arcname=target_file)
                #print target_file + " is uploading"
                log_success(target_file)
            else:
                log_non_exist(org_file)
                print org_file + " is not exist.............................................."
            #print "tar, out size: " + str(sys.getsizeof(out))
            if index in final_line_list:
                print "sending to s3.............................................................."
                #s3.upload_fileobj(out, bucket_name, 'batch41.tar',ExtraArgs={'Metadata': {'snowball-auto-
                fout.write(out.getvalue())
                out.seek(0)
                out.truncate()
try:
    add_metadata_to_s3(bucket_name, batch_tar)
except:
    print "updating metadata failed"
else:
    meta_out = s3.head_object(Bucket=bucket_name, Key=key_name)
    print ('metadata info: %s' % str(meta_out))
    log_success(str(meta_out))
```

The Challenge of The 2nd Try

It worked very well when I tested in S3. It shorten the consuming time from 2h30m to 30min. The problem occurred when adding the **metadata: snowball-auto-extract=true**. Adding metadata is very crucial because it makes tar file extract automatically when uploading to S3. But snowballedge doesn't support **copyobject** api which make adding metadata is possible, while snowball provides s3 interface api. Without this metadata, all my effort to create the script was worthless.

But customer already ordered and received the 2nd snowballedge as soon as he returned the 1st snowballedge, so we had no time to fix it, and decided to use it anyway. And we added some script to download the tar file and adding metadata with below command. Even though combining this script and some manual job was not perfect, It reduced huge time comparing to previous work. With this process, we migrated all 90% remaining files(about 100 million files) and the customer satisfied with finishing it within 5days.

```
aws s3 cp - s3://mybucket/batch01.tar --metadata snowball-auto-extract=true --endpoint http://192.0.2.0:80
```

Final and Successful Solution: multi-part uploading

However I couldn't content with this result when the script was incomplete still. So, I tried to find out how to remove the dependency of *smart_open* and produce the same result while adding metadata. The answer was in the source code in the *smart_open*. I had dug into the source code of *smart_open* and *multi-part uploading** was used there to deal with streaming data. Here is the final script, the complete version.

```
#!/bin/python

'''
filename: tar_to_s3_v7_multipart.py
status: completed
version: v7
way: using multi-part uploading
ref: https://gist.github.com/teasherm/bb73f21ed2f3b46bc1c2ca48ec2c1cf5
'''

import boto3
import tarfile
import io
import os.path
from datetime import datetime
import sys
```

```
bucket_name = "your-own-dest-seoul"
s3 = boto3.client('s3', region_name='ap-northeast-2')
#s3 = boto3.client('s3', region_name='ap-northeast-2', endpoint_url='https://s3.ap-northeast-2.amazonaws.c
tarfiles_one_time = 1000
source_file = 'filelist_dir1_10000.txt'

## Caution: you have to modify rename_file function to fit your own naming rule
def rename_file(org_file):
    return org_file.replace('\n','') + "_new_buffer"
org_files_list = open(source_file).readlines()
target_files_list = list(map(rename_file, org_files_list))
## to use same name (org file name == target file name), uncomment below line
#target_files_list = org_files_list

#### don't need to modify from here
current_time = datetime.now().strftime("%Y%m%d_%H%M%S")
error_file = ('error_%s_%s.log' % (source_file, current_time))
successlog_file = ('success_%s_%s.log' % (source_file, current_time))
batch_tar = ('snowball-batch-%s-%s.tar' % (source_file, current_time))
key_name = batch_tar
out = io.BytesIO()
parts = []

line_break = len(org_files_list) / tarfiles_one_time + 1
final_line_list = [ i*tarfiles_one_time-1 for i in range(1,line_break)]
final_line_list.append(len(org_files_list)-1)
s3_location = "s3://" + bucket_name + "/" + batch_tar

def add_metadata_to_s3(bucket_name, batch_tar):
    s3.copy_object(Key=key_name, Bucket=bucket_name,
                CopySource={"Bucket": bucket_name, "Key": key_name},
                Metadata={"snowball-auto-extract": "true"},
                MetadataDirective="REPLACE")
def log_error(org_file, str_suffix):
    with open(error_file,'a+') as err:
        err.write(org_file + str_suffix)
def log_success(target_file, str_suffix):
    with open(successlog_file,'a+') as success:
        success.write(target_file + str_suffix)

def flush_mem(out):
    out.seek(0)
    out.truncate()

def create_mpu():
    mpu = s3.create_multipart_upload(Bucket=bucket_name, Key=key_name, Metadata={"snowball-auto-extract":
    mpu_id = mpu["UploadId"]
    return mpu_id

def upload_mpu(mpu_id, data, index):
    part = s3.upload_part(Body=data, Bucket=bucket_name, Key=key_name, UploadId=mpu_id, PartNumber=index)
    parts.append({"PartNumber": index, "ETag": part["ETag"]})
    print ('parts list: %s' % str(parts))
    return parts

def complete_mpu(mpu_id, parts):
    result = s3.complete_multipart_upload(
        Bucket=bucket_name,
        Key=key_name,
        UploadId=mpu_id,
        MultipartUpload={"Parts": parts})
    return result

def main():
    mpu_id = create_mpu()
    parts_index = 1

    with tarfile.open(fileobj=out, mode="w") as tar:
        for index in range(len(org_files_list)):
            org_file = org_files_list[index].replace('\n','')
            target_file = target_files_list[index]
            if os.path.isfile(org_file):
                tar.add(org_file, arcname=target_file)
                #print target_file + " is uploading"
                log_success(target_file, " is archived successfully\n")
```

```
            else:
                log_error(org_file," does not exist\n")
                print org_file + " is not exist.........................................."
            #print "tar, out size: " + str(sys.getsizeof(out))
            if index in final_line_list:
                print "sending to s3.............................................................."
                print target_file + " is uploading"
                mpu_parts = upload_mpu(mpu_id, out.getvalue(), parts_index)
                parts_index += 1
                out.seek(0)
                out.truncate()

    print(complete_mpu(mpu_id, mpu_parts))
    ### print metadata
    meta_out = s3.head_object(Bucket=bucket_name, Key=key_name)
    print ('\n\n metadata info: %s' % str(meta_out))
    log_success(str(meta_out), '!!\n')

    print ("\n\n tar file: %s" % key_name)
    log_success(key_name, ' is uploaded successfully')

if __name__ == "__main__":
    main()
```

Code Explanation

At this point, I will explain some important point of this code.

```
tarfiles_one_time = 1000
```

This variable means a number of files which can be on memory, and if 1,000 files are archived and accumulated on memory, it will send to snowball or s3. This value can be modified depending on the system's memory size.

```
source_file = 'filelist_dir1_10000.txt'
```

source_file variable is the manifest file which containing original file list. All files listed in manifest will be archived in one file. So user should split the file lists not to be over 100GB.

- snippet of source_file

```
➜  s3_snowball git:(master) ✗ cat filelist_dir1_10000.txt | head -n 10
Logs/dir1/file_1
Logs/dir1/file_2
Logs/dir1/file_3
Logs/dir1/file_4
Logs/dir1/file_5
Logs/dir1/file_6
Logs/dir1/file_7
Logs/dir1/file_8
Logs/dir1/file_9
Logs/dir1/file_10
```

```
    def rename_file(org_file)
```

This *rename_file* function make new file name with specified rule set. If user don't want to change the file name, comment it.

```
    error_file = ('error_%s_%s.log' % (source_file, current_time))
    successlog_file = ('success_%s_%s.log' % (source_file, current_time))
```

All archived files will be listed in *successlog_file* and if original file doesn't exist in the directory, this information will stored in *error_file*. This log will be useful to check which file is failed or not.

```
    batch_tar = ('snowball-batch-%s-%s.tar' % (source_file, current_time))
```

batch_tar and *key_name* are same variable, it means the name of tar file.

```
    def add_metadata_to_s3(bucket_name, batch_tar):
    def flush_mem(out):
```

Both *add_metadata_to_s3* and *flush_mem* is not used in this code, but I remained it for future use.

```
    def create_mpu():
    def upload_mpu(mpu_id, data, index):
    def complete_mpu(mpu_id, parts):
```

These three functions are core component of this scripts. Using this *multi_part_upload* api, I can reduce the usage of memory and disk.

```
    out = io.BytesIO()
    out.getvalue()
```

out variable is dealing with streaming data, and *out.getvalue()* transfer the tar archived data into the sending buffer to snowball.

Output of When Script run

Here is the output. *file_999* and *file_1000* are in error, I deleted it intentionally to check whether error is detected well or not.

```
[ec2-user@ip-172-31-38-36 tmp]$ time python tar_to_s3_v7_multipart.py
Logs/dir1/file_999 is not exist.............................................
Logs/dir1/file_1000 is not exist.............................................
sending to s3.............................................................
Logs/dir1/file_1000_new_buffer is uploading
parts list: [{'PartNumber': 1, 'ETag': '"fe4b9305c800c1fac9a81a1e940c4c84"'}]
sending to s3.............................................................
Logs/dir1/file_2000_new_buffer is uploading
parts list: [{'PartNumber': 1, 'ETag': '"fe4b9305c800c1fac9a81a1e940c4c84"'}, {'PartNumber': 2, 'ETag': '"
sending to s3.............................................................
Logs/dir1/file_3000_new_buffer is uploading
parts list: [{'PartNumber': 1, 'ETag': '"fe4b9305c800c1fac9a81a1e940c4c84"'}, {'PartNumber': 2, 'ETag': '"
sending to s3.............................................................
Logs/dir1/file_4000_new_buffer is uploading
parts list: [{'PartNumber': 1, 'ETag': '"fe4b9305c800c1fac9a81a1e940c4c84"'}, {'PartNumber': 2, 'ETag': '"
sending to s3.............................................................
Logs/dir1/file_5000_new_buffer is uploading
parts list: [{'PartNumber': 1, 'ETag': '"fe4b9305c800c1fac9a81a1e940c4c84"'}, {'PartNumber': 2, 'ETag': '"
sending to s3.............................................................
Logs/dir1/file_6000_new_buffer is uploading
parts list: [{'PartNumber': 1, 'ETag': '"fe4b9305c800c1fac9a81a1e940c4c84"'}, {'PartNumber': 2, 'ETag': '"
sending to s3.............................................................
Logs/dir1/file_7000_new_buffer is uploading
parts list: [{'PartNumber': 1, 'ETag': '"fe4b9305c800c1fac9a81a1e940c4c84"'}, {'PartNumber': 2, 'ETag': '"
sending to s3.............................................................
Logs/dir1/file_8000_new_buffer is uploading
parts list: [{'PartNumber': 1, 'ETag': '"fe4b9305c800c1fac9a81a1e940c4c84"'}, {'PartNumber': 2, 'ETag': '"
sending to s3.............................................................
Logs/dir1/file_9000_new_buffer is uploading
parts list: [{'PartNumber': 1, 'ETag': '"fe4b9305c800c1fac9a81a1e940c4c84"'}, {'PartNumber': 2, 'ETag': '"
sending to s3.............................................................
Logs/dir1/file_10000_new_buffer is uploading
parts list: [{'PartNumber': 1, 'ETag': '"fe4b9305c800c1fac9a81a1e940c4c84"'}, {'PartNumber': 2, 'ETag': '"
{u'ETag': '"d9494fe6e3cedb00e98f432e67d03ef7-10"', u'Bucket': 'alex-s3-mv-dest-seoul', u'Location': 'https


 metadata info: {u'AcceptRanges': 'bytes', u'ContentType': 'binary/octet-stream', 'ResponseMetadata': {'HT
```

```
  tar file: snowball-batch-filelist_dir1_10000.txt-20191230_014519.tar

real    0m30.375s
user    0m8.512s
sys     0m4.406s
```

This job took only 30 seconds dealing with 10,000 files. When comparing with initial test; it took 1m36sec for 100 files, you will notice its performance improved very well.

Limitation

- tar file should not be over 100GB to extract automatically.
  - ref: https://docs.aws.amazon.com/snowball/latest/developer-guide/batching-small-files.html
- multi-part upload can't be over 10,000 parts
  - ref: s3 multipart upload limits: https://docs.aws.amazon.com/AmazonS3/latest/dev/qfacts.html

Conclusion

Finally I completed this script within 1 month effort. It may is too simple and easy to an expert, but not to me. I think migrating 100 million files can be happen to somebody, so I am sharing my experience and the result source code. I hope it can help you when you confront same situation.