MapReduce

ME 766: Project

by

Anmol 140010018
Krishna Kumar 140010056
Gopal 140010057

Overview

MapReduce is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster. Conceptually similar approaches have been very well known since 1995 with the MPI standard having reduce and scatter operations.

In this project we implement the Mapreduce algo in such a form that it can perform mapping and reducing functions on a large file with help of 2 or PC's.

Features:-

Python impementation of Mapreduce
Parallel support with MPI for Python and C++

Problem Statement:-

There is a large file is distributed into smaller chunks(of about 500MB) and stored on cloud. We have to perform the map and reduce function on each chunk and stored back on the cloud.

Approach:-

- First the own cloud API is used to find out the number of chunks
- Than the files is diveded into number of hosts available and each host downloads its chunks.(This data is transfered via a ssh connection)
- Once the chunks have been downloaded than MPI starts on each hosts to map and later reduce in parallel.
- The final result is sent to the master via sftp and the result is combined.

Dependencies

For python:-
- Numpy
- Pyexpect
- mpi4py
- git
- owncloud
- json

For C++:-
- mpi

Running on Linux

All the hosts must have the same version of the libraries mentioned above. Also the version of python to be used is 2.7

For this one has to provide json files that will take the input.

Some json files are :- (Sample)
“src/helping_functions/own_cloud.json”

```
{
    "username"  : "Cloud_login",
    "password"  : "Cloud_password",
    "file_name" : "map_data_",
    "num_list"  : [1, 2]
}
```

“src/main/mpi_init.json”

```
{
    "file_num"          : [2],
    "file_name"         : "file_",
    "host_list"         : "192.168.1.10, 192.168.1.11",
    "usernames"         : "anmol1696, anmol1696",
    "passwords"         : "*****, *****",
    "file_num_host"     : [1 ,3]
}
```

Command is:-

```
$ sudo mpirun -np 4 python -m src.main.main_mpi
```

Modules Used

1. Own Cloud API:-
   Using the owncloud api we were able to store and install the large data files. The varuna cloud space was provided by aerospace department. Here we will were able to store the largefiles and also the results of the same
2. Freeing cache:-
   The buffer RAM is constanly being cleared. We clear the RAM which is filled by opening the files.
   For this we have used.
   ```
   $ sudo sh -c "sync; echo 1 > /proc/sys/vm/drop_caches"
   ```
   This helped the ram to not overflow.
3. MapReduce:-
   There are seperate mapping and reducing function, which are independent of wether we call them in serial or parallel. They are genral repesentation of mapping and reducing function.
   For this Project we have only implemented the word count, mapping and reducing funtions.

   P.S:- For further improving on this project, we can simpily add/replace the genral mapping and reducing fucntion
4. SSH:-
   Since the inbuilt connections of 'mpirun' over multiple hosts require the hosts and clients to have the exact same harddisk locations. Also, calling hostfile from mpirun had some issuses due to difference in the OS of the system that we used.
   So we wrote our own version of ssh for communicating between the systems.
5. C++:-
   Due to performance lag of python with mpi4py, we wrote a script in c++ inorder to perform the mapping and the reducing functions. We wanted to provide the mapped numpy array from python to c++, where it could perform the reduce function in parallel more effectively.

   P.S:- Due to complications with cython for running mpi, we were not able to connect the python code with c++. Due to this we faced a lot of problem with C++ itself.
   For now we provide the C++ code with a file to perform the map reduce independent of python. Due to this we were not able to extract maximum performance which we had earlier thought of.
6. Github:-
   For version control we used git and github.
   https://github.com/Anmol1696/MapReduce

Algorithm

The algorithm used is of simple map and reduce for word count. Here we have to calculate the total number of time each word occured in the file.

Mapping:-
Here we go through the file and store all the words in a numpy array. On this stage the words from the files is directly stored in the array without any modiications. This is done in serial, due to presence of only single read/write buffer

Reducing:-
For this we use the python counter library inorder to calculate the wordcount. The mapped array is divided into as many number of processors. Than for each part we perform the wrod count. This is than sent back to the master node where the results from each node is combined into one large output. This part is done in serial.

The output is of the form :-

```
[['fawn' '60']
 ['biennials' '47']
 ['jinrikishas' '64']
 ...,
 ['vane' '55']
 ['expands' '48']
 ['jawbone' '55']]
```

Where first string is the word and the second is its word count.

PS: This was a result for a randomly genrated file.

Benchmarking

The following is the result of the mapreduce being run on a single host.(Here the file size is the chunk size, which is the size of the smallest partition of a very large file)

File Size	Python Serial	Python Parallel	C++ Serial	C++ Parallel
52MB	15.8s	9.2s	7.1s	6.5s
496MB	724s	532s	35.4s	12.3s
1.2GB	1789s	1521s	65s	17.5s

The C++ was initailly slow, due to cost for calling the mpi calls. But for a large size the c++ was much fater than than python as expected.(Since C++ is compiled and python is an interpreted language)

Drawbacks

Our main moto was to make a programme that can be used on a bunch on laptops, a process which was to be performed on the servers. For this we wanted to connect the laptops and have all the extra stuff in python. The main algo part, we first wrote in python using mpi4py. On a very later stage we realised that the performance gain were not very high. This was due to the fact that python uses GIL lock. This did not allow the mpirun to access all the processors in parallel simultaneously.

For this we thought that we would write the parallel part in c++ and connect to the rest of the programe with cython. In this we failed and were not able to connect due to difference in complieres and calling methods