%D

CS322:Big Data

Final Class Project Report

Project (FPL Analytics / YACS coding): YACS Coding **Date:** 30-11-2020

SNo	Name	SRN	Class/Section
1	Saksham Gupta	PES1201800140	5G
2	B.N. Om Shreenidhi	PES1201800176	5G
3	Sahith Kurapati	PES1201800032	5A
4	Vishruth Reddy	PES1201800102	5D

Introduction

Big data is a field that deals with analyzing and extracting information from large and complex data-sets that cannot be done using traditional methods. An integral part of the architecture is its file distribution system which efficiently manages the organization of the files across all the nodes.

In this project we have successfully implemented such a centralized scheduler. Our system accepts requests for a job, distributes it amongst the worker nodes, simulates the tasks and receives the completed tasks from each worker. We have designed the project in a flexible manner, to be compatible with any system.

Our aim in this project was to get a better understanding of Big Data architectures like YARN through the task of building one.

Related work

Some of the resources referenced in making this project are:

[1] https://realpython.com/python-sockets/ - For socket programming

[2] https://docs.cloudera.com/HDPDocuments/HDP3/HDP-3.1.5/data-operating-system/content/apache_yarn.html - To get a better understanding of the working of YARN

[3] https://medium.com/python-features/using-locks-to-prevent-data-races-in-threads-in-python-b03dfcfbadd6 - To learn how to overcome race conditions using locks and semaphores.

Design

The design has been made as user friendly as possible to work on any system. The user can specify any number of requests as well as the scheduling algorithm of their choice(RANDOM, LL, RR). There can be any number of workers with however many slots in each worker specified through a configuration file and passed to the master.

The design is divided into two parts:

[1] Master

[2] Worker

Master

The master is a single program that runs three separate threads. The first thread listens for incoming requests(jobs), the second thread distributes the tasks amongst the available workers using one out of three available scheduling algorithms. The third thread listens for completed tasks from workers.

a) **Thread One**: The first thread is constantly listening for job requests. It inserts each job into a pool. The pool contains all the jobs that are currently being executed and those which are still waiting to be executed. It uses semaphores to prevent any race condition that could occur by inserting a new job into the pool.

b) **Thread Two**: This thread constantly checks the pool for any pending job. It distributes the tasks of each job amongst the workers by one of three algorithms. It uses semaphores to prevent race conditions which might occur after assigning the tasks to any worker.

    1. RANDOM: In this scheduling algorithm each task of each job is randomly assigned a worker from the list of available workers. If no worker is available, it releases the semaphore to enable the third thread to listen for completed tasks thereby freeing up a slot in a worker and then reacquires the lock immediately after.
    2. LL (Least Loaded):In this scheduling algorithm each task of each job is assigned to the worker which has the most number of free slots. Similar to the previous case, if no free worker exists it temporarily lets go of the semaphore so that thread three can free up a worker and then assigns the task to that worker by reacquiring the semaphore immediately after.
    3. RR(Round Robin): In this algorithm each task of each job is assigned to the next available worker. If that worker is not available then it moves to the next worker and so on. If no worker is free then it deals with this race condition in the same manner as mentioned in the above two algorithms

c) **Thread Three**: This thread is constantly listening to incoming completed tasks from the worker. Once it receives this completed task, It frees up a slot in that particular worker and deletes the completed task from its respective job in the pool. It uses semaphores to prevent race conditions which might occur when freeing up a slot and deleting a task from the pool.

There are two variables in the master that are shared amongst the three threads. They are the list of free workers and the pool of jobs. We used semaphores to deal with the race conditions arising from modifying these two variables.

Worker

The worker is a single program of which many instances are run to simulate different workers. In this case there are three workers (any number of workers can be run). Each worker has a predefined number of slots as given in the configuration json file. However this information is not available to the worker and is managed by the master. Each worker has two threads. They are:

a) **Thread One**: This thread constantly listens for incoming tasks from the master. It adds each new task into an execution pool. It uses python's in built threading lock function to deal with race conditions that might occur when inserting a new task into the pool. While inserting it acquires the lock and after inserting sets it free.
b) **Thread Two**: This thread checks if there are tasks in the execution pool. If there are no tasks in the pool it temporarily sets the lock free so that thread one can insert a new task into the execution pool and then reacquires the lock. If tasks exist in the execution pool then it reduces the numerical value of the duration for each task by 1 in every iteration. If any task reaches 0 then it is sent back to the master as completed. After reducing the duration for every task by 1 it simulates a time interval of 1 second by using the sleep() function. In doing so it correctly mimics the behavior of a task being executed for a specific amount of time.

Results

After building the project we ran several tests on the three different algorithms with varying number of requests. Through our tests we found that The RANDOM scheduling method was not very time effective for a large number of requests. While LL and RR gave similar results for a large number of requests. For fewer requests the time varies a lot and no clear conclusion can be reached.

From this it was clear to understand that proper scheduling methods are more effective than random ones. We did not have any hyper parameters to tune as a result most tests ended with the similar results.

Problems

Initially the program used just two threads one for listening for incoming requests and assigning the tasks to the workers and another thread for listening to completed tasks sent by the workers. This turned out to be a problem because the program could not simultaneously assign tasks to workers and listen for incoming requests. Further it would

finish each job completely and only then start the next job. This problem was tackled using three threads instead of just two.

In the worker the main problem lay in simulating the delay of executing the task for a certain duration of time. Furthermore, each task in worker had to be executed simultaneously. This was dealt with by introducing the sleep() function for a duration of 1 second for all the tasks at the same time. By doing this we were able to simulate the execution correctly.

Conclusion

This project was exciting all the way to the end. It introduced us to a lot of new concepts and more over how to actually code these new concepts. The most interesting part was trying to come up with solutions to handle the race conditions between the threads.

It gave us an insight into the working of YARN and how several Big Data architectures manage to organize their file systems to effectively analyze large and complex amounts of data.

With respect to the algorithms the choice of a random scheduler is less optimal when it comes to a large number of requests while Round Robin and Least Loaded take almost same amounts of time, though still lesser than random.

Working on this project has been a wonderful learning experience.

EVALUATIONS:

SNo	Name	SRN	Contribution (Individual)
1	Sahith Kurapati	PES1201800032	Worked on the entire Worker program, Logging information and the bash script to run all files simultaneously
2	Saksham Gupta	PES1201800140	Performed analysis of mean, median and mode using the log information and plotted graphs for the same

3	B.N. Om Shreenidhi	PES1201800176	Worked on scheduling the tasks to be sent to the workers using the three algorithms and updating the unfinished jobs in the pool after completion of tasks.
4	Vishruth Reddy	PES1201800102	Set up the socket connections for communication between all three files, Request, Master and Worker

(Leave this for the faculty)

Date	Evaluator	Comments	Score

CHECKLIST:

SNo	Item	Status
1.	Source code documented	
2.	Source code uploaded to GitHub – (access link for the same, to be added in status →)	
3.	Instructions for building and running the code. Your code must be usable out of the box.	