Distributed Compute for Text Recognition

Pranav Palani Acharya
Dept. of Computer Science
Indiana University
Bloomington, USA
pachary@iu.edu

Gavin Henry Lewis
Dept. of Computer Science
Indiana University
Bloomington, USA
gavlewis@iu.edu

Gaurav Narasimha Atavale
Dept. Of Data Science
Indiana University
Bloomington, USA
gatavale@iu.edu

Ashish Patidar
Dept. Of Data Science
Indiana University
Bloomington, USA
apatida@iu.edu

***Abstract*—It is well established that the advancements in the field of ML and AI is directly dependent on the available compute power. Ambitious projects involving Terra Bytes of data and complicated mathematical operations require huge computing resources. This becomes harder, especially when one resource provider/server is required to cater to the demands of such projects. Cloud provides a solution to these problems, by presenting computing resources through the use of virtualization technologies. This enables users to lease the resources from the service provider and run the tasks in a virtual environment. The growth and explosion of the cloud based applications and solutions can be attributed to the ease in access to such resources and often with little or no performance degradation. This can often lead to sub-optimal resource allocation and non optimal resource utilization. In this project, we intend to explore an approach where we try to pool computational resources present locally without compromising the performance of the task and utilizing resources optimally in a distributed manner. Our report implements the application of distribute compute using client server model where server divides the task among the clients and then aggregates the results that is obtained by the clients. The entire stack for the implementation is implemented from scratch with python and no other mainstream framework is used. A comparative analysis with other framework is also employed.**

***Index Terms*—distributed compute, machine learning, computer vision, task assignment, splitting, result aggregation**

I. Introduction

Distributed computing architectures have taken the main stage for the success of modern machine learning and deep learning algorithms. These architectures achieve incremental computational and storage capability in turn enabling scalability. A critical challenge in realizing this promise of scalability is the availability of adequate individual computing sources and efficient methods for communicating and coordinating information between distributed machines. Large Cloud service providers would not have issues with computing sources but they suffer with issues we discussed in the last section. In order to solve these issues, we are exploring a method where we have a different approach to procuring computing resources and combine it with the power of distributed computing to achieve the desired tasks. Given that any advanced Machine Learning/Deep Learning project can be divided into smaller chunks, these tasks make it an ideal candidate to test the distributed computing designs on the cloud.

Nowadays, we as individual users process machines which have good compute power. Most of the time we do not use these resources, but we would use them when we run a heavier task. Here, with this project we are exploring an approach where we collate multiple such local machines and make use of their unused commute power to achieve a resource intensive task for a fixed duration. This is achieved by hosting software/programs on the cloud which would access the availability of individual resources, allocate/schedule tasks and later aggregate results and send them to the user. This feature can also be used as a tool to enhance cloud resources with ideal local machines.

II. Related Work and Gap Analysis

A similar implementation of our idea is the MLlib library developed on the spark framework. Using spark we can execute machine learning in a distributed manner, but it equally distributes the task to each node. In this project, we are trying to distribute machine learning tasks on the basis of available resources utilizing every bit of performance of clients. We can compare our outcomes with the outcomes of MLlib on different parameters like memory usage, resource utilization, and time taken for execution. [2]

The system is also implements tasks similar to that of Hadoop where the server distributes the tasks among the clients and then aggregates the result back to the server based on the provided map-reduce function. [1]

III. System Architecture

The system that we have designed has the following architecture. The clients that are depicted may be on the cloud or they may be the ones that are present on the local machines as these clients provide the resources back to the server over the internet. The publicly available clients can be thus used to pool resources for distributed compute back to the client host.

The detailed system architecture can be seen in the image 1

IV. Server Services Designed

The following are the server services / features that are designed.

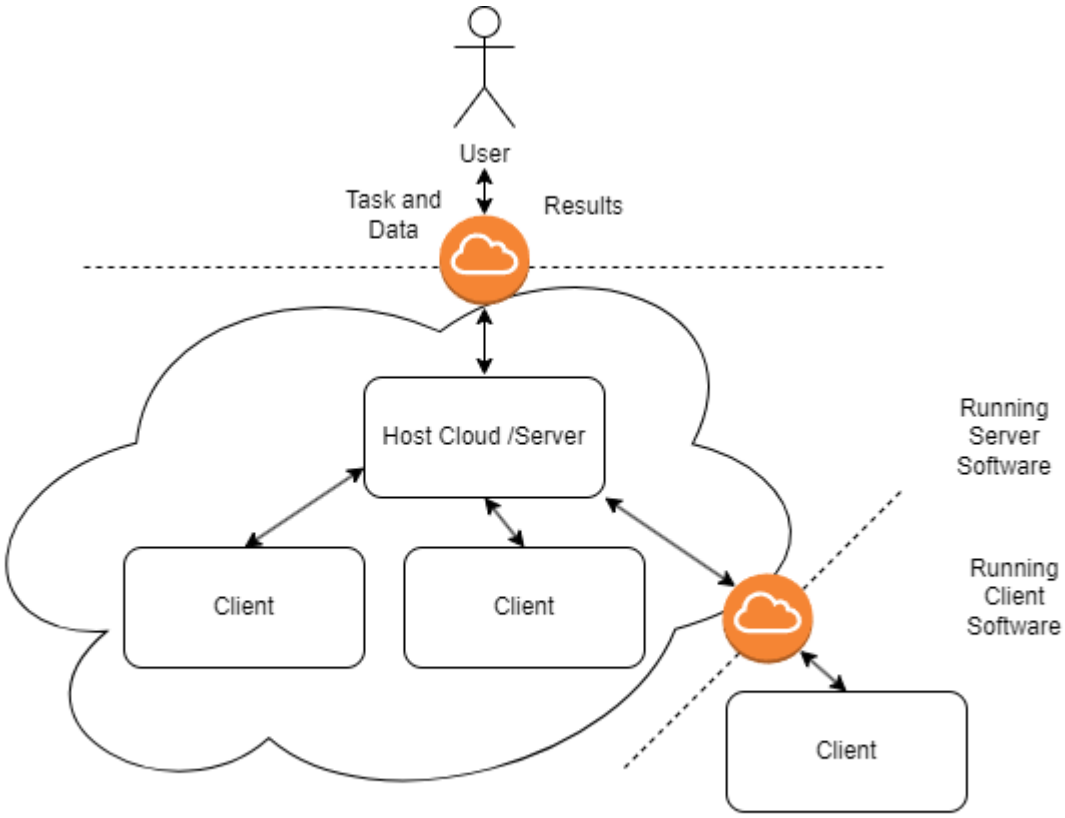

Fig. 1. Architecture of the system designed

A. Ping Clients

The ping clients service that is implemented pings all clients in the system. the input is a list of all clients that are configured to run. This service receives the client status if it is live or not. If it is live it receives the HTTP status code 200 and the assigns the value 1 indicating the corresponding client is active. This also obtains the memory that is available for each of the client. [5]

This service is run at an interval of 5 seconds using the python BackgroundScheduler method.

B. Send Jobs to Clients

The send jobs to clients service takes in the job. Then it splits the job based on the splitter function that is provided to it. After the task is split, the individual tasks are sequentially assigned to the clients based on their availability from the previous ping response. The file and the program that the client is supposed to execute is sent to the client using TCP protocol.

The jobs are sent to client as a bundle of the code and the respective data that is required to process them. For the example of word count, the jobs are sent in the form of splits of a file, and in the case of the example of text extraction, the code with the model and a batch of image may be passed to the system. Custom task splitting modules have to be formed for the system rather than a general task split.

1) Task Split: The task split functionality provides with the flexibility of assignment of the tasks dynamically rather than that of a system with a fixed approach of split. The task split module can be custom fed to the system based on the type of task that need to be sent: text or a batch of images.

2) Task Assignment: The task is assigned to the clients based on the tasks splits that are performed. The current implementation assigns the tasks sequentially to the clients that are predefined to the server. [4]

C. Receive and Aggregation

This service receives the outputs that are computed by the clients over TCP and then aggregates the results back for presentation. The main role of this module is to efficiently hide the complexity of the underlying implementation of the distributed compute and provide the abstracted result back to the client.

1) Receiving the Results: The results are received by the server using the HTTP protocol. The server validates if it receives a valid response. It return a HTTP 404 error if it does not receive any valid file response from the client .

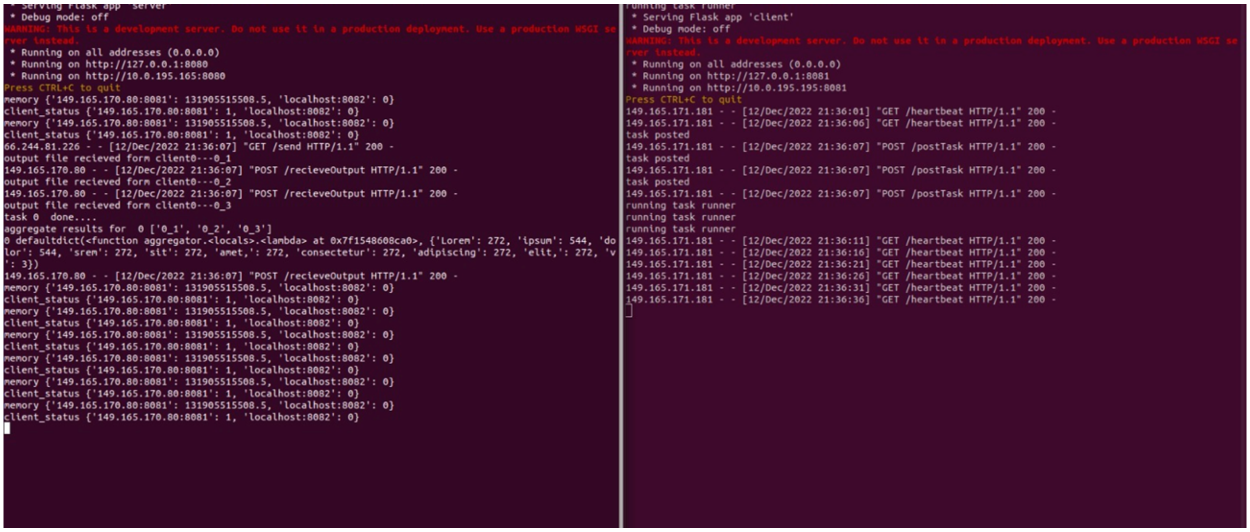

Fig. 2. Client server results of distributed word count on Jetstream2

2) Aggregation of Results: The results are aggregated using the aggregation function that is provided as an input to the server. This function will aggregate the raw results that is received by the clients and then bundle them for presentation back to the user.

V. Client Services Designed

The following are the client services designed. The functions for the task execution and result return back to the server are run on separate threads.

A. Heartbeat service

This service responds to the server ping message with the memory available in the client system for computation. This memory is obtained from the python 'psutil' module.

B. Posting of Tasks

This TaskPost service accepts accepts jobs from the server and assigns a local store of memory for the files. It also takes in the files and assigns corresponding job ids to it so that it may execute the tasks smoothly. The tasks are identified using the identifiers of program ID and job ID which are allocated by the task splitter in the server module.

C. Running Of Tasks

The task runner runs the tasks that are posted in the client machine. This is the phase where the actual execution of the tasks are done. The results are stored as files on the client. This will enable logging as well as sending of corresponding files back to the server.

D. Return of Results

The results of the run are sent to the server using the HTTP POST request which posts the output file with the JSON of the response. In the case of Text extraction the the text that is extracted from the image is returned as a dictionary of the image name along with the actual text that is extrated.

VI. Validation for implemented systems / Testing

The following two tests were performed on the system to test the working of the implementation.

A. Word Count

Distributed word count was implemented using our system where a large file was split by the server and then sent to the client system. The word count program was run client system. The client machine assigns the process id to the task and returns the result of the word count of the section of the file that was assigned to the client. The individual results of the word count is then aggregated by the server. Thus, the word count was successfully implemented in the client machine.

B. Text Extraction

Implemented large scale text extraction using distributed system using Python-tesseract. [3] In this implementation we provide the system with the folder containing the images, splitting code, the code for extraction (code executing on the server) and the aggregation code. The splitting function in our case, splits the prediction images into batches of images for each client to run. The client once assigned extracts text and returns back to the server.

The aggregation of the results are performed by the server, which returns a results in the form of a dictionary which contains the index as the image name and the value as the extracted text from the image.

VII. Deployment

Our implementation was successfully implemented and tested for Linux based systems as the code stores files which utilizes the Linux architecture. The implementation of the architecture

The deployment was done using machines on Jetstream 2 which was provided to us for this course. We setup a server where the server section of the code was running and multiple

clients where the client code was running. We also logged the client public IP addresses to the server so that the server will know which devices to look for to divide the code and execution.

VIII. Code and Development

To implement the system we used Python and other supporting python3 packages. The version control done using Git.

Repository Link:
https://github.com/pranavacharya/distributedCompute

IX. Future Scope

- Training of machine learning models on different clients
- Development for programming languages other than python
- Execution of more complicated problems like NLP, Deep Learning, and Reinforcement Learning, etc.
- More efficient way for distribution

X. Results

	Our solution	Spark distributed system
Performance (minutes) when 1 clients	8	6.3
Performance (minutes) when 3 clients	3.5	1
Performance (minutes) when 5 clients	2.5	0.5
Installation Overhead	10 KB	600 MB
Pre-requisites(to be installed)	Python3	Python3, Spark

The tests were performed on a 16 Core, 64 GB RAM Linux servers on same data set

XI. Conclusion

We have successfully implemented client-server model of a distributed compute system hosted on the cloud. The system was successfully tested to implement a distributed approach for word count and text extraction. The results were examined and compared to the other common distributed machines indicating the approaches.

Acknowledgment

Professor Dr. Dingwen Tao helped us understand the project better and guided us with the project.

References

[1] Hadoop: Setting up a Single Node Cluster. https://hadoop.apache.org/ Accessed:2022-12-16.

[2] Machine Learning Library (MLlib) Guide https://spark.apache.org/docs/latest/ml-guide.html Accessed:2022-12-16.

[3] Python-tesseract https://pypi.org/project/pytesseract/ Accessed:2022-12-16.

[4] Apache Map Reduce https://www.ibm.com/topics/mapreduce Accessed:2022-12-16.

[5] HTTP response status codes 200 OK https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/200 Accessed:2022-12-16.