Engineering Cloud Computing (Fall 2022)

Distributed Compute

Project team

1. Pranav Acharya
2. Gavin Lewis
3. Ashish Patidar
4. Gaurav Atavale

Professor
Dr. Dingwen Tao

1. Project overview

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 TBs 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 suboptimal resource allocation and unoptimized 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.

2. Project 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.

3. Related work and gap analysis

In modern-day computing explosive growth in the size and complexity of data, and new computational methods and platforms demands the need for parallel and distributed computing especially for machine learning and deep learning tasks on BigData. Our project provides such a way to assign unutilised resources of various client machines for a specific task, there is already some research conducted on that front. One of the most impactful work in that field is development of MLlib: Spark's distributed open source machine learning library.

MapReduce allows parallel execution of functions for large data sets using key-value pairs and it handles the partitioning, scheduling, machine failures, and inter-machine communication at run-time. 100's of terabytes of data can be easily distributed among 1000s of machines for parallel computing using MapReduce. There are many programs or processes with acyclic data flow which cannot be expressed using key-value pair, spark is such a framework which retains scalability for tasks that cannot be executed using mapreduce. Spark is a fault-tolerant cluster computing system which generalized MapReduce for large-scale data processing which provides API's in Java, Python and other common programming languages.

4. Proposed tasks (with detailed methodology)

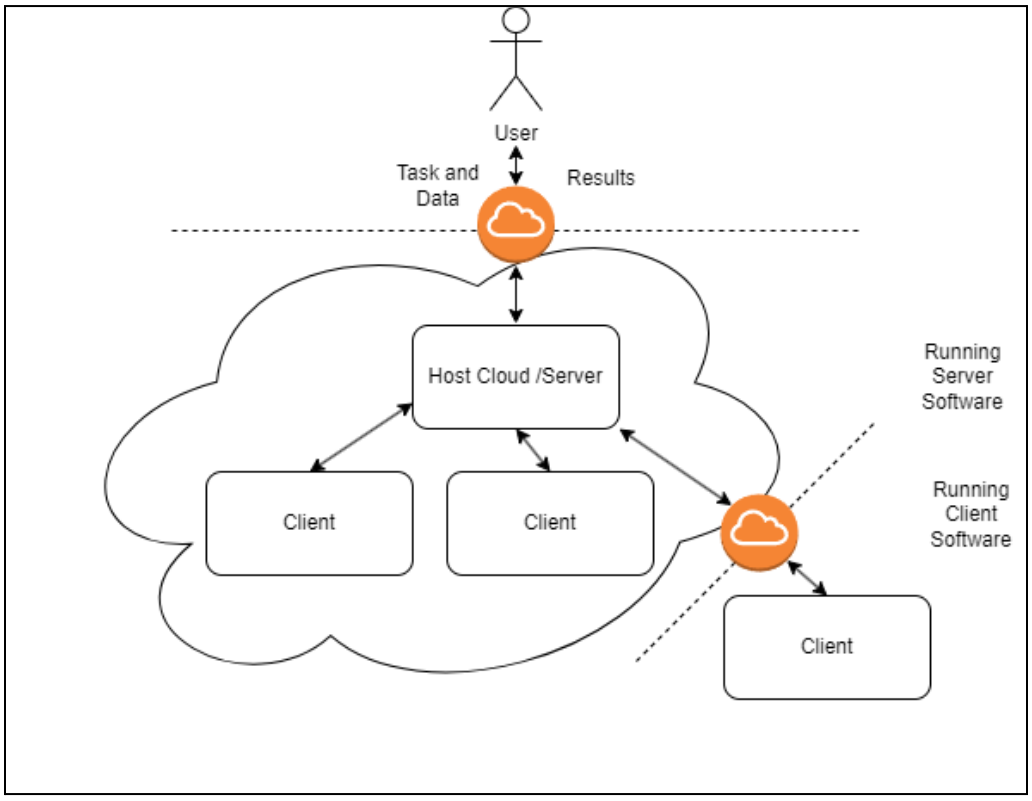
Our project targets implementation of two major softwares - one for the cloud server software which will accept job requests and distribute the tasks and the other one for

the client software which will accept the tasks from the server and will be responsible for the execution of the task assigned.

The cloud server software will accept jobs from the user and would be the only interface that will be visible and abstracted to the end user of the solution created. The software will be responsible for management of the client software, distribution of tasks among available clients, assignment of tasks and aggregation of results. We are currently exploring the feasibility of the variety of the tasks that are being distributed among the clients for optimal performance standards.

The client software which is responsible for the execution of tasks and will always provide a heartbeat message over the network to make sure that the host knows the resources that the individual client has available and that it is in a ready state to accept the tasks for execution. Once the task is assigned to the client by the host then it will execute the task and send the results back to the server. This client which is responsible for the execution may be present on the cloud or in a local environment providing the resources to the host software.

5. Architecture



6. Features

A. Host
   a. Ping clients / Monitors - Host pings all the available clients and list's the available clients.

    b. Task creation - Host divides the given task into small chunks.
    c. Task Scheduling - Host decides which client to associate each task with
    d. Task Assignment - Host sends the function and the chunk file to client
    e. Result aggregation - Host collects the result from the clients and puts the final result together.
B. Client
    a. Client Heartbeat - Client exposes an API for the host server to send ping and perform health check
    b. Receive Job from Host - Client receives a job file and a parallel program from host server.
    c. Task Status - Clients upon request send the statuses of jobs back to host.
    d. Run Jobs - Client runs the program on the file received and dumps the output to a folder.
    e. Send output - Client returns the compressed output folder back to the host.

7. Team members and workload allocation

1. Pranav - Client API development, Client resource monitoring, Cloud deployment
2. Gavin - Host API development, Health check and load balancing, Cloud deployment
3. Ashish - Parallel task development, Job division functionality, Cloud deployment
4. Gaurav - Host API development, Task scheduling functionality, Result aggregation feature

8. Planned timeline

Phase 1 - 28th October, 2022 - Client Server communication in distributed environment

Phase 2 - 9th November, 2022 - Task division, scheduling and allocation

Phase 3 - 30th November, 2022 - Result aggregation, cloud deployment

9. References

1. MapReduce: Simplified Data Processing on Large Clusters, *Jeffrey Dean and Sanjay Ghemawat*

2. Spark: Cluster Computing with Working Sets Matei *Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica University of California, Berkeley*

3. The Hadoop Distributed File System, *K. Shvachko, H. Kuang, S. Radia and R. Chansler*

4. MLlib: Machine Learning in Apache Spark, *Xiangrui Meng†*