International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 06 Issue: 04 | Apr 2019 www.irjet.net p-ISSN: 2395-0072

ALPYNE - A grid computing framework

Mohit Udupa[1], Aditya K Pramod[2], Manthan S S[3]

[1,2,3]*Eighth Semester, Dept. Of CSE, The National Institute of Engineering, Mysuru*

Abstract - *Alpyne is a grid computing framework that helps to set up a system of loosely connected computers that work together and can be viewed as a single system. The users can upload custom code or programs to make exhaustive use of grid computing framework. Alpyne includes restful API(Application Programming Interface) service, python libraries and applications for grid computing, built for easy installation, setup of all the required nodes and make use of the interconnect network. It makes running programs on several commodity hardware nodes very easy. Alpyne also supports services like high availability and load balancing. Load balancing is done based on all available nodes, choosing the suitable nodes and distributing the tasks based on the computing power of the available nodes. Our service also provides support for node failure management and efficient dynamic task scheduling. The interface allows users to visualize the resources used by all different nodes and jobs made on that compute nodes. All computing intensive jobs can be processed faster than on a single stand-alone machine. The framework is based on docker containers for virtualized environments for users and a file system interface on top of mongoDB(Database) for data storage. The grid makes use of interfaces modules for the above mentioned methods which can be easily switched to other user developed interfaces making it easy to use other methods of virtualization and data storage.*

Key Words: **Grid computing framework, high availability, load balancing, dynamic task scheduling, failure management, docker environment, django REST API.**

1. INTRODUCTION

With the popularization of cloud computing and distributed processing, the structure and setup time for the grid has become more and more complex and tedious task to do. Alpyne is a framework for grid computing built using python. It makes it possible to run several functions and programs on a system with several commodity hardware nodes.

Alpyne framework for computer grid helps us set up loosely connected computers that work together and they can be viewed as a single system. Unlike cluster computers, computer grids have more flexibility can be easily scaled, controlled and scheduled by software.

Grids are usually deployed to improve performance and availability over that of a single computer, while typically being much more cost-effective than single computers of comparable speed or availability.

Alpyne provides with all necessary endpoints for easy installation and setup of compute and data nodes. The service also provides features to enhance the functionality of the framework by through client side APIs for easy building of managers. The users can upload custom code and make exhaustive use of grid computing hardware. All computing intensive jobs like training, testing and running predictions on Machine learning models and Big data analysis can be processed faster than on a single stand-alone machine.

The goal is to get more computing power and better reliability by orchestrating a number of low-cost commercial off-the-shelf computers. High availability and Load balancing are given highest priority at all times by our service and features have been built for the same.

Other features include and web-based UI(User Interface) for monitoring the grid. This will allow system administrators to scale, reallocate, remove and assign computing hardware to the grid during runtime.

2. EXISTING SYSTEM

Hadoop, Apache Spark, Apache Storm, Flink and Samza are all open source and some of the most popular frameworks for grid computing. Although the above-mentioned frameworks support applications in python for grid computing, they are not primarily designed to work with it. There can be some overhead when dealing with applications that use some special modules, data types and operations from python[1].

All these above-mentioned systems require the data and code to be in specific formats. While big data isn't exclusively made for big businesses, not all big data platforms are suited for small data needs. Unfortunately, the Hadoop framework happens to be among them. Due to its high capacity design, the Hadoop Distributed File System or HDFS, lacks the ability to efficiently support the random reading of small files (lesser than 128MB, the block size of HDFS). As a result, it is not recommended for organizations with small quantities of data. Most of the existing systems use batch processing which makes it slower. The setup time of existing solutions take a long time and makes it inefficient for small tasks[2].

They also require a large set of dependencies to be installed, updated and maintained, which can sometimes be OS and/or hardware specific. This means setting up a

grid will take a long time and it can sometimes not be the best use of a developer's time especially when the requirement for the grid is for a short time.

3. OBJECTIVES

The objective is to build a framework of libraries and server applications that will seamlessly work with each other. System administrators will be able to easily install the grid system onto their computing hardware and clients will be able to interact with the above-mentioned grid and queue code and data for execution on the grid.

The libraries provided for the client will also include guidelines for the client to design their own grid applications for map, reduce and filter based problems. This can be used to extend the grid system to run applications on data analysis, machine learning, etc.

The compute nodes and data nodes will use modular interfaces to standard applications like dockers for computing and databases for data storage which can be easily swapped out for a different module to change the back-end of the nodes.

Separate data, compute and management planes allow the data nodes and compute nodes to be used independently. There will be no one manager to the system but every client will manage his own access to the grid. The overall access management to the grid will be distributed among the nodes in the grid.

Expose all nodes in the grid via REST(Representational State Transfer) or any other popular interface, making every element in the grid programmable and customizable without the need for any major code upgrade, this can be used to emulate any behaviour in the grid.

4. PROPOSED SYSTEM

The proposed system is a simple and easy to set up grid computing framework built completely in python. The system can be set up at a node with minimal difficulty since it requires only python installation for the required operating system. It uses django restful service for communication between clients, database and compute nodes.

It primarily uses TCP(Transmission Control Protocol) as the message passing intermediate system. The system will not rely on an always distributed file system solution like HDFS. Instead, it will only distribute data as and when needed by the client.

It will be compatible with all of pythons modules, data types and operations, including custom user-defined types. The client can interleave multiple compute nodes and container databases to program as they require to suit different codes.

The framework will provide blueprints and guidelines for Map, Filter and Reduce operations. Users can use these templates to implement their own code and push it to all the nodes using the user interface or the API service. The system will include all functionalities of current grid computing frameworks like Load balancing, Fault tolerance, Data redundancy, Co-location of code and data, high availability etc.

The system will use custom application layer protocols designed for low latency and interrupt based communication between nodes with support for encryption and compression. The /system will also support an easy to use web-based UI(User Interface) to manage and control the grid.

Our system uses a benchmark standard to test the capabilities of all the available commodity hardware nodes, takes decisions according to the results and divides the tasks accordingly to all these compute nodes.

5. SYSTEM ARCHITECTURE

The Alpyne network consists of three components namely the compute nodes, the database and the client code. The compute nodes and the database together form the Alpyne pool. The client code acts as manager to the Alpyne pool. The client defined code is also the code that runs on the Alpyne grid which makes use of the all the features that are provided in the grid. The following illustrates elements of the Alpyne grid in detail.

5.1 ALPYNE POOL

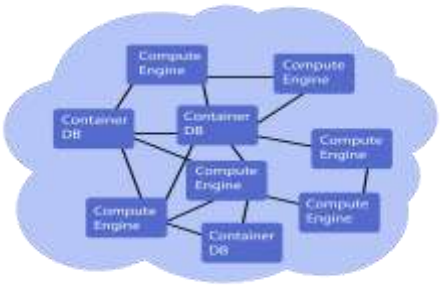

Fig -1: Alpyne Pool

The Alpyne pool consists of one or more elements such as the compute node or the container database. Every element in the Alpyne pool has soft links with each other. So the compute nodes and the container databases although connected to each other via the same shared network can operate independently and self standing.

The Alpyne pool can be scaled up or down without affecting the working of the pool as of the independence of the compute nodes and the container databases. The Client code acts as the manager to the entire Alpyne pool. The client can decide to have adequate and feasible number of compute nodes and databases to handle his grid computing tasks.

This pool can be located in the same datacenter / building or anywhere across the internet. Each of the pools elements will be given a score of performance. This score will be based on the hardware specifications, network connectivity and other performance metrics of the element. The manager uses this score to allocate jobs and data to these elements.

5.2 ELEMENTS OF THE ALPYNE POOL

1) Compute node :

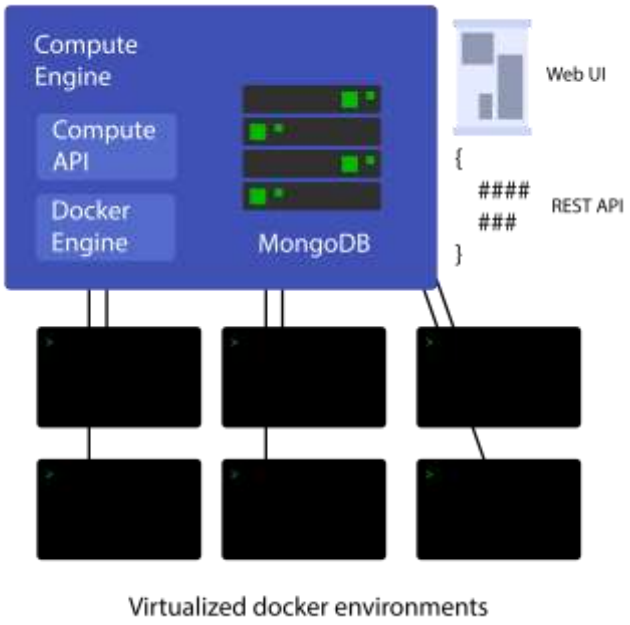

Fig -2: Compute engine

The compute node is a restful and a programmable computation environment used to execute client defined jobs. The restful service provides features to make Rest API calls to allocate jobs, delete jobs, get status and results of those client defined jobs. The compute node is programmable as it relies on docker environments.

Since it is based on docker environments all the resources like storage, processing and memory resources are virtualized, secure and metered as required by the admin. The compute node has its own database to keep track of all the jobs allocated by the client and the supporting files required by those client defined jobs.

A compute node can operate without the need of an external database or any specialized client library. On failure, the compute node restarts itself and picks up any client jobs that might have crashed due to the failure or closed unintentionally. This feature provides local recovery for all the compute nodes.

The compute node also comes with a simple web-UI. This UI provides a read-only access to the current resource usage of the compute node. This includes details about the running, queued and stopped jobs. The UI also provides a look at the overall load on the compute node. The reason for the UI to be read-only is to make it independent in the grid. By stripping the some access in the UI, the managers can be more effective with their control.

2) Container Database :

Fig -3: Container Database

The container databases uses a mongoDB in the backend. This Database has a container abstraction interface to organize files and folder like structures, which are used by the client jobs. These files can be used by compute nodes for running any user defined jobs. Multiple databases can be setup in mirror mode to allow for data duplication and redundancy. Every database is independent of each other and is unaware of the global shared storage.

The container database provides API calls for interaction between compute nodes and the mongoDB which its based on. Using these API calls files and folder like structures can be accessed by the compute nodes whenever client job requires it to access them. The access to the container database is authenticated and only authorized clients can access them.

5.3 ALPYNE MANAGER

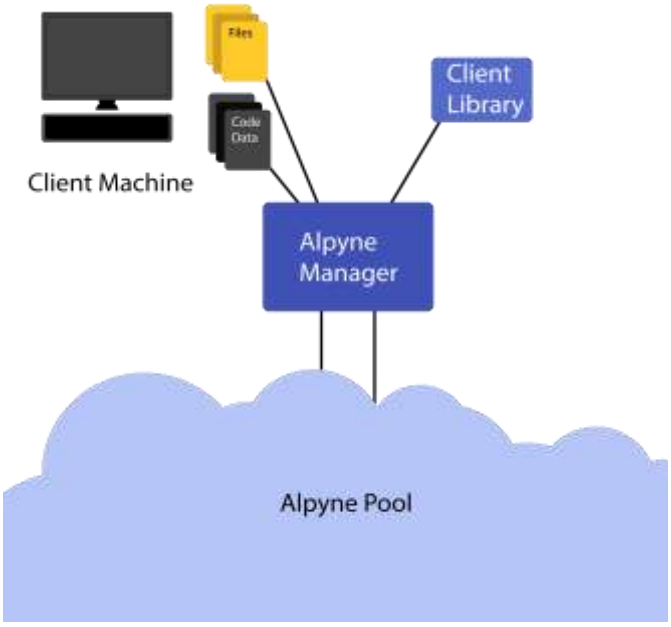

Fig -4: Alpyne manager

The Alpyne manager is the integral part of the Alpyne grid. It is the client side code that uses the restful compute nodes and the container databases to run client defined jobs on the Alpyne pool. Since the Alpyne manager code is maintained by the client the entire extent of the job will only be known only to the client keeping it hidden from rest of the Alpyne pool, hence ensuring security.

The client can use the restful nature of the compute nodes and the set of tools provided by the client side library to implement his own level of cluster management system. The client library provides API calls to push data into the container databases and pull data from the container databases. The client library also provides API calls to interact with the compute nodes and provides functions for parallel operations like map, filter and reduce operations. The manager can be a simple script to run a job on one compute node without support from any container databases or a complicated service, implementing code and data replication to further increase reliability and code distribution.

5.4 ALPYNE GRID

The Alpyne grid consists of one or more Alpyne managers and one or more Alpyne pool elements. An Alpyne manager can access the pool via login credentials which can be obtained by registering to the pool elements. This registration process can either be set open or administered. The following is the schematic Alpyne grid diagram.

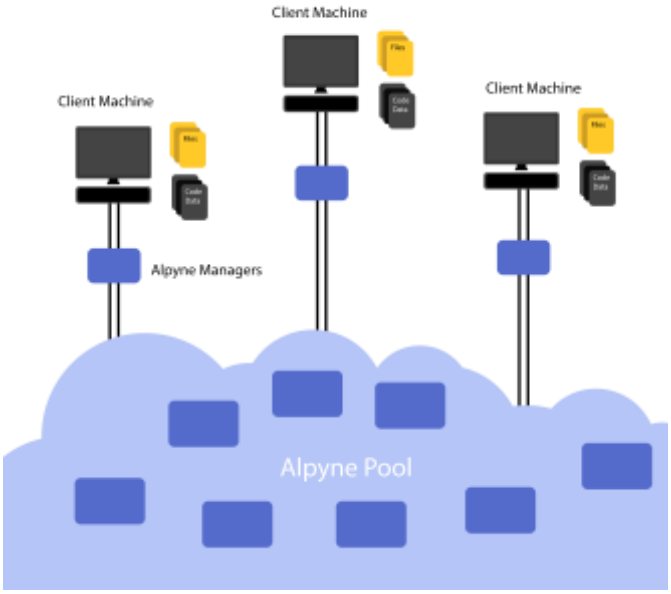

Fig - 5: Alpyne grid

6. SYSTEM WORKFLOW

In this section we explain the overall workflow of our Alpyne grid computing system, in addition to some of the prominent features the system provides.

6.1 JOB ALLOCATION

A client job begins with a set of tasks. The Alpyne manager uses the distribution API from the client library to distribute the tasks between multiple compute nodes in the Alpyne pool. This task distribution is based on how much load the compute node is experiencing and also the custom benchmark scores from those compute nodes.

Next the Alpyne manager uploads the data and required files into the container databases using the client library. The Alpyne manager chooses the required level of data duplication that is required for the client defined task. The client library also allows clients to upload live python objects to the container databases in suitable pickle formats.

Then the Alpyne manager allocates the distributed jobs to the compute nodes and waits for the results from those compute nodes. The Alpyne manager can also take care of failovers based on client task requirements. These allocated jobs can be of the type map, filter or reduce operations. Once the job execution is finished, the client can read the results back from the container databases.

6.2 FEATURES PROVIDED

If a compute node fails then the Alpyne manager can reallocate the incomplete jobs to a different compute node or it can wait for the same compute node to pick up any of the jobs that might have crashed due to the failure or closed unintentionally once it automatically restarts.

If one of the container databases fail then the redundant copies of the data can be used as backup by the compute engines and recovers from failure. Database recovery tactics have been researched and perfected for a long time. The mongoDB data nodes can be easily setup in mirror mode. They also provide rollback strategies like checkpoint recovery.

If the Alpyne manager fails then the Alpyne pool elements like the compute nodes and container databases will still continue to execute the task at hand. Once the Alpyne manager restarts it can fetch back the results from the container databases. This asynchronous behaviour is because the Alpyne pool does not require any constant monitoring from the Alpyne manager to run. This is how the system reduces the risk of single point failures and achieve good reliability with limited hardware. Using the above services high availability and failure recovery is achieved.

7. CONCLUSION AND FUTURE ENHANCEMENTS

The Alpyne grid computing system is a framework of libraries and server applications that will seamlessly work with each other. System administrators will be able to easily install the grid system onto their computing hardware and clients will be able to interact with the above-mentioned grid and queue code and data for execution on the grid. Hence the promised easy installation process is achieved with less hassle. Features like high availability, failure recovery, dynamic task rescheduling, node failure management can be easily utilised by clients based on their requirements. As compute nodes are based on docker environments they are highly modular and scalable. Due to this virtualization the client data is secure and is managed at his own resolution.

Although the final product obtained at the end of the development cycle met all the expectations that was set out for in the beginning, the framework can still be improved with some more development. One of such improvements would be the addition of service jobs. These jobs will run as a service which will never end. These can be used to deploy web based applications in Alpyne. Another improvement would be by adding a proper file system. Docker containers can be used for this process, providing more flexibility with how data is moved between the elements of the pool.

With such improvements along with other performance and security improvements, the Alpyne grid can be used for applications like, Big-Data, Machine learning, Web hosting, Programming sandbox and many more.

REFERENCES

[1] Improvements in Big Data Hadoop Several hybrid efficiency methods by Ye Zhou and Amir Esmailpour.

[2] Hadoop Configuration Tuning With Ensembling Modeling and Metaheuristic Optimization by Xingcheng Hua, Micheal C. Huang and Peng Liu.