2024

# Engineering for Data Analysis Project Report

MSC SCIENTIFIC AND DATA INTENSIVE COMPUTING

GUI, AOXUE

# Table of Contents

# Introduction

My project was to design a distributed data pipeline to help a biochemistry research team at University College London Medical School build a distributed protein structure prediction system. I used Ansible as a configuration management tool to optimise the data processing flow, and Prometheus and Grafana to provide visualisations for monitoring and logging. In the next section, the report delves into the technical details of the system, and its design principles, and discusses future performance optimisations.

# Technology Selection

## DevOps configuration management tools => Ansible

Chef, Puppet, Salt Stack and Ansible are mature DevOps configuration management tools on the market. They are all idempotent, which helps maintain a clean and tidy system environment. In addition, these tools provide numerous useful built-in modules that help simplify system configuration and reduce the possibility of errors.

However, learning Chef and Puppet is relatively time-consuming because the programming languages they use are not familiar to me, which may increase the difficulty of system construction. These two tools are more suitable for building complex environments, and my project is relatively simple, so Salt Stack and Ansible are more suitable choices for me.

Compared to Salt Stack, although they both support agentless mode, Ansible has wider community support and clearer documentation, which made it easier for me to find timely solutions to my problems. Therefore, I chose to use Ansible as my DevOps configuration management tool.

## Monitoring and visualization tools => Prometheus + Grafana

### Prometheus

The main reason I chose to use Prometheus is that it comes with timing data capabilities and only periodically grabs the state of the monitored component over the HTTP protocol, without the need for a special SDK or complex integration steps.

Its significant advantage is that if the component can provide an HTTP interface, it can be easily connected to the monitoring system and has minimal impact on the performance of the monitored system. In addition, by using the Node Exporter component of Prometheus, you can easily collect various system indicators such as CPU, memory, disk, and network to achieve comprehensive system-level monitoring. The significant advantage is that if the component provides an HTTP interface, it can be easily accessed by the monitoring system with minimal impact on the performance of the monitored system. In addition, by using Prometheus' Node Exporter component, it is very easy to collect a variety of system metrics, such as CPU, memory, disk, and network, to achieve comprehensive system-level monitoring, and the Node Exporter itself occupies few system resources to ensure stable operation.

### Grafana

The reason for choosing Grafana is because it supports integration with Prometheus and provides flexible dashboard configuration and diverse data display and analysis methods.

# System Design

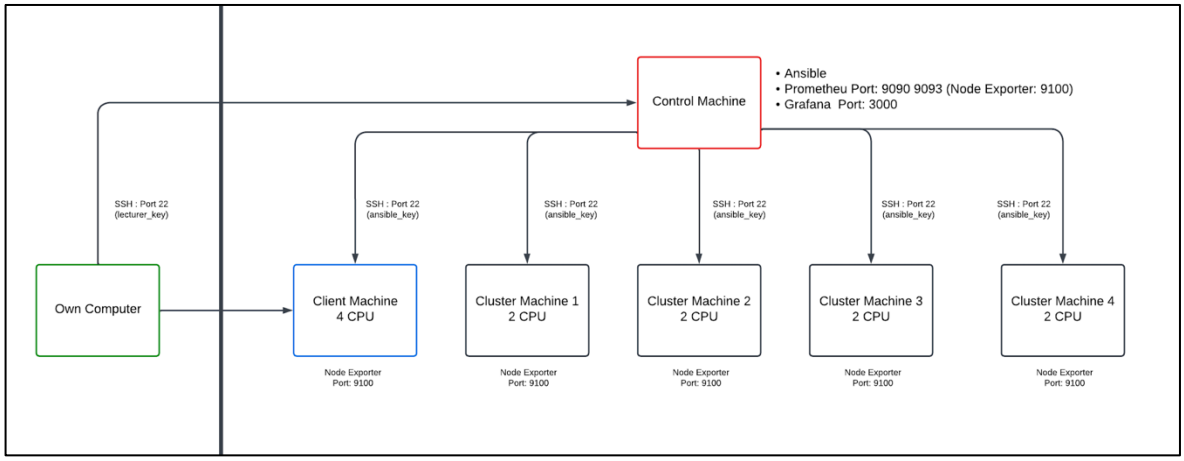


*Figure 1*

## Main components of the system

### Own Computer

On the “Own Computer”, I have set up SSH keys for secure remote access to the “Control Machine” and “Client Machine”. Additionally, for enhanced programming efficiency, I utilize Visual Studio Code as my editor.

### Client Machine

On the “Client Machine”, I first built a proper virtual environment and developed code in that environment. This was done mainly because the operating environment of “Own Computer” is different from the AWS machine. To avoid possible bugs due to operating system differences, I chose to do the development work directly on the “Client Machine”.

Once development is complete, I export the dependency packages from the virtual environment and generate the requirements.txt file, which simplifies the process of building the same virtual environment on a clustered machine.

### Control Machine

On the “Control Machine”, I have installed and configured Ansible along with its inventory. Moreover, to achieve visualized monitoring of the operational status of all machines in my system, I have also deployed Prometheus and Grafana.

#### *The main tasks of the Control Machine*

1. Configure Inventory to simplify the management of synchronization across multiple machines

2. Gather system information about "Client Machine" and "Cluster Machines" for subsequent configuration of the correct virtual environment and design of distributed data pipelines
3. Configure the firewall to open the required ports on each machine
4. Collect disk partition information and perform mounting
5. Build and configure the virtual environment
6. Deploy datasets and distribute relevant documentation for executing the data pipeline
7. Run tests to ensure that the entire system is operating as expected
8. Installing Prometheus and Grafana
9. Configure Prometheus' prometheus.yml and alert_rules.yaml files and deploy Node Exporter
10. Gather the results from the "Client Machine" and "Cluster Machines" and get the Final results

## Security Design

### Secure Access

On the “Control Machine”, I used the ssh-keygen command to generate a new pair of SSH keys, named ansible_key and ansible_key.pub. I then distributed ansible_key.pub to both the “Client Machine” and the “Cluster Machines”. To integrate ansible_key.pub into their respective authorized_keys files, I executed the command cat ansible_key.pub >> authorized_keys. This step is crucial to ensure that the “Control Machine” can securely establish SSH connections with the “Client Machine” and “Cluster Machines” using the ansible_key private key. Following this configuration, the “Cluster Machines” are now equipped with new authentication keys, rendering the lecturer_key unnecessary. This password-less access mode also improves the efficiency of automated scripts. Although lecturer_key can be removed from authorized_keys here, I decided to keep lecturer_key in the authorized_keys file for the time being in case the lecturer needs to access my machine in the future.

### Enable firewall and customize port configuration

I have enabled the firewall feature for each machine to secure the system and only open specific ports for the appropriate services (Figure 1 details which ports are open on each machine).

## Distributed Pipeline Design

In designing the distributed pipeline, I adopted a horizontal scaling strategy to enhance the processing power of the entire system. The main advantage of this strategy is that it effectively spreads the computational load and significantly improves the scalability of the system.

With no access to Virtual Private Cloud configuration information and no access to S3 storage buckets, I, therefore, turned to a strategy of mounting Elastic Block Store volumes on “Client Machine” and “Cluster Machines”. On these machines, I deployed the necessary virtual environments as well as the datasets for data analysis. Next, I split the experiment_ids.txt document into several data fragments and distributed them to the nodes. Each node will independently process one or more of its assigned data fragments based on its performance characteristics to effectively contribute to the efficient analysis of the data.

# Performance Discussion

## First Attempt

In my initial attempts to assign work to my system, I split the task file (experiment_ids.txt) into seven roughly equal-sized pieces. I assigned three pieces to the “Client Machine” and one piece to each of the four “Cluster Machines”. This decision was based on the information I gathered about the system, which revealed that the “Client Machine” has four logical processors, while each “Cluster Machine” has two. Therefore, I plan to use three of the logical processors on the “Client Machine” to process three chunks of data in parallel, maximizing its CPU processing power. The remaining logical processor is reserved to ensure the smooth operation of the operating system, efficient management of background services and processes, and stable execution of Node Exporter monitoring tasks.

The same logic is applied to “Cluster Machine”, i.e., assigning one logical processor to each machine to process tasks while leaving one logical processor to ensure that other tasks and the operating system can run properly. However, through the monitoring of the Grafana Dashboard, I found that when the Client Machine tries to process three pieces of data at the same time, it is not as efficient as expected. This inefficiency may be due to resource competition between multiple logical processors.

## Second Attempt

In the second attempt, I split the data evenly into six parts. The "Client Machine" runs two pieces of data, while each "Cluster Machine" still runs one piece of data. Compared to the first attempt, there was a slight improvement in efficiency.

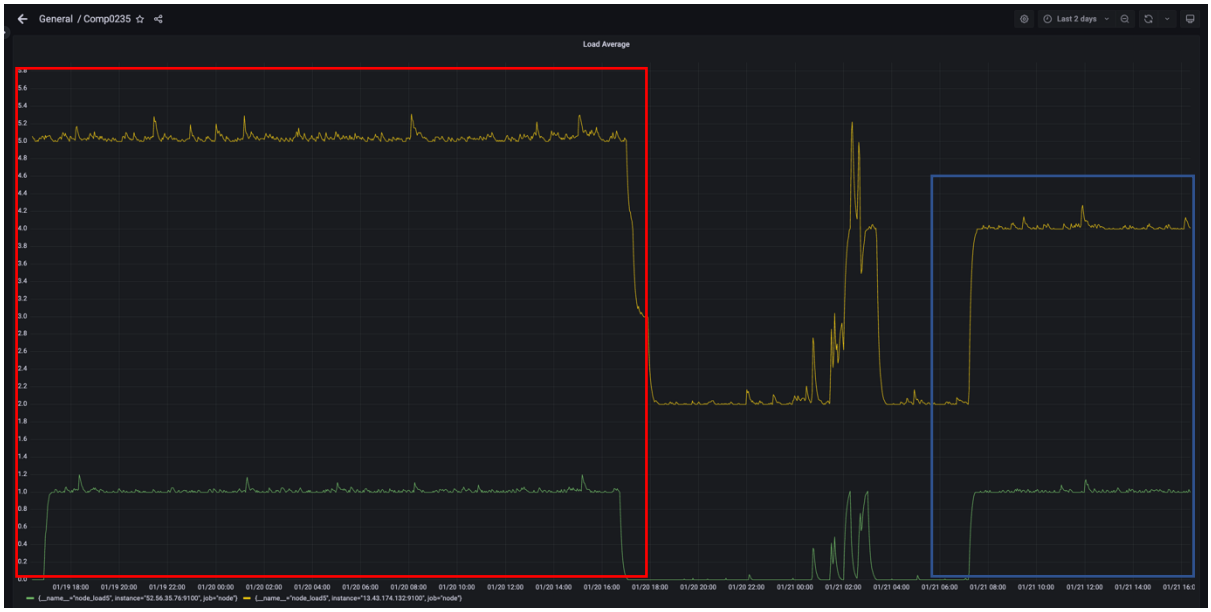


*Figure 2 - Load Average*

In Figure 2, the data in the red boxes represent the information related to the first attempt, while the data in the blue boxes correspond to the second attempt. The comparison shows

that the load fluctuations of the "Client Machine" appear smaller in the second attempt than in the first attempt. This might indicate that resource contention may have lessened.

## Future Attempt

Due to competing resources, my system currently fails to achieve optimal efficiency. For this reason, I plan to optimise my distributed data pipeline in the future using an architecture that combines Spark and Hadoop Distributed File System (HDFS). This architecture allows Spark to automatically split large files stored on HDFS into small chunks and process these chunks in parallel on different Spark worker nodes. To improve efficiency and avoid resource contention, I intend to allocate the appropriate number of cores and memory to each Spark worker node. At the same time, I will use Spark's dynamic resource allocation feature to dynamically adjust the number of executors based on the current workload. This not only helps to improve the computational efficiency of the whole system but also ensures the balanced use of cluster resources and system stability. In addition, I plan to store the processed data in a database to make it easier for subsequent researchers to access and analyse the data. This will support the rapid generation of different types of data files on demand and enable efficient data lookup.

## Fulfillment Of Required Features

1. Should use an appropriate configuration system. We have covered Ansible and Salt, but others are available.

   I used Ansible

2. Make use of an appropriate datastore for the complete human proteome contained in file uniprotkb_proteome_UP000005640_2023_10_04.fasta.gz. This should be able to return appropriate records from a list of arbitrary protein IDs.

   I've rewritten the select_ids.py script, which accepts an arbitrary list of protein IDs as input and retrieves the complete information corresponding to those IDs from the uniprotkb_proteome_UP000005640_2023_10_05. fasta file, ultimately returning a query with all the results in the fasta file.

3. Make use of appropriate monitoring and logging of your mini-cluster and your data analysis pipeline.


*Figure 3*

Table

| Time | __name__ | alertname | alertstate | instance | severity | device | fstype | job | mountpoint | Value |
|---|---|---|---|---|---|---|---|---|---|---|
| 2024-01-21 15:37:12.373 | ALERTS | HighCpuUsage | firing | 13.43.174.132:9100 | warning | | | | | 1 |
| 2024-01-21 15:37:12.373 | ALERTS | HighDiskUsage | firing | 13.43.174.132:9100 | warning | /dev/nvme0n1p4 | xfs | node | / | 1 |
| 2024-01-21 15:37:12.373 | ALERTS | HighDiskUsage | firing | 13.43.32.240:9100 | warning | /dev/nvme0n1p4 | xfs | node | / | 1 |
| 2024-01-21 15:37:12.373 | ALERTS | HighDiskUsage | firing | 13.43.50.108:9100 | warning | /dev/nvme0n1p4 | xfs | node | / | 1 |
| 2024-01-21 15:37:12.373 | ALERTS | HighDiskUsage | firing | 3.10.1.48:9100 | warning | /dev/nvme0n1p4 | xfs | node | / | 1 |
| 2024-01-21 15:37:12.373 | ALERTS | HighDiskUsage | firing | 52.56.35.76:9100 | warning | /dev/nvme0n1p4 | xfs | node | / | 1 |

*Figure 4*

4. Should collate the results calculated on the client machines and make them available to the researchers.

   The files get_all_experiment_fasta.yaml, process_data_get_final_results.yaml, and processing_fasta_data.py are utilized to gather and compute the results, with the final data being stored in profile_out.csv and hits_output.csv.