COMP0235 Coursework Challenge:

*Building a distributed analysis system for human proteome analysis*

Marat Davudov

Table of Contents

[1. Introduction 3](#_Toc155538112)

[2. Design and Implementation 3](#_Toc155538113)

[2.1 Installing Dependencies 3](#_Toc155538114)

[2.3 Dividing and Distributing Work 4](#_Toc155538115)

[2.3 Recording and Combining Results 5](#_Toc155538116)

[2.4 Security 5](#_Toc155538117)

[2.5 Monitoring 5](#_Toc155538118)

[3. Conclusion 6](#_Toc155538119)

# Introduction

In this coursework challenge we were asked to create a distributed analysis system for protein analysis. The task was to use a pre-written data analysis pipeline and distribute the task across 5 worker machines to make the processing times more feasible for a biochemistry research team. This report outlines the steps taken to fulfil the task and describes the encountered challenges. The code can be found on GitHub: https://github.com/hyperleash/cw0235.

# Design and Implementation

I have decided to use Ansible throughout the entire process, configuring the cluster, dividing the work between the nodes, and collecting back the result. This decision was motivated by several factors. Firstly, in contrast to other technologies (e.g. Salt) Ansible is agentless, which allows for simple deployment, as it does not require any additional software to be installed on them. Secondly, Ansible is self-documenting and is very easy to read and understand. This is beneficial, as the research team will be able to quickly understand the logic of the distributed system and potentially even modify it without having to spend a lot of time learning the tool. Finally, because the analysis pipeline is quite simple and does not require complex task distribution and orchestration utilising tools like Kubernetes and Docker. Although it could be made more efficient by incorporating these technologies, using Ansible alone and parallelising the task over multiple nodes already allows to significantly reduce processing times for the research team and at the same time keep the system understandable.

The following subsections will talk through the challenges encountered throughout the process of creating the distributed analysis system for the specified task and describes each step of the implementation in more detail.

## 2.1 Installing Dependencies

Most dependencies were not large and could be easily installed on the workers’ disks. The packages used for analysis are installed using built-in ansible modules and the scripts themselves are transferred directly from the control node. However, because hhsuite and s4pred libraries require a lot of disk space for the pdb70 database and model weights I had to decide on an appropriate way to store them.

The weights file for s4pred had to present locally on all machines, and a larger data volume was mounted on the workers to accommodate it. The pdb70 database for hhsuite could potentially be installed on an s3 bucket. I considered this solution because the pdb70 database is 78G and was hosted on a very slow network, so downloading it only once to the s3 bucket could be a more efficient solution in terms of memory resources and a better suit for frequently accessed data. However, because the hhsearch is I/O bound, having faster storage was more important and I decided to put pdb70 on the mounted volume.

To put pdb70 on each of the node’s local storage I first download it to only one of the nodes and transfer it from there to the others. This allows the system to rely less on the slow network where the archive is hosted. This way if a new node is added to the cluster, it will be ready to work much faster by getting the file from one of the nodes instead of waiting for a long download. I also learned later that smaller nodes in the cluster do not have enough disk space to hold the archive and fully unzip it at the same time. Therefore, the only way to distribute the files was to unzip the archive on one bigger client nodes and copy the directory to others.

## 2.3 Dividing and Distributing Work

After the workers were configured, I needed to find a way to evenly divide the work between them. Initially, the ids of the sequences to be processed are stored in a text file on the control node. To divide them I transfer the entire file to all the workers and then extract the appropriate number of lines for each node.

The worker nodes have different numbers of cpu threads available to them and parts of the pipeline can be further parallelised over them to reduce the processing time. Particularly, s4pred and hhsearch steps allow to specify the number of threads to use. So that the pipeline script can be parallelised over an arbitrary number of threads, I calculate the number of threads available during runtime using python multiprocessing module and pass that into s4pred and hhsearch commands.

Since the nodes have different number of cpu threads available to them, the task division process needs to take that into account. To do that, I first gather and set facts about the nodes and the task by recording the total number of ids to process, the number of vcpus on each node, and total number of threads on the cluster. Since the nodes can have different number of ids assigned to them, I couldn’t find a neat expression that would allow to fetch the correct ids locally without having information about which ids where assigned to other nodes. To solve that I split the division process in several steps.

First, on the control node I iterate over the workers and on each iteration calculate the amount of work based according to this formula:

In the master task file, each id is stored on a separate line and therefore the line number acts as an index. Throughout the loop I keep track of the current start and end index of the id range for the current node. The start index is initially one and the end index is calculated by adding to the start index the number of ids for calculated for this node. At the end of each iteration, the start index is reassigned to be the current end index plus 1. The ranges are stored in an array. Finally, in the following ansible task, each of the worker nodes can look up what range of ids were assigned to it and extract them from the master file stored locally.

This strategy is effective; however, it can still be further improved. For example, it does not consider the processing time variability depending on the sequence being processed and ignores the fact that some parts of the pipeline script don’t make use of multithread processing. A more efficient solution would make use of dynamic load balancing. Kubernetes would be much better suited for this task and therefore this would be an argument against using Ansible for this part of the system in this case.

## 2.3 Recording and Combining Results

The last step of the analysis pipeline parses a.hhr file produced by hhsearch and stores the id of the sequence, best hit, and some statistics in a results file. Importantly, each time it is invoked, it overwrites the previous contents of the output file. This is not suitable for recording results from multiple runs of the analysis. I attempted to fix it by simply switching the write mode to append, however this resulted in the output file containing duplicates. I tried to fix it in several ways, however, was unable to find the root of the problem and produce an elegant solution. The only solution that worked was to modify the script so that it keeps overwriting the file and then reading it again to write to a new output file in append mode.

After the analysis pipeline correctly records all the results, they are combined on the control node using ansible built-in fetch module. It copies the specified file from each of the target nodes and stores it in a folder under the target’s IP as a name. This allows to collect the file with the same name from multiple nodes without overwriting it. Next, the files are concatenated together, and a python script is run. It uses pandas to read the results, extract the required information and produce the two files specified by the researchers.

## 2.4 Security

I provide additional security to the system; I have created a key generation and distribution process that the research can use. The lecturer master key that is used during the cluster creation process can be used to access any of the clusters created for this module. This is dangerous, because if a leak happens on one of the control nodes, all clusters will be compromised. To avoid that, the researchers can run the key exchange playbook that generates a new key on the control node and adds it to authorised keys on all the worker nodes. To further improve the security of the system firewalld is installed and enabled on all of nodes in the cluster.

In a real-work scenario, the key exchange process would also remove the lecturer key file from the control node and remove the key from authorised keys on the workers. However, in the context of this coursework this is not implemented as it would interfere with the lecturer team’s ability to manage and troubleshoot the cluster in case of an emergency. It should still be possible to add the key back through the management console or aws cli, but for convenience it is left intact.

## 2.5 Monitoring

To monitor the state of the system I used Prometheus with Grafana for better visualisation. I configured my Grafana dashboard to display memory usage, load average, and percentage of tasks completed by each of the worker nodes (Figure 1). Metrics used to calculate memory usage and load average are reported by Prometheus node exporter by default. To enable progress tracking, I created a custom Prometheus metric for the worker nodes and added it to the node exporter configuration. To update this metric, I have added an additional step to the analysis pipeline, that runs at python script at the end of each task. The script counts the number of lines in the local results file and divides it by the total number of ids to be processed. It then updates the metric by re-writing the appropriate .prom file.

To make Prometheus aware of the node exporters, the addresses of all the worker nodes need to be added to the appropriate field in the Prometheus configuration file. I create the configuration file using a jinja2 template that automatically fills it with addresses of all workers available to the control node. This means that the monitoring part of the system can adapt to any cluster size.

A screenshot of a graph

Description automatically generated

(Figure 1: *Grafana dashboard*)

# Conclusion

The developed distributed analysis system for proteome analysis presented in this report makes a significant step towards improving the data processing efficiency of the biochemistry research team. Using Ansible with its agentless architecture and self-documenting code allowed to create a balance between an understandable and performant system that can be easily used by a research team unfamiliar with infrastructure-as-code tools.

Several challenges were encountered during the development, such as the management of large dependencies like pdb70 database and creating an effective work distribution process. The work distribution solution successfully addresses the different computational capabilities of each worker node and therefore ensures an appropriate distribution of tasks and significant reduction in processing time, compared to the researchers’ initial unparallelised solution. The system is capable of processing 6000 ids in under 48 hours.

However, the system can be further improved using dynamic load balancing to account for parts of the pipeline that can be parallelised over multiple cpu threads. This can be addressed by incorporating containerisation and Kubernetes for more advanced orchestration that is better suited for a complex load balancing process. This however will certainly reduce the transparency of the system and increase overhead due to increased complexity.

In addition, security measures such as the key generation and distribution module and the use of firewalld improve the security of the system and protect the cluster used by the researchers as well the other clusters provisioned using the same key in case of a leak.

Finally, the monitoring part of the system, implemented with Prometheus and Grafana, allows to visualise the current state of the system to help detect and troubleshoot potential issues and keep track of the progress in real time.

Overall, the system achieves its primary goal of reducing the processing times for the research team and therefore can be considered a valid solution. Additionally, the is scalable and takes into consideration future needs of the research team. For example, the task distribution and monitoring parts of the system account for potential variability in the size of the cluster and computational resources of individual nodes.