COMP0235 Coursework challenge

David Roman Frischer

January 1, 2024

Contents

1 Introduction

The goal of this coursework is to distribute the task of predicting possible 3D structures of proteins in the Human Genome. It's assumed that the reader is familiar with the pipeline shown in Figure 1, and analyzing one sequence may take up to 20 minutes to generate results. This work delivers a PySpark system to efficiently distribute the workload, along with a full-stack application that enables researchers to use the system without the need for command line expertise.

2 Methodology

The pipeline in Figure 1 displays the Directed Acyclic Graph (DAG) for analyzing a single protein. As shown, this process can be encapsulated as a single task, where each worker in the distributed system is assigned multiple proteins to predict their 3D structure. By distributing the work, the analysis will be parallelised across the number of workers. This allows researchers to run more analyses in less time.

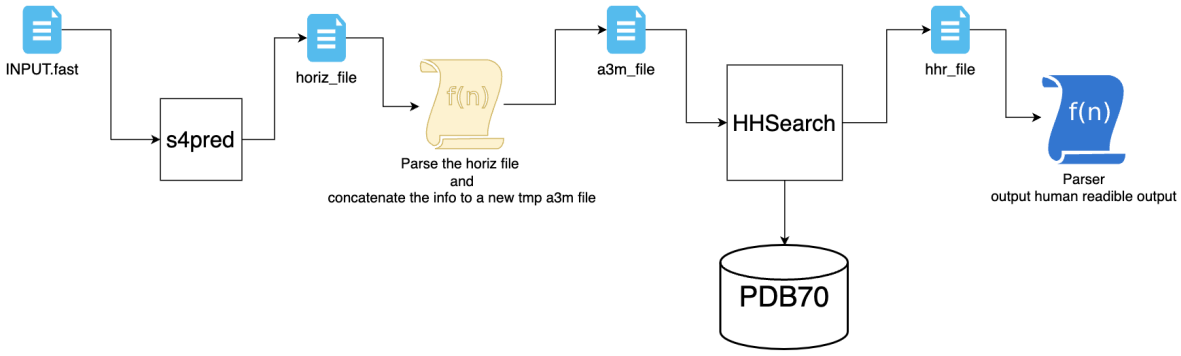

Figure 1: 3D Structure Prediction Pipeline

3 Requirements

During the development of the program, three requirement elements should be considered to ensure successful delivery.

3.1 Idempotency

All installations must be idempotent, meaning they should not download or install files again if they already exist on the machine. For example, without idempotency, the installation setup would consistently take 3 hours due to the lengthy database downloading time. With idempotency, Ansible will skip the download if the database is already installed on the virtual machine.

3.2 Stability

Researchers should always have access to the platform. Even when certain processes require high CPU usage, the web platform and Spark processes should always be running.

3.3 Connectivity

Using microservices increases the complexity of communication between processes. It is important that processes can communicate with each other through the LAN and that the SSH and web ports of the Client VM are always accessible through the WAN for managing and using the platform, respectively.

4 Cloud Infrastructure Setup

Ansible (https://www.ansible.com/) was chosen for setting up the infrastructure because it is lightweight, easy to install, and allows for granular configuration using roles, a IaC tool learned in COMP0235 lectures.

To execute Ansible scripts, the host machine should have the SSH private key to access all other virtual machines (VMs) in the system, and the Git token for repository access.[1]

Figure 2 illustrates the Ansible infrastructure with the Ansible roles to be installed on the VMs. The host VM, positioned at the top of the hierarchy, runs Ansible scripts to install the roles on the Worker VMs and the Client VM. The order of the installations is indicated by following the ascending numbers in the arrows.

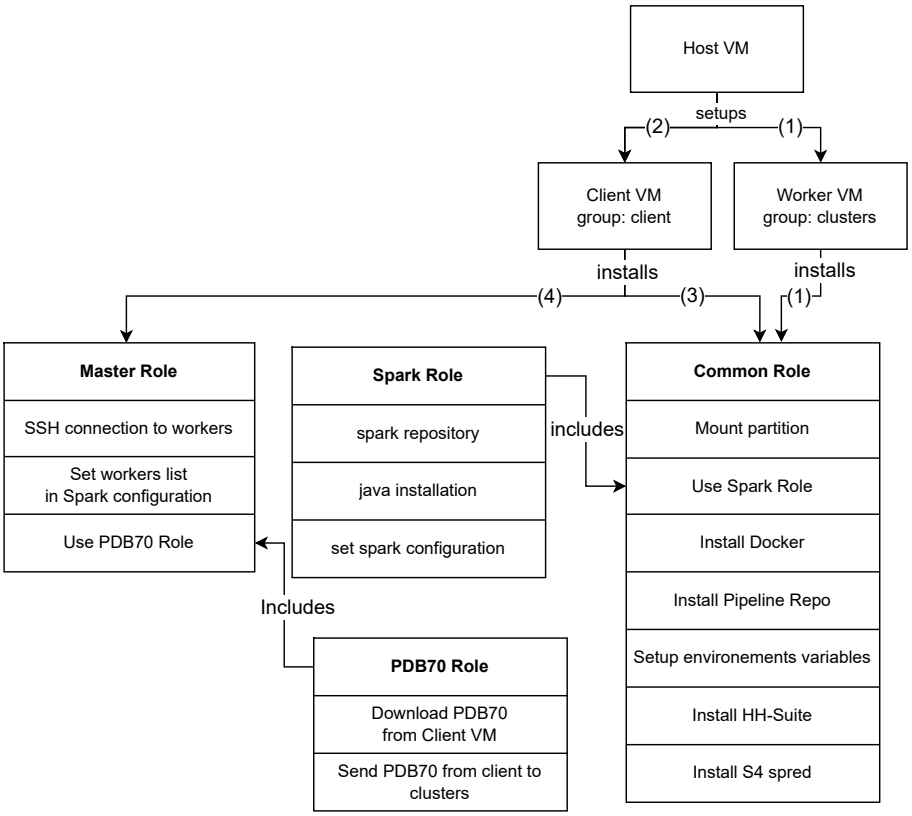

Figure 2: Ansible infrastructure

Four Ansible roles have been executed:

Role	Description
Common Role	Installations required for both Client VM and Worker VMs.
Master Role	All installations for the Client VM.
Spark Role	Installed within the Common Role, as all VMs must be either a Spark master or a Spark worker.
PDB70 Role	Download the PDB70 database in the client VM and send the PDB70 to the Workers VM from the client.

Table 1: Ansible Roles with their Descriptions

[1]These secrets are in the zip folder provided in the submission.

5 Choice of the Distributed System Engine

While Apache Spark was chosen as the Distributed System Engine, several other tools could have potentially served in this role for the project. Options such as a Hadoop cluster or a Message Queue tool like Celery or RabbitMQ were taken into consideration. The decision ultimately hinged on the learning motivation, fastest comprehension, and ease of installation associated with one of the tools. Apache Spark was as an excellent choice due to its simplicity, efficiency, and scalability with remote workers. Furthermore, it provides a logging GUI from the Master, introducing an additional reason of convenience to the system.

6 System Architecture

Figure 3 illustrates all the services in the system.

6.1 Client VM

The Client VM serves as the central component of the software, hosting all services except for the Spark workers in the distributed system.

- **Spark Master/GUI**: runs locally without Docker. Due to its complexity and the requirement to access to remote workers, a decision was made to run it locally. The Spark Master also provides a GUI for monitoring and logging of workers, accessible through NGINX.
- **NodeJS**: operates as the backend service for the platform. The frontend interacts with its API to perform tasks such as uploading a fasta file, searching a protein, initiating a run from a list of IDs, checking the current status of a run, and viewing a summary of all runs.
- **NGINX**: serves the frontend and acts as a reverse-proxy for the backend under `/api`. It also serves the Spark GUI under `/spark-master`.
- **Flask app**: is a simple Flask server responsible for launching a pipeline and starting a subprocess with Spark. It runs locally because it needs to initiate a subprocess to start a Spark application in the background.
- **Pipeline Script**: is executed from the Flask app as a subprocess to start a run.
- **Database**: is housed in a Docker container using the latest PostgreSQL docker image. PostgreSQL was chosen due to prior experience and its ease of connection through Object-Relational Mapping (ORM) software in Python and NodeJS.

6.1.1 Docker containers

NGINX, NodeJS, and the Database run within Docker containers. Docker simplifies the process of starting and stopping these services with a single command line using a `docker-compose.yml` file and eases the restoration process if the Client VM needs to be restored.

6.2 Worker VMs

Each worker VM has a copy of the PDB70 database and all necessary scripts and weights to run a prediction of a 3D protein structure. Even though is not showed in Figure 3, the workers has a client connection to the PostgreSQL database to save the results of a protein prediction.

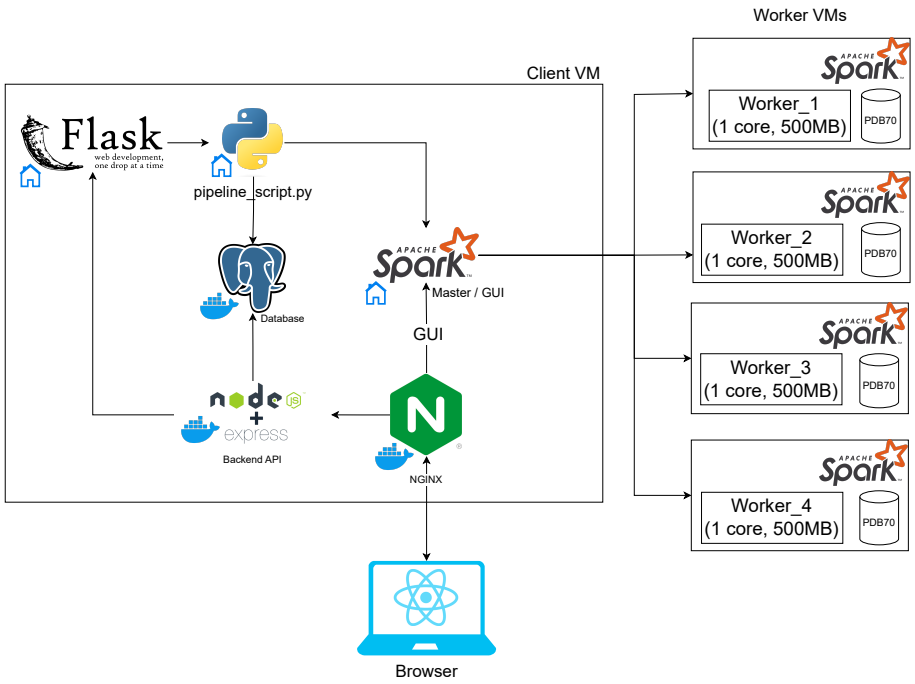

Figure 3: System Architecture

7 Data Management

The proteomics database in the PostgreSQL docker has three models and their relationships shown in Figure 4. The underlined attributes serve as the primary keys, uniquely identifying each tuple.

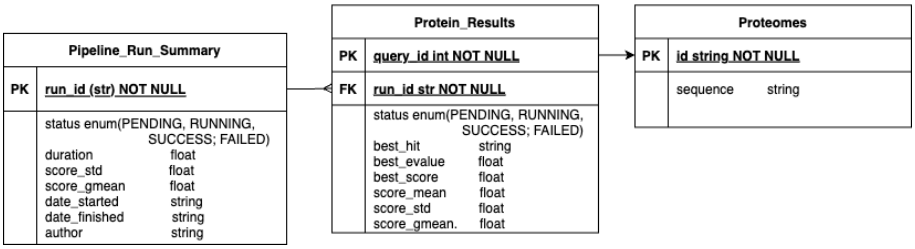

Figure 4: Relational Database Model of the Proteomics database

- **Pipeline Run Summary:**
  The pipeline run summary contains all the metadata for a run, such as the start date, the author, and the aggregated results of a run.
- **Protein Results:**
  The protein results tuple is the outcome of a protein 3D prediction. It also includes metadata such as its best hit, best evaluation, best mean, and so on.
- **Proteomes:**
  The proteomes model is the smallest unit of the database architecture, containing only the ID and the sequence of a given proteome.

8 Task Distribution

The task distribution is automatically done by PySpark through the `SparkContext.parallelize`[2] instance to form a RDD. The snipped code in Listing 1 is used to distribute the multiple protein ids to analyse.

```
# Distribute the ids in the sequence_list array into num_workers workers.
parallelised_data = spark.sparkContext.parallelize(sequence_list,numSlices=
    num_workers)
# Run the process_sequence function on each given id and sequence
parallelised_data.foreach(lambda x: process_sequence(x[0], x[1], run_id,
    sequence_list.index(x)))
```

Listing 1: Task Distribution with Spark

- `sequence_list`: A list of ids to be analysed during the run.
- `num_workers`: A configuration variable set to the number of available workers in the Spark environment.

The pipeline script is designed with the understanding that a run might fail during execution. Therefore, if a run is restarted, it will only process a list of IDs that have a pending, running, or failed status. For an already completed run, it will not redo successful predictions.

9 Temporary Files and Results Collation

During the pipeline process, each worker writes temporary files (.horiz, .a3m, .hhr, and .out). After a successful pipeline run, the worker saves the results to a `Protein_Results` record in the proteomics database. Subsequently, it deletes all temporary files from the completed run and initiates a new one if there are remaining sequences to analyse in its queue.

Once all workers complete their tasks, the Spark master queries the database to get the intermediate results. It concludes the run by generating the `merge_results.csv`, `best_hits.csv`, and `profile_output` files. Additionally, it stores this metadata in the corresponding tuple in the `Pipeline_Run_Summary` model.

10 Monitoring and Logging

Monitoring and logging can be approached in two ways.

1. **Developer's Perspective:** Developers can access Spark workers logs through the Master Spark GUI. It provides information on running time, the number of live workers, and the CPU and memory usage for a run.
2. **Researcher's Perspective:** Researchers can monitor their runs via the web platform. Clicking on the run ID in the list reveals details such as the number of proteins yet to be predicted, those currently running, failed and successful predictions. If a run fails, researchers can restart it from its last saved point without developer assistance. The platform's navigation header displays on the right the health status of services and the number of live workers in the Spark system.

11 Security

11.1 Network

The project incorporates good security practices, with room for improvement. Initially, only the Client VM has ports 22 for SSH and 80 for serving NGINX open. All other VMs have only port 22 open in the WAN, following AWS's default network settings. This pre-configured setup avoids the need for a custom firewall. To enhance security, it's advisable to restrict access to Cluster VMs only within the LAN.

[2] https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.SparkContext.parallelize.html

11.2 SSH Keys

A security consideration in the project is the use of the same SSH key for accessing all VMs. While this decision simplifies setup and access, a more secure approach would involve one SSH key pair from the Host to the Client and another SSH key pair from the client to the clusters.

11.3 Secrets

Critical secrets, such as `git_token`, database passwords, or RSA keys, are not accessible through the GitHub account. Access to these secrets requires contacting an administrator. Typically, keys and secrets should be regularly changed to enhance system security. For instance, the `git_token` will remain valid until March 22, 2024.

12 Challenges and Trade-offs

During the project development, several challenges were encountered, including the learning curve of Spark usage and the complexity of communication systems among services and virtual machines.

12.1 Hardware

The provided hardware for the project is considered sufficient for running the pipeline. Table 2 outlines the specifications of the available VMs.

	Quantity	Cores	RAM	Swap Memory	Disc Storage
Host VM	1	1	7.4G	0G	9.4G
Client VM	1	4	7.4G	0G	220.7G
Cluster VM	4	2	3.5G	0G	108G

Table 2: Hardware Specifications

A challenge with the hardware is the substantial disk storage needed for zipping (≈27GB) and unzipping (≈ 78GB) the PDB70. To send a copy of the PDB70 to the workers, the database had to be first unzipped in the Client VM, and then the uncompressed database was directly copied to the Cluster VMs.

Another challenge was the absence of SWAP Memory. SWAP memory is crucial to prevent VM crashes when a high amount of RAM is used. Unfortunately, the RAM usage the pipeline was underestimated, leading to VM crashes for very long protein sequences.

The hh-search algorithm, for example, uses a substantial amount of RAM. As shown in Figure 5, when analysing the sequence with the ID `sp|Q8WXI7|MUC16_HUMAN` in the Client VM, it can reach up to 62.2% of its RAM. To be precise, it utilises 4.6 GB of RAM, surpassing the maximum capacity of a Cluster VM, which has 3.5 GB RAM and no SWAP memory. Consequently, the Cluster VM crashes when processing this sequence.

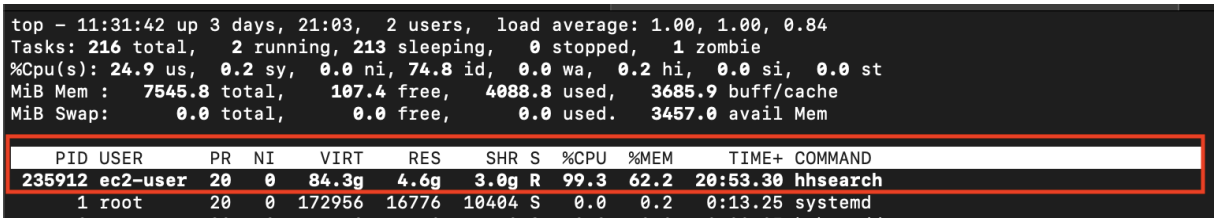

Figure 5: RAM usage of hh-search for id sp|Q8WXI7|MUC16_HUMAN in the Client VM.

During the analysis of the 6000 IDs, it was observed that the following sequences caused a crash when run in the Cluster VM. These sequences had to be executed in the Client VM to obtain the results:

- `sp|Q8WXI7|MUC16_HUMAN`
- `tr|A0A0A0MTR7|A0A0A0MTR7_HUMAN`
- `tr|A0A7P0MQR8|A0A7P0MQR8_HUMAN`
- `tr|A0A0G2JRA1|A0A0G2JRA1_HUMAN`

12.2 Spark configurations

Spark configuration was also a challenge because of the lack of experience. Spark can be highly configured and requires an understanding of how the source code is sent to workers. Configuring the reverse-proxy for spark GUI was also a bit challenging and even went to personal contribution to a question on the StackOverflow platform[3].

13 Future Improvements

- **Spark API:** Currently, a Flask Server is utilised in the project to initiate a Spark application in subprocess. A potential improvement is to directly use the Spark API to manage the applications, allowing for better control over starting, stopping, and monitoring their status.
- **Access token:** For enhanced security, access to the backend should only be granted through credential-based sign-in (e.g: JWT token). Once the backend verifies the client's identity, it will issue an access token to the client. This access token must be included in the header of subsequent requests made by the client to communicate with the backend service.
- **Full Dockerisation:** In the current system architecture, some processes run inside Docker containers, while others run directly on the physical host server. A more scalable and available solution would involve placing everything into Docker containers. The latter simplify the start and stop of processes and their network interactions.
- **Kubernetes Cluster:** An alternative consideration is moving the entire system into a Kubernetes cluster, provided all microservices have Docker images. This approach facilitates microservice updates without user access interruption and allows for efficient horizontal scaling of workers when the master detects a large list of sequences.

14 GitHub repository

The source code for the project is accessible on GitHub at the following link: https://github.com/davidfrisch/DataEng1. The repository includes all the source code, installation instructions, system startup guidelines, and brief examples for initiating and monitoring the pipeline.

15 Conclusion

In conclusion, the development and implementation of the system have been influenced by several key achievements. The decision to employ Ansible for the automated installation of essential files, Apache Spark as the engine for the distributed system, and the creation of a full-stack platform to enhance the initiation, execution, and monitoring of processes for a specified list of sequence IDs.

[3] https://stackoverflow.com/questions/45971127/wrong-css-location-of-spark-application-ui/77633558#77633558

A User Guide for the Platform

The guide is a walkthrough of the platform and will show every feature the platform has to offer. If the system is running, the platform should be available at the address of the Client VM from a web browser.

For reference, a Python client capable of executing all the features of the web platform is also accessible. A guide on its usage is provided in the GitHub project documentation in the `startup_guide.md` file.

A.1 Add a FASTA file

Users can upload a FASTA file on the platform by visiting the "Upload" page and dropping the file.

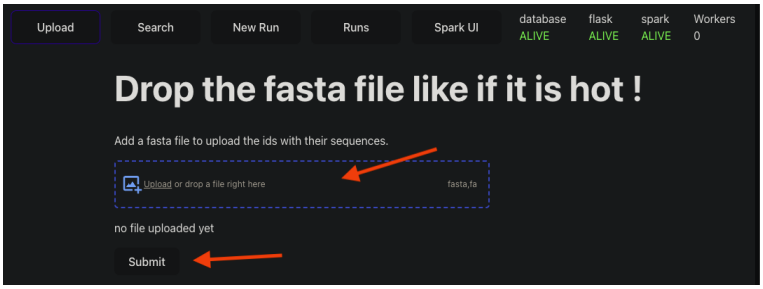

Figure 6: Drop the FASTA file and press submit

The server will send back the sequences successfully added to the database, those already present (which won't be added again), and those that failed.

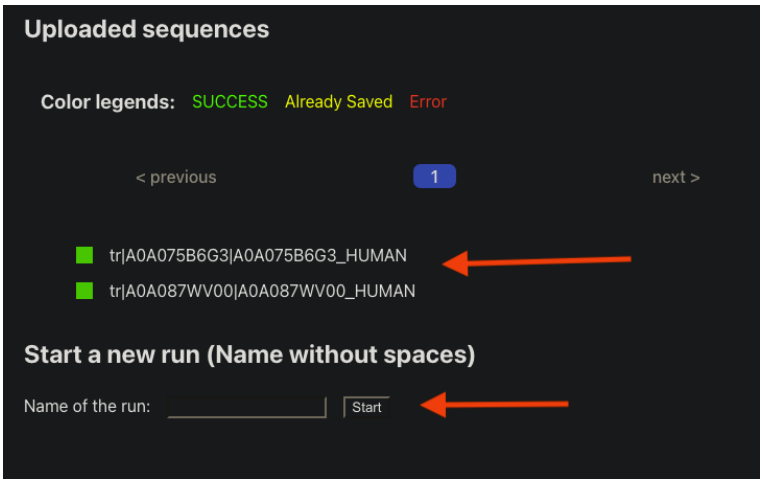

Figure 7: Status of the upload and start a run based on the FASTA file

On this page, users can initiate a run by entering the run name, derived from the IDs provided by the FASTA file, and then clicking "Start".

A.2 Search for a sequence ID

Users can search for a sequence ID by visiting the "Search" page. They should enter the ID in the input field and then click on the search button.

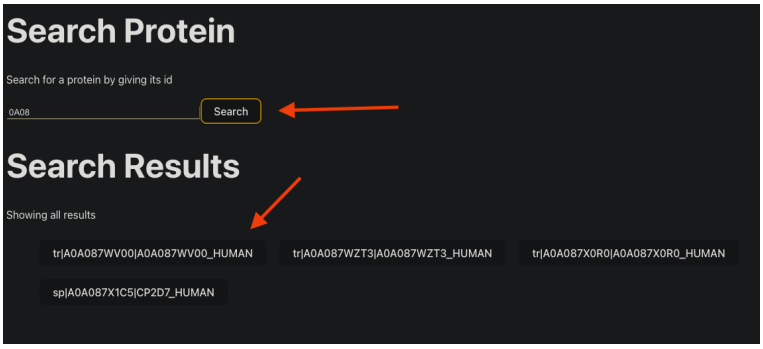

Figure 8: Search by id and select the best match

The server will return information about the searched ID. In the event that multiple IDs are found, users can choose one by clicking on the respective button.

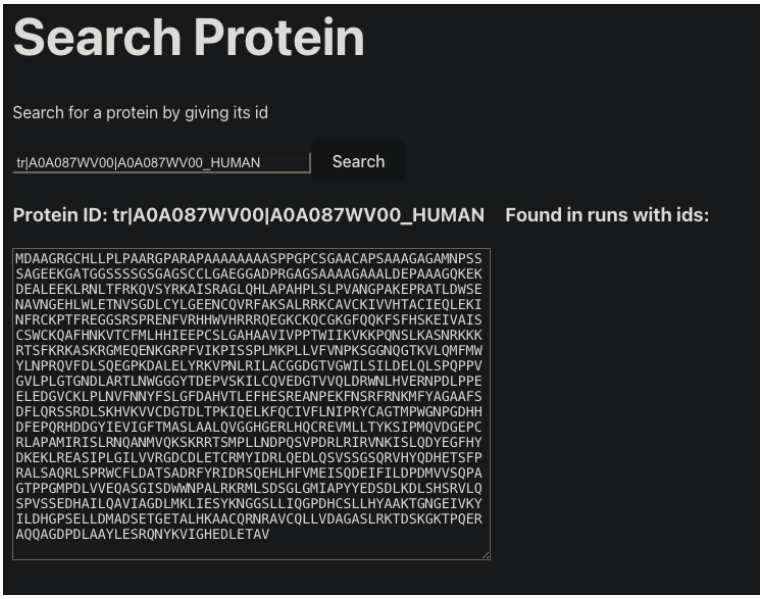

Figure 9: Information about a protein

The platform will display the ID, sequence, and all the runs where this has been analysed.

A.3 Start a run from a list of sequence ids

Users can initiate a run, from the "New Run" page, by providing a list of IDs in a text file. The file can be uploaded by dragging and dropping it onto the platform.

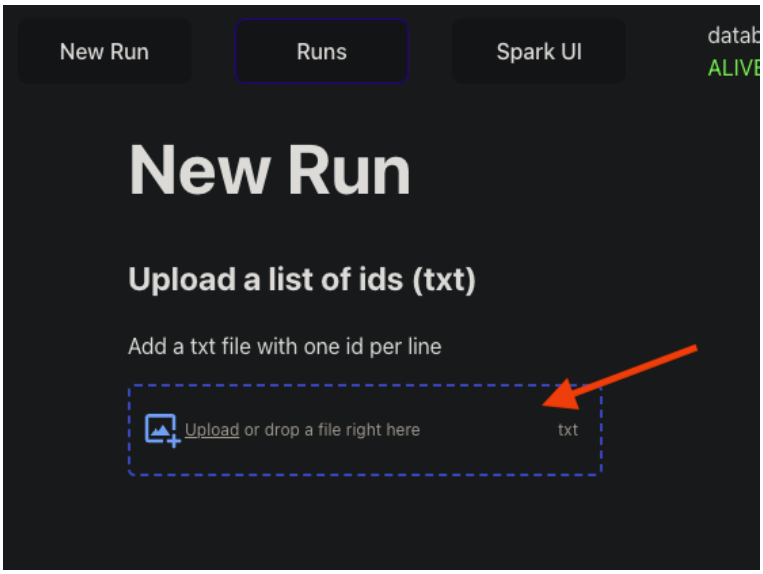

Figure 10: Upload ids by dropping it in the platform

The server will detect the absent IDs and recommend to the user either commencing the run without those missing IDs or clicking on the button pointed by the orange arrow, as shown in Figure **??**, to be redirected to the "Upload" page.

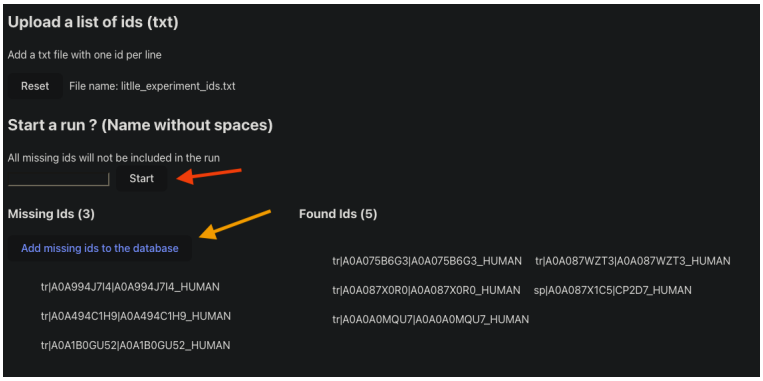

Figure 11: Upload ids by dropping it in the platform

Upon clicking "Start", the user may be redirected to the Spark UI, providing a developer's perspective for monitoring the run.

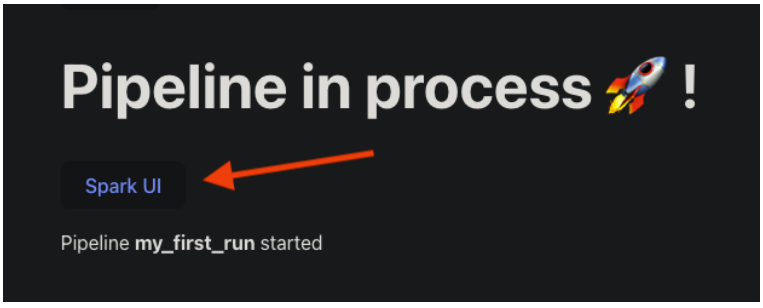

Figure 12: Successful Launch of a run

A.4 Access Spark GUI from the platform

The Spark GUI provides information on running applications, the count of active workers, and logs the steps of the analysis for a prediction.

Figure 13 displays a list of all current and previous applications. Clicking on a worker, as illustrated in Figure 14, will navigate to the worker's logs.

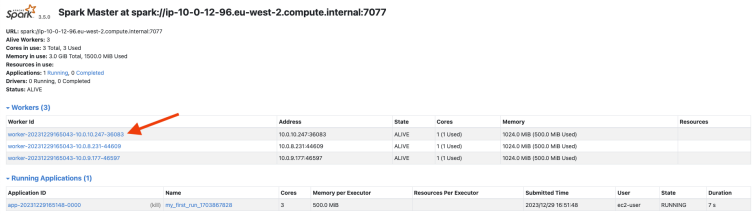

Figure 13: List of current and past applications

Figure 15 shows the logs of a Spark worker. Timestamp and prediction step is indicated at each line.

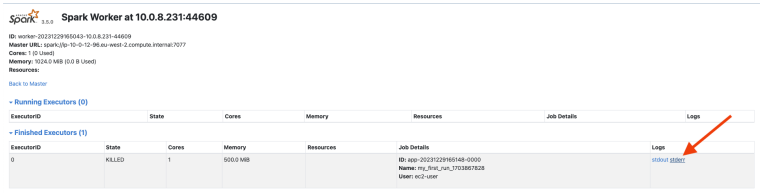

Figure 14: List of applications for which a worker performed tasks.

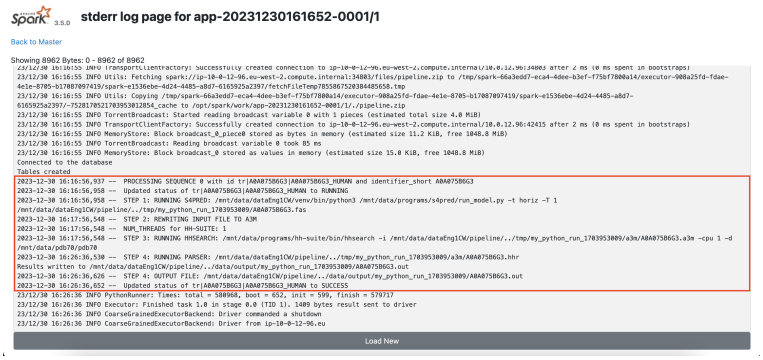

Figure 15: Logs detailing a prediction from a Spark worker.

A.5 View progress of a run

The web platform also provides a real-time update on the current status of the run by clicking on its ID from the "Runs" page.

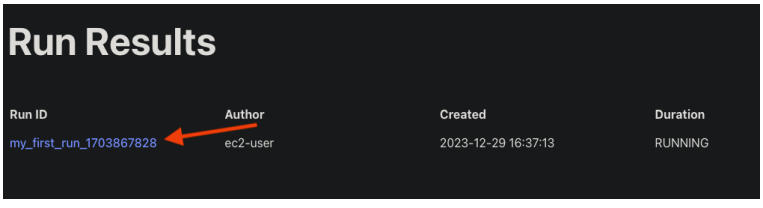

Figure 16: Select a run from the list of runs

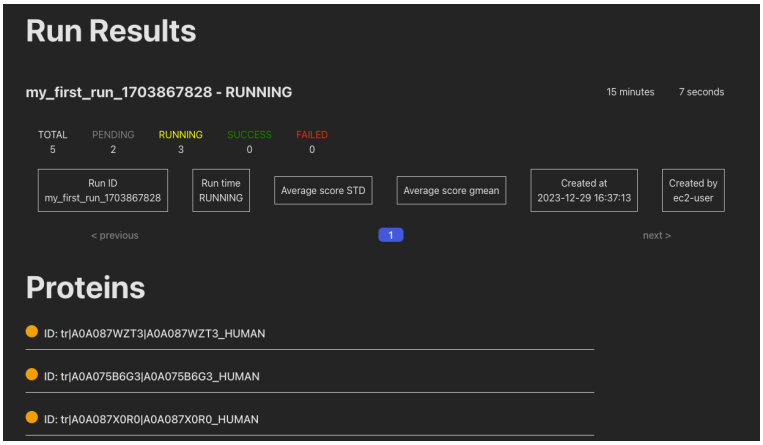

Figure 17: List of running predictions

Figure 17 shows all proteins that will be predicted in the run.

Once a worker has completed predicting a protein, the user can view the results and access more information by being redirected to the RCSB page.

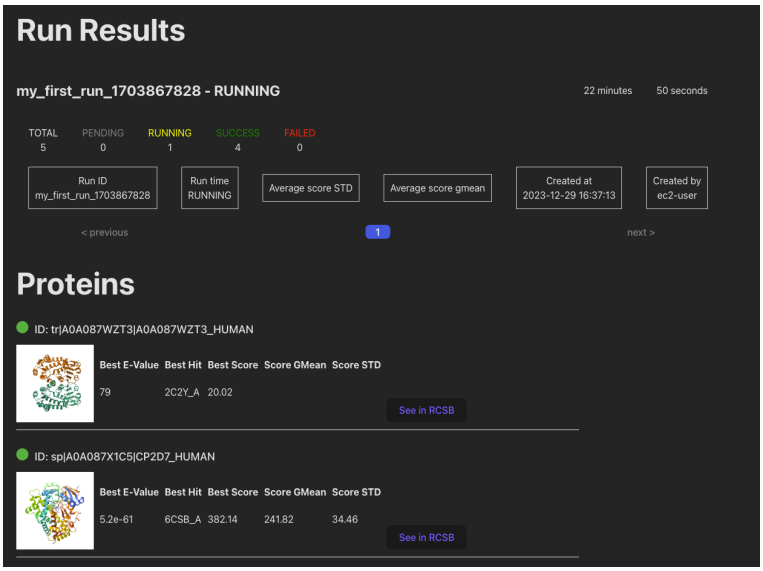

Figure 18: Successful predictions and 1 prediction still in process.

A.6 Retry a run

If a run a crashes, users can attempt to retry the run by clicking on the "Retry Run" button. This action initiates a new run, placing all pending, running, and failed runs to a pending state.

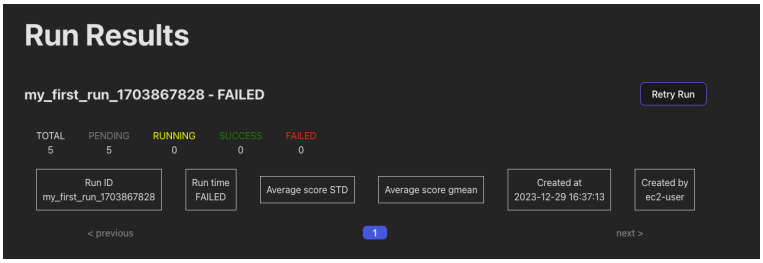

Figure 19: 5 predictions are still pending

A.7 Download the results of a run

Once there are no more pending predictions, the run will be considered finished. Users can download the `merge_results.csv`, `profile_output.csv`, and `best_hits.csv` files by clicking on the "Download" button.

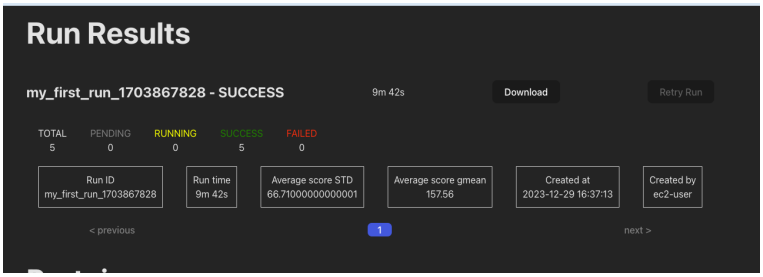

Figure 20: Download results of a run by a click of a button