



**UNIVERSITY
OF MALAYA**
K U A L A L U M P U R

WQD 7008 – Parallel & Distributed Computing

Group Project

Faculty of Computer Science & Information Technology

Programme Master of Data Science

Academic Session 2021/2022

Semester 1

Lecturer Dr. Liew Chee Sun

Heart Disease Prediction System With Parallel and Distributed Workflow

Group Members

Name	Matric Number
HuiJun Liu	S2142285
Lingyu Meng	S2131391
Zhu Daming	S2141978
Jinxing Li	S2128176
Ruixue Zhang	S2142119

Table of Contents

<u>1. INTRODUCTION</u>	<u>3</u>
1.1 DATA SOURCE	4
1.2 DATA PREPROCESSING	5
1.3 WHY USE HTCONDOR	5
1.4 OBJECTIVE	5
1.5 PARALLEL & DISTRIBUTED COMPUTING	6
1.6 HTCONDOR HIGH THROUGHPUT COMPUTING ENVIRONMENT	7
<u>2. LITERATURE REVIEW</u>	<u>7</u>
<u>3. ARCHITECTURE DESIGN</u>	<u>8</u>
<u>4. BACKGROUND OF USE CASE</u>	<u>10</u>
<u>5. SETTING UP DISTRIBUTED DATA PROCESSING WORKFLOW</u>	<u>14</u>
5.1 AWS INSTANCES	14
5.2 SETTING HTCONDOR POOL	15
5.3 SETTING NETWORK FILE SYSTEMS (NFS)	16
5.5 DAGMAN WORKFLOW	18
5.5.1 OUR WORKFLOW	18
5.5.2 DETAILED PROCESS	19
5.6 CHALLENGES	24
<u>6. RESULTS AND DISCUSSIONS</u>	<u>25</u>
<u>7. CONCLUSION AND FUTURE WORKS</u>	<u>26</u>
<u>8. REFERENCES</u>	<u>28</u>



1. Introduction

This article researches the uses of parallel and distributed computing techniques to analyze large amounts of patient data and predict the risk of heart disease. The goal of the system is to provide healthcare providers with a tool for early detection and intervention of heart disease, which can lead to better outcomes for patients. The system can include several key components such as data collection, data preprocessing, model training, and model evaluation.

The data collection component is responsible for collecting patient data such as age, sex, blood pressure, cholesterol levels, and other risk factors. This data can be collected from electronic health records, medical imaging, and other sources.

The data preprocessing component is responsible for cleaning, normalizing, and preparing the data for analysis. This can include tasks such as removing missing values, handling outliers, and encoding categorical variables.

The model training component is responsible for using machine learning algorithms to train a predictive model on the preprocessed data. This component can use parallel and distributed computing techniques to train the model on large amounts of data in a reasonable amount of time.

The model evaluation component is responsible for evaluating the performance of the trained model on a holdout dataset. This component can use parallel and distributed computing techniques to perform hyperparameter tuning and feature selection.

The overall workflow of the system can be managed by a Workflow Execution Manager (WEM) based on HTCondor DAGMan (Directed Acyclic Graph Manager) engine, which makes the system highly extensible and domain-agnostic.

In summary, uses parallel and distributed computing techniques to analyze large amounts of patient data, train a predictive model, and evaluate its performance. It is designed to provide healthcare providers with a tool for early detection and intervention of heart disease, which can lead to better outcomes for patients.

1.1 Data Source

Our data source is The Heart Disease Dataset from Kaggle

<https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-heart-disease>

The original dataset contains 18 variables and 319,795 observations. The following table shows the descriptions of each variable.

Our methodology contains 5 steps which are data collection, EDA, data cleaning, modeling, and assessment. First we conduct exploratory data analysis to find patterns and examine the data quality. At the stage of data cleaning, we deal with the noisy data and scale the data. After that, 3 different models will be applied to predict the target variable heart disease. Finally we assess the models based on evaluation metrics and determine the best model. We will conduct the whole process on both the local machine and AWS EC2 instance respectively. We will assess the running time in the conclusion.

1.2 Data preprocessing

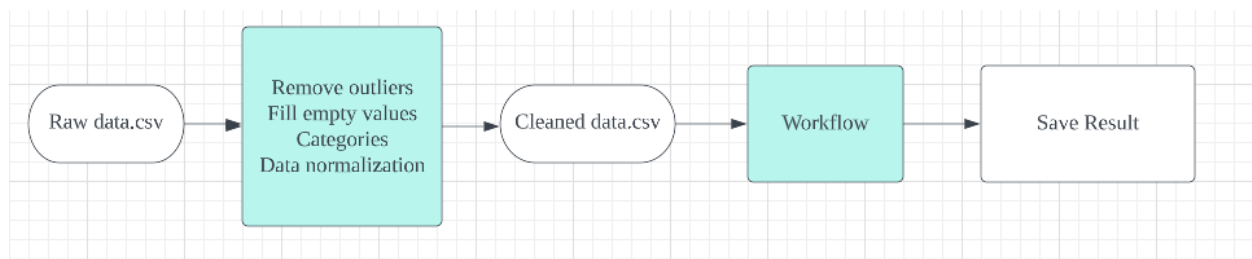


Figure 1.1 Data preprocessing

After we get the original data through Kaggle, removed outliers, using the mean or median to fill in null values, categories, and data normalization, clean data is obtained, and then we add clean data to our workflow.(Figure 1.1)

1.3 Why use Htcondor

Using HTCondor, the dataset can be split into smaller chunks and distributed across multiple machines for parallel processing. The dataset can be preprocessed and then fed into a machine learning algorithm for training. Once the model is trained, HTCondor can be used to evaluate the model's performance on the test set of the dataset in parallel.

Additionally, HTCondor can be used to perform hyperparameter tuning, which is the process of finding the best set of parameters for a machine learning model. This can be done by using HTCondor to train multiple models with different parameter values in parallel and then evaluating their performance on the test set of the dataset.

HTCondor also provides a powerful scheduling algorithm that allows to assign jobs to the resources that are available. This helps to optimize the use of computational resources and can significantly speed up the training and evaluation of machine learning models.

1.4 Objective

We have two main objectives for our project

- To predict heart disease using 3 different ML models.

- To reduce the training time by running it in a parallel and distributed computing environment.

1. To Utilizing HTCondor for large-scale parallel training By splitting the heart disease dataset into multiple chunks and training machine learning models on multiple machines in parallel, HTCondor can significantly speed up the training process.
2. To Utilizing HTCondor for hyperparameter tuning By training multiple models with different parameter values on multiple machines in parallel and evaluating their performance, HTCondor can help to find the optimal set of parameters for a machine learning model.
3. To Utilizing HTCondor for performance evaluation of the prediction model By evaluating the performance of the prediction model on multiple machines in parallel and using different evaluation metrics, HTCondor can help to assess the quality of the model.
4. To Utilizing HTCondor for feature selection By selecting and filtering features on multiple machines in parallel, HTCondor can help to identify the most informative features for predicting heart disease.
5. To Utilizing HTCondor for model deployment By using HTCondor to deploy the model on multiple machines, it can provide the ability to handle large scale predictions and processing in parallel.
6. To Utilizing HTCondor for scaling up or down the resources based on the load, this can save the cost and improve the efficiency of the system.

1.5 Parallel & Distributed Computing

In recent years, parallel and distributed computing have become increasingly important as data and computation requirements have grown. With the increasing availability of large-scale data and the need to analyze and make decisions from it in real-time, parallel and distributed computing have become a necessary tool to handle these requirements.

One of the most notable trends in parallel and distributed computing is the use of cloud computing. Cloud-based services such as Amazon Web Services (AWS) and Microsoft Azure allow companies and organizations to easily access and scale computing resources on-demand.

This has made it easier for companies and organizations to take advantage of parallel and distributed computing without having to invest in their own infrastructure.

Another trend is the use of parallel and distributed computing in machine learning and artificial intelligence. With the growing amount of data and the need to train larger and more complex models, parallel and distributed computing have become essential for training machine learning models in a reasonable amount of time.

1.6 HTCondor High Throughput Computing Environment

HTCondor is a high-throughput computing (HTC) system that allows for the distributed execution of computations across multiple machines. It is designed to manage and schedule the execution of large numbers of batch jobs and can be used to process large data sets or perform complex simulations. HTCondor can be used to manage both Linux and Windows machines, and it supports a wide range of job types, including serial, parallel, and MPI jobs.

2. Literature Review

Machine learning and parallel computing have been used as fast tools for data mining, data training, and feature selection.

Ramezani et al. proposed a load balancing method that reduces service response time, memory usage, job completion time, and power consumption. This shows that using clusters for load balancing can reduce our costs and improve the accuracy of predictions when dealing with large amounts of data.

Grzeszczyk argue that the use of parallel computing during supervised machine learning can reduce computing time. Sequential computing is often insufficient for solving large-scale problems such as complex simulations or real-time tasks. So, using three parallel models (shared memory, distributed memory, and hybrid models), experimental development with machine learning and neural network algorithms, they study and optimize the computation time during the learning process of supervised perceptrons, and arrive at this in conclusion.

Jung-Lok In the analysis of astronomy data efficiently, a workflow system is proposed in analyzing large-scale . Workflow execution manager (WEM) which represents the sequences of tasks with their dependencies is based on the HTCondor DAGMan (Directed Acyclic Graph MANager)engine, it makes WEM domain-agnostic and highly extensible. This proves the importance of workflow and usage of DAGMan in analyzing large scales of data.

In addition, we studied articles on heart disease prediction to improve the accuracy of heart disease prediction by applying machine learning techniques to find important features. Mohan et al. used 7 different machine learning algorithms, among which the linear mixed random forest model (HRFLM) was the most accurate prediction model for heart disease with an accuracy of 88.7%.

3. Architecture Design

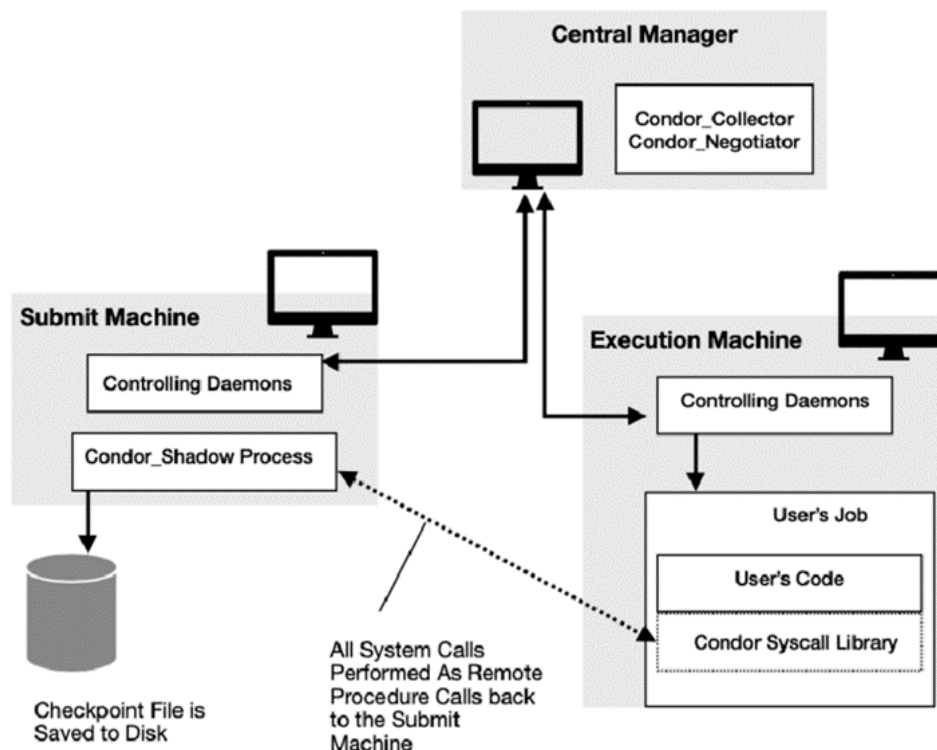


Figure 3.1 Architecture design

The Central Manager This is the main component of HTCondor that coordinates the activities of all other components. It keeps track of the status of all machines in the pool, schedules jobs, and manages the allocation of resources.

The Scheduler This component is responsible for deciding which jobs to run on which machines. It uses a scheduling algorithm to make decisions based on factors such as available resources, priority, and job requirements.

The Start This component runs on each machine in the pool and is responsible for reporting the machine's status to the Central Manager and starting and stopping jobs as directed by the Scheduler.

The Shadow This component runs on each machine and acts as an intermediary between the Startd and the job. It monitors the job's progress and communicates with the Startd to request resources or terminate the job.

The Executor This component runs on each machine and is responsible for starting and stopping jobs. It communicates with the Shadow to request resources and receive instructions.

The Collector This component collects and stores information about the status of machines and jobs in the pool. It provides a centralized location for querying the status of the pool.

The Negotiator This component is responsible for negotiating the allocation of resources between the Central Manager and the Scheduler. It takes into account the availability of resources and the priority of the jobs.

The NFS clients from Submit and Execute Nodes mount the shared folder from Master Node to /home/ec2-user/project.

Here is the summary of the tools used for our set up

Table 3.1 Tool Descriptions

Tool	Description
AWS	To set up three machines in the cloud

HTCondor	Resource management system for compute intensive jobs
DAGman	Manage and queue jobs within a specified DAG for execution on remote machines
NFS	Used for a centralized file management
Python	<p>Scripting language used for machine learning model training</p> <p>Packages installed</p> <ul style="list-style-type: none"> - pandas - sklearn - xgboost - seaborn
Terminal (Windows or Mac)	To connect to each server's shell and to transfer files

4. Background of use case

Parallel computing has gained a great influence on scientific researches and in our daily life, especially when dealing with big data. Parallel computing is a method to solve large problems that can be divided into smaller ones which can be solved simultaneously, and has been used widely. Computer cluster can be considered as an implementation of parallel computing, which can be viewed as a single system in many respects, overcoming the limited performance of a single computer and improving performance. Computer cluster are also used to support high performance of parallel visualization of large medical datasets. With the development of internet, medical image can be displayed based on the web, remote collaboration over medical image are also available by using web services (Wu et al., 2014). In the past decade, machine learning, especially deep learning, has been successfully applied in many areas including computer vision, speech recognition, and natural language processing. In large-scale machine learning models, such as deep neural networks, there are usually millions and billions of model parameters. They require large amounts of training data to learn these parameters. The whole training process on large-scale training datasets is both computation-intensive and memory-intensive. Thus, the training of large-scale machine learning models is also very time consuming. For instance, training AlexNet on ImageNet costs five and six days with two GTX 580 GPUs. Furthermore, many powerful computational facilities such as GPU and TPU need to be used to train large-scale machine learning models, leading to huge economic costs. Hence it is very important to come up with parallel and distributed training algorithms to drastically reduce the training overhead (Gu et al., 2018). Distributed and parallel processing emerged as a solution to solve complex problems using nodes that have multiple processors or nodes connected to each other across a network (Kambatla et al., 2014). The shift from sequential processing to distributed and parallel processing provides high performance and reliability for applications. But, it also introduces new challenges in terms of hardware architectures, inter-process communication technologies, algorithms and systems design.

Our proposed work is how to execute parallel machine learning workflows in a distributed environment, or rather, how one can run machine learning algorithms in a distributed and parallel manner during the classification. Distributed is reflected in the application of htcondor, which uses three nodes, master submit execute, to build a distributed system. Each component in the system represents a different role to perform different tasks. The second embodiment is the application of nfs. Nfs is a distributed file system that can mount files. Parallelism is reflected in

our machine learning workflow that needs to train three models at the same time to save time. Using DAGMan can achieve parallel submission tasks. DAGMan is a HTCondor tool that allows multiple jobs to be organized in workflows, represented as a directed acyclic graph (DAG). A DAGMan workflow automatically submits jobs in a particular order, such that certain jobs need to complete before others start running. This allows the outputs of some jobs to be used as inputs for others, and makes it easy to replicate a workflow multiple times in the future.

According to the World Health Organization, heart disease is the leading cause of death globally, accounting for 33% of all deaths in 2019. Heart disease diagnoses roughly tripled from 271 million in 1990 to 523 million in 2019, while heart disease deaths increased from 12.1 million to 18.6 million (World Health Organization, 2021). In this project, several independent variables (BMI, Smoking, Alcohol, Drinking, Stroke, Physical Health, Diff Walking, Sex, Age Category, Diabetic, Physical Activity, Sleep Time, Asthma, Kidney Disease, Skin Cancer) will be used along with a dependent variable (heart disease) during the training phase to build a suitable prediction model to predict how likely a patient to get heart disease in the future. Another use case is the development of personalized treatment plans for heart disease patients. By using machine learning and parallel computing to analyze large amounts of patient data, healthcare providers can identify patterns and risk factors that are specific to each patient. This information can be used to create personalized treatment plans that are tailored to the individual patient's needs, which can lead to better outcomes. Additionally, parallel computing has been used to develop and improve models for simulating the cardiac mechanics, which allows to evaluate different therapeutic strategies and the effect of different pathologies in the heart. This is beneficial for researchers, engineers and clinicians, because they can analyze the cardiac mechanics in silico before applying it in vivo.

Table 4.1 Dataset Information

ATTRIBUTES	TYPES	DESCRIPTION
------------	-------	-------------

HeartDisease	Categorical	Respondents with coronary heart disease (CHD) or myocardial infarction (MI) history
BMI	Numerical	Body Mass Index (BMI)
Smoking	Categorical	Respondents who have smoked at least 100 cigarettes in their entire life? Note: 5 packs = 100 cigarettes
AlcoholDrinking	Categorical	Heavy drinkers · Adult men having more than 14 drinks per week · Adult women having more than 7 drinks per week
Stroke	Categorical	Respondents who have history of stroke
Physical Health	Numerical	During past 30 days, how many days respondent had physical illness and injury?
DiffWalking	Categorical	Respondents who experience walking difficulties
Sex	Categorical	Gender of the respondents

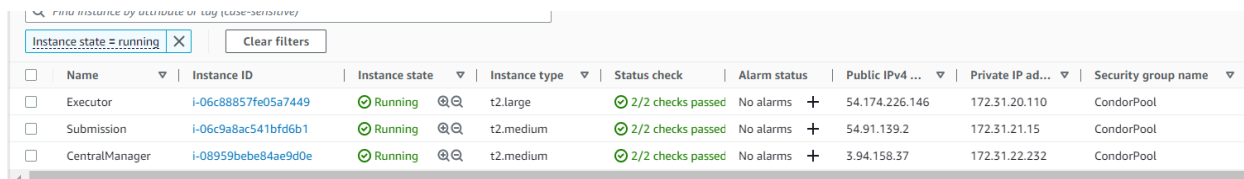
AgeCategory	Categorical	Age of the respondents
Diabetic	Categorical	Respondents who have history of diabetic
PhysicalActivity	Categorical	Respondents who reported doing physical activity or exercise during the past 30 days other than their regular job
SleepTime	Numerical	Respondent's average sleep time in a day (24 hours)
Asthma	Categorical	Respondents who have history of Asthma
KidneyDisease	Categorical	Respondents who have history of kidney disease
SkinCancer	Categorical	Respondents who have history of skin cancer

Our dataset consists of 15 variables and 319796 rows.

5. Setting up distributed data processing workflow

5.1 AWS Instances

We launched 3 AWS EC2 instances of Central Manager, Submission and Executor. The Executor is assigned a better performance than the master and submission nodes.



Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Public IPv4 ...	Private IP ad...	Security group name
Executor	i-06c88857fe05a7449	Running	t2.large	2/2 checks passed	No alarms	54.174.226.146	172.31.20.110	CondorPool
Submission	i-06c9a8ac541bfd6b1	Running	t2.medium	2/2 checks passed	No alarms	54.91.139.2	172.31.21.15	CondorPool
CentralManager	i-08959bebe84ae9d0e	Running	t2.medium	2/2 checks passed	No alarms	3.94.158.37	172.31.22.232	CondorPool

Figure 5.1.1 AWS EC2 instances

Table 5.1 details of our three machines in the cloud

Machine	System	CPU's	RAM
Executor	Ubuntu 20.04	2	8
Submission	Ubuntu 20.04	2	4
CentralManager	Ubuntu 20.04	2	4

5.2 Setting HTCondor pool

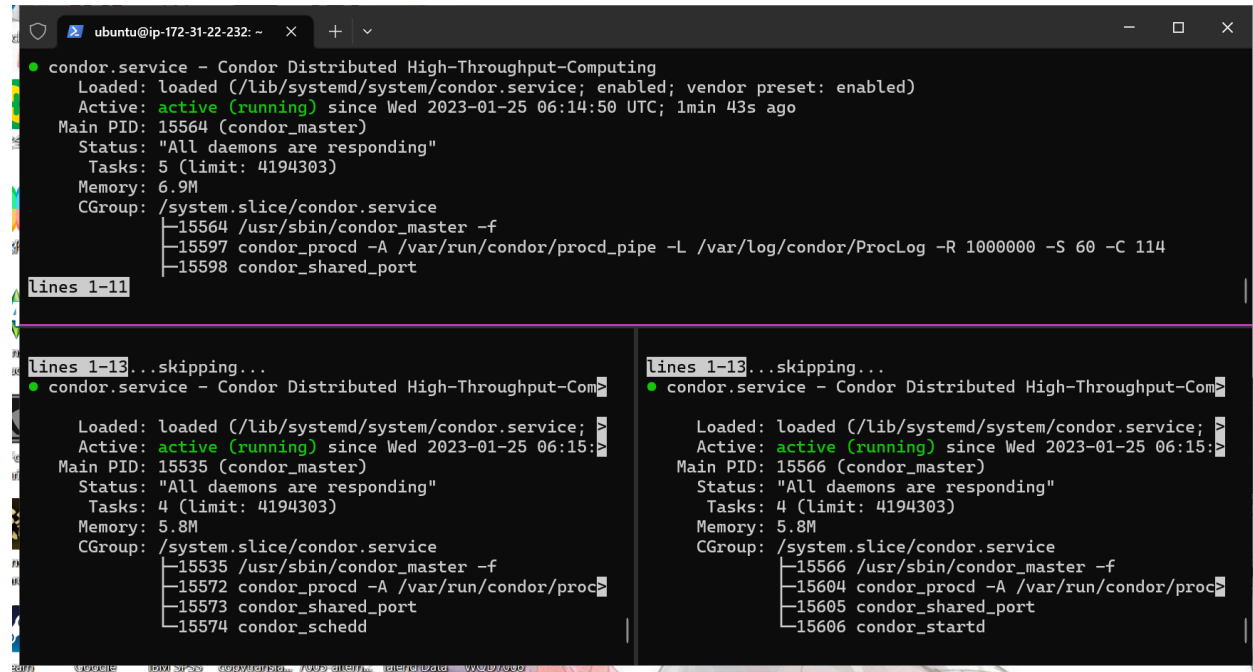
First, we assign roles to each machine by modifying the /etc/hosts file.

```
127.0.0.1 localhost

172.31.22.232 CentralManager
172.31.21.15 SubmissionHost
172.31.20.110 Executor
# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts

~
```

Then we can get htcondor using the command from the user manual. We can check the status of the condor by `sudo systemctl status condor`.



```
● condor.service - Condor Distributed High-Throughput-Computing
   Loaded: loaded (/lib/systemd/system/condor.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2023-01-25 06:14:50 UTC; 1min 43s ago
     Main PID: 15564 (condor_master)
    Status: "All daemons are responding"
       Tasks: 5 (limit: 4194303)
      Memory: 6.9M
     CGroup: /system.slice/condor.service
            └─15564 /usr/sbin/condor_master -f
              └─15597 condor_procd -A /var/run/condor/procd_pipe -L /var/log/condor/ProcLog -R 1000000 -S 60 -C 114
                └─15598 condor_shared_port

lines 1-11
```

```
lines 1-13...skipping...
● condor.service - Condor Distributed High-Throughput-Computing
   Loaded: loaded (/lib/systemd/system/condor.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2023-01-25 06:15:00 UTC; 1min 43s ago
     Main PID: 15535 (condor_master)
    Status: "All daemons are responding"
       Tasks: 4 (limit: 4194303)
      Memory: 5.8M
     CGroup: /system.slice/condor.service
            └─15535 /usr/sbin/condor_master -f
              └─15572 condor_procd -A /var/run/condor/procd_pipe -L /var/log/condor/ProcLog -R 1000000 -S 60 -C 114
                └─15573 condor_shared_port
                  └─15574 condor_schedd

lines 1-13...skipping...
● condor.service - Condor Distributed High-Throughput-Computing
   Loaded: loaded (/lib/systemd/system/condor.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2023-01-25 06:15:00 UTC; 1min 43s ago
     Main PID: 15566 (condor_master)
    Status: "All daemons are responding"
       Tasks: 4 (limit: 4194303)
      Memory: 5.8M
     CGroup: /system.slice/condor.service
            └─15566 /usr/sbin/condor_master -f
              └─15604 condor_procd -A /var/run/condor/procd_pipe -L /var/log/condor/ProcLog -R 1000000 -S 60 -C 114
                └─15605 condor_shared_port
                  └─15606 condor_startd
```

5.3 Setting Network File Systems (NFS)

We set the Submission node as the NFS server and exported the directory of `/home/ubuntu/project`.


```

ubuntu@ip-172-31-21-15:~$ sudo systemctl status nfs-server
● nfs-server.service - NFS server and services
   Loaded: loaded (/lib/systemd/system/nfs-server.service; vendor preset: enabled)
   Active: active (exited) since Wed 2023-01-25 06:27:05 UTC; 1min 1s ago
   Main PID: 16643 (code=exited, status=0/SUCCESS)
     Tasks: 0 (limit: 4689)
    Memory: 0B
     CGroup: /system.slice/nfs-server.service

Jan 25 06:27:05 ip-172-31-21-15 systemd[1]: Starting NFS >
Jan 25 06:27:06 ip-172-31-21-15 systemd[1]: Finished NFS >
ubuntu@ip-172-31-21-15:~$ |

```

```

ubuntu@ip-172-31-21-15:~$ cat /etc/exports
# /etc/exports: the access control list for filesystems wh
ich may be exported
#
#       to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes          hostname1(rw,sync,no_subtree_check) hos
tname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4           gss/krb5i(rw,sync,fsid=0,crossmnt,no_su
btree_check)
# /srv/nfs4/homes    gss/krb5i(rw,sync,no_subtree_check)
#
/home/ubuntu/project  *(rw,sync,no_subtree_check)

```

Use the **sudo mount** command to mount the nfs file systems.

```

# Example for NFSv4:
# /srv/nfs4           gss/krb5i(rw,sync,fsid=0,crossmnt,no_su
btree_check)
# /srv/nfs4/homes    gss/krb5i(rw,sync,no_subtree_check)
#
/home/ubuntu/project  *(rw,sync,no_subtree_check)
ubuntu@ip-172-31-21-15:~$ cd project
ubuntu@ip-172-31-21-15:~/project$ ls
test
ubuntu@ip-172-31-21-15:~/project$ |
Processing triggers for libc-bin (2.31-0ubuntu9.9) ...
ubuntu@ip-172-31-20-110:~$ sudo mount SubmissionHost:/home
/ubantu/project data/
ubuntu@ip-172-31-20-110:~$ ls
data
ubuntu@ip-172-31-20-110:~$ cd data
ubuntu@ip-172-31-20-110:~/data$ touch test
ubuntu@ip-172-31-20-110:~/data$ ls
test
ubuntu@ip-172-31-20-110:~/data$

```

5.5 DAGMan WorkFlow

In this section, we will perform how we apply DAGMan workflow to manage the machine learning process.

5.5.1 Our workflow

The following figure shows the detailed information of each node. First we conduct exploratory data analysis to extract patterns, the output of this node is some statistical data and interesting plots. The next node is data cleaning which we prepare cleaned and scaled data used for modeling. The output of this node is the new dataset and an empty csv file used to collect results from the following node. After that is a parallel processing, we conduct 3 different ML models on the cleaned dataset and record the performance of each model, the result will be written to the existing csv file. Finally is the last node of model assessment, which we use the result file to determine the best model.

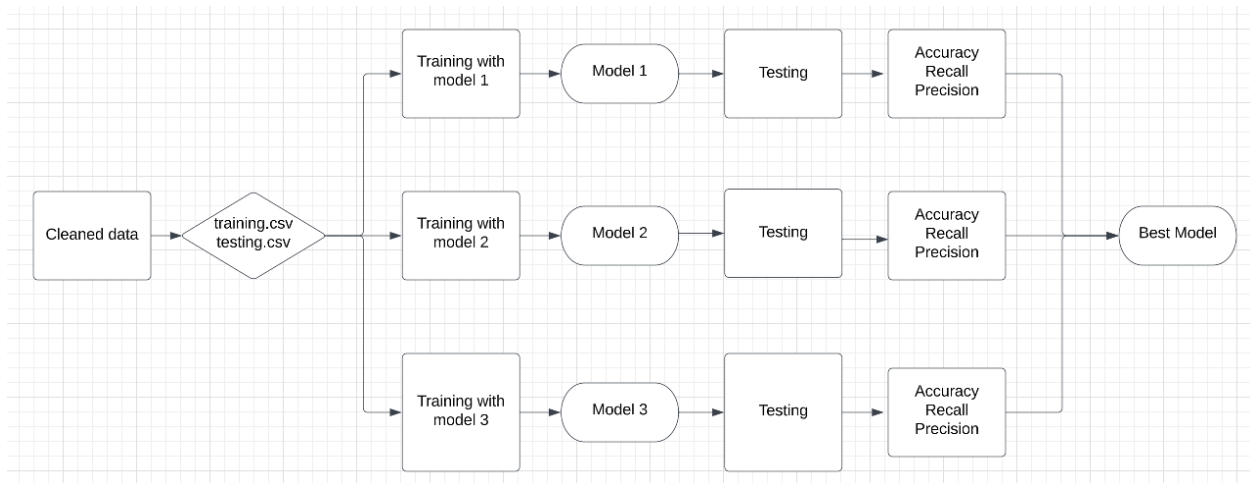


Figure 5.5.1 Machine learning workflow

Table 5.5.1 Job Information

Job	Description
-----	-------------

A	EDA
B	Data cleaning
C1	Train kNN Model
C2	Train XGBoost Model
C3	Train Random Forest Model
D	Model Assessment

5.5.2 Detailed process

First we use the **scp** command to upload our dataset and python scripts to EC2 instance. The client machine works as an NFS client and will mount the folder shared by the submission node.

```

ubuntu@ip-172-31-21-15:~$ cd project
ubuntu@ip-172-31-21-15:~/project$ ls
test
ubuntu@ip-172-31-21-15:~/project$ logout
Connection to ec2-54-91-139-2.compute-1.amazonaws.com closed.
PS C:\Users\86158> scp -i "khszd.pem" heart_2020_cleaned.csv eda.py cleaning.py model1.py model2.py model3.py assess.py ubuntu@ec2-54-91-139-2.compute-1.amazonaws.com:/home/ubuntu/project/
heart_2020_cleaned.csv 100% 24MB 2.6MB/s 00:09
eda.py 100% 1502 5.5KB/s 00:00
cleaning.py 100% 2373 8.8KB/s 00:00
model1.py 100% 1245 5.2KB/s 00:00
model2.py 100% 1226 4.6KB/s 00:00
model3.py 100% 1258 4.8KB/s 00:00
assess.py 100% 70 0.3KB/s 00:00
PS C:\Users\86158> ssh -i "khszd.pem" ubuntu@ec2-54-91-139-2.compute-1.amazonaws.com
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.15.0-1028-aws x

```

```

8 updates can be applied immediately.
6 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

New release '22.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Wed Jan 25 05:58:19 2023 from 202.186.48.59
ubuntu@ip-172-31-21-15:~$ ls
project
ubuntu@ip-172-31-21-15:~$ cd project
ubuntu@ip-172-31-21-15:~/project$ ls
assess.py eda.py model1.py model3.py
cleaning.py heart_2020_cleaned.csv model2.py test
ubuntu@ip-172-31-21-15:~/project$

Processing triggers for systemd (245.4-4ubuntu3.19) ...
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for libc-bin (2.31-0ubuntu9.9) ...
ubuntu@ip-172-31-20-110:~$ sudo mount SubmissionHost:/home/ubuntu/project data/
ubuntu@ip-172-31-20-110:~$ ls
data
ubuntu@ip-172-31-20-110:~$ cd data
ubuntu@ip-172-31-20-110:~/data$ touch test
ubuntu@ip-172-31-20-110:~/data$ ls
test
ubuntu@ip-172-31-20-110:~/data$ ls
assess.py eda.py model1.py model3.py
cleaning.py heart_2020_cleaned.csv model2.py test
ubuntu@ip-172-31-20-110:~/data$ |

```

Next we need to do some essential changes to python files. We add the code `#!/usr/bin/env python3` at the first line to each python file. And change the path of the dataset to `/home/ubuntu/data/heart_2020_cleaned.csv`.

```
#!/usr/bin/env python3
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
data = pd.read_csv("/home/ubuntu/data/heart_2020_cleaned.csv")
```

Next we create a dag input file to specify the jobs in the workflow, and use the Parent Child command to specify the dependencies within the dag.

We create submission files for each job. Below is an example of a submission file.

```
ubuntu@ip-172-31-21-15:~/project$ cat cleaning.sub
executable      = cleaning.py
log             = cleaning.log
output          = cleaning.out
error           = cleaning.err
should_transfer_files = Yes
when_to_transfer_output = ON_EXIT
queue
```

Next we can submit the dag file using the **condor_submit_dag** command.

```
Job A eda.sub
Job B cleaning.sub
Job C1 model1.sub
Job C2 model2.sub
Job C3 model3.sub
Job D assess.sub

Parent A Child B
Parent B Child C1 C2 C3
Parent C1 C2 C3 Child D
~
~
~
~
~
~
~
~
~
~
"workflow.dag" 10L, 170C
```

```
ubuntu@ip-172-31-21-15: ~/project$ vim workflow.dag
ubuntu@ip-172-31-21-15:~/project$ condor_submit_dag workflow.dag

-----
-----
File for submitting this DAG to HTCondor                : workflow.dag.condor.sub
Log of DAGMan debugging messages                        : workflow.dag.dagman.out
Log of HTCondor library output                          : workflow.dag.lib.out
Log of HTCondor library error messages                  : workflow.dag.lib.err
Log of the life of condor_dagman itself                  : workflow.dag.dagman.log

Submitting job(s).
1 job(s) submitted to cluster 33.
-----
-----
```

To see the status of jobs in the job queues , run the **condor_q** command. Can see the jobs are running.

```
ubuntu@ip-172-31-21-15:~/project$ condor_q

-- Schedd: ip-172-31-21-15.ec2.internal : <172.31.21.15:9618?... @ 01/25/23 14:31:01
OWNER    BATCH_NAME    SUBMITTED    DONE    RUN    IDLE    H
ubuntu   ID: 27          1/25 13:36    -       -       -
ubuntu   workflow.d       1/25 14:28    -       2       -

Total for query: 3 jobs; 0 completed, 0 removed, 0 idle, 2 running, 1 held, 0 suspended
Total for ubuntu: 3 jobs; 0 completed, 0 removed, 0 idle, 2 running, 1 held, 0 suspended
Total for all users: 3 jobs; 0 completed, 0 removed, 0 idle, 2 running, 1 held, 0 suspended
```

The following files will be generated after finishing the job.

```
workflow.dag
workflow.dag.condor.sub
workflow.dag.dagman.log
d.csv workflow.dag.dagman.out
workflow.dag.lib.err
workflow.dag.lib.out
workflow.dag.metrics
workflow.dag.nodes.log
```

As we can see from the metrics file, total time cost is 236.347 seconds.

```
ubuntu@ip-172-31-21-15:~/project$ cat workflow.dag.metrics
{
  "client": "condor_dagman",
  "version": "9.12.0",
  "planner": "",
  "planner_version": "",
  "type": "metrics",
  "wf_uuid": "",
  "root_wf_uuid": "",
  "start_time": 1674656930.845,
  "end_time": 1674657167.191,
  "duration": 236.347,
  "exitcode": 0,
  "dagman_id": "33",
  "parent_dagman_id": "",
  "rescue_dag_number": 0,
  "jobs": 6,
  "jobs_failed": 0,
  "jobs_succeeded": 6,
  "dag_jobs": 0,
  "dag_jobs_failed": 0,
  "dag_jobs_succeeded": 0,
  "total_jobs": 6,
  "total_jobs_run": 6,
}
```

5.6 Challenges

```
td_503_983a>
...
007 (082.000.000) 2023-01-26 02:30:43 Shadow exception!
Error from slot1@ip-172-31-26-158.ec2.internal: Failed to execute '/var/lib/condor/execute/dir_1010/condor_exec.exe': (errno=2: 'No such file or directory')
0 - Run Bytes Sent By Job
845 - Run Bytes Received By Job
...
012 (082.000.000) 2023-01-26 02:30:43 Job was held.
Error from slot1@ip-172-31-26-158.ec2.internal: Failed to execute '/var/lib/condor/execute/dir_1010/condor_exec.exe': (errno=2: 'No such file or directory')
Code 6 Subcode 2
...
```

```
ubuntu@ip-172-31-78-160:~/Script$ cat code1_errors.txt
```

Traceback (most recent call last):

File "code1.py", line 1, in <module>

import numpy as np

ModuleNotFoundError: No module named 'numpy'

We faced lots of challenges during this project work because none of our group members has a CS background. We are unfamiliar with the linux system and command. First we split the

original python file into 6 parts. But when we tried to submit the job via DAGMan, error messages appeared continuously. So we have to check with each single file. The first error is because we installed many versions of python and didn't specify the environment we wanted to use. Some errors are due to the inconsistency of python libraries. We need to upgrade them accordingly. Another problem is that new files will be generated to the working directory after running the python code, however, it shows permission denied over time. We took a long time to find the solution that we must change directory permissions using the chmod command before submitting the job.

6. Results and Discussions

This gives us run times of about 7 minutes on a single machine and 236 seconds on AWS EC2 instances. Algorithm KNN, XGBoost and Random Forest all have good performance, with a performance of 90.84%, 91.36% and 91.07 respectively. Among them, XGBoost has the best performance. Because XGBoost algorithm has good processing speed and accuracy for low and medium dimensional data. In addition, XGBoost performs second-order Taylor expansion on the loss function to increase the accuracy of the model. XGBoost adds regular terms to the objective function. It also offers greater flexibility. Since the actual sample of patients with heart disease was small, we used SMOTE to reduce the variance.

Model parallelism places different subnets of the model on different devices and implements the forward method accordingly to move intermediate output between devices. Because parts of the model can only run on any single device, a group of devices can work together to serve a larger model.

Table 6.1 Comparison of time spent on sequential and parallel training models

Model Training Workflow	Mode	Elapsed Time(seconds)	
		Sequential	HTCondor
KNN	Parallel(DAGman)	420	236.347

XGBoost			
Random Forest			

Table 6.2 The training results of the model

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
KNeighborsClassifier	90.84	90.83	90.84	90.83
XGBClassifier	91.36	91.35	91.34	91.36
RandomForestClassifier	91.07	91.05	91.03	91.07

7. Conclusion and future works

In conclusion, a heart disease prediction system using parallel and distributed workflows can be a powerful tool for identifying individuals at risk of heart disease. By leveraging parallel and distributed computing technologies such as HTCondor, the system can be optimized to handle large amounts of data and perform complex calculations quickly and efficiently. In our project, the running time was reduced from 420s for a single machine to 236.347s for HTCondor, the efficiency increased by 43.73%, and the efficiency and accuracy of precision operation were both improved.

Limitation:

- 1) Limitations of time and resources: Due to limited time and resources, we only used DAGman for distributed training. And We lack a more detailed design of a distributed system.
- 2) Lack of performance verification from different angles: We only compared the time of distributed training in the same environment, lacking different setup and validation from the perspective of stability and performance.

Future work:

- 1) Incorporating more data sources: The system could be expanded to include more data sources, such as genomic data, which could provide additional information for predicting the risk of heart disease.
- 2) Improving the model: The predictive model could be further improved by experimenting with different algorithms, architectures, and techniques such as deep learning, and ensembles methods.
- 3) Integration with other systems and platforms: The system could be integrated with other systems and platforms to enable real-time monitoring and analysis of patients' health status.
- 4) Make an app that helps patients predict heart disease.

8.References

- Ramezani, F., Lu, J., Taheri, J., & Zomaya, A. Y. (2017). A Multi-Objective Load Balancing System for Cloud Environments. *The Computer Journal*.
<https://doi.org/10.1093/comjnl/bxw109>
- Grzeszczyk, M. K. (2018, December). Optimization of Machine Learning Process Using Parallel Computing. ResearchGate; Wydawnictwo Naukowe Gabriel Borowski (WNGB).
https://www.researchgate.net/publication/329700652_Optimization_of_Machine_Learning_Process_Using_Parallel_Computing
- Jung-Lok;Jin, Y. (2020). A Workflow Execution System for Analyzing Large-scale Astronomy Data on Virtualized Computing Environments. *International Journal of Contents*, 16(4), 16–25. <https://doi.org/10.5392/IJoC.2020.16.4.016>
- Mohan, S., Thirumalai, C., & Srivastava, G. (2019). Effective Heart Disease Prediction Using Hybrid Machine Learning Techniques. *IEEE Access*, 7, 81542–81554.
<https://doi.org/10.1109/access.2019.2923707>