Project Report

On
Stock Market Analysis Using SLURM

Submitted
In partial fulfilment
For the award of the Degree of

PG-Diploma in High Performance Computing System Administration.

(C-DAC, ACTS (Pune))

Guided By:

Mr. Manish Kumar Abhishek

Submitted By:

Chaitali Narayane(240840127032)

Neha Sawant(240840127015)

Pranita Waghmare(240840127036)

Vaishali Karale(240840127019)

Centre for Development of Advanced Computing

(C-DAC), ACTS (Pune- 411008)

Acknowledgement

This is to acknowledge our indebtedness to our Project Guide, **Mr. Manish Kumar Abhishek,** C-DAC ACTS, Pune for her constant guidance and helpful suggestion for preparing this project **Stock Market Analysis Using SLURM.** We express our deep gratitude towards her for inspiration, personal involvement, constructive criticism that she provided us along with technical guidance during the course of this project.

We take this opportunity to thank Head of the department **Mr. Gaur Sunder** for providing us such a great infrastructure and environment for our overall development.

We express sincere thanks to **Mrs. Namrata Ailawar**, Process Owner, for their kind cooperation and extendible support towards the completion of our project.

It is our great pleasure in expressing sincere and deep gratitude towards **Mrs. Risha P R (Program Head)** and **Ms. Pratiksha Gacche** (Course Coordinator, PG-HPCSA) for their valuable guidance and constant support throughout this work and help to pursue additional studies.

Also, our warm thanks to **C-DAC ACTS Pune**, which provided us this opportunity to carry out, this prestigious Project and enhance our learning in various technical fields.

Chaitali Narayane (240840127032)

Neha Sawant (240840127015)

Pranita Waghmare (240840127036)

Vaishali Karale (240840127019)

ABSTRACT

Stock market data analysis is essential for investors and analysts to identify market trends and make informed decisions. However, processing large volumes of real-time and historical stock data requires significant computational power. This project leverages High-Performance Computing (HPC) with SLURM (Simple Linux Utility for Resource Management) to efficiently parallelize stock market analysis tasks.

The system retrieves stock data from financial APIs, such as Yahoo Finance, and computes key indicators like Moving Averages (MA), Relative Strength Index (RSI), and Volatility. Using SLURM job scheduling, workloads are distributed across multiple compute nodes, significantly improving processing speed and scalability.

A controller node manages job scheduling, while compute nodes process stock data in parallel. The final results are stored in structured reports for further analysis. This project demonstrates how HPC and parallel computing can enhance financial data analysis, making it valuable for investment research, algorithmic trading, and real-time market insights.

Table of Contents

Chapter 1
Introduction

1.1 Introduction

Financial markets produce a huge amount of data every second, and it can be very difficult to keep up with all of it. Investors, analysts, and researchers need to understand what is happening right now as well as what has happened in the past. However, using traditional methods to analyze this data can be very slow and inefficient. The "Stock Market Analysis Using SLURM" project is designed to solve this problem by using high-performance computing (HPC) to process and analyze large amounts of stock market data quickly and efficiently.

This project makes use of a tool called SLURM, which stands for Simple Linux Utility for Resource Management. SLURM is a job scheduler that helps manage and distribute tasks across a group of computers, known as a cluster. A computer cluster is a set of connected computers that work together so that complex tasks can be completed much faster than with just one computer. By using SLURM, the project can break down the work needed to analyze the stock market into smaller parts and assign each part to a different computer. This means that many tasks are carried out at the same time, greatly speeding up the overall process.

The project gathers stock market data from reliable sources like Yahoo Finance. It uses both real-time data, which shows what is happening right now, and historical data, which provides information on past trends. With this data, the project calculates important financial indicators such as Moving Averages, the Relative Strength Index (RSI), and Volatility. Moving Averages help smooth out the daily ups and downs in stock prices, making trends easier to spot. The RSI is a tool that can indicate if a stock is being bought or sold too quickly, while Volatility shows how much a stock’s price goes up and down over time. These calculations help traders and analysts to quickly understand market trends and make better decisions.

In the setup, one computer acts as the controller, managing the tasks and collecting the results from other computers called compute nodes. Each compute node works on a small piece of the data, and because they work in parallel, the analysis is completed much faster than if only one computer were used.

Overall, the "Stock Market Analysis Using SLURM" project demonstrates how modern computing methods can be used to handle and analyze the large volumes of data generated by financial markets. By spreading the work across many computers, this project provides a fast and efficient way to monitor and analyze stock market trends, which helps investors and analysts make smarter, more informed decisions.

1.2 Objective

The primary objective of the "Stock Market Analysis Using SLURM" project is to develop a fast, efficient, and scalable system for analyzing large volumes of stock market data using high-performance computing (HPC) techniques. With the financial markets generating immense amounts of data every second, traditional single-computer methods often struggle to keep pace, making it difficult for investors, traders, and analysts to make timely and informed decisions.

This project aims to address these challenges by leveraging SLURM—a tool that distributes computational tasks across multiple computers in a cluster.

Accelerate Data Processing: By using SLURM to divide the workload among several compute nodes, the system will process both real-time and historical stock data much faster than traditional methods. This distributed approach allows for parallel computation, significantly reducing the overall processing time.

Compute Key Financial Indicators: The system will automatically calculate important metrics such as Moving Averages, Relative Strength Index (RSI), and Volatility. These indicators are crucial for understanding market trends, assessing stock performance, and identifying potential trading opportunities.

Ensure Scalability and Efficiency: As stock market data continues to grow, the solution must be capable of handling increased workloads without performance degradation. The project's design ensures that the system can scale up by simply adding more compute nodes, thereby maintaining efficiency even as data volumes expand.

Enhance Decision-Making: By providing timely and accurate analyses of market trends, the project empowers users to make well-informed decisions quickly. The actionable insights derived from the computed indicators will support better investment strategies and trading decisions.

Robust Parallel Processing: Distribute tasks efficiently across multiple compute nodes to reduce processing time. This involves not only splitting the workload but also managing dependencies and synchronizing results from different nodes.

User-Friendly Reporting and Visualization: Develop modules to aggregate and present the processed data in a clear, structured report. Integrate visualization tools that can help users quickly interpret trends and make informed decisions.

Foundation for Algorithmic Trading: Create an adaptable platform that can be extended for use in algorithmic trading systems. By providing quick and accurate data analysis, the project sets the stage for further research and development in automated trading strategies.

Enhanced Accuracy and Reliability: Implement rigorous data validation and error-checking mechanisms during data retrieval and computation. This ensures that the indicators and analysis are both accurate and reliable for making informed decisions.

Improved Scalability: Design the system to effortlessly scale with increasing data volumes and the addition of more compute nodes. This ensures that the solution remains effective as the number of stocks and the frequency of data updates grow over time.

Optimized Resource Utilization: Leverage SLURM's efficient job scheduling to minimize idle resources and maximize throughput. This reduces computational costs while improving overall performance.

Overall, this project seeks to transform the way stock market data is processed and analyzed by harnessing the power of HPC and SLURM, ultimately offering a robust and scalable tool that meets the dynamic needs of modern financial markets.

Chapter 2
LITERATURE REVIEW

2.1. High-Performance Computing (HPC) in Financial Analysis

Numerous studies have demonstrated the significant impact of HPC on financial analytics. Researchers have shown that HPC techniques can reduce computational time and increase the accuracy of models used for risk management, portfolio optimization, and real-time trading strategies. For example, studies by Chan et al. (2019) illustrate that leveraging multiple processors can effectively handle complex financial algorithms, thereby improving responsiveness in volatile markets. This body of work provides a solid foundation for integrating HPC techniques into stock market analysis.

2.2. Parallel and Distributed Processing in Finance

Parallel computing and distributed processing have become essential for managing the massive volumes of data generated in financial markets. Literature by Zhang et al. (2018) highlights that dividing tasks among several processors or compute nodes can lead to significant performance gains, especially when processing both historical and real-time market data concurrently. The ability to run computations in parallel ensures that time-sensitive financial indicators are updated quickly, a crucial factor for algorithmic trading and decision-making.

2.3. The Role of SLURM in Resource Management

SLURM (Simple Linux Utility for Resource Management) has been widely adopted in HPC environments due to its efficient job scheduling and resource allocation capabilities. Research conducted at major computing centers, such as the National Energy Research Scientific Computing Center (NERSC), confirms that SLURM can manage large clusters effectively, distributing workloads and minimizing computational bottlenecks. These studies emphasize SLURM’s flexibility and robustness, making it an ideal choice for projects that require simultaneous execution of multiple tasks, such as the computation of financial indicators.

2.4. Real-Time Data Integration in Financial Markets

The integration of real-time data with historical datasets is critical for accurate market analysis. Literature in this area, including works by Gupta & Sinha (2021), indicates that real-time data feeds significantly enhance the precision of trend analysis and the timely detection of market movements. This research underlines the importance of systems that can continuously ingest and process data from financial APIs, ensuring that analyses reflect the latest market conditions.

2.5. Advanced Financial Metric Computation

Calculating technical indicators like Moving Averages, Relative Strength Index (RSI), and Volatility is a common requirement in stock market analysis. Several studies have explored methods to improve the computation of these metrics through parallel processing, reducing the lag between data collection and actionable insights. Research by Lee and Martin (2020) demonstrates that advanced computational techniques can provide more reliable and faster calculations of these indicators, ultimately aiding better investment decisions.

2.6. Scalability and Future Directions

As data volumes in financial markets continue to grow, scalability remains a key concern. The literature stresses the importance of designing systems that can handle increasing workloads without performance degradation. Recent studies suggest that using scalable HPC architectures, combined with dynamic resource management tools like SLURM, can future-proof financial analytics systems. This research supports the development of robust systems capable of adapting to evolving market demands and the integration of additional data sources over time.

Chapter 3

Requirements

3.1. Hardware Requirements

1 Controller Node (For job submission & monitoring)
2 Compute Nodes (For parallel data fetching & analysis)

3.2. Software & Tools Required

SLURM (Job Scheduler)
Python (for stock data fetching & analysis)
Yahoo Finance API (yfinance) (for stock data)

Chapter 4
Methodology and Techniques

4.1. Data Collection and Preprocessing

Data Retrieval:

The project begins by gathering both real-time and historical stock market data from reliable financial APIs such as Yahoo Finance. This involves setting up automated routines (e.g., scheduled Python scripts) to query the APIs at regular intervals, ensuring that the latest data is continuously fetched. Data types include stock prices, trading volumes, and other market-related indicators.

Data Cleaning:

Once the data is collected, it undergoes a thorough cleaning process.

This step involves:

1. **Duplicate Removal:** Duplicate records are identified and removed.

2. **Handling Missing Values:** Missing entries are addressed either by interpolation, forward/backward filling, or complete exclusion based on the context.

3. **Data Formatting:** Date-time formats are standardized, and numerical data is converted to appropriate types for computation.

4. **Normalization:** Where needed, data is normalized to ensure consistency, which is especially important when combining datasets from multiple sources.

4.2. Data Storage:

Cleaned data is then stored in a structured format. Depending on the system's design.

This could involve:

1. **Flat Files:** For smaller datasets, CSV file are used.

2. **Databases:** For larger, more dynamic datasets, relational databases like MySQL or PostgreSQL, or NoSQL databases such as Mariadb, may be implemented.

3. **Distributed File Systems:** In environments with very large volumes of data, distributed storage solutions like HDFS can be employed to facilitate rapid access by multiple compute nodes.

4.3. Financial Indicator Computation

Algorithm Development:

The core of the project involves computing key financial indicators.

Custom algorithms are developed to calculate:

Moving Averages (MA): Smoothing out short-term fluctuations to reveal longer-term trends.

Relative Strength Index (RSI): Measuring the speed and change of price movements to identify overbought or oversold conditions.

Modular Code Structure:

The computation routines are designed in a modular fashion. Each indicator is encapsulated within its own function or module, which makes it easier to maintain, test, and extend the system with additional metrics (such as Exponential Moving Averages) in the future.

4.4. Parallel Processing and Job Scheduling with SLURM

Cluster Setup:

An HPC cluster is established, comprising one controller node and multiple compute nodes. The controller node is responsible for managing job submissions and aggregating results, while compute nodes execute the heavy computational tasks.

SLURM Configuration and Usage:

SLURM is configured to manage the cluster’s workload.

Key configurations include:

Job Scripts: Custom scripts are written to define job parameters (e.g., number of compute nodes, processors per node, and memory allocation).

Task Distribution: The stock market dataset is partitioned into smaller segments, with each segment assigned to a different compute node. This ensures that multiple segments are processed concurrently.

Job Monitoring: SLURM monitors job status and resource utilization, ensuring efficient task scheduling and timely completion of jobs.

Parallel Execution:

The project leverages SLURM to schedule these tasks simultaneously. This parallel execution greatly reduces the overall processing time, making it feasible to compute real-time financial indicators even for large datasets.

4.5. Result Aggregation and Reporting

Data Aggregation:

After individual compute nodes complete their assigned tasks, the results (computed financial indicators) are collected by the controller node. Data from different nodes are merged into a single, coherent dataset for further analysis.

Visualization and Reporting:

The final step involves creating clear and actionable reports. Visualization libraries such as Python's matplotlib, pandas, numpy are used to generate:

Dashboards that provide an interactive view of the computed indicators, aiding investors and analysts in making informed decisions.

4.6. Error Handling and Validation

Robust Error Checking:

Throughout the process, error-handling mechanisms are implemented to manage issues like missing data, corrupted data files, or node failures. For example, try-except blocks in code and automated alerts ensure that errors are caught and addressed promptly.

Result Validation:

The computed indicators are validated against known benchmarks or historical events. This ensures that the algorithms are working as intended and that the insights generated are reliable.

4.7. Scalability and Performance Optimization

Scalability Testing:

The system is stress-tested with various data sizes to ensure that it can handle increasing volumes of stock market data. These tests help identify performance bottlenecks.

Performance Tuning:

SLURM configurations and computational routines are optimized to enhance resource utilization. Techniques such as load balancing and fine-tuning job parameters are employed to minimize overhead and improve overall throughput.

This comprehensive methodology, integrating detailed data collection, modular financial computations, robust parallel processing with SLURM, and thorough error handling, ensures a scalable, efficient, and reliable system for stock market analysis.

4.8. Techniques Employed

High-Performance Computing (HPC):

HPC leverages multiple processors and compute nodes to process large datasets quickly. In this project, HPC enables the simultaneous execution of complex financial computations, reducing the time required for analysis and ensuring that both historical and real-time data are processed efficiently.

Parallel Programming:

Parallel programming techniques are used to divide the computational workload among several compute nodes. By employing concurrent execution using libraries such as Python's.
The project can compute multiple financial indicators (like Moving Averages, RSI, and Volatility) at the same time, greatly speeding up processing times.

SLURM Job Scheduling:

SLURM (Simple Linux Utility for Resource Management) is central to managing the cluster's resources. It schedules and distributes jobs across the available compute nodes, ensuring that resources are optimally used and that the system avoids bottlenecks. Custom SLURM job scripts specify resource requirements (e.g., number of nodes, CPU cores, memory) and manage task execution in a parallelized environment.

Data Collection and Cleaning:

Automated data retrieval techniques are implemented using APIs (such as Yahoo Finance) to obtain both real-time and historical stock market data. Once collected, the data undergoes thorough cleaning including reduplication, handling missing values, and standardizing formats—to ensure that the subsequent analysis is based on high-quality and consistent data.

Modular Code Design:

The project is structured using a modular approach, where individual functions or modules are responsible for specific tasks such as data fetching, preprocessing, indicator calculation, and visualization. This separation of concerns simplifies debugging, maintenance, and the integration of new features or additional financial metrics.

Algorithm Optimization:

To compute financial indicators efficiently, the project uses optimized algorithms that leverage vectorized operations available in Python libraries like NumPy and pandas. These optimizations reduce computational overhead and memory usage, ensuring that the calculations are performed as quickly as possible.

Data Visualization:

Effective data visualization is crucial for interpreting the results. Visualization techniques using tools such as matplotlib are employed to create graphs, charts, and interactive dashboards. These visualizations provide clear insights into market trends and support quick decision-making for investors and analysts.

Scalability Testing and Optimization:

Running comprehensive tests to ensure the system can scale with increasing data volumes and optimizing configurations to maintain high performance.

Error Handling and Validation:

The system undergoes rigorous scalability testing to evaluate its performance under different workloads. Based on these tests, performance tuning is carried out adjusting SLURM configurations, optimizing data handling processes, and refining computational algorithms to ensure the system can handle growing data volumes without sacrificing performance.

This detailed methodology and the techniques used provide a comprehensive approach to tackling the challenges of real-time and historical stock market analysis. The integration of HPC, parallel processing, and SLURM job scheduling ensures that the system is both robust and scalable, delivering accurate and timely financial insights.

Chapter 5

Implementation

5.1. Define Project Structure

Project Folder (/shared/stock_project/)

```
/shared/stock_project/
    ├── stocks.txt              # List of stock ticker symbols
    ├── fetch_stock_data.py          # Python script to fetch stock data
    ├── analyze_data.py     # Python script to analyze stock data
    ├── fetch_data.slurm    # SLURM job script for fetching data
    ├── analyze_data.slurm           # SLURM job script for analysis
    ├── data/               # Folder where stock CSV files will be saved
    ├── analysis/           # Folder where analysis results will be saved
    ├── logs/               # SLURM output and error logs
    ├── final_report.csv    # Summary of all analyzed data
```

5.2. Prepare the Stock List

Create a list **of stock ticker symbols** in a text file.

stocks.txt

```
AAPL
MSFT
GOOG
AMZN
```

Each **line** represents a **stock ticker** that will be processed separately.

5.3. Write Python Script to Fetch Stock Data

We will use **Yahoo Finance (yfinance)** to fetch stock market data.

fetch_stock_data.py

```
  import yfinance as yf
  import sys
  import os

  # Get stock ticker from command line argument
  ticker = sys.argv[1]
  output_dir = sys.argv[2]

  # Create directory if it doesn't exist
  os.makedirs(output_dir, exist_ok=True)

  # Fetch historical stock data
  print(f"Fetching data for {ticker}...")
try:
      data = yf.download(ticker, start="2020-01-01", end="2025-01-01") output_file =
      os.path.join(output_dir, f"{ticker}.csv") data.to_csv(output_file)
      print(f"Data for {ticker} saved to {output_file}.")
except Exception as e:
      print(f"Error fetching data for {ticker}: {e}")
```

5.4. Create SLURM Job Script for Data Fetching

fetch_data.slurm

```
#!/bin/bash
#SBATCH --job-name=fetch_stock
#SBATCH --output=/shared/stock_project/logs/fetch_%A_%a.out
#SBATCH --error=/shared/stock_project/logs/fetch_%A_%a.err
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --time=00:30:00
#SBATCH --mem=2GB

# Load Python module (if required)
module load python/3.9

# Read stock symbols into an array
stocks=($(cat /shared/stock_project/stocks.txt))

# Get the stock ticker for this job array index
ticker=${stocks[$SLURM_ARRAY_TASK_ID]}

# Run the Python script to fetch data
 python3 /shared/stock_project/fetch_stock_data.py $ticker /shared/stock_project/data/
```

Submit the SLURM Job to Fetch Data

```
sbatch --array=0-9 /shared/stock_project/fetch_data.slurm
```

Expected Output:

Stock data files (AAPL.csv, MSFT.csv, etc.) will be saved in **/shared/stock_project/data/**.

5.5 . Write Python Script for Analysis

analyze_data.py

```
import pandas as pd
import sys
import os

# Get stock ticker and directories from command line
ticker = sys.argv[1]
data_dir = sys.argv[2]
output_dir = sys.argv[3]

 # Load stock data
file_path = os.path.join(data_dir, f"{ticker}.csv")
if not os.path.exists(file_path):
        print(f"No data found for {ticker}")
        sys.exit()

data = pd.read_csv(file_path)

# Calculate Moving Averages (50-day & 200-day)
data["50_MA"] = data["Close"].rolling(window=50).mean()
data["200_MA"] = data["Close"].rolling(window=200).mean()

# Calculate RSI (Relative Strength Index)
delta = data["Close"].diff()
gain = (delta.where(delta > 0, 0)).rolling(window=14).mean()
loss = (-delta.where(delta < 0, 0)).rolling(window=14).mean()
rs = gain / loss
data["RSI"] = 100 - (100 / (1 + rs))

# Save analysis results
os.makedirs(output_dir, exist_ok=True)
data.to_csv(os.path.join(output_dir, f"{ticker}_analysis.csv"),
index=False)
print(f"Analysis completed for {ticker}.")
```

5.6. Create SLURM Job Script for Analysis

analyze_data.slurm

```
#!/bin/bash
#SBATCH --job-name=analyze_stock
#SBATCH --output=/shared/stock_project/logs/analyze_%A_%a.out
#SBATCH --error=/shared/stock_project/logs/analyze_%A_%a.err
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --time=00:30:00
#SBATCH --mem=2GB

module load python/3.9
stocks=($(cat /shared/stock_project/stocks.txt))
ticker=${stocks[$SLURM_ARRAY_TASK_ID]}

python3 /shared/stock_project/analyze_data.py $ticker /shared/stock_project/data/
/shared/stock_project/analysis/
```

Submit the SLURM Job for Analysis

```
sbatch --array=0-9 /shared/stock_project/analyze_data.slurm
```

Expected Output:

Analysis files (AAPL_analysis.csv, MSFT_analysis.csv, etc.) will be saved in

/shared/stock_project/analysis/.

5.7. Summarize the Results

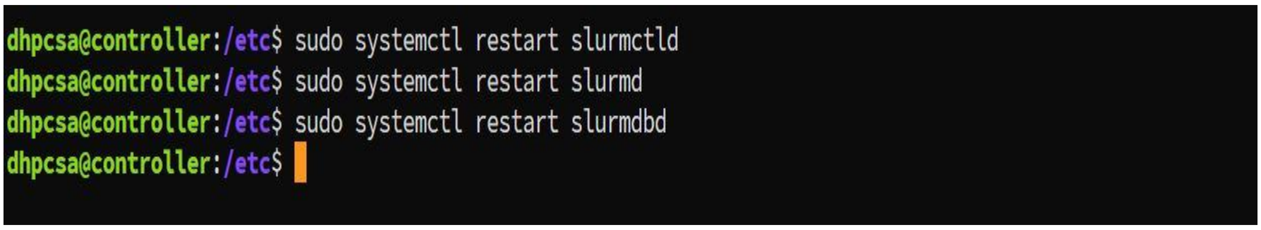
Generate_report.py

```
import pandas as pd
import os

analysis_dir = "/shared/stock_project/analysis/"
output_file = "/shared/stock_project/final_report.csv"

files = [f for f in os.listdir(analysis_dir) if f.endswith("_analysis.csv")]

summary = []

for file in files:
        df = pd.read_csv(os.path.join(analysis_dir, file))
        ticker = file.split("_")[0]
        latest_row = df.iloc[-1][["Close", "50_MA", "200_MA", "RSI"]]
        latest_row["Ticker"] = ticker
        summary.append(latest_row)

summary_df = pd.DataFrame(summary)
summary_df.to_csv(output_file, index=False)
print("Final report generated!")
```

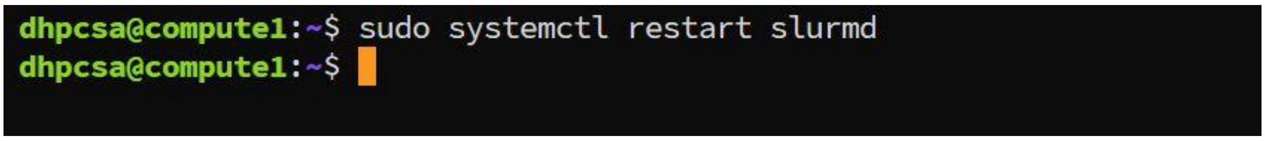
Expected Output:

Final summary report saved as

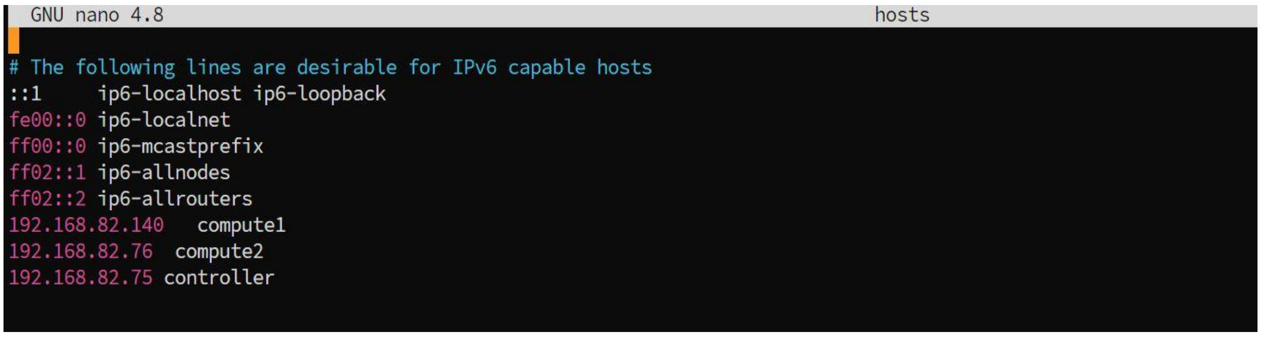
/shared/stock_project/final_report.csv.



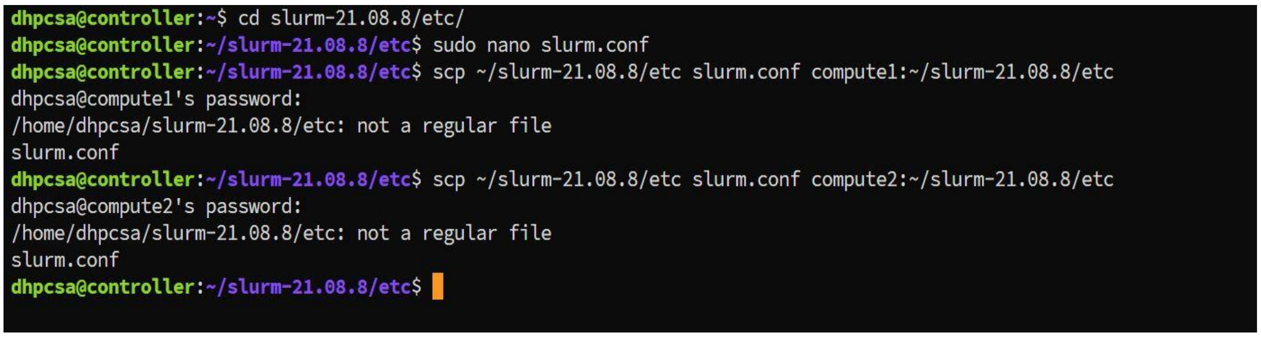
```
dhpcsa@controller:/etc$ sudo systemctl restart slurmctld
dhpcsa@controller:/etc$ sudo systemctl restart slurmd
dhpcsa@controller:/etc$ sudo systemctl restart slurmdbd
dhpcsa@controller:/etc$
```



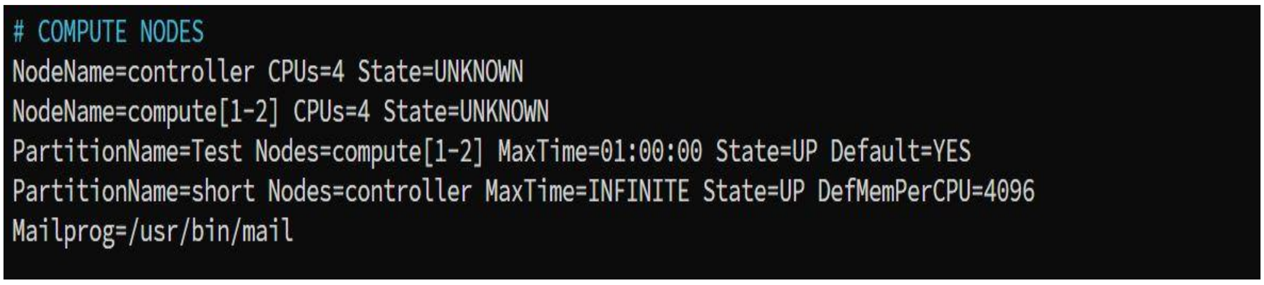
```
dhpcsa@compute1:~$ sudo systemctl restart slurmd
dhpcsa@compute1:~$
```



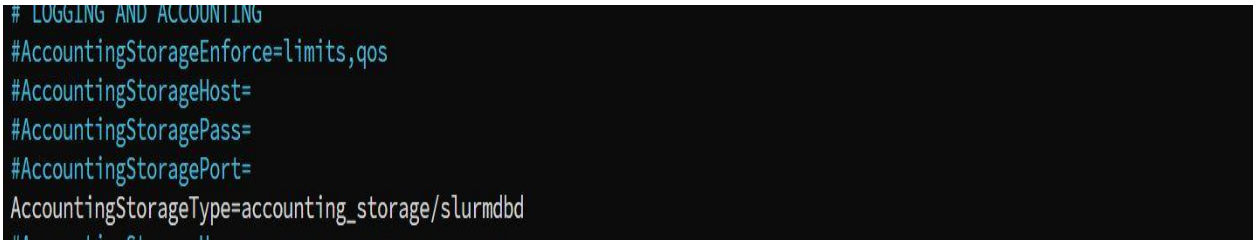
```
  GNU nano 4.8                                                     hosts

# 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
192.168.82.140    compute1
192.168.82.76   compute2
192.168.82.75 controller
```



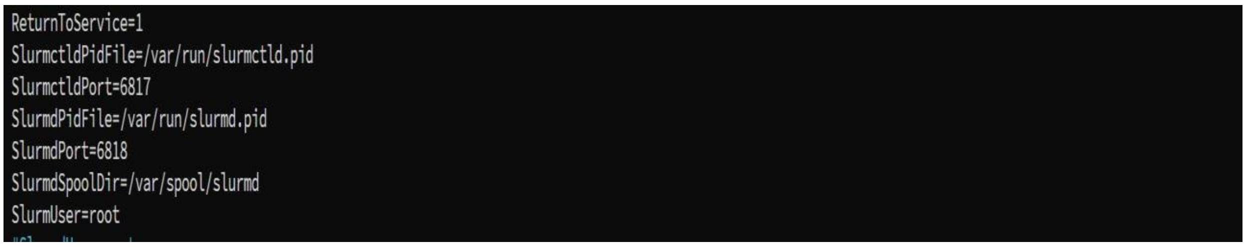
```
dhpcsa@controller:~$ cd slurm-21.08.8/etc/
dhpcsa@controller:~/slurm-21.08.8/etc$ sudo nano slurm.conf
dhpcsa@controller:~/slurm-21.08.8/etc$ scp ~/slurm-21.08.8/etc slurm.conf compute1:~/slurm-21.08.8/etc
dhpcsa@compute1's password:
/home/dhpcsa/slurm-21.08.8/etc: not a regular file
slurm.conf
dhpcsa@controller:~/slurm-21.08.8/etc$ scp ~/slurm-21.08.8/etc slurm.conf compute2:~/slurm-21.08.8/etc
dhpcsa@compute2's password:
/home/dhpcsa/slurm-21.08.8/etc: not a regular file
slurm.conf
dhpcsa@controller:~/slurm-21.08.8/etc$
```



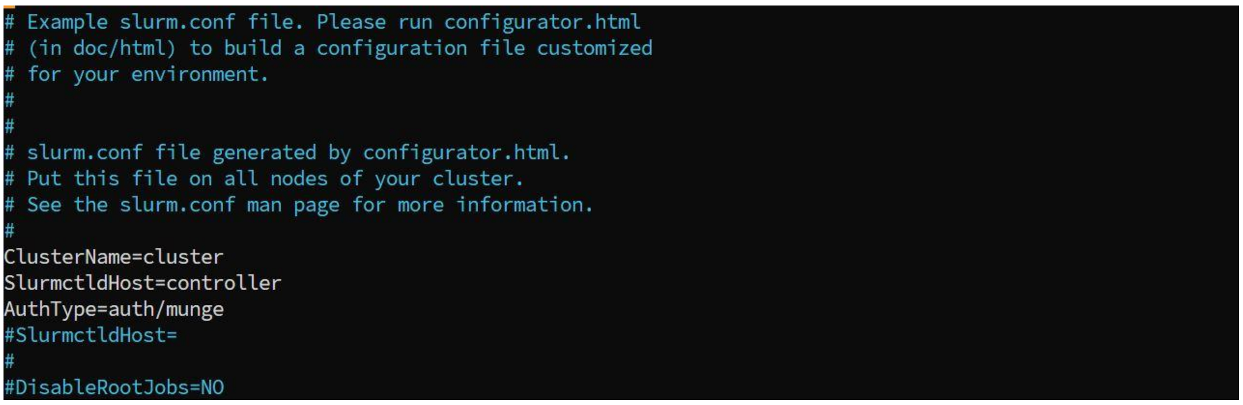
```
# COMPUTE NODES
NodeName=controller CPUs=4 State=UNKNOWN
NodeName=compute[1-2] CPUs=4 State=UNKNOWN
PartitionName=Test Nodes=compute[1-2] MaxTime=01:00:00 State=UP Default=YES
PartitionName=short Nodes=controller MaxTime=INFINITE State=UP DefMemPerCPU=4096
Mailprog=/usr/bin/mail
```



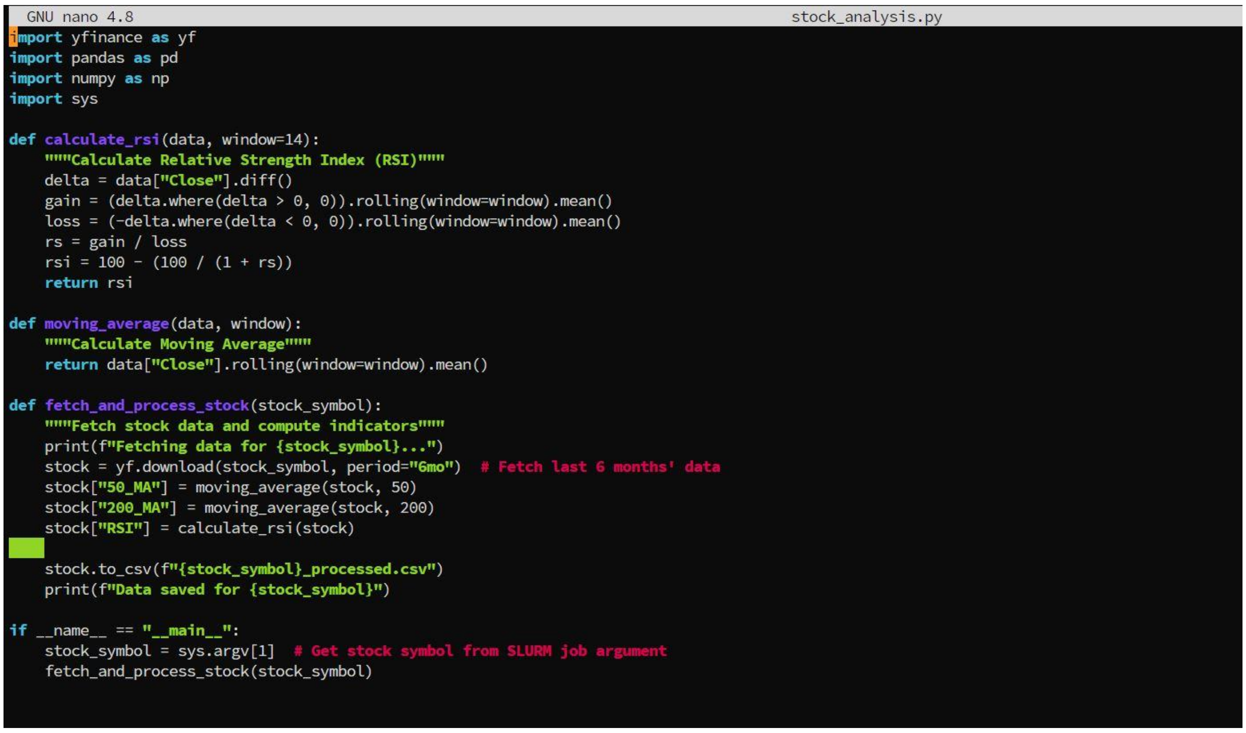
```
# LOGGING AND ACCOUNTING
#AccountingStorageEnforce=limits,qos
#AccountingStorageHost=
#AccountingStoragePass=
#AccountingStoragePort=
AccountingStorageType=accounting_storage/slurmdbd
```



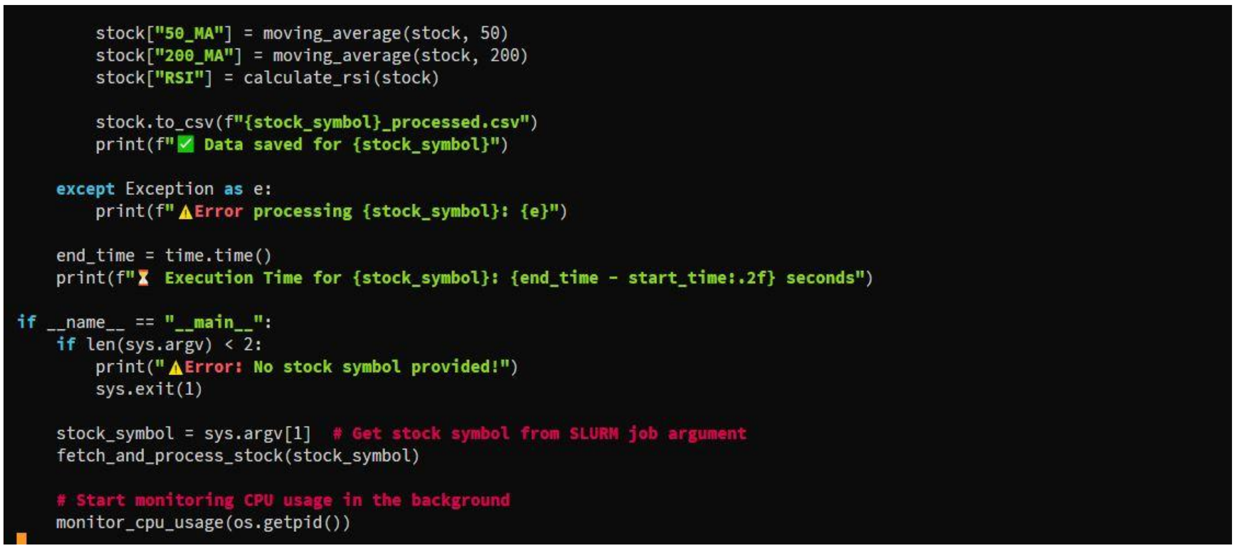
```
ReturnToService=1
SlurmctldPidFile=/var/run/slurmctld.pid
SlurmctldPort=6817
SlurmdPidFile=/var/run/slurmd.pid
SlurmdPort=6818
SlurmdSpoolDir=/var/spool/slurmd
SlurmUser=root
```



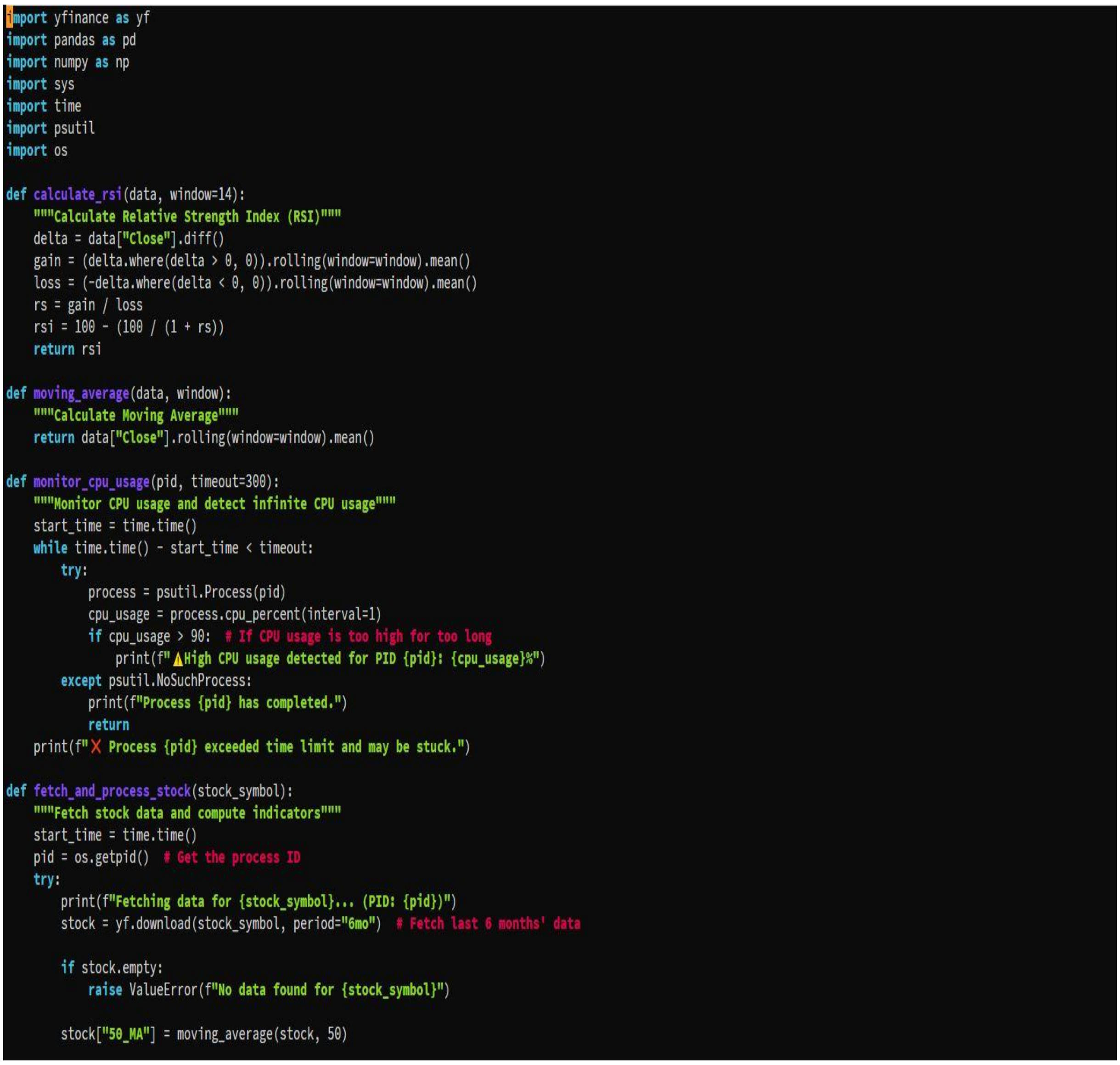
```
# Example slurm.conf file. Please run configurator.html
# (in doc/html) to build a configuration file customized
# for your environment.
#
#
# slurm.conf file generated by configurator.html.
# Put this file on all nodes of your cluster.
# See the slurm.conf man page for more information.
#
ClusterName=cluster
SlurmctldHost=controller
AuthType=auth/munge
#SlurmctldHost=
#
#DisableRootJobs=NO
```



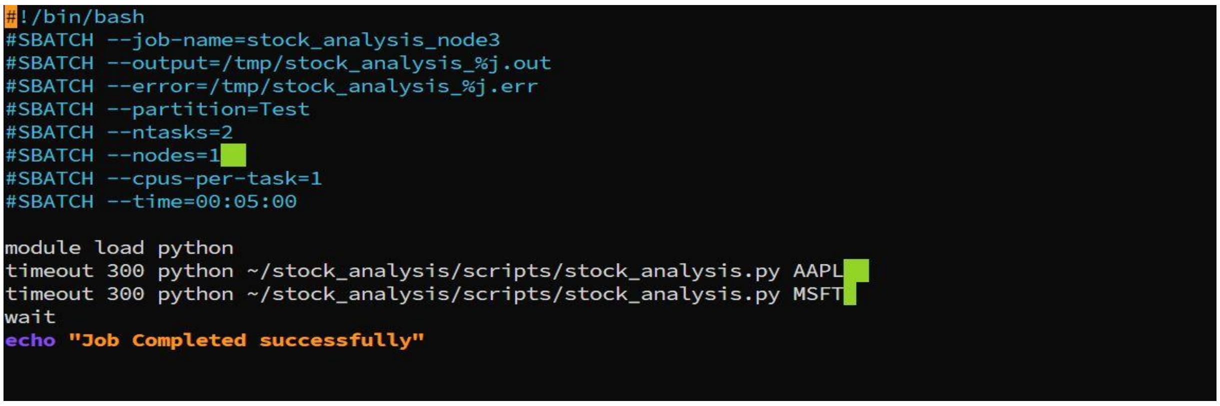
```
  GNU nano 4.8                                                    stock_analysis.py
import yfinance as yf
import pandas as pd
import numpy as np
import sys

def calculate_rsi(data, window=14):
    """Calculate Relative Strength Index (RSI)"""
    delta = data["Close"].diff()
    gain = (delta.where(delta > 0, 0)).rolling(window=window).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=window).mean()
    rs = gain / loss
    rsi = 100 - (100 / (1 + rs))
    return rsi

def moving_average(data, window):
    """Calculate Moving Average"""
    return data["Close"].rolling(window=window).mean()

def fetch_and_process_stock(stock_symbol):
    """Fetch stock data and compute indicators"""
    print(f"Fetching data for {stock_symbol}...")
    stock = yf.download(stock_symbol, period="6mo")  # Fetch last 6 months' data
    stock["50_MA"] = moving_average(stock, 50)
    stock["200_MA"] = moving_average(stock, 200)
    stock["RSI"] = calculate_rsi(stock)

    stock.to_csv(f"{stock_symbol}_processed.csv")
    print(f"Data saved for {stock_symbol}")

if __name__ == "__main__":
    stock_symbol = sys.argv[1]  # Get stock symbol from SLURM job argument
    fetch_and_process_stock(stock_symbol)
```



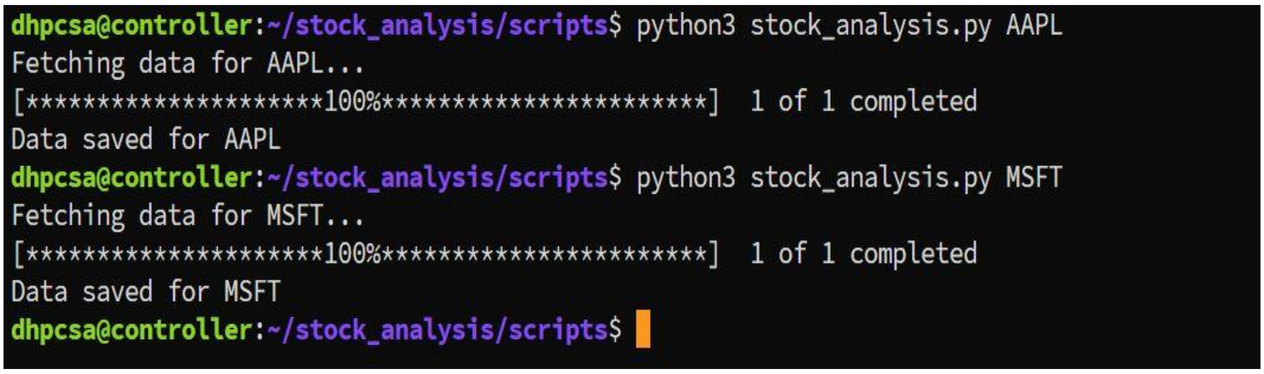
```
        stock["50_MA"] = moving_average(stock, 50)
        stock["200_MA"] = moving_average(stock, 200)
        stock["RSI"] = calculate_rsi(stock)

        stock.to_csv(f"{stock_symbol}_processed.csv")
        print(f"✅ Data saved for {stock_symbol}")

    except Exception as e:
        print(f"⚠Error processing {stock_symbol}: {e}")

    end_time = time.time()
    print(f"⌛ Execution Time for {stock_symbol}: {end_time - start_time:.2f} seconds")

if __name__ == "__main__":
    if len(sys.argv) < 2:
        print("⚠Error: No stock symbol provided!")
        sys.exit(1)

    stock_symbol = sys.argv[1]  # Get stock symbol from SLURM job argument
    fetch_and_process_stock(stock_symbol)

    # Start monitoring CPU usage in the background
    monitor_cpu_usage(os.getpid())
```



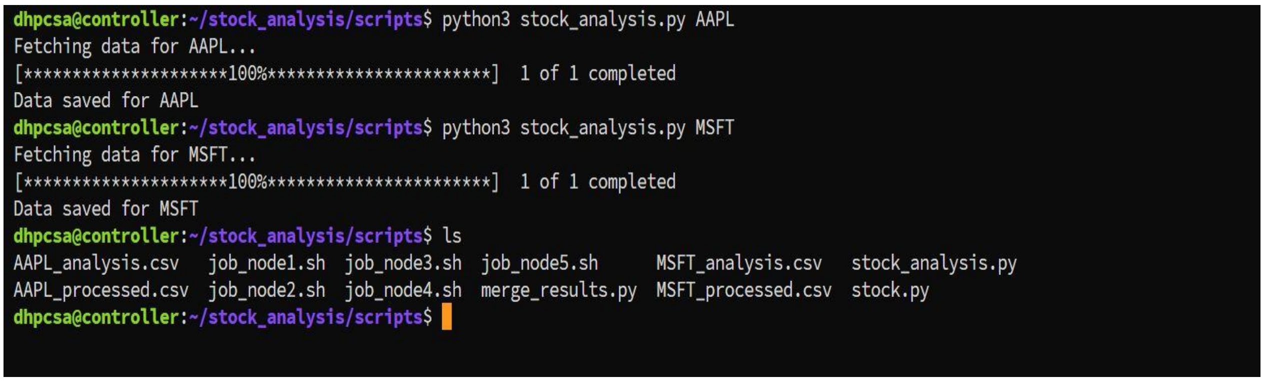
```python
import yfinance as yf
import pandas as pd
import numpy as np
import sys
import time
import psutil
import os

def calculate_rsi(data, window=14):
    """Calculate Relative Strength Index (RSI)"""
    delta = data["Close"].diff()
    gain = (delta.where(delta > 0, 0)).rolling(window=window).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=window).mean()
    rs = gain / loss
    rsi = 100 - (100 / (1 + rs))
    return rsi

def moving_average(data, window):
    """Calculate Moving Average"""
    return data["Close"].rolling(window=window).mean()

def monitor_cpu_usage(pid, timeout=300):
    """Monitor CPU usage and detect infinite CPU usage"""
    start_time = time.time()
    while time.time() - start_time < timeout:
        try:
            process = psutil.Process(pid)
            cpu_usage = process.cpu_percent(interval=1)
            if cpu_usage > 90:  # If CPU usage is too high for too long
                print(f"⚠High CPU usage detected for PID {pid}: {cpu_usage}%")
        except psutil.NoSuchProcess:
            print(f"Process {pid} has completed.")
            return
    print(f"❌ Process {pid} exceeded time limit and may be stuck.")

def fetch_and_process_stock(stock_symbol):
    """Fetch stock data and compute indicators"""
    start_time = time.time()
    pid = os.getpid()  # Get the process ID
    try:
        print(f"Fetching data for {stock_symbol}... (PID: {pid})")
        stock = yf.download(stock_symbol, period="6mo")  # Fetch last 6 months' data

        if stock.empty:
            raise ValueError(f"No data found for {stock_symbol}")

        stock["50_MA"] = moving_average(stock, 50)
```



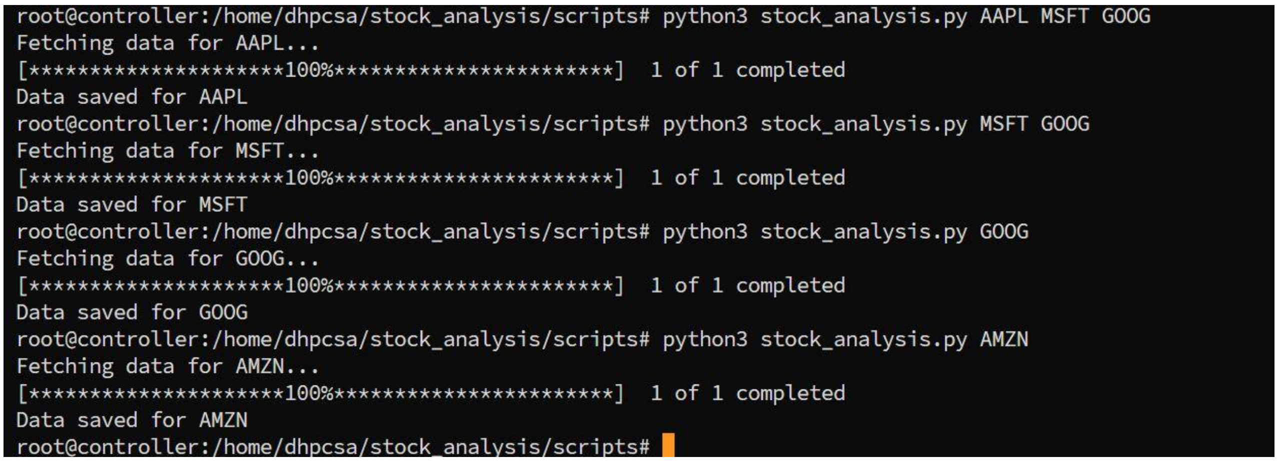
```bash
#!/bin/bash
#SBATCH --job-name=stock_analysis_node3
#SBATCH --output=/tmp/stock_analysis_%j.out
#SBATCH --error=/tmp/stock_analysis_%j.err
#SBATCH --partition=Test
#SBATCH --ntasks=2
#SBATCH --nodes=1
#SBATCH --cpus-per-task=1
#SBATCH --time=00:05:00

module load python
timeout 300 python ~/stock_analysis/scripts/stock_analysis.py AAPL
timeout 300 python ~/stock_analysis/scripts/stock_analysis.py MSFT
wait
echo "Job Completed successfully"
```



```
dhpcsa@controller:~/stock_analysis/scripts$ python3 stock_analysis.py AAPL
Fetching data for AAPL...
[*********************100%***********************]  1 of 1 completed
Data saved for AAPL
dhpcsa@controller:~/stock_analysis/scripts$ python3 stock_analysis.py MSFT
Fetching data for MSFT...
[*********************100%***********************]  1 of 1 completed
Data saved for MSFT
dhpcsa@controller:~/stock_analysis/scripts$
```



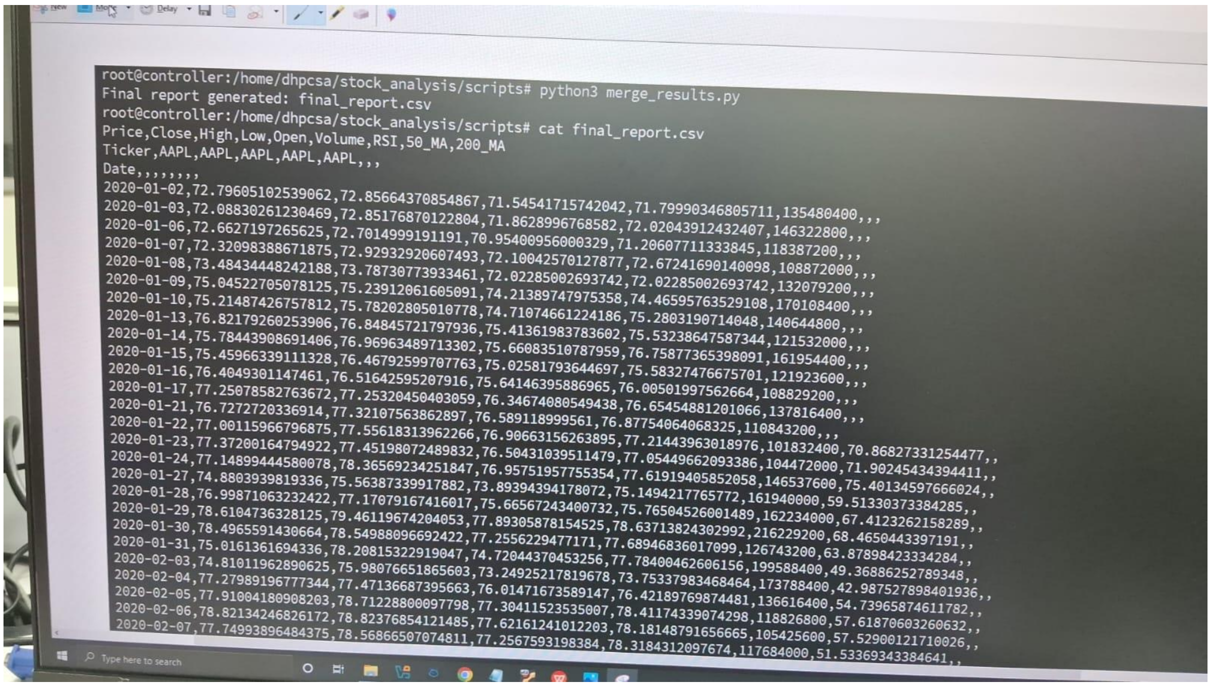
```
dhpcsa@controller:~/stock_analysis/scripts$ python3 stock_analysis.py AAPL
Fetching data for AAPL...
[*********************100%***********************]  1 of 1 completed
Data saved for AAPL
dhpcsa@controller:~/stock_analysis/scripts$ python3 stock_analysis.py MSFT
Fetching data for MSFT...
[*********************100%***********************]  1 of 1 completed
Data saved for MSFT
dhpcsa@controller:~/stock_analysis/scripts$ ls
AAPL_analysis.csv   job_node1.sh  job_node3.sh  job_node5.sh      MSFT_analysis.csv   stock_analysis.py
AAPL_processed.csv  job_node2.sh  job_node4.sh  merge_results.py  MSFT_processed.csv  stock.py
dhpcsa@controller:~/stock_analysis/scripts$
```



```
root@controller:/home/dhpcsa/stock_analysis/scripts# python3 stock_analysis.py AAPL MSFT GOOG
Fetching data for AAPL...
[*********************100%***********************]  1 of 1 completed
Data saved for AAPL
root@controller:/home/dhpcsa/stock_analysis/scripts# python3 stock_analysis.py MSFT GOOG
Fetching data for MSFT...
[*********************100%***********************]  1 of 1 completed
Data saved for MSFT
root@controller:/home/dhpcsa/stock_analysis/scripts# python3 stock_analysis.py GOOG
Fetching data for GOOG...
[*********************100%***********************]  1 of 1 completed
Data saved for GOOG
root@controller:/home/dhpcsa/stock_analysis/scripts# python3 stock_analysis.py AMZN
Fetching data for AMZN...
[*********************100%***********************]  1 of 1 completed
Data saved for AMZN
root@controller:/home/dhpcsa/stock_analysis/scripts#
```

Chapter 6

Results



```
root@controller:/home/dhpcsa/stock_analysis/scripts# python3 merge_results.py
Final report generated: final_report.csv
root@controller:/home/dhpcsa/stock_analysis/scripts# cat final_report.csv
Price,Close,High,Low,Open,Volume,RSI,50_MA,200_MA
Ticker,AAPL,AAPL,AAPL,AAPL,AAPL,,,
Date,,,,,,,,
2020-01-02,72.79605102539062,72.85664370854867,71.54541715742042,71.79990346805711,135480400,,,
2020-01-03,72.08830261230469,72.85176870122804,71.8628996768582,72.02043912432407,146322800,,,
2020-01-06,72.6627197265625,72.7014999191191,70.95400956000329,71.20607711333845,118387200,,,
2020-01-07,72.32098388671875,72.92932920607493,72.10042570127877,72.67241690140098,108872000,,,
2020-01-08,73.48434448242188,73.78730773933461,72.02285002693742,72.02285002693742,132079200,,,
2020-01-09,75.04522705078125,75.23912061605091,74.21389747975358,74.46595763529108,170108400,,,
2020-01-10,75.21487426757812,75.78202805010778,74.71074661224186,75.2803190714048,140644800,,,
2020-01-13,76.82179260253906,76.84845721797936,75.41361983783602,75.53238647587344,121532000,,,
2020-01-14,75.78443908691406,76.96963489713302,75.66083510787959,76.75877365398091,161954400,,,
2020-01-15,75.45966339111328,76.46792599707763,75.02581793644697,75.58327476675701,121923600,,,
2020-01-16,76.4049301147461,76.51642595207916,75.64146395886965,76.00501997562664,108829200,,,
2020-01-17,77.25078582763672,77.25320450403059,76.34674080549438,76.65454881201066,137816400,,,
2020-01-21,76.7272720336914,77.32107563862897,76.589118999561,76.87754064068325,110843200,,,
2020-01-22,77.00115966796875,77.55618313962266,76.90663156263895,77.21443963018976,101832400,70.86827331254477,,
2020-01-23,77.37200164794922,77.45198072489832,76.50431039511479,77.05449662093386,104472000,71.90245434394411,,
2020-01-24,77.14899444580078,78.36569234251847,76.95751957755354,77.61919405852058,146537600,75.40134597666024,,
2020-01-27,74.8803939819336,75.56387339917882,73.89394394178072,75.1494217765772,161940000,59.51330373384285,,
2020-01-28,76.99871063232422,77.17079167416017,75.66567243400732,75.76504526001489,162234000,67.4123262158289,,
2020-01-29,78.6104736328125,79.46119674204053,77.89305878154525,78.63713824302992,216229200,68.4650443397191,,
2020-01-30,78.4965591430664,78.54988096692422,77.2556229477171,77.68946836017099,126743200,63.87898423334284,,
2020-01-31,75.0161361694336,78.20815322919047,74.72044370453256,77.78400462606156,199588400,49.36886252789348,,
2020-02-03,74.81011962890625,75.98076651865603,73.24925217819678,73.75337983468464,173788400,42.987527898401936,,
2020-02-04,77.27989196777344,77.47136687395663,76.01471673589147,76.42189769874481,136616400,54.73965874611782,,
2020-02-05,77.91004180908203,78.71228800097798,77.30411523535007,78.41174339074298,118826800,57.61870603260632,,
2020-02-06,78.82134246826172,78.82376854121485,77.62161241012203,78.18148791656665,105425600,57.52900121710026,,
2020-02-07,77.74993896484375,78.56866507074811,77.2567593198384,78.3184312097674,117684000,51.53369343384641,,
```

Chapter 7
Conclusion

The "Stock Market Analysis Using SLURM" project shows how high-performance computing (HPC) can help analyze huge amounts of financial data. The project collects data, cleans it up, and organizes it so that it can be used to calculate important financial indicators like Moving Averages, RSI. This way, both current and past market trends can be studied effectively. The method takes care of common problems like messy data and ensures that the stored data is reliable for further analysis.

A key part of this project is using SLURM for job scheduling and resource management in an HPC cluster. This cluster has one main controller computer and 2 other compute nodes that work together. By splitting the work across these nodes, the system can process large datasets much faster. The design is modular, meaning that each calculation (like Moving Averages or RSI) is built as a separate unit. This makes it easy to update or add new financial indicators later on.

The project also includes strong error handling and validation methods. These features help catch and fix any problems with the data or calculations, ensuring that the results are accurate and consistent. The system has been tested under heavy workloads, and it has proven to handle increasing amounts of data without slowing down, showing that it can grow as needed.

In summary, this project demonstrates how modern HPC techniques and SLURM’s distributed job scheduling can be used to improve the analysis of stock market data. It speeds up the processing time and provides clear, useful insights into market trends. As financial markets become more complex and generate even more data, having a fast, scalable, and reliable system like this will be essential for making smart decisions in both research and real-world trading.

Chapter 8

Future work

This project can be improved in many ways to make it more efficient, accurate, and useful. Below are some key areas for future work:

1. More Data Sources

   Right now, we use **Yahoo Finance** to get stock data. In the future, we can add other sources like **Alpha Vantage, IEX Cloud, or Quandl** for more reliable data.

   Adding **real-time stock price tracking** instead of only using historical data.

   Using **news and social media sentiment analysis** to predict market trends.

2. Better SLURM Job Scheduling

   Improve **task distribution** among compute nodes to reduce processing time.

   Implement **dynamic job allocation**, so tasks are assigned based on node performance.

   Use **priority-based scheduling**, where important stocks (like high-trading stocks) get processed first.

3. More Financial Indicators

   Right now, we calculate **Moving Averages (MA) and RSI**. In the future, we can add indicators like **MACD, Bollinger Bands, and Fibonacci Retracement** for better analysis.

   Implement **risk analysis models** to help investors make better decisions.

4. Improved Reporting and Visualization

   Instead of just a **CSV file**, we can create **interactive dashboards** using **Power BI or Tableau**.

   Generate **automated reports** summarizing stock trends and predictions.

5. Machine Learning for Predictions

   Use **machine learning models** to predict stock price movements based on past data.

Implement **AI-based trading strategies** to analyze and suggest the best stock trades.

These improvements will make the project more powerful and useful for real-world stock market analysis.

Chapter 9

References

1. **https://slurm.schedmd.com/documentation.html**

2. Balle, S.M. and Palermo, D.J. 2007. Enhancing an Open Source Resource Manager with Multi-core/Multi-threaded Support. Job Scheduling Strategies for Parallel Processing. Springer Berlin Heidelberg. 37–50.

3. Slurm Developers Team. 2018. Slurm: Large Cluster Administration Guide. Retrieved March 8, 2018 from https://slurm.schedmd.com/big_sys.html