Word Count Operation on the Complete Works of William Shakespeare

Cloud Computing

Paul Fontanges; Stu No. : 461720

Prepared for 31 March , 2025

Paul Fontanges
Student Number : 461720

Abstract

Cloud computing has revolutionized the way data is processed and stored, enabling scalable and efficient solutions for large scale data analysis. This project explores a distributed word count application using AWS services to process Shakespeare's works. The architecture leverages Amazon EC2 instances for parallel computation, S3 for data storage, and SQS for job distribution, ensuring an efficient and scalable implementation. By comparing different deployment configurations single instance, two instance, and four instance setups we analyze the impact of resource distribution on performance. The findings highlight the trade offs between computation speed, resource utilization, and cost, offering insights into optimizing cloud based data processing pipelines.

Paul Fontanges
Student Number : 461720

Contents

List of Figures

Paul Fontanges
Student Number : 461720

1 Introduction

The mission involves designing and implementing an application that allows parallel processing of large datasets in a distributed Cloud environment. The primary goal is to develop a solution capable of breaking down and efficiently processing a large file, specifically the complete works of William Shakespeare. This application should be able to count the words in this file in a distributed manner, using elastic and scalable resources provided by the Cloud platform. The proposed solution relies on the use of Amazon Web Services (AWS), with a particular focus on EC2 for parallel task execution and S3 for file storage. The main challenge of this project is to design an architecture capable of breaking the data into work packets and distributing them across multiple processing nodes to optimize performance while respecting the resource constraints of the Cloud environment. To achieve this, several copies of the text file are used to simulate a high workload, and the data is sent to a maximum of 4 EC2 instances to avoid exceeding the resource limitations associated with the Cloud environment. The use of queuing features (such as AWS SQS or other custom systems) allows for managing the distribution of tasks and aggregating the results from each processing node. However, throughout the development of this solution, several issues were encountered, mainly related to IAM access management and the restrictions associated with Cloud resource permissions. These challenges have impacted the smooth progress of the project and need to be addressed in a dedicated section of the report. This project highlights the importance of resource and permission management in a distributed Cloud environment to ensure smooth and secure task execution.

2 Methodology

2.1 AWS Architecture

The solution leverages multiple AWS services, including EC2, S3, and SQS, to perform a distributed word count on a large text dataset. The goal is to split the input file into multiple segments, process them in parallel across multiple EC2 instances, and aggregate the results to obtain the final word count.

The AWS architecture consists of the following components:

- **Amazon S3:** Used to store the input file and final aggregated word count results.

- **Amazon EC2:** Runs the distributed word count script across multiple instances, each processing a portion of the file.
- **Amazon SQS:** Facilitates communication between EC2 instances and the central system, ensuring task tracking and result collection.

2.2 AWS Services Used

2.2.1 Amazon S3

Role: An S3 bucket is used to store both the source text file and the aggregated word count results. This ensures accessibility and scalability of the data processing pipeline.

Process: The input file is divided into multiple smaller files before being uploaded to the S3 bucket. Each EC2 instance downloads its assigned segment, processes it, and uploads the results back to S3.

2.2.2 Amazon EC2

Role: Multiple EC2 instances process the text in parallel, each handling a separate portion of the dataset to enhance efficiency and reduce processing time.

Configuration: The instances run Amazon Linux 2 AMI and are provisioned as `t2.micro` types for cost-effectiveness. The `deploy_shakespeare_wordcount.sh` script orchestrates the deployment of instances and execution of the word count script.

2.2.3 Amazon SQS

Role: AWS SQS is used to track the status of processing tasks, allowing the system to monitor progress and handle errors efficiently.

Implementation: Each EC2 instance sends updates on task execution (starting, completion, errors) to the SQS queue, ensuring real-time tracking and debugging.

2.3 Data Preprocessing

Prior to distributed processing, the input file undergoes preprocessing to optimize performance:

- **Local Text Cleaning** Locally, we remove irrelevant parts (such as the table of contents and Gutenberg project information), and certain lines are removed. Lines 1 to 79 and 196038 to 196388 are removed. Furthermore, non-alphanumeric characters are removed, and text is normalized to lowercase for consistency.
- **File Splitting:** The dataset is divided into smaller chunks, enabling parallel processing (Here we decided to split it into 4 blocks of the same size).

2.4 EC2 Instance Configuration and Execution

To enable parallel processing, multiple EC2 instances are launched with the following configurations:

- **AMI:** Amazon Linux 2
- **Instance Type:** `t2.micro`

Once the instances are launched, they execute the `distributed_wordcount.py` script, which:

- Downloads and processes the assigned text file segment.
- Computes word frequencies and uploads the results to S3.
- Sends task status updates to the SQS queue for monitoring.

Finally, the `aggregate_results.py` script consolidates the individual results into a final word count output.

2.5 Distributed Processing Workflow

Figure 1 illustrates the architecture of our distributed word count pipeline, which leverages multiple AWS services to process large text files efficiently. The workflow consists of the following main steps:

- **Data Preparation:** The source file (`shakespeare.txt`) is pre-processed and split into four smaller segments before being uploaded to an Amazon S3 bucket.

- **Deployment:** A deployment script (`deploy_shakespeare_wordcount.sh`) running on LearnerLab initializes four EC2 instances.
- **Parallel Processing:** Each EC2 instance downloads a specific segment of the text file from S3 and executes a word count script (`distributed_wordcount.py`).
- **Results Aggregation:** Once the distributed word count is complete, the results from all instances are aggregated.
- **Status Updates:** Each EC2 instance sends execution status updates to an SQS queue for monitoring and logging purposes.

This architecture ensures that the workload is evenly distributed across multiple instances, reducing processing time and improving efficiency. The use of Amazon S3 allows for centralized storage of both input and output files, while SQS facilitates communication and monitoring.

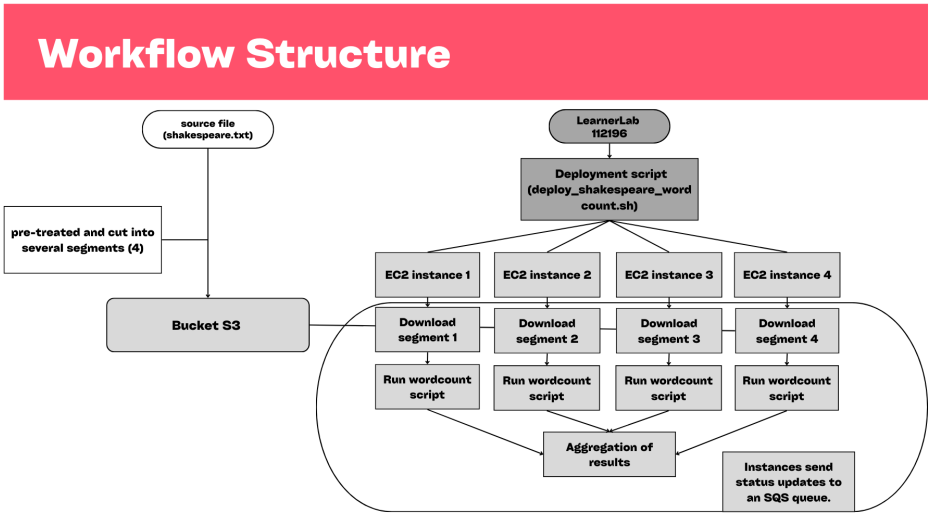

Figure 1: System pipeline

3 Implementation

3.1 Overview

The distributed word count processing system is implemented on AWS EC2, involving several steps: data retrieval, deployment, parallel execution, and result aggregation. This section focuses on the implementation using four EC2 instances.

3.2 Pipeline Structure

The pipeline follows these key steps:

- **Download data from Amazon S3:** Input text files are stored in an S3 bucket and retrieved for processing.
- **Deploy to EC2 instances:** Scripts and data are deployed to EC2 instances for distributed execution.
- **Parallel execution:** Each EC2 instance processes a portion of the data in parallel to compute word frequencies.
- **Result aggregation:** The results from all nodes are aggregated into a final report.

3.3 Implementation Details

The implementation is orchestrated using a Bash script that manages the entire pipeline.

3.3.1 Configuration and Initialization

Key variables such as project name, EC2 key pair path, S3 bucket, and queue names are defined, along with the IP addresses of the EC2 instances. The EC2 instance is configured using a Bash script that automates file transfers, environment setup, and dependency installation.

Listing 1: Pipeline Configuration

```
PROJECT_NAME="shakespeare_wordcount"
KEY_PATH="/mnt/vocwork4/work/eee_W_4331436/asn3940321_1/.ssh/key-wordcount
    .pem"
BUCKET_NAME="shakespeare-wordcount-bucket-fontanges"
QUEUE_NAME="shakespeare-wordcount-queue"
INSTANCES=("34.227.222.82" "54.89.192.106" "54.147.17.113" "3.80.30.146")
```

Helper functions are defined for logging execution timestamps and calculating execution time for each step.

3.3.2 Downloading Data from S3

The following script downloads text files from the S3 bucket into a local directory.

For example, here with the solution using just one instance :

Listing 2: Downloading Data from S3

```
download_files_from_s3() {
    mkdir -p ./data
    for i in {0..3}; do
        aws s3 cp s3://${BUCKET_NAME}/block_${i}.txt ./data/
    done
}
```

3.4 Deploying Files to EC2 Instances

The deployment function `deploy_to_instances` distributes the necessary files across multiple EC2 instances. Each instance is configured with AWS credentials, necessary directories, and dependencies like `boto3` and `pandas`. The input data is transferred, and instances are ready for parallel processing.

3.5 Executing Word Count on EC2 Instances

The function `run_distributed_processing` runs parallel processing on each EC2 instance. Each instance executes the `distributed_wordcount.py` script to count word occurrences, storing the results in structured output files.

3.6 Aggregating Results

The function `aggregate_results` collects and processes the output files from all EC2 instances, generating a summary report with the top 10 most frequent words.

3.6.1 Final Output and Upload to S3

The aggregated results are saved and uploaded to S3 for further analysis.

3.7 Execution Summary

A summary of the execution is displayed, including the total execution time.

Listing 3: Execution Summary

```
echo "======================================"
echo "Distributed processing successfully completed"
echo "Total execution time: $(display_execution_time $START_TIME_TOTAL
    $END_TIME_TOTAL)"
echo "======================================"
```

3.8 Execution Flow for Single Instance

The execution for a single EC2 instance follows this sequence:

- Download input files from S3.
- Deploy files to the EC2 instance.
- Execute text processing on the instance.
- Aggregate and summarize the results.
- Upload the final report to S3.

4 Analysis

In this section, we present the results obtained for the different implementations of our code:

- Single EC2 instance implementation
- Two EC2 instances implementation
- Four EC2 instances implementation

4.1 Performance Analysis

To evaluate the efficiency of our implementations, we measured the execution time for each solution. As expected, execution time decreases with the number of instances used (Figure 2).

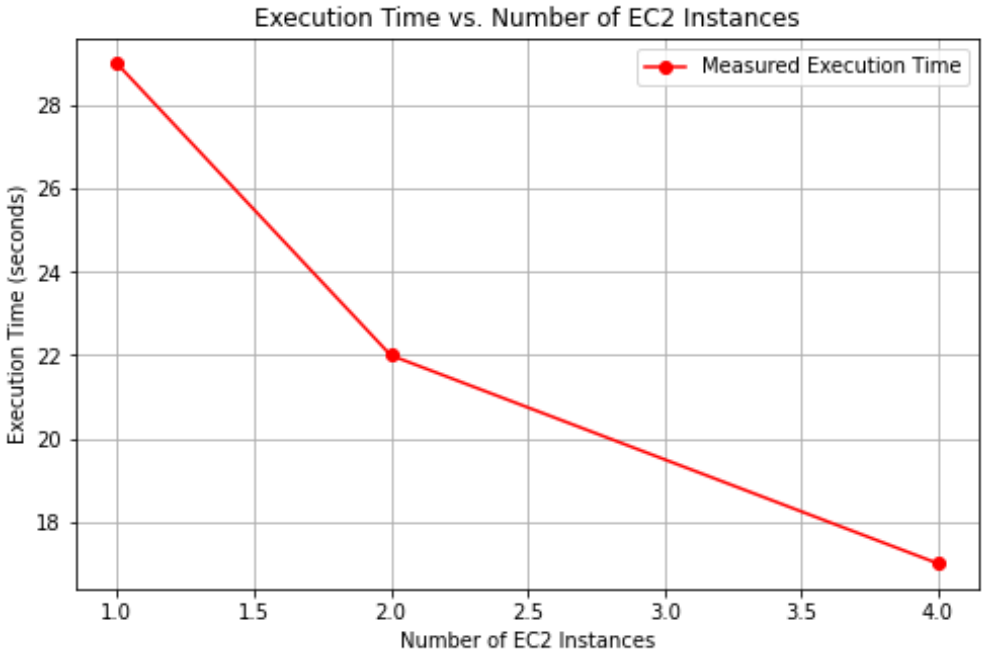

Figure 2: Measured Execution Time

- Using a single EC2 instance, the execution time was **29 seconds**.
- Using two EC2 instances, the execution time was significantly reduced.
- Using four EC2 instances, the execution time was **17 seconds**.

This nearly halves the processing time, demonstrating the advantages of parallelization in AWS cloud computing.

4.2 Cost Analysis

Cost analysis was conducted based on AWS pricing, excluding Free Tier benefits. We observed a substantial increase in cost with the number of instances used (Figure 3):

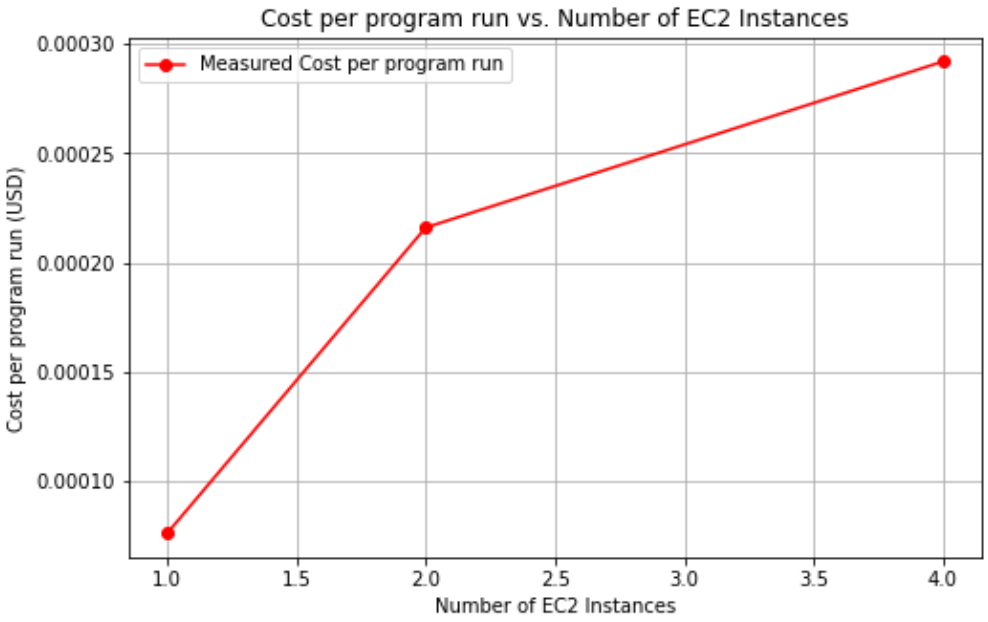

Figure 3: Measured Cost per program run

- A single execution on one instance costs approximately **$0.00029**.
- Comparing one instance to four instances, there was an increase of over **300%** (from **$0.000076** to **$0.00029**).
- The total cost of script development, debugging, and testing was approximately **$2**.

4.3 Validation of Results

We verified our implementation by analyzing the word count results. The total number of words and the top 10 most frequent words remained consistent across different implementations, confirming the correctness of our approach.

```
Summary of results:
Total number of words: 988813

Top 10 most frequent words:
1. 'the' - 30354 occurrences
2. 'and' - 28501 occurrences
3. 'i' - 24201 occurrences
4. 'to' - 21095 occurrences
5. 'of' - 18728 occurrences
6. 'a' - 16488 occurrences
7. 'you' - 14652 occurrences
8. 'my' - 13204 occurrences
9. 'in' - 12530 occurrences
10. 'that' - 12271 occurrences

Aggregation time: 0.08 seconds
[2025-03-30 11:48:12] Successfully aggregated results in result_all_1.txt
upload: ./result_all_1.txt to s3://shakespeare-wordcount-bucket-fontanges/result_all_1.txt
[2025-03-30 11:48:13] Result_all_1.txt file uploaded to S3
[2025-03-30 11:48:13] Aggregation completed in 0h 0m 1s

====================================
SINGLE INSTANCE ANALYSIS SUMMARY
====================================
Processing successfully completed on a single instance
Total execution time: 0h 0m 29s
Estimated AWS cost: .0000760 USD
====================================
```

Figure 4: Terminal Results using 1 instance

```
Summary of results:
Total number of words: 988813

Top 10 most frequent words:
1. 'the' - 30354 occurrences
2. 'and' - 28501 occurrences
3. 'i' - 24201 occurrences
4. 'to' - 21095 occurrences
5. 'of' - 18728 occurrences
6. 'a' - 16488 occurrences
7. 'you' - 14652 occurrences
8. 'my' - 13204 occurrences
9. 'in' - 12530 occurrences
10. 'that' - 12271 occurrences

Aggregation time: 0.09 seconds
[2025-03-30 11:56:25] Successfully aggregated results in result_all_2.txt
upload: ./result_all_2.txt to s3://shakespeare-wordcount-bucket-fontanges/result_all_2.txt
[2025-03-30 11:56:26] Result_all_2.txt file uploaded to S3
[2025-03-30 11:56:26] Aggregation completed in 0h 0m 1s

====================================
TREATMENT SUMMARY
====================================
Distributed processing (Two Instances) successfully completed
Total execution time: 0h 0m 22s
Estimated AWS cost: .0002160 USD
====================================
```

Figure 5: Terminal Results using 2 instance

```
Report successfully generated in 'result_all.txt'

Summary of results:
Total number of words: 988813

Top 10 most frequent words:
1. 'the' - 30354 occurrences
2. 'and' - 28501 occurrences
3. 'i' - 24201 occurrences
4. 'to' - 21095 occurrences
5. 'of' - 18728 occurrences
6. 'a' - 16488 occurrences
7. 'you' - 14652 occurrences
8. 'my' - 13204 occurrences
9. 'in' - 12530 occurrences
10. 'that' - 12271 occurrences
[2025-03-30 11:37:13] Successfully aggregated results in result_all.txt
upload: ./result_all.txt to s3://shakespeare-wordcount-bucket-fontanges/result_all.txt
[2025-03-30 11:37:14] Result_all.txt file uploaded to S3
[2025-03-30 11:37:14] Aggregation completed in 0h 0m 1s

====================================
TREATMENT SUMMARY
====================================
Distributed processing successfully completed
Total execution time: 0h 0m 17s
Estimated AWS cost: .0002920 USD
====================================
```

Figure 6: Terminal Results using 4 instance

For further validation, we compared our results with external estimates. Shakespeare's total word count is estimated to be around **885,000 words**. Our results were in the expected range but showed a discrepancy of approximately **100,000 words**. This indicates potential differences in text processing and counting methods, requiring further investigation into data handling and analysis techniques.

5 Conclusions

This study demonstrated the advantages of distributed computing in cloud environments through the implementation of a word count application on AWS. The results confirmed that increasing the number of EC2 instances significantly improves processing speed by distributing the workload efficiently. However, this comes at the cost of increased infrastructure expenses and more complex orchestration. The optimal configuration depends on balancing performance with cost effectiveness. Future work could explore auto scaling mechanisms and serverless alternatives to further optimize efficiency and cost. These insights contribute to

best practices for designing scalable data processing architectures in the cloud.

6 Reference

1. AWS *Amazon EC2 On-Demand Pricing.* Available `https://aws.amazon.com/ec2/pricing/on-demand/?nc1=h_ls`.

2. AWS *Amazon S3 On-Demand Pricing.* Available `https://aws.amazon.com/s3/pricing/`.

3. AWS *Amazon S3 On-Demand Pricing - Request and Data Retrieval.* Available `https://aws.amazon.com/s3/pricing/`.

4. AWS *Amazon SQS pricing.* Available `https://aws.amazon.com/sqs/pricing/`.

5. Folger Shakespeare Library *How many words did Shakespeare write?.* Available `https://www.folger.edu/explore/shakespeares-works/frequently-asked-questions/#:~:text=How%20many%20words%20did%20Shakespeare,criteria%20of%20what%20is%20included`.