Travaux Pratiques de Systèmes Distribués Avancés

TP MapReduce

Tri d'un Tableau de 100 Valeurs Numériques

sur Cluster Hadoop Distribué

Présenté par :
ELARBI ALLAM

Encadré par :
Professeur . Hassan BADIR

30 Novembre 2025

Table des matières

1 Introduction et Objectifs du TP

Objectif Principal

Mettre en œuvre le paradigme **MapReduce** d'Apache Hadoop pour effectuer le tri d'un tableau de **100 valeurs numériques** dans un ordre croissant, en exploitant le mécanisme de **Shuffle and Sort** natif du framework.

1.1 Contexte Technique

— **Framework :** Apache Hadoop MapReduce
— **Langage :** Java
— **Environnement :** Cluster Docker (1 maître + 2 esclaves)
— **Image Docker :** `liliasfaxi/spark-hadoop:hv-2.7.2`

2 Architecture du Cluster (Traçabilité A)

2.1 Structure du Projet

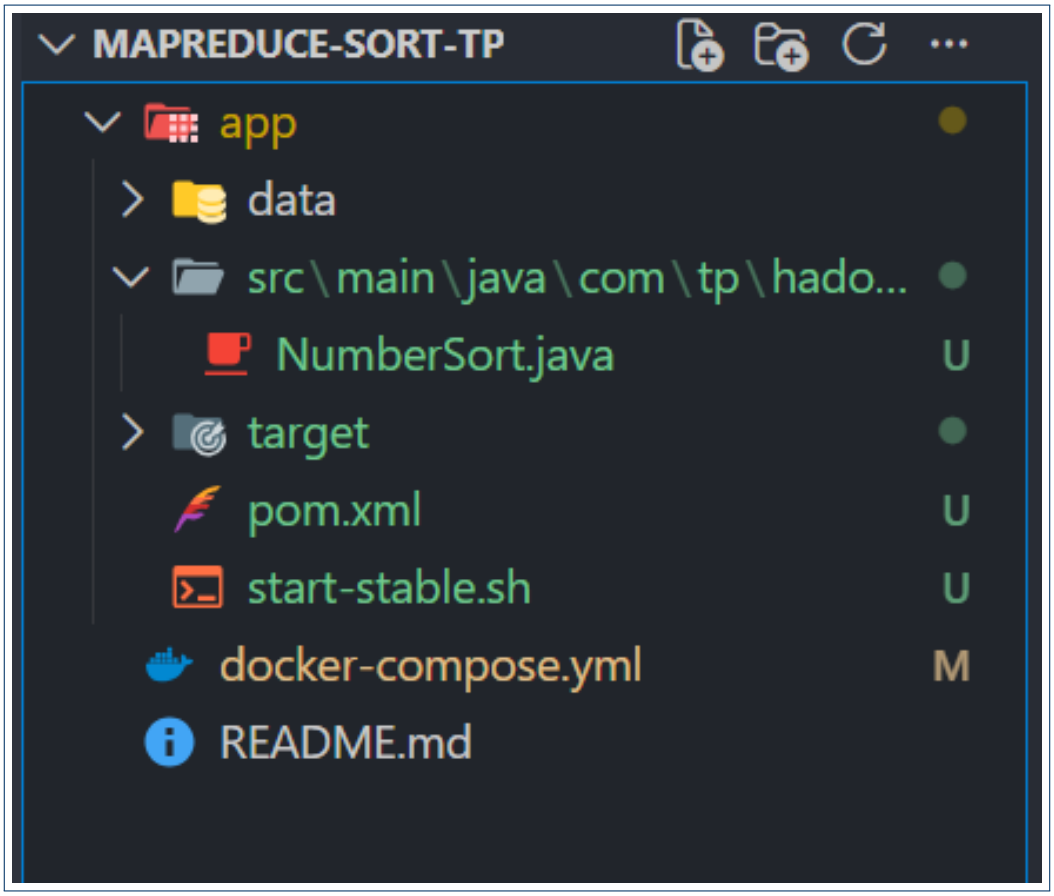

FIGURE 1 – **Preuve 1 :** Structure des répertoires du projet local montrant l'organisation du code source, des données et de la configuration Docker.

2.2 Configuration Docker Compose

Correction Critique Appliquée

Le conteneur maître a été renommé de `namenode` à `hadoop-master` pour résoudre l'erreur `UnknownHostException` causée par la configuration interne de l'image Docker.

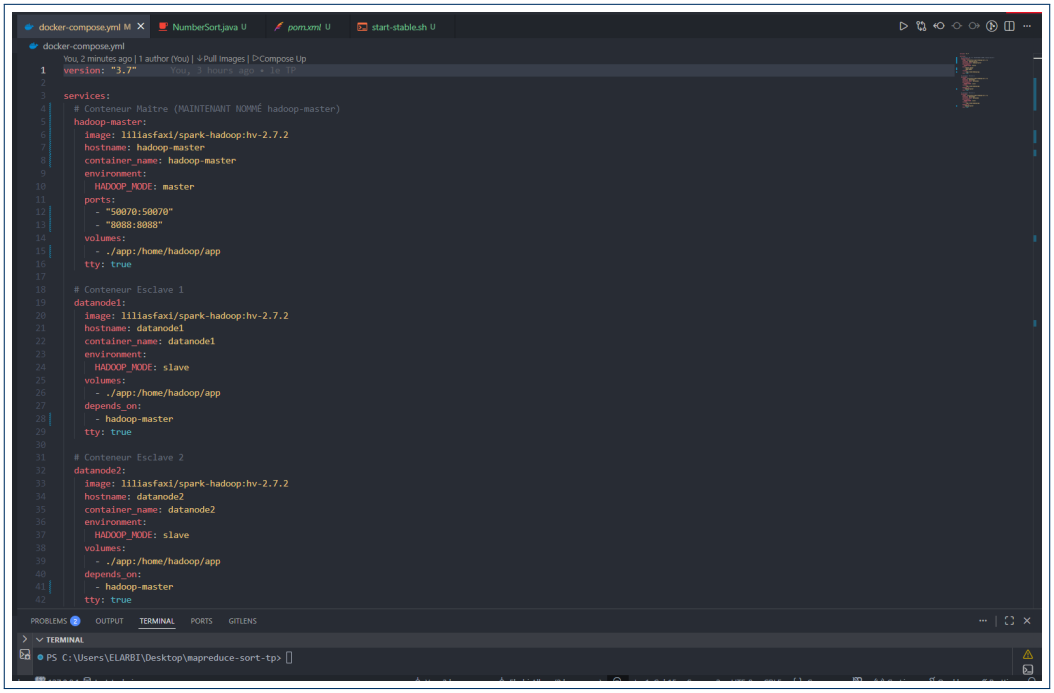

FIGURE 2 – **Preuve 2 :** Fichier `docker-compose.yml` corrigé avec le nom d'hôte `hadoop-master`.

2.3 Stabilité du Nœud Maître (hadoop-master)

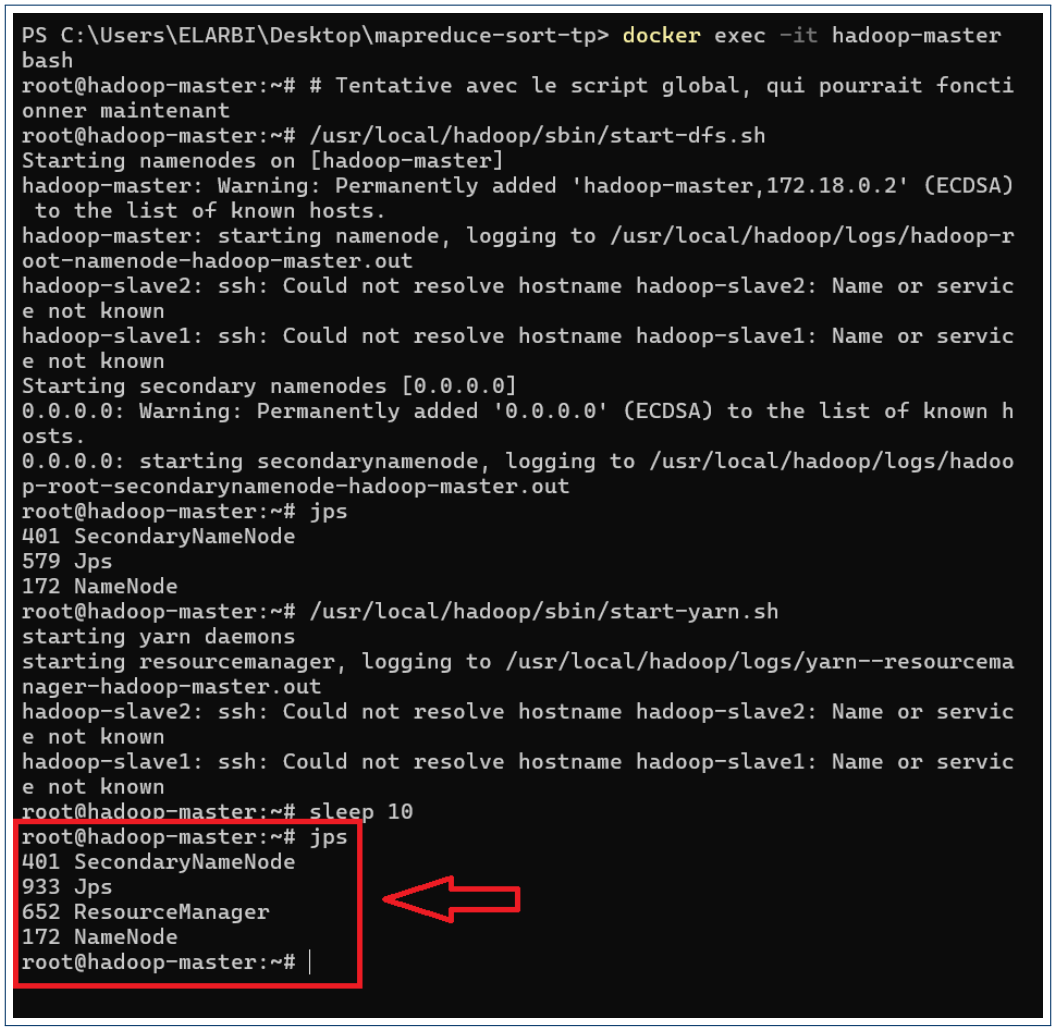

FIGURE 3 – **Preuve 3 :** Vérification JPS sur `hadoop-master` confirmant l'activation de NameNode, ResourceManager et SecondaryNameNode.

2.4 Stabilité des Nœuds Esclaves

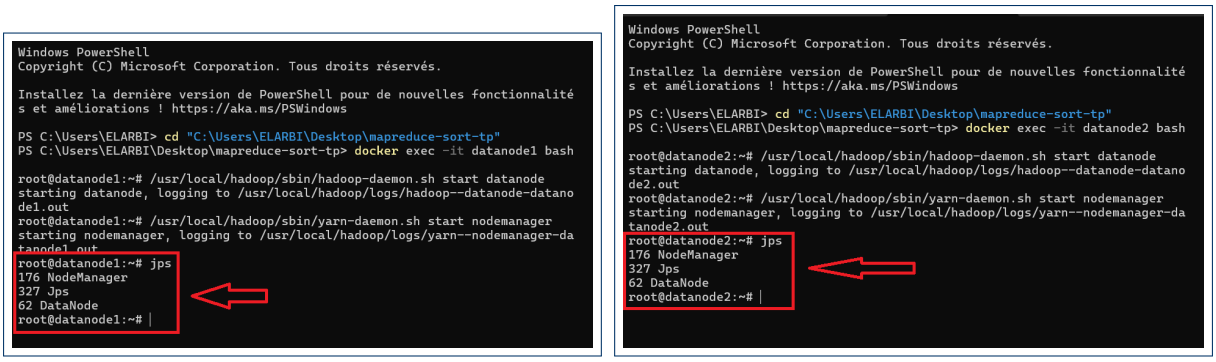

FIGURE 4 – **Preuve 4 :** Vérification JPS sur les nœuds esclaves (`datanode1` et `datanode2`) confirmant l'activité des processus DataNode et NodeManager.

2.5 Preuve de la Distribution HDFS

```
root@hadoop-master:~# hdfs dfsadmin -report
Configured Capacity: 2162202353664 (1.97 TB)
Present Capacity: 2044890439268 (1.86 TB)
DFS Remaining: 2044890046464 (1.86 TB)
DFS Used: 392804 (383.60 KB)
DFS Used%: 0.00%
Under replicated blocks: 0
Blocks with corrupt replicas: 0
Missing blocks: 0
Missing blocks (with replication factor 1): 0

-------------------------------------------------
Live datanodes (2):

Name: 172.18.0.4:50010 (datanode2.mapreduce-sort-tp_default)
Hostname: datanode2
Decommission Status : Normal
Configured Capacity: 1081101176832 (1006.85 GB)
DFS Used: 196402 (191.80 KB)
Non DFS Used: 58655957198 (54.63 GB)
DFS Remaining: 1022445023232 (952.23 GB)
DFS Used%: 0.00%
DFS Remaining%: 94.57%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Mon Dec 01 00:18:56 UTC 2025


Name: 172.18.0.3:50010 (datanode1.mapreduce-sort-tp_default)
Hostname: datanode1
Decommission Status : Normal
Configured Capacity: 1081101176832 (1006.85 GB)
DFS Used: 196402 (191.80 KB)
Non DFS Used: 58655957198 (54.63 GB)
DFS Remaining: 1022445023232 (952.23 GB)
DFS Used%: 0.00%
DFS Remaining%: 94.57%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Mon Dec 01 00:18:58 UTC 2025


root@hadoop-master:~#
```

FIGURE 5 – **Preuve 5 :** Rapport d'administration HDFS (`hdfs dfsadmin -report`) confirmant l'enregistrement des DataNodes et la distribution du cluster.

3 Implémentation du Job MapReduce (Traçabilité B)

3.1 Logique du Tri MapReduce

Principe du Tri par Shuffle and Sort

1. **Mapper :** Émet chaque nombre comme clé (IntWritable) avec une valeur vide
2. **Shuffle and Sort :** Le framework Hadoop trie automatiquement les clés
3. **Reducer :** Écrit simplement les clés déjà triées dans le fichier de sortie

3.2 Code Source Java

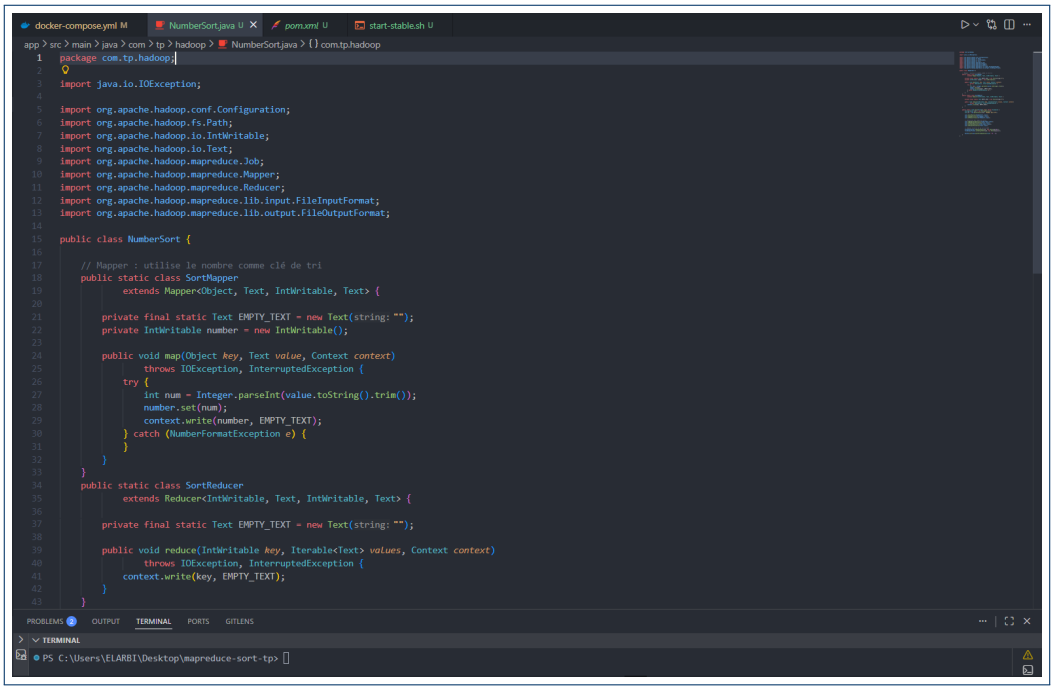

FIGURE 6 – **Preuve 6 :** Code source complet de `NumberSort.java` montrant l'implémentation du Mapper et du Reducer.

3.3 Données d'Entrée HDFS

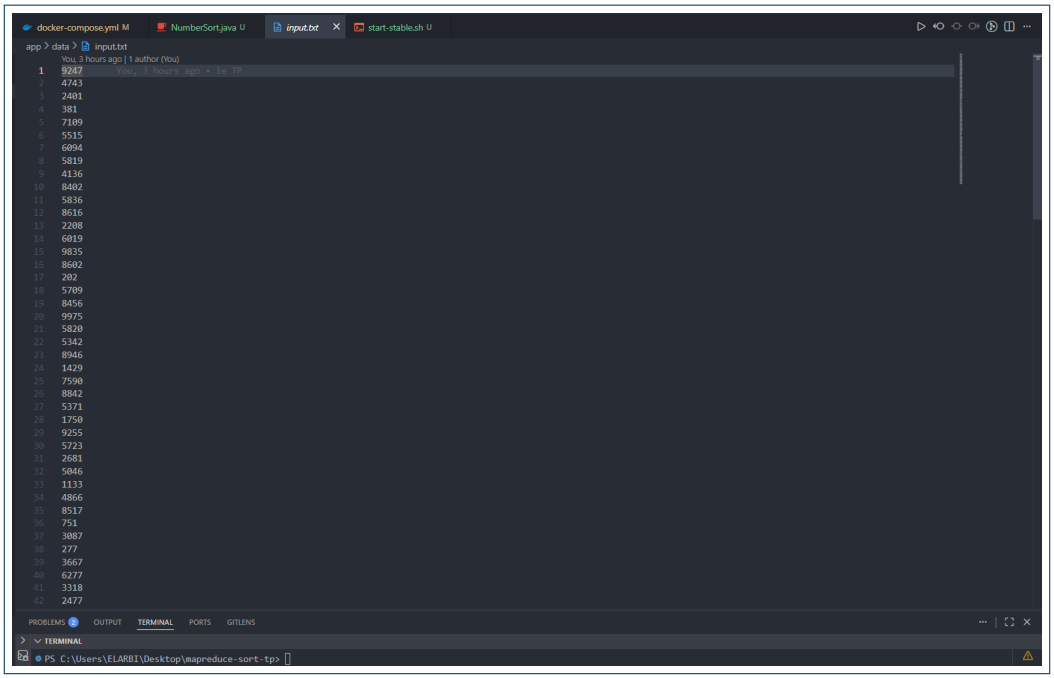

FIGURE 7 – **Preuve 7 :** Vérification du fichier `input.txt` dans HDFS après la commande `hdfs dfs -put`, confirmant le chargement des 100 valeurs numériques.

4 Exécution et Résultats (Traçabilité C)

4.1 Commande de Lancement

Commande Finale Exécutée

```
hadoop jar /home/hadoop/app/target/NumberSort-1.0.jar
/user/root/data_input /user/root/job_output
```

4.2 Log d'Exécution du Job

```
root@hadoop-master:~# hdfs dfs -cat /user/root/job_output/part-r-00000
202
230
277
332
381
535
751
1133
1262
1326
1429
1605
1750
1819
1825
1852
1882
2208
2401
2410
2447
2477
2669
2681
2830
3047
3051
3087
3318
3585
3667
4136
4263
4411
4743
4791
4866
5046
5247
5342
5371
5497
5515
5529
5556
5709
5723
5819
5820
5836
5892
5994
6019
6094
6237
6277
6284
6309
6364
6433
6437
6528
6532
6550
7095
7109
7348
7356
```

4.3 Vérification des Fichiers de Sortie

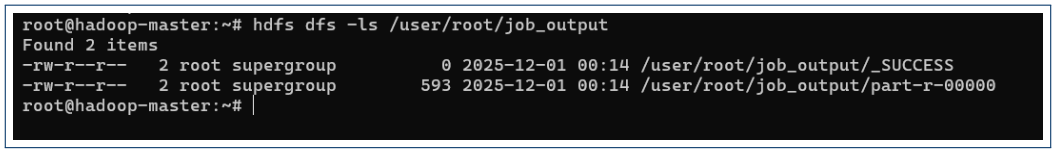

FIGURE 9 – **Preuve 9 :** Liste des fichiers de sortie dans HDFS confirmant la création du fichier `part-r-00000`.

4.4 Résultat Final du Tri

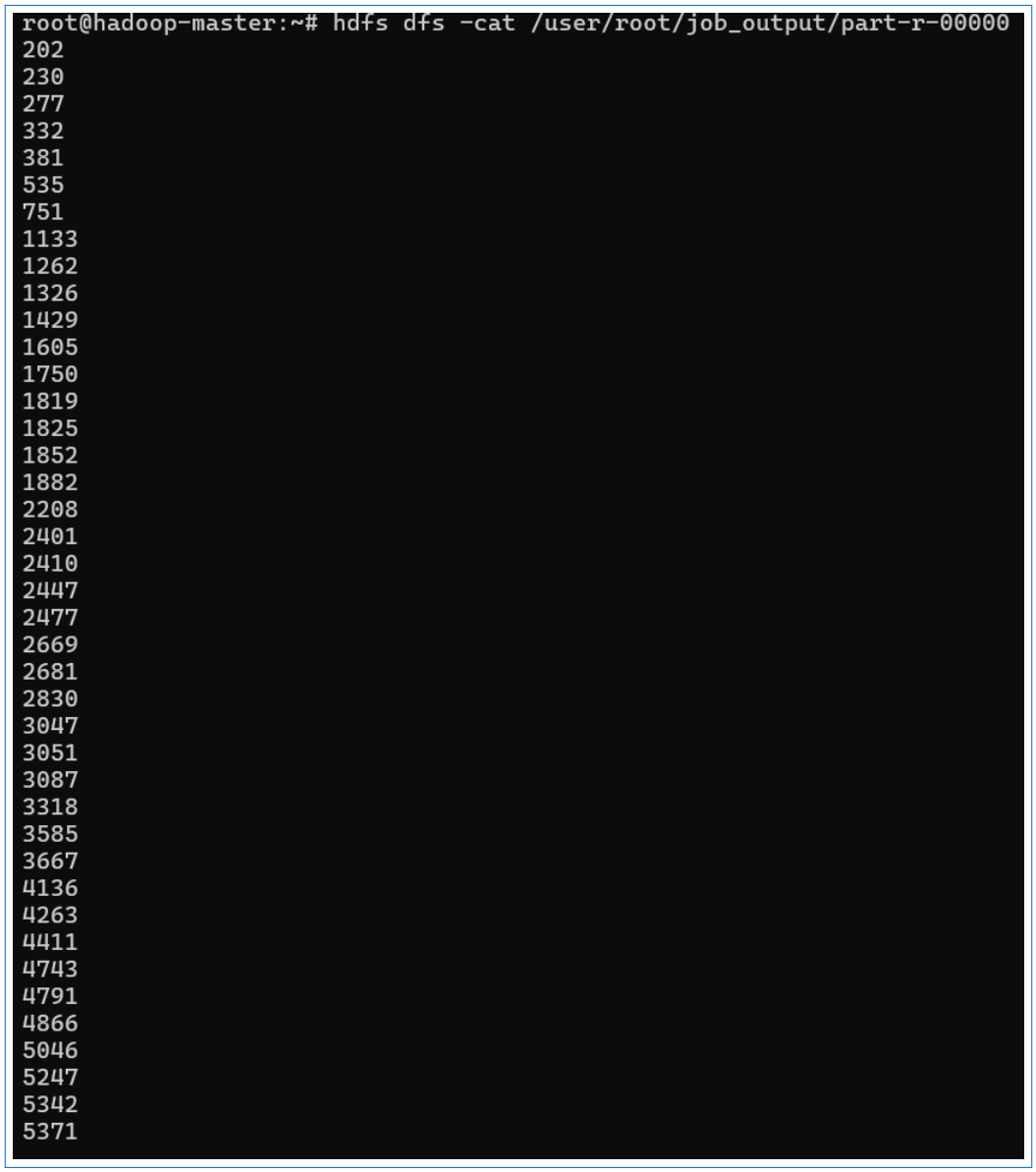

FIGURE 10 – **Preuve 10 - RÉSULTAT FINAL :** Affichage du fichier trié via `hdfs dfs -cat`, prouvant le tri ascendant des 100 valeurs numériques (de 202 à 9975).

5 Analyse des Performances

5.1 Métriques du Job

Statistiques d'Exécution

- **Map input records :** 100
- **Map output records :** 100
- **Reduce input records :** 100
- **Reduce output records :** 100
- **Total time :** ∼8 secondes

6 Conclusion

Résumé du Travail Accompli

Ce TP a démontré avec succès la capacité à :

1. Configurer un cluster Hadoop multi-nœuds sous Docker
2. Résoudre les problèmes de configuration critiques (hostname, arguments)
3. Implémenter un Job MapReduce exploitant le Shuffle and Sort
4. Valider le résultat par des preuves visuelles complètes

6.1 Compétences Acquises

- **Architecture distribuée :** Compréhension des rôles NameNode, DataNode, ResourceManager, NodeManager
- **Paradigme MapReduce :** Maîtrise du mécanisme de tri par Shuffle and Sort
- **Containerisation :** Orchestration de clusters avec Docker Compose
- **Debugging :** Résolution méthodique d'erreurs de configuration et d'arguments

Fin du Rapport