Journal of Applied Informatics and Computing (JAIC)
Vol.8, No.2, December 2024, pp. 390~399
e-ISSN: 2548-6861

Comparison of Hadoop Mapreduce and Apache Spark in Data Processing with Hgrid247-DE

Firmania Dwi Utami [1]*, Femi Dwi Astuti [2]**
* Informatika, Universitas Teknologi Digital Indonesia
** Universitas Teknologi Digital Indonesia
firmania.dwi23@students.utdi.ac.id[1], femi@utdi.ac.id[2]

Article Info

Article history:

Received 2024-09-24
Revised 2024-10-14
Accepted 2024-10-25

Keyword:

Data,
Apache Spark,
Mapreduce,
Hadoop

ABSTRACT

In today's rapidly evolving information technology landscape, managing and analyzing big data has become one of the most significant challenges. This paper explores the implementation of two major frameworks for big data processing: Hadoop MapReduce and Apache Spark. Both frameworks were tested in three scenarios sorting, summarizing, and grouping using HGrid247-DE as the primary tool for data processing. A diverse set of datasets sourced from Kaggle, ranging in size from 3 MB to 260 MB, was employed to evaluate the performance of each framework. The findings reveal that Apache Spark generally outperforms Hadoop MapReduce in terms of processing speed due to its in-memory data handling capabilities. However, Hadoop MapReduce proved to be more efficient in specific scenarios, particularly when dealing with smaller tasks or when memory resources are limited. This is largely because Apache Spark can experience overhead when initializing tasks for smaller jobs. Furthermore, Hadoop MapReduce's reliance on disk I/O makes it more suitable for tasks involving vast amounts of data that surpass available memory. In contrast, Spark excels in situations where quick iterative processing and real-time data analysis are essential. This study provides valuable insights into the strengths and limitations of each framework, offering guidance for practitioners and researchers when selecting the appropriate tool for specific big data processing requirements, particularly with respect to speed, memory usage, and task complexity.

I. PENDAHULUAN

Di era perkembangan teknologi informasi yang pesat saat ini, terdapat layanan yang berkembang sangat cepat, yaitu teknologi internet. Dengan internet, semua perangkat elektronik yang memiliki alamat Internet Protocol (IP) dapat saling terhubung, memungkinkan pertukaran informasi secara real-time dan mendukung berbagai aplikasi yang memudahkan kehidupan sehari-hari. Layanan software kini tersedia dalam berbagai bentuk, baik berbayar maupun gratis, dan dapat diakses oleh setiap pengguna di seluruh dunia. Dengan semakin kompleksnya teknologi yang dikembangkan, konsumsi data penyimpanan pun meningkat secara eksponensial. Data yang ada di aplikasi internet tidak hanya berjumlah besar, tetapi juga bervariasi dalam bentuk dan struktur, menjadikannya tantangan tersendiri untuk dikelola dan dianalisis secara efektif [1].

Istilah big data telah menjadi bagian dari pembicaraan sehari-hari dalam dunia teknologi. Big data tidak hanya mencakup volume data yang besar, tetapi juga beragam jenis data dan kecepatan dalam memproses data tersebut[2]. Menurut Viktor Mayer-Schönberger dan Kenneth Cukier (2013), big data merupakan kemampuan untuk menganalisis data dalam jumlah besar dengan cara yang sebelumnya tidak mungkin dilakukan, untuk menemukan pola baru dan membuat keputusan yang lebih baik[3]. Pengolahan data menjadi semakin penting, terutama di sektor bisnis, kesehatan, dan pemerintahan, di mana analisis data yang akurat dapat menghasilkan wawasan berharga. Data yang dihasilkan dari berbagai sumber, termasuk media sosial, sensor, transaksi online, dan aplikasi mobile, menunjukkan

bahwa kita kini berada di era informasi yang sangat kaya, yang memerlukan alat dan teknik yang tepat untuk mendapatkan manfaat maksimal darinya [4].

Namun, pengolahan data bukanlah tugas yang mudah. Diperlukan teknologi yang tepat untuk menangani volume, variasi, dan kecepatan data yang ada. Beberapa alat dan sistem telah dikembangkan untuk membantu dalam pengolahan data, termasuk Hadoop, Spark, Flink, Kafka, dan RapidMiner. Setiap alat memiliki keunggulan dan fokus tertentu, yang membuatnya lebih cocok untuk jenis pekerjaan tertentu. Apache Hadoop misalnya, merupakan salah satu framework yang paling banyak digunakan, yang menerapkan model pemrograman Mapreduce untuk memproses data besar. Hadoop menyediakan solusi penyimpanan dan pengolahan data terdistribusi yang memungkinkan organisasi untuk mengelola dataset besar dengan cara yang efisien [5].

Salah satu komponen utama Hadoop adalah Hadoop Distributed File System (HDFS), yang dirancang untuk menyimpan dan mengelola data besar secara efisien dalam lingkungan komputasi terdistribusi. Dalam prakteknya, HDFS membagi file besar menjadi blok-blok yang lebih kecil dan menyimpannya di berbagai node dalam kluster, memungkinkan pemrosesan data secara paralel dan meningkatkan keandalan melalui replikasi blok. Namun, ada beberapa kelemahan dalam sistem ini, termasuk waktu replikasi yang lama dan kinerja yang dapat terpengaruh oleh penggunaan disk[6]. Dalam banyak kasus, pemrosesan yang dilakukan melalui HDFS dapat menjadi *bottleneck*, terutama ketika volume data sangat besar dan diperlukan kecepatan pemrosesan yang tinggi.

Menanggapi keterbatasan ini, Apache memperkenalkan Apache Spark, yang menawarkan teknologi Resilient Distributed Datasets (RDDs) untuk memproses data secara lebih efektif dan cepat. Dengan memanfaatkan pemrosesan in-memory,Apache Spark mampu meningkatkan kecepatan *read* dan *write* data, serta memberikan efisiensi yang lebih baik dalam mengelola beban kerja besar. Penggunaan memori dalam Apache Spark menjadi faktor kunci yang menentukan kecepatan pemrosesan data, di mana semakin besar kapasitas memori, semakin cepat proses dapat dilakukan. Namun, jika memori yang tersedia tidak sebanding dengan data yang akan diproses, penggunaan Hadoop Mapreduce masih bisa lebih cepat dibandingkan Spark dalam beberapa situasi tertentu [7].

Dalam artikel ini, penulis akan mengimplementasikan penggunaan Hadoop Mapreduce dan Apache Spark dalam mengolah beberapa dataset yang berbeda, guna mengevaluasi kecepatan dan efisiensi masing-masing engine dalam pengolahan data, menggunakan tools Hgrid247-DE Seperti perbedaan cara Hadoop Mapreduce dan Apache Spark mengolah data kecil dari 3 MB sampai dengan 200 MB dan beberapa skenario pemrosesan data yang bentuk kata,angka dan tanggal,Devachree(2022) spark dikenal lebih cepat dalam pemrosesan data, terutama ketika data dapat dimuat dalam memori (RAM). Spark bisa berjalan 100 kali lebih cepat di memori dan 10 kali lebih cepat di disk dibandingkan dengan MapReduce, karena Spark memanfaatkan caching di memori selama pemrosesan. Sebaliknya, MapReduce harus menulis setiap langkah hasil kembali ke disk, yang mengakibatkan penurunan kinerja untuk tugas yang membutuhkan banyak siklus baca/tulis.

Melalui percobaan yang berfokus pada Extract, Transform, Load(ETL) pada pengolahan data ini, diharapkan dapat diperoleh wawasan yang lebih mendalam mengenai kelebihan dan kekurangan masing-masing framework dalam konteks pengolahan tersebut, serta memberikan panduan bagi praktisi dan peneliti yang ingin memilih alat yang tepat untuk kebutuhan mereka.

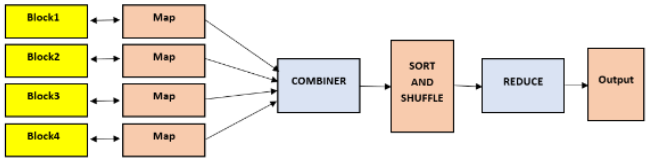
II. METODE PENELITIAN

Pada bagian ini akan dijelaskan bagaimana langkah - langkah yang dilakukan pada penelitian ini,seperti penjelasan dan cara kerja Hadoop maprecude,Apache Spark dan Hgrid 247 DE. Pada bagian ini juga dijelaskan Langkah Langkah pemrosesan data seperti persiapan data uji dan pembuatan program uji.

A. Konsep Dasar

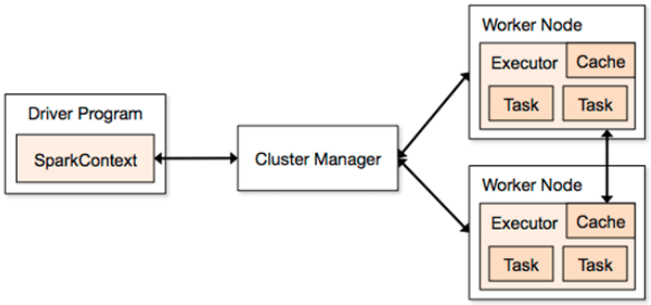
1) *Hadoop.* Salah satu software yang biasa digunakan untuk pengolahan data adalah Hadoop. Hadoop merupakan library software yang menyerupai framework open source dari bahasa pemrograman Java di bawah lisensi Apache yang digunakan untuk melakukan pemrosesan Data menggunakan model pemrograman sederhana. Hadoop telah banyak dikenal oleh perusahaan-perusahaan besar seperti Microsoft, Oracle, IBM dan sejenisnya. Secara ringkas Hadoop adalah software yang mampu menghubungkan banyak komputer untuk dapat bekerja sama dan saling terhubung untuk menyimpan dan mengelola data dalam satu kesatuan.[6]

2) *Mapreduce.* Model pemrograman yang diperkenalkan oleh Google untuk memproses dan menghasilkan dataset besar secara terdistribusi. Jeffrey Dean dan Snjay Ghemawat(2004) menekankan bahwa Mapreduce dirancang untuk memprosesdata dalam skala besar secara paralel di di kluster computer besar [8]. Gambar 1 menjelaskan komponen utama dalam Mapreduce yaitu Map dan Reduce. Data inputan yang bervolume besar akan dipecah menjadi blok blok yang lebih kecil berupa pasangan kunci-nilai ysng dapat disimpan dalam HDFS. Selanjutnya Map akan bekerja dengan cara mengirimkan blok blok data inputan tadi menuju mapper-mapper yang berjalan secara pararel dan akan menghasilkan berupa output berupa pasangan kunci-nilai baru. Setelah fase map selesai,output dari mapper yang berupa pasangan kunci-nilai akan diurutkan dan dikelompokan berdasarkan kunci yang sama yang biasanya disebut proses shuffing dan shorting untuk di proses pada fase Reduce,yaitu proses menggabungkan data berdasarkan kunci dari fase shuffling dan sorting. Setelah fase Reduces selesai,hasil akhir dari pengolahan data akan disimpan di HDFS.



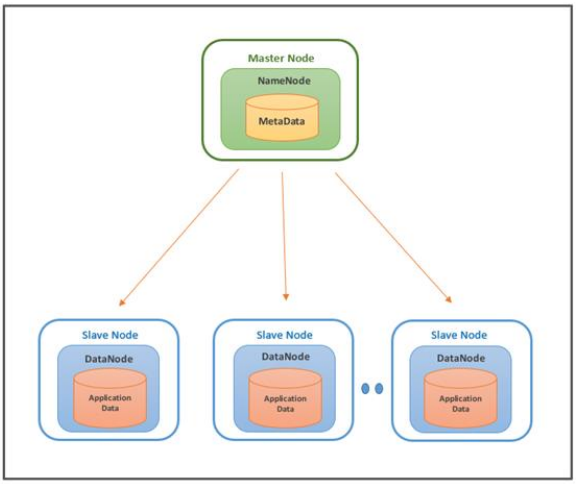
Gambar 1. Komponen Hadoop Mapreduce

3) *Apache Spark:* Apache Spark bekerja dengan cara yang efisien dan scalable untuk tugas pemrosesan data di lingkungan distribusi. Dalam lingkungan klaster, Spark memanfaatkan model pemrosesan data in-memory, yang berarti data disimpan dalam memori RAM daripada di disk. Ini mengurangi latensi akses data dan meningkatkan kecepatan pemrosesan secara signifikan. Apache Spark memiliki arsitektur berupa master dan slave (disebut driver dan worker) yang diperantarai oleh sebuah cluster manager. Cluster manager dapat menggunakan standalone, yarn, atau mesos[9]. Secara garis besar, arsitektur Apache Spark dapat dilihat pada Gambar 2.



Gambar 2. Arsitektur Apache Spark

4) *HDFS:* Hadoop Distributed File System (HDFS) adalah sistem penyimpanan terdistribusi yang dirancang untuk menyimpan dan mengelola data besar secara efisien dalam lingkungan komputasi terdistribusi.

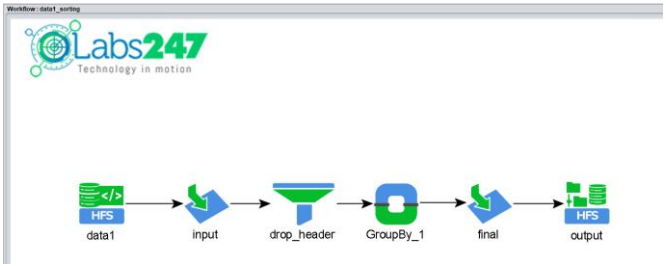


Gambar 3. Komponen HDFS

HDFS membagi file besar menjadi blok-blok yang lebih kecil dan menyimpannya di berbagai node dalam kluster, memungkinkan pemrosesan data secara paralel dan meningkatkan keandalan melalui replikasi blok[10]. HDFS dioptimalkan untuk akses data yang cepat dan dapat menangani kegagalan node, menjadikannya ideal untuk aplikasi data seperti analisis dan pemrosesan data besar. HDFS juga menyediakan mekanisme untuk memantau dan mengelola penyimpanan data, serta mendukung skala horizontal dengan menambah node baru sesuai kebutuhan.

5) *Hgrid247- DE:* HGrid247 Data Engineering adalah tools untuk melakukan data engineering di atas platform data. HGrid247 DE membantu data engineer untuk melakukan pembersihan data, integrasi antar data, transformasi, dan reformating data untuk digunakan dalam proses berikutnya, seperti analisa data, pembelajaran mesin, dan aktivitas data science lainnya[11-16]. HGrid247 DE mendukung berbagai framework data terkemuka, di antaranya adalah Map Reduce, Apache Spark, Apache Tez, Apache Beam dan Apache Flink. Tools ini juga menyediakan fitur yang memungkinkan untuk mengalihkan engine target dari satu framework ke framework yang lain.

Selain itu juga HGrid247 DE juga mendukung format parquet, avro ataupun orc yang cukup banyak digunakan dalam berbagai framework data. HGrid247 DE juga mendukung pembacaan terhadap aplikasi atau system, seperti Hive, HBase, Twitter, database melalui JDBC, MQTT dan sebagainya[8]. Pada artikel ini,Hgrid247-DE akan kita gunakan untuk membuat proses ETL (Extract, Transform, Load) yang selanjutnya kita akan jalankan menggunakan Hadoop Mapreduce dan Apache Spark. Graphical User Interface(GUI) Hgrid247-DE dapat dilihat dari gambar 3.



Gambar 4. GUI Hgrid247-DE

Dari gambar 4 diatas,dapat kita lihat sebuah workflow yang kita buat untuk melakukan sorting data,dengan menggunakan 1 data input dan akan menghasilkan 1 data output. Pada workflow diatas terdapat beberapa transformasi data,seperti filter dan sorting. Jadi nantinya Hgrid247-DE akan kita gunakan untuk menjalankan proses yang berbeda-beda dengan data yang kita miliki. Data input yang akan digunakan akan disimpan dalam HDFS dan akan kita tambahkan pada saat menjalankan pemrosesan menggunakana Hadoop Mapreduce atau Apache Spark.

B. *Konfigurasi Komputer Node*

Untuk melakukan pengolahan data menggunakan Hadoop Mapreduce dan Apache Spark kita perlu menyiapkan beberapa hal,diantaranya komputer dengan sistem operasi centos 7.9,6 core dengan RAM sebesar 29.29 GB. Pada komputer ini kita melakukan instalasi Java Runtime Environment(JRE) 1.8.0_372,Hadoop 3 dan Apache Spark 2.3.2. Memory default yang digunakan untuk menjalankan Apache Spark adalah 1 GB memory,1 core dan 512 MB untuk driver memory. Selanjutnya kita dapat mengatur kembali memori,executor dan core yang akan digunakan saat memproses data menggunakan Apache Spark.

C. *Persiapan Data Uji*

Data yang akan digunakan dalam penelitian ini merupakan dataset yang diperoleh dari platform Kaggle, dengan format file CSV dan ukuran yang bervariasi. Dataset ini mencakup beberapa ukuran yang berbeda, yaitu 3 MB, 13 MB, 63 MB, 103 MB, 225 MB, hingga 260 MB. Variasi ukuran ini memberikan tantangan tersendiri dalam hal pengolahan data, karena semakin besar ukuran dataset, semakin kompleks pula proses komputasi yang dibutuhkan. Tabel 1 akan menjelaskan karakteristik rinci dari masing-masing dataset tersebut, yang nantinya akan diolah menggunakan metode Hadoop Mapreduce dan Apache Spark untuk mengukur efisiensi kinerja kedua framework dalam mengolah data.

TABEL 1.
KARAKTERISTIK DATA INPUTAN

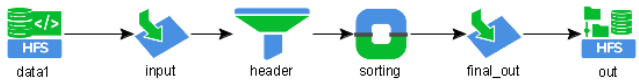
Nama	Deskripsi	Size
Data1	- Data bertipe csv dengan delimititer koma(,). - Data memiliki total baris 21698. - Data berupa Kumpulan text.	3 MB
	- Data1 memiliki kolom sebagai berikut: discipline_title,slug_game,event_title,event_gender,medal_type,participant_type,participant_title,athlete_url,athlete_full_name,country_name,country_code,country_3_letter_code	
Data2	- Data bertipe csv dengan delimititer koma(,). - Data berupa kumpulan text dan tanggal. - Data memiliki total baris 98620.	13 MB
	- Data2 memiliki kolom sebagai berikut: passenger_id,first_name,last_name,gender,age,nationality,airport_name,airport_country_code,country_name,airport_continent,continents,departure_date,arrival_airport,pilot_name,flight_status.	
Data3	- Data bertipe csv dengan delimititer koma(,). - Data memiliki jumlah baris 103064. - Data berupa kumpulan text dan tanggal.	63 MB
	- Data3 memiliki kolom sebagai berikut: Title,Authors,Description,Category,Publisher,Price,PublishDate,Publish,qty.	
Data4	- Data bertipe csv dengan delimititer koma(,). - Data memiliki jumlah baris 1200000. - Data berupa kumpulan text dan tanggal.	103 MB
	- Data5 memiliki kolom sebagai berikut: date,salesperson,customer_name,car_make,car_model,car_year,sale_price,commission_rate,commission_earned.	
Data5	- Data bertipe csv dengan delimititer koma(,). - Data memiliki jumlah baris 1048576. - Data berupa Kumpulan text,angka dan tanggal.	225 MB
	- Data6 memiliki kolom sebagai berikut: no,trans_date_trans_time,cc_num,category,amt,first,last,gender,street,city,state,zip,lat,long,city_pop,dob,trans_num,unix_time,merch_lat,merch_long,is_fraud,merch_zipcode	
Data6	- Data bertipe csv dengan delimititer koma(,). - Data memiliki jumlah baris 1048576. - Data berupan Kumpulan text,angka,dan tanggal.	260 MB
	- Data6 memiliki kolom sebagai berikut: App_Name,App_Id,Category,Rating,Rating_Count,Installs,Minimum_Installs,Maximum_Installs,Free,Price,Currency,Size,Minimum_Android,Developer_Id,Developer_Website,Developer_Email,Content_Rating,Privacy_Policy,Ad_Supported,In_App_Purchases,Editors_Choice,Scraped_Time	

D. *Pembuatan Program Uji*

Pada pengujian ini,terdapat 3 jenis proses yang akan kita gunakan untuk melihat perbandingan penggunaan Mapreduce dan Spark dalam mengolah data,yaitu sorting,summary dan grouping yang masing-masing menggunakan 6 data diatas yang akan disimpan pada HDFS. Proses ini dibuat menggunakan tools Hgrid24-DE dengan tujuan untuk mengolah data dengan beberapa proses yang berbeda,selanjutnya kita akan hasil dari proses tersebut.

1. Sorting

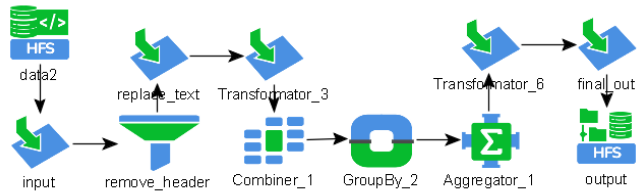
Untuk membuat proses sorting menggunakan HGrid247-DE menggunakan 6 data yang sudah kita siapkan,kita perlu membuat workflownya terlebih dahulu, Pada 6 data yang kita punya,kolom yang akan kita gunakan untuk sorting adalah dalam bentuk tanggal. Contoh workflow pada proses sorting dapat dilihat pada gambar 5.



Gambar 5. Workflow proses Sorting pada Hgrid247-DE

2. Summary

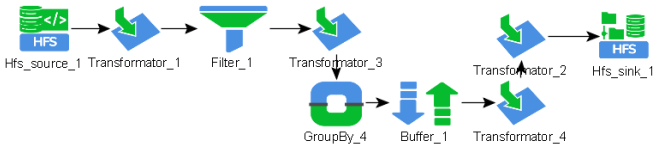
Pada pembuatan proses summary pada 6 data yang kita punya,kita akan menggunakan kolom yang berisi text dan angka,Dimana kolom text akan kita gunakan untuk proses group dan angka akan kita gunakan pada proses aggregasi berdasarkan group tersebut. Contoh workflow pembuatan proses summary yang dibuat menggunakan Hgrid247-DE dapat dilihat pada gambar 6.



Gambar 6. Workflow proses Summary pada Hgrid247-DE

3. Grouping

Pembuatan proses grouping menggunakan Hgrid247-DE disini kita membuat proses ranking berdasarkan group. Kolom yang akan kita jadikan group adalah beberapa kolom berbentuk tanggal,angka dan text yang memungkinkan untuk mengelompokkan data. Contoh workflow untuk proses grouping dapat dilihat pada gambar 7.



Gambar 7. Workflow proses Summary pada Hgrid247-DE

Setelah pembuatan workflow selesai,langkah selanjutnya kita akan menjalankan file jar yang berisi hasil compile dari workflow yang sudah kita buat dan menjalankannya menggunakan Hadoop Mapreduce dan Apache Spark dengan menggunakan perintah berikut.

```
Spark-submit                                --class
FirmaniaJurnal.data1_sorting
FirmaniaJurnal_Spark2.jar                   /user/hdp-
nifi/in/olympic_medals.csv                  /user/hdp-
nifi/in/data1_sorting          >            /home/hdp-
nifi/firmania/data1_sorting_Spark.log
```

```
hadoop                jar                   /home/hdp-
nifi/firmania/FirmaniaJurnal_mr.jar \
FirmaniaJurnal.data1_sorting \
/user/hdp-nifi/in/olympic_medals.csv \
/user/hdp-nifi/out/data1_sorting                    >
data1_sorting.log
```

III. HASIL DAN PEMBAHASAN

Setelah tahap persiapan data input dan pembuatan program uji yang terdiri dari tiga bagian utama, kita telah mendapatkan hasil dari total 108 kali percobaan menggunakan pengaturan penggunaan memori sebesar 1,2 dan 4 GB. Percobaan dilakukan dengan beberapa skenario untuk melihat performa dalam memproses data menggunakan dua engine yang berbeda, yaitu Hadoop Mapreduce dan Apache Spark. Misalnya percobaan pada setiap dataset menggunakan menggunakan memori 4 GB, dilakukan 36 kali pengujian:masing masing 18 pengujian menggunakan Hadoop Mapreduce dan 18 pengujian menggunakan Apache Spark dengan tujuan mengukur performa masing-masing framework dalam menjalankan tugas yaitu sorting, summary, dan grouping.

Pertama, untuk sorting, dataset diurutkan berdasarkan kriteria tertentu,pada data yang kita miliki,kita melakukan proses sorting menggunakan kolom yang berisi tanggal. Pada tabel 4 memperlihatkan data yang berbentuk text pada 18 kali percobaan menggunakan Hadoop Mapreduce dan 18 kali menggunakan Apache Spark menghasilkan 35 dari 36 percobaan membuktikan Apache Spark lebih cepat dalam melakukan proses sorting pada kolom yang berupa tanggal,sedangkan pada 1 percobaan proses sorting di data6 yang berukuran 260 MB menghasilkan Hadoop mapreduce lebih cepat dibandingkan Apache Spark dengan selisih 2 detik. Hal itu bisa terjadi ketika Apache Spark mengalami overhead dikarenakan tugas kecil hanya memerlukan satu atau dua operasi sederhana, proses inisialisasi Apache Spark dan pengelolaan konteks eksekusi bisa saja memakan waktu lebih lama daripada eksekusi tugas itu sendiri, sehingga menyebabkan latensi tambahan. Jadi walaupun data dan operasi yang akan kita jalankan terbilang kecil,Apache Spark bisa saja memakan waktu lebih lama karena tetap harus menginisialisasi semua komponen. Kemudian pada saat menjalankan data dengan ukuran yang cukup besar,kita bisa saja menambah pengaturan memori dan jumlah executor agar bisa meningkatkan kinerja Apache Spark, tetapi tidak selalu efektif untuk memperbaiki masalah kinerja pada tugas-tugas kecil. Berikut percobaan untuk menambah memori dan eksekutor pada Apache Spark.

```
spark-submit                                  --class
FirmaniaJurnal.data6_sorting  --num-executors  4
--executor-cores    2    --executor-memory    8G
FirmaniaJurnal_spark2.jar                   /user/hdp-
nifi/in/Google-Playstore.csv                /user/hdp-
nifi/in/data6_sorting          >            /home/hdp-
nifi/firmania/data6_sorting_spark.log
```

Pada hasil percobaan menggunakan pengaturan memori dan executor spark yaitu dengan menggunakan 8 GB executor memory dengan 4 executor yang kemudian menghasilkan 57 detik untuk memproses data,artinya 13 detik lebih cepat dari sebelumnya.

Pada table 2,3 dan 4,percobaan dengan dataset berukuran 1 GB, Apache Spark secara konsisten menunjukkan

keunggulan dalam hal waktu eksekusi. Apache Spark menyelesaikan proses Sorting dengan waktu rata-rata 2 hingga 5 kali lebih cepat dibandingkan Hadoop MapReduce. Misalnya, pada dataset pertama (data1), waktu eksekusi Sorting dengan Hadoop Mapreduce mencapai 142 detik, sementara Apache Spark hanya membutuhkan 45 detik. Untuk proses Summary dan Grouping, hasil yang serupa juga terlihat, di mana Spark lebih unggul, terutama pada proses Grouping yang menunjukkan perbedaan waktu yang signifikan, seperti pada data6 dengan waktu 166 detik pada Spark dibandingkan dengan 276 detik pada Hadoop.

Ketika memori ditingkatkan menjadi 2 GB, performa kedua framework meningkat, tetapi Apache Spark tetap menunjukkan keunggulannya. Misalnya, untuk proses Sorting, Spark menyelesaikan tugas dalam 20 hingga 88 detik, jauh lebih cepat dibandingkan Hadoop yang membutuhkan waktu antara 65 hingga 144 detik. Proses Summary dan Grouping juga menunjukkan pola yang sama, dengan Spark tetap menjadi yang tercepat.

Pada percobaan dengan memori terbesar, yaitu 4 GB, perbedaan antara kedua framework semakin jelas. Apache Spark mampu menyelesaikan Sorting dalam waktu 7 hingga 70 detik, sementara Hadoop membutuhkan waktu antara 33 hingga 74 detik. Untuk proses Summary, Spark menunjukkan kecepatan yang lebih tinggi lagi, dengan waktu yang rata-rata lebih dari dua kali lebih cepat dibandingkan Hadoop. Dalam proses Grouping, Hadoop membutuhkan waktu hingga 69 detik, sedangkan Spark hanya memerlukan waktu maksimal 38 detik.

Secara keseluruhan, percobaan ini menunjukkan bahwa Apache Spark secara konsisten lebih cepat dibandingkan Hadoop MapReduce, terutama dalam proses yang melibatkan manipulasi data seperti Sorting dan Grouping. Ini karena Spark memanfaatkan in-memory processing, yang menghindari penulisan data ke disk secara berulang seperti yang terjadi pada Hadoop Mapreduce. Ketika ukuran memori meningkat, kedua framework menunjukkan peningkatan performa, tetapi Apache Spark tetap unggul, khususnya pada dataset yang lebih besar, yang semakin mempertegas efisiensi dan kecepatan pemrosesan Spark dibandingkan dengan Hadoop MapReduce.

TABLE 2.
HASIL PERGUJIAN SPARK DAN HADOOP PADA PENGUNAAN MEMORI 1 GB

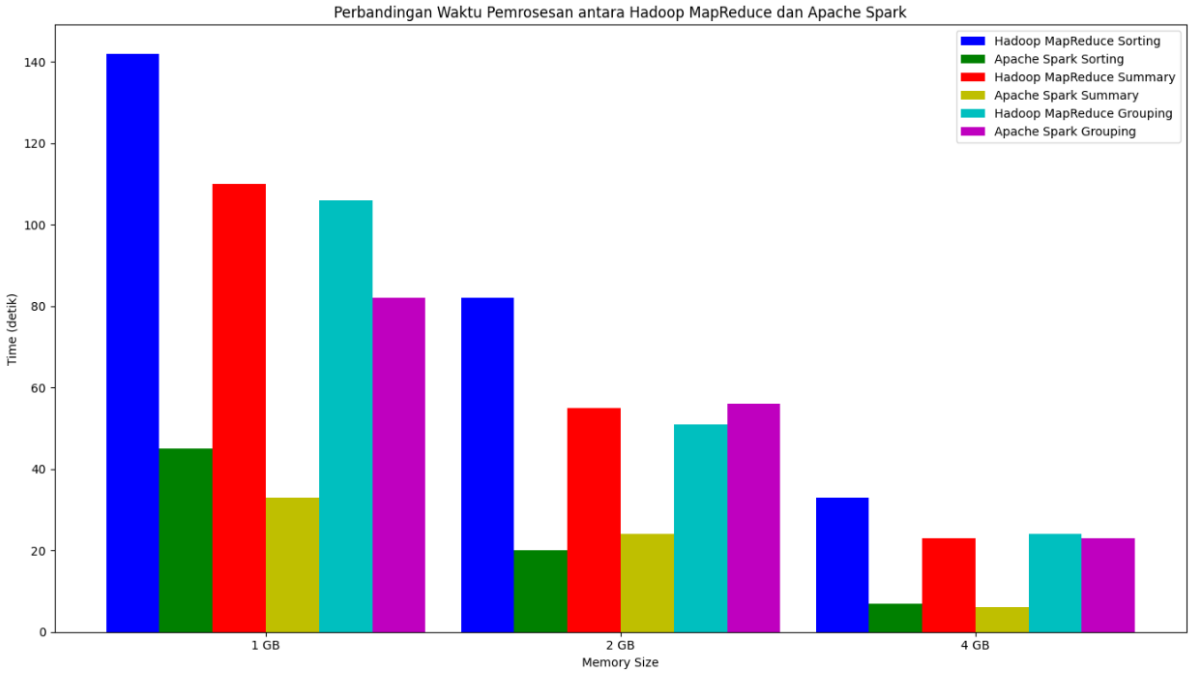
Nama	Memory	Node	Hadoop MapReduce Sorting(detik)	Apache Spark Sorting(detik)	Hadoop MapReduce Summary(detik)	Apache Spark Summary(detik)	Hadoop MapReduce Grouping(detik)	Apache Spark Grouping(detik)
data1	1 GB	3	142	45	110	33	106	82
data2	1 GB	3	168	34	165	42	146	62
data3	1 GB	3	158	73	143	56	98	48
data4	1 GB	3	221	150	160	76	236	172
data5	1 GB	3	280	154	144	70	276	150
data6	1 GB	3	305	278	198	75	276	166

TABEL 3.
HASIL PERGUJIAN SPARK DAN HADOOP PADA PENGUNAAN MEMORI 2 GB

Nama	Memory	Node	Hadoop MapReduce Sorting(detik)	Apache Spark Sorting(detik)	Hadoop MapReduce Summary(detik)	Apache Spark Summary(detik)	Hadoop MapReduce Grouping(detik)	Apache Spark Grouping(detik)
data1	2 GB	3	82	20	55	24	51	56
data2	2 GB	3	86	26	86	28	88	34
data3	2 GB	3	65	22	75	32	52	31
data4	2 GB	3	99	84	76	38	120	88
data5	2 GB	3	144	88	74	34	149	75
data6	2 GB	3	138	130	110	41	142	120

TABLE 4.
HASIL PERGUJIAN SPARK DAN HADOOP PADA PENGUNAAN MEMORI 4 GB

Nama	Memory	Node	Hadoop MapReduce Sorting(detik)	Apache Spark Sorting(detik)	Hadoop MapReduce Summary(detik)	Apache Spark Summary(detik)	Hadoop MapReduce Grouping(detik)	Apache Spark Grouping(detik)
data1	4 GB	3	33	7	23	6	24	23
data2	4 GB	3	38	8	39	8	39	13
data3	4 GB	3	36	12	33	9	24	7
data4	4 GB	3	53	36	38	17	59	41
data5	4 GB	3	74	37	34	15	69	35
data6	4 GB	3	68	70	50	14	69	38



Gambar 8. Perbandingan Waktu antara Hadoop Mapreduce dan Apache park berdasarkan memori dan proses yang digunakan

Dari hasil percobaan diatas, Hadoop MapReduce dan Apache Spark memiliki beberapa perbedaan signifikan dalam hal kecepatan, kemudahan penggunaan, serta pengolahan data seperti grouping, ranking, sorting, dan summary seperti pada tabel 5.

Untuk sorting, Hadoop MapReduce menggunakan shuffle and sort, yang memerlukan banyak operasi I/O dan waktu yang lebih lama, sementara Apache Spark mampu melakukan sorting secara in-memory, menggunakan fungsi seperti sortBy() atau orderBy(), yang jauh lebih cepat,dalam hal summary, Hadoop MapReduce memerlukan beberapa tahapan untuk melakukan berbagai agregasi seperti sum() atau avg(), sedangkan Apache Spark dapat melakukannya dengan lebih efisien menggunakan fungsi agregasi bawaan dalam satu langkah.

Untuk grouping, Hadoop MapReduce menggunakan shuffle and sort yang relatif lebih lambat, sementara Apache Spark menggunakan metode seperti groupBy() atau reduceByKey() yang lebih efisien karena dilakukan di memori. Dalam hal ranking, Hadoop Apache Spark mendukung fungsi-fungsi seperti rank(), dense_rank(), dan orderBy() secara langsung. I/O Intensive juga menjadi kelemahan Hadoop MapReduce karena sangat bergantung pada disk, sedangkan Apache Spark meminimalkan akses I/O dengan memproses data di memori.

Secara keseluruhan,Apache Spark unggul dalam hal efisiensi dan kecepatan dibandingkan Hadoop MapReduce, terutama untuk pekerjaan yang melibatkan operasi memori dan agregasi data besar,namun dalam beberapa kasus misalnya di mana dataset jauh lebih besar daripada kapasitas memori yang tersedia, Hadoop MapReduce dapat lebih cepat seperti dalam pengolahan log server web yang berukuran terabyte, Hadoop MapReduce akan mengolah data secara bertahap dari disk, menghindari masalah yang mungkin muncul di Spark, seperti out of memory,kemudian dalam lingkungan di mana kegagalan node sering terjadi, Hadoop

MapReduce memiliki mekanisme yang efisien untuk memulai ulang tugas yang gagal. Misalnya, jika ada masalah dalam pengolahan batch data dalam pengaturan Hadoop MapReduce, hanya bagian yang gagal yang diulang, sedangkan Apache Spark mungkin harus mengulang lebih banyak pekerjaan karena sistem lineage yang panjang.

TABEL 5.
PERBEDAAN CARA HADOOP MAPREDUCE DAN APACHE SPARK DALAM MENGOLAH DATA.

Aspek	Hadoop MapReduce	Apache Spark
Kecepatan	Lebih lambat karena bergantung pada disk	Lebih cepat karena in-memory processing
Kemudahan Penggunaan	Kompleks, membutuhkan banyak tahapan	Lebih mudah dengan API bawaan
Sorting	Menggunakan **shuffle and sort**, prosesnya lambat karena melibatkan disk I/O	Dilakukan secara in-memory, menggunakan sortBy() atau orderBy()
Summary	Membutuhkan beberapa tahapan MapReduce untuk berbagai agregasi	Dapat dilakukan langsung dengan fungsi agregasi bawaan seperti sum(), avg(), dll.
Grouping	Menggunakan tahap shuffle and sort	Menggunakan groupBy() atau reduceByKey()
Ranking	Harus diimplementasikan manual	Mendukung rank(), dense_rank(), dan orderBy() secara native

Pada saat menjalankan job pararel,Hadoop Mapreduce menggunakan proses map dan reduce yang dapat diterapkan secara terditribusi. Pada setiap proses mapping dan proses reducing bersifat independent sehingga proses dapat dijalankan secara paralel pada waktu yang sama, selama output dari proses mapping mengirimkan key value yang sesuai dengan proses reducingnya. Ketika Hadoop MapReduce dijalankan secara paralel dan disk penyimpanan sudah penuh, dan tidak terdapat ruang untuk menulis output dari tugas map atau reduce, job MapReduce akan masuk kedalam antrian(queue) yang nantinya job akan dijalankan ketika sudah terdapat ruang job baru dan akan dieksekusi saat ruang disk udah cukup.

Sedangkan pada Apache Spark akan menjalankan job secara paralel dengan membagi data menjadi bagian-bagian kecil yang disebut partitions. Setiap partition diproses secara independen oleh executor yang berjalan di berbagai node dalam cluster. Ini memungkinkan Apache Apache Spark mengalokasikan executor berdasarkan kebutuhan dan kapasitas yang ada, sehingga beberapa task dapat berjalan secara bersamaan.

Ketika Apache Spark menghadapi masalah kehabisan memori, ia memiliki beberapa mekanisme untuk mengatasinya. Pertama, Apache Spark dapat menyimpan intermediate data ke disk menggunakan proses yang disebut spill. Ini terjadi ketika memori hampir penuh dan Apache Spark perlu mengeluarkan data yang tidak lagi dibutuhkan dari memori untuk memberi ruang bagi data baru. Kedua, Apache Spark memungkinkan pengaturan tingkat penyimpanan untuk RDD (Resilient Distributed Dataset), sehingga Anda dapat memilih untuk menyimpan data di memori, disk, atau kombinasi keduanya. Selain itu, Spark memiliki fitur Garbage Collection yang membersihkan data yang tidak diperlukan setelah job selesai. Dengan berbagai opsi ini, Apache Spark dapat mengelola penggunaan memori dengan lebih baik dan mencegah kehabisan memori saat menjalankan job paralel.

Pada tabel 6 kita bisa melihat perbedaan proses ETL (Extract, Transform, Load) antara Hadoop MapReduce dan Apache Spark berdasarkan jenis proses ETL yang lebih cocok untuk masing-masing framework.

TABLE 6.
PERBEDAAN PROSES ETL (EXTRACT, TRANSFORM, LOAD) ANTARA HADOOP MAPREDUCE DAN APACHE SPARK

Jenis Proses ETL	Hadoop MapReduce	Apache Spark
Extract (Ekstraksi data dari berbagai sumber besar)	**Cocok untuk pengolahan data yang sangat besar dan tersebar**, seperti data log server atau data dalam HDFS. MapReduce dapat mengekstrak data secara batch dengan baik dari sumber berbasis file besar.	**Lebih cepat dan efisien untuk ekstraksi data real-time atau semi-real-time** dari berbagai sumber data termasuk API dan streaming data.
Transform (Pembersihan dan penggabungan data besar)	**Lebih baik untuk pembersihan data batch besar secara paralel**, seperti agregasi besar-besaran atau transformasi yang tidak memerlukan banyak iterasi. Ideal untuk tugas-tugas seperti pengurutan data atau agregasi yang tidak memerlukan kecepatan tinggi.	**Lebih cepat untuk transformasi kompleks dan berulang yang melibatkan banyak iterasi**, seperti analisis data machine learning atau pembaruan model, di mana Spark memanfaatkan in-memory processing yang membuatnya jauh lebih cepat dibandingkan MapReduce.
Load (Menyimpan data ke target seperti database)	**Efisien untuk penulisan batch ke database skala besar** atau penyimpanan data terstruktur yang besar. Proses MapReduce menulis hasil ke disk setelah setiap tahap, sehingga ideal untuk proses batch yang memerlukan penyimpanan data setelah transformasi besar.	**Lebih baik untuk penulisan real-time ke data store** seperti HBase, Cassandra, atau Redis. Spark juga mendukung koneksi langsung ke data stores dan proses loading cepat untuk tugas-tugas real-time atau semi-batch tanpa overhead menulis ke disk.
Pemrosesan Batch Besar	**MapReduce unggul dalam pemrosesan batch berskala besar**, terutama ketika dataset tidak dapat masuk ke dalam memori. Ideal untuk mengolah data besar secara paralel tanpa memerlukan real-time processing.	**Spark juga mendukung pemrosesan batch, tetapi lebih unggul dalam batch kecil hingga menengah** yang dapat ditangani secara in-memory, sehingga jauh lebih cepat untuk batch yang berukuran sedang.

Pemrosesan Real-Time	**Tidak cocok untuk pemrosesan real-time** karena sifat batch-nya dan ketergantungan pada disk, menyebabkan latensi tinggi untuk data yang perlu diproses dalam waktu nyata.	**Spark lebih unggul dalam pemrosesan real-time**, terutama dengan integrasi Spark Streaming, yang memungkinkan pengolahan data langsung dari sumber seperti Kafka atau Flume, dengan latensi rendah.
Iterasi Berulang	**Kurang efisien untuk tugas yang membutuhkan iterasi berulang**, seperti training model machine learning atau analisis regresi, karena setiap iterasi perlu membaca dan menulis data kembali ke disk.	**Sangat cocok untuk tugas yang melibatkan iterasi berulang** karena Spark memproses data langsung di memori, sehingga menghindari overhead yang signifikan dari penulisan data ke disk.
Skalabilitas & Toleransi Kegagalan	**MapReduce sangat baik dalam skalabilitas dan toleransi kegagalan** dengan arsitektur berbasis disk yang stabil dan dukungan untuk pengolahan data dalam node-node terdistribusi. Cocok untuk data skala besar yang memerlukan keandalan tinggi.	**Spark juga memiliki skalabilitas yang baik, tetapi toleransi kegagalan lebih lemah** dalam beberapa kasus, terutama ketika ada keterbatasan memori. Namun, mekanisme checkpointing Spark dapat membantu mengurangi risiko kehilangan data dalam pemrosesan real-time.

Dalam mengimplementasikan Hadoop Mapreduce dan Apache Spark dalam proyek big data yang sesungguhnya,ada beberapa hambatan yang biasanya kita temui,misalnya dalam pemrosesan data yg komplek,Hadoop MapReduce lambat untuk pekerjaan yang memerlukan banyak iterasi (seperti machine learning),sedangkan Apache Spark lebih baik untuk iterasi, tetapi bisa lambat jika pengelolaan memori tidak efisien.

Hadoop MapReduce cenderung memiliki latensi yang lebih tinggi dibandingkan dengan Apache Spark. I/O disk yang sering bisa mengurangi throughut, terutama saat dataset yang sangat besar dikelola. Sedangkan pada Apache Spark meskipun lebih cepat karena operasinya yang in-memory, Apache Spark masih menghadapi tantangan dalam hal latensi ketika ukuran dataset jauh melampaui kapasitas memori yang tersedia, memaksa penggunaan disk.

Dalam infratruktur,Apache Spark menggunakan memori (RAM) lebih banyak daripada Hadoop MapReduce, sehingga bisa memerlukan node dengan memori besar. Ini mungkin meningkatkan biaya pada level hardware, terutama jika memori yang tersedia tidak cukup dan perlu ditingkatkan.

Namun, karena operasinya berbasis in-memory, Apache Spark bisa mengurangi biaya penyimpanan disk dibanding HadoopMapReduce karena memproses data lebih cepat tanpa banyak akses disk.

IV. KESIMPULAN

Dari hasil penelitian ini,maka dapat diambil beberapa Kesimpulan. Kinerja dua framework utama dalam pengolahan data, Hadoop MapReduce dan Apache Spark, menggunakan berbagai dataset dengan ukuran yang bervariasi dan memori yang digunakan menunjukkan bahwa Apache Spark secara umum unggul dalam hal kecepatan pemrosesan. Percobaan dilakukan pada tiga tugas utama: sorting, summary, dan grouping, dengan enam dataset yang berbeda dan dua hingga empat gigabyte memori yang dialokasikan.

Pada tugas sorting, Apache Spark lebih cepat dalam kebanyakan percobaan, meskipun ada satu kasus, yaitu pada dataset terbesar dengan ukuran 260 MB (data6), di mana Hadoop MapReduce lebih cepat dengan waktu 68 detik dibandingkan Apache Spark yang memerlukan 70 detik. Keunggulan kecepatan Apache Spark dapat dikaitkan dengan arsitekturnya yang mengurangi waktu akses disk. Namun, pada kasus tugas-tugas kecil, Hadoop MapReduce seringkali lebih cepat karena overhead yang dihadapi Spark saat menginisialisasi komponen.

Dalam tugas summary, Apache Spark secara konsisten menunjukkan kinerja yang lebih baik. Proses pengelompokan dan agregasi data berjalan lebih cepat pada Spark, berkat kemampuannya memanfaatkan memori untuk memproses data langsung. Sebagai contoh, pada data1, Spark menyelesaikan tugas grouping dalam 23 detik, sedangkan Hadoop MapReduce memerlukan 24 detik. Pada tugas grouping juga, Spark menunjukkan kecepatan lebih tinggi dalam semua percobaan.

Meskipun Apache Spark lebih efisien dalam kecepatan pemrosesan ETL (Extract, Transform, Load), Hadoop MapReduce memiliki keunggulan dalam menangani data yang lebih besar dari kapasitas memori yang tersedia. Misalnya, dalam pemrosesan data log server yang berukuran melebihi kapasitas memori, Hadoop MapReduce bisa lebih efisien karena menggunakan disk-based processing. Dalam kasus tersebut, proses MapReduce yang menulis hasil ke disk di setiap tahap dapat lebih stabil dan menghindari potensi kegagalan yang mungkin terjadi ketika Spark kehabisan memori pada tugas-tugas besar.

Setiap framework memiliki kekuatan dan kelemahan masing-masing, sehingga pemilihan framework yang tepat sangat bergantung pada kebutuhan spesifik terkait dengan ukuran data, kompleksitas tugas, serta ketersediaan sumber daya komputasi, terutama memori. Hasil penelitian ini dapat menjadi panduan praktis bagi peneliti dan profesional dalam memilih alat pengolahan data yang sesuai untuk kebutuhan proyek big data.

DAFTAR PUSTAKA

[1] S. M. Metev & V. P. Veiko, Laser Assisted Microtechnology, 2nd ed., R. M. Osgood, Jr., Ed. Berlin, Germany: Springer-Verlag, 1998.

[2] E. Ramadhan, "Analisis Perbandingan Performa Apache Spark dan Hadoop Mapreduce pada Mapreduce Framework Menggunakan Algoritma Support Vector Machine," Sarjana thesis, Universitas Siliwangi, 2023.

[3] Mayer-Schönberger and K. Cukier, Data: A Revolution That Will Transform How We Live, Work, and Think. New York: Eamon Dolan/Houghton Mifflin Harcourt, 2013.

[4] A. Wibowo, Teori Ekonomi Berbasis Data. Semarang, Indonesia: Universitas Sains & Teknologi Komputer (Universitas STEKOM), 2023.

[5] P. A. T. Taqwin, A. B. Osmond, and R. Latuconsina, "Implementasi Metode Mapreduce Pada Data Berbasis Hadoop Distributed File System," Program Studi S1 Sistem Komputer, Fakultas Teknik Elektro, Universitas Telkom, 2023.
[6] F. S. Muhammad, "Analisis Implementasi Sistem Informasi Manajemen di Sekolah Menengah Pertama 1 Batam," *Jurnal Infotek*, vol. 6, no. 1, pp. 39-48, 2023.
[7] R. A. Rahman, A. K. Widiastuti, dan M. A. N. E. Syafri, "Pengaruh Penerapan Metode Pembelajaran Kooperatif Tipe STAD terhadap Hasil Belajar Siswa pada Materi Kesehatan," *Jurnal Teknik Industri*, vol. 20, no. 2, pp. 205-214, 2022.
[8] A. R. D. L. Raj, M. C. K. L. K. K. and N. R. B. K., "Introduction to Hadoop for Data," *International Journal of Computer Applications*, vol. 179, no. 45, pp. 1-6, 2018.
[9] R. J. K. Dwianto and D. P. Sari, "Implementasi Apache Spark pada Data Berbasis Hadoop Distributed File System," *Jurnal Teknik Informatika*, vol. 10, no. 2, pp. 100-107, 2022.
[10] P. A. T. Taqwin, A. B. Osmond, dan R. Latuconsina, "Implementasi Metode Mapreduce pada Data Berbasis Hadoop Distributed File System," *Jurnal Teknik Informatika*, vol. 20, no. 2, pp. 1-10, 2023.
[11] H. B. Y. Manik, "HGRID247 Data Engineering," Data Learns 247, 2024
[12] O. Maakoul, S. Azzouzi and M. E. H. Charaf, "An Optimal Method for Testing Jobs' Execution in MapReduce Based Systems," 2023 9th International Conference on Control, Decision and Information Technologies (CoDIT), Rome, Italy, 2023, pp. 2433-2438, doi: 10.1109/CoDIT58514.2023.10284294.
[13] N. Nelmiawati, N. C. Kushardianto, A. H. Tohari, Y. P. Hasibuan, and D. E. Kurniawan, "Rancang Bangun Lab Komputer Virtual Berbasis Cloud Computing Menggunakan Openstack Pada Jaringan Terpusat," Journal of Applied Informatics and Computing, vol. 2, no. 1, Art. no. 1, Jul. 2018, doi: 10.30871/jaic.v2i1.821.
[14] D. E. Kurniawan, I. Ahmad, M. R. Ridho, F. Hidayat, and A. A. Js, "Analysis of performance comparison between Software-Based iSCSI SAN and Hardware-Based iSCSI SAN," J. Phys.: Conf. Ser., vol. 1351, no. 1, p. 012009, Nov. 2019, doi: 10.1088/1742-6596/1351/1/012009.
[15] A. Singh et al., "A Comparative Study of Bigdata Tools: Hadoop Vs Spark Vs Storm," 2023 IEEE 4th KhPI Week on Advanced Technology (KhPIWeek), Kharkiv, Ukraine, 2023, pp. 1-5, doi: 10.1109/KhPIWeek61412.2023.10311577.
[16] P. Sewal and H. Singh, "A Critical Analysis of Apache Hadoop and Spark for Big Data Processing," 2021 6th International Conference on Signal Processing, Computing and Control (ISPCC), Solan, India, 2021, pp. 308-313, doi: 10.1109/ISPCC53510.2021.9609518.