



**EVALUASI PERFORMA HADOOP DAN SPARK PADA
DIGITALOCEAN MENGGUNAKAN HIBENCH DALAM
KONFIGURASI *PSEUDO DISTRIBUTED***

TUGAS AKHIR

Dimas Wahyu Saputro

120450081

**PROGRAM STUDI SAINS DATA
FAKULTAS SAINS
INSTITUT TEKNOLOGI SUMATERA
LAMPUNG SELATAN**

2024



**EVALUASI PERFORMA HADOOP DAN SPARK PADA
DIGITALOCEAN MENGGUNAKAN HIBENCH DALAM
KONFIGURASI *PSEUDO DISTRIBUTED***

TUGAS AKHIR

Diajukan sebagai syarat untuk memperoleh gelar sarjana

Dimas Wahyu Saputro

120450081

**PROGRAM STUDI SAINS DATA
FAKULTAS SAINS
INSTITUT TEKNOLOGI SUMATERA
LAMPUNG SELATAN**

2024

LEMBAR PENGESAHAN

Tugas Akhir Sarjana dengan judul **Evaluasi Performa Hadoop dan Spark pada DigitalOcean menggunakan HiBench dalam Konfigurasi Pseudo Distributed** adalah benar dibuat oleh saya sendiri dan belum pernah dibuat dan diserahkan sebelumnya, baik sebagian ataupun seluruhnya, baik oleh saya ataupun orang lain, baik di Institut Teknologi Sumatera maupun di institusi pendidikan lainnya.

Lampung Selatan, 6 Mei 2024

Penulis,



Dimas Wahyu Saputro
NIM 120450081

Diperiksa dan disetujui oleh,

Pembimbing I,

Pembimbing II,

Tirta Setiawan, S.Pd., M.Si.
NIP. 199008222022031003

Riksa Meidy Karim, S.Kom., M.Si., M.Sc.

Disahkan oleh,
Koordinator Program Studi Sains Data
Fakultas Sains
Institut Teknologi Sumatera

Tirta Setiawan, S.Pd., M.Si.
NIP 199008222022031003

HALAMAN PERNYATAAN ORISINALITAS

Tugas Akhir ini adalah karya saya sendiri, dan semua sumber baik yang dikutip maupun dirujuk telah saya nyatakan benar

Nama : Dimas Wahyu Saputro

NIM : 120450081

Tanda Tangan :

Tanggal :

**HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI
TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS**

Sebagai civitas akademik Institut Teknologi Sumatera, saya yang bertanda tangan di bawah ini:

Nama : Dimas Wahyu Saputro

NIM : 120450081

Program Studi : Sains Data

Jurusan : Fakultas Sains

Jenis Karya : Tugas Akhir

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Institut Teknologi Sumatera **Hak Bebas Royalti Noneksklusif (Non-exclusive Royalty Free Right)** atas karya ilmiah saya yang berjudul:

**Evaluasi Performa Hadoop dan Spark pada DigitalOcean menggunakan
HiBench dalam Konfigurasi *Pseudo Distributed***

beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Noneksklusif ini Institut Teknologi Sumatera berhak menyimpan, mengalihmedia/formatkan, mengelola dalam bentuk pangkalan data (database), merawat, dan memublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencripta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Lampung Selatan

Pada tanggal : 6 Mei 2024

Yang menyatakan (Dimas Wahyu Saputro)

Evaluasi Performa Hadoop dan Spark pada DigitalOcean menggunakan HiBench dalam Konfigurasi *Pseudo Distributed*

Dimas Wahyu Saputro (120450081)

Pembimbing I Tirta Setiawan, S.Pd., M.Si.

Pembimbing II Riksa Meidy Karim, S.Kom., M.Si., M.Sc.

ABSTRAK

Kata Kunci: Kata Kunci 1, Kata Kunci 2

Thesis Title

Dimas Wahyu Saputro (120450081)
Pembimbing I Tirta Setiawan, S.Pd., M.Si.
Pembimbing II Riksa Meidy Karim, S.Kom., M.Si., M.Sc.

ABSTRACT

Keyword: Keyword 1, Keyword 2

MOTTO

PERSEMBAHAN

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

KATA PENGANTAR

(Tuliskan maksud penulisan laporan, misal “Laporan penelitian ini dimaksud kan untuk memenuhi salah ”.....Pada halaman ini mahasiswa berkesempatan untuk menyatakan terima kasih secara tertulis kepada pembimbing dan pihak lain yang telah memberi bimbingan, nasihat, saran dan kritik, kepada mereka yang telah membantu melakukan penelitian, kepada perorangan atau lembaga yang telah memberi bantuan keuangan, materi dan/atau sarana.

Cara menulis kata pengantar beraneka ragam, tetapi hendaknya menggunakan kalimat yang baku. Ucapan terima kasih agar dibuat tidak berlebihan dan dibatasi pada pihak yang terkait secara ilmiah (berhubungan dengan subjek/materi penelitian).

Tempat penyusunan TA, tgl-bln-thn
Penulis,

Dimas Wahyu Saputro

DAFTAR ISI

Halaman Judul	i
LEMBAR PENGESAHAN	ii
Halaman Pernyataan Orisinalitas	iii
Halaman Persetujuan Publikasi	iv
Abstrak	v
Abstract	vi
Motto	vii
Persembahan	viii
Kata Pengantar	ix
DAFTAR ISI	x
DAFTAR GAMBAR	xi
DAFTAR TABEL	xii
I PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	3
1.3 Tujuan	3
1.4 Manfaat	3
1.5 Batasan Masalah	3
II LANDASAN TEORI	5
2.1 Tinjauan Pustaka	5
2.1.1 Konsep <i>Big Data</i>	5
2.1.2 Text Processing dan Feature Extraction	6
2.1.3 MapReduce	6
2.1.4 Apache Hadoop	8
2.1.5 Mode Kerja Hadoop	8
2.1.6 Hadoop Distributed File System (HDFS)	9
2.1.7 Hadoop YARN	10
2.1.8 Apache Spark	10
2.1.9 Integrasi Hadoop dan Spark	11
2.1.10 DigitalOcean	12
2.1.11 Bash Script	12
2.1.12 HiBench	12

2.1.13	Beban Kerja <i>Micro Benchmark</i> dan Sumber Data	13
2.1.14	Beban Kerja WordCount	14
2.1.15	Beban Kerja Sort	15
2.1.16	Beban Kerja TeraSort	15
2.1.17	Metriks yang Dihasilkan	15
2.1.18	Word Count - Hadoop	16
2.1.19	Word Count - Spark	16
2.1.20	Sort - Hadoop	17
2.1.21	Sort - Spark	18
2.2	Penelitian Terdahulu	18
III	METODOLOGI PENELITIAN	20
3.1	Tempat dan Jadwal Kegiatan Penelitian	20
3.2	Alur Penelitian	20
3.3	Penjabaran Langkah Penelitian	21
3.3.1	Identifikasi Masalah dan Studi Literatur	21
3.3.2	Membangun <i>Virtual Machine</i> di DigitalOcean	21
3.3.3	Pemasangan dan Konfigurasi Perangkat Lunak	22
3.3.4	Eksperimen	24
3.3.5	Proses Analisis dan Evaluasi Hasil Eksperimen	26
IV	Hasil dan Pembahasan	28
4.1	Hasil Penelitian	28
4.1.1	Persebaran Waktu Eksekusi pada Hadoop dan Spark	28
4.1.2	Persebaran <i>Throughput</i> pada Hadoop dan Spark	29
4.1.3	Rata-rata Waktu Eksekusi pada Hadoop dan Spark	30
4.1.4	Rata-rata <i>Throughput</i> pada Hadoop dan Spark	31
4.1.5	Rate of Change	32
4.1.6	Penggunaan CPU	32
4.1.7	Utilisasi Sistem	33
V	Penutup	43
5.1	Kesimpulan	43
5.2	Saran	43
DAFTAR PUSTAKA		44
LAMPIRAN		48
A	Pembuatan <i>Virtual Machine</i> (VM) pada DigitalOcean	49
B	Instalasi dan Konfigurasi Perangkat Lunak Prasyarat	52
C	Instalasi dan Konfigurasi Hadoop	55
D	Instalasi dan Konfigurasi Spark	60
E	Instalasi dan Konfigurasi HiBench	61

DAFTAR GAMBAR

Gambar 2.1	Beberapa Alat di Dunia Data [23]	6
Gambar 2.2	Cara Kerja MapReduce	7
Gambar 2.3	Implementasi MapReduce pada Word Count [29]	7
Gambar 2.4	Arsitektur Hadoop	8
Gambar 2.5	Mode Kerja Hadoop [34]	9
Gambar 2.6	Arsitektur HDFS [36]	10
Gambar 2.7	Arsitektur YARN [38]	11
Gambar 2.8	Komponen Spark	11
Gambar 2.9	Integrasi Spark dan Hadoop	12
Gambar 2.10	Proses yang Terjadi di HiBench [19]	13
Gambar 3.1	Jadwal Penelitian	20
Gambar 3.2	Diagram Alir Penelitian	21
Gambar 3.3	Alur Instalasi Perangkat Lunak	24
Gambar 3.4	Alur Instalasi Perangkat Lunak Prasyarat	25
Gambar 3.5	Alur Instalasi dan Konfigurasi Hadoop	25
Gambar 3.6	Alur Instalasi dan Konfigurasi Spark	26
Gambar 3.7	Alur Instalasi dan Konfigurasi HiBench	26
Gambar 4.1	Persebaran Waktu Eksekusi <i>Word Count</i> (Hadoop, Spark) . .	28
Gambar 4.2	Persebaran Waktu Eksekusi <i>Sort</i> (Hadoop, Spark)	29
Gambar 4.3	<i>Throughput Word Count</i> (Hadoop, Spark)	30
Gambar 4.4	<i>Throughput Sort</i> (Hadoop, Spark)	31
Gambar 4.5	Rata-rata Waktu Eksekusi (<i>Sort</i>)	32
Gambar 4.6	Rata-rata Waktu Eksekusi (<i>Word Count</i>)	33
Gambar 4.7	Rata-rata <i>Throughput</i> (<i>Sort</i>)	34
Gambar 4.8	Rata-rata <i>Throughput</i> (<i>Word Count</i>)	35
Gambar 4.9	dur	36
Gambar 4.10	th	37
Gambar 4.11	hadoop-spark	38
Gambar 4.12	Penggunaan CPU (<i>Sort</i>)	38
Gambar 4.13	Penggunaan CPU (<i>Word Count</i>)	39
Gambar 4.14	State (<i>Sort</i>)	39
Gambar 4.15	State (<i>Word Count</i>)	40
Gambar 4.16	Utilisasi Sistem (<i>Sort</i>) pada Input Data 100 KB	40
Gambar 4.17	Utilisasi Sistem (<i>Word Count</i>) pada Input Data 100 KB . .	41
Gambar 4.18	Utilisasi Sistem (<i>Sort</i>) pada Input Data 15 GB	41
Gambar 4.19	Utilisasi Sistem (<i>Word Count</i>) pada Input Data 15 GB . .	42

DAFTAR TABEL

Tabel 2.1	Beban Kerja pada HiBench [19]	14
Tabel 2.2	Penelitian Terdahulu	19
Tabel 3.1	Konfigurasi Perangkat Keras	22
Tabel 3.2	Perangkat Lunak yang Dibutuhkan	23
Tabel 3.3	Beban Kerja yang Digunakan	26
Tabel 3.4	Eksperimen yang Akan diuji Coba	27

BAB I

PENDAHULUAN

1.1 Latar Belakang

Perusahaan dan organisasi menghasilkan dan menyimpan data dalam skala besar setiap hari dengan tingkat pertumbuhannya yang dinamis [1]. Pertumbuhan jumlah data diperkirakan akan meningkat hingga 5x lipat pada tahun 2025 dengan *Global Datasphere* diproyeksikan tumbuh dari 33 *Zettabytes* (ZB) pada tahun 2018 menjadi 175 ZB [2] pada tahun 2025. Jumlah data tersebut membutuhkan pengolahan dengan kecepatan tinggi sehingga dapat dimanfaatkan untuk keperluan bisnis dan pengambilan keputusan [3]. Analisis data transaksi nasabah pada perbankan dapat digunakan untuk mendeteksi *fraud* dan meningkatkan keamanan [4]. Data pasien pada bidang kesehatan dapat memantau wabah penyakit dan menemukan pola pengobatan yang optimal [5]. Sementara itu, data interaksi pengguna pada *e-commerce* diolah untuk memberikan rekomendasi produk personal dan merancang strategi peningkatan penjualan [6]. Semakin besar data yang bisa ditangani, semakin banyak peluang analisis dan *value* yang bisa dihasilkan. Namun, semakin besar volume data yang harus diolah, semakin kompleks pula tantangan yang dihadapi dalam mengelolanya secara efisien dan efektif [7].

Tantangan utama dalam mengelola volume data yang besar adalah memastikan ketersediaan sumber daya komputasi yang memadai [7]. Pendekatan konvensional pemrosesan data besar seperti memperbanyak jumlah *storage* secara vertikal, dan penggunaan sistem basis data NoSQL dapat memungkinkan pengolahan data yang skalabel dan fleksibel. Namun, ketika skala dan kompleksitas data semakin meningkat, komputasi terdistribusi menjadi pilihan yang lebih tepat karena memiliki sifat *fault-tolerant*[8].

Komputasi terdistribusi adalah cara untuk mencapai paralelisme dengan menggabungkan beberapa mesin independen yang berbeda [9]. Dalam komputasi terdistribusi, data besar dibagi ke dalam sejumlah *node* atau server yang bekerja bersama-sama untuk mengolahnya [10]. Dua teknologi yang umum digunakan dalam komputasi terdistribusi ini adalah Apache Hadoop dan Apache Spark [11]. Hadoop dan Spark adalah dua platform komputasi *big data* yang paling populer dan banyak digunakan di seluruh dunia. Teknologi ini menawarkan berbagai kemampuan untuk mengelola, menyimpan, dan menganalisis data dalam skala besar.

MapReduce adalah alat yang digunakan untuk komputasi terdistribusi, dirancang khusus untuk menulis, membaca, dan memproses jumlah data yang besar [12]. Pemrosesan data dalam MapReduce ini terdiri dari tiga tahap: fase *Map*, fase *Shuffle*, dan fase *Reduce*. Dalam teknik ini, berkas-berkas besar dibagi menjadi beberapa blok kecil dengan ukuran yang sama dan didistribusikan ke seluruh klaster untuk penyimpanan. MapReduce dan sistem file terdistribusi (HDFS) adalah bagian inti dari sistem Hadoop, sehingga komputasi dan penyimpanan bekerja bersama-sama di seluruh *node* yang membentuk klaster komputer [13]. Hadoop MapReduce me-

merlukan akses ke penyimpanan untuk membaca dan menulis data, sehingga dapat memperlambat proses komputasi, sehingga hadirlah Spark.

Spark, di sisi lain, menawarkan teknologi *Resilient Distributed Datasets* (RDDs) untuk mendukung proses *Map* dan *Reducing* secara lebih efektif dan cepat [14]. Spark bukan hanya alternatif Hadoop, tetapi juga menyediakan berbagai fungsi, misalnya mendukung *MLib*, *GraphX*, dan *Spark streaming* untuk analisis data besar [15]. Spark menggunakan memori untuk menyimpan data sehingga dapat mengurangi siklus baca dan tulis. Perbedaan mendasar ini mengakibatkan menarik untuk melihat perbandingan performa antara keduanya. Salah satu cara untuk membandingkan performa keduanya adalah menggunakan tolok ukur Hibench.

Tolok ukur HiBench adalah salah satu tolok ukur kinerja yang paling sering digunakan. HiBench mencakup sejumlah tugas *benchmarking* yang mencerminkan berbagai jenis pemrosesan data, seperti pengolahan batch, aliran data, *query*, atau pun *machine learning* [16]. Oleh karena itu, HiBench adalah alat yang cocok untuk mengukur dan membandingkan kinerja antara Hadoop dan Spark dalam berbagai skenario penggunaan.

Penelitian tentang evaluasi performa Hadoop dan Spark menggunakan HiBench telah beberapa kali dilakukan. Shi et al. [17] melakukan penelitian dengan dua alat yang dirancang untuk mengukur kinerja MapReduce dan Spark dalam berbagai skenario beban kerja. Penelitian tersebut mengevaluasi kinerja dalam pekerjaan *batch* dan iteratif, dengan fokus pada komponen-komponen penting seperti *shuffle*, dan *caching*. Hasil penelitian menunjukkan bahwa Spark lebih cepat daripada Hadoop dalam beberapa kasus, terutama ketika menangani tugas-tugas pemrosesan data yang lebih kecil. Namun, ketika ukuran data meningkat, Hadoop terbukti lebih efisien. Selanjutnya, perbandingan kinerja antara Hadoop dan Spark juga disorot oleh penelitian Samadi et al. [13], yang menggunakan delapan tolok ukur dari HiBench. Penelitian ini menunjukkan bahwa Spark cenderung lebih efisien ketika menangani data dalam jumlah kecil atau saat memproses tugas dalam memori, sementara Hadoop lebih sukses ketika beban kerja melibatkan operasi I/O penyimpanan yang intensif. Selain itu, penelitian oleh Satish dan Rohan [18] menyoroti perbandingan kinerja antara Hadoop dan Spark khususnya dalam konteks algoritma *K-means*. Penelitian itu menemukan bahwa Spark dapat mencapai kecepatan hingga tiga kali lipat dibandingkan Hadoop, dengan catatan bahwa performa Spark sangat bergantung pada ukuran memori yang memadai.

Berdasarkan penelitian sebelumnya, penelitian ini bertujuan untuk menyelidiki perbandingan kinerja antara Hadoop dan Spark dengan menggunakan tolok ukur HiBench dengan studi kasus tertentu. Pemahaman mendalam mengenai kekuatan dan kelemahan masing-masing teknologi dalam berbagai konteks pemrosesan data akan membuat organisasi atau peneliti dapat dengan mudah membuat keputusan yang lebih informasional. Selain itu, penelitian ini akan dilakukan dengan memanfaatkan Infrastruktur sebagai Layanan (IaaS) yang disediakan oleh DigitalOcean, memungkinkan penggunaan sumber daya komputasi dalam skala yang fleksibel dan efisien. Dengan demikian, penelitian ini akan memberikan kontribusi berharga dalam membantu pemangku kepentingan dalam pemilihan teknologi pemrosesan data dalam

lingkungan komputasi terdistribusi .

1.2 Rumusan Masalah

Adapun rumusan masalah dalam penelitian ini adalah sebagai berikut:

1. Bagaimana implementasi Hadoop, Spark, dan HiBench di DigitalOcean?
2. Bagaimana kinerja Hadoop dan Spark ketika diuji menggunakan beban kerja *Micro Benchmarks* yang disediakan oleh HiBench?
3. Bagaimana perbandingan kinerja antara Hadoop dan Spark dalam mode *pseudo distributed* dalam konteks pemrosesan data dalam skala besar dengan menggunakan tolok ukur HiBench?

1.3 Tujuan

Penelitian ini memiliki tujuan, yaitu:

1. Untuk mengetahui implementasi Hadoop, Spark, dan HiBench di DigitalOcean.
2. Untuk mengetahui kinerja Hadoop dan Spark ketika diuji menggunakan beban kerja *Micro Benchmarks* yang disediakan oleh HiBench.
3. Untuk mengetahui perbandingan kinerja antara Hadoop dan Spark dalam mode *pseudo distributed* saat memproses data dalam skala besar dengan menggunakan tolok ukur HiBench.

1.4 Manfaat

Hasil dari penelitian ini diharapkan akan memberikan manfaat sebagai berikut:

1. Penelitian ini akan memberikan informasi yang berguna bagi organisasi yang sedang mempertimbangkan pemilihan platform *Big Data*, sehingga *stakeholder* dapat membuat keputusan yang lebih terinformasi.
2. Penelitian ini akan membantu dalam memahami lebih dalam kinerja Hadoop dan Spark dalam berbagai skenario pemrosesan data.
3. Hasil dari penelitian ini dapat menjadi dasar untuk penelitian lebih lanjut dalam pengembangan dan peningkatan platform *Big Data*.

1.5 Batasan Masalah

Penelitian ini memiliki beberapa batasan yang perlu diperhatikan sebagai berikut:

1. Penelitian ini akan fokus pada perbandingan kinerja antara Hadoop dan Spark dalam mode *pseudo-distributed*.
2. Pengujian kinerja akan menggunakan HiBench, sebuah tolok ukur kinerja yang umum digunakan dalam penelitian *Big Data*.

3. Implementasi Hadoop dan Spark akan menggunakan salah satu penyedia layanan awan, yaitu *DigitalOcean*.
4. Penelitian ini akan berfokus pada aspek kinerja. Aspek lain seperti keamanan dan administrasi tidak akan dibahas secara rinci.

BAB II

LANDASAN TEORI

2.1 Tinjauan Pustaka

Penelitian ini menggunakan beberapa teori dasar supaya memperjelas proses penelitian dan memberikan pemahaman lebih lanjut. Teori-teori dasar yang berhubungan dan digunakan dalam penelitian adalah sebagai berikut.

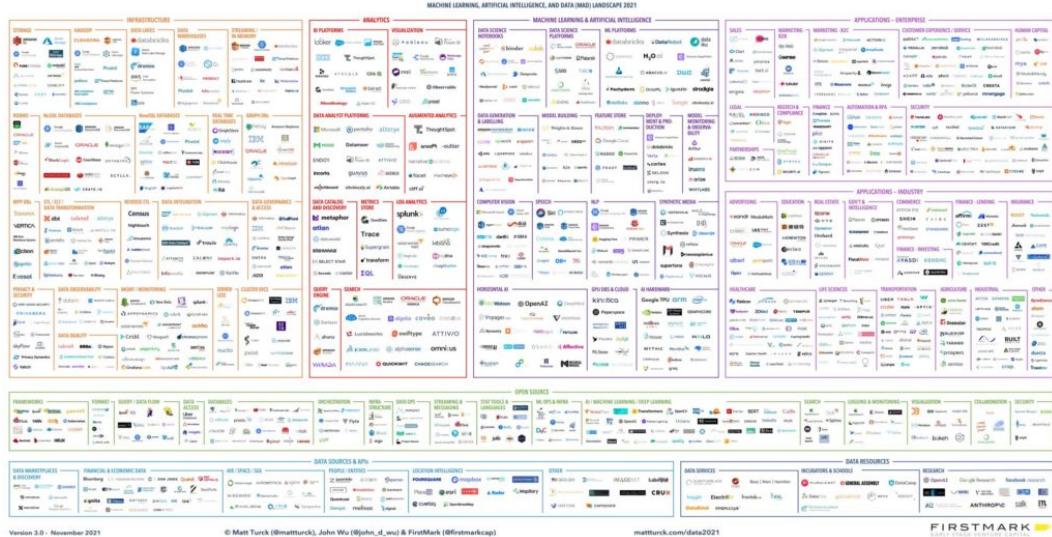
2.1.1 Konsep *Big Data*

Big Data biasanya sering didefinisikan bersama dengan kompleksitas suatu data [19]. Berbeda dengan tradisional data, *Big Data* merujuk pada pertumbuhan data dalam berbagai format, baik dari struktur, semi-terstruktur, dan tidak terstruktur [20]. *Big Data* memiliki banyak jenis sehingga membutuhkan teknologi yang lebih bertenaga serta algoritma yang lebih canggih. Pendekatan teknologi yang sering digunakan oleh *Business Intelligence* biasanya tidak dapat lagi efisien jika digunakan.

Big Data biasanya didefinisikan menjadi tiga karakteristik (3V), yaitu *Volume*, *Velocity*, dan *Variety* [21]. *Volume* berkaitan dengan jumlah data yang terbentuk atau dibuat secara terus menerus oleh beragam perangkat, seperti telepon genggam dan aplikasi (sosial media, sensor, IoT). Jumlah data diharapkan tumbuh 5x lipat pada tahun 2020 [21]. Selanjutnya, *Velocity* memberikan makna bahwa data bertumbuh secara cepat dan harus diproses secara cepat juga untuk memberikan informasi yang berguna [22]. YouTube adalah ilustrasi yang tepat untuk menggambarkan bagaimana pertumbuhan data begitu cepat. Terakhir, *Variety* berkaitan dengan variasi sumber dan format data.

Penerapan dari *big data* tidak hanya terbatas pada pengumpulan dan penyimpanan data, tetapi juga meliputi analisis dan pengolahan data tersebut untuk menghasilkan wawasan yang berguna. Beberapa sektor yang telah menerapkan *big data* secara luas meliputi kesehatan, keuangan, ritel, dan pemerintahan [20]. Dalam sektor kesehatan, *big data* digunakan untuk menganalisis informasi pasien secara massal guna meningkatkan kualitas perawatan dan menemukan pola-pola penyakit. Sementara itu, di sektor keuangan, *big data* membantu dalam analisis risiko, deteksi penipuan, dan personalisasi layanan untuk pelanggan.

Perkembangan alat dalam *big data* juga sangat pesat seperti pada Gambar 2.1. Salah satu contoh signifikan adalah penggunaan teknologi *machine learning* dan *artificial intelligence* (AI) dalam pengolahan data. AI dan *machine learning* memungkinkan analisis data yang lebih akurat dan cepat, bahkan dengan volume dan variasi data yang sangat besar. Alat seperti Hadoop dan Spark telah menjadi standar dalam industri untuk mengelola dan memproses data besar. Selain itu, penggunaan *cloud computing* dalam *big data* memungkinkan penyimpanan dan pengolahan data dalam skala yang lebih besar dan lebih fleksibel.



Gambar 2.1 Beberapa Alat di Dunia Data [23]

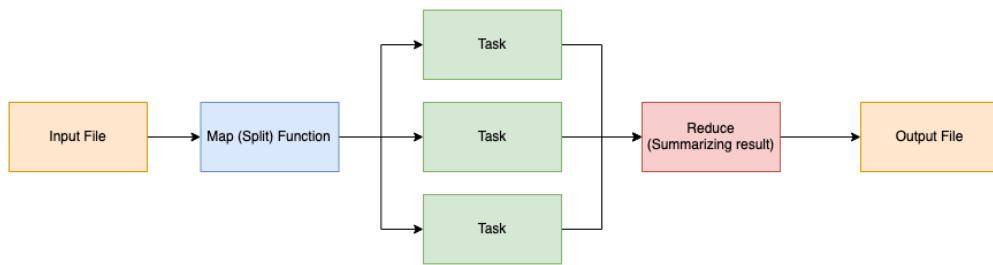
2.1.2 Text Processing dan Feature Extraction

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

2.1.3 MapReduce

MapReduce adalah model pemrograman dan implementasi teknik pemrosesan data berukuran besar yang pertama kali dipopulerkan oleh Google pada tahun 2004[24]. MapReduce menawarkan pemrosesan data yang dapat diandalkan serta *fault-tolerant manner* (tahan terhadap kesalahan). MapReduce berjalan secara paralel dan berada pada lingkungan komputasi terdistribusi [25]. Model ini mengadopsi arsitektur tersentraliasi, yaitu satu *node* berperan sebagai *master* dan *node* yang lain berperan sebagai *workers* atau *slave* [26], [27]. *Master node* bertanggung jawab untuk melakukan penjadwalan kerja, dan *slave node* berperan untuk menjalankan eksekusi kerja.

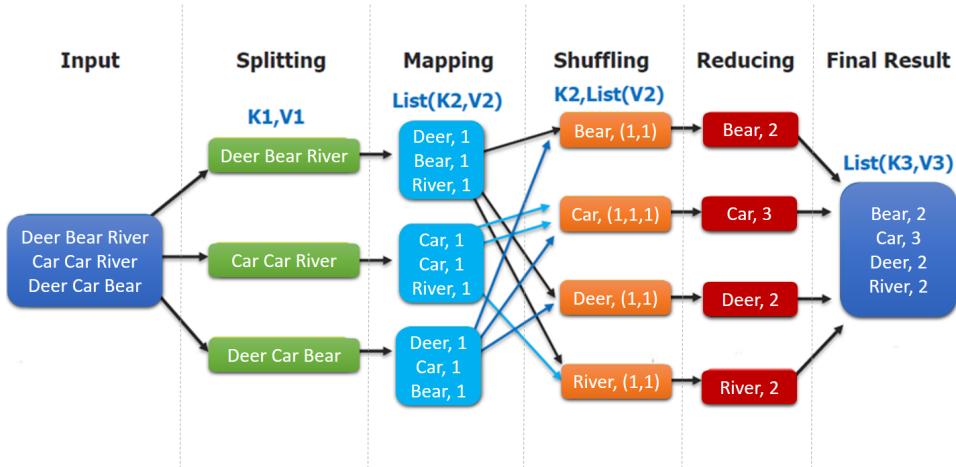
MapReduce terdiri dari fungsi *Map* dan fungsi *Reduce* [28]. Kedua fungsi ini tersebar di seluruh *slave node* yang terhubung dalam klaster dan berjalan secara paralel. Fungsi *Map* berperan untuk membagi masalah besar menjadi masalah yang lebih kecil dan mendistribusikannya ke *slave node*. Hasil pemrosesan dari *slave node* akan dikumpulkan oleh *master node* melalui fungsi *Reduce*. Sesuai dengan Gambar



Gambar 2.2 Cara Kerja MapReduce

2.2, hasil dari proses *Reduce* yang akan dikirimkan sebagai hasil akhir proses MapReduce.

Implementasi MapReduce pada *Word Count*[7] dapat dilihat pada Gambar 2.3. Pada proses MapReduce, data masukan akan melalui beberapa tahapan pemrosesan. Pertama, data akan dipecah menjadi bagian-bagian yang lebih kecil pada proses pemecahan data masukan (*splitting*). Dalam kasus Hadoop MapReduce, data idealnya akan dipecah menjadi beberapa blok berukuran maksimal 128MB.



Gambar 2.3 Implementasi MapReduce pada Word Count [29]

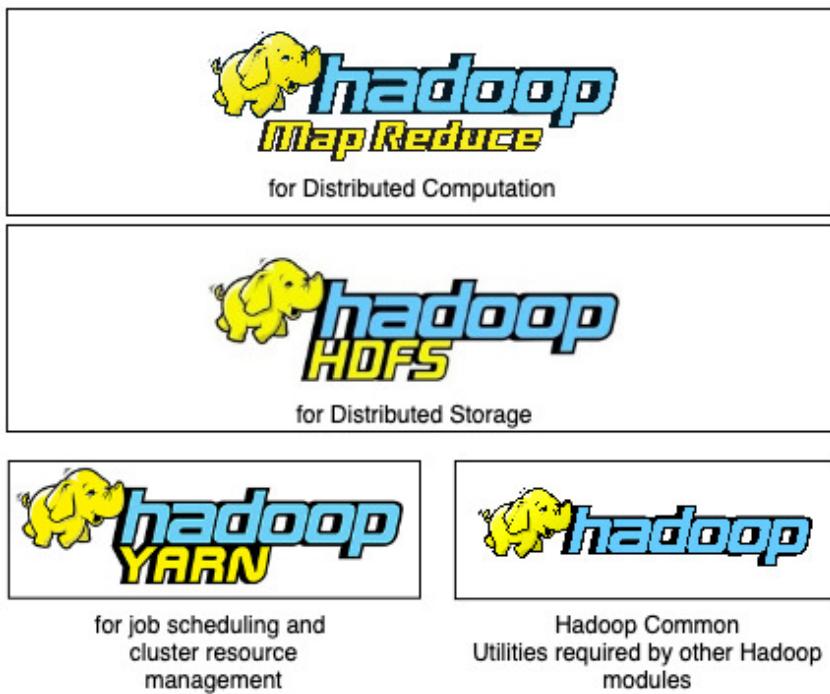
Kemudian, blok data tersebut akan diproses lebih lanjut pada tahap pemetaan (*mapping*). Pemetaan merupakan salah satu tahapan terpenting dalam MapReduce. Pada tahap ini, blok data yang sudah dipecah akan diproses untuk menghasilkan pasangan kunci-nilai (*key-value pairs*) sementara, seperti pada contoh kasus *wordcount* yang menghasilkan pasangan kunci-nilai *Dear:1*, *Bear:1*, dan *River:1*. Pemetaan dapat melibatkan satu atau beberapa mesin pekerja (*worker*) yang memproses blok data secara paralel.

Selanjutnya adalah tahap pengocokan (*shuffling*) di mana pasangan kunci-nilai hasil pemetaan yang tersebar di beberapa mesin akan dikumpulkan berdasarkan kesamaan kuncinya agar bisa diproses lebih lanjut. Misalnya semua pasangan dengan kunci *Bear* dikumpulkan dalam satu mesin.

Pada tahap terakhir yaitu pengurangan (*reducing*), dilakukan agregasi terhadap pasangan kunci-nilai dengan kunci yang sama untuk menghasilkan keluaran akhir. Seperti pada contoh kasus *wordcount*, pasangan *Bear:1* dan *Bear:1* akan dijumlahkan menjadi *Bear:2* oleh proses pengurangan.

2.1.4 Apache Hadoop

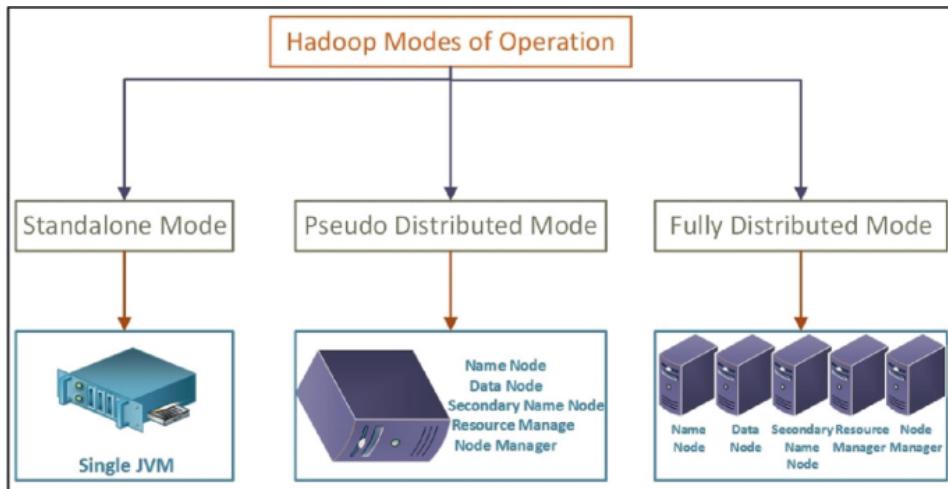
Apache Hadoop adalah perangkat lunak sumber terbuka yang ditulis dengan bahasa pemrograman Java untuk pemrosesan dan penyimpanan data menggunakan komputasi terdistribusi [30]. Hadoop dapat diinstalasi pada satu *node* komputer, atau ratusan *node* komputer yang digabungkan dalam sebuah klaster [31]. Berkaitan dengan pemrosesan data, Hadoop mengimplementasikan model MapReduce untuk pemrosesan data secara paralel dan cepat. Selain itu, Hadoop menyediakan sistem penyimpanan data terdistribusi yang dikenal sebagai Hadoop Distributed File System (HDFS) untuk akses data, pemrosesan, dan komputasi [32]. Arsitektur Hadoop secara umum dapat dilihat pada Gambar 2.4.



Gambar 2.4 Arsitektur Hadoop

2.1.5 Mode Kerja Hadoop

Hadoop dapat dijalankan dalam tiga mode operasi yang berbeda yaitu *standalone*, *pseudo-distributed*, dan *fully distributed* [33]. Dalam *standalone mode*, semua proses Hadoop berjalan pada satu node tunggal menggunakan sistem berkas lokal tanpa memerlukan konfigurasi kustom pada Hadoop seperti pada Gambar 2.5.



Gambar 2.5 Mode Kerja Hadoop [34]

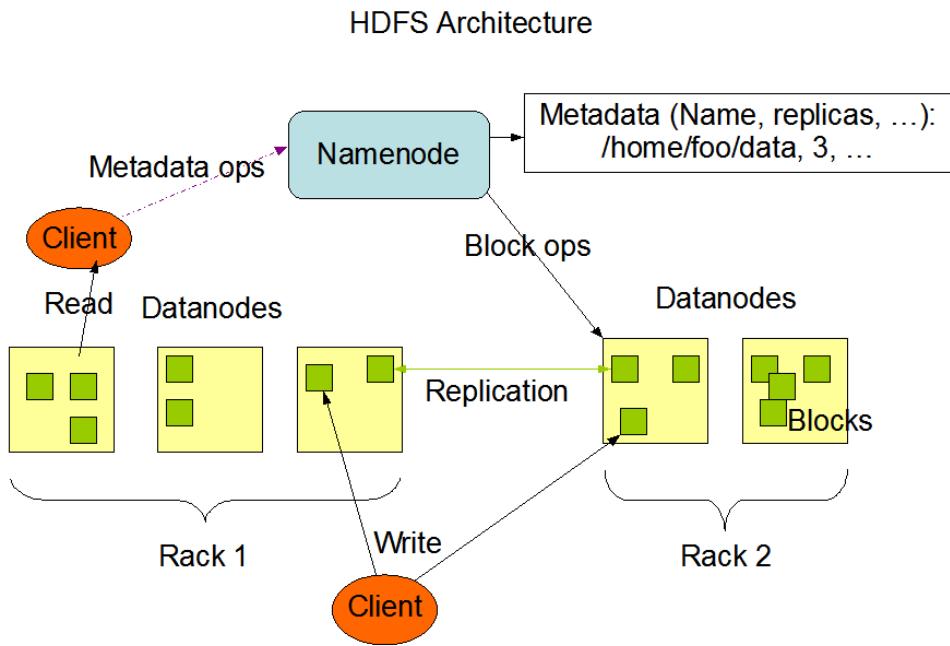
Pseudo-distributed mode menjalankan semua komponen Hadoop pada satu node tunggal tetapi menyimulasikan kluster dengan komunikasi antar proses melalui socket jaringan, sehingga memerlukan konfigurasi pada berkas *core-site*, *mapred-site*, dan *hdfs-site*. Sedangkan *fully distributed mode* menyebarkan proses Hadoop ke beberapa node dalam kluster sebenarnya yang biasanya digunakan untuk tahap produksi. *Fully distributed mode* mendukung skalabilitas, ketersediaan tinggi, dan keamanan dengan memerlukan instalasi Hadoop dan konfigurasi kluster pada setiap node.

2.1.6 Hadoop Distributed File System (HDFS)

Hadoop Distributed File System adalah sistem file terdistribusi yang dikembangkan sebagai bagian dari Hadoop [35]. HDFS dirancang khusus untuk menyimpan data dalam jumlah besar dan memungkinkan pemrosesan data secara paralel. Beberapa fitur utama dari HDFS antara lain skalabilitas, toleransi kesalahan, *streaming access*, dan cocok untuk aplikasi *batch* seperti MapReduce.

Secara struktur, HDFS terdiri dari NameNode sebagai *node master* yang mengelola *metadata* dan *namespace*, serta DataNode sebagai *node slave* yang bertugas menyimpan data sebenarnya dalam bentuk blok seperti pada Gambar 2.6. Berkas di HDFS dipartisi menjadi satu atau lebih blok berukuran 64MB atau 128MB, kemudian didistribusikan dan disimpan di beberapa DataNode. Setiap block direplikasi (biasanya 3x) di DataNode yang berbeda untuk toleransi kesalahan. Replikasi blok di *node/rack* yang berbeda juga meningkatkan ketersediaan HDFS.

Dengan desain terdistribusi, HDFS sangat populer digunakan bersama framework Hadoop untuk memproses *big data* [37]. Namun, ketergantungan pada *single* NameNode dan performa akses data acak yang kurang optimal menjadi kelemahan utama HDFS. Secara keseluruhan, HDFS telah terbukti menjadi pilihan matang untuk penyimpanan data massal secara terdistribusi.



Gambar 2.6 Arsitektur HDFS [36]

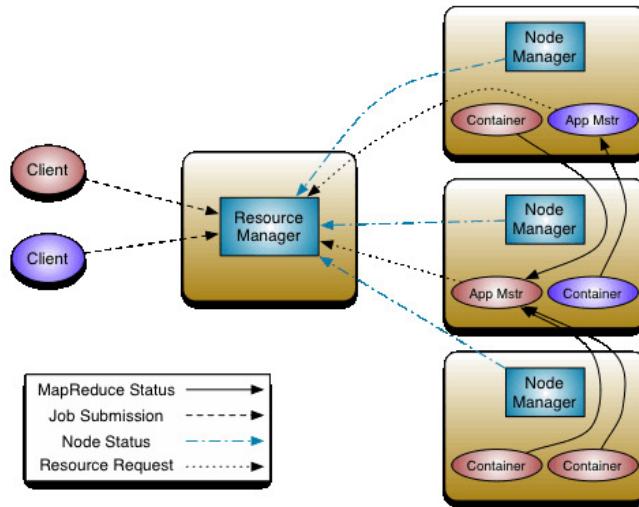
2.1.7 Hadoop YARN

Hadoop YARN (Yet Another Resource Negotiator) adalah manajer sumber daya dan sistem penjadwalan untuk kluster Hadoop. Komponen ini diperkenalkan dalam Hadoop 2.x sebagai evolusi dari Hadoop MapReduce 1.x, yang mengintegrasikan manajemen sumber daya dan pemrosesan data dalam satu sistem. YARN memungkinkan kluster untuk menjalankan berbagai aplikasi secara bersamaan dengan efisiensi yang lebih baik. YARN memisahkan fungsi manajemen sumber daya dari mekanisme pemrosesan data, yang sebelumnya keduanya tertanam dalam MapReduce. Dengan demikian, YARN dapat mendukung berbagai paradigma pemrosesan data di atas Hadoop, selain MapReduce, seperti *real-time processing* dan *graph processing*.

Struktur utama YARN terdiri dari *ResourceManager* yang bertugas mengkoordinasikan alokasi sumber daya di seluruh kluster, dan *NodeManager* yang berjalan di setiap *node* untuk mengawasi penggunaan sumber daya dan mengelola *container* tempat aplikasi dijalankan. *ApplicationMaster* adalah komponen khusus untuk setiap aplikasi yang bertanggung jawab untuk negosiasi sumber daya dengan *ResourceManager* dan bekerja dengan *NodeManager* untuk menjalankan dan memantau *tasks* seperti pada Gambar 2.7.

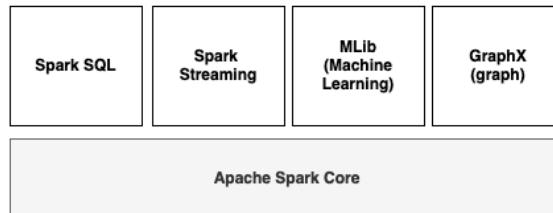
2.1.8 Apache Spark

Apache Spark diperkenalkan oleh Apache Software Foundation sebagai *framework* pemrosesan data paralel *open-source* yang dirancang untuk mempercepat pemrosesan *big data* dibandingkan dengan Hadoop MapReduce [39]. Meskipun sama-sama menggunakan model pemrosesan MapReduce, Spark bukanlah hasil modi-



Gambar 2.7 Arsitektur YARN [38]

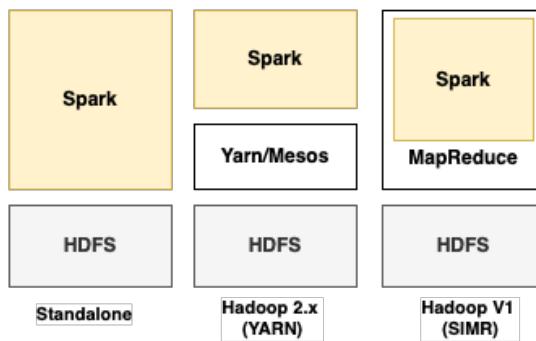
fikasi dari Hadoop MapReduce[7]. Hal ini dikarenakan Spark menggunakan teknologi tersendiri yaitu *Resilient Distributed Datasets* (RDDs) yang memungkinkan Spark memproses data secara *in-memory* sehingga lebih cepat. Selain itu, Spark memiliki klaster pengolahan data tersendiri sehingga dapat berjalan independen tanpa Hadoop. Dengan performa tinggi serta dukungan untuk pemrosesan data secara interaktif, Spark banyak digunakan untuk pemrosesan data skala besar. Komponen yang terdapat pada Spark dapat dilihat pada Gambar 2.8



Gambar 2.8 Komponen Spark

2.1.9 Integrasi Hadoop dan Spark

Integrasi Spark dengan Hadoop dapat dilakukan melalui tiga metode berbeda seperti pada Gambar 2.9 [40]. Pertama, metode *Standalone* mengharuskan Spark menempati tempat di atas HDFS (*Hadoop Distributed File System*). Dalam skenario ini, Spark dan MapReduce berjalan berdampingan untuk menangani semua pekerjaan Spark pada kluster. Kedua, metode Hadoop Yarn memungkinkan Spark berjalan pada Yarn tanpa memerlukan instalasi sebelumnya atau akses *root*. Hal ini memfasilitasi integrasi Spark ke dalam ekosistem Hadoop, atau memungkinkan komponen lain berjalan di atas integrasi Hadoop dan Spark. Terakhir, metode *Spark in MapReduce* (SIMR). Dengan SIMR, pengguna dapat memulai Spark dan menggunakan *shell*-nya tanpa memerlukan akses administratif.



Gambar 2.9 Integrasi Spark dan Hadoop

2.1.10 DigitalOcean

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

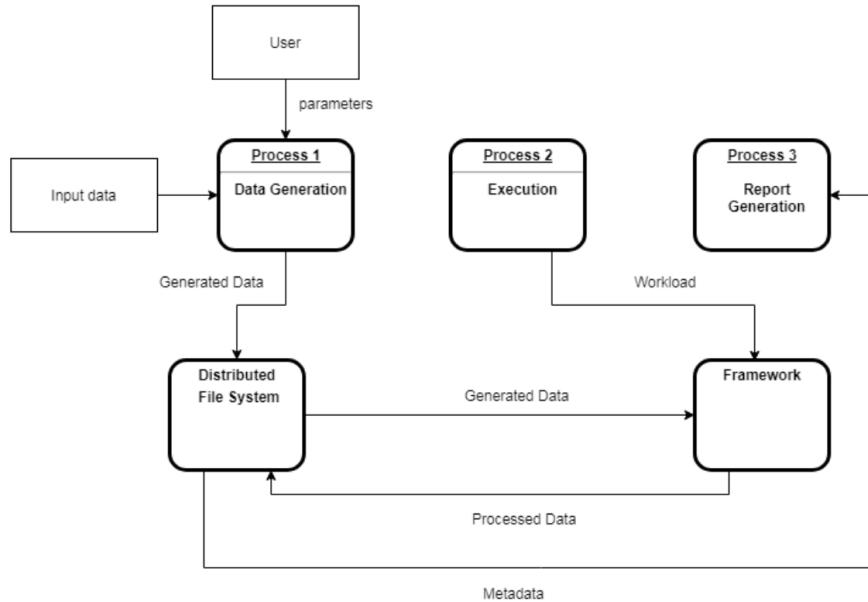
2.1.11 Bash Script

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

2.1.12 HiBench

HiBench memudahkan dalam eksekusi pengukuran berbagai beban kerja karena HiBench sudah membungkus sekumpulan perintah dalam bentuk *shell script*[1]. Pengguna hanya perlu menjalankan perintah untuk HiBench melakukan persiapan data. Selanjutnya, pengguna bisa langsung melakukan pengukuran beban kerja. Hasilnya dapat terlihat langsung pada laporan HiBench. Secara umum, alur kerja

HiBench terlihat seperti pada Gambar 2.10.



Gambar 2.10 Proses yang Terjadi di HiBench [19]

HiBench terdiri dari 3 proses utama. Proses pertama, pengguna melakukan konfigurasi parameter *Data Generation*. Selanjutnya, *Data Generation* akan melakukan pembentukan data yang nantinya akan disimpan pada *Distributed File System* (DFS). Data ini yang akan digunakan pada proses selanjutnya. Proses kedua adalah proses eksekusi. Pengguna akan memicu salah satu beban kerja pada HiBench. Selanjutnya, HiBench akan memberi perintah kepada perangkat lunak (Hadoop/Spark) untuk menjalankan beban kerja tersebut. Setiap melakukan pengukuran, data yang digunakan adalah data dari *Distributed File System* yang sebelumnya sudah dibentuk. Hasil dari eksekusi ini akan disimpan kembali di DFS. Proses terakhir adalah proses pembentukan laporan. Hasil dari proses sebelumnya akan diambil serta akan dibuatkan laporan secara otomatis. Dalam laporan otomatis yang diberikan oleh HiBench, terdapat beberapa metriks yang tersedia, meliputi *Execution Time* dan *Throughput*. *Execution Time* memiliki makna seberapa lama suatu kejadian berlangsung. Waktu yang dihitung adalah waktu diantara waktu awal dan waktu terakhir kejadian. Metriks ini dihitung dalam skala detik. Selanjutnya, *Throughput* menghitung berapa banyak unit informasi yang dapat diproses oleh sistem dalam waktu tertentu. Metriks ini dinyatakan dalam *byte/detik*.

2.1.13 Beban Kerja *Micro Benchmark* dan Sumber Data

HiBench versi 7.1 memiliki 29 beban kerja (*workload*) yang dapat diuji [41]. Beban kerja ini dikategorikan menjadi 7 kategori, yaitu *micro*, *ml* (*machine learning*), *sql*, *graph*, *websearch and streaming*. Tabel 2.1 menunjukkan macam-macam beban kerja yang dapat diuji. *Workload name* mengindikasikan algoritma utama atau operasi apa yang dilakukan. *Workload type* merepresentasikan kategori dari beban kerja. *Operation types* menunjukkan klasifikasi jenis operasi yang dilakukan.

Workload Submission Policy berguna untuk mengetahui bagaimana cara pengguna untuk mengatur atau mengonfigurasikan beban kerja.

Tabel 2.1 Beban Kerja pada HiBench [19]

Workload Name	Workload Type	Operation Type	Workload Submission Policy	Software Stack
Sort	Micro Benchmark	Algorithm	Pre-Specified Process	Hadoop, Spark
WordCount	Micro Benchmark	Algorithm	Pre-Specified Process	Hadoop, Spark
Terasort	Micro Benchmark	Algorithm	Pre-Specified Process	Hadoop, Spark
Sleep	Micro Benchmark	Algorithm	Pre-Specified Process	Hadoop, Spark
enhanced DFSIO	Micro Benchmark	IO	Parameter Control	Hadoop, Spark
Bayesian Classification	Machine Learning	Algorithm	Parameter Control	Spark
K-means clustering	Machine Learning	Algorithm	Parameter Control	Spark
Logistic Regression	Machine Learning	Algorithm	Parameter Control	Spark
Alternating Least Squares(ALS)	Machine Learning	Algorithm	Parameter Control	Spark
Gradient Boosting Trees (GBT)	Machine Learning	Algorithm	Parameter Control	Spark
Linear Regression	Machine Learning	Algorithm	Parameter Control	Spark
Latent Dirichlet Allocation	Machine Learning	Algorithm	Parameter Control	Spark
Principal Components Analysis (PCA)	Machine Learning	Algorithm	Parameter Control	Spark
Random Forest	Machine Learning	Algorithm	Parameter Control	Spark
Support Vector Machine (SVM)	Machine Learning	Algorithm	Parameter Control	Spark
Support Vector Machine(SVM)	Machine Learning	Algorithm	Parameter Control	Spark
Singular Value Decomposition	Machine Learning	Algorithm	Parameter Control	Spark
Scan, Join, Aggregate	SQL	EO	Pre-Specified Process	Hadoop, Spark
PageRank	Websearch	Algorithm	Parameter Control	Spark
Nutch indexing	Websearch	Algorithm	Parameter Control	Spark, Nutch
NWeight	Graph	Algorithm	Parameter Control	Spark(with GraphX or Pregel)
Identity	Streaming	Algorithm, IO	Parameter Control	Spark Streaming, Flink, Storm and Gearpump
Repartition	Streaming	Algorithm, IO	Parameter Control	Spark Streaming, Flink, Storm and Gearpump
Stateful Wordcount	Streaming	Algorithm, IO	Parameter Control	Spark Streaming, Flink, Storm and Gearpump
Fixwindow	Streaming	Algorithm, IO	Parameter Control	Spark Streaming, Flink, Storm and Gearpump

Beban kerja *micro benchmarks* merupakan kategori khusus yang dirancang untuk menguji kemampuan *raw processing power* [19]. Dalam kategori ini, terdapat tiga beban kerja populer, yaitu Sort, WordCount, dan TeraSort [16]. Beban kerja Sort dan WordCount merepresentasikan pekerjaan MapReduce [12]. Beban kerja Sort akan mengurutkan setiap kata dalam berkas input. Beban kerja WordCount akan melakukan tugas pemetaan (*map task*) dan mengeluarkan output (kata, 1) untuk setiap kata dalam inputnya.

Data masukan untuk beban kerja Sort dan WordCount dihasilkan menggunakan program RandomWriter dan RandomTextWriter yang nantinya akan dibuat melalui proses *Data Generation*. Sementara itu, beban kerja TeraSort akan mengurutkan data input yang dihasilkan oleh TeraGen.

2.1.14 Beban Kerja WordCount

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet,

consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

2.1.15 Beban Kerja Sort

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

2.1.16 Beban Kerja TeraSort

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

2.1.17 Metriks yang Dihasilkan

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

2.1.18 Word Count - Hadoop

Algorithm 1: Word Count MapReduce

Masukan: Berkas teks dengan setiap baris sebagai kalimat.

Keluaran: Berkas sequence dengan setiap baris sebagai (kata, jumlah).

Tahap Map:

```
for setiap baris baris dalam teks masukan do
    kata ← tokenisasi(baris);
    for setiap kata k dalam kata do
        Keluarkan(k, 1);
```

Tahap Combine (Opsional):

```
for setiap pasangan kunci-nilai (k, [j1, j2, ..., jn]) do
    jumlah ← Σi=1n ji;
    Keluarkan(k, jumlah);
```

Tahap Reduce:

```
for setiap pasangan kunci-nilai (k, [j1, j2, ..., jn]) do
    jumlah_total ← Σi=1n ji;
    Keluarkan(k, jumlah_total);
```

2.1.19 Word Count - Spark

Algorithm 2: Scala Word Count

Masukan: Berkas teks pada HDFS dengan setiap baris sebagai kalimat.

Keluaran: Berkas teks pada HDFS dengan setiap baris sebagai (kata, jumlah).

1. Inisialisasi SparkContext;
 2. *data* ← Muat data teks dari HDFS;
 3. *kata* ← Pisahkan setiap baris dalam *data* menjadi kata-kata individual;
 4. *pasangan* ← Ubah setiap kata menjadi pasangan (kata, 1);
 5. *jumlah* ← Jumlahkan nilai untuk setiap kata menggunakan ‘reduceByKey’;
 6. Simpan *jumlah* ke HDFS;
 7. Hentikan SparkContext;
-

2.1.20 Sort - Hadoop

Algorithm 3: Sort Hadoop MapReduce

Masukan: Data pada HDFS (format dapat ditentukan)

Keluaran: Data terurut pada HDFS (format dapat ditentukan)

Inisialisasi:

1. Inisialisasi konfigurasi dan JobClient;
2. Tentukan jumlah reducer berdasarkan konfigurasi dan kapasitas cluster;
3. Tentukan format input, format output, kelas kunci output, dan kelas nilai output (dapat ditentukan pengguna);
4. Buat objek Job dan atur konfigurasi dasar (nama, jar, mapper, reducer, jumlah reducer, format input/output, kelas kunci/nilai output);
5. Atur lokasi input dan output pada HDFS;
6. **if pengguna meminta total-order sort then**
 - Lakukan sampling data input;
 - Buat berkas partisi untuk total-order sort;
 - Atur konfigurasi total-order sort pada job;

Tahap Map:

for setiap masukan (*kunci, nilai*) **do**

- Emit (keluarkan) pasangan (*kunci, nilai*);

Tahap Reduce:

for setiap (*kunci, [nilai1, nilai2, ...]*) **do**

- for** setiap nilai *v* dalam daftar nilai **do**
 - Emit (keluarkan) pasangan (*kunci, v*);

Eksekusi:

7. Cetak informasi job (node, lokasi input/output, jumlah reducer);
 8. Jalankan job dan tunggu hingga selesai;
 9. Cetak waktu eksekusi job;
-

2.1.21 Sort - Spark

Algorithm 4: Scala Sort

Masukan: Berkas teks pada HDFS

Keluaran: Berkas teks terurut pada HDFS

1. Inisialisasi SparkContext;
 2. Tentukan jumlah partisi untuk pengurutan;
 3. $data \leftarrow$ Muat data teks dari HDFS dan ubah setiap baris menjadi pasangan (baris, 1);
 4. Buat objek HashPartitioner dengan jumlah partisi yang ditentukan;
 5. $terurut \leftarrow$ Urutkan $data$ berdasarkan kunci menggunakan ‘sortByKeyWithPartitioner’ dengan HashPartitioner;
 6. Ambil nilai dari setiap pasangan terurut (yaitu, hanya teks baris);
 7. Simpan data terurut $terurut$ ke HDFS;
 8. Hentikan SparkContext;
-

2.2 Penelitian Terdahulu

Penelitian terdahulu mengenai evaluasi performa Hadoop dan Spark dapat dilihat pada Tabel 2.2.

Tabel 2.2 Penelitian Terdahulu

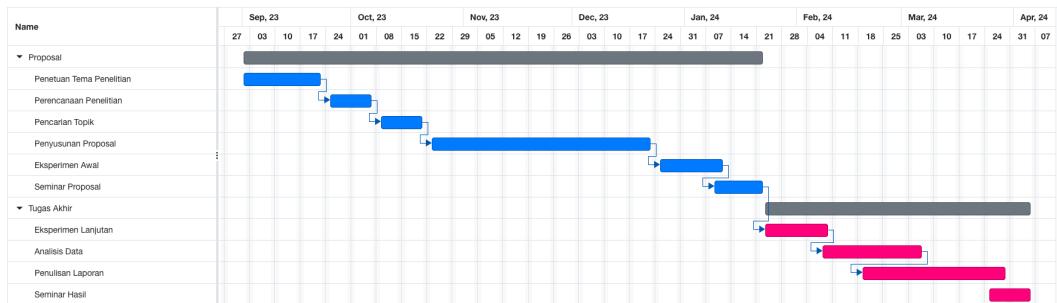
No.	Nama Peneliti	Tahun	Judul	Metode	Hasil Penelitian
1	N. Ahmed, Andre L. C. Barczak, Teo Sosnjak, Mohammed A. Rashid [10]	2020	<i>A comprehensive performance analysis of Apache Hadoop and Apache Spark for large scale data sets using HiBench</i>	Penelitian ini menyelidiki parameter-parameter yang paling berdampak, yaitu <i>input splits</i> , dan <i>shuffle</i> , untuk membandingkan kinerja antara Hadoop dan Spark, dengan menggunakan klaster yang diimplementasikan di laboratorium. Guna mengevaluasi kinerja, dua beban kerja dipilih, yakni WordCount dan TeraSort. Metrik kinerja diukur berdasarkan tiga kriteria: waktu eksekusi, <i>throughput</i> , dan <i>speedup</i> .	Kinerja kedua sistem sangat bergantung pada ukuran data masukan dan pemilihan parameter yang tepat. Analisis hasil menunjukkan bahwa Spark memiliki kinerja yang lebih baik dibandingkan dengan Hadoop ketika set data kecil, mencapai peningkatan kecepatan hingga dua kali lipat dalam beban kerja WordCount dan hingga 14 kali lipat dalam beban kerja TeraSort ketika nilai parameter default dikonfigurasi ulang.
2	Rendiyono Wahyu Saputro, Aminuddin, Yuda Munarko [11]	2020	Perbandingan Kinerja Komputasi Hadoop dan Spark untuk Memprediksi Cuaca (Studi Kasus: <i>Storm Event Database</i>)	Mengimplementasikan gugus komputer untuk memproses dataset dengan berbagai ukuran dan dalam jumlah komputer yang berbeda.	Hadoop memerlukan waktu yang lebih sedikit dibandingkan dengan Spark. Hal tersebut karena nilai <i>throughput</i> dan <i>throughput/node</i> Hadoop lebih tinggi daripada Apache Spark.
3	Yassir Samadi, Mostapha Zbakh, Claude Tadonki [1]	2018	<i>Performance comparison between Hadoop and Spark frameworks using HiBench benchmarks</i>	Perbandingan kinerja diimplementasikan pada mesin virtual (VM). Untuk membandingkannya, digunakan HiBench. Perbandingan dilakukan berdasarkan tiga kriteria: waktu eksekusi, <i>throughput</i> , dan <i>speedup</i> . Beban kerja WordCount diuji dengan ukuran data yang berbeda.	Spark lebih efisien dibandingkan Hadoop dalam menangani jumlah data yang besar. Namun, Spark memerlukan alokasi memori yang lebih tinggi, karena memuat data yang akan diproses ke dalam memori dan menyimpannya dalam cache untuk sementara.
4	Satish Gopalani, Rohan Arora [18]	2015	<i>Comparing Apache Spark and Map Reduce with Performance Analysis using K-Means</i>	Hadoop dan Spark dibandingkan menggunakan algoritma pembelajaran mesin (KMeans). Ukuran data yang digunakan adalah sebesar 64MB, 1240MB dengan satu <i>node</i> , dan 1240MB dengan dua <i>node</i> .	Hasil-hasil penelitian dengan jelas menunjukkan bahwa kinerja Spark jauh lebih tinggi dari segi waktu, di mana setiap ukuran dataset mengakibatkan penurunan waktu pemrosesan hingga tiga kali lipat dibandingkan dengan Hadoop.

BAB III

METODOLOGI PENELITIAN

3.1 Tempat dan Jadwal Kegiatan Penelitian

Dalam melaksanakan sebuah penelitian, perencanaan waktu merupakan komponen kritis yang memastikan alur penelitian dapat berjalan dengan terstruktur dan sistematis. Gambar 3.1 menyajikan jadwal penelitian yang telah dirancang untuk penelitian ini. Jadwal tersebut mencakup rentang waktu mulai dari September 2023 hingga April 2024 dan menguraikan berbagai kegiatan yang akan dilakukan selama periode tersebut. Selanjutnya, penelitian ini akan dilaksanakan di Laboratorium Komputer, Institut Teknologi Sumatera.

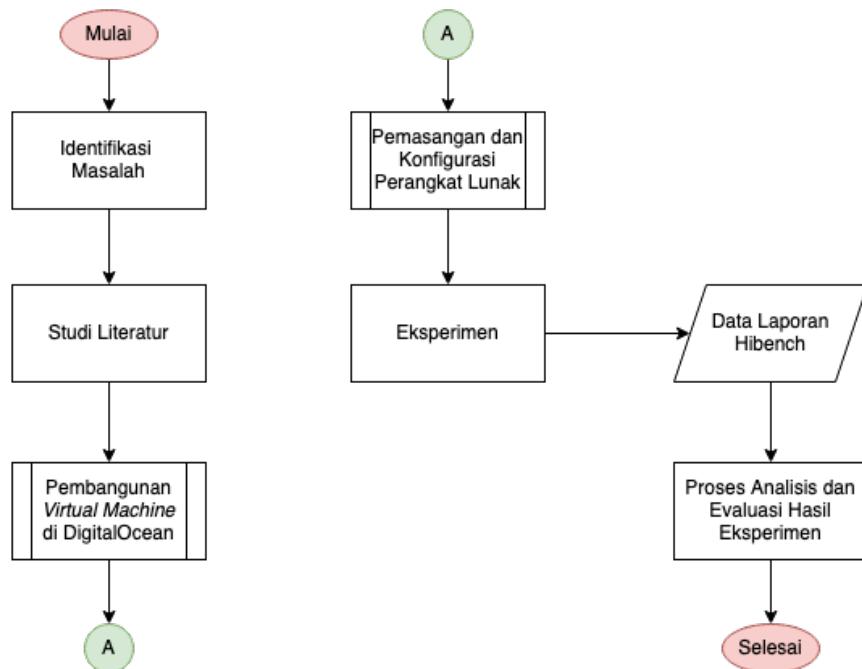


Gambar 3.1 Jadwal Penelitian

3.2 Alur Penelitian

Adapun diagram alir penelitian ini ditunjukkan pada Gambar 3.2 terdapat enam tahapan. Langkah awal yang dilakukan pada penelitian ini adalah melakukan identifikasi masalah, yaitu proses mencari, menghimpun, serta menemukan permasalahan yang nantinya akan diselesaikan. Setelah melakukan identifikasi masalah, langkah selanjutnya adalah studi literatur. Studi literatur adalah tahapan untuk mencari solusi dari permasalahan yang sebelumnya sudah kita definisikan. Pencarian solusi ini dapat melalui membaca referensi ilmiah terdahulu, baik melalui jurnal, buku, dokumentasi resmi, tesis, dan lain-lain. Tahapan ini akan memberikan pemahaman mendasar mengenai permasalahan yang sudah didapatkan sebelumnya.

Kemudian, penelitian ini akan dilanjutkan pada tahap membangun *virtual machine* di DigitalOcean. DigitalOcean adalah perusahaan penyedia layanan awan *Infrastructure as a Service* (IaaS) yang memberikan banyak pilihan kepada pengguna untuk menggunakan berbagai jenis layanan sesuai dengan kebutuhan, salah satunya yaitu *virtual machine*. *Virtual Machine* tersebut dapat dihentikan atau dihapus kapanpun saat tidak lagi diperlukan. Ketika infrastruktur sudah siap digunakan, penelitian dilanjutkan ke tahap pemasangan perangkat lunak, seperti Hadoop, Spark, dan Hi-Bench. Selanjutnya dilakukan eksperimen pada beban kerja *Micro Benchmarks*. Akhirnya, hasil dari eksperimen akan digunakan untuk proses analisis dan evaluasi.



Gambar 3.2 Diagram Alir Penelitian

3.3 Penjabaran Langkah Penelitian

Adapun untuk lebih memperjelas lagi dari setiap langkah yang ada pada Gambar 3.2, dijabarkan secara rinci tahapan-tahapan yang dilakukan pada penelitian ini.

3.3.1 Identifikasi Masalah dan Studi Literatur

Langkah awal penelitian ini adalah identifikasi masalah dan studi literatur. Identifikasi masalah dapat dipahami sebagai tahapan mendefinisikan masalah sehingga masalah tersebut dapat terukur dan jelas untuk dijadikan landasan dalam latar belakang penelitian. Setelah masalah berhasil diidentifikasi, langkah selanjutnya adalah studi literatur yang mana dalam proses ini dilakukan pengumpulan berbagai macam informasi, referensi, dan konsep dasar yang menjadi landasan dasar dari penelitian. Langkah ini dapat dilakukan melalui membaca artikel ilmiah pendukung, buku-buku yang ditulis oleh para ahli, dan jika berkaitan dengan pemrograman dapat melihat dari dokumentasi resmi. Pada tahap ini juga dilakukan analisis terhadap penelitian terdahulu dan dibandingkan dengan identifikasi masalah yang didapatkan untuk membuka celah penelitian baru sehingga penelitian ini dapat bermanfaat.

3.3.2 Membangun *Virtual Machine* di DigitalOcean

Konfigurasi perangkat keras merupakan aspek penting dalam mengevaluasi kinerja aplikasi *big data* berbasis Hadoop dan Spark. DigitalOcean, sebagai penyedia layanan infrastruktur sebagai layanan (IaaS), memberikan pengguna kebebasan penuh

untuk membuat, mengonfigurasi, dan mengelola berbagai infrastruktur yang telah disediakan. Dalam konteks penelitian ini, diperlukan penggunaan mesin virtual, yang dalam DigitalOcean dikenal sebagai "Droplets," yang memungkinkan untuk menyesuaikan berbagai aspek seperti sistem operasi, kapasitas penyimpanan, jumlah prosesor, dan parameter lainnya sesuai dengan kebutuhan spesifik penelitian.

Tabel 3.1 Konfigurasi Perangkat Keras

Nama Parameter	Nilai Parameter
Lokasi Pusat Data	Singapore - Datacenter 1 - SGP1
Sistem Operasi	Ubuntu 20.04 (LTS) x64
Jenis Droplet	Basic
Prosesor	Premium AMD - 4 Core
Memori	8 GB
Penyimpanan	160 GB NVMe SSD

Penelitian ini mengadopsi mode *pseudo-distributed* yang memungkinkan penggunaan hanya satu *virtual machine* dalam konfigurasi *single node*. Walaupun hanya menggunakan satu *virtual machine*, mode *pseudo-distributed* memungkinkan setiap proses dalam klaster beroperasi secara independen, menciptakan lingkungan di mana semua proses berjalan mandiri satu sama lain. Hal ini memungkinkan untuk lebih berfokus pada pengumpulan data dan analisis, tanpa perlu melakukan konfigurasi yang rumit terkait dengan pengaturan klaster. Spesifikasi perangkat keras yang digunakan untuk *virtual machine* dalam mode *pseudo-distributed* sesuai pada Tabel 3.1. Penjelasan lengkap tentang pembuatan *virtual machine* (VM) pada *platform* DigitalOcean dan cara mengakses VM tersebut disajikan pada Lampiran A.

3.3.3 Pemasangan dan Konfigurasi Perangkat Lunak

Pemasangan dan konfigurasi perangkat lunak merupakan hal yang krusial dalam penelitian ini. Perangkat lunak yang diperlukan ditunjukkan pada Tabel 3.2.

Alur kerja instalasi perangkat lunak dalam penelitian ini dapat dilihat pada Gambar 3.3. Pada gambar, terdapat tiga bagian utama, yaitu *prerequisites* (perangkat lunak prasyarat) ditandai dengan warna biru, alat penyimpanan dan pemrosesan *Big Data* ditandai dengan warna oranye, dan alat untuk mengukur kinerja *big data* ditandai dengan warna hijau. Semua perangkat lunak dijalankan pada sistem operasi Ubuntu 20.04 LTS x64.

3.3.3.1 Instalasi Perangkat Lunak Prasyarat

Ada beberapa perangkat lunak yang perlu diimplementasikan sebelum memasang Hadoop, Spark, Hive, dan HiBench, yaitu:

1. Ubuntu 20.04 LTS x64
2. Git
3. Java 8 dan Maven

Tabel 3.2 Perangkat Lunak yang Dibutuhkan

Perangkat Lunak	Deskripsi
Ubuntu 20.04 LTS x64	Sistem operasi Linux berbasis Ubuntu
Git	Sistem kontrol versi untuk mengelola perubahan dalam kode sumber perangkat lunak
Maven	Perangkat lunak manajemen proyek Java
Java 8	
Python 3.7	Bahasa pemrograman dasar
Scala 2.x	
Hadoop	Perangkat lunak pengolahan data terdistribusi untuk penyimpanan dan manajemen data besar
Spark	Kerangka kerja pemrosesan data terdistribusi yang berjalan di atas Hadoop
Hibench	Alat yang digunakan untuk mengukur kinerja Hadoop dan Spark

4. Python 3.7
5. Scala 2.x

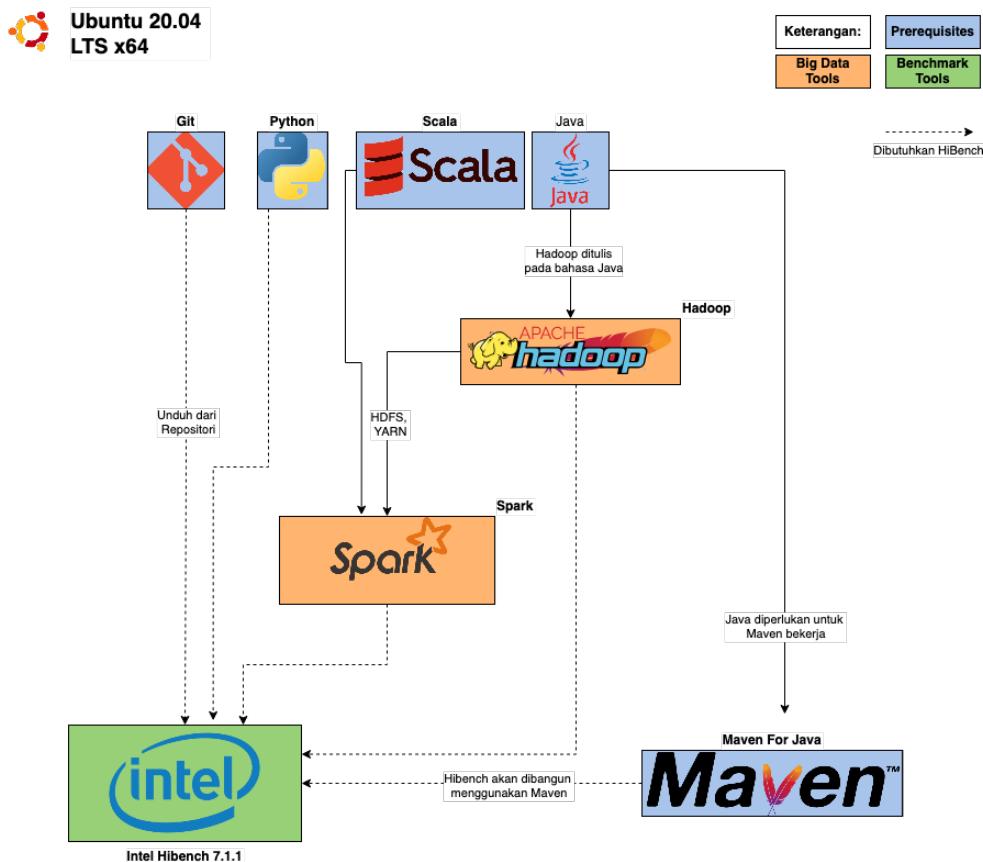
Pemasangan dan konfigurasi perangkat lunak pada tahapan ini tidak membutuhkan urutan. Akan tetapi, pada penelitian ini dibuatkan alur untuk pemasangan dan konfigurasi perangkat lunak prasyarat seperti pada Gambar 3.4. Penjelasan lengkap mengenai tata cara instalasi dan konfigurasi perangkat lunak prasyarat ini disajikan pada Lampiran B.

3.3.3.2 Instalasi dan Konfigurasi Hadoop

Hadoop adalah perangkat lunak *open source* yang efektif dalam menyimpan dan memproses data dalam skala besar. Daripada menggunakan satu komputer besar untuk menyimpan dan memproses data, Hadoop memungkinkan pengklasteran beberapa komputer untuk menganalisis set data besar secara paralel dengan lebih cepat. Ada beberapa perangkat lunak prasyarat yang perlu dipasang sebelum menggunakan Hadoop. Setelah perangkat lunak prasyarat berhasil dipasang, Hadoop juga dapat dipasang mengikuti panduan lengkap pada Lampiran C. Secara umum, alur yang harus dilakukan meliputi pengunduhan berkas Hadoop. Selanjutnya akan dilakukan pengubahan kepemilikan berkas ke *user hdfsuser*. Karena Hadoop tidak mendukung IPv6, maka fitur ini perlu dimatikan juga. Alur pemasangan dan konfigurasi Hadoop lebih jelas sesuai dengan Gambar 3.5.

3.3.3.3 Instalasi dan Konfigurasi Spark

Apache Spark adalah sebuah kerangka kerja pengolahan data terdistribusi yang sangat cepat dan efisien. Spark dan Hadoop memiliki hubungan yang erat. Spark dapat berjalan di atas *Hadoop Distributed File System* (HDFS) dan dapat menggunakan Hadoop YARN sebagai manajer sumber daya. Oleh karena itu, instalasi Spark membutuhkan Hadoop sudah terpasang lebih dahulu. Alur pemasangan dan



Gambar 3.3 Alur Instalasi Perangkat Lunak

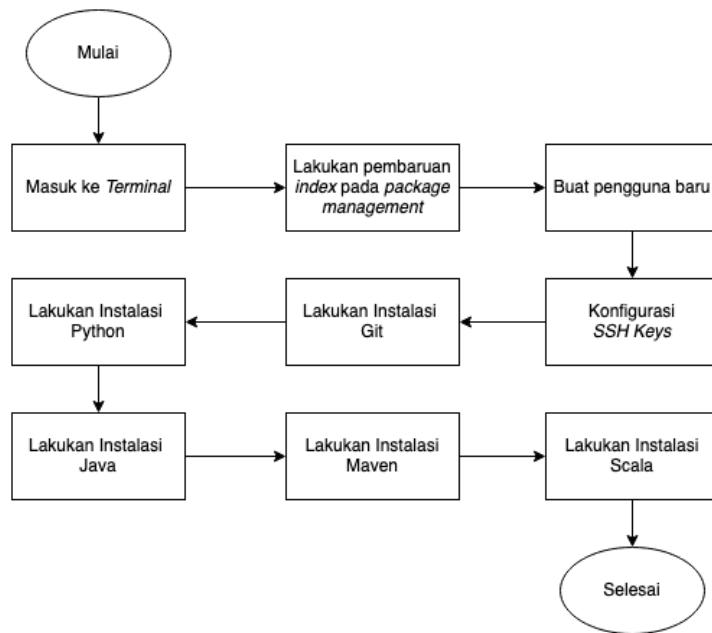
konfigurasi spark terlihat seperti pada Gambar 3.6. Apabila Hadoop sudah berhasil terpasang, langkah selanjutnya adalah memasang Spark seperti pada Lampiran D.

3.3.3.4 Instalasi dan Konfigurasi HiBench

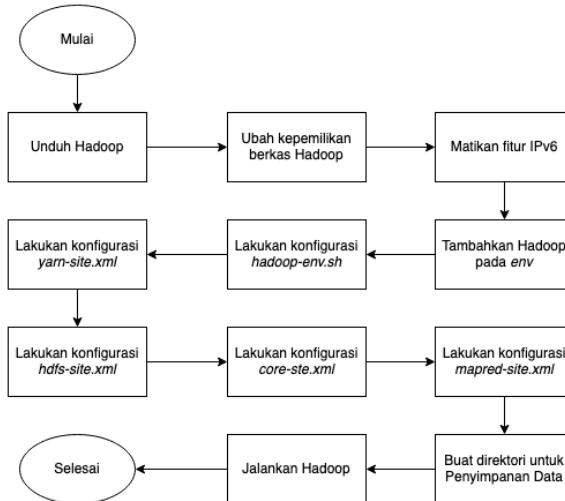
Sebelum melakukan eksperimen, diperlukan suatu perangkat lunak pengukuran kinerja sistem *Big Data*, yaitu HiBench. HiBench tidak dapat digunakan secara langsung ketika sudah berhasil diunduh, melainkan harus dilakukan pembangunan beberapa modul yang dibutuhkan dengan Maven dan konfigurasi beberapa parameter. Secara umum, alur instalasi dan konfigurasi HiBench sesuai dengan Gambar 3.7. Berkas HiBench diunduh dari repositori, dilanjutkan dengan pembangunan beberapa modul yang nantinya dibutuhkan. Selanjutnya, dilakukan konfigurasi beberapa berkas seperti *hibench.conf*, *hadoop.conf*, dan *spark.conf*. Jika telah dilakukan konfigurasi, dapat dilanjutkan dengan menjalankan beban kerja atau eksperimen. Lebih lanjut, pemasangan dan konfigurasi HiBench dijelaskan pada Lampiran E.

3.3.4 Eksperimen

Eksperimen merupakan tahapan yang dilakukan setelah perangkat keras dan perangkat lunak sudah berhasil terpasang. Pada tahapan ini, dilakukan berbagai per-



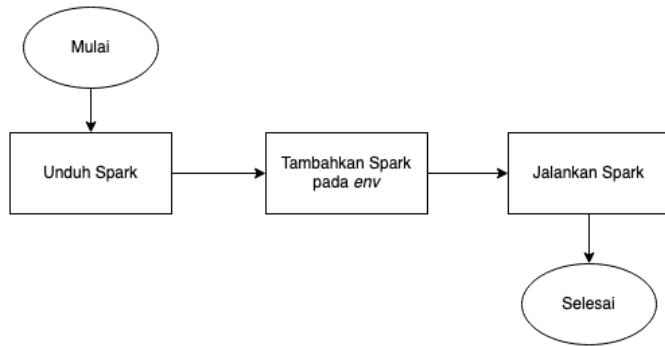
Gambar 3.4 Alur Instalasi Perangkat Lunak Prasyarat



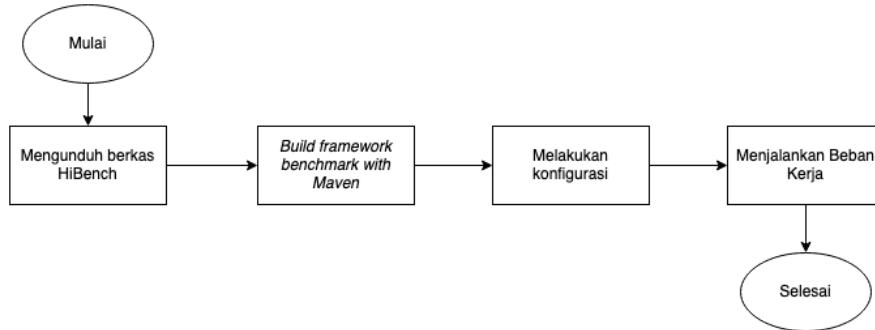
Gambar 3.5 Alur Instalasi dan Konfigurasi Hadoop

cobaan untuk menjawab pertanyaan pada masalah yang sebelumnya sudah didefinisikan. Beberapa beban kerja yang akan diuji coba pada tahapan ini dapat dilihat pada Tabel 3.3.

Pengujian dimulai dengan mengeksekusi beban kerja *Micro Benchmarks*, yang terdiri dari *WordCount*, *Sort*, dan *TeraSort*, pada DigitalOcean menggunakan parameter data yang divariasikan, yaitu 1GB; 5 GB; 10GB; 25 GB; dan 50 GB. Parameter input data dapat disesuaikan pada berkas *hibench.conf*. Hasil yang diukur berupa



Gambar 3.6 Alur Instalasi dan Konfigurasi Spark



Gambar 3.7 Alur Instalasi dan Konfigurasi HiBench

Tabel 3.3 Beban Kerja yang Digunakan

Tipe Beban Kerja	Nama Beban Kerja	Sumber Data	Perangkat Lunak
Micro Benchmarks	Sort	RandomTextWriter	Hadoop, Spark
	WordCount	RandomTextWriter	
	TeraSort	Hadoop TeraGen	

lama waktu eksekusi (*execution time*) dan *throughput per node*. Uji coba yang diajukan pada HiBench dilakukan sebanyak masing-masing satu uji coba. Pengujian secara lengkap sesuai pada Tabel 3.4. Hasil dan konfigurasi dari setiap pengujian dapat diakses melalui berkas *HiBench Report* yang nantinya akan diproses lebih lanjut pada tahap analisis dan evaluasi.

3.3.5 Proses Analisis dan Evaluasi Hasil Eksperimen

Analisis pada penelitian ini dilakukan dengan membandingkan hasil setiap eksperimen yang sebelumnya sudah dilakukan. Data yang akan dianalisis didapatkan melalui data eksperimen sebelumnya pada berkas *hibench.report*. Selanjutnya, tahapan ini ditutup dengan evaluasi terhadap hasil dari analisis yang diperoleh.

Tabel 3.4 Eksperimen yang Akan diuji Coba

Beban Kerja	Input Data	Perangkat Lunak	Hasil yang Diukur
Sort, TeraSort, WordCount	1 GB	Hadoop	<i>Execution time, throughput per node</i>
Sort, TeraSort, WordCount	5 GB	Hadoop	
Sort, TeraSort, WordCount	10 GB	Hadoop	
Sort, TeraSort, WordCount	25 GB	Hadoop	
Sort, TeraSort, WordCount	50 GB	Hadoop	
Sort, TeraSort, WordCount	1 GB	Spark	
Sort, TeraSort, WordCount	5 GB	Spark	
Sort, TeraSort, WordCount	10 GB	Spark	
Sort, TeraSort, WordCount	25 GB	Spark	
Sort, TeraSort, WordCount	50 GB	Spark	

BAB IV

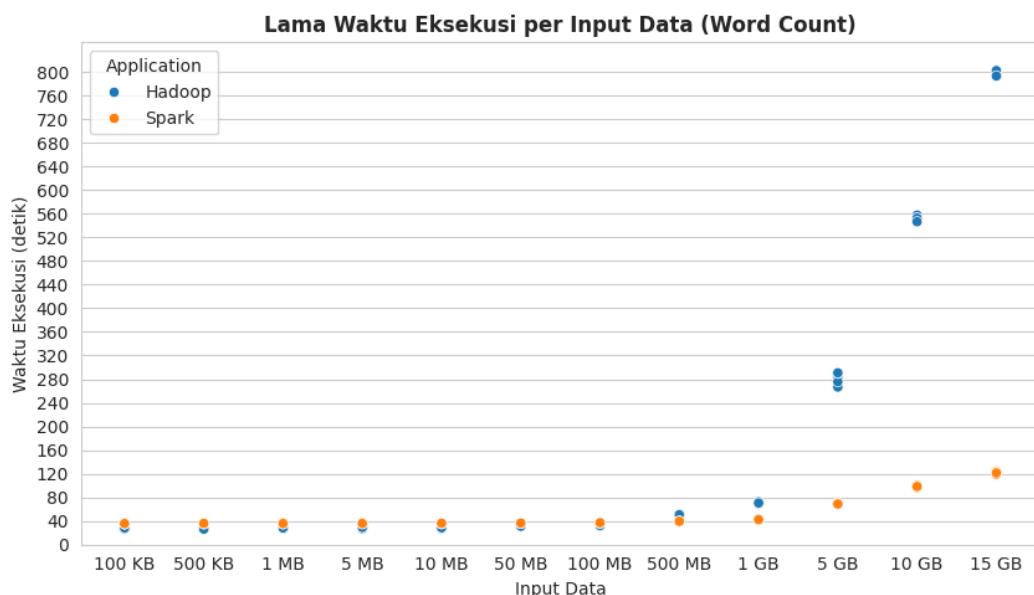
HASIL DAN PEMBAHASAN

4.1 Hasil Penelitian

Penelitian ini membandingkan kinerja Hadoop dan Spark pada *platform cloud* DigitalOcean menggunakan alat pengujian data besar yang bernama HiBench pada lingkup *Micro Benchmarks*, yaitu *Word Count* dan *Sort* dengan data masukan berupa teks. Setiap pengujian, nilai execution time (waktu eksekusi) memiliki satuan detik dan nilai *throughput* memiliki satuan megabita per detik. Ukuran data input akan diubah secara bertahap untuk merepresentasikan berbagai ukuran data pada dunia nyata.

4.1.1 Persebaran Waktu Eksekusi pada Hadoop dan Spark

Waktu eksekusi adalah waktu yang diperlukan dalam memproses data. Nilai parameter ini didapatkan dengan cara mencari selisih antara waktu awal dan waktu akhir saat Apache Hadoop dan Apache Spark dijalankan atau dihentikan untuk memproses input data dengan beban kerja masing-masing. Satuan pengukuran untuk parameter waktu eksekusi adalah detik atau *seconds*. Setiap beban kerja dijalankan sebanyak 5x untuk mendapatkan hasil yang lebih terukur.



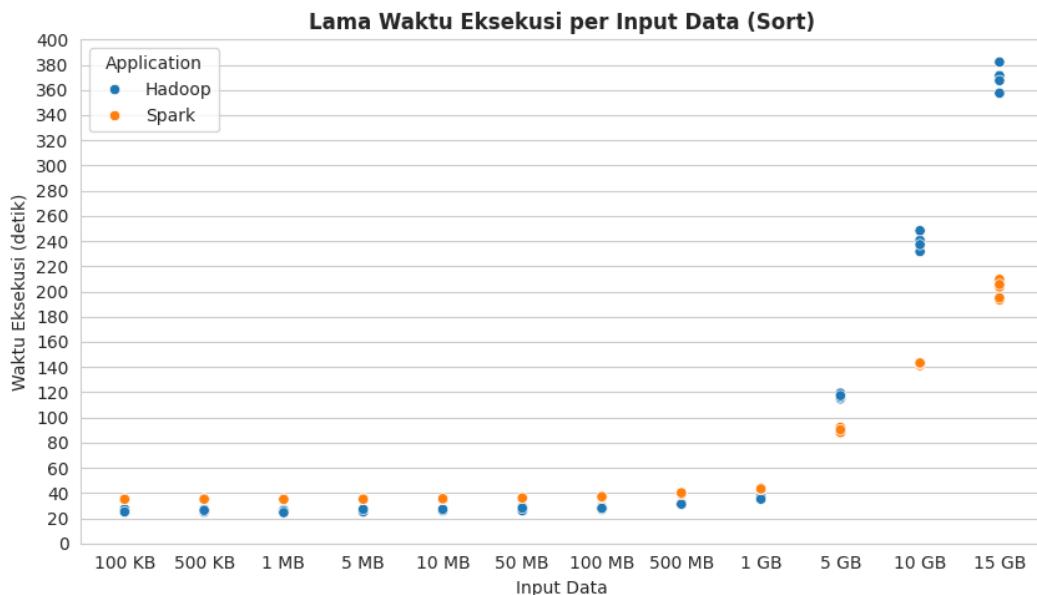
Gambar 4.1 Persebaran Waktu Eksekusi *Word Count* (Hadoop, Spark)

Gambar 4.2 dan 4.1 menyajikan scatter plot yang membandingkan performa Hadoop dan Spark dalam dua tugas pemrosesan data yang berbeda: sorting dan word count. Sumbu x pada kedua gambar menunjukkan variasi ukuran input data, mulai dari 100 KB hingga 15 GB, sedangkan sumbu y menunjukkan waktu eksekusi dalam detik.

Pada Gambar 4.2, terlihat bahwa Spark secara konsisten mengungguli Hadoop dalam tugas sorting. Titik data Spark selalu berada di bawah titik data Hadoop, menunjukkan waktu eksekusi yang lebih singkat untuk semua ukuran data. Perbedaan performa ini semakin signifikan seiring bertambahnya ukuran data. Sebagai contoh, pada ukuran data 15 GB, Spark menyelesaikan tugas sorting dalam waktu kurang dari 240 detik, sedangkan Hadoop membutuhkan waktu lebih dari 360 detik. Selain itu, titik data Spark lebih rapat, mengindikasikan variabilitas performa yang lebih rendah dan menghasilkan waktu eksekusi yang lebih konsisten.

Pada Gambar 4.1, Spark masih menunjukkan performa yang lebih baik daripada Hadoop pada sebagian besar ukuran data. Namun, perbedaan performanya tidak sebesar pada tugas sorting. Pada ukuran data kecil (di bawah 1 GB), kedua framework menunjukkan waktu eksekusi yang relatif sama. Pada ukuran data yang lebih besar, Spark tetap lebih cepat, kecuali pada 15 GB di mana Hadoop menunjukkan waktu eksekusi yang sedikit lebih cepat, namun dengan variabilitas yang lebih tinggi.

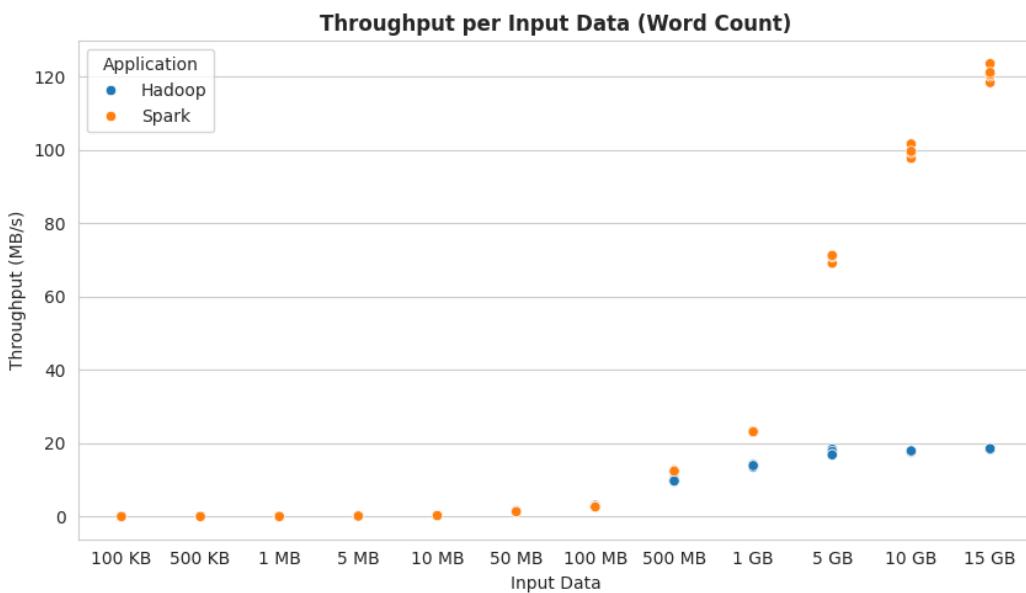
Secara keseluruhan, hasil ini menunjukkan bahwa Spark umumnya lebih unggul dan konsisten daripada Hadoop dalam menangani tugas pemrosesan data, terutama untuk sorting. Namun, perbandingan performa dapat bervariasi tergantung pada jenis tugas dan ukuran data.



Gambar 4.2 Persebaran Waktu Eksekusi *Sort* (Hadoop, Spark)

4.1.2 Persebaran *Throughput* pada Hadoop dan Spark

Kedua gambar di atas menyajikan scatter plot yang membandingkan throughput Hadoop dan Spark dalam dua tugas pemrosesan data: sorting (Gambar 4.4) dan word count (Gambar 4.3). Sumbu x pada kedua gambar menunjukkan variasi ukuran input data, sedangkan sumbu y menunjukkan throughput dalam MB/s.



Gambar 4.3 *Throughput Word Count (Hadoop, Spark)*

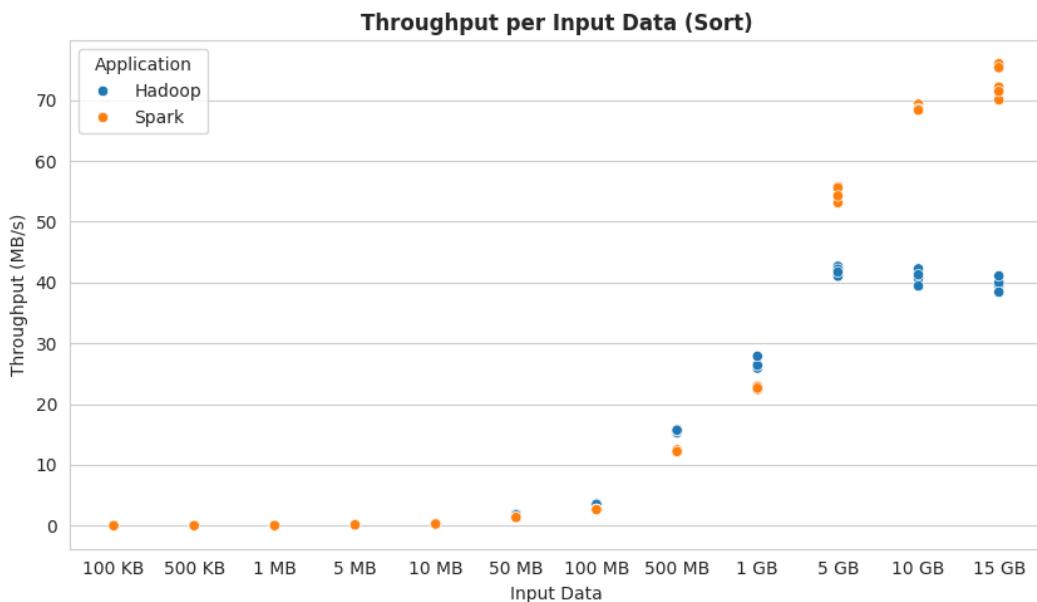
Pada tugas sorting (Gambar 4.4), Spark menunjukkan peningkatan throughput yang signifikan seiring dengan bertambahnya ukuran data. Pada ukuran data terbesar (15 GB), Spark mencapai throughput sekitar 40 MB/s. Sebaliknya, Hadoop menunjukkan peningkatan throughput yang lebih lambat dan hanya mencapai sekitar 35 MB/s pada ukuran data yang sama. Hal ini menunjukkan bahwa Spark mampu memanfaatkan sumber daya secara lebih efisien untuk memproses data dalam jumlah besar.

Pada tugas word count (Gambar 4.3), Spark mencapai throughput yang lebih tinggi daripada Hadoop untuk sebagian besar ukuran data. Perbedaan throughput paling mencolok terlihat pada ukuran data menengah (100 MB - 1 GB), di mana Spark mencapai throughput lebih dari 15 MB/s, sedangkan Hadoop hanya mencapai sekitar 5 MB/s. Meskipun Hadoop menunjukkan peningkatan throughput yang signifikan pada ukuran data terbesar (15 GB), mendekati throughput Spark, Spark tetap memiliki keunggulan dalam efisiensi pemrosesan data, terutama untuk ukuran data yang lebih kecil.

Hasil ini menunjukkan bahwa Spark secara umum lebih efisien dalam memanfaatkan sumber daya untuk memproses data dalam jumlah besar dibandingkan dengan Hadoop, baik untuk tugas sorting maupun word count.

4.1.3 Rata-rata Waktu Eksekusi pada Hadoop dan Spark

Gambar 4.5 dan 4.6 menyajikan line plot yang menggambarkan rata-rata waktu eksekusi Hadoop dan Spark untuk tugas sorting dan word count dengan berbagai ukuran data. Sumbu x pada kedua gambar menunjukkan ukuran input data, sedangkan sumbu y menunjukkan rata-rata waktu eksekusi dalam detik. Garis vertikal pada kedua gambar menunjukkan titik di mana Spark mulai menunjukkan performa yang lebih cepat dibandingkan Hadoop.



Gambar 4.4 *Throughput Sort* (Hadoop, Spark)

Pada Gambar 4.5, terlihat bahwa Spark secara konsisten lebih cepat daripada Hadoop untuk semua ukuran data pada tugas sorting. Perbedaan performa semakin signifikan seiring dengan bertambahnya ukuran data. Titik di mana Spark mulai mengungguli Hadoop terjadi pada ukuran data 1 GB. Pada titik ini, waktu eksekusi Spark mulai menurun secara signifikan dibandingkan dengan Hadoop yang meningkat secara eksponensial.

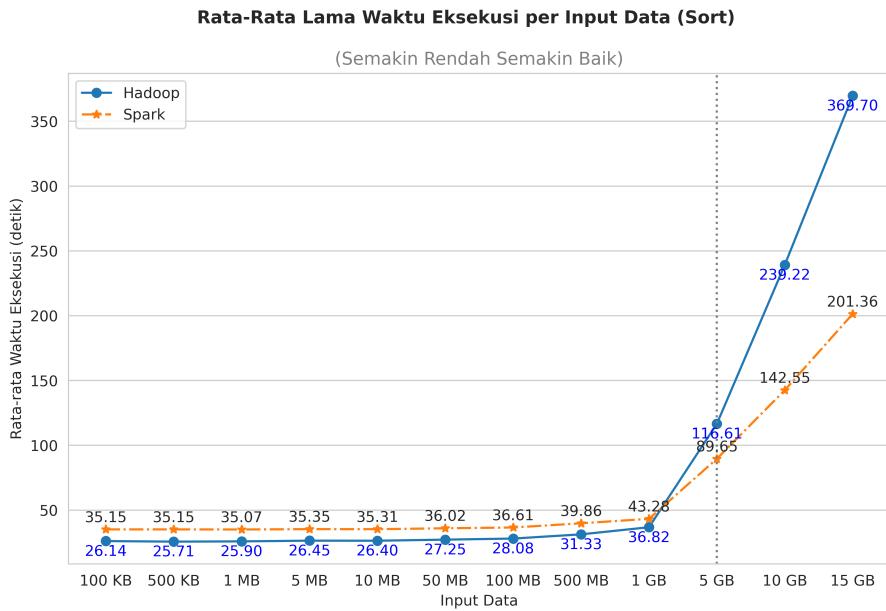
Pada Gambar 4.6, Spark juga menunjukkan performa yang lebih baik daripada Hadoop pada sebagian besar ukuran data pada tugas word count. Perbedaan performa mulai terlihat pada ukuran data 100 MB, dan Spark terus unggul hingga ukuran data terbesar (15 GB).

4.1.4 Rata-rata Throughput pada Hadoop dan Spark

Gambar 4.7 dan 4.8 menyajikan line plot yang menggambarkan rata-rata throughput Hadoop dan Spark untuk tugas sorting dan word count dengan berbagai ukuran data. Sumbu x pada kedua gambar menunjukkan ukuran input data, sedangkan sumbu y menunjukkan rata-rata throughput dalam MB/s. Garis vertikal pada kedua gambar menunjukkan titik di mana Spark mulai menunjukkan throughput yang lebih tinggi dibandingkan Hadoop.

Pada tugas sorting, Spark menunjukkan peningkatan throughput yang signifikan seiring dengan meningkatnya ukuran data. Pada ukuran data kecil (di bawah 500 MB), throughput Hadoop dan Spark relatif rendah dan sebanding. Namun, setelah titik 500 MB, Spark secara konsisten menunjukkan throughput yang lebih tinggi, mencapai 68.68 MB/s pada 15 GB dibandingkan dengan 45.39 MB/s untuk Hadoop.

Pada tugas word count, Spark juga mengungguli Hadoop dalam hal throughput, meskipun dengan perbedaan yang tidak sebesar pada tugas sorting. Spark mulai



Gambar 4.5 Rata-rata Waktu Eksekusi (*Sort*)

menunjukkan throughput yang lebih tinggi pada ukuran data 100 MB. Pada ukuran data 15 GB, Spark mencapai throughput 99.405 MB/s, sedangkan Hadoop hanya mencapai 18.407 MB/s.

4.1.5 Rate of Change

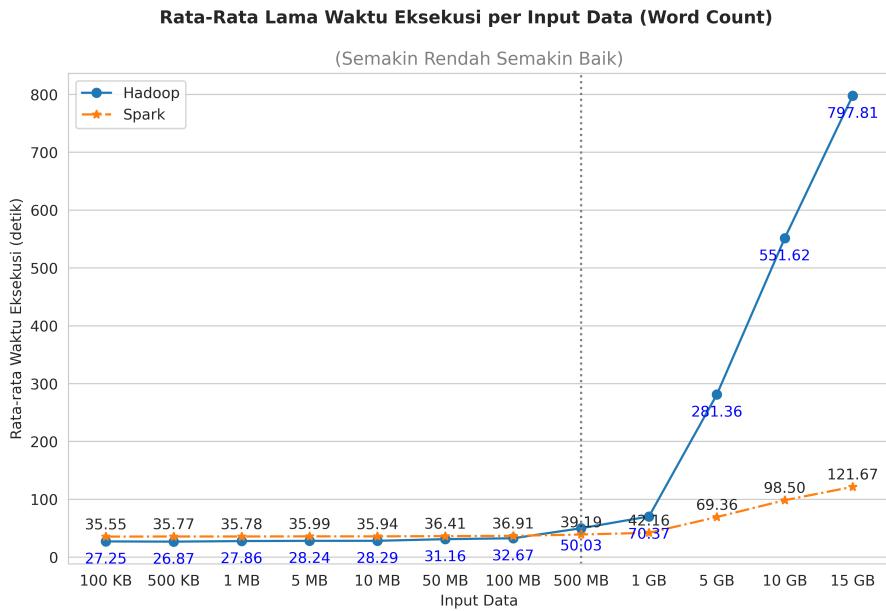
4.1.6 Penggunaan CPU

Gambar 4.12 dan 4.13 menunjukkan pola penggunaan CPU oleh Hadoop dan Spark untuk tugas sorting dan word count pada berbagai ukuran data. Sumbu x mewakili waktu dalam detik, sedangkan sumbu y mewakili persentase penggunaan CPU. Setiap baris grafik menunjukkan ukuran data yang berbeda, mulai dari 100 KB hingga 15 GB.

Pada kedua tugas, terlihat bahwa Spark cenderung menunjukkan pola penggunaan CPU yang lebih stabil dan konsisten dibandingkan dengan Hadoop. Penggunaan CPU Spark umumnya lebih tinggi dan merata di sepanjang waktu eksekusi, mengindikasikan pemanfaatan sumber daya yang lebih efisien dan pemrosesan paralel yang lebih optimal. Di sisi lain, penggunaan CPU Hadoop cenderung lebih fluktuatif, dengan periode lonjakan dan penurunan yang signifikan.

Pada tugas sorting (Gambar 4.12), perbedaan pola penggunaan CPU antara Hadoop dan Spark semakin terlihat pada ukuran data yang lebih besar. Spark mempertahankan penggunaan CPU yang tinggi dan stabil, sementara Hadoop menunjukkan fluktuasi yang lebih besar dan cenderung menurun pada akhir eksekusi. Hal ini menunjukkan bahwa Spark lebih mampu menangani data besar secara efisien dan konsisten.

Pada tugas word count (Gambar 4.13), perbedaan pola penggunaan CPU tidak se-



Gambar 4.6 Rata-rata Waktu Eksekusi (*Word Count*)

jelas pada tugas sorting. Namun, Spark tetap menunjukkan penggunaan CPU yang lebih stabil dan tinggi secara keseluruhan dibandingkan dengan Hadoop.

Gambar 4.14 dan 4.15 menyajikan bar chart yang menggambarkan perbandingan persentase state U_s (user) antara Hadoop dan Spark untuk tugas sorting dan word count pada berbagai ukuran data dan skenario. State U_s merepresentasikan waktu CPU yang dihabiskan dalam mode user, yang menunjukkan waktu yang dihabiskan untuk menjalankan kode aplikasi.

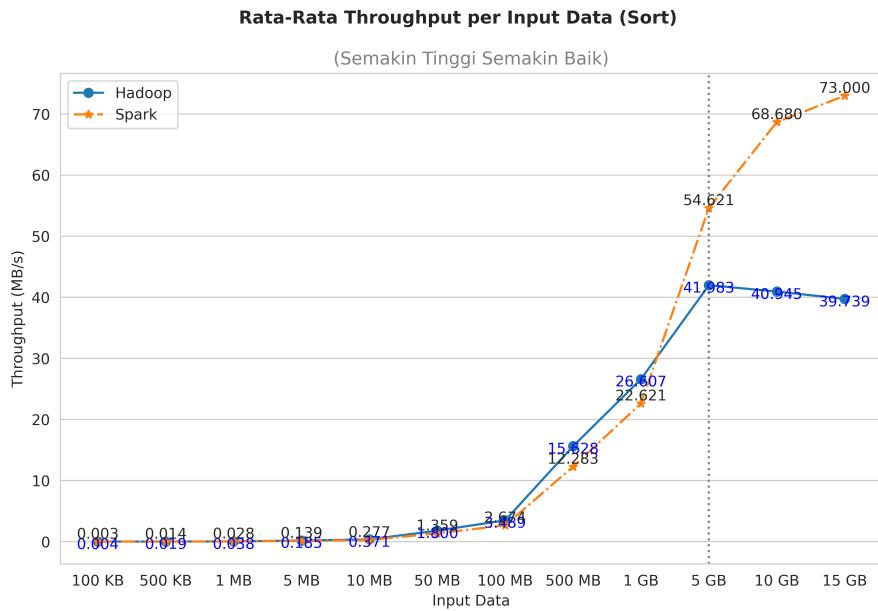
Pada kedua tugas, Spark secara konsisten menunjukkan persentase state U_s yang lebih tinggi dibandingkan dengan Hadoop. Hal ini menunjukkan bahwa Spark lebih efisien dalam memanfaatkan waktu CPU untuk menjalankan kode aplikasi, sembari Hadoop menghabiskan lebih banyak waktu dalam state lain, seperti state sistem (S_s) atau state idle (I_s).

Pada tugas sorting (Gambar 4.14), perbedaan persentase state U_s antara Spark dan Hadoop semakin terlihat pada ukuran data yang lebih besar. Hal ini menunjukkan bahwa Spark lebih mampu memaksimalkan penggunaan CPU untuk tugas komputasi intensif seperti sorting, terutama saat menangani data dalam jumlah besar.

Pada tugas word count (Gambar 4.15), meskipun Spark masih menunjukkan persentase state U_s yang lebih tinggi, perbedaannya tidak sejelas pada tugas sorting. Hal ini mungkin karena tugas word count tidak seintensif sorting secara komputasi.

4.1.7 Utilisasi Sistem

Gambar ?? dan ?? menyajikan informasi pemantauan sistem yang membandingkan penggunaan sumber daya komputasi oleh Hadoop dan Spark selama menjalankan tugas sorting dan word count dengan ukuran data "fiveteengig". Setiap gambar



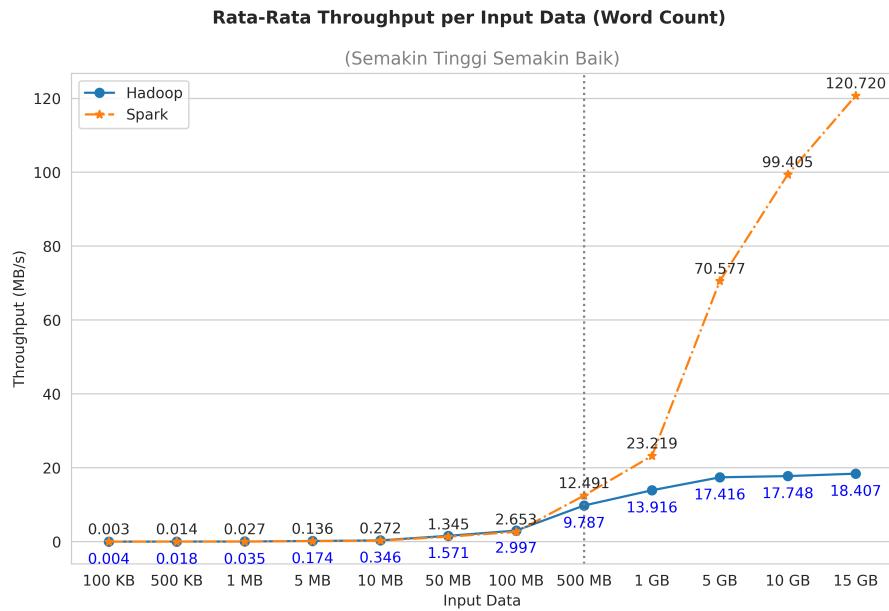
Gambar 4.7 Rata-rata *Throughput (Sort)*

terdiri dari tiga grafik yang menunjukkan penggunaan CPU, Disk I/O, dan memori seiring berjalannya waktu.

Pada kedua tugas, Spark menunjukkan pola penggunaan CPU yang lebih tinggi dan konsisten dibandingkan dengan Hadoop. Grafik penggunaan CPU Spark menunjukkan garis yang cenderung mendatar di dekat tingkat utilisasi maksimum, mengindikasikan bahwa Spark mampu memaksimalkan pemanfaatan CPU untuk pemrosesan data secara terus-menerus. Di sisi lain, grafik penggunaan CPU Hadoop menunjukkan fluktuasi yang lebih besar, dengan periode lonjakan dan penurunan yang signifikan. Hal ini menunjukkan bahwa Hadoop mengalami periode idle yang lebih lama dan tidak memanfaatkan sumber daya CPU seefisien Spark.

Hadoop menunjukkan aktivitas Disk I/O yang jauh lebih tinggi dibandingkan dengan Spark, terutama pada tugas sorting (Gambar ??). Grafik Disk I/O Hadoop menunjukkan lonjakan aktivitas baca dan tulis yang signifikan sepanjang waktu eksekusi. Hal ini sesuai dengan pendekatan berbasis disk Hadoop yang membutuhkan pembacaan dan penulisan data ke disk secara intensif. Sebaliknya, Spark, dengan arsitektur in-memory, meminimalkan operasi Disk I/O. Grafik Disk I/O Spark menunjukkan aktivitas yang jauh lebih rendah dan stabil, yang berkontribusi pada peningkatan performanya.

Spark menunjukkan penggunaan memori yang lebih tinggi dan stabil dibandingkan dengan Hadoop, terutama pada tugas word count (Gambar ??). Grafik penggunaan memori Spark menunjukkan garis yang cenderung mendatar pada tingkat utilisasi yang tinggi, menunjukkan bahwa Spark menyimpan data dalam RAM untuk akses yang lebih cepat dan pemrosesan yang efisien. Penggunaan memori Hadoop lebih rendah dan fluktuatif, menunjukkan bahwa Hadoop tidak memanfaatkan memori secara optimal.



Gambar 4.8 Rata-rata *Throughput* (*Word Count*)

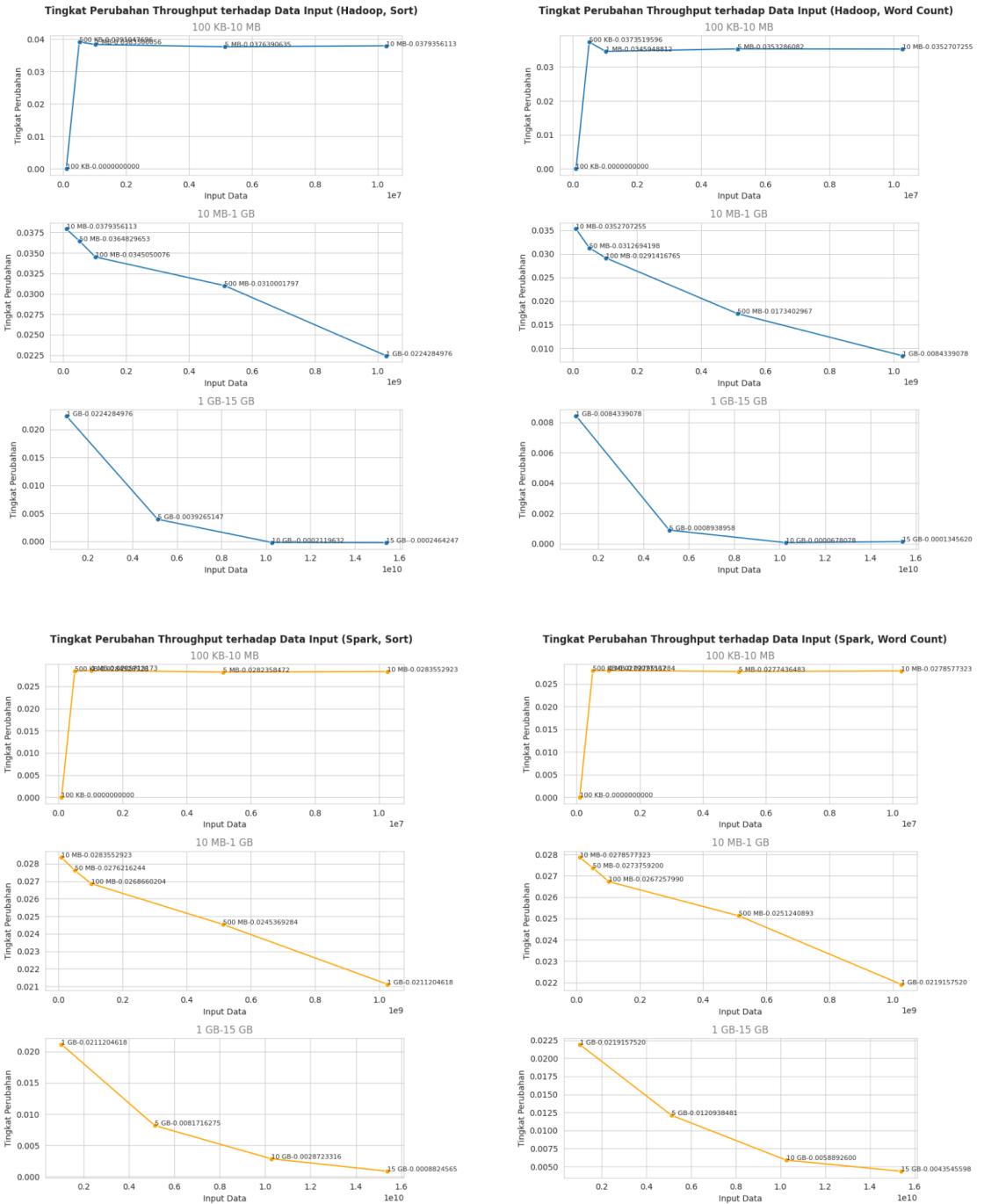
Analisis pemantauan sistem menegaskan keunggulan Spark dalam hal efisiensi dan optimasi penggunaan sumber daya komputasi dibandingkan dengan Hadoop. Spark mampu memaksimalkan penggunaan CPU, meminimalkan operasi Disk I/O, dan memanfaatkan memori secara efisien, yang berkontribusi pada performa dan skalabilitas yang lebih baik dalam tugas-tugas pemrosesan data besar.

4.1.7.1 Input Data 100 KB

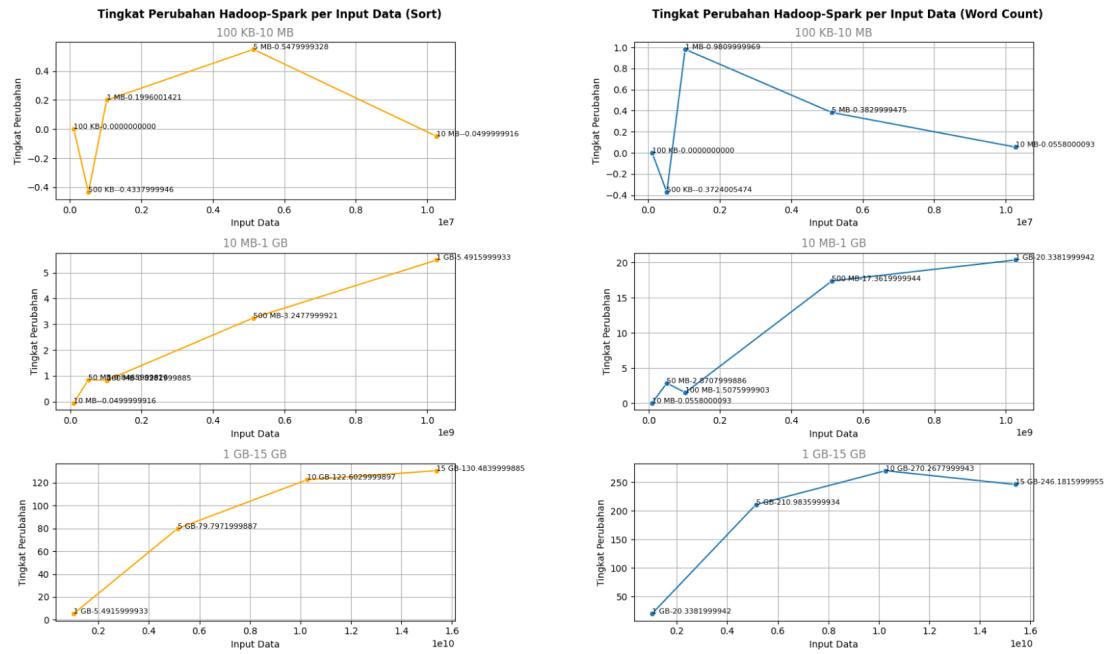
4.1.7.2 Input Data 15 GB



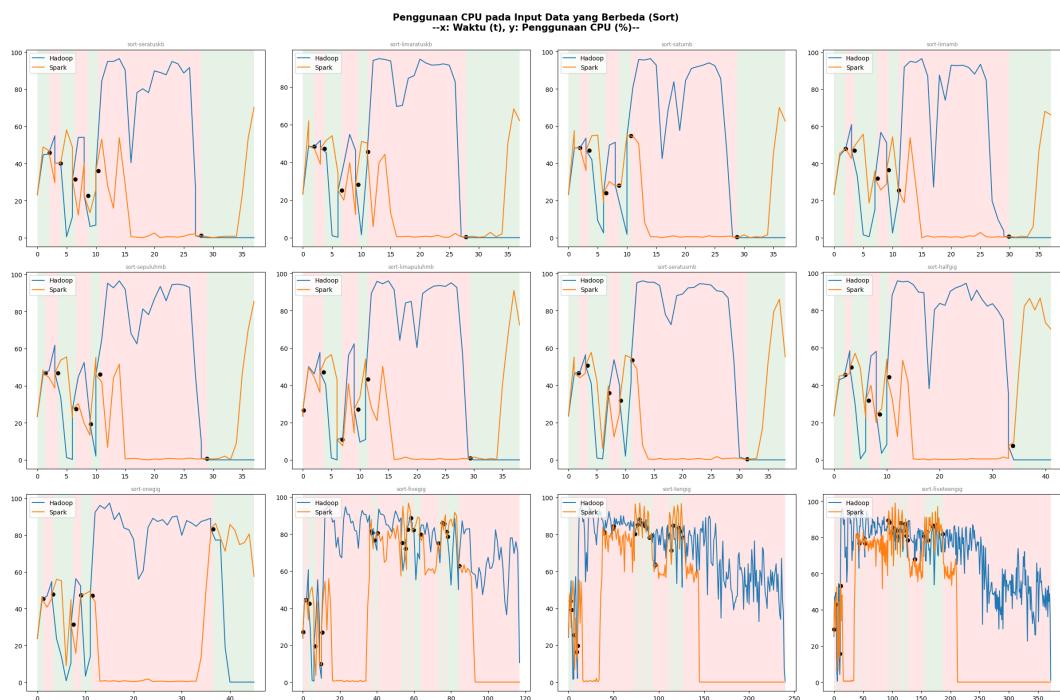
Gambar 4.9 dur



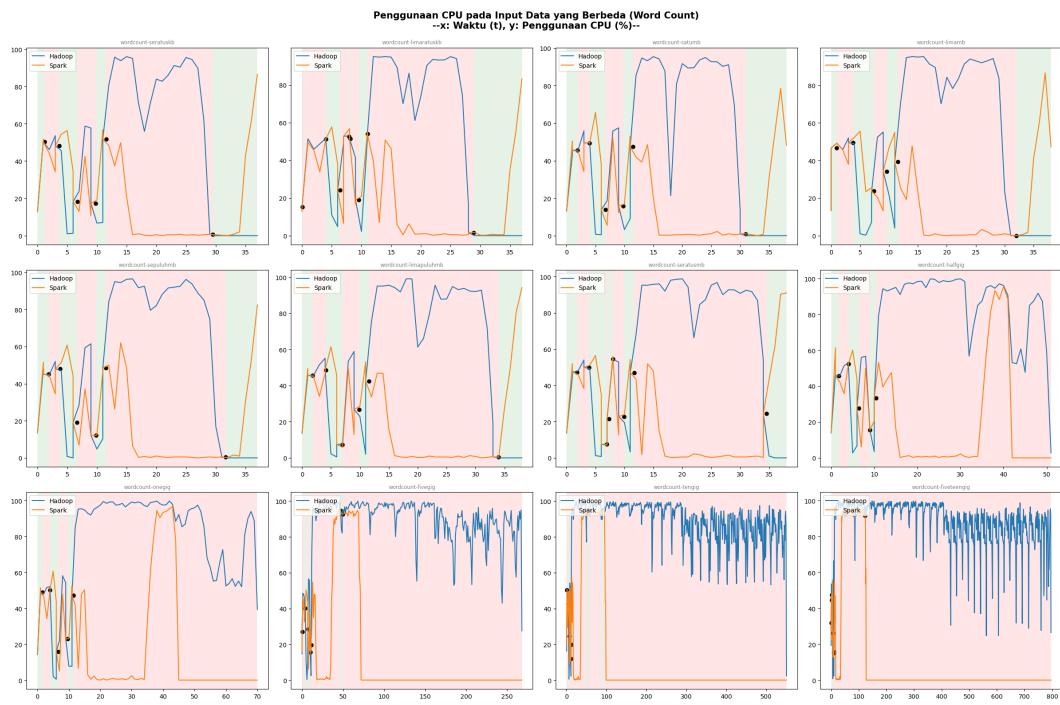
Gambar 4.10 th



Gambar 4.11 hadoop-spark



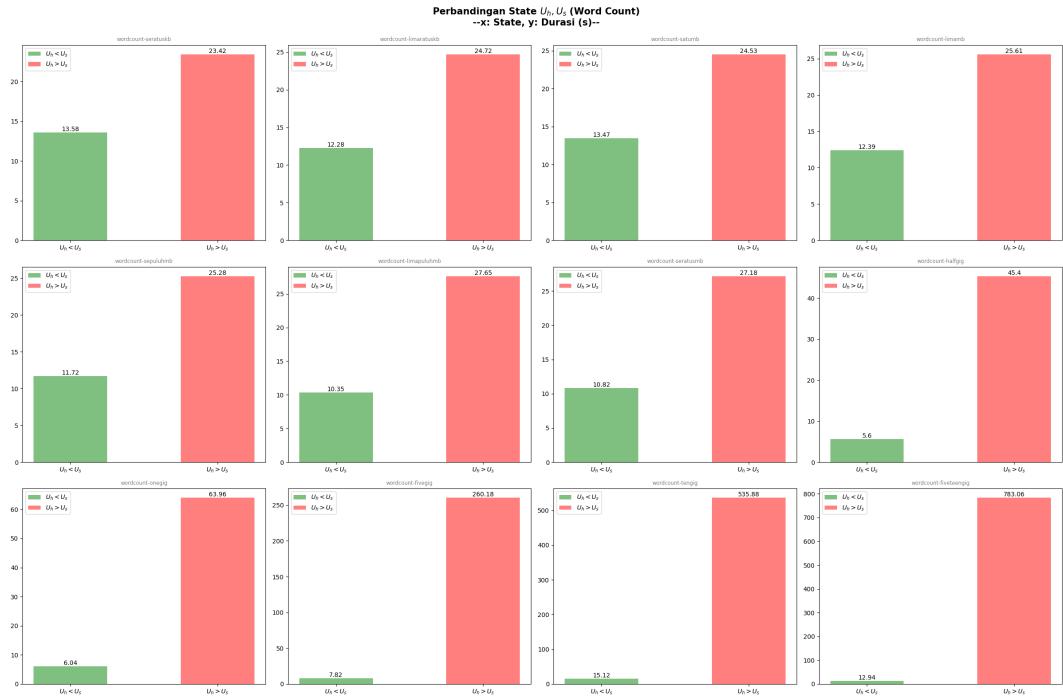
Gambar 4.12 Penggunaan CPU (Sort)



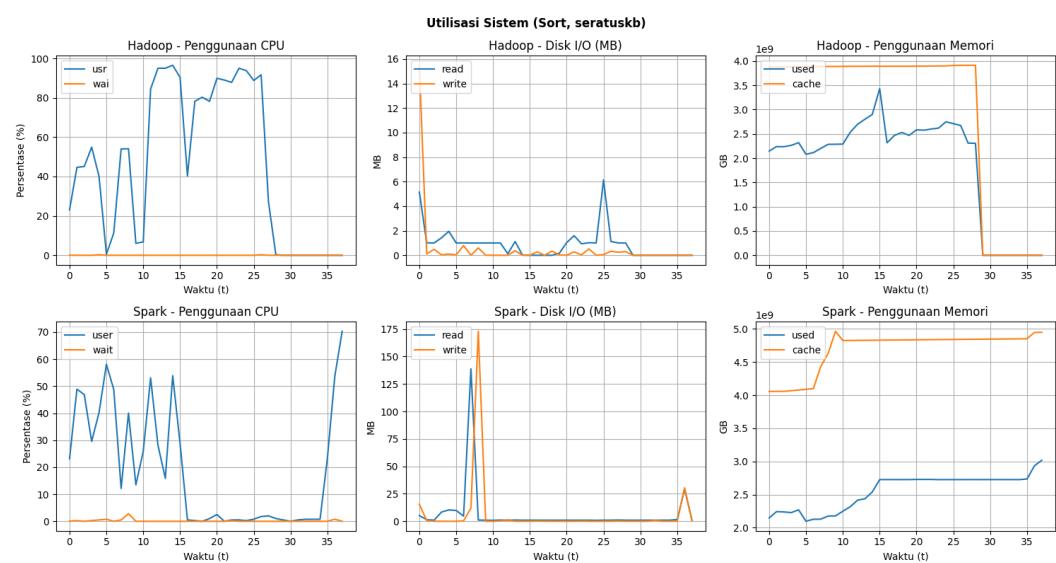
Gambar 4.13 Penggunaan CPU (Word Count)



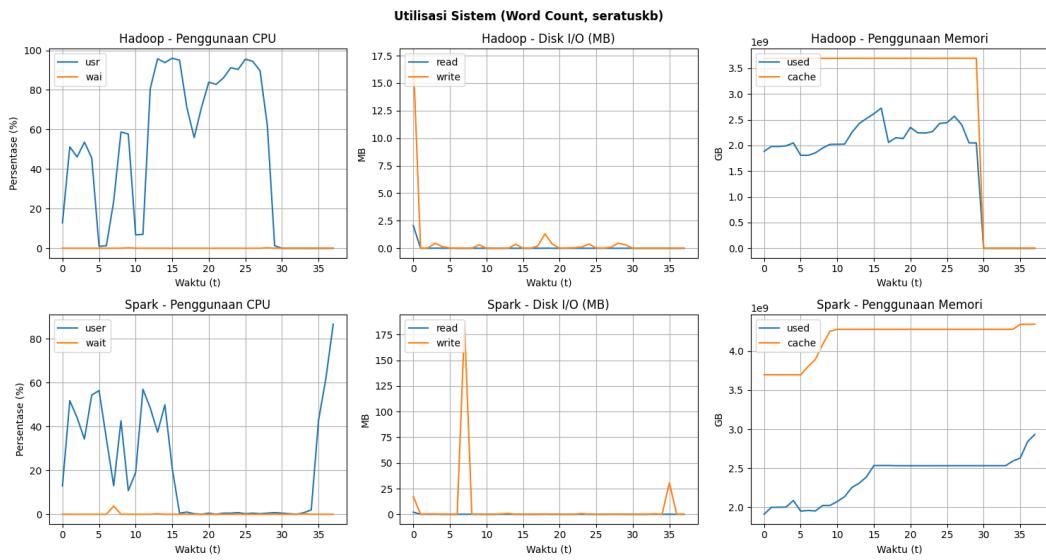
Gambar 4.14 State (Sort)



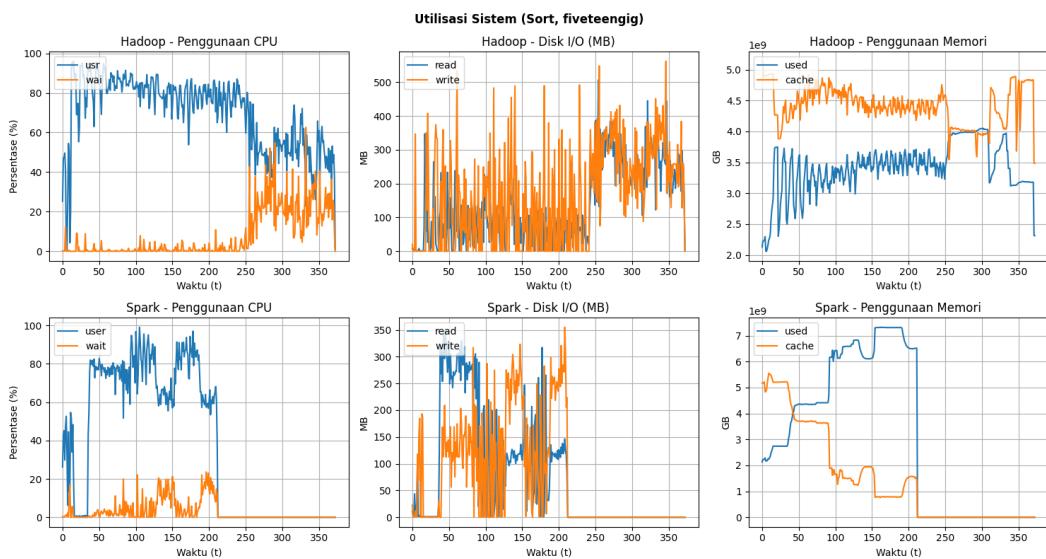
Gambar 4.15 State (Word Count)



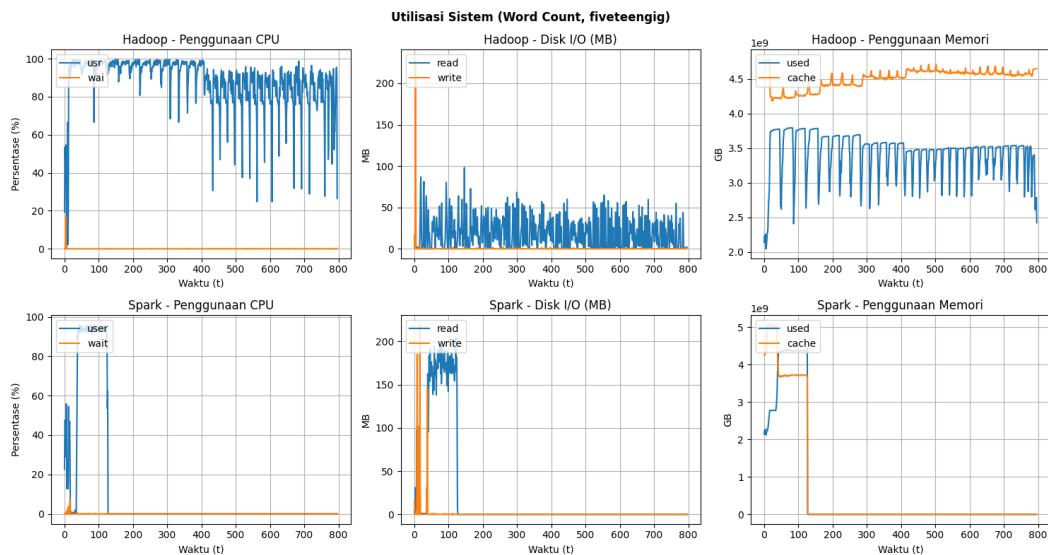
Gambar 4.16 Utilisasi Sistem (Sort) pada Input Data 100 KB



Gambar 4.17 Utilisasi Sistem (Word Count) pada Input Data 100 KB



Gambar 4.18 Utilisasi Sistem (Sort) pada Input Data 15 GB



Gambar 4.19 Utilisasi Sistem (Word Count) pada Input Data 15 GB

BAB V

PENUTUP

5.1 Kesimpulan

Penelitian ini melakukan evaluasi komprehensif terhadap kinerja Hadoop dan Spark dalam konteks pemrosesan *Big Data*. Dengan menggunakan beban kerja *Micro Benchmarks*, penelitian ini berhasil mengukur dan membandingkan performa kedua platform tersebut dalam mode *pseudo-distributed*. Hasil eksperimen menunjukkan perbedaan signifikan dalam hal efisiensi dan penggunaan sumber daya antara Hadoop dan Spark, dengan Spark menunjukkan keunggulan dalam sejumlah aspek. Analisis ini memberikan wawasan berharga bagi organisasi yang ingin memilih platform Big Data yang tepat, memastikan keputusan mereka didasarkan pada informasi yang akurat dan relevan.

5.2 Saran

Untuk penelitian selanjutnya, disarankan untuk menggali lebih dalam aspek keamanan dan administrasi dari kedua *platform* tersebut. Penelitian yang lebih fokus pada pengukuran aspek skalabilitas dan ketersediaan Hadoop dan Spark dalam berbagai konfigurasi kluster juga akan bermanfaat. Selain itu, penelitian tentang integrasi teknologi baru seperti kontainer dan orkestrasi kluster dalam pemrosesan *Big Data* dapat menjadi topik yang menjanjikan. Akhirnya, implementasi kasus penggunaan nyata dan studi komparatif dalam lingkungan produksi akan memberikan pemahaman yang lebih dalam tentang kinerja dan kegunaan kedua *platform* ini dalam skenario dunia nyata.

DAFTAR PUSTAKA

- [1] Yassir Samadi, Mostapha Zbakh **and** Claude Tadonki. “Performance Comparison between Hadoop and Spark Frameworks Using HiBench Benchmarks”. *inConcurrency and Computation: Practice and Experience*: 30.12 (2018), e4367. ISSN: 1532-0634. DOI: 10.1002/cpe.4367. ([urlseen](#) 21/09/2023).
- [2] David Reinsel, John Gantz **and** John Rydning. “The Digitization of the World from Edge to Core”. *in*(2018).
- [3] Cecilia Adrian, Rusli Abdullah, Rodziah Atan **and others**. “Expert Review on Big Data Analytics Implementation Model in Data-driven Decision-Making”. *in2018 Fourth International Conference on Information Retrieval and Knowledge Management (CAMP)*: Kota Kinabalu, Malaysia: IEEE, **march** 2018, **pages** 1–5. ISBN: 978-1-5386-3812-5. DOI: 10.1109/INFRKM.2018.8464770. ([urlseen](#) 17/01/2024).
- [4] Briyan Efflin Syahputra **and** Akhmad Afnan. “Pendeteksian Fraud: Peran Big Data dan Audit Forensik”. *inJurnal ASET (Akuntansi Riset)*: 12.2 (**december** 2020), **pages** 301–316. ISSN: 2541-0342. DOI: 10.17509/jaset.v12i2.28939. ([urlseen](#) 18/01/2024).
- [5] Thirsya Widya Sulaiman, Raka Bagas Fitriansyah, Ahmad Rafif Alaudin **and others**. “LITERATURE REVIEW: PENERAPAN BIG DATA DALAM KESEHATAN MASYARAKAT”. *in*: (2023).
- [6] Nico Fernando, Mery Mery, Jessica Jessica **and others**. “Utilization of Big Data In E-Commerce Business”. *inConference Series*: 3 (**november** 2020), **pages** 62–67. DOI: 10.34306/conferenceseries.v3i1.383.
- [7] *KOMPARASI KECEPATAN HADOOP MAPREDUCE DAN APACHE SPARK DALAM MENGOLAH DATA TEKS — Jurnal Ilmiah Matrik*. <https://journal.binadarma.ac.id/index.php/jurnalmatrik/article/view/1649>. ([urlseen](#) 21/09/2023).
- [8] Muntadher Saadoon, Siti Hafizah Ab. Hamid, Hazrina Sofian **and others**. “Fault Tolerance in Big Data Storage and Processing Systems: A Review on Challenges and Solutions”. *inAin Shams Engineering Journal*: 13.2 (**march** 2022), **page** 101538. ISSN: 2090-4479. DOI: 10.1016/j.asej.2021.06.024. ([urlseen](#) 18/01/2024).
- [9] Devipsita Bhattacharya, Faiz Currim **and** Sudha Ram. “Evaluating Distributed Computing Infrastructures: An Empirical Study Comparing Hadoop Deployments on Cloud and Local Systems”. *inIEEE Transactions on Cloud Computing*: 9.3 (**july** 2021), **pages** 1075–1088. ISSN: 2168-7161. DOI: 10.1109/TCC.2019.2902377. ([urlseen](#) 13/10/2023).
- [10] N. Ahmed, Andre L. C. Barczak, Teo Susnjak **and others**. “A Comprehensive Performance Analysis of Apache Hadoop and Apache Spark for Large Scale Data Sets Using HiBench”. *inJournal of Big Data*: 7.1 (**december** 2020), **page** 110. ISSN: 2196-1115. DOI: 10.1186/s40537-020-00388-5. ([urlseen](#) 23/11/2023).

- [11] Rendiyono Saputro, Aminuddin Aminuddin **and** Yuda Munarko. “Perbandingan Kinerja Komputasi Hadoop Dan Spark Untuk Memprediksi Cuaca (Studi Kasus : Storm Event Database)”. *inJurnal Repotor*: 2 (**march** 2020), **page** 463. DOI: 10.22219/repositor.v2i4.93.
- [12] Jeffrey Dean **and** Sanjay Ghemawat. “MapReduce: Simplified Data Processing on Large Clusters”. *inProceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation - Volume 6*: OSDI’04. USA: USENIX Association, **december** 2004, **page** 10. ([urlseen](#) 28/09/2023).
- [13] Yassir Samadi, Mostapha Zbakh **and** Claude Tadonki. “Comparative Study between Hadoop and Spark Based on Hibench Benchmarks”. *in2016 2nd International Conference on Cloud Computing Technologies and Applications (CloudTech)*: Marrakech, Morocco: IEEE, **may** 2016, **pages** 267–275. ISBN: 978-1-4673-8894-8. DOI: 10.1109/CloudTech.2016.7847709. ([urlseen](#) 24/09/2023).
- [14] Hossein Ahmadvand, Maziar Goudarzi **and** Fouzhan Foroutan. “Gapprox: Using Gallup Approach for Approximation in Big Data Processing”. *inJournal of Big Data*: 6.1 (**february** 2019), **page** 20. ISSN: 2196-1115. DOI: 10.1186/s40537-019-0185-4. ([urlseen](#) 29/09/2023).
- [15] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin **andothers**. “Spark: Cluster Computing with Working Sets”. *inProceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*: HotCloud’10. USA: USENIX Association, **june** 2010, **page** 10. ([urlseen](#) 28/09/2023).
- [16] Shengsheng Huang, Jie Huang, Jinquan Dai **andothers**. “The HiBench Benchmark Suite: Characterization of the MapReduce-Based Data Analysis”. *in()*.
- [17] Juwei Shi, Yunjie Qiu, Umar Farooq Minhas **andothers**. “Clash of the Titans: MapReduce vs. Spark for Large Scale Data Analytics”. *inProceedings of the VLDB Endowment*: 8.13 (**september** 2015), **pages** 2110–2121. ISSN: 2150-8097. DOI: 10.14778/2831360.2831365. ([urlseen](#) 28/09/2023).
- [18] Satish Gopalani **and** Rohan Arora. “Comparing Apache Spark and Map Reduce with Performance Analysis Using K-Means”. *inInternational Journal of Computer Applications*: 113.1 (**march** 2015), **pages** 8–11. ISSN: 09758887. DOI: 10.5120/19788-0531. ([urlseen](#) 13/10/2023).
- [19] Alexander Barosen **and** Sadok Dalin. *Analysis and Comparison of Interfacing, Data Generation and Workload Implementation in BigDataBench 4.0 and Intel HiBench 7.0*. 2018. ([urlseen](#) 28/12/2023).
- [20] Ahmed Oussous, Fatima-Zahra Benjelloun, Ayoub Ait Lahcen **andothers**. “Big Data Technologies: A Survey”. *inJournal of King Saud University - Computer and Information Sciences*: 30.4 (**october** 2018), **pages** 431–448. ISSN: 1319-1578. DOI: 10.1016/j.jksuci.2017.06.001. ([urlseen](#) 28/12/2023).

- [21] Borko Furht **and** Flavio Villanustre. “Introduction to Big Data”. **in***september* 2016: **pages** 3–11. ISBN: 978-3-319-44548-9. DOI: 10.1007/978-3-319-44550-2_1.
- [22] Amanpreet Kaur Sandhu. “Big Data with Cloud Computing: Discussions and Challenges”. **in***Big Data Mining and Analytics: 5.1* (**march** 2022), **pages** 32–40. ISSN: 2096-0654. DOI: 10.26599 / BDMA . 2021 . 9020016. (**urlseen** 28/12/2023).
- [23] *Red Hot: The 2021 Machine Learning, AI and Data (MAD) Landscape*. <https://mattturck.com/data2021/>. **september** 2021. (**urlseen** 28/12/2023).
- [24] Khushboo Kalia **and** Neeraj Gupta. “Analysis of Hadoop MapReduce Scheduling in Heterogeneous Environment”. **in***Ain Shams Engineering Journal: 12.1* (**march** 2021), **pages** 1101–1110. ISSN: 20904479. DOI: 10.1016/j.asej .2020.06.009. (**urlseen** 08/11/2023).
- [25] Kavitha C **and** Anita X. “Task Failure Resilience Technique for Improving the Performance of MapReduce in Hadoop”. **in***ETRI Journal: 42.5* (2020), **pages** 748–760. ISSN: 2233-7326. DOI: 10.4218 / etrij . 2018 - 0265. (**urlseen** 08/11/2023).
- [26] Herodotos Herodotou. *Hadoop Performance Models*. **june** 2011. DOI: 10.48550/arXiv.1106.0940. arXiv: 1106.0940 [cs]. (**urlseen** 08/11/2023).
- [27] M. Bakratsas, P. Basaras, D. Katsaros **and others**. “Hadoop MapReduce Performance on SSDs for Analyzing Social Networks”. **in***Big Data Research: 11* (**march** 2018), **pages** 1–10. ISSN: 22145796. DOI: 10.1016 / j . bdr . 2017.06.001. (**urlseen** 08/11/2023).
- [28] Abolfazl Gandomi, Midia Reshadi, Ali Movaghar **and others**. “HybSMRP: A Hybrid Scheduling Algorithm in Hadoop MapReduce Framework”. **in***Journal of Big Data: 6.1* (**november** 2019), **page** 106. ISSN: 2196-1115. DOI: 10.1186/s40537-019-0253-9. (**urlseen** 08/11/2023).
- [29] *MapReduce - Distributed Computing in Java* 9 [Book]. <https://www.oreilly.com/library/view/distributed-computing-in/9781787126992/5fef6ce5-20d7-4d7c-93eb-7e669d48c2b4.xhtml>. (**urlseen** 08/11/2023).
- [30] *Apache Hadoop*. <https://hadoop.apache.org/>. (**urlseen** 08/11/2023).
- [31] Stathis Maneas **and** Bianca Schroeder. “The Evolution of the Hadoop Distributed File System”. **in***2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*: **may** 2018, **pages** 67–74. DOI: 10.1109/WAINA.2018.00065. (**urlseen** 08/11/2023).
- [32] Chetna Dabas, Parmeet Kaur, Nimisha Gulati **and others**. “Analysis of Comments on Youtube Videos Using Hadoop”. **in***2019 Fifth International Conference on Image Information Processing (ICIIP)*: **november** 2019, **pages** 353–358. DOI: 10.1109 / ICIIP47207 . 2019 . 8985907. (**urlseen** 08/11/2023).

- [33] Tomcy John **and** Pankaj Misra. *Data Lake for Enterprises: Leveraging Lambda Architecture for Building Enterprise Data Lake*. Birmingham, UK, Mumbai: Packt Publishing, 2017. ISBN: 978-1-78728-134-9.
- [34] Swagat Khatai, Siddharth Swarup Rautaray, Swetaleena Sahoo **and others**. “An Implementation of Text Mining Decision Feedback Model Using Hadoop MapReduce”. in*Trends of Data Science and Applications: Theory and Practices*: byeditor Siddharth Swarup Rautaray, Phani Pemmaraju **and** Hrushikesh Mohanty. Studies in Computational Intelligence. Singapore: Springer, 2021, **pages** 273–306. ISBN: 978-981-336-815-6. DOI: 10.1007/978-981-33-6815-6_14. (**urlseen** 09/11/2023).
- [35] Kumar Abhishek, Department of CSE, NIT Patna, Ashok Rajpath, Mahendru, Patna - 800005, Bihar, India, Manish Kumar Verma **and others**. “Integrated Hadoop Cloud Framework (IHCF)”. in*Indian Journal of Science and Technology*: 10.10 (february 2017), **pages** 1–8. ISSN: 0974-5645, 0974-6846. DOI: 10.17485/ijst/2017/v10i10/107943. (**urlseen** 10/11/2023).
- [36] *Apache Hadoop 3.3.6 – HDFS Architecture*. <https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html>. (**urlseen** 10/11/2023).
- [37] Hatim Talal Almansouri **and** Youssef Masmoudi. “Hadoop Distributed File System for Big Data Analysis”. in*2019 4th World Conference on Complex Systems (WCCS)*: Ouarzazate, Morocco: IEEE, **april** 2019, **pages** 1–5. ISBN: 978-1-72811-232-9. DOI: 10.1109/ICoCS.2019.8930804. (**urlseen** 08/11/2023).
- [38] *Apache Hadoop 3.3.6 – Apache Hadoop YARN*. <https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html>. (**urlseen** 28/12/2023).
- [39] *Apache Spark™ - Unified Engine for Large-Scale Data Analytics*. <https://spark.apache.org/>. (**urlseen** 10/11/2023).
- [40] *Apache Spark - Introduction*. https://www.tutorialspoint.com/apache_spark.htm. (**urlseen** 30/12/2023).
- [41] *Intel-Bigdata/HiBench*. Intel-bigdata. **december** 2023. (**urlseen** 30/12/2023).

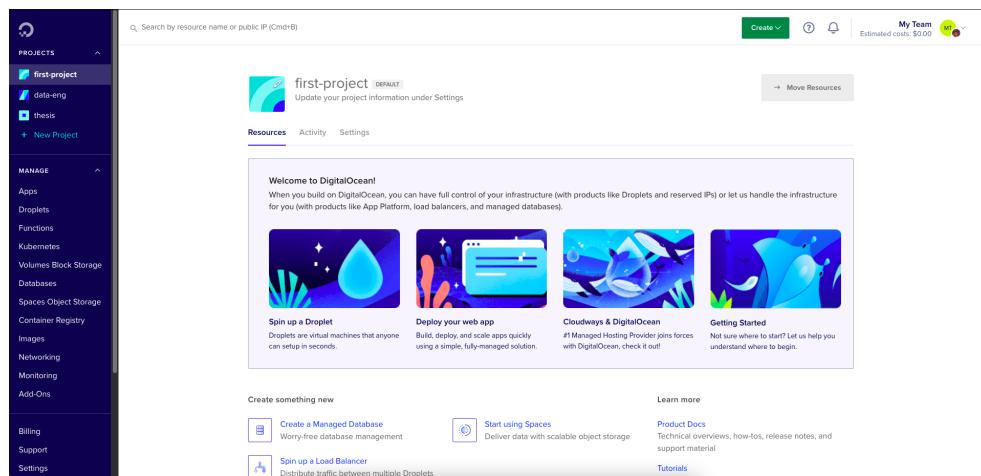
LAMPIRAN

Lampiran A

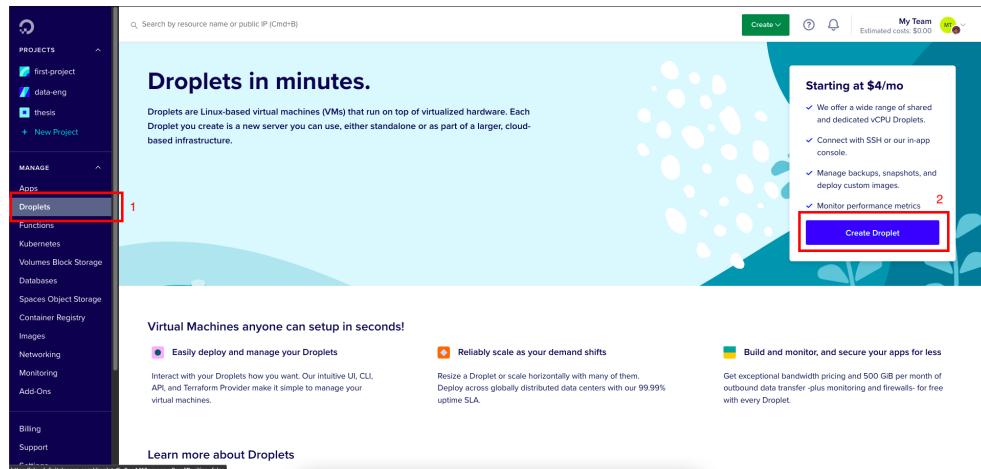
PEMBUATAN VIRTUAL MACHINE (VM) PADA DIGITALOCEAN

Langkah-langkah pembuatan VM pada DigitalOcean dijelaskan seperti berikut,

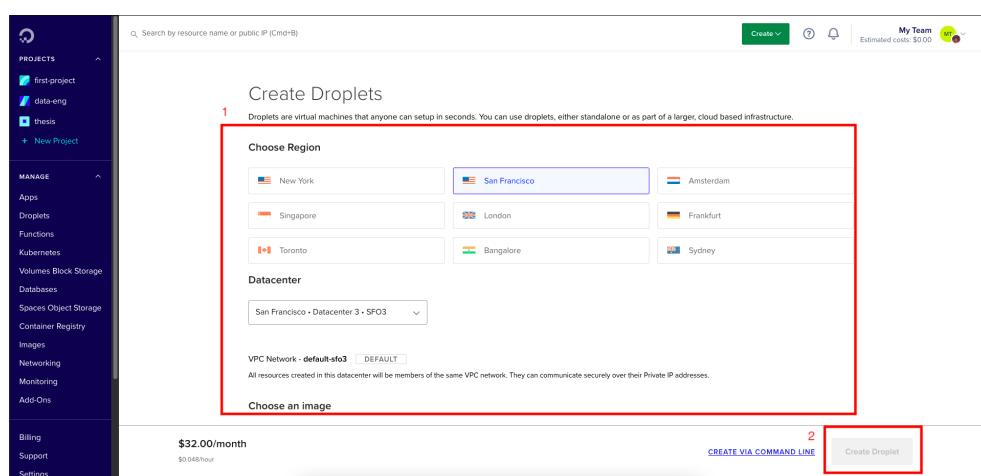
1. Buatlah akun DigitalOcean terlebih dahulu. Jika belum memiliki akun DigitalOcean, disarankan untuk mendaftar melalui *Github Student Developer Pack* sehingga nantinya akan diberikan kredit \$200 secara gratis. Jika sudah memiliki akun DigitalOcean, silakan melakukan *login*.
2. Halaman dasbor DigitalOcean akan ditampilkan setelahnya.



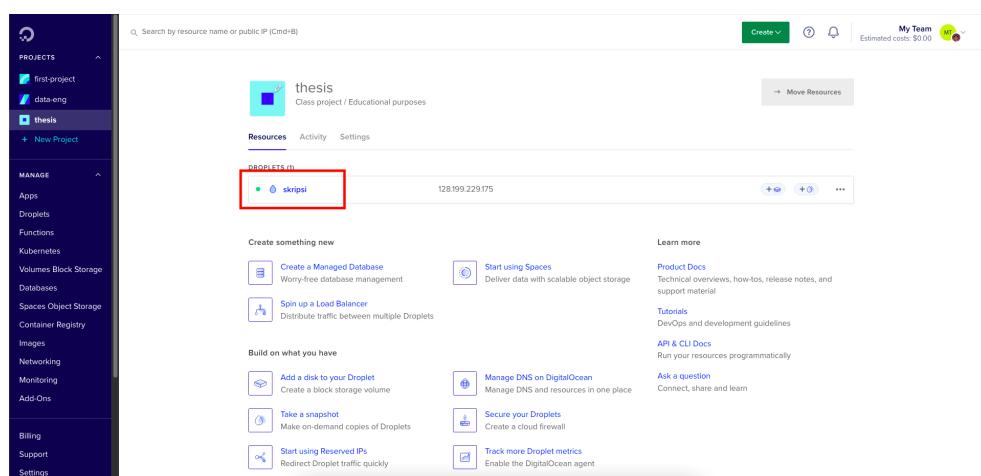
3. Perhatikan menu di sebelah kiri pada laman dasbor DigitalOcean. Tekan Droplets untuk masuk ke laman pembuatan VM. Selanjutnya, tekan *Create Droplets* berwarna biru untuk melakukan konfigurasi VM yang akan dibuat nantinya.



4. Pada laman pembuatan Droplets, lakukan konfigurasi sesuai dengan Tabel 3.1. Jika telah selesai melakukan konfigurasi, tekan *Create Droplets*.

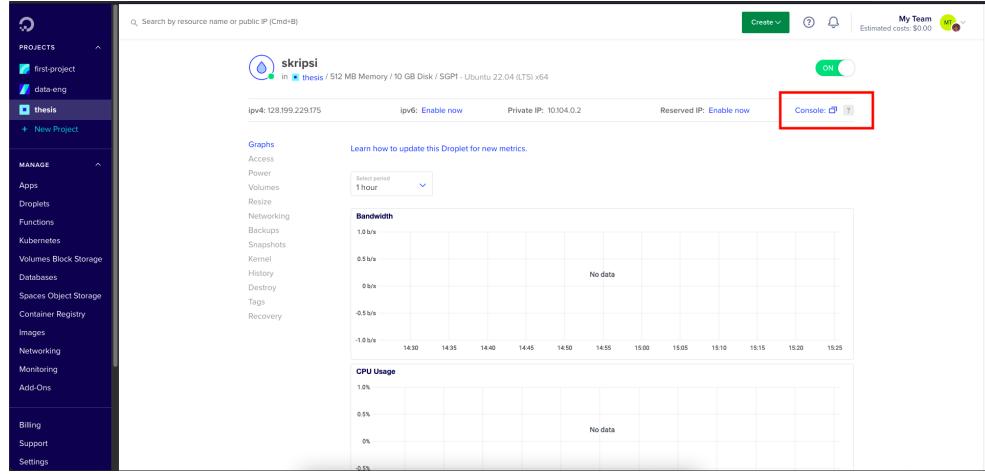


5. Jika pembuatan Droplets berhasil, laman dasbor *Projects* DigitalOcean akan terlihat. Pada bagian Resources akan terlihat Droplets yang baru saja kita buat. Selanjutnya, tekan nama Droplets yang baru saja dibuat.



6. Selanjutnya, laman konfigurasi Droplets akan terlihat. Jika diperlukan konfigurasi lanjutan dapat diatur melalui laman ini. Pada tahap ini hanya akan

fokus pada konfigurasi perangkat lunak tanpa konfigurasi perangkat keras lebih jauh. Untuk masuk ke VM yang sudah dibuat, tekan Console. Tab baru akan dibuka.



7. Akhirnya, lakukan konfigurasi perangkat lunak pada bagian ini.

```

skripsi - DigitalOcean Droplet Web Console
https://cloud.digitalocean.com/droplets/39235116/terminal/ui

Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.15.0-67-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:   https://landscape.canonical.com
 * Support:      https://ubuntu.com/advantage

System information disabled due to load higher than 1.0
Expanded Security Maintenance for Applications is not enabled.

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

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

root@skripsi:-# 

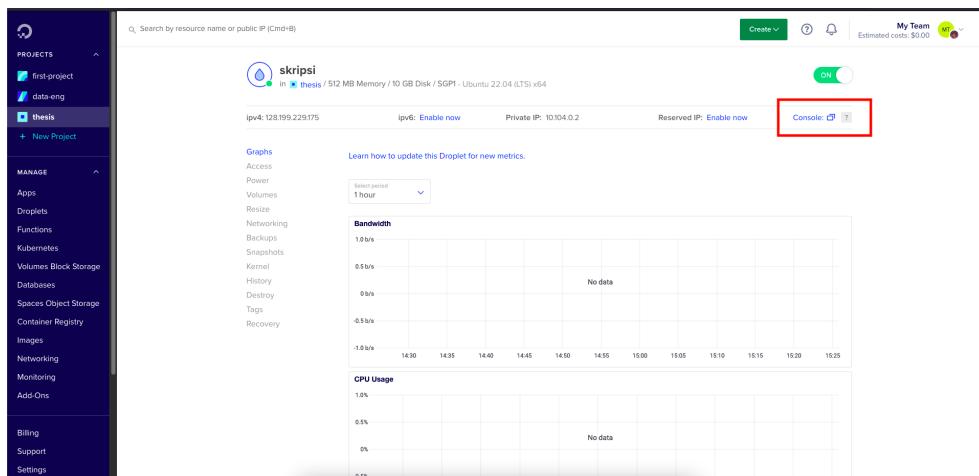
```

Lampiran B

INSTALASI DAN KONFIGURASI PERANGKAT LUNAK PRASYARAT

Pemasangan dan konfigurasi perangkat lunak adalah hal yang krusial. Sebelum dilakukan pemasangan perangkat lunak penyimpanan dan pemrosesan *big data*, tentunya perlu disiapkan perangkat lunak prasyarat. Perangkat lunak prasyarat yang dibutuhkan meliputi Git, Java dan Maven, Python, serta Scala. Langkah-langkah pemasangan dan konfigurasi perangkat lunak akan dijelaskan sebagai berikut,

1. Pastikan Droplets pada DigitalOcean sudah dibuat. Masuk ke *Virtual Machine* (VM) yang sebelumnya sudah dibuat melalui *Console* yang berada pada laman konfigurasi Droplets DigitalOcean.



2. Jika Droplets baru saja dibuat, perlu dilakukan pembaruan *index* pada *package management*. *Package management* adalah sistem atau sekumpulan alat yang digunakan untuk mengotomatiskan penginstalan, peningkatan, konfigurasi, dan penggunaan perangkat lunak. Pembaruan *package management* dapat dilakukan dengan `sudo apt update`.
3. Membuat Pengguna Baru
 - (a) Pertama, buatlah grup baru yang bernama *hadoop* dengan perintah `sudo addgroup hadoop`.
 - (b) Kemudian, tambahkan pengguna baru *hdfsuser* dalam grup *hadoop* yang sama dengan perintah `sudo adduser --ingroup hadoop hdfsuser`.
 - (c) Berikan *hdfsuser* izin *root* yang diperlukan untuk pemasangan file. Hak istimewa pengguna *root* dapat diberikan dengan memperbarui file *sudoers*. Buka file *sudoers* dengan menjalankan perintah `sudo visudo`. Tambahkan baris berikut, yaitu `hdfsuser ALL=(ALL:ALL) ALL`.
 - (d) Sekarang, simpan perubahan dan tutup editor.

- (e) Selanjutnya, mari beralih ke pengguna baru yang telah dibuat untuk instalasi lebih lanjut menggunakan perintah su - hdfsuser.

4. Pengaturan *SSH keys* untuk Hadoop

- (a) Hadoop menggunakan *Secure Shell* (SSH) untuk menjalankan proses antara *master nodes* dan *slave nodes*. Penggunaan SSH akan memberikan banyak keuntungan, salah satunya adalah kecepatan. Jika sebuah klaster aktif dan berjalan, komunikasi antar *nodes* akan berjalan terlalu sering. Begitu pula dengan *job tracker* yang harus sering mengirimkan informasi *task to task* dengan cepat. Lakukan pemasangan ssh dan sshd dengan cara sudo apt-get install ssh dan sudo apt-get install sshd pada terminal.
- (b) Selanjutnya, lakukan pembuatan *SSH keys* dengan cara ssh-keygen -t rsa. Jika pembuatan *SSH keys* sudah dilakukan, jalankan perintah cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys.
- (c) Ubah perizinan berkas dengan perintah chmod og-wx ~/.ssh/authorized_keys.
- (d) Terakhir, untuk memverifikasi koneksi aman sudah terjadi, lakukan ssh localhost.

5. Instalasi Git

- (a) Git dapat dipasang menggunakan perintah sudo apt install git. Pengguna akan diminta konfirmasi untuk menginstall. Ketik y kemudian tekan enter.
- (b) Untuk mengecek versi Git, dapat menggunakan perintah git --version.

6. Instalasi Python

- (a) Python dapat dipasang menggunakan perintah sudo apt-get install python2 && sudo apt install python3.7. Pengguna akan diminta konfirmasi untuk menginstall. Ketik y kemudian tekan enter.
- (b) Untuk mengecek versi Python, dapat menggunakan perintah python --version.

7. Instalasi Java 8 dan Maven

- (a) Java 8 dapat dipasang menggunakan perintah sudo apt install openjdk-8-jre-headless openjdk-8-jdk. Pengguna akan diminta konfirmasi untuk menginstall. Ketik y kemudian tekan enter.
- (b) Versi dari Java dapat dilihat menggunakan perintah java -version.

- (c) Selanjutnya, instalasi Maven dapat dilakukan menggunakan perintah `sudo apt-get -y install maven`.
- (d) Informasi dari Maven beserta Java yang digunakan dapat dilihat menggunakan perintah `mvn -version`.

8. Instalasi Scala

- (a) Scala yang akan dipasang adalah versi 2.12. Jika menggunakan manajer paket, versi yang akan dipasang adalah versi terbaru. Untuk mengunduh versi spesifik dari Scala, dapat menggunakan perintah `sudo wget https://downloads.lightbend.com/scala/2.12.0/scala-2.12.0.deb`.
- (b) Scala dapat dipasang menggunakan perintah `sudo dpkg -i scala-2.12.0.deb`.
- (c) Versi Scala dapat dilihat melalui perintah `scala -version`.

Lampiran C

INSTALASI DAN KONFIGURASI HADOOP

Langkah-langkah pemasangan dan konfigurasi Hadoop akan dijelaskan sebagai berikut,

1. Unduh Hadoop
 - (a) Pastikan perangkat lunak prasyarat sudah berhasil dipasang dan dilakukan konfigurasi. Sebelum dilakukan pemasangan Hadoop, diperlukan untuk mengunduh berkas Hadoop terlebih dahulu dengan perintah `cd /usr/local`, dilanjutkan dengan `sudo wget https://archive.apache.org/dist/hadoop/common/hadoop-3.2.0/hadoop-3.2.0.tar.gz`.
 - (b) Ekstrak berkas Hadoop yang sudah diunduh tadi dengan perintah `sudo tar xvzf hadoop-3.2.0.tar.gz`. Hasil ekstrak berkas Hadoop akan disimpan pada direktori yang sama.
 - (c) Selanjutnya, untuk memudahkan kedepannya, ganti nama folder Hadoop dengan perintah `sudo mv hadoop-3.2.0 hadoop`.
2. Mengubah Kepemilikan Berkas Hadoop
 - (a) Setelah berkas Hadoop sudah berhasil terunduh, selanjutnya ubah kepemilikan berkas Hadoop ke `hdfsuser` yang sebelumnya sudah kita buat dengan perintah `sudo chown -R hdfsuser:hadoop /usr/local/hadoop`.
 - (b) Tambahkan kekuasaan untuk membaca, menulis, dan mengeksekusi pada foler Hadoop dengan perintah `sudo chmod -R 777 /usr/local/hadoop`.
3. Mematikan *IPv6 Networks*
 - (a) Saat ini Hadoop belum mendukung penggunaan *IPv6 Networks*. Hadoop hanya dibangun dan diuji coba pada *IPv4 Networks*. Untuk mematikan IPv6, dapat dimulai dengan menjalankan perintah `cat /proc/sys/net/ipv6/conf/all/disable_ipv6`.
 - (b) Jika hasil yang diberikan bukan angka 1, maka beberapa langkah tambahan harus dijalankan. Jalankan perintah `sudo nano /etc/sysctl.conf`, kemudian tambahkan beberapa baris potongan kode berikut pada akhir berkas,

```
1 # Disable ipv6
2 net.ipv6.conf.all.disable_ipv6=1
3 net.ipv6.conf.default_disable_ipv6=1
4 net.ipv6.conf.lo.disable_ipv6=1
5
```

- (c) Simpan berkas. Kemudian jalankan perintah `sudo sysctl -p` untuk mengaktifkan perubahan.

4. Menambahkan Hadoop pada *Environments Variables*

- (a) Hadoop perlu ditambahkan pada *Environments Variabels* untuk memudahkan dalam melakukan eksekusi. Untuk menambahkannya, jalankan perintah `sudo nano ~/.bashrc`.
 - (b) Tambahkan beberapa baris kode berikut pada akhir berkas *bashrc*.

```
1 # HADOOP ENVIRONMENT
2 export HADOOP_HOME=/usr/local/hadoop
3 export HADOOP_CONF_DIR=/usr/local/hadoop/etc/
4 hadoop
5     export HADOOP_MAPRED_HOME=/usr/local/hadoop
6     export HADOOP_COMMON_HOME=/usr/local/hadoop
7     export HADOOP_HDFS_HOME=/usr/local/hadoop
8     export YARN_HOME=/usr/local/hadoop
9     export PATH=$PATH:/usr/local/hadoop/bin
10    export PATH=$PATH:/usr/local/hadoop/sbin
11
12 # HADOOP NATIVE PATH
13 export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/
14 lib/native
15     export HADOOP_OPTS=-Djava.library.path=
16 $HADOOP_PREFIX/lib
```

- (c) Untuk mendapatkan perubahan dapat dilakukan dengan perintah source `~/.bashrc`.

5. Konfigurasi Hadoop

- (a) Hadoop menggunakan berkas .xml untuk melakukan konfigurasi pada semua prosesnya. Biasanya, letak direktori untuk melakukan konfigurasi terletak pada \$HADOOP_HOME/etc/hadoop. Oleh karena itu, jalankan perintah `cd /usr/local/hadoop/etc/hadoop/`.
 - (b) Konfigurasi berkas *hadoop-env.sh* dapat dilakukan dengan perintah `sudo nano hadoop-env.sh`, dilanjutkan dengan menambahkan beberapa baris kode seperti di bawah ini,

```
1 export HADOOP_OPTS=-Djava.net.preferIPv4Stack=true
2 export JAVA_HOME=/usr
3 export HADOOP_HOME_WARN_SUPPRESS="TRUE"
4 export HADOOP_ROOT_LOGGER="WARN,DRFA"
5 export HDFS_NAMENODE_USER="hdfsuser"
6 export HDFS_DATANODE_USER="hdfsuser"
7 export HDFS_SECONDARYNAMENODE_USER="hdfsuser"
8 export YARN_RESOURCEMANAGER_USER="hdfsuser"
9 export YARN_NODEMANAGER_USER="hdfsuser"
```

- (c) Konfigurasi berkas *yarn-site.xml* dapat dilakukan dengan perintah `sudo nano yarn-site.xml`, dilanjutkan dengan menambahkan beberapa baris kode seperti di bawah ini,

```
1 <property>
2   <name>yarn.nodemanager.aux-services</name>
3   <value>mapreduce_shuffle</value>
4 </property>
5 <property>
6   <name>yarn.nodemanager.aux-services.mapreduce.
7     shuffle.class</name>
8   <value>org.apache.hadoop.mapred.ShuffleHandler</
9     value>
10 </property>
```

- (d) Konfigurasi berkas *hdfs-site.xml* dapat dilakukan dengan perintah `sudo nano hdfs-site.xml`, dilanjutkan dengan menambahkan beberapa baris kode seperti di bawah ini,

```
1 <property>
2   <name>dfs.replication</name>
3   <value>1</value>
4 </property>
5 <property>
6   <name>dfs.namenode.name.dir</name>
7   <value>/usr/local/hadoop/yarn_data/hdfs/namenode</
8     value>
9 </property>
10 <property>
11   <name>dfs.datanode.data.dir</name>
12   <value>/usr/local/hadoop/yarn_data/hdfs/datanode</
13     value>
14 </property>
15 <property>
16   <name>dfs.namenode.http-address</name>
17   <value>localhost:50070</value>
18 </property>
```

- (e) Konfigurasi berkas *core-site.xml* dapat dilakukan dengan perintah `sudo nano core-site.xml`, dilanjutkan dengan menambahkan beberapa baris kode seperti di bawah ini,

```
1 <property>
2   <name>hadoop.tmp.dir</name>
3   <value>/bigdata/hadoop/tmp</value>
4 </property>
5 <property>
6   <name>fs.default.name</name>
7   <value>hdfs://localhost:9000</value>
8 </property>
```

- (f) Konfigurasi berkas *mapred-site.xml* dapat dilakukan dengan perintah

`sudo nano mapred-site.xml`, dilanjutkan dengan menambahkan beberapa baris kode seperti di bawah ini,

```
1 <property>
2   <name>mapred.framework.name</name>
3   <value>yarn</value>
4 </property>
5 <property>
6   <name>mapreduce.jobhistory.address</name>
7   <value>localhost:10020</value>
8 </property>
9
```

6. Membuat Direktori Hadoop untuk Menyimpan Data

- (a) Sesuai dengan apa yang ditulis pada `core-site.xml`, langkah pertama yang harus dilakukan adalah membuat direktori sementara untuk dfs menyimpan berkas dengan menjalankan perintah di bawah. Jalankan perintah berikut baris per baris.

```
1 sudo mkdir -p /bigdata/hadoop/tmp
2 sudo chown -R hdfsuser:hadoop /bigdata/hadoop/tmp
3 sudo chmod -R 777 /bigdata/hadoop/tmp
4
```

- (b) Selanjutnya, jalankan perintah berikut untuk membuat direktori untuk menyimpan berkas data sekaligus mengganti kepemilikan berkas. Jalankan perintah berikut baris per baris.

```
1 sudo mkdir -p /usr/local/hadoop/yarn_data/hdfs/
namenode
2 sudo mkdir -p /usr/local/hadoop/yarn_data/hdfs/
datanode
3 sudo chmod -R 777 /usr/local/hadoop/yarn_data/hdfs/
/namenode
4 sudo chmod -R 777 /usr/local/hadoop/yarn_data/hdfs/
/datanode
5 sudo chown -R hdfsuser:hadoop /usr/local/hadoop/
yarn_data/hdfs/namenode
6 sudo chown -R hdfsuser:hadoop /usr/local/hadoop/
yarn_data/hdfs/datanode
7
```

- (c) Konfigurasi untuk Hadoop sudah selesai dan dapat dilanjutkan untuk menjalankan *Resource Manager* dan *Node Manager*

7. Menjalankan Hadoop

- (a) Sebelum menjalankan *Hadoop Core Services*, klaster harus dibersihkan dengan cara melakukan *format* pada *namenode*. Jalankan perintah `hdfs namenode -format`.
- (b) Untuk menjalankan layanan Hadoop, dapat dilakukan dengan perintah `start-all.sh`.

- (c) Perintah `jps` dapat dilakukan untuk mengecek apakah layanan Hadoop sudah berjalan.
- (d) Untuk memberhentikan layanan Hadoop, dapat dilakukan dengan perintah `stop-all.sh` pada terminal.

Lampiran D

INSTALASI DAN KONFIGURASI SPARK

Langkah-langkah pemasangan dan konfigurasi Spark akan dijelaskan sebagai berikut,

1. Unduh Berkas Spark

- (a) Pastikan perangkat lunak prasyarat sudah berhasil dipasang dan dilakukan konfigurasi. Sebelum dilakukan pemasangan Spark, diperlukan untuk mengunduh berkas Spark terlebih dahulu dengan perintah `cd /usr/local`, dilanjutkan dengan `sudo wget https://archive.apache.org/dist/spark/spark-3.0.0/spark-3.0.0-bin-hadoop3.2.tgz`.
- (b) Ekstrak berkas Spark yang sudah diunduh tadi dengan perintah `sudo tar xvf spark-3.0.0-bin-hadoop3.2.tgz`. Hasil ekstrak berkas Spark akan disimpan pada direktori yang sama.
- (c) Selanjutnya, untuk memudahkan kedepannya, ganti nama folder Spark dengan perintah `sudo mv spark-3.0.0-bin-hadoop3.2 spark`.

2. Menambahkan Spark pada *Environments Variables*

- (a) Spark perlu ditambahkan pada *Environments Variables* untuk memudahkan dalam melakukan eksekusi. Untuk menambahkannya, jalankan perintah `sudo nano ~/.bashrc`.
- (b) Tambahkan beberapa baris kode berikut pada akhir berkas *bashrc*.

```
1 # SPARK ENVIRONMENT
2 export PATH=$PATH:/usr/local/spark/bin
3 export YARN_CONF_DIR=$HADOOP_HOME/etc/hadoop
4 export SPARK_HOME=$PATH:/usr/local/spark/bin
5
```

- (c) Untuk mendapatkan perubahan dapat dilakukan dengan perintah `source ~/.bashrc`.

3. Menjalankan *Spark Shell*

- (a) Pastikan bahwa Spark sudah ditambahkan pada *environments variables* dengan perintah `spark-submit --version`.
- (b) Jalankan layanan Hadoop dengan perintah `start-all.sh`.
- (c) Jalankan `spark-shell` dengan YARN menggunakan perintah `spark-shell --master yarn`.

Lampiran E

INSTALASI DAN KONFIGURASI HIBENCH

Langkah-langkah pemasangan dan konfigurasi HiBench akan dijelaskan sebagai berikut,

1. Unduh berkas HiBench
 - (a) Pastikan perangkat lunak prasyarat dan perangkat lunak *Big Data* sebelumnya sudah berhasil dipasang dan dilakukan konfigurasi. Tahap selanjutnya adalah mengunduh berkas HiBench dengan perintah
`git clone https://github.com/Intel-bigdata/HiBench.git`. Pastikan berada pada folder `/home/hdfsuser`.
 - (b) Selanjutnya, berikan perizinan ke folder HiBench dengan cara
`sudo chmod 755 HiBench`.
2. Membangun *framework benchmark*
 - (a) Sebelum HiBench dapat digunakan, diperlukan pembangunan beberapa modul yang dibutuhkan, misalnya modul *data generation*, modul *hadoopbench*, dan modul *sparkbench*. Langkah awal yang diperlukan adalah masuk ke folder HiBench dengan perintah `cd HiBench`.
 - (b) Selanjutnya, pastikan versi Hadoop, Spark, dan Scala sudah sesuai. Jalankan perintah
`mvn -Phadoopbench -Psparkbench -Dhadoop=2.6 -Dspark=1.5 -Dmodules -Pmicro clean package`. Perintah ini akan membangun modul yang dibutuhkan menggunakan Maven. Tidak semua modul akan dipasang. Modul yang akan dipasang salah satunya adalah modul untuk *micro benchmark*.
 - (c) Jika ingin pembangunan modul yang lebih luas cakupannya dapat menggunakan perintah
`mvn -Phadoopbench -Psparkbench -Dspark=1.5 -Dscala=2.11 clean package`.
3. Jalankan perintah `sudo apt install bc` supaya berkas Hibench *Report* dapat muncul.
4. Lakukan konfigurasi pada berkas *hibench.conf*. Buka berkas tersebut dengan perintah `sudo nano conf/hibench.conf`.
5. Lakukan beberapa perubahan pada baris sesuai dengan contoh di bawah ini.

```
1      hibench.masters.hostnames          hdfs://localhost:9000
2      hibench.slaves.hostnames           localhost
3
```

6. Menjalankan *Benchmark Hadoop*

- (a) Lakukan konfigurasi pada berkas *hadoop.conf*. Sebelum itu, salin *template* konfigurasinya dengan perintah `cp conf/hadoop.conf.template conf/hadoop.conf`.

(b) Buka berkas *hadoop.conf* dengan perintah `sudo nano conf/hadoop.conf`. Selanjutnya, lakukan beberapa perubahan seperti pada contoh di bawah.

```
1 # Hadoop home
2 hibench.hadoop.home      /usr/local/hadoop
3
4 # The path of hadoop executable
5 hibench.hadoop.executable ${hibench.hadoop.
6 home}/bin/hadoop
7
8 # Hadoop configuration directory
9 hibench.hadoop.configure.dir ${hibench.hadoop.
10 home}/etc/hadoop
11
12 # The root HDFS path to store HiBench data
13 hibench.hdfs.master      hdfs://localhost:9000
14
15 # Hadoop release provider. Supported value: apache
16 hibench.hadoop.release    apache
```

- (c) Simpan perubahan yang sudah dilakukan.
 - (d) Untuk menjalankan beban kerja yang sudah dirancang sebelumnya, pertah yang digunakan sebagai berikut. Jalankan baris per baris.

```
1     bin/workloads/micro/<nama-beban-kerja>/prepare/  
2         prepare.sh  
3     bin/workloads/micro/<nama-beban-kerja>/hadoop/  
4         run.sh
```

- (e) Untuk melihat hasil dari *benchmark* dapat mengakses berkas pada `<HiBench_Root>/report/hibench.report`

7. Menjalankan *Benchmark* Spark

- (a) Lakukan konfigurasi pada berkas *spark.conf*. Sebelum itu, salin *template* konfigurasinya dengan perintah `cp conf/spark.conf.template conf/spark.conf`.
 - (b) Buka berkas *spark.conf* dengan perintah `sudo nano conf/spark.conf`. Selanjutnya, lakukan beberapa perubahan seperti pada contoh di bawah.

```
1 # Spark home
2 hibench.spark.home          /usr/local/spark
3
4 # Spark master
5 #   standalone mode: spark://xxx:7077
6 #   YARN mode: yarn-client
```

```
7      hibench.spark.master      yarn  
8
```

- (c) Simpan perubahan yang sudah dilakukan.
- (d) Untuk menjalankan beban kerja yang sudah dirancang sebelumnya, pertah yang digunakan sebagai berikut. Jalankan baris per baris.

```
1      bin/workloads/micro/<nama-beban-kerja>/prepare/  
2          prepare.sh  
3      bin/workloads/micro/<nama-beban-kerja>/spark/run  
.sh
```

- (e) Untuk melihat hasil dari *benchmark* dapat mengakses berkas pada <HiBench_Root>/report/hibench.report