Desain Infrastruktur: Kubernetes dan Hadoop sebagai Penyimpanan Data Terdistribusi

Rangga Perwiratama[*1]

[1] Program Studi Sistem Informasi, Universitas Atma Jaya Yogyakarta

E-mail: rangga.perwiratama@uajy.ac.id*[1]

Abstrak. Penelitian tentang Sistem Penyimpanan Data Terdistribusi (*Distributed Data Storage Systems/*DDSS) seperti Hadoop mengalami beberapa tantangan di masa lalu karena integrasi teknik penyimpanan tradisional dengan aplikasi penyimpanan terdistribusi yang kompleks. Namun, kemajuan dalam teknologi kontainerisasi dan manajemen klaster telah membuka jalan untuk beralih dari solusi yang berpusat pada perangkat lunak menjadi solusi yang berpusat pada infrastruktur. Makalah ini mengusulkan pemanfaatan Kubernetes sebagai lingkungan kerja untuk ekosistem Hadoop. Pendekatan kami menunjukkan bahwa logika dasar dari klaster Hadoop dapat diadaptasi dengan mulus ke klaster Kubernetes, meningkatkan efisiensi dan skalabilitas dalam penyimpanan dan pemrosesan data.

Kata kunci: Distributed Data Storage Systems (DDSS), Kontainerisasi, Kubernetes, Hadoop, Manajemen Klaster

Abstract. *The research on Distributed Data Storage Systems (DDSS) such as Hadoop has historically faced challenges due to the complex integration of traditional storage techniques with distributed storage applications. However, advancements in containerization and cluster management technologies have paved the way for transitioning from software-centric to infrastructure-centric solutions. This paper proposes leveraging Kubernetes as the working environment for the Hadoop ecosystem. Our approach demonstrates that the fundamental logic of a Hadoop cluster can be seamlessly adapted to a Kubernetes cluster, enhancing efficiency and scalability in data storage and processing.*

Keywords: *Distributed Data Storage Systems (DDSS), Containerization, Kubernetes, Hadoop, Cluster Management*

1. Pendahuluan

Hadoop adalah framework open-source yang dirancang untuk penyimpanan dan pemrosesan dataset besar yang terdistribusi [1]. Awalnya dikembangkan oleh Doug Cutting dan Mike Cafarella dan sekarang dikelola oleh Apache Software Foundation. Arsitektur Hadoop dibangun untuk menangani data dalam jumlah besar dengan mendistribusikan beban kerja ke banyak mesin dalam satu kluster [2]. Komponen intinya meliputi Hadoop Distributed File System (HDFS) untuk penyimpanan, dan MapReduce, model pemrograman untuk pemrosesan data. Hadoop banyak digunakan dalam analisis big data, yang memungkinkan organisasi memperoleh wawasan dari data mereka secara efisien dan hemat biaya [3].

Kubernetes, yang sering disingkat K8s, adalah platform open source orkestrasi kontainer yang dikembangkan oleh Google dan kini dikelola oleh Cloud Native Computing Foundation (CNCF) [4]. Platform ini mengotomatiskan penyebaran, penskalaan, dan pengelolaan aplikasi yang dikontainerisasi, yang ringan, portabel, dan berjalan secara konsisten di berbagai lingkungan [5]. Kubernetes mengabstraksi infrastruktur yang mendasarinya, yang memungkinkan pengembang untuk fokus membangun aplikasi daripada mengelola infrastruktur [6]. Fitur utamanya meliputi peluncuran dan pembatalan otomatis, penemuan layanan, penyeimbangan beban, orkestrasi penyimpanan, dan kemampuan pemulihan mandiri, yang menjadikannya alat yang hebat untuk mengelola aplikasi kompleks yang terdistribusi [7].

Menggabungkan Hadoop dengan Kubernetes menghadirkan beberapa keuntungan signifikan yang mengatasi keterbatasan penerapan Hadoop tradisional. Salah satu manfaat utamanya adalah peningkatan pemanfaatan sumber daya. Dalam pengaturan Hadoop yang umum, sumber daya sering kali dialokasikan secara statis, yang menyebabkan inefisiensi dan pemanfaatan yang kurang [8], [9]. Kubernetes, dengan alokasi sumber daya yang dinamis dan kemampuan orkestrasi kontainernya, memastikan bahwa sumber daya digunakan secara optimal. Dengan menjalankan Hadoop di Kubernetes, organisasi dapat mencapai manajemen sumber daya yang lebih baik, yang mengarah pada penghematan biaya dan peningkatan kinerja. Elastisitas yang disediakan oleh Kubernetes memungkinkan penskalaan klaster Hadoop ke atas atau ke bawah berdasarkan permintaan, yang memastikan bahwa infrastruktur dapat menangani berbagai beban kerja secara efisien.

Keunggulan penting lainnya adalah penyederhanaan penerapan dan pengelolaan. Pengelolaan klaster Hadoop bisa jadi rumit dan memakan waktu, memerlukan pengetahuan khusus dan intervensi manual yang signifikan [10]. Kubernetes mengabstraksikan sebagian besar kerumitan ini melalui fitur-fitur otomatisasinya. Kubernetes menyediakan peluncuran otomatis, pembatalan, dan kemampuan pemulihan mandiri, yang menyederhanakan penerapan dan pemeliharaan klaster Hadoop. Hal ini mengurangi overhead operasional dan menurunkan hambatan masuk bagi organisasi yang ingin memanfaatkan Hadoop untuk pemrosesan big data [11]. Selain itu, pengelolaan konfigurasi deklaratif Kubernetes memastikan konsistensi di seluruh lingkungan kerja, sehingga memudahkan penerapan klaster Hadoop di lingkungan hybrid dan multi-cloud [10], [12].

Integrasi Hadoop dengan Kubernetes juga meningkatkan kelincahan dan fleksibilitas operasi big data secara keseluruhan. Kubernetes dirancang untuk mendukung aplikasi berbasis cloud dan terintegrasi secara mulus dengan berbagai layanan dan perangkat cloud [13], [14]. Kompatibilitas ini memungkinkan organisasi untuk membangun jalur data yang lebih tangguh dan terukur, dengan menggabungkan komponen berbasis cloud lainnya seperti Apache Spark, Kafka, dan berbagai solusi penyimpanan yang ada [15]. Dengan menjalankan Hadoop di Kubernetes, organisasi dapat memanfaatkan ekosistem berbasis cloud yang lebih luas, yang menghasilkan solusi data yang lebih inovatif dan komprehensif. Kombinasi ini memberdayakan organisasi untuk merespons perubahan persyaratan bisnis dengan lebih cepat dan berinovasi dengan kecepatan yang lebih tinggi [16], [17].

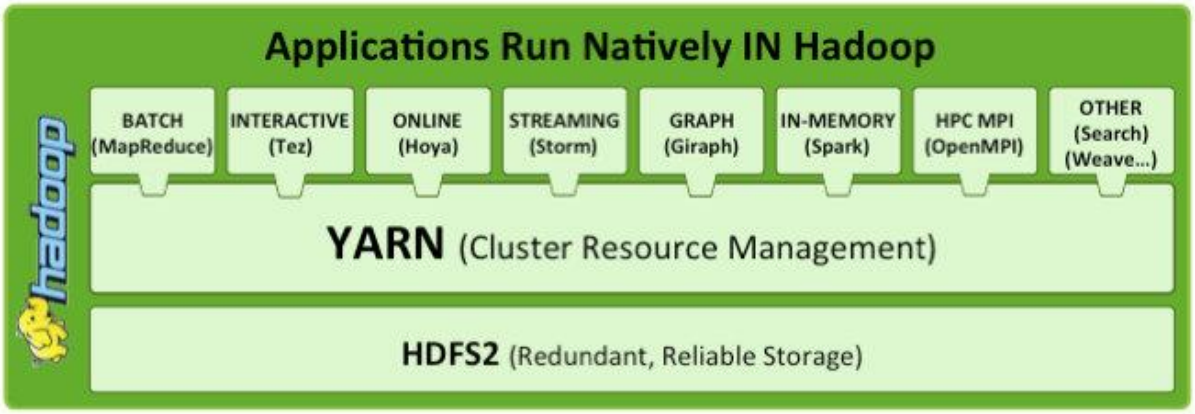
Menggabungkan Hadoop dengan Kubernetes memanfaatkan kekuatan kedua teknologi tersebut, sehingga memungkinkan pemrosesan big data yang efisien dan terukur dalam lingkungan yang terkontainerisasi. Menjalankan Hadoop di Kubernetes menawarkan beberapa manfaat, seperti pemanfaatan sumber daya yang lebih baik, pengelolaan yang lebih mudah, dan integrasi yang lebih baik dengan ekosistem berbasis cloud. Sinergi ini memungkinkan organisasi untuk menerapkan kluster Hadoop secara dinamis, menskalakannya sesuai permintaan, dan menyederhanakan operasi melalui kemampuan otomatisasi Kubernetes, yang pada akhirnya menghasilkan solusi big data yang lebih fleksibel dan hemat biaya.

Tujuan utama dari makalah ini adalah untuk mengeksplorasi integrasi Hadoop dengan Kubernetes dan mengevaluasi manfaat dan tantangan dari kombinasi tersebut. Dengan meneliti teknologi yang mendasarinya, makalah ini bertujuan untuk memberikan pemahaman yang komprehensif tentang bagaimana Kubernetes dapat meningkatkan penerapan, pengelolaan, dan kinerja kluster Hadoop. Ini mencakup analisis terperinci tentang alokasi sumber daya, skalabilitas, dan otomatisasi, serta penilaian

implikasi praktis dan kasus penggunaan potensial bagi organisasi yang ingin memanfaatkan kedua teknologi tersebut. Makalah ini berupaya untuk menawarkan wawasan dan pedoman untuk mengimplementasikan Hadoop secara efektif pada Kubernetes, mengatasi rintangan umum, dan menyoroti praktik terbaik. Tujuan penting lainnya adalah untuk menunjukkan aplikasi dunia nyata dan tolok ukur kinerja Hadoop yang berjalan pada Kubernetes. Melalui studi kasus dan contoh, makalah ini akan menggambarkan bagaimana integrasi ini dapat menghasilkan pemrosesan big data yang lebih efisien dan pemanfaatan sumber daya yang lebih baik. Dengan memberikan contoh konkret dan data empiris, makalah ini bertujuan untuk memvalidasi manfaat teoritis yang dibahas dan menawarkan kerangka kerja yang jelas dan dapat ditindaklanjuti bagi organisasi untuk mengadopsi pendekatan ini. Tujuan utamanya adalah untuk membekali pembaca dengan pengetahuan dan alat yang dibutuhkan untuk berhasil menerapkan dan mengelola kluster Hadoop di lingkungan Kubernetes, sehingga membuka kemungkinan baru untuk solusi big data yang dapat diskalakan, fleksibel, dan hemat biaya.

2. Metode

Komponen inti Hadoop meliputi Hadoop Distributed File System (HDFS), MapReduce, dan Yet Another Resource Negotiator (YARN). HDFS adalah lapisan penyimpanan yang dirancang untuk penyimpanan yang toleran terhadap kesalahan dan throughput tinggi dari kumpulan data besar dengan mendistribusikan data ke beberapa mesin dalam satu kluster. MapReduce menyediakan model pemrograman untuk memproses data secara paralel, di mana data dibagi menjadi beberapa bagian untuk diproses oleh fungsi map dan kemudian diagregasi oleh fungsi reduce. YARN, yang diperkenalkan di Hadoop 2.0, memisahkan manajemen sumber daya dari pemrosesan data, mendukung beberapa mesin seperti Apache Spark dan Tez. Arsitektur YARN mencakup ResourceManager untuk manajemen sumber daya secara keseluruhan dan NodeManager untuk sumber daya node individual, yang meningkatkan skalabilitas dan fleksibilitas. Ilustrasi arsitektur Hadoop dapat dilihat pada Gambar 1.

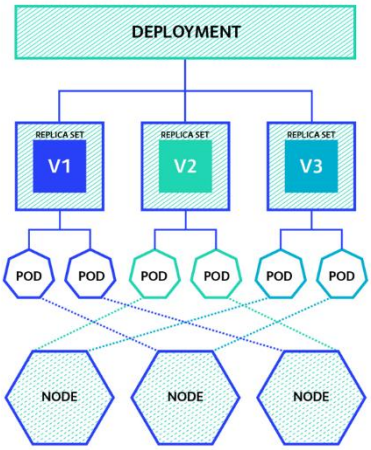


Gambar 1. Arsitektur Hadoop

Bersama-sama, komponen-komponen ini membentuk tulang punggung Hadoop. HDFS memastikan penyimpanan yang andal dan dapat diskalakan, MapReduce memfasilitasi pemrosesan data paralel yang efisien, dan YARN mengoptimalkan pemanfaatan sumber daya dan mendukung beragam mesin pemrosesan. Integrasi ini memungkinkan Hadoop untuk menangani berbagai beban kerja dan aplikasi big data, menjadikannya alat yang hebat untuk analisis data dan pengambilan keputusan di bidang-bidang seperti keuangan, perawatan kesehatan, dan e-commerce.

Kubernetes, yang sering disingkat K8s, dibangun berdasarkan beberapa konsep inti: Pod, Service, dan Deployment. Pod adalah unit terkecil yang dapat di-deploy di Kubernetes, yang mewakili satu contoh proses yang sedang berjalan di dalam kluster. Setiap Pod dapat berisi satu atau beberapa kontainer yang berbagi namespace jaringan dan volume penyimpanan yang sama, yang memungkinkan container untuk

berkomunikasi dengan mudah dan bekerja bersama sebagai satu unit. Service, di sisi lain, menyediakan titik akhir yang stabil untuk mengakses Pod. Service mengabstraksikan detail yang mendasari tentang bagaimana Pod didistribusikan di seluruh kluster, yang memungkinkan komunikasi yang lancar antara berbagai bagian aplikasi. Service menggunakan label dan selector untuk secara dinamis merutekan lalu lintas ke Pod yang sesuai, memastikan ketersediaan tinggi dan penyeimbangan beban.



Gambar 2. Arsitektur Deployment pada Kubernetes

Deployment adalah abstraksi tingkat tinggi untuk mengelola siklus hidup Pod. Deployment menentukan status aplikasi yang diinginkan, termasuk jumlah replika, image kontainer yang akan digunakan, dan strategi pembaruan. Deployment memungkinkan pembaruan bergulir, yang memungkinkan perubahan diterapkan secara bertahap tanpa waktu henti, dan rollback, yang memungkinkan sistem kembali ke status sebelumnya jika terjadi kesalahan. Dengan menentukan Deployment, pengembang dapat dengan mudah meningkatkan atau menurunkan skala aplikasi, mengelola pembaruan, dan memastikan konsistensi di seluruh kluster. Bersama-sama, Pod, Service, dan Deployment membentuk fondasi Kubernetes, yang memungkinkan orkestrasi, penskalaan, dan pengelolaan aplikasi yang dikontainerisasi secara andal dan efisien. Ilustrasinya dapat dilihat pada Gambar 2.

Sebelum menerapkan Hadoop pada Kubernetes, beberapa prasyarat dan pengaturan lingkungan penting dilakukan untuk memastikan implemetasi yang lancar. Pertama, diperlukan klaster Kubernetes yang operasional, yang dapat diperoleh melalui berbagai metode seperti layanan cloud terkelola seperti Google Kubernetes Engine (GKE), Amazon Elastic Kubernetes Service (EKS), atau Azure Kubernetes Service (AKS), atau dengan menyiapkan klaster secara manual menggunakan perangkat seperti kubeadm atau Minikube untuk pengembangan lokal. Klaster Kubernetes harus memiliki sumber daya yang memadai, termasuk CPU, memori, dan penyimpanan, untuk mendukung komponen Hadoop dan beban kerja pemrosesan data2. Selain itu, perlu menginstal dan mengonfigurasi kubectl, alat baris perintah Kubernetes, untuk berinteraksi dengan klaster. Untuk menyiapkan Minikube, persyaratan minimumnya meliputi setidaknya 2 GB RAM, 2 core CPU, dan 20 GB ruang disk kosong. Koneksi internet diperlukan untuk menjalankan Minikube pada awalnya, dan diperlukan lapisan kontainerisasi seperti Docker. Spesifikasi ini memastikan bahwa Minikube dapat menjalankan klaster Kubernetes lokal secara efektif untuk tujuan pengembangan dan pengujian.

Selanjutnya, lingkungan harus dipersiapkan dengan mengonfigurasi pengaturan dan dependensi yang diperlukan. Ini termasuk membuat volume penyimpanan persisten untuk HDFS Hadoop guna menyimpan data secara persisten, mengonfigurasi jaringan untuk memfasilitasi komunikasi antara komponen Hadoop, dan menerapkan langkah-langkah keamanan yang sesuai seperti Role-Based Access Control (RBAC) dan kebijakan jaringan. Lebih jauh, mengontainerisasi komponen Hadoop (NameNode, DataNode,

ResourceManager, dll.) menggunakan gambar Docker sangatlah penting. Image Hadoop Docker bawaan yang tersedia dari sumber terpercaya melalui Docker Hub dapat digunakan, atau image kustom yang disesuaikan dengan kebutuhan spesifik dapat dibuat. Terakhir, komponen Hadoop yang dikontainerisasi ini harus dikonfigurasi dan disebarkan menggunakan manifes Kubernetes (file YAML) atau bagan Helm untuk mengotomatiskan dan mengelola proses penyebaran secara efektif.

Menginstal dan mengonfigurasi Hadoop di Kubernetes melibatkan beberapa langkah penting untuk memastikan penyebaran dan operasi yang lancar. Pertama, versi komponen Hadoop yang dikontainerisasi, seperti NameNode, DataNode, ResourceManager, dan NodeManager, harus diperoleh atau dibuat menggunakan gambar Docker. Gambar-gambar ini dapat bersumber dari repositori official atau dibuat khusus untuk memenuhi persyaratan tertentu. Selanjutnya, manifes Kubernetes (file YAML) atau bagan Helm digunakan untuk menentukan konfigurasi penyebaran untuk komponen-komponen ini. Konfigurasi ini mencakup menentukan permintaan dan batasan sumber daya, menyiapkan volume penyimpanan persisten untuk HDFS, dan menentukan pengaturan jaringan untuk memungkinkan komunikasi antar komponen. Proses penyebaran dimulai menggunakan perintah kubectl apply atau Helm, yang mengatur pembuatan Deployment, Service, RBAC dan ConfigMap yang diperlukan. Setelah disebarkan, kinerja dan kesehatan klaster Hadoop dipantau menggunakan perangkat bawaan Kubernetes dan solusi pemantauan tambahan seperti Prometheus dan Grafana. Penyetelan dan penyesuaian yang tepat dilakukan untuk mengoptimalkan pemanfaatan sumber daya dan memastikan pengoperasian klaster Hadoop yang efisien dalam lingkungan Kubernetes.

Alokasi dan penjadwalan sumber daya merupakan aspek penting dalam menjalankan Hadoop di Kubernetes, yang memastikan kinerja optimal dan penggunaan sumber daya klaster yang efisien. Kubernetes mengelola alokasi sumber daya melalui penjadwalnya, yang menetapkan Pod ke node berdasarkan sumber daya yang tersedia dan batasan yang ditentukan. Dalam konteks Hadoop, hal ini melibatkan pengalokasian sumber daya CPU dan memori ke berbagai komponen Hadoop, seperti NameNode, DataNode, ResourceManager, dan NodeManager. Kubernetes menggunakan permintaan dan batasan sumber daya yang ditetapkan dalam konfigurasi penerapan untuk memastikan bahwa setiap komponen menerima sumber daya yang diperlukan sekaligus mencegah komitmen berlebih dan penolakan penjadwalan. Selain itu, Kubernetes mendukung fitur penjadwalan tingkat lanjut seperti aturan afinitas dan anti-afinitas, yang dapat digunakan untuk mengontrol penempatan Pod Hadoop guna mencapai lokalitas data dan toleransi kesalahan. Alokasi dan penjadwalan sumber daya yang tepat sangat penting untuk menjaga kinerja, stabilitas, dan efisiensi klaster Hadoop yang berjalan di Kubernetes, yang memungkinkannya menangani beban kerja big data yang beragam dan menuntut secara efektif.

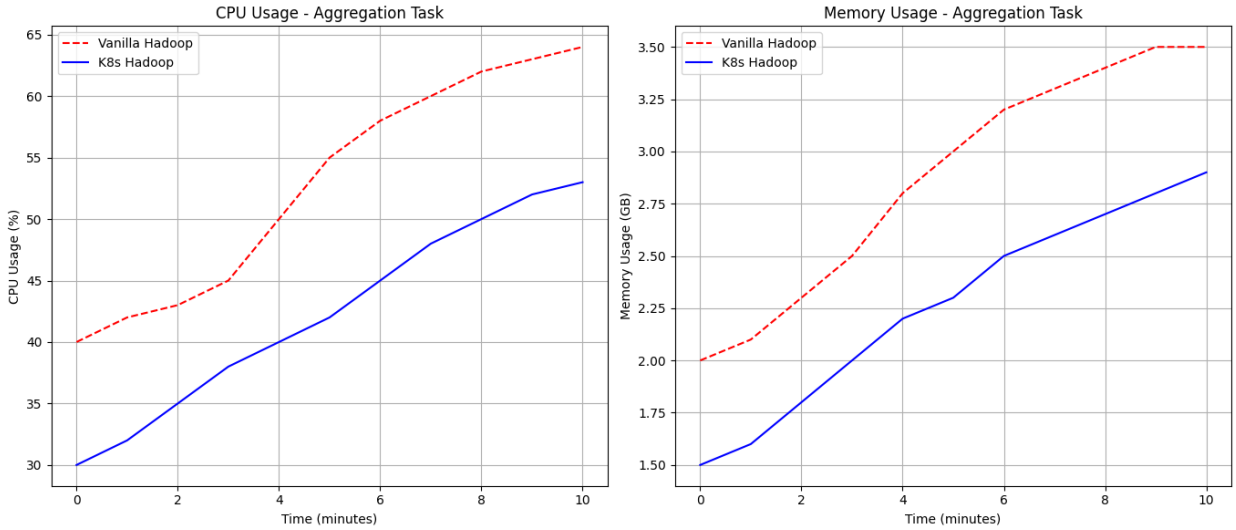
Lokalitas data dan pertimbangan jaringan sangat penting saat menjalankan Hadoop di Kubernetes untuk memastikan pemrosesan data yang efisien dan meminimalkan latensi. Lokalitas data mengacu pada konsep pemrosesan data sedekat mungkin dengan lokasi penyimpanannya, sehingga mengurangi kebutuhan untuk mentransfer kumpulan data besar melalui jaringan. Dalam pengaturan Hadoop tradisional, lokalitas data dicapai dengan menjadwalkan tugas pada node tempat data berada. Namun, dalam lingkungan Kubernetes, mencapai lokalitas data bisa lebih menantang karena sifat orkestrasi kontainer yang dinamis. Penjadwal Kubernetes dapat dikonfigurasi untuk mempertimbangkan lokalitas data dengan menggunakan aturan afinitas node dan anti-afinitas, yang memastikan bahwa Pod ditempatkan pada node tempat data yang diperlukan disimpan atau di dekatnya.

Pertimbangan jaringan juga memainkan peran penting dalam kinerja kluster Hadoop di Kubernetes. Model jaringan Kubernetes mengabstraksi infrastruktur jaringan yang mendasarinya, menyediakan antarmuka jaringan yang konsisten untuk komunikasi antar Pod. Memastikan koneksi jaringan latensi rendah dan bandwidth tinggi antara komponen Hadoop sangat penting untuk transfer dan pemrosesan data yang efisien. Kebijakan jaringan dapat digunakan untuk mengelola dan mengamankan komunikasi antar Pod, sementara jaringan overlay dan service mesh dapat meningkatkan kinerja dan keandalan jaringan. Dengan mempertimbangkan lokasi data dan konfigurasi jaringan secara cermat, organisasi dapat

mengoptimalkan kinerja Hadoop di Kubernetes, memastikan bahwa beban kerja pemrosesan data ditangani secara efisien dan efektif.

3. Hasil dan Pembahasan

Perbandingan kinerja antara Vanilla Hadoop (istilah untuk instalasi standar) dan Hadoop pada Kubernetes mengungkap perbedaan signifikan dalam penggunaan CPU dan memori di berbagai tugas, khususnya dalam konteks beban kerja agregasi. Analisis deret waktu memberikan wawasan lebih mendalam tentang peningkatan efisiensi yang dihasilkan oleh orkestrasi kontainer saat menjalankan pekerjaan pemrosesan data berskala besar. Perbandingan penggunaan CPU dan memori dapat dilihat pada Gambar 3.



Gambar 1. Perbandingan CPU dan Memori

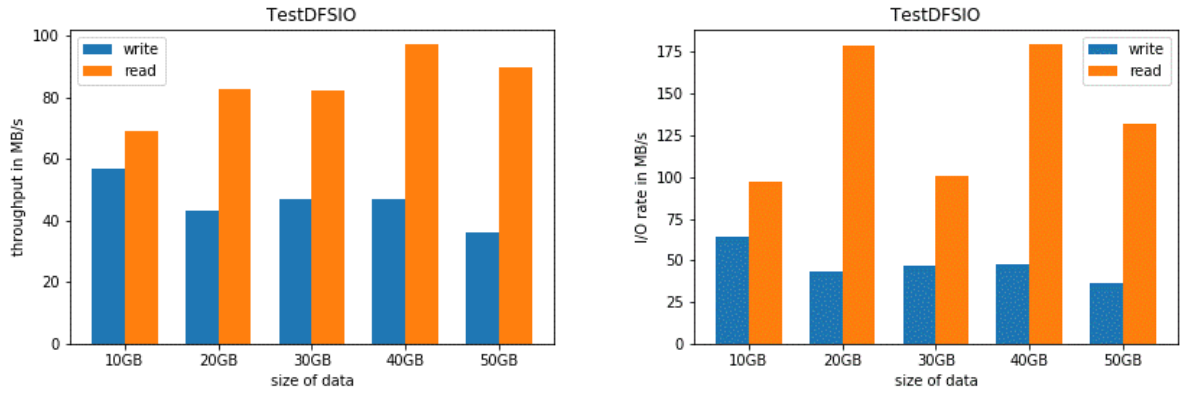
Pola penggunaan CPU yang diamati dalam studi ini menunjukkan keuntungan yang jelas bagi Hadoop di Kubernetes dibandingkan dengan penerapan Vanilla Hadoop tradisional. Di semua tugas, Hadoop berbasis Kubernetes secara konsisten menunjukkan konsumsi CPU yang lebih rendah, dengan tren alokasi sumber daya yang lebih lancar dari waktu ke waktu. Misalnya, dalam tugas Agregasi, sementara Vanilla Hadoop menunjukkan peningkatan yang stabil dalam penggunaan CPU, yang berpuncak pada sekitar 64%. Sementara itu, Hadoop di Kubernetes mencapai titik jenuh pada 53%. Jejak CPU yang berkurang ini dapat dikaitkan dengan penjadwalan sumber daya dinamis Kubernetes dan mekanisme distribusi beban kerja, yang memungkinkan pemanfaatan node komputasi yang optimal. Tren ini menunjukkan bahwa Kubernetes secara efektif mengurangi perebutan sumber daya melalui penjadwalan pod dinamis, mengurangi siklus idle, dan meningkatkan responsivitas sistem secara keseluruhan. Ketimpangan ini menggarisbawahi keterbatasan penerapan Hadoop monolitik di tempat, yang sering kali gagal memanfaatkan sumber daya perangkat keras secara efektif karena arsitekturnya yang kaku. Di sisi lain, lingkungan Kubernetes yang terkontainerisasi memungkinkan kontrol dan pengoptimalan sumber daya yang sangat teliti.

Pola serupa muncul dalam penggunaan memori di tugas agregasi ini, memperkuat gagasan bahwa Kubernetes menyediakan lingkungan eksekusi yang lebih hemat sumber daya. Dalam tugas Agregasi, konsumsi memori Vanilla Hadoop meningkat secara progresif dari 2,0 GB menjadi 3,5 GB, yang menunjukkan bahwa alokasi memori bersifat statis dan kurang optimal. Sebaliknya, Hadoop pada Kubernetes menunjukkan peningkatan yang lebih bertahap, hanya mencapai 2,9 GB saat tugas selesai. Perbedaan ini menyoroti kemampuan Kubernetes untuk mengalokasikan memori secara dinamis berdasarkan tuntutan beban kerja, sehingga meminimalkan pemborosan dan meningkatkan ketersediaan sumber daya untuk proses bersamaan. Peningkatan ini dapat dikaitkan dengan isolasi tingkat kontainer

Kubernetes, yang mengurangi pembagian memori yang tidak perlu di seluruh tugas dan mengoptimalkan penggunaan sumber daya melalui siklus hidup pod yang terkendali. Hasilnya menunjukkan bahwa strategi manajemen memori Kubernetes, seperti cgroups dan limits, secara efektif meminimalkan inflasi memori yang tidak perlu sekaligus memastikan eksekusi tugas berjalan tanpa gangguan.

Pengurangan yang diamati dalam konsumsi CPU dan memori saat menggunakan Hadoop pada Kubernetes memiliki implikasi yang luas untuk pengoptimalan kinerja dalam lingkungan pemrosesan data terdistribusi. Pertama, hasilnya menggarisbawahi peran penting orkestrasi kontainer dalam mengatasi inefisiensi sumber daya yang terkait dengan penerapan Hadoop monolitik. Kemampuan Kubernetes untuk mengalokasikan sumber daya secara dinamis memastikan bahwa komputasi dan memori digunakan hanya saat diperlukan, sehingga mengurangi biaya operasional dan meningkatkan throughput klaster. Kedua, tren penggunaan sumber daya yang lebih lancar yang diamati di bawah Kubernetes menunjukkan efektivitasnya dalam mengelola beban kerja multi-vendor yang merupakan persyaratan umum dalam lingkungan big data modern. Dengan mencegah perebutan sumber daya dan menyeimbangkan beban tugas di seluruh node, Kubernetes meminimalkan hambatan kinerja yang sering ditemui dalam sistem Hadoop tradisional.

Benchmark TestDFSIO berfungsi sebagai ukuran yang andal untuk kinerja I/O HDFS, yang mengevaluasi throughput baca dan tulis pada berbagai ukuran data. Hasil pengamatan untuk Hadoop pada Kubernetes mengungkap tren penting dalam throughput (grafik kiri) dan rasio I/O (grafik kanan) pada Gambar 4, yang menjelaskan efisiensi dan skalabilitas sistem saat mengalami peningkatan beban kerja.



Gambar 2. *Throughput* dan I/O

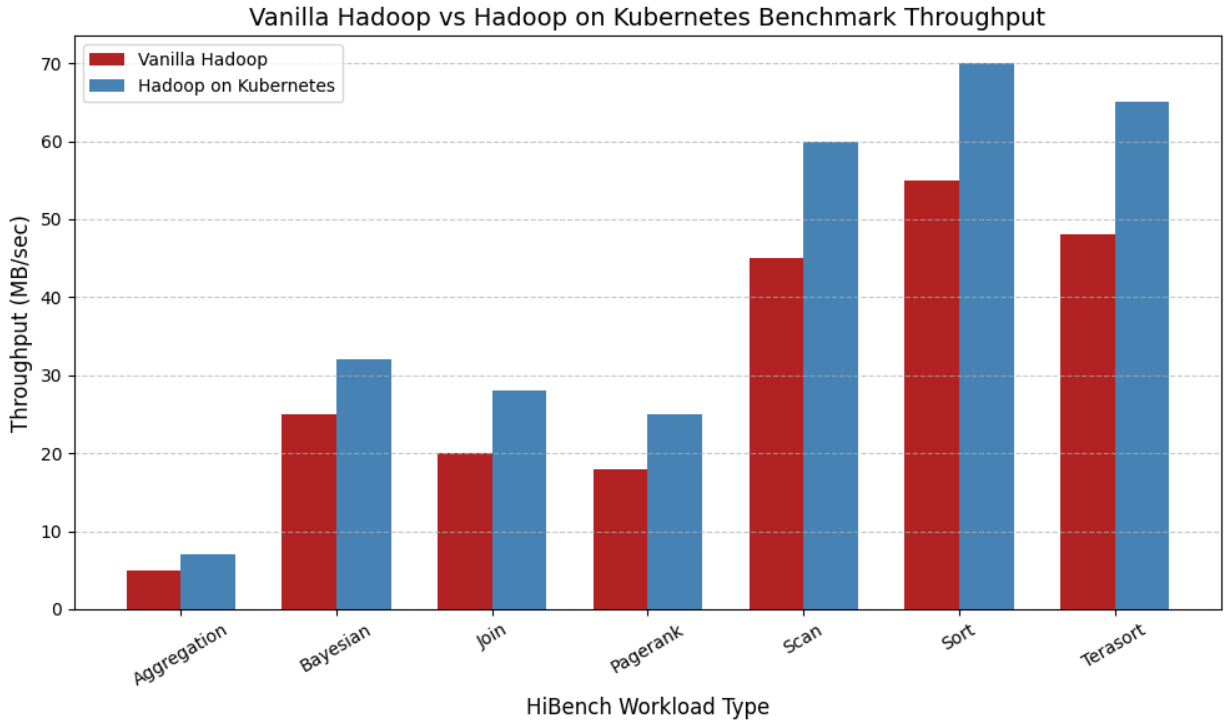
Pada grafik sebelah kiri, throughput penulisan menunjukkan tren yang relatif konsisten di berbagai ukuran data, meskipun pada tingkat yang jauh lebih rendah dibandingkan dengan throughput pembacaan. Misalnya, kinerja penulisan rata-rata antara 40 MB/s dan 50 MB/s saat ukuran data meningkat dari 10 GB menjadi 50 GB. Menariknya, penurunan kecil dalam throughput penulisan diamati pada 20 GB dan 50 GB, yang menunjukkan potensi kemacetan yang dapat timbul dari perebutan sumber daya atau overhead jaringan yang mendasarinya di lingkungan terdistribusi Kubernetes. Perilaku ini menyoroti karakteristik mendasar dari operasi penulisan dalam HDFS: replikasi data dan persistensi tingkat blok menimbulkan latensi. Pada Kubernetes, orkestrasi persistensi data antara node, bersama dengan lapisan abstraksi yang dikontainerisasi, dan dapat memperburuk penundaan penulisan, khususnya di bawah beban yang meningkat. Namun, throughput penulisan yang relatif stabil menunjukkan bahwa Hadoop pada Kubernetes mempertahankan konsistensi yang wajar dalam pemanfaatan sumber daya dan distribusi data, terlepas dari tantangan ini.

Sebaliknya, throughput baca secara signifikan mengungguli operasi tulis di semua ukuran data, dengan kinerja puncak 95 MB/s pada 40 GB dan sedikit penurunan berikutnya menjadi 90 MB/s pada 50 GB. Tren ini sejalan dengan sifat operasi baca HDFS, yang diuntungkan oleh lokalitas data dan mekanisme pengambilan yang efisien. Kinerja baca yang unggul pada Kubernetes dapat dikaitkan dengan beberapa faktor:

a. Skalabilitas Tingkat Pod: Kubernetes secara dinamis menjadwalkan operasi baca di seluruh pod, mengoptimalkan penggunaan sumber daya dan meminimalkan pertentangan.
b. Mekanisme Caching: Permintaan baca HDFS sering kali memanfaatkan caching dalam memori, terutama saat diakses berulang kali, mengurangi penundaan I/O disk.
c. Pemanfaatan Jaringan yang Ditingkatkan: Jaringan overlay Kubernetes meningkatkan distribusi beban kerja baca, sehingga mengurangi latensi. Namun, sedikit penurunan yang diamati pada 50GB menunjukkan bahwa operasi pembacaan mungkin mengalami penurunan hasil karena ukuran data melampaui ambang batas tertentu, mungkin karena meningkatnya overhead jaringan atau inefisiensi penyimpanan dalam kluster.

Grafik di sebelah kanan semakin menguatkan perbedaan antara kinerja baca dan tulis, karena kecepatan I/O untuk operasi baca secara konsisten melampaui operasi tulis di semua ukuran data. Pada 20 GB dan 40 GB, kecepatan baca I/O mencapai puncaknya pada sekitar 175 MB/dtk, yang mencerminkan pengoptimalan Hadoop untuk beban kerja baca berurutan. Di sisi lain, kecepatan tulis I/O tetap relatif sederhana, mencapai titik puncaknya pada sekitar 40 MB/dtk di seluruh kumpulan data. Kesenjangan yang mencolok antara kecepatan baca dan tulis I/O ini menggarisbawahi tantangan inheren dalam penerapan Hadoop yang dikontainerisasi: beban kerja yang banyak menulis cenderung memperbesar overhead yang diperkenalkan oleh runtime kontainer dan mekanisme penyimpanan persisten Kubernetes. Tidak seperti penerapan Hadoop bare-metal tradisional, di mana I/O disk dikelola secara langsung, Kubernetes menambahkan lapisan abstraksi yang berpotensi membatasi throughput penulisan.

Gambar 5 membandingkan throughput (dalam MB/detik) Vanilla Hadoop dan Hadoop pada Kubernetes di berbagai jenis beban kerja via HiBench—Agregasi, Bayesian, Join, Pagerank, Scan, Sort, dan TeraSort. Metrik kinerja memberikan wawasan tentang efisiensi Hadoop saat dijalankan pada kluster bare-metal tradisional (Vanilla Hadoop) versus lingkungan Kubernetes yang terkontainerisasi.



Gambar 3. *Throughput* Berbagai Beban

Di semua jenis beban kerja, Hadoop pada Kubernetes secara konsisten mencapai throughput yang lebih tinggi dibandingkan dengan Vanilla Hadoop. Hal ini menyoroti keuntungan menjalankan Hadoop di lingkungan Kubernetes, termasuk:

a. Efisiensi Manajemen Sumber Daya: Kubernetes menyediakan penjadwalan sumber daya dinamis, yang mengoptimalkan pemanfaatan komputasi dan I/O.
b. Skalabilitas: Orkestrasi kontainer Kubernetes memungkinkan penskalaan horizontal yang lebih baik, yang menghasilkan peningkatan kinerja dalam beban kerja komputasi yang berat.
c. Isolasi Sumber Daya: Tugas yang dikontainerisasi mendapat manfaat dari isolasi, mengurangi gangguan antar pekerjaan.

Beban Kerja Ringan (Agregasi, Bayesian), yang melibatkan I/O minimal dan overhead komputasi, mendapat manfaat utama dari penjadwalan sumber daya Kubernetes. Peningkatan throughput, meskipun moderat, konsisten, yang menunjukkan bahwa manajemen kontainer ringan Kubernetes dapat menangani pekerjaan yang lebih kecil secara efisien. Join dan Pagerank melibatkan pengacakan signifikan dan pertukaran data menengah, yang secara tradisional membebani kinerja Hadoop. Sementara Kubernetes memberikan beberapa peningkatan melalui pemanfaatan sumber daya yang lebih baik, perolehan kinerja kurang terlihat karena sifat kompleks dari operasi pengacakan data. Scan, Sort, dan TeraSort melibatkan I/O berurutan dan pengurutan skala besar, yang mendapat manfaat substansial dari skalabilitas dan isolasi sumber daya Kubernetes. Beban kerja Sort dan TeraSort, khususnya, menunjukkan perolehan throughput terbesar, yang menunjukkan bahwa kemampuan Kubernetes untuk mengatur tugas-tugas I/O yang berat di beberapa node secara signifikan meningkatkan kinerja.

4. Kesimpulan

Menjalankan Hadoop di Kubernetes menyatukan kekuatan kedua teknologi untuk menciptakan lingkungan pemrosesan big data yang tangguh dan fleksibel. Komponen inti Hadoop—HDFS untuk penyimpanan, MapReduce untuk pemrosesan data, dan YARN untuk manajemen sumber daya—memungkinkan penanganan data terdistribusi yang efisien. Mengintegrasikan komponen-komponen ini dengan Kubernetes memanfaatkan kemampuan orkestrasinya yang tangguh, alokasi sumber daya yang dinamis, dan fitur-fitur otomatisasi, yang mengarah pada peningkatan pemanfaatan sumber daya, manajemen yang lebih mudah, dan skalabilitas yang lebih baik. Kombinasi ini memungkinkan organisasi untuk menerapkan kluster Hadoop secara dinamis, menskalakannya sesuai kebutuhan, dan berintegrasi dengan lancar dengan perangkat dan layanan cloud-native lainnya.

Manajemen sumber daya, pemantauan, dan pencatatan yang efektif sangat penting untuk menjaga kinerja dan kesehatan kluster Hadoop di Kubernetes. Menerapkan permintaan dan batasan sumber daya Kubernetes memastikan bahwa setiap komponen Hadoop menerima sumber daya yang diperlukan tanpa pertentangan. Solusi pemantauan komprehensif seperti Prometheus dan Grafana memberikan wawasan waktu nyata tentang kinerja kluster, sementara solusi pencatatan terpusat seperti tumpukan ELK memungkinkan manajemen dan pemecahan masalah log yang efisien. Dengan mengadopsi praktik ini, organisasi dapat mengoptimalkan penggunaan sumber daya, mengidentifikasi dan mengatasi hambatan kinerja, dan memastikan kelancaran pengoperasian infrastruktur big data mereka.

Pertimbangan keamanan sangat penting saat menerapkan Hadoop di Kubernetes. Kontrol Akses Berbasis Peran (RBAC) membatasi akses ke sumber daya berdasarkan peran pengguna, sementara kebijakan jaringan mengontrol komunikasi antara Pod, sehingga meningkatkan keamanan. Mengelola informasi sensitif dengan rahasia Kubernetes dan memperbarui komponen Hadoop dan Kubernetes secara berkala membantu mengurangi kerentanan. Dengan mengikuti praktik terbaik keamanan, seperti menegakkan prinsip hak istimewa paling rendah dan melakukan audit keamanan rutin, organisasi dapat melindungi data mereka dan memastikan lingkungan pemrosesan big data yang aman. Bersama-sama, strategi ini menciptakan pengaturan Hadoop di Kubernetes yang tangguh dan efisien, yang mampu menangani beragam beban kerja big data secara efektif.

Pekerjaan di masa mendatang dalam integrasi Hadoop pada Kubernetes kemungkinan akan difokuskan pada peningkatan otomatisasi dan penyederhanaan proses penyebaran. Mengembangkan operator Kubernetes yang lebih canggih untuk Hadoop dapat mengotomatiskan tugas-tugas rumit seperti penskalaan, failover, dan manajemen konfigurasi, sehingga mengurangi kebutuhan akan intervensi dan keahlian manual. Selain itu, mengintegrasikan lebih banyak teknologi dan perangkat berbasis cloud dalam ekosistem Hadoop akan memfasilitasi penyebaran dan manajemen yang lancar di seluruh lingkungan hybrid dan multi-cloud. Ini termasuk meningkatkan interoperabilitas dengan kerangka kerja pemrosesan dan analisis big data lainnya, seperti Apache Spark dan Kafka, untuk menciptakan jalur data yang lebih kohesif dan kuat. Kemajuan ini akan memungkinkan organisasi untuk memanfaatkan Hadoop pada Kubernetes secara lebih efektif, sehingga mendorong efisiensi dan inovasi dalam pemrosesan big data.

Bidang penting lain yang menjadisssss pekerjaan mendatang adalah meningkatkan keamanan dan kepatuhan untuk penerapan Hadoop di Kubernetes. Karena peraturan privasi data menjadi lebih ketat, penting untuk mengembangkan fitur keamanan yang kuat seperti metode enkripsi yang disempurnakan, kontrol akses, dan pemeriksaan kepatuhan otomatis. Pekerjaan mendatang juga harus mengeksplorasi adopsi arsitektur tanpa server dan layanan mikro dalam ekosistem Hadoop untuk meningkatkan ketangkasan, skalabilitas, dan efisiensi sumber daya. Dengan mengatasi tantangan ini, pengembangan mendatang akan berkontribusi untuk menciptakan solusi pemrosesan big data yang lebih serbaguna, aman, dan efisien. Yang pada akhirnya akan memberdayakan organisasi untuk memperoleh wawasan dan nilai yang lebih dalam dari data mereka sambil tetap mematuhi standar peraturan.

Referensi

[1] H. T. Almansouri and Y. Masmoudi, “Hadoop Distributed File System for Big data analysis,” Proceedings of 2019 IEEE World Conference on Complex Systems, WCCS 2019, pp. 1–16, 2019, doi: 10.1109/ICoCS.2019.8930804.

[2] A. A. Bhuyar, “A Review Paper on Big Data and Hadoop is solution for Big Data,” Interantional Journal of Scientific Research in Engineering and Management, vol. 06, no. 03, 2022, doi: 10.55041/ijsrem11911.

[3] A. A. Hussein, “Using Hadoop Technology to Overcome Big Data Problems by Choosing Proposed Cost-efficient Scheduler Algorithm for Heterogeneous Hadoop System (BD3),” J Sci Res Rep, vol. 26, no. 9, pp. 58–84, 2020, doi: 10.9734/jsrr/2020/v26i930310.

[4] M. K. Abhishek, D. R. Rao, and K. Subrahmanyam, “Framework to Deploy Containers using Kubernetes and CI/CD Pipeline,” International Journal of Advanced Computer Science and Applications, vol. 13, no. 4, pp. 522–526, 2022, doi: 10.14569/IJACSA.2022.0130460.

[5] K. Senjab, S. Abbas, N. Ahmed, and A. ur R. Khan, “A survey of Kubernetes scheduling algorithms,” Journal of Cloud Computing, vol. 12, no. 1, pp. 1–26, 2023, doi: 10.1186/s13677-023-00471-1.

[6] A. Jeffery, H. Howard, and R. Mortier, “Rearchitecting Kubernetes for the Edge,” EdgeSys 2021 - Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking, Part of EuroSys 2021, pp. 7–12, 2021, doi: 10.1145/3434770.3459730.

[7] L. A. Vayghan, M. A. Saied, M. Toeroe, and F. Khendek, “A Kubernetes controller for managing the availability of elastic microservice based stateful applications,” Journal of Systems and Software, vol. 175, 2021, doi: 10.1016/j.jss.2021.110924.

[8] Y. Benlachmi, A. El Yazidi, and M. L. Hasnaoui, “A Comparative Analysis of Hadoop and Spark Frameworks using Word Count Algorithm,” International Journal of Advanced Computer Science and Applications, vol. 12, no. 4, pp. 778–788, 2021, doi: 10.14569/IJACSA.2021.0120495.

[9] F. Ullah, S. Dhingra, X. Xia, and M. A. Babar, "Evaluation of distributed data processing frameworks in hybrid clouds," Journal of Network and Computer Applications, vol. 224, pp. 1–10, 2024, doi: 10.1016/j.jnca.2024.103837.

[10] Y. Jiang, Q. Liu, W. Dannah, D. Jin, X. Liu, and M. Sun, "An Optimized Resource Scheduling Strategy for Hadoop Speculative Execution Based on Non-cooperative Game Schemes," Computers, Materials and Continua, vol. 62, no. 2, pp. 713–729, 2020, doi: 10.32604/cmc.2020.04604.

[11] M. Adam Ibrahim Fakherldin, K. Adam, N. Akma Abu Bakar, and M. Abdul Majid, "Weather Data Analysis Using Hadoop: Applications and Challenges," in IOP Conference Series: Materials Science and Engineering, 2019. doi: 10.1088/1757-899X/551/1/012044.

[12] K. H. Yoo, C. K. Leung, and A. Nasridinov, "Big Data Analysis and Visualization: Challenges and Solutions," 2022. doi: 10.3390/app12168248.

[13] M. Bacou, A. Tchana, and D. Hagimont, "Your containers should be WYSIWYG," Proceedings - 2019 IEEE International Conference on Services Computing, SCC 2019 - Part of the 2019 IEEE World Congress on Services, pp. 56–64, 2019, doi: 10.1109/SCC.2019.00022.

[14] J. Thijsman, M. Sebrechts, F. De Turck, and B. Volckaert, "Trusting the Cloud-Native Edge: Remotely Attested Kubernetes Workers," Proceedings - International Conference on Computer Communications and Networks, ICCCN, 2024, doi: 10.1109/ICCCN61486.2024.10637515.

[15] T. Patanshetti, A. A. Pawar, D. Patel, and S. Thakare, "Auto tuning of hadoop and spark parameters," International Journal of Engineering Trends and Technology, vol. 69, no. 11, pp. 22–33, 2021, doi: 10.14445/22315381/IJETT-V69I11P204.

[16] L. Yin, K. Chen, Z. Jiang, and X. Xu, "A Fast Parallel Random Forest Algorithm Based on Spark," Applied Sciences (Switzerland), vol. 13, no. 10, 2023, doi: 10.3390/app13106121.

[17] A. Ed-daoudy, K. Maalmi, and A. El Ouaazizi, "A scalable and real-time system for disease prediction using big data processing," Multimed Tools Appl, vol. 82, no. 20, pp. 30405–30434, 2023, doi: 10.1007/s11042-023-14562-3.