EVALUASI KOMPARATIF PENGHEMATAN RUANG PADA *BTRFS* ANTARA PENYIMPANAN *VM WHITE-BOX* DAN *BLACK-BOX*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Zhafran Rama Azmi
NIM: 215150207111025

PROGRAM STUDI TEKNIK INFORMATIKA
DEPARTEMEN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2025

PENGESAHAN

EVALUASI KOMPARATIF PENGHEMATAN RUANG PADA *BTRFS* ANTARA PENYIMPANAN *VM WHITE-BOX* DAN *BLACK-BOX*

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Zhafran Rama Azmi
NIM: 215150207111025

Skripsi ini telah diuji dan dinyatakan lulus pada
2 Februari 2026
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing 1	Dosen Pembimbing 2
Ir. Primantara Hari Trisnawan, M.Sc.	Reza Andria Siregar, S.T., M.Kom.
NIP. 196809121994031002	NIP. 197906212006041003

Mengetahui
Ketua Departemen Teknik Informatika

Rekyan Regasari Mardi Putri, S.T., M.T., Ph.D.
NIP. 2011027704142001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar referensi.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 2 Feburari 2026

Zhafran Rama Azmi

NIM: 215150207111025

PRAKATA

Segala puji dan syukur penulis panjatkan kepada Allah SWT atas rahmat dan karunia-Nya dalam penyelesaian skripsi ini dengan baik. Skripsi ini dikerjakan selaku persyaratan untuk memperoleh gelar Sarjana Komputer pada Program Studi Teknik Informatika, Departemen Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya. Selama pengerjaan skripsi ini, penulis didukung dan dibimbing oleh banyak pihak secara langsung maupun tidak langsung. Dengan penuh rasa hormat, penulis menyampaikan terima kasih yang mendalam kepada pihak-pihak yang disebut sebelumnya, tidak antara lain:

1. Ir. Primantara Hari Trisnawan, M.Sc., selaku dosen pembimbing I, memberikan panduan, masukan dan arahan yang sangat membantu dalam penyusunan skripsi ini.
2. Reza Andria Siregar, S.T., M.Kom., selaku dosen pembimbing II, memberikan panduan dan masukan yang tajam sehingga skripsi dapat diselesaikan dengan lebih baik.
3. Bayu Priyambadha, S.Kom., M.Kom., Ph.D. selaku ketua Program Studi Teknik Informatika, telah membantu dalam pengurusan transkrip nilai.
4. Rekyan Regasari Mardi Putri, S.T., M.T., Ph.D., selaku ketua Departemen Teknik Informatika, atas bantuannya dalam pengesahan skripsi ini.
5. Kedua orang tua penulis, yang selalu memberikan dukungan jasmani, rohani, moral dan motivasi yang selalu mengalir.
6. Teman-teman penulis, yang memberikan semangat, saran dan diskusi selama pengerjaan skripsi ini.
7. Pengembang-pengembang perangkat-perangkat lunak yang dipakai di penelitian ini.

Penyelesaian skripsi ini dapat tercapai tidak lepas dari bantuan dan dukungan dari pihak-pihak yang telah disebutkan. Dengan penuh rasa syukur, penulis mengucapkan terima kasih yang sebesar-besarnya atas arahan, perhatian dan masukan yang telah diberikan.

Malang, 2 Februari 2026

Zhafran Rama Azmi

zhafran.rama@student.ub.ac.id

ABSTRAK

Zhafran Rama Azmi, Evaluasi Komparatif Penghematan Ruang pada *Btrfs* Antara Penyimpanan *VM White-Box* dan *Black-Box*

Pembimbing: Ir. Primantara Hari Trisnawan, M.Sc. dan Reza Andria Siregar, S.T., M.Kom.

Penelitian ini mengevaluasi efektivitas penghematan ruang penyimpanan VM (VM) pada sistem berkas Btrfs dengan membandingkan dua model penyimpanan: *white-box* (melalui Virtio-FS) dan *black-box* (citra diska tradisional). Eksperimen dilakukan dengan membuat 10 VM yang identik, menumbuhkan datanya melalui simulasi beban kerja, kemudian mengukur penghematan ruang yang dicapai melalui fitur Copy-on-Write (CoW), deduplikasi (duperemove), dan kompresi (Zstd level 3). Hasil menunjukkan bahwa model *white-box* secara signifikan lebih unggul dalam efisiensi ruang dan kecepatan deduplikasi. Pada skala 10 VM tanpa kompresi, *white-box* mencapai penghematan 74,10% (dari baseline 27,69 GiB menjadi 7,17 GiB), sementara *black-box* hanya 29,66% (dari 30,41 GiB menjadi 21,39 GiB). Dengan kompresi, *white-box* mencapai penghematan maksimal 82,28%, dan *black-box* 73,78%. Waktu deduplikasi *white-box* (≈12–15 menit) 6–12 kali lebih cepat daripada *black-box* (≈29 menit hingga 2,5 jam). Analisis mikro terhadap basis data hashfile mengungkap bahwa keunggulan *white-box* berasal dari kemampuannya mengidentifikasi duplikasi secara akurat pada level berkas dengan rasio extent kembar >80%, sementara *black-box* bergulat dengan fragmentasi internal yang mengurangi efektivitas deduplikasi.

Kata kunci: virtualisasi, btrfs, deduplikasi, *white-box*, *black-box*, virtio-fs, sistem berkas, penyimpanan, citra diska, penghematan penyimpanan

ABSTRACT

Zhafran Rama Azmi, *Comparative Evaluation of Space Savings of VM Storage between White-box and Black-box Storage on Btrfs*

***Supervisors:* Ir. Primantara Hari Trisnawan, M.Sc. *and* Reza Andria Siregar, S.T., M.Kom.**

This study evaluates the effectiveness of virtual machine (VM) storage space savings on the Btrfs filesystem by comparing two storage models: white-box (via Virtio-FS) and black-box (traditional disk images). Experiments were conducted by creating 10 identical VMs, growing their data through workload simulation, then measuring space savings achieved through Copy-on-Write (CoW), deduplication (duperemove), and compression (Zstd level 3). Results show that the white-box model is significantly superior in space efficiency and deduplication speed. At a scale of 10 VMs without compression, white-box achieved 74.10% space savings (from a 27.69 GiB baseline to 7.17 GiB), while black-box only reached 29.66% (from 30.41 GiB to 21.39 GiB). With compression, white-box achieved maximum savings of 82.28%, and black-box 73.78%. White-box deduplication time (≈12–15 minutes) was 6–12 times faster than black-box (≈29 minutes to 2.5 hours). Micro-analysis of hashfile databases revealed that white-box superiority stems from its ability to accurately identify duplication at file level with duplicate extent ratios >80%, while black-box struggles with internal fragmentation that reduces deduplication effectiveness.

Keywords: virtualization, btrfs, deduplication, white-box, black-box, virtio-fs, filesystem, storage, disk image, storage savings

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR KODE

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Virtualisasi telah menjadi tulang punggung infrastruktur *cloud* modern, memungkinkan konsolidasi beban kerja dengan menjalankan banyak mesin virtual (VM) di atas perangkat keras fisik yang sama (Borra, 2024b; 2024a). Namun, adopsi virtualisasi yang masif membawa tantangan besar dalam efisiensi penyimpanan. Penelitian terdahulu menunjukkan bahwa kumpulan citra (*image*) VM memiliki tingkat redundansi data yang sangat tinggi, seringkali mencapai 80% kemiripan antar citra karena mayoritas VM menjalankan sistem operasi dan pustaka perangkat lunak yang identik (Clements dkk., 2009; Jin dan Miller, 2009; Jayaram dkk., 2011).

Redundansi ini berpotensi untuk dikurangi melalui teknik deduplikasi data. Deduplikasi data untuk citra VM dapat dipisah menjadi dua berdasarkan tingkat pengetahuan sistem terhadap struktur data. Yang pertama adalah deduplikasi *black-box* yang memperlakukan citra VM sebagai objek biner yang tidak transparan. Deduplikasi dilakukan pada tingkat chunk atau blok data tanpa memahami isi atau struktur internal berkas. Meski dapat mengurangi redundansi, metode ini kurang optimal karena mengabaikan semantik data, seperti struktur sistem berkas di dalam citra, sehingga berpotensi melewatkan peluang deduplikasi tambahan (Jayaram dkk., 2011). Kedua adalah deduplikasi *white-box* yang memanfaatkan pengetahuan semantik tentang struktur internal berkas (misalnya, memahami struktur sistem berkas di dalam VHD). Dengan pengetahuan ini, sistem dapat mengidentifikasi dan menghilangkan redundansi dengan lebih akurat dan efisien, meskipun memerlukan proses analisis awal (Meyer dan Bolosky, 2012; Thwel dan Sinha, 2021). Untuk berkas besar dengan pola akses kompleks seperti citra VM, pengetahuan semantik atau antarmuka sistem berkas yang secara eksplisit mendukungnya diperlukan untuk mencapai optimasi yang optimal.

Tantangan utama menerapkan deduplikasi *white-box* adalah bahwa citra VM tradisional (berformat biner) menyembunyikan struktur dan semantik data di dalamnya dari sistem berkas host. Keterbatasan ini menuntut adanya metode yang mampu memberikan informasi dekomposisi dari isi citra VM agar dapat dikelola secara granular, seperti yang ditawarkan oleh sistem Expelliarmus yang memodelkan VM sebagai grafik semantik untuk memisahkan citra dasar dan paket perangkat lunak (Saurabh dkk., 2019). Menjawab kebutuhan transparansi data tersebut, penelitian ini memanfaatkan Virtio-FS. Virtio-FS memainkan peran penting karena memungkinkan sebuah direktori (folder) di *host* diekspos secara langsung ke VM sebagai sistem berkas yang dapat di-*boot*. Dengan demikian, isi VM—berupa berkas dan direktori sistem operasi—tersaji dalam bentuk aslinya dan dapat sepenuhnya dipahami serta diproses oleh sistem berkas *host* tanpa perlunya dekomposisi citra VM yang rumit layaknya Expelliarmus.

Kemampuan ini membuka peluang untuk memanfaatkan sistem berkas host yang memiliki fitur deduplikasi bawaan, seperti Btrfs. Btrfs dipilih karena beberapa pertimbangan: (1) merupakan teknologi upstream dalam kernel Linux sehingga memiliki integrasi dan dukungan yang baik, berbeda dengan ZFS yang out-of-tree; (2) lebih sederhana dalam konfigurasi dan operasional dibandingkan solusi berbasis clustering atau lapisan blok seperti VDO; (4) ; serta (3) telah terbukti digunakan dalam skala produksi di perusahaan seperti Meta, yang melaporkan penghematan biaya penyimpanan yang signifikan (berorde miliar dolar), meskipun lebih banyak diterapkan untuk kontainer daripada VM (Open Source Summit, 2023).

Dengan menggabungkan Virtio-FS yang mengubah paradigma penyimpanan VM menjadi direktori yang semantic-aware dan Btrfs yang menyediakan deduplikasi transparan di sisi host, maka dapat diterapkan pendekatan deduplikasi *white-box* terhadap kumpulan citra VM. Namun, belum jelas sejauh mana keunggulan pendekatan *white-box* ini dibandingkan dengan deduplikasi *black-box* tradisional pada Btrfs, baik dari segi peningkatan rasio deduplikasi maupun trade-off kinerja yang mungkin terjadi. Oleh karena itu, penelitian ini bertujuan untuk melakukan analisis komparatif antara kedua pendekatan tersebut guna mengukur secara empiris perbandingan tingkat deduplikasi dan waktu eksekusi yang diperlukan, sehingga dapat memberikan evaluasi yang konkret mengenai efektivitas dan efisiensi masing-masing metode dalam mengoptimalkan penyimpanan citra VM.

1.2 Rumusan Masalah

Berdasarkan latar belakang tersebut, masalah yang akan dikaji adalah:

1. Seberapa besar penghematan ruang penyimpanan nyata yang dapat dicapai dengan menerapkan teknik deteksi kemiripan *black-box* pada citra VM menggunakan fitur deduplikasi dan kompresi Btrfs?
2. Apakah penerapan teknik *white-box* (melalui mekanisme *filesystem passthrough*) mampu menghasilkan efisiensi ruang yang lebih tinggi dibandingkan teknik *black-box*?
3. Bagaimana perbandingan biaya kinerja (waktu proses deduplikasi) antara penerapan teknik *black-box* dan *white-box*?

1.3 Tujuan

Penelitian ini bertuju untuk:

1. Mengukur efektivitas praktis dari teknik *black-box* dalam mengurangi ukuran penyimpanan VM pada sistem berkas modern.
2. Menganalisis peningkatan efisiensi yang ditawarkan oleh teknik *white-box* melalui implementasi Virtiofs.
3. Mengevaluasi perbandingan beban kerja sistem (waktu eksekusi) yang dibutuhkan untuk menjalankan proses penghematan ruang pada kedua teknik tersebut.

1.4 Manfaat

Penelitian ini diharapkan dapat bermanfaat sebagai berikut:

1. **Manfaat Teoritis**: Memperbarui studi literatur tentang teknik *white-box* dan *black-box* (Jayaram, 2011) dengan data eksperimen nyata, membuktikan bahwa teknik *white-box* kini layak dan mudah diterapkan.
2. **Manfaat Praktis**: Memberikan data acuan bagi pengelola sistem TI bahwa beralih ke metode penyimpanan berbasis *white-box* bisa menghemat ruang penyimpanan dan waktu pemrosesan secara drastis.

1.5 Batasan Masalah

Penelitian ini dibatasi oleh asumsi-asumsi berikut:

1. Virtualisasi menggunakan ***Kernel-based Virtual Machine*** **(KVM)** dengan **QEMU** dan **Libvirt**.
2. Sistem berkas tempat penyimpanan adalah **Btrfs** yang mengaktifkan fitur kompresi (Zstd) dan deduplikasi (menggunakan alat `duperemove`).
3. **Teknik *Black-box*** diuji menggunakan citra diska format **raw** (citra d*iska jarang*), di mana *host* hanya melihat blok data mentah.
4. **Teknik *White-box*** diuji menggunakan **Virtiofs**, di mana *host* memiliki akses langsung ke struktur berkas *guest*.
5. Sistem operasi tamu (*guest*) yang digunakan adalah Linux (Fedora).
6. Parameter utama yang dianalisis adalah **ruang penyimpanan (GiB)** dan **waktu eksekusi deduplikasi**.

1.6 Sistematika Pembahasan

Sistematika pembahasan dalam skripsi ini adalah sebagai berikut:

BAB 1 PENDAHULUAN Menjelaskan alasan pemilihan topik (pentingnya efisiensi penyimpanan), masalah yang diangkat (uji praktis *black-box* vs *white-box*), tujuan, manfaat, batasan, serta alur pembahasan.

BAB 2 LANDASAN KEPUSTAKAAN Membahas teori dasar, termasuk cara kerja virtualisasi, penjelasan mendalam tentang teknik deteksi kemiripan *white-box* dan *black-box*, serta teknologi pendukung seperti Virtiofs dan fitur Btrfs.

BAB 3 METODOLOGI PENELITIAN Merinci rencana pengujian, skenario simulasi data, variabel yang diukur, serta langkah-langkah pengambilan data.

BAB 4 IMPLEMENTASI EKSPERIMEN Menjelaskan teknis pelaksanaan penelitian, mulai dari penyiapan server, pembuatan skrip otomatisasi (Ansible) untuk simulasi pertumbuhan data, hingga eksekusi pengujian.

BAB 5 ANALISIS DAN PEMBAHASAN Menyajikan data hasil percobaan, menganalisis mengapa satu teknik lebih unggul dari yang lain berdasarkan data *hashfile* dan waktu eksekusi, serta menghubungkannya dengan teori.

BAB 6 PENUTUP Berisi kesimpulan akhir dari penelitian dan saran untuk pengembangan di masa depan.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Penelitian Terdahulu

Jin dan Miller (2009) melakukan analisis empiris mendalam terhadap efektivitas deduplikasi menggunakan pendekatan *black-box* pada citra diska VM dari repositori publik. Penelitian ini membuktikan bahwa teknik pemotongan data ukuran tetap (*fixed-size chunking*) mampu memberikan rasio deduplikasi yang sangat kompetitif dibandingkan teknik ukuran variabel (*variable-size chunking*), namun dengan beban komputasi yang jauh lebih ringan. Temuan ini mengungkap bahwa redundansi data pada VM sebagian besar bersifat struktural, berasal dari kesamaan sistem operasi dan aplikasi, serta keberadaan blok-blok kosong (*zero-filled blocks*).

Pada periode yang sama, Clements dkk. (2009) memperkenalkan DeDe, sebuah sistem deduplikasi terdesentralisasi yang dibangun di atas sistem berkas klaster berbasis blok (*block-based*). Berbeda dengan Jin dan Miller yang menganalisis citra secara *offline*, Clements dkk. menerapkan deduplikasi pada level blok di lingkungan SAN (*Storage Area Network*). Penelitian ini mendemonstrasikan tantangan dalam mengelola deduplikasi pada arsitektur *black-box*, di mana sistem berkas harus menjaga konsistensi data tanpa mengetahui semantik berkas di dalam sistem operasi *guest*. Meskipun DeDe berhasil mengurangi penggunaan ruang hingga 80%, pendekatannya menuntut modifikasi kompleks pada lapisan sistem berkas untuk menangani koordinasi antar *host*.

Melengkapi perspektif tersebut, Jayaram dkk. (2011) memperluas skala analisis ke lingkungan pusat data *cloud* produksi. Studi ini mengklasifikasi teknik deteksi kemiripan menjadi teknik *white-box* (berbasis pemahaman struktur sistem berkas) dan teknik *black-box* (berbasis pemindaian blok). Jayaram dkk. mengakui bahwa teknik *white-box* memiliki potensi akurasi yang lebih tinggi karena pemahaman semantik yang dimilikinya. Namun, dalam studi mereka, pendekatan *black-box* tetap dipilih karena penerapan teknik *white-box* pada masa itu dianggap memiliki hambatan teknis yang tinggi terkait rekonstruksi semantik.

Menjawab tantangan kompleksitas tersebut, Saurabh dkk. (2019) mengembangkan Expelliarmus, sebuah sistem manajemen citra VM yang sadar semantik (*semantics-aware*). Berbeda dengan teknik tradisional, sistem ini memodelkan citra VM sebagai grafik semantik untuk melakukan dekomposisi cerdas antara citra dasar dan paket perangkat lunak. Pendekatan ini terbukti mampu mengoptimalkan ukuran repositori secara signifikan, mencapai efisiensi penyimpanan 2,2 hingga 16 kali lipat dibandingkan sistem terdahulu seperti Mirage dan Hemera, sekaligus meningkatkan kinerja penerbitan dan pengambilan citra.

Lebih lanjut mengenai efisiensi granularitas, Meyer dan Bolosky (2012) memberikan wawasan krusial melalui studi terhadap 857 sistem berkas. Mereka

menemukan bahwa deduplikasi berbasis berkas utuh (*whole-file*) mampu mencapai sekitar tiga perempat (75%) dari penghematan ruang yang dihasilkan oleh deduplikasi level blok yang paling agresif. Temuan ini mengindikasikan bahwa pendekatan yang lebih sederhana dan sadar struktur (*structure-aware*)—seperti model *white-box*—tetap sangat kompetitif secara efisiensi ruang tanpa kompleksitas fragmentasi blok yang ekstrem,.

Dalam tinjauan literatur yang lebih modern, Thwel dan Sinha (2021) mempertegas kategorisasi pendekatan deduplikasi menjadi berbasis konten (*content-aware*) dan agnostik (*format-agnostic*). Mereka menyoroti bahwa deduplikasi *content-aware* memanfaatkan pengetahuan eksplisit tentang format data atau struktur sistem berkas untuk mempartisi data secara logis. Hal ini menghindari masalah pergeseran data (*byte shifting*) yang sering terjadi pada pendekatan blok ukuran tetap, menegaskan bahwa evolusi teknologi penyimpanan kini semakin memungkinkan penerapan deduplikasi yang memanfaatkan semantik data untuk optimasi yang lebih cerdas,.

2.2 Posisi Penelitian Ini

Penelitian ini memosisikan diri untuk mengisi celah teknis dan metodologis yang ditinggalkan oleh penelitian-penelitian sebelumnya. Jika Clements dkk. (2009) berfokus pada optimasi arsitektur berbasis blok (*block-based*) di lingkungan SAN dan Jayaram dkk. (2011) menyoroti hambatan teknis dalam merekonstruksi semantik *guest* pada pendekatan *white-box*, penelitian ini mengajukan pendekatan alternatif yang lebih praktis,. Meskipun Saurabh dkk. (2019) telah menawarkan solusi semantik melalui dekomposisi grafik, pendekatan tersebut menuntut kompleksitas pemodelan yang tinggi,. Di sisi lain, Meyer dan Bolosky (2012) memberikan landasan teoritis bahwa deduplikasi berbasis berkas sebenarnya memiliki potensi penghematan yang tinggi (hingga 87% pada kasus *backup*), namun belum banyak dieksplorasi dalam konteks virtualisasi *live* yang efisien.

Berangkat dari potensi tersebut, penelitian ini mengimplementasikan model penyimpanan *white-box* modern dengan memanfaatkan Virtio-FS. Teknologi ini mengatasi kendala "kotak hitam" dengan memungkinkan *host* mengakses sistem berkas *guest* secara langsung (*filesystem passthrough*), sehingga menghilangkan kebutuhan akan rekonstruksi semantik citra yang rumit seperti yang dikhawatirkan oleh Jayaram dkk.,. Sebagai landasan penyimpanan, penelitian ini memilih Btrfs bukan hanya karena statusnya sebagai teknologi *upstream* Linux, tetapi secara spesifik karena dukungannya terhadap deduplikasi *out-of-band* dan fitur *Copy-on-Write* (CoW) yang matang,. Fitur *out-of-band* ini krusial bagi penelitian untuk memisahkan beban kerja I/O reguler dari proses deduplikasi, memungkinkan pengukuran kinerja waktu yang lebih akurat. Dengan demikian, penelitian ini akan menguji secara empiris apakah integrasi Virtio-FS dan Btrfs mampu merealisasikan efisiensi ruang *white-box* yang diprediksi oleh literatur, sekaligus menawarkan kinerja waktu deduplikasi yang lebih unggul dibandingkan model *black-box* (citra diska tunggal) konvensional.

2.3 Virtualisasi

Virtualisasi adalah teknologi yang memungkinkan pembuatan representasi berbasis perangkat lunak (virtual) dari sesuatu yang bersifat fisik, seperti server, penyimpanan, atau jaringan. Inti dari virtualisasi adalah *Hypervisor* atau *Virtual Machine Monitor* (VMM), sebuah lapisan perangkat lunak yang bertugas membuat dan menjalankan VM (*Virtual Machine* - VM). *Hypervisor* seperti XEN dan QEMU/KVM memungkinkan satu komputer fisik (*host*) untuk menjalankan beberapa sistem operasi (*guest*) secara bersamaan dengan membagi sumber daya perangkat keras seperti CPU, memori, dan penyimpanan secara terisolasi (Binu dan Kumar, 2011).

Salah satu cara yang digunakan oleh hypervisor untuk menyetor data VM adalah dengan memanfaatkan suatu berkas pada host sebagai diska VM, berkas ini disebut citra VM (*VM image*) atau citra diska (*disk image*) yang fungsi awalnya adalah untuk mengarsip klona dari media penyimpanan fisik sebagai berkas (Morton, 2013). Citra ini dapat dibelah jadi dua kategori, yaitu datar (*flat*) dan (*sparse*). Berikut penjelasannya:

1. **Citra Diska Datar (*Flat Disk Image*):** Berkas berukuran tetap dengan satu blok fisik teralokasi untuk setiap blok logis berkas. Jika pengguna membuat diska virtual 10 GB, berkas ini akan langsung menempati ruang 10 GB di *host*. Blok yang belum digunakan oleh *guest* biasanya terisi nol (*zero-filled*), yang dapat menyebabkan pemborosan ruang penyimpanan jika tidak ditangani dengan baik.
2. **Citra Diska Jarang (*Sparse Disk Image*):** Berkas berukuran variabel dengan suatu ukuran maksimal yang dipandang sebagai ukuran diska oleh *guest*. Ukuran berkas ini bertumbuh seiring dengan terjadinya penulisan data baru. Jin dan Miller (2009) mencatat bahwa meskipun citra *sparse* dirancang agar efisien, citra ini tetap dapat mengandung blok-blok berisi nol jika sistem operasi *guest* secara eksplisit menulis data nol ke diska tersebut. Oleh karena itu, mekanisme seperti *TRIM/UNMAP/DISCARD* sangat penting untuk menjaga efisiensi citra jenis ini dengan memberitahu *host* untuk melepas alokasi blok yang tidak lagi digunakan (Jones, 2025).

2.4 Deduplikasi

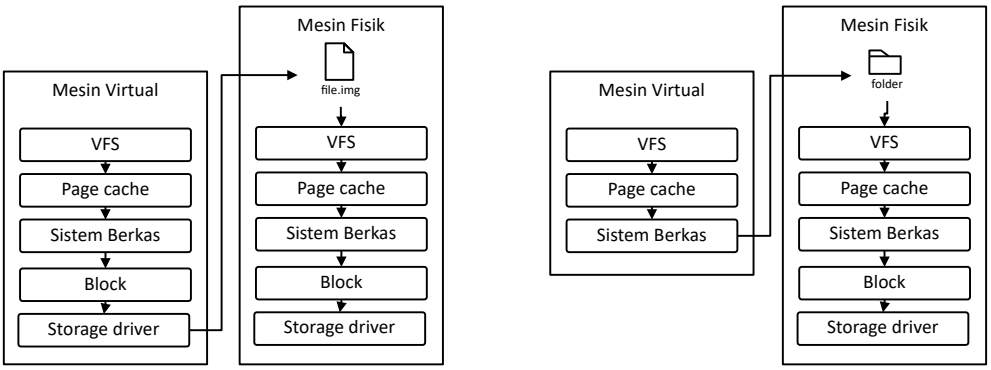
Deduplikasi data, yang sering disebut sebagai kompresi cerdas (*intelligent compression*) atau penyimpanan instansi tunggal (*single-instance storage*), adalah teknik khusus untuk menghilangkan salinan data yang berulang secara otomatis. Dalam proses ini, sistem menganalisis data untuk mengidentifikasi pola redundansi; jika duplikasi ditemukan, data tersebut tidak disimpan kembali, melainkan digantikan dengan penunjuk (*pointer*) logis ke data unik yang telah disimpan sebelumnya. Teknik ini sangat krusial untuk mengoptimalkan kapasitas penyimpanan dan mengurangi redundansi yang dihasilkan oleh banyak VM dengan sistem operasi yang identik.

2.4.1 Klasifikasi Berdasarkan Granularitas

Teknik identifikasi kemiripan data umumnya diklasifikasikan berdasarkan granularitas atau seberapa besar unit data yang diproses. Berdasarkan literatur (Meyer dan Bolosky, 2012; Saurabh dkk., 2019; Thwel dan Sinha, 2021), teknik ini dibagi menjadi tiga kategori utama:

1. Deduplikasi Level Blok (*Block-level*): Teknik ini yang disebut sebagai deduplikasi *black-box* pada penelitian ini, teknik ini memecah aliran data atau citra diska menjadi potongan-potongan (*chunks*) yang lebih kecil tanpa mempedulikan format berkas. Teknik ini dibagi lagi menjadi dua metode:
    - Fixed-size Chunking (Potongan Ukuran Tetap): Memotong data berdasarkan ukuran blok tetap (misal: 4 KB). Metode ini cepat dan efisien secara komputasi, namun rentan terhadap masalah pergeseran data (*offset shifting*). Jika satu bita disisipkan di awal berkas, seluruh batas potongan berikutnya akan bergeser, mengubah nilai *hash* seluruh blok, dan menggagalkan deteksi duplikasi,.
    - Variable-size Chunking (Potongan Ukuran Variabel): Menggunakan algoritma seperti *Rabin fingerprinting* untuk menentukan batas potongan berdasarkan isi konten (*content-defined*). Metode ini tahan terhadap masalah pergeseran data dan mampu mendeteksi duplikasi dengan lebih akurat (granularitas halus), namun membutuhkan sumber daya komputasi (CPU) yang jauh lebih intensif,.
2. Deduplikasi Level Berkas (*File-level*): Dikenal juga sebagai *Single Instance Storage* (SIS), teknik ini mendeteksi redundansi dengan membandingkan satu berkas utuh. Jika dua berkas memiliki *hash* identik, hanya satu salinan fisik yang disimpan. Metode ini memiliki *overhead* metadata yang rendah,. Meyer dan Bolosky (2012) menemukan bahwa pendekatan ini mampu mencapai sekitar 75% dari penghematan yang ditawarkan oleh deduplikasi level blok pada penyimpanan primer, menjadikannya alternatif yang efisien untuk beban kerja tertentu.
3. Deduplikasi Sadar Semantik (*Semantic-aware*)**:** Teknik ini yang disebut sebagai deduplikasi *white-box* pada penelitian ini, berbeda dengan teknik blok yang agnostik terhadap isi, teknik ini memahami format data atau struktur internal objek untuk mempartisi data secara logis. Dalam konteks virtualisasi, Saurabh dkk. (2019) mengembangkan sistem *Expelliarmus* yang memodelkan citra VM sebagai grafik semantik. Pendekatan ini melakukan dekomposisi cerdas untuk memisahkan citra dasar (*base image*) dan paket perangkat lunak, sehingga memungkinkan deteksi kemiripan yang lebih presisi pada level fungsional dibandingkan sekadar pemindaian bita mentah.

2.5 Model Citra VM *White-Box* dan Citra VM *Black-Box*



Gambar 2.1 Arsitektur Citra VM *Black-box* (Kiri) dan Citra *White-box*(Kanan)

Pemilihan model penyimpanan virtual sangat menentukan seberapa efektif teknik deduplikasi dapat diterapkan oleh sistem berkas *host*. Penelitian ini membedakan dua model utama berdasarkan visibilitas data dan struktur penyimpanannya:

1. Model Penyimpanan *Black-box* (Agnostik Semantik)**:** Dalam model ini, seluruh sistem operasi dan data VM dikemas dalam satu berkas citra diska tunggal (seperti `.img` atau `.qcow2`). Bagi sistem berkas *host*, citra ini hanyalah sebuah berkas biner besar yang "buram" (*opaque*).
    - Keterbatasan**:** *Host* tidak memiliki akses ke metadata atau struktur logis di dalam VM. Akibatnya, deduplikasi hanya dapat dilakukan pada level blok mentah (*block-level*) tanpa memahami konteks data tersebut (Saurabh dkk., 2019). Pendekatan ini sering kali membatasi kemampuan untuk mengidentifikasi fungsionalitas yang dapat digunakan kembali (*reusable functionality*) secara efisien.
2. Model Penyimpanan *White-box* (Sadar Semantik): Model ini tidak menggunakan citra diska tunggal, melainkan menyajikan penyimpanan VM sebagai direktori atau subvolume pada *host* yang diakses langsung melalui teknologi *filesystem passthrough* (Virtio-FS).
    - Keunggulan Semantik: *Host* memiliki visibilitas penuh terhadap hierarki berkas asli (misalnya `/usr`, `/bin`). Transparansi ini memungkinkan penerapan Deduplikasi Sadar Semantik (*Semantic-aware Deduplication*). Sebagaimana dijelaskan oleh Saurabh dkk. (2019), pendekatan sadar semantik memungkinkan sistem untuk mendekomposisi data berdasarkan unit logisnya—seperti memisahkan citra dasar (*base image*) dan paket perangkat lunak—sehingga hanya konten unik yang disimpan (Saurabh dkk., 2019).
    - Peran Btrfs: Dengan model *white-box*, sistem berkas Btrfs pada *host* dapat mengenali batas-batas berkas secara presisi. Hal ini memungkinkan Btrfs melakukan deduplikasi *extent* (menggunakan *reflink*) pada level berkas

atau paket yang identik antar-VM, sebuah efisiensi yang sulit dicapai jika data terbungkus dalam format *black-box*.

Gambar 2.1 mengilustrasikan perbedaan ini: Model *Black-box* mengisolasi semantik data dari *host*, sedangkan model *White-box* mengekspos struktur semantik tersebut, memungkinkan Btrfs melakukan optimasi penyimpanan yang lebih cerdas dan granular.

2.6 Perbandingan Teknologi Sistem Berkas Terbagi (*Shared Filesystem*)

Implementasi penyimpanan VM yang *white-box* sangat bergantung pada mekanisme sistem berkas terbagi antara *host* dan *guest*. Terdapat beberapa teknologi yang digunakan, namun masing-masing memiliki karakteristik arsitektur dan kinerja yang berbeda. Tabel 2.1 merangkum perbandingan antara NFS, Virtio-9p, dan Virtiofs.

- **NFS (*Network File System*)**: Sebuah protokol jaringan dan terstandarisasi (Noveck dan Lever, 2020). Meskipun dapat digunakan untuk berbagi direktori ke VM, pendekatan ini memiliki *overhead* kinerja dari tumpukan protokol TCP/IP dan kompleksitas konfigurasi untuk digunakan sebagai citra *boot*. Administrator perlu menyiapkan server NFS, mengkonfigurasi ekspor, dan memastikan VM dapat mengakses jaringan sebelum citra *boot* di-*mount (Foundries.io, Ltd., 2025; The kernel development community, 2025)*.
- **Virtio-9p**: Sistem berkas terbagi paravirtual yang lebih awal, berbasis pada protokol 9P dari sistem operasi Plan 9 yang bekerja di atas transport Virtio (QEMU Developers, 2025), sehingga tidak memiliki overhead jaringan. Namun, kinerjanya dianggap kurang memuaskan untuk beban kerja modern dan kini telah dianggap usang (deprecated) (Hajnoczi dkk., 2019) demi solusi yang lebih baru.
- **Virtiofs**: Dirancang sebagai suksesor virtio-9p (Hajnoczi dkk., 2019) yang berkinerja tinggi. Virtio-FS bertujuan untuk menyediakan kinerja dan semantik yang mendekati sistem berkas lokal dengan memanfaatkan secara penuh kedekatan antara guest dan host (Hajnoczi, 2020; Virtio-FS Developers, 2024).

Karena kemudahan konfigurasi untuk digunakan sebagai citra *boot* dan status tidak usang, penelitian ini memanfaatkan Virtio-FS untuk penyimpanan *white-box*.

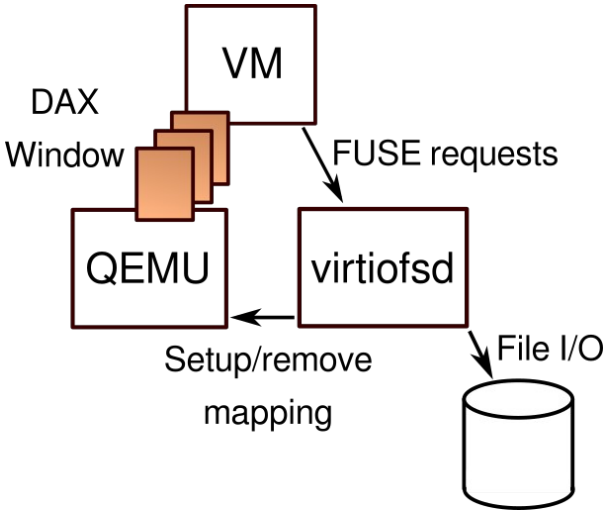
Tabel 2.1 Perbandingan teknologi berbagi berkas host-guest

Fitur	NFS	virtio-9p	Virtio-FS
Mekanisme	Protokol Jaringan (RPC di atas TCP/IP)	Protokol 9P di atas transport Virtio	Protokol *FUSE* di atas transport Virtio
Kinerja	Sedang (dibatasi oleh *overhead*	Rendah ke Sedang	Tinggi[1]

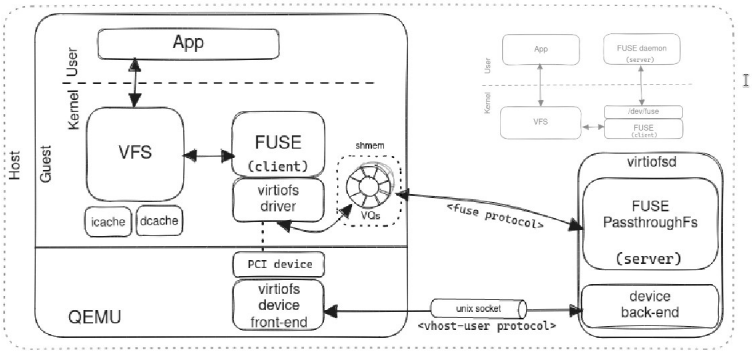
	jaringan)		
Overhead	Tinggi (*Stack* TCP/IP)	Rendah	Sangat Rendah
Semantik FS	Semantik Jaringan (misal, *caching* bisa lemah)	Mendekati POSIX, tapi tidak lengkap	Mendekati semantik FS Lokal
Kompleksitas	Tinggi (konfigurasi server, ekspor dan lingkungan pra-*boot*)	Sedang (konfigurasi QEMU)	Sedang (konfigurasi QEMU + *daemon*)

1 Dengan memanfaatkan fitur DAX

2.7 Sistem Berkas Virtio-FS



Gambar 2.2 Arsitektur Virtio-FS (Virtio-FS Developers, 2024)



Gambar 2.3 Detail Arsitektur Virtio-FS (Czenczek dan Maglione, 2023)

Virtiofs (Hajnoczi dkk., 2019; Virtio-FS Developers, 2024) adalah sebuah sistem berkas terbagi (*shared filesystem*) yang dirancang khusus untuk lingkungan tervirtualisasi dan memungkinkan sebuah direktori pada host untuk diakses secara langsung oleh *guest* sebagai sebuah sistem berkas. Gambar 2.2 dan Gambar 2.3 dan menunjukkan secara garis besar arsitekturnya, terdapat tiga komponen utama:

- **Virtiofsd**: perangkat lunak *daemon* yang berjalan pada sisi *host* dan mengeksekusi operasi I/O pada host atas perintah *guest*. Gambar 2.3 menggambarkan daemon ini memetakan I/O tersebut ke QEMU melalui *shmem* untuk kemudian diakses oleh *guest*. Secara opsional pemetaan bisa bersifat DAX (*Direct Access*). QEMU dan daemon ini berkomunikasi melalui protokol vhost-user.
- **QEMU**: QEMU berperan sebagai VMM yang bertugas memulai, mengatur dan mengawasi suatu VM. QEMU mampu menyediakan suatu DAX window dalam

bentuk PCI bar yang membuka sebuah wilayah memori agar VM bisa mengakses data secara langsung.

- **VM**: Sebagai sistem yang diladeni oleh Virtiofsd. Bisa diperhatikan pada Gambar 2.3, Virtio-FS diekspos pada VM oleh QEMU sebagai suatu PCI device yang dapat dibaca oleh *driver* Virtio-FS pada kernel sistem operasi *guest* dan dengan *driver* ini, *guest* dapat membuat permintaan *FUSE* kepada *daemon* yang ada pada host.

Walaupun *fuse* berperan sebagai protokol, tidak seperti implementasi *FUSE* tradisional yang terbatas pada satu sistem operasi, Virtio-FS dirancang untuk bekerja melintasi batas isolasi antara *guest* dan *host* dengan memanfaatkan transport *virtio*. Kinerja *Virtio-FS* mampu menyaingi kinerja *Viritio-blk* dengan penerapan fitur DAX yang tengah dikerjakan (Hajnoczi dkk., 2019).

2.8 Sistem Berkas Btrfs

Btrfs (*B-tree File System*) adalah sistem berkas modern berbasis *Copy-on-Write* (*CoW*) untuk Linux yang dirancang untuk mengatasi keterbatasan sistem berkas terdahulu dalam hal toleransi kesalahan, perbaikan dan administrasi dengan satuan data terendah setelah blok berupa *extent* (BTRFS Developers, 2025c). Pengembangan Btrfs dimulai pada tahun 2007 oleh Chris Mason (Salter, 2021) dengan arsitektur yang didasarkan pada struktur data *B-tree* yang optimal untuk mekanisme CoW sebagaimana dipaparkan dalam penelitian "*B-trees, Shadowing, and Clones*" (Rodeh, 2008).

Berbeda dengan sistem berkas terdahulu yang bersifat *journaling* (seperti *EXT4*) yang menimpa data di tempat (overwrite), Btrfs selalu menulis data baru ke lokasi blok yang berbeda sebelum memperbarui metadata *pointer*. Hal ini menjamin konsistensi sistem berkas tanpa memerlukan journaling data yang berlebihan dan memungkinkan fitur-fitur seperti *snapshot* yang efisien, kompresi transparan dan deduplikasi data (Tran dkk., 2024). Selanjutnya akan membahas soal fitur-fitur Btrfs yang dimanfaatkan untuk penelitian ini.

2.8.1 Kompresi Data Transparan

Kompresi data adalah pengkodean data dengan jumlah bit yang kurang dari representasi aslinya. Alhasil sistem berkas mampu menyimpan data dengan penggunaan ruang yang lebih sedikit sehingga menghemat ruang dan berpotensi mengurangi waktu baca dan tulis suatu data(Tran dkk., 2024)—sangat tergantung kinerja waktu algoritma dan CPU.

Kompresi data transparan merupakan fitur sistem berkas untuk mengompres suatu berkas ketika program menulis data serta mendekompresi ketika program membaca data tanpa pengetahuan program tersebut. Btrfs melakukan kompresi (BTRFS Developers, 2025a) pada level *extent* berkas sebelum data ditulis ke piringan. Data dipecah menjadi potongan kecil (maksimal 128 KiB) dan dikompres secara paralel. Btrfs mendukung tiga algoritma:

- ZLIB: Menawarkan rasio kompresi yang tinggi namun dengan kinerja kompresi/dekompresi yang lebih lambat.
- LZO: Dirancang untuk kinerja namun dengan rasio kompresi yang lebih rendah.
- ZSTD: Diperkenalkan di kernel 4.14, algoritma ini menawarkan keseimbangan terbaik dengan rasio kompresi yang sebanding dengan ZLIB namun kecepatan yang mendekati LZO. Penelitian ini menggunakan ZSTD karena karakteristiknya yang cocok untuk beban kerja *real-time* dengan rasio kompresi yang tetap bagus, level kompresi baku untuk ZSTD di Btrfs adalah tiga (3).

Btrfs juga menerapkan heuristik prakompresi (*pre-compression heuristics*). Sebelum melakukan kompresi penuh, sistem melakukan evaluasi statistik cepat (seperti frekuensi bita dan entropi Shannon) pada data. Jika data terdeteksi sulit dikompres (misalnya data yang sudah terenkripsi), sistem akan menandai berkas tersebut dengan *flag* `NOCOMPRESS` untuk menghindari pemborosan siklus CPU. Heuristik tersebut bisa dilewati dengan opsi mount `compress-force=<algoritma>`.

2.8.2 Deduplikasi

Btrfs menerapkan deduplikasi tipe *out-of-band* dan terdapat dua program yang menerapkan deduplikasi untuk Btrfs, *BEES* dan *Duperemove* (Fasheh, 2025; Zygo, 2025). Di mana *BEES* adalah program deduplikasi yang berorientasi blok dan *Duperemove* adalah program deduplikasi yang berorientasi berkas—yaitu membaca daftar berkas dan mengidentifikasi data duplikat pada level *extent* dan identifikasi pada level blok yang opsional. *Duperemove* menjadi pilihan untuk penelitian ini karena cocok untuk berkas yang berasal dari citra dasar yang sama (BTRFS Developers, 2025b).

2.8.3 *Reflink* dan Salinan Tertaut

Fitur *Reflink* (*Reference Link*) (BTRFS Developers, 2025d) adalah implementasi langsung dari arsitektur CoW Btrfs. Perintah `cp --reflink=always` menciptakan salinan berkas yang "dangkal" (*shallow copy*). Secara internal, Btrfs membuat *inode* baru yang berbagi *extent* data fisik yang sama dengan berkas asli, menawarkan salinan yang efisien namun tetap terisolasi—modifikasi pada salah satu *reflink* akan memicu alokasi blok baru hanya untuk reflink yang berubah, membiarkan berkas lainnya tetap utuh.

2.8.4 *Subvolume* dan *Snapshot*

Btrfs memiliki konsep *subvolume (BTRFS Developers, 2025e)*, yang merupakan bagian dari sistem berkas dengan hierarki berkas/direktori dan ruang nama *inode* yang independen. Sebuah *subvolume* dapat berbagi *extents* berkas dengan *subvolume* lainnya dalam satu *pool* penyimpanan fisik yang sama. Berbeda dengan *Logical Volume* pada LVM yang beroperasi di level blok, *subvolume* Btrfs beroperasi pada level *extent* berkas. *Subvolume* tampak dan berperilaku seperti

direktori biasa yang dapat diganti namanya atau dipindahkan. Namun, ia memiliki ID numerik persisten (*subvolid*) dan dapat di-*mount* secara terpisah layaknya sistem berkas independen.

Snapshot—yaitu cuplikan sistem berkas pada suatu poin waktu (Tran dkk., 2024)—pada Btrfs adalah sebuah *subvolume* yang diinisialisasi dengan konten yang sama persis dengan suatu *subvolume* lain. Berkat mekanisme CoW dan struktur B-tree, pembuatan *snapshot* berlangsung instan dan tidak menduplikasi data secara fisik. *Snapshot* dan aslinya berbagi semua blok data sampai terjadi modifikasi (BTRFS Developers, 2025e).

2.9 Metode Analisis Statistik dan Visualisasi Data

Metode-metode ini menjadi alat untuk mengolah data kuantitatif yang dihasilkan dari eksperimen, sehingga dapat ditarik kesimpulan yang objektif dan valid.

2.9.1 Statistik Deskriptif

Statistik deskriptif digunakan untuk meringkas dan mendeskripsikan karakteristik utama dari suatu kumpulan data. Dalam penelitian ini, beberapa ukuran deskriptif yang dimanfaatkan adalah:

- **Rata-rata (Mean)**: Nilai tengah yang mewakili keseluruhan data, dihitung dengan menjumlahkan semua nilai dan membaginya dengan jumlah data. Digunakan untuk mengetahui kecenderungan sentral dari suatu variabel, seperti rata-rata panjang *extent*.
- **Median**: Nilai tengah setelah data diurutkan. Ukuran ini lebih robust terhadap *outlier* dibandingkan rata-rata.
- **Modus**: Nilai yang paling sering muncul dalam suatu dataset.
- **Koefisien Variasi (Coefficient of Variation / CV)**: Ukuran relatif dari dispersi data, yang dihitung dengan rumus

$$CV = \frac{\sigma}{\mu} \times 100\,\% \quad (1)$$

di mana σ adalah standar deviasi dan μ adalah rata-rata. CV digunakan untuk membandingkan tingkat variabilitas data dari kelompok yang berbeda, meskipun satuannya berbeda, seperti dalam analisis stabilitas waktu eksekusi.

2.9.2 Analisis Regresi dan Pemodelan

Analisis regresi digunakan untuk memodelkan hubungan antara variabel dependen (hasil) dan variabel independen (prediktor), serta untuk memprediksi nilai.

- **Regresi Linear**: Memodelkan hubungan linier antara dua variabel dengan persamaan garis lurus $y = ax + b$, di mana a adalah kemiringan dan b adalah intersep. Digunakan untuk menganalisis tren sederhana.

- **Regresi Polinomial (Kuadratik)**: Memodelkan hubungan non-linier dengan persamaan $y = ax2 + bx + c$. Metode ini cocok untuk data yang menunjukkan pola percepatan atau perlambatan pertumbuhan, seperti peningkatan waktu eksekusi yang tidak proporsional terhadap penambahan VM.
- **Koefisien Determinasi (R^2)**: Mengukur seberapa baik model regresi menjelaskan variasi dalam data. Nilai R^2 berkisar antara 0 hingga 1, di mana nilai mendekati 1 menunjukkan bahwa model dapat menjelaskan sebagian besar variasi data observasi.

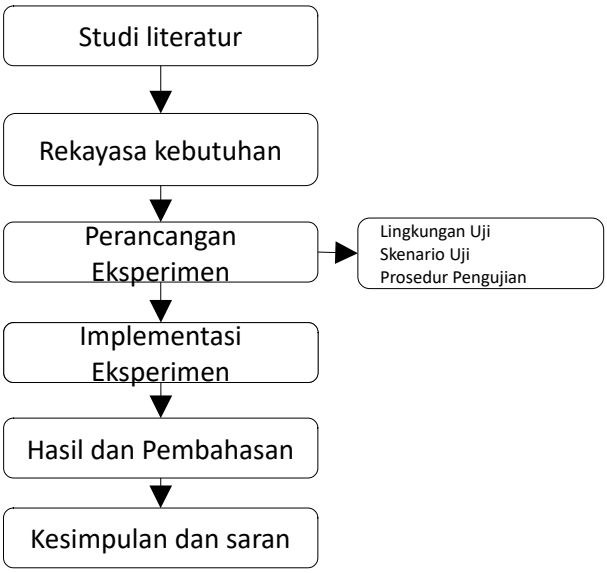
2.9.3 Teknik Visualisasi Data

Visualisasi data adalah penyajian data dalam bentuk grafis untuk memudahkan pemahaman pola, tren, dan *outlier*. Teknik yang digunakan antara lain:

- **Grafik Garis (Line Chart)**: Menampilkan perubahan suatu metrik (seperti ukuran penyimpanan atau waktu eksekusi) secara berkelanjutan terhadap variabel lain (misalnya jumlah VM). Efektif untuk menunjukkan tren temporal atau progresif.
- **Grafik Batang (Bar Chart)**: Membandingkan besaran nilai dari beberapa kategori atau kelompok yang berbeda, seperti perbandingan ukuran akhir antar skenario UPR.
- **Violin Plot**: Menggabungkan *box plot* dengan estimasi kepadatan probabilitas (*kernel density estimation*). Diagram ini tidak hanya menunjukkan median, kuartil, dan rentang data, tetapi juga bentuk distribusinya, sehingga sangat berguna untuk membandingkan distribusi ukuran *extent* antar kelompok eksperimen.
- **Diagram Pencar (Scatter Plot) dengan Garis Regresi**: Menampilkan titik-titik data individu untuk dua variabel, dilengkapi dengan garis model regresi yang dipasang. Berguna untuk mengevaluasi kekuatan hubungan dan kecocokan model.

BAB 3 METODOLOGI PENELITIAN

Penelitian ini menggunakan metode eksperimen kuantitatif untuk membandingkan penggunaan ruang mesin-VM dengan metode penyimpanan dan pemafaatan fitur Btrfs yang berbeda. Metode ini bertuju menentukan tingkat penghematan yang bisa diraih dengan memanfaatkan beragam fitur sistem berkas Btrfs yaitu reflink, snapshot, deduplikasi serta kompresi antara citra VM *white-box* dan *black-box*. Penelitian ini hendak membuat sejumlah VM yang terbuat dari citra dasar yang sama dan menumbuhkannya melalui otomasi deterministik pada volume Btrfs yang terkompres dan yang tidak terkompres. Kemudian pada citra-citra VM akan dijalankan proses deduplikasi dan waktu eksekusi proses ini akan direkam. Untuk memastikan konsistensi data waktu eksekusi, proses deduplikasi akan dijalankan berulang kali. Alur metodologi penelitian terilustrasi dalam Gambar 3.1.



Gambar 3.1
Metodologi penelitian

3.1 Studi Literatur

Studi literatur dilakukan guna memahami teori, konsep dan kerangka kerja yang mendasari penelitian melalui sumber-sumber literatur, buku, jurnal, dokumentasi, artikel dan sumber lainnya. Studi literatur difokuskan pada teknologi sistem berkas Btrfs, penyimpanan VM dan deduplikasi.

3.2 Rekayasa Kebutuhan [Revisi]

Pada tahap ini, diidentifikasi serangkaian kebutuhan fungsional yang harus dipenuhi oleh lingkungan pengujian untuk memastikan penelitian dapat berjalan dengan valid, konsisten, dan terukur. Kebutuhan ini menjadi landasan bagi perancangan arsitektur dan pemilihan teknologi pada tahap implementasi nanti.

3.2.1 Platform Host

- **Sistem Operasi:** Diperlukan sistem operasi berbasis Linux yang mendukung virtualisasi berbasis kernel (KVM) secara *native*. Dukungan *native* ini krusial untuk meminimalkan *overhead* sistem operasi dan memastikan alokasi sumber daya yang optimal bagi VM selama pengujian beban tinggi.
- **Sistem Berkas Btrfs:** Dipilih sebagai fondasi penyimpanan karena statusnya sebagai teknologi *upstream* dalam kernel Linux dan dukungan spesifiknya terhadap deduplikasi *out-of-band*,. Fitur *out-of-band* ini menjadi syarat mutlak penelitian untuk memisahkan beban kerja I/O reguler dari proses deduplikasi. Hal ini memungkinkan pengukuran kinerja waktu deduplikasi secara terisolasi dan akurat, berbeda dengan sistem deduplikasi *inline* (seperti VDO atau ZFS) yang memproses data secara bersamaan saat ditulis sehingga menyulitkan pengukuran dampak kinerja yang terpisah.

3.2.2 Platform Virtualisasi

- **Hypervisor Tipe-1 (QEMU/KVM):** Diperlukan *hypervisor* yang terintegrasi langsung dengan kernel *host*. QEMU/KVM dipilih karena memiliki dukungan matang terhadap **Virtio-FS**,. Teknologi ini merupakan prasyarat mutlak untuk skenario *white-box* agar direktori *host* dapat diekspos sebagai sistem berkas *root* yang dapat di-*boot* oleh VM, sebuah kemampuan yang terbatas pada *hypervisor* tipe-2 lainnya.
- **API Manajemen VM (Libvirt):** Diperlukan antarmuka standar untuk mengelola siklus hidup VM (pembuatan, konfigurasi, penghapusan) secara terprogram. Penggunaan Libvirt menjamin bahwa konfigurasi perangkat keras virtual identik untuk setiap pengulangan uji, sehingga meminimalkan variabel pengganggu (*confounding variables*).

3.2.3 Otomasi dan Provisioning

- **Inisialisasi Awal VM (Cloud-init):** Diperlukan mekanisme otomatis untuk melakukan konfigurasi awal (nama *host*, jaringan, kunci SSH) pada saat *booting* pertama. Hal ini meniadakan intervensi manual yang rentan kesalahan manusia (*human error*) dan mempercepat proses penyebaran VM dalam jumlah banyak.
- **Alat Manajemen Konfigurasi (Ansible):** Diperlukan alat otomasi untuk menjalankan simulasi pertumbuhan data. Ansible dipilih untuk menjamin konsistensi dan determinisme data. Dengan Ansible, urutan penulisan, penghapusan, dan modifikasi berkas dapat direplikasi secara persis pada 10 hingga 40 VM yang diuji. Determinisme ini vital agar perbandingan rasio deduplikasi antar-kelompok menjadi valid secara statistik dan tidak bias akibat perbedaan perlakuan manual.

3.2.4 Pengukuran dan Analisis

- **Utilitas Analisis Ruang (Compsize):** Diperlukan alat ukur spesifik Btrfs yang mampu membedakan antara ukuran data logis, ukuran terkompresi, dan

ukuran fisik setelah deduplikasi. Alat standar seperti `du` atau `df` tidak dapat digunakan dalam penelitian ini karena tidak mampu mendeteksi penghematan ruang hasil *reflink* dan kompresi transparan secara terperinci.

- **Utilitas Deduplikasi (Duperemove):** Diperlukan perangkat lunak deduplikasi *out-of-band* yang bekerja pada level *extent* untuk sistem berkas Btrfs. Alat ini dipilih karena kemampuannya memindai dan melakukan deduplikasi data duplikat tanpa mengganggu integritas sistem berkas yang sedang berjalan (*live filesystem*), memungkinkan simulasi deduplikasi berkala sesuai skenario dunia nyata.

3.3 Perancangan Eksperimen

Penelitian ini dirancang untuk melakukan analisis komparatif terhadap penghematan ruang penyimpanan antara dua model arsitektur penyimpanan VM yang secara fundamental berbeda: Penyimpanan *black-box* dan Penyimpanan *white-box*. Tujuan utamanya adalah untuk mengukur dan membandingkan efektivitas berbagai tingkatan strategi penghematan (CoW, deduplikasi, kompresi) pada kedua model arsitektur tersebut.

3.3.1 Lingkungan Uji

Lingkungan pengujian akan terdiri dari suatu lab yang menjalankan sistem operasi Linux dengan KVM sebagai *hypervisor*. Untuk memfasilitasi perbandingan, akan disiapkan dua jenis citra VM dasar. (1) **Citra dasar** (*Base Image*): sebuah berkas citra diska jarang yang berisi instalasi sistem operasi dan aplikasi standar. Aset ini akan digunakan untuk skenario pengujian penyimpanan VM *black-box*. (2) **Direktori dasar** (*Base Directory*): Sebuah direktori yang berupa *subvolume* pada sistem berkas host yang berisi salinan lengkap sistem operasi dan aplikasi standar yang identik dengan isi citra dasar dan akan digunakan untuk skenario pengujian penyimpanan VM *white-box*. Aset-aset ini merupakan citra yang menyimpan sistem operasi Fedora Server 42.

Akan disiapkan dua volume yang identik pada host, tempat semua aset pengujian seperti citra diska dan direktori akan disimpan. (1) **Volume normal**: Diformat dengan sistem berkas Btrfs, namun tanpa memanfaatkan fitur kompresi sistem berkas itu dan (2) **Volume terkompresi**: Diformat dengan sistem berkas Btrfs, namun dengan memanfaatkan fitur kompresinya.

Seluruh pengujian dilakukan pada mesin lokal dengan CPU 8 core 16 threads dan memori 16 GB pada sistem operasi Arch Linux, dengan 4 core dan 8 threads terisolasi hanya untuk proses deduplikasi guna menjalankan pengujian agar terjauhi dari ketimpangan sumber daya. Proses deduplikasi ini juga akan diberikan prioritas I/O tertinggi. Spesifikasi mesin pengujian dapat dilihat pada Tabel 3.1.

Tabel 3.1 Spesifikasi Mesin Lokal

CPU	AMD Ryzen™ 7 7735 HS 8 core 16 threads

Memori	16 GB LPDDR5 6400 Mhz
Sistem Operasi	Arch Linux
Kernel	Linux 6.17.9-arch1-1.1
Versi Duperemove	0.15.2

3.3.2 Persiapan Dataset Eksperimen

Sebelum pengukuran dilakukan, diperlukan serangkaian prosedur untuk membangkitkan dataset VM yang valid, konsisten, dan representatif. Prosedur ini menjamin bahwa setiap skenario pengujian dilakukan pada objek yang identik. Tahapan persiapan dataset adalah sebagai berikut:

1. **Inisialisasi Citra Dasar:** Citra dasar dan direktori dasar disalin ke kedua volume penyimpanan (*volume normal* dan *volume terkompresi*). Hal ini diperlukan karena fitur *Copy-on-Write* (reflink dan snapshot) pada Btrfs hanya dapat bekerja dalam batas satu sistem berkas (*filesystem boundary*).
   - Untuk model *Black-box*, citra dasar disalin sebagai citra diska jarang.
   - Untuk model *White-box*, direktori dasar disalin sebagai *subvolume*.
2. **Pembangkitan dan Pertumbuhan VM:** Dari citra dasar tersebut, dibuat 10 salinan tertaut (*linked copies*) menggunakan *reflink* (untuk *black-box*) dan *snapshot* (untuk *white-box*). Kesepuluh VM ini kemudian dijalankan dan ukurannya "ditumbuhkan" menggunakan otomasi *Ansible Playbook* sesuai profil pertumbuhan yang telah ditentukan.
3. **Normalisasi Ruang (TRIM):** Setelah proses pertumbuhan data selesai, khusus pada VM model *Black-box*, dilakukan operasi `fstrim`. Langkah ini bertujuan melepaskan alokasi blok kosong (*zero blocks*) yang mungkin terbentuk selama proses penulisan data pada citra d*iska jarang*, sehingga perbandingan penggunaan ruang dengan model *White-box* (yang berbasis direktori) menjadi adil.
4. **Isolasi Dataset Deduplikasi:** Untuk menjaga integritas dataset utama, proses deduplikasi (`duperemove`) tidak dijalankan langsung pada 10 citra VM yang telah tumbuh. Sebaliknya, dibuat salinan tertaut baru (menggunakan *snapshot* atau *reflink*) dari dataset yang telah tumbuh tersebut ke dalam direktori pengujian terpisah. Dengan demikian, dataset asli tetap utuh dan dapat digunakan kembali untuk skenario pengujian yang berbeda tanpa perlu melakukan simulasi pertumbuhan ulang.

3.3.3 Pertumbuhan Citra VM

Untuk merepresentasikan lingkungan produksi yang realistis di mana data terus berubah, penelitian ini menerapkan simulasi beban kerja (*workload simulation*) yang dijalankan dalam serangkaian tahapan sekuensial. Simulasi ini dieksekusi dalam tujuh iterasi logis yang merepresentasikan tahapan perubahan keadaan data (*state transition*) secara bertahap. Tujuan utama dari tahapan ini adalah menciptakan divergensi data antar VM yang bermula dari citra dasar yang

identik, sehingga efektivitas deduplikasi dan kompresi dapat diuji pada kondisi data yang heterogen.

Berdasarkan karakteristik penggunaan peladen pada umumnya, didefinisikan tiga profil pertumbuhan data (*growth profiles*) yang berbeda. Setiap profil memiliki serangkaian instruksi untuk setiap iterasi yang meliputi instalasi paket, modifikasi berkas teks, kompilasi kode, hingga pembuatan data biner acak. Rincian karakteristik, aktivitas dan pemetaan VM untuk setiap profil disajikan pada Tabel 3.2.

Tabel 3.2 Profil Pertumbuhan Data VM

Profil	Tiper Server	Deskripsi Aktivitas	Untuk VM
A	Web/API Server	**Karakteristik:** Mensimulasikan peladen web (Nginx) dengan pola penulisan log yang intensif dan fragmentasi berkas. Aktivitas Utama: 1. Pemasangan paket `nginx`, `git`, dan kloning repositori kode (Podman). 2. Simulasi penulisan log akses (`access.log`) secara kumulatif (metode *append*). 3. Simulasi unggahan pengguna (*user uploads*) berupa banyak berkas kecil (50 berkas @ 2MB) menggunakan `fio`.	vm-01, vm-04, vm-06, vm-09
B	CI/CD Build Server	**Karakteristik:** Mensimulasikan peladen integrasi berkelanjutan (*Build Server*) dengan tingkat perputaran data (*data churn*) yang tinggi akibat kompilasi ulang. Aktivitas Utama: 1. Pemasangan lingkungan Java (OpenJDK 21) dan Apache Maven. 2. Kompilasi proyek *Spring Petclinic* yang menghasilkan artefak biner (.class dan .jar) yang berubah setiap iterasi. 3. Pembuatan data uji acak berukuran sedang (250 MB).	vm-02, vm-05, vm-08
C	Utility Server	**Karakteristik:** Mensimulasikan peladen utilitas umum untuk pemantauan dan pencadangan data (*backup*).	vm-03, vm-07, vm-10

	Aktivitas Utama: 1. Instalasi alat administrasi (htop, tmux, ansible). 2. Penulisan berkas biner acak berukuran besar (400 MB) menggunakan /dev/urandom untuk mensimulasikan arsip cadangan. 3. Pembaruan paket sistem rutin.

Secara alur logika, iterasi ke-1 dan ke-2 difokuskan pada inisialisasi lingkungan (*initial setup*) dan pemasangan perangkat lunak dasar. Iterasi ke-3 hingga ke-7 difokuskan pada simulasi beban kerja aktif dan pemeliharaan, termasuk eksekusi pembaruan paket sistem (`dnf update`) yang dijalankan pada semua profil untuk mensimulasikan perubahan data sistem operasi yang wajar terjadi pada siklus hidup server.

3.3.4 Definisi Kelompok Eksperimen

Penelitian ini mendefinisikan empat kelompok eksperimen utama berdasarkan kombinasi model penyimpanan (*black-box* atau *white-box*) dan konfigurasi volume Btrfs (normal atau terkompresi). Keempat kelompok ini menjadi kerangka dasar untuk seluruh skenario pengujian, baik untuk pengukuran penghematan ruang maupun kinerja waktu. Definisi masing-masing kelompok dirangkum dalam Tabel 3.3.

Tabel 3.3 Definisi Kelompok Eksperimen

Kode Kelompok	Model Penyimpanan	Konfigurasi Volume Btrfs	Deskripsi
G-1	*Black-box*	Normal (tanpa kompresi)	Citra VM berupa citra diska jarang format RAW yang disimpan pada volume Btrfs tanpa kompresi.
G-2	*Black-box*	Terkompresi (zstd level 3)	Citra VM berupa citra diska jaran format RAW yang disimpan pada volume Btrfs dengan kompresi transparan menggunakan algoritma Zstd level 3.
G-3	*White-box*	Normal (tanpa kompresi)	Citra VM berupa subvolume Btrfs yang diekspos ke *guest* melalui Virtio-FS (*filesystem passthrough*) pada volume Btrfs tanpa kompresi.
G-4	*White-box*	Terkompresi	Citra VM berupa subvolume

	(zstd level 3)	Btrfs yang diekspos ke *guest* melalui Virtio-FS pada volume Btrfs dengan kompresi transparan menggunakan algoritma Zstd level 3.

Seluruh kelompok eksperimen menggunakan basis sistem operasi yang sama (Fedora Server 42) dan melalui proses simulasi pertumbuhan data yang identik melalui skenario yang telah ditentukan. Pengelompokan ini memungkinkan analisis yang terisolasi terhadap pengaruh masing-masing variabel—yaitu model penyimpanan dan status kompresi—terhadap efektivitas penghematan ruang dan kinerja sistem.

3.3.5 Skenario Uji

Pengujian dibagi menjadi dua kategori utama berdasarkan parameter yang diukur, yaitu **Uji Penghematan Ruang Penyimpanan** dan **Uji Kinerja Waktu Deduplikasi**.

3.3.5.1 Uji Penghematan Ruang Penyimpanan

Skenario ini bertujuan untuk mengukur efektivitas penghematan ruang disk yang dihasilkan oleh fitur Copy-on-Write (CoW) Btrfs dan utilitas duperemove. Pengujian pada skenario ini dilakukan tanpa pengulangan (*single run*). Hal ini didasarkan pada karakteristik objek uji yang berupa himpunan data statis (*static datasets*).

Pengukuran dilakukan pada **empat kelompok eksperimen (G-1 hingga G-4)** seperti yang telah didefinisikan. Setiap kelompok tersebut akan melalui tahapan pengukuran sesuai rincian skenario pada

Tabel 3.4.

Tabel 3.4 Rincian Skenario Uji Penghematan Ruang Penyimpanan

Kode	Skenario	Deskripsi
UPR-1	Ukuran Dasar (Tanpa Optimasi)	Kondisi di mana setiap VM memiliki salinan data fisik yang independen dan penuh (*full copy*), tanpa berbagi blok data antar VM.
UPR-2	*Copy-on-Write* (Reflink/Snapshot)	Kondisi di mana VM merupakan turunan dari citra dasar yang saling berbagi blok data identik melalui mekanisme *reflink* atau *snapshot* sebelum ada data baru ditulis.
UPR-3	Deduplikasi Penuh (*Full Dedupe*)	Kondisi UPR-2 yang telah diproses oleh algoritma deduplikasi purnawaktu (*post-process*) untuk menyatukan kembali blok data yang terpisah akibat

		penulisan baru.
UPR-4	Deduplikasi Parsial (128K)	Sama dengan UPR-3, namun proses deduplikasi dibatasi hanya pada sebagian berkas dengan ukuran *chunk* data 128 KiB.
UPR-5	Deduplikasi Parsial (64K)	Sama dengan UPR-3, namun proses deduplikasi dibatasi hanya pada sebagian berkas dengan ukuran *chunk* data 64 KiB.

3.3.5.2 Uji Kinerja Waktu Deduplikasi

Skenario ini bertujuan untuk mengukur waktu yang dibutuhkan (Waktu Real, *User*, dan *Sys*) untuk menjalankan proses deduplikasi duperemove. Berbeda dengan uji penghematan ruang, pengukuran waktu bersifat stokastik dan dipengaruhi oleh kondisi sistem, sehingga pengujian dilakukan dengan repetisi (*pengulangan*) sebanyak 3 kali (*triplo*) untuk menjamin validitas data.

Untuk efisiensi waktu penelitian tanpa mengurangi akurasi pembacaan tren, pengambilan sampel jumlah VM dilakukan dengan interval [1, 2, 4, 6, 8, 10]. Pengujian dilakukan pada keempat kelompok eksperimen (G-1 hingga G-4) yang telah didefinisikan sebelumnya. Dengan demikian, dapat diukur perbandingan waktu deduplikasi antara model *black-box* dan *white-box*, serta pengaruh kompresi pada masing-masing model, tanpa perlu kode kategori tambahan.

3.3.6 Analisis Statistik Basis Data Deduplikasi

Selain pengukuran penggunaan ruang disk secara makro, penelitian ini juga melakukan analisis mikro terhadap basis data hash (*hashfile*) yang dihasilkan oleh utilitas `duperemove`. *Hashfile* ini merupakan basis data berbasis SQLite yang menyimpan metadata mengenai struktur berkas, *extent*, dan blok data yang telah dipindai selama proses deduplikasi.

Analisis ini bertujuan untuk memvalidasi granularitas deduplikasi yang terjadi dan memahami karakteristik distribusi data pada skenario **UPR-3**, **UPR-4**, dan **UPR-5** pada grup G-1 hingga G-4. Data diekstraksi secara *post-mortem* dari tabel `files`, `extents`, dan `blocks` yang terdapat di dalam *hashfile*. Parameter yang diukur dalam analisis ini dirincikan pada Tabel 3.5.

Tabel 3.5 Parameter Analisis Statistik Hashfile

Parameter	Deskripsi	Manfaat
Total Extent Dipindai	Jumlah total segmen data kontinu (*extents*) yang dipetakan dari seluruh berkas VM.	Menggambarkan tingkat fragmentasi data yang diproses oleh *duperemove*.
Rata-rata Panjang	Nilai rata-rata ukuran *extent* dalam satuan	Menunjukkan efisiensi pemecahan data; *extent* yang lebih besar

Extent	Bytes.	umumnya lebih efisien dalam pemrosesan I/O.
Jumlah Extent Kembar	Jumlah *extent* yang memiliki nilai *hash* (digest) identik dengan *extent* lain (redundansi).	Indikator utama potensi deduplikasi. Semakin tinggi nilainya, semakin banyak ruang yang dapat dihemat melalui mekanisme *clone*/*reflink*.
Jumlah Blok Kembar	Jumlah blok data (pada skenario blok statis) yang memiliki nilai *hash* identik.	Digunakan khusus untuk memvalidasi efektivitas deduplikasi pada level blok yang lebih kecil (misal: sisa potongan data).

Pengambilan data untuk analisis ini dilakukan secara kumulatif seiring dengan penambahan jumlah VM (dari 1 hingga 10 VM), guna melihat tren peningkatan redundansi data (data kembar) yang terbentuk seiring bertambahnya populasi VM yang identik.

3.4 Teknik Analisis Data

Data yang diperoleh dari eksperimen diklasifikasikan menjadi dua kategori: data deterministik (penggunaan ruang penyimpanan) dan data stokastik (waktu eksekusi). Oleh karena itu, teknik analisis data dibedakan berdasarkan karakteristik variabel yang diukur.

3.4.1 Hitung Penghematan Ruang Penyimpanan

Pengukuran ruang penyimpanan bersifat deterministik, di mana nilai yang dihasilkan konstan untuk input dan konfigurasi yang sama karena bekerja pada aset data yang statis. Oleh karena itu, analisis inferensial (uji statistik) tidak diterapkan pada data ini. Analisis dilakukan secara **deskriptif komparatif** untuk menghitung persentase penghematan ruang (*Space Savings*).

Penghematan dihitung dengan membandingkan penggunaan ruang disk aktual (*Disk Usage*) dari skenario eksperimen terhadap penggunaan ruang dasar (*Referenced*) yang merepresentasikan penyimpanan tanpa optimasi. Formula yang digunakan adalah:

$$Penghematan\%(x)=(\frac{UPR_1-UPR_x}{UPR_1})\times 100\% \quad (2)$$

Di mana:

- UPR_1: Ukuran data logis (
- UPR_x: Ukuran fisik pada disk setelah penerapan fitur (UPR-2, UPR-3, UPR-4 dan UPR-5).

Hasil perhitungan akan disajikan dalam bentuk grafik garis untuk memvisualisasikan tren penghematan seiring bertambahnya jumlah VM (1 hingga 10 unit).

3.4.2 Analisis Kinerja Waktu Deduplikasi

Data waktu eksekusi deduplikasi bersifat stokastik karena dapat dipengaruhi oleh variabel lingkungan sistem yang fluktuatif, seperti cache, penjadwalan proses, dan aktivitas I/O latar belakang. Oleh karena itu, teknik analisis statistik diterapkan untuk menjamin validitas dan reliabilitas data. Waktu eksekusi diukur dalam tiga pengulangan (*triplo*) untuk setiap konfigurasi jumlah VM (1, 2, 4, 6, 8, 10) pada keempat kelompok eksperimen (G-1 hingga G-4).

Analisis dilakukan dengan pendekatan berikut:

1. **Statistik Deskriptif dan Stabilitas**:
    - Rata-rata (*mean*) dari ketiga pengulangan dihitung sebagai nilai representatif.
    - Koefisien Variasi (CV) dihitung untuk setiap konfigurasi menggunakan.
2. **Analisis Tren dan Pemodelan (Curve Fitting)**:
    - Tren peningkatan waktu eksekusi terhadap jumlah VM dimodelkan menggunakan analisis regresi.
    - Dua model diuji untuk setiap kelompok dan komponen waktu (Total, Pemindaian, Deduplikasi):
        - **Model Linear**: $y = ax + b$, untuk mendeteksi pertumbuhan konstan.
        - **Model Polinomial Kuadratik**: $y = ax^2 + bx + c$, untuk mendeteksi percepatan pertumbuhan.
    - Kualitas kecocokan model dievaluasi menggunakan **Koefisien Determinasi (**R^2**)**. Model dengan R^2 terdekat ke 1 dipilih sebagai model prediktif terbaik.
    - Model terpilih kemudian digunakan untuk melakukan **ekstrapolasi**, memperkirakan waktu eksekusi pada skala VM yang lebih besar (hingga 200 VM) dan untuk menghitung **titik potong** antar kelompok, yaitu jumlah VM di mana waktu eksekusi dua kelompok menjadi sama.

3.4.3 Perbandingan Penghematan dan Waktu

Untuk memberikan rekomendasi praktis yang mempertimbangkan *trade-off* antara ruang dan waktu, didefinisikan metrik terintegrasi **Persentase Penghematan per Menit**. Metrik ini mengkuantifikasi seberapa cepat suatu arsitektur menghasilkan penghematan ruang. Perhitungan dilakukan dengan dua pendekatan:

3. **Analisis Aktual (10 VM)**:

$$Efisiensi = \frac{Persentase\ Penghematan\ UPR-3}{Waktu\ Deduplikasi\ Total\ (menit)} \quad (3)$$

    Waktu deduplikasi total dihitung sebagai selisih antara Waktu Total dan Waktu Pemindaian.
4. **Analisis Proyeksi (Skala Besar)**:

- Model regresi untuk **penghematan ruang** (dari data UPR-3) dan **waktu deduplikasi** (dari model terbaik di 3.4.2) digunakan untuk memproyeksikan nilai kedua metrik pada jumlah VM yang lebih besar.
- Metrik efisiensi kemudian dihitung pada setiap titik proyeksi untuk menganalisis skalabilitas kinerja tiap arsitektur. Analisis ini mengidentifikasi **skala optimal** di mana suatu arsitektur memberikan keuntungan terbesar.

3.4.4 Mikroanalisis

Analisis mikro dilakukan terhadap basis data SQLite (*hashfile*) yang dihasilkan oleh `duperemove` untuk memahami karakteristik deduplikasi pada level blok dan *extent*. Data dari tabel `files`, `extents`, dan `blocks` diekstrak untuk setiap skenario (UPR-3, UPR-4, UPR-5) dan kelompok eksperimen.

Teknik analisis yang digunakan meliputi:

5. **Analisis Komposisi dan Distribusi**:
   - **Statistik Deskriptif**: Menghitung total, rata-rata, median, modus, dan simpangan baku untuk parameter seperti jumlah berkas, jumlah *extent*, dan rata-rata panjang *extent*.
   - **Analisis Distribusi**: Distribusi ukuran *extent* divisualisasikan menggunakan *violin plot* untuk membandingkan pola fragmentasi antar kelompok.
6. **Analisis Hubungan dan Korelasi**:
   - Parameter mikro (seperti Total Extent, Jumlah Extent Kembar) dikorelasikan dengan metrik makro (Waktu Eksekusi, Persentase Penghematan) melalui analisis kualitatif dan visualisasi *heatmap*.
   - **Rasio Kembar** dihitung untuk mengukur efektivitas identifikasi duplikasi:

$$Rasio\ Kembar = \frac{Extent\ Kembar}{Total\ Extent} \times 100\,\% \qquad (4)$$

7. **Analisis Deduplikasi Parsial**:
   - Data dari skenario UPR-4 dan UPR-5 dibandingkan dengan UPR-3 untuk mengukur dampak *chunking*.
   - Efektivitas deduplikasi level blok dianalisis dengan membandingkan **Jumlah Blok Kembar** terhadap total blok yang diproses, serta kaitannya dengan peningkatan persentase penghematan ruang.

Melalui kombinasi teknik analisis makro (3.4.1-3.4.3) dan mikro (3.4.4) ini, penelitian ini bertujuan membangun pemahaman yang komprehensif dan multilevel mengenai kinerja sistem.

3.5 Implementasi Eksperimen

Implementasi eksperimen dilakukan dengan mengoperasionalkan rancangan yang telah ditetapkan pada bagian perancangan ke dalam lingkungan nyata. Fokus implementasi adalah pada penciptaan lingkungan yang konsisten dan

terukur, di mana seluruh variabel terkendali kecuali variabel bebas yang sedang diuji. Otomasi menjadi tulang punggung pelaksanaan untuk memastikan bahwa proses provisi VM, simulasi pertumbuhan, deduplikasi, dan pengumpulan data dapat direproduksi dengan hasil yang identik. Penggunaan Ansible memungkinkan eksekusi tugas-tugas tersebut secara paralel dan terstruktur, sementara isolasi sumber daya melalui systemd slice menjamin bahwa pengukuran waktu tidak terpengaruh oleh aktivitas sistem lain. Implementasi juga memastikan bahwa data yang dihasilkan siap untuk dianalisis dengan teknik yang telah ditetapkan, baik data makro (ukuran penyimpanan, waktu eksekusi) maupun data mikro (hashfile duperemove).

3.6 Hasil dan Pembahasan

Pada bagian ini, data yang telah dikumpulkan dari implementasi eksperimen diolah dan disajikan untuk menjawab pertanyaan penelitian. Hasil dianalisis secara bertahap, dimulai dari gambaran umum pencapaian penghematan ruang, dilanjutkan dengan analisis kinerja waktu, dan diakhiri dengan mikroanalisis untuk memahami perilaku sistem pada level blok dan extent. Pembahasan tidak hanya menyajikan angka, tetapi juga menginterpretasikannya dalam konteks teori yang mendasari, seperti mekanisme deduplikasi, fragmentasi data, dan arsitektur penyimpanan. Pola yang muncul dari data—seperti tren penghematan yang melandai dan perbedaan stabilitas kinerja antar kelompok—dijelaskan dengan merujuk pada karakteristik Btrfs dan model penyimpanan. Pembahasan juga mengevaluasi implikasi praktis dari temuan tersebut bagi administrator sistem dalam memilih arsitektur penyimpanan VM.

3.7 Kesimpulan dan Saran

Kesimpulan penelitian dirumuskan dengan menyimpulkan seluruh temuan kunci yang menjawab rumusan masalah. Kesimpulan menyatakan keberhasilan atau kegagalan hipotesis awal, serta mengonfirmasi atau menyangkal temuan dari penelitian terdahulu. Saran dikemukakan dalam dua arah: saran perbaikan untuk penelitian sejenis di masa depan (misalnya, perluasan skala, peningkatan validitas statistik, atau eksplorasi teknologi alternatif), serta saran praktis bagi pengelola infrastruktur TI yang ingin mengadopsi temuan penelitian ini dalam lingkungan produksi. Saran disusun berdasarkan keterbatasan yang dihadapi selama penelitian dan peluang pengembangan yang teridentifikasi dari hasil analisis.

BAB 4 IMPLEMENTASI EKSPERIMEN

Template VM dapat dilihat sebagai lampiran B.1, sedangkan implementasi profil sebagai variabel YAML Ansible terdapat pada lampiran B.2. Bab ini menjelaskan implementasi teknis dari eksperimen yang dirancang pada BAB 3, termasuk konfigurasi sistem, provisi VM, simulasi pertumbuhan data, deduplikasi, serta pengumpulan dan perhitungan data.

4.1 Volume-Volume Penyimpanan

Dua volume Btrfs diformat dengan algoritma hashing `xxhash64`. Perbedaannya terletak pada opsi mount yang digunakan:

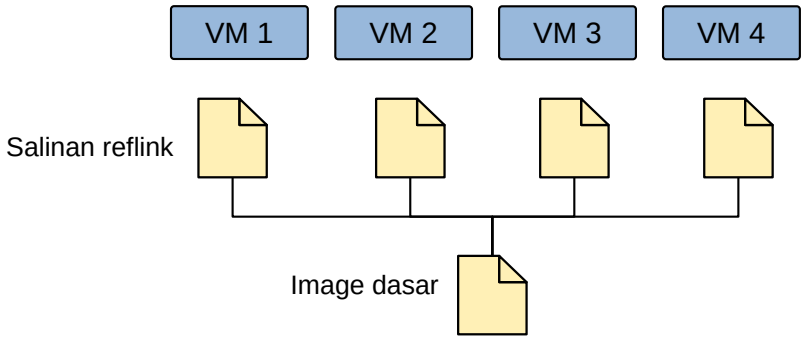
- **Volume** `virt`, dikonfigurasi dengan opsi standar (defaults) tanpa kompresi.
- **Volume** `virt-comp`, dikonfigurasi dengan opsi `compress=force=zstd,defaults` untuk mengaktifkan kompresi dengan algoritma Zstandard.

Opsi `compress=force=zstd` digunakan agar seluruh data dikompres, mengabaikan heuristik internal Btrfs yang mungkin tidak mengompres data yang dianggap tidak kompresibel. Volume ini digunakan untuk menguji efek kombinasi deduplikasi dan kompresi. Volume virt akan menjadi tempat pengujian G-1 dan G-3 sedangkan volume virt-comp menjadi tempat pengujian G-2 dan G-4.

4.2 Citra Dasar

Citra dasar yang berbasis Fedora Server 42 dibuat menjadi 10 VM yang identik. Dua pendekatan berbeda digunakan untuk dua model VM.

4.2.1 VM *Black-box* (Klona Tertaut)

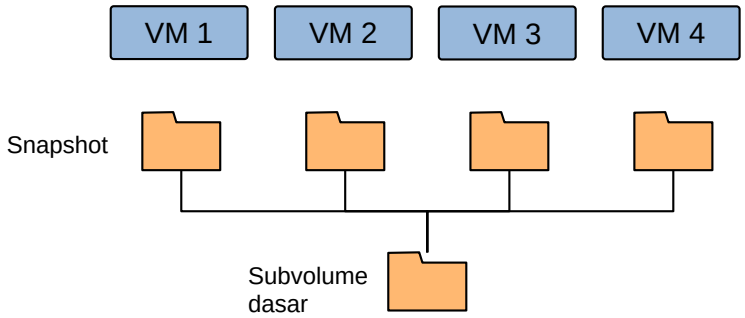


Gambar 4.1 Pembagian citra dasar untuk model *Black-box* menggunakan salinan reflink.

Untuk model ***Black-box***, citra dasar berupa berkas citra diska jarang (`.img`) yang diformat dengan sistem berkas Ext4, penggunaan citra diska jarang berformat *raw* disengaja karena fitur canggih CoW yang disediakan format Qcow2 dari QEMU sudah disediakan oleh Btrfs sebagai sistem berkas host. Setiap instans VM mendapatkan salinan *reflink* (CoW) dari citra dasar ini menggunakan kemampuan Btrfs. Dengan demikian, setiap VM berbagi blok data yang sama

dengan citra dasar hingga terjadi penulisan data baru. Pendekatan ini diilustrasikan pada Gambar 4.1.

4.2.2 VM *White-Box*

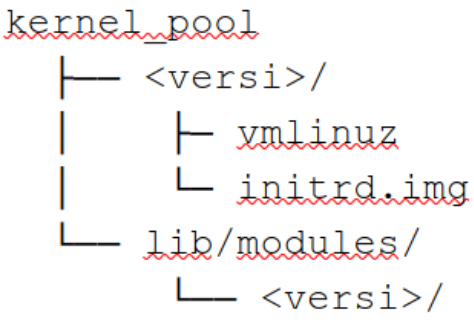


Gambar 4.2 Pembagian citra dasar untuk model *White-box* menggunakan snapshot Btrfs.

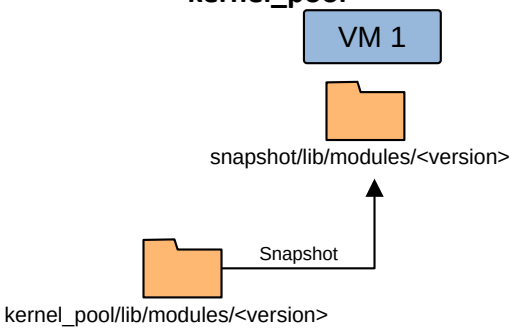
Untuk model ***White-box***, citra dasar tidak disimpan sebagai berkas `.img`, melainkan sebagai **subvolume Btrfs**. Setiap instans VM dibuat sebagai *snapshot* dari subvolume dasar tersebut. Snapshot bersifat ringan karena hanya merekam perubahan terhadap subvolume asal dan dapat dibuat dengan cepat tanpa duplikasi data. Snapshot ini kemudian diekspos ke VM sebagai direktori root melalui **virtio-fs**. Pendekatan ini diilustrasikan pada Gambar 4.2.

4.3 Konfigurasi *booting* VM *White-box*

VM *White-box* menggunakan *QEMU direct linux boot* untuk menghindari ketergantungan pada kernel di dalam guest yang dapat berubah akibat pembaruan. Kernel dikelola sepenuhnya oleh host. Sebuah direktori khusus bernama *kernel_pool* disiapkan di host untuk menyimpan berbagai versi kernel dan modulnya. Strukturnya ditunjukkan pada Gambar 4.3.



Gambar 4.3 Struktur Direktori kernel_pool



Gambar 4.4 Pemasokan Modul-Modul Kernel ke direktori VM *White-box*

Modul kernel dari *kernel_pool/lib/modules/<versi>/* dibuatkan snapshot CoW ke dalam direktori */lib/modules/<versi>* di rootfs VM, seperti diilustrasikan pada Gambar 4.4. Hal ini memastikan kernel dapat menemukan dan memuat modul yang diperlukan saat boot.

4.4 Provisi VM

Seluruh 40 VM (2 model × 2 volume × 10 instans) diprovisi secara otomatis menggunakan playbook Ansible. Proses ini dibagi menjadi empat tahap utama:

4.4.1 Logika Pembangkitan Konfigurasi Dinamis

Playbook menggunakan templat Jinja2 untuk membangun array konfigurasi (`vms_config`) secara dinamis. Logika ini mengiterasi daftar nama VM dan menggabungkan konfigurasi umum dengan parameter spesifik setiap VM. Contoh untuk model *Black-box* (Grup G-1 dan G-3) ditunjukkan pada Kode 1, dan untuk *White-box* (Grup G-2 dan G-4) pada Kode 2.

```
...  ...
139  {% set blockfull_vms = [] %}
140  {% for vm_name in blockfull_vm_names %}
141  {% set _ = blockfull_vms.append({
142      "name": vm_name,
143      "tags": ["blockfull"],
144      "base_image": base_image_ext4,
145      "disks": [
```

```
146         {
147           "path": image_pool ~ '/instances/' ~ vm_name ~ '.img',
148           "boot_order": 1
149         } | combine(common_disk_config)
150       ],
151       "cloudinit_data": { 'hostname': vm_name } | combine(common_cloudinit_data)
152     } | combine(common_config, recursive=true)) %}
153   {% endfor %}
...   ...
```

Kode 1 Cuplikan logika pembangkitan konfigurasi untuk model *Black-box*.

```
...   ...
165   {% set blockless_vms = [] %}
140   {% for vm_name in blockless_vm_names %}
141   {% set _ = blockless_vms.append({
142       "name": vm_name,
143       "fstype": "Virtio-FS",
144       "type": "blockless",
145       "tags": ["blockless"],
146       "base_image":  base_subvol,
147       "filesystems": [
148         {
149           "path": fs_pool ~ '/instances/vm-' ~ (loop.index | format('%02d'))
150         } | combine(common_fs_config)
151       ],
152       "os": {
153         "direct_boot": true,
154         "kmod_src": kernel_pool ~ '/lib/modules/' ~ kernel_ver,
...         # ... (parameter kernel, initrd, cmdline) ...
157       },
158       "cloudinit_data":{ 'hostname': vm_name } | combine( common_cloudinit_data )
159     } | combine(common_config, recursive=true)) %}
160   {% endfor %}
...   ...
```

Kode 2 Cuplikan logika pembangkitan konfigurasi untuk model *White-box*.

Logika yang sama persis diterapkan untuk kategori terkompresi (`_comp`), yang pada akhirnya menghasilkan satu *array* `vms_config` besar berisi 40 konfigurasi VM yang lengkap.

4.4.2 Templat XML VM

Konfigurasi XML lengkap untuk setiap VM dirender dari templat Jinja2 (lihat lampiran B.1). Templat ini membaca array `vms_config` dan menghasilkan elemen yang sesuai. Untuk model *Black-box*, templat menghasilkan elemen `<disk>` seperti pada Kode 3. Untuk model *White-box*, templat menghasilkan elemen `<filesystem>` untuk virtio-fs seperti pada Kode 4.

```
...   ...
69      {% if vm_config.disks is defined %}
70      {% for disk in vm_config.disks %}
```

```
71      <disk type='{{ disk.type }}' device='{{ disk.device | default
        ('disk') }}'>
72        <driver name='qemu' type='{{ disk.driver.type }}'
        io='{{ disk.driver.io | default ('io_uring') }}'
        discard='{{ disk.driver.discard | default ('ignore')}}'/>
73        <source file='{{ disk.path }}'/>
74        <target dev='{{ disk.target.dev }}' bus='{{ disk.target.bus |
        default("virtio") }}' {% if disk.target.emulate_ssd | default
        (false) %}rotation_rate='1'{% endif %}/>
75        {% if disk.boot_order is defined %}
76        <boot order='{{ disk.boot_order }}'/>
77        {% endif %}
78      </disk>
79      {% endfor %}
80      {% endif %}
...     ...
```

Kode 3 Cuplikan templat XML untuk perangkat blok (*Black-box*).

```
...     ...
81      {% if vm_config.filesystems is defined %}
82      {% for fs in vm_config.filesystems %}
83      <filesystem type='{{ fs.type | default ('mount') }}'
        accessmode='{{ fs.accessmode | default ('passthrough') }}'>
84        <driver type='{{ fs.driver.type | default ('virtiofs') }}'
        {% if fs.driver.queue is defined %}queue='{{ fs.driver.queue}}'{%
        endif %}/>
85        <binary path='{{ fs.virtiofsd.binary | default
        ('/usr/libexec/virtiofsd') }}' xattr='{{ fs.virtiofsd.xattr |
        default ('on') }}'/>
86          {# use implied defaults if not defined #}
87          {% if fs.virtiofsd.cache is defined %}
88          <cache mode='{{ fs.virtiofsd.cache }}'/>
89          {% endif %}
90          {% if fs.virtiofsd.sandbox is defined %}
91          <sandbox mode='{{ fs.virtiofsd.sandbox }}'/>
92          {% endif %}
93          {% if fs.virtiofsd.lock is defined %}
94          <lock posix='{{ fs.virtiofsd.lock.posix }}'
        flock='{{ fs.virtiofsd.lock.flock }}'/>
95          {% endif %}
96          {% if fs.virtiofsd.thread_pool is defined %}
97          <thread_pool size='{{ fs.virtiofsd.thread_pool }}'/>
98          {% endif %}
99        <source dir="{{ fs.path }}"/>
100       <target dir='{{ fs.tag }}'/>
101     </filesystem>
102     {% endfor %}
103     {% endif %}
...     ...
```

Kode 4 Cuplikan templat XML untuk virtio-fs (*White-box*).

4.4.3 Alur Playbook Povisi

Playbook provisi dilampirkan pada B.3 dan menjalankan serangkaian tugas berikut:

1. **Persiapan Cloud-Init**: Menghasilkan berkas ISO cloud-init untuk setiap VM.
2. **Pembuatan Diska (*Black-box*)**: Membuat salinan reflink dari citra dasar dengan perintah `cp --reflink=always`.
3. **Pembuatan Snapshot (*White-box*)**: Membuat snapshot Btrfs dari subvolume dasar.
4. **Pembuatan Subvolume Modul Kernel (*White-box*)**: Snapshot modul kernel dari `kernel_pool`.
5. **Definisi VM**: Mendaftarkan VM ke libvirt menggunakan konfigurasi XML yang telah dirender.

4.5 Implementasi Simulasi Pertumbuhan

Simulasi pertumbuhan data di dalam VM diimplementasikan dengan playbook Ansible yang dijalankan selama 7 hari simulasi di mana satu hari adalah satu sebuah himpunan aksi. Playbook utama (`simulate_growth.yml`) menargetkan semua VM dan melakukan loop untuk setiap hari, memanggil tugas-tugas pertumbuhan.

4.5.1 Penentuan Profil Pertumbuhan

Setiap VM dipetakan ke profil pertumbuhan tertentu (A sampai J) berdasarkan suffix namanya (misal, `-01` untuk profil A). Profil ini mendefinisikan jadwal aksi harian yang mensimulasikan penggunaan nyata seperti instalasi paket, kloning repositori, kompilasi, dan penulisan log. Profil-profil ini ditentukan oleh kode pada lampiran B.2.

4.5.2 Playbook Pertumbuhan VM

```
1  - name: Play 1 - Simulate Growth on Guest VMs
2    hosts: vms
3    become: true
4
5    tasks:
6      - name: SIMULATING DAY {{ item }}
7        ansible.builtin.include_tasks:
8          file: include/guest_run.yml
9        loop: "{{ range(1, 8) | list }}"
10       loop_control:
11         loop_var: day_number
```

Kode 5 Playbook Utama Simulasi Pertumbuhan

```
1  - name: "DAY {{ day_number }} | Determine and apply growth profile"
2    vars:
```

```
    # Ekstrak sufiks dua digit dari hostname (misal, 'blockless-vm-01' -> '01')
    vm_suffix: "{{ ansible_hostname.split('-')[-1] }}"
    # Cari kunci profil (A, B, C...) dari peta
    profile_key: "{{ profile_map[vm_suffix] }}"
    # Dapatkan data profil aktual menggunakan kunci
    profile: "{{ growth_profiles[profile_key] }}"
  ansible.builtin.include_tasks: guest_actions.yml
```

Kode 6 Berkas Logika Pemetaan Profil

```
---
# Langkah 1: Buat daftar datar (flat list) dari semua aksi yang dijadwalkan untuk hari ini.
- name: "Determine actions to run on Day {{ day_number }}"
  ansible.builtin.set_fact:
    todays_actions: "{{ todays_actions | default([]) + item.actions }}"
  loop: "{{ profile.daily_schedule }}"
  loop_control:
    loop_var: item
  # Kondisi 'when' ini memfilter jadwal, sehingga hanya aksi untuk hari ini yang ditambahkan.
  when:
    (
      '-' in item.days and
      day_number in range(item.days.split('-')[0] | int, (item.days.split('-')[1] | int) + 1)
    )
    or
    (
      '-' not in item.days and
      day_number == (item.days | int)
    )

# Langkah 2: Jalankan loop sederhana pada daftar datar yang baru dibuat.
- name: "Execute actions for Day {{ day_number }}"
  ansible.builtin.include_tasks: perform_single_action.yml
  loop: "{{ todays_actions | default([]) }}"
  loop_control:
    loop_var: action_item
    label: "{{ action_item.op }}"
```

Kode 7 Berkas Logika Pemfilteran Aksi Harian

```
---
- name: "Execute Action: {{ action_item.op }}"
  vars:
    # Variabel ini secara cerdas menemukan path/dest dari item aksi.
    target_path: "{{ action_item.path | default(action_item.dest | default('')) }}"
  block:
    - name: "Ensure parent directory exists for {{ target_path }}"
```

```
      when: target_path | length > 0
      ansible.builtin.file:
        path: "{{ target_path | dirname }}"
        state: directory
        mode: '0755'

    - name: "OP: Install Package(s)"
      when: action_item.op == 'install_package'
      ansible.builtin.dnf:
        name: "{{ action_item.name }}"
        state: present

    - name: "OP: Git Clone - {{ action_item.repo }}"
      when: action_item.op == 'git_clone'
      ansible.builtin.git:
        repo: "{{ action_item.repo }}"
        dest: "{{ target_path }}"

    - name: "OP: Build Java Project with Maven"
      when: action_item.op == 'build_java_project'
      ansible.builtin.command:
        cmd: "mvn package -DskipTests"
        chdir: "{{ target_path }}"
      changed_when: true

    - name: "OP: Simulate Log File Growth"
      when: action_item.op == 'log_simulation'
      ansible.builtin.command: >
        fio --name=log_write --directory={{ target_path | dirname }}
        --filename={{ target_path }} --ioengine=libaio --rw=write
        --bs=4k --size={{ action_item.size }}
      changed_when: true

    - name: "OP: Simulate User Uploads"
      when: action_item.op == 'user_uploads'
      ansible.builtin.command: >
        fio --name=uploads --directory={{ target_path }}
        --ioengine=libaio --rw=write --bs=1M
        --size={{ action_item.total_size }} --
nrfiles={{ action_item.num_files }}
      changed_when: true

    - name: "OP: Generate Random Data from /dev/urandom"
      when: action_item.op == 'random_data'
      ansible.builtin.command:
        cmd: "dd if=/dev/urandom of={{ target_path }} bs=1M count={{
action_item.size.split('M')[0] }}"
        creates: "{{ target_path }}"
      changed_when: true

    - name: "OP: Update All Packages (Excluding Kernel)"
      when: action_item.op == 'dnf_update'
      ansible.builtin.dnf:
```

```
59          name: '*'
60          state: latest
61          exclude: ['kernel*', 'initramfs*', 'grub2*']
```

Kode 8 Berkas Eksekusi Aksi Pertumbuhan

Untuk setiap hari (setiap iterasi *loop*), *playbook* `grow_vms.yml` (Kode 5) memanggil `include/guest_run.yml` dan meneruskan nomor hari saat ini sebagai variabel `day_number`. Berkas `guest_run.yml` (Kode 6) bertanggung jawab untuk memetakan setiap VM ke profil pertumbuhan yang sesuai, sesuai dengan rancangan pengujian. Logika ini mengambil sufiks dari nama *host* VM (misal, `01`), menggunakannya untuk mencari kunci profil (misal, `A`) dari variabel `profile_map`, dan kemudian mengambil data profil lengkap (jadwal dan aksi) dari variabel `growth_profiles`. Setelah profil ditentukan, *playbook* memanggil `guest_actions.yml`. Berkas `guest_actions.yml` (Kode 6) kemudian memfilter jadwal di dalam profil untuk menemukan aksi-aksi yang dijadwalkan untuk `day_number` saat ini, dan mengeksekusinya satu per satu. Berkas terakhir dalam rantai ini adalah `perform_single_action.yml`(Kode 8), yang berisi implementasi teknis dari setiap operasi yang menghasilkan pertumbuhan data. Berkas ini dipanggil berulang kali untuk setiap aksi yang dijadwalkan. Seperti yang terlihat pada Kode 8, *playbook* ini mengimplementasikan berbagai skenario penggunaan nyata, termasuk instalasi perangkat lunak (DNF), kompilasi kode (`mvn`), penulisan *log* (FIO), unggahan berkas (FIO), dan pembaruan sistem. Operasi-operasi inilah yang secara kolektif menghasilkan pertumbuhan data unik dan duplikat pada setiap VM selama 7 hari simulasi.

4.6 TRIM Citra VM *Black-box*

Karena citra VM *Black-box* berupa citra diska jarang, blok yang dihapus di dalam guest OS tidak otomatis dikembalikan ke host. Untuk memastikan perbandingan yang adil dengan model *White-box* (di mana penghapusan langsung membebaskan ruang di host), perintah `fstrim` dijalankan di dalam setiap guest VM *Black-box* setelah simulasi selesai. Konfigurasi `discard='unmap'` pada disk XML memastikan perintah TRIM diteruskan ke Btrfs di host, sehingga blok yang tidak terpakai dilepaskan dari citra diska jarang.

4.7 Jalankan Duperemove

Proses deduplikasi aktif dilakukan menggunakan `duperemove`. Untuk menguji efek parameter, tiga set salinan CoW dari 10 instans VM (baik *Black-box* maupun *White-box*) dibuat. `duperemove` dijalankan dengan parameter berbeda pada setiap set:

- **default**: tanpa parameter tambahan (pengaturan bawaan).
- **128k-partial**: `-b 128k --dedupe-options partial`.
- **64k-partial**: `-b 64k --dedupe-options partial`.

Setiap proses dijalankan dalam `systemd slice` yang diisolasi (CPU dan IO) untuk stabilitas pengujian.

4.7.1 Isolasi kerjaan

Untuk memastikan konsistensi dan stabilitas pengujian, proses deduplikasi diisolasi dalam sebuah **systemd slice** khusus. Isolasi ini bertujuan untuk:

1. **Minimalkan Interferensi**: Mencegah proses lain di sistem host mengganggu proses `duperemove` dengan membatasi penggunaan CPU dan I/O hanya pada core dan thread yang ditentukan.
2. **Konsistensi Pengukuran**: Memastikan pengukuran waktu eksekusi tidak terpengaruh oleh fluktuasi beban sistem yang tidak terkontrol.
3. **Prioritasi Sumber Daya**: Memberikan prioritas tinggi pada proses `duperemove` dalam slice tersebut untuk mengurangi latency I/O dan CPU.

Konfigurasi slice didefinisikan dalam berkas /etc/systemd/system/kinerjance.slice dengan konten seperti pada Kode 9.

```
1  [Unit]
2  Description=Slice for high-kinerjance, CPU-isolated applications
3  Before=slices.target
4
5  [Slice]
6  AllowedCPUs=0-3,8-11
7  CPUWeight=500
8  IOWeight=500
```

Kode 9 kinerjance.slice

Penjelasan konfigurasi:

- **AllowedCPUs=0-3,8-11**: Proses dalam slice hanya diizinkan berjalan pada 4 core fisik (0-3) dan 4 thread hyper-threading yang sesuai (8-11), yaitu setengah dari total core sistem (dalam konfigurasi CPU 8 core/16 thread). Hal ini memastikan proses deduplikasi tidak menggunakan seluruh kapasitas CPU.
- **CPUWeight=500** dan **IOWeight=500**: Memberikan bobot prioritas tinggi (dalam skala 1–1000) untuk akses CPU dan I/O, sehingga proses dalam slice ini mendapat jatah sumber daya yang besar dibandingkan slice lainnya.

Untuk menjalankan `duperemove` dalam slice yang telah dikonfigurasi, digunakan perintah `systemd-run` dengan argumen `--slice=kinerjance.slice`. Dengan demikian, setiap proses deduplikasi berjalan dalam lingkungan yang terisolasi dan terkontrol, sehingga variabel pengganggu eksternal dapat diminimalkan dan hasil pengukuran waktu eksekusi menjadi lebih andal untuk analisis perbandingan.

4.8 Pengumpulan Data

Pengumpulan data dilakukan secara iteratif setelah penambahan setiap instans VM (dari 1 hingga 10) untuk setiap skenario. Alat `compsize` (Borowski,

2025) digunakan untuk menganalisis penggunaan ruang Btrfs. Tiga kolom utama dari output `comsize` dimanfaatkan:

- **Disk Usage**: Ukuran data aktual di disk setelah kompresi dan deduplikasi.
- **Uncompressed**: Ukuran data setelah deduplikasi tetapi sebelum kompresi (sama dengan Disk Usage pada volume tanpa kompresi).
- **Referenced**: Ukuran data logis total seolah-olah tidak ada deduplikasi atau CoW (baseline untuk klona penuh).

Data untuk skenario UPR-1 (klona penuh tanpa efisiensi) diambil dari kolom `Referenced` pada volume tanpa kompresi Grup G-1 dan G-3 dan dari kolom `Disk Usage` pada salinan non-CoW VM dari Grup G-2 dan G-4 . Data untuk skenario UPR-2 hingga UPR-5 diambil dari kolom `Disk Usage` pada volume yang sesuai. Data ukuran-ukuran VM disajikan dalam lampiran sebagai A.1, data waktu eksekusi sebagai Lampiran A.2 dan A.3 serta data rangkuman hashfile yang dihasilkan duperemove sebagai A.4.

BAB 5 HASIL DAN PEMBAHASAN

5.1 Deskripsi Hasil Eksperimen

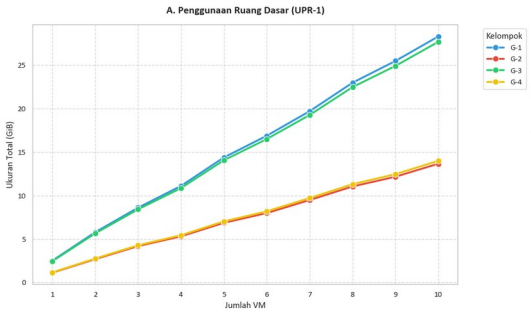
Bab ini menyajikan hasil eksperimen dan pembahasan komprehensif mengenai efektivitas teknik penghematan ruang penyimpanan pada VM antara model penyimpanan *white-box* dan *black-box* menggunakan sistem berkas Btrfs. Penelitian ini menguji dua aspek utama: (1) penghematan ruang disk yang dicapai melalui kombinasi fitur Copy-on-Write (CoW), deduplikasi, dan kompresi, dan (2) kinerja waktu yang dibutuhkan untuk proses deduplikasi. Seluruh data yang digunakan dalam analisis ini bersumber dari dataset-dataset yang dihasilkan selama penelitian yang dilampirkan dalam LAMPIRAN A.

Secara keseluruhan, eksperimen mengonfirmasi bahwa pendekatan *white-box* (yang memanfaatkan Virtio-FS untuk mengekspos struktur berkas *guest* ke *host*) secara signifikan lebih unggul dalam hal efisiensi ruang penyimpanan dan kecepatan deduplikasi dibandingkan pendekatan *black-box* tradisional (yang memperlakukan citra VM sebagai blok data buram). Namun, temuan eksperimen juga mengungkap nuansa penting terkait peran kompresi, pola fragmentasi data, dan skalabilitas masing-masing pendekatan.

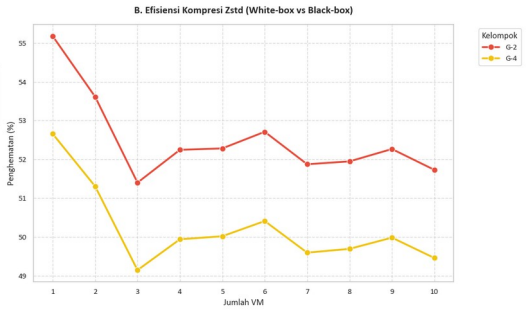
5.2 Analisis Penghematan Ruang Penyimpanan

Bagian ini memaparkan hasil analisis mendalam mengenai efektivitas berbagai teknik reduksi data—meliputi kompresi, Copy-on-Write (CoW), dan deduplikasi terhadap efisiensi ruang penyimpanan pada sistem berkas Btrfs. Evaluasi dilakukan dengan membandingkan dua model penyimpanan VM.

5.2.1 Pengaruh Kompresi



Gambar 5.1 Skenario UPR-1 pada grup G1, G2, G3 dan G4



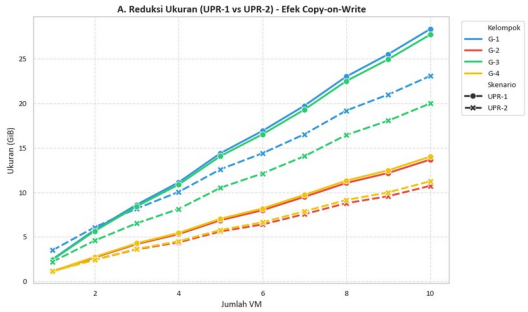
Gambar 5.2 Persentase penghematan dari UPR-1 pada grup G2 dan G4 terhadap grup G1 dan G3 masing-masing

Sebelum mengevaluasi mekanisme berbagi data antar-VM, analisis awal dilakukan untuk mengukur efektivitas kompresi transparan Zstd level 3 terhadap Ukuran Dasar (UPR-1). Perbandingan ini dilakukan antara skenario UPR-1 pada G-1 dan G-3 melawan G-2 dan G-4, sebagaimana diilustrasikan dalam Gambar 5.1 dan Gambar 5.2.

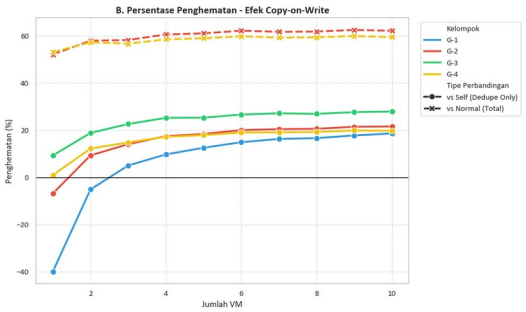
Berdasarkan data pengukuran, penerapan kompresi menunjukkan dampak reduksi ukuran yang signifikan pada kedua model penyimpanan. Pada konfigurasi satu VM (1 VM), **G-2** mampu mereduksi ukuran citra dari 2,50 GiB (pada **G-1**) menjadi 1,12 GiB, yang setara dengan penghematan sebesar 55,18%. Seiring bertambahnya jumlah VM menjadi sepuluh, tingkat penghematan sedikit menurun namun tetap stabil di angka 51,73%. Tingginya efisiensi pada **G-2** ini menunjukkan bahwa algoritma Zstd bekerja sangat efektif pada file citra monolitik `.img`, di mana blok data tersusun secara sekuensial, memungkinkan algoritma untuk menemukan pola redundansi lokal dengan lebih baik.

Sementara itu pada **G-4**, kompresi mereduksi ukuran penyimpanan 1 VM dari 2,44 GiB (pada **G-3**) menjadi 1,16 GiB, mencatatkan penghematan sebesar 52,66%. Pada skala sepuluh VM, penghematan tercatat sebesar 49,46%. Meskipun angka ini tergolong tinggi, efisiensi **G-4** sedikit lebih rendah dibandingkan **G-2**. Hal ini disebabkan oleh karakteristik *white-box* yang menyimpan data sebagai ribuan berkas kecil terpisah. Kompresi pada file-file kecil seringkali kurang optimal karena adanya *overhead* metadata per file dan pemecahan data yang membatasi jendela pencarian pola kompresi. Secara keseluruhan, **G-2** memiliki keunggulan tipis dalam efisiensi kompresi murni karena struktur datanya yang lebih kontinu.

5.2.2 Pengaruh UPR-2



Gambar 5.3 Ukuran VM UPR-1 Dibanding UPR-2



Gambar 5.4 Persentase Penghematan UPR-2 Terhadap UPR-1

Sub-bab ini mengevaluasi efektivitas mekanisme Copy-on-Write (CoW) saat VM diduplikasi (kloning). Analisis membandingkan UPR-1 dengan UPR-2 (CoW via Reflink/Snapshot). Hasil pengukuran penggunaan ruang dan persentase penghematan divisualisasikan pada Gambar 5.3 dan Gambar 5.4 secara berturut.

Pada volume normal (**G-1** dan **G-3**), skenario UPR-2 menunjukkan dinamika yang berbeda drastis. Pada konfigurasi awal 1 VM, **G-1** mengalami pembengkakan ukuran (*overhead*) yang signifikan, dengan penghematan negatif sebesar -39,99% (ukuran aktual 3,50 GiB berbanding 2,50 GiB pada UPR-1). Hal ini disebabkan oleh perilaku alokator Btrfs saat menangani operasi penulisan acak (*random write*) dan perintah discard/trim dari guest OS pada file sparse hasil reflink. Ketidakselarasan (*misalignment*) antara blok data baru dan extent lama menciptakan *slack space* (ruang sela) yang tidak terpakai namun tetap teralokasi. Sebaliknya, **G-3** masih mencatat efisiensi positif sebesar 9,24%.

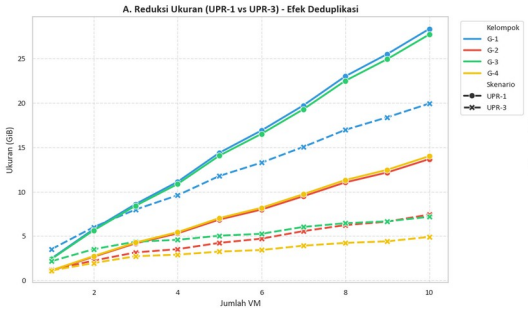
Namun, tren inefisiensi awal pada **G-1** ini berangsur membaik seiring bertambahnya jumlah VM. Pada skala sepuluh VM, **G-3** terbukti jauh lebih efisien dalam memanfaatkan fitur CoW, mencapai penghematan 27,90%. **G-1** tertinggal dengan penghematan 18,59%. Data ini mengonfirmasi bahwa *white-box* lebih superior dalam mekanisme CoW karena Btrfs dapat mengelola pembagian data (*sharing*) secara granular pada level berkas, menghindari fragmentasi internal yang terjadi pada file citra besar.

Grafik persentase penghematan juga memperlihatkan fenomena kelandaian (*plateau*) yang jelas. Laju peningkatan penghematan melambat secara asimtotik seiring bertambahnya jumlah VM. Sebagai contoh, pada **G-3**, kenaikan penghematan drastis terjadi saat transisi dari 1 ke 2 VM (naik ~9,6%), namun melambat signifikan saat transisi dari 9 ke 10 VM (hanya naik ~0,3%). Kelandaian ini membuktikan bahwa keuntungan utama dari mekanisme CoW berasal dari pembagian citra dasar (*base image*) yang statis.

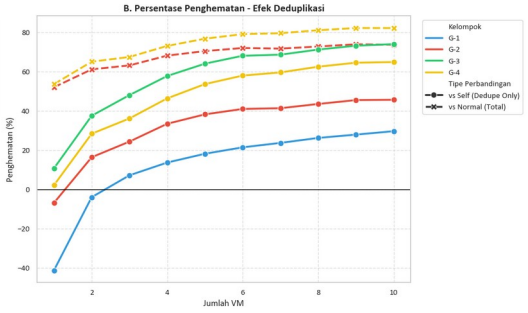
Ketika dikombinasikan dengan kompresi (**G-2** dan **G-4**), total penghematan (dibandingkan terhadap UPR-1 volume normal) menjadi sangat besar. Pada skala sepuluh VM, **G-2** mencatat total penghematan 62,13%, sedikit mengungguli **G-4** yang mencapai 59,42%. Keunggulan **G-2** di sini semata-mata didorong oleh efisiensi kompresi dasarnya yang lebih baik.

Namun, jika dilihat dari efisiensi mekanisme CoW itu sendiri (dengan membandingkan UPR-2 terkompresi terhadap UPR-1 terkompresi), terlihat bahwa dampak CoW menurun. Penghematan murni CoW pada **G-2** hanya sebesar 21,55% dan **G-4** sebesar 19,71%. Angka ini lebih rendah dibandingkan penghematan CoW di volume normal (18,59% pada **G-1** dan 27,90% pada **G-3**). Penurunan efisiensi relatif ini terjadi karena proses kompresi telah menghilangkan sebagian besar redundansi data di awal. Selisih efisiensi antara kedua model juga menyempit pada volume terkompresi (~1,8%) dibandingkan volume normal (~9,3%), menunjukkan bahwa kompresi bertindak sebagai *equalizer* yang menyamarkan kelemahan struktural *black-box*.

5.2.3 Pengaruh UPR-3



Gambar 5.5 Ukuran VM UPR-1 dibanding UPR-3



Gambar 5.6 Persentase Penghematan UPR-3 Terhadap UPR-1

Sub-bab ini menganalisis dampak deduplikasi *out-of-band* menggunakan `duperemove` (skenario UPR-3) terhadap ukuran dasar (UPR-1), sebagaimana ditunjukkan pada Gambar 5.6 dan Gambar 5.5 yang menunjukkan ukuran dan

persentase. Hasil eksperimen menunjukkan divergensi hasil yang ekstrem, menegaskan peran krusial arsitektur penyimpanan.

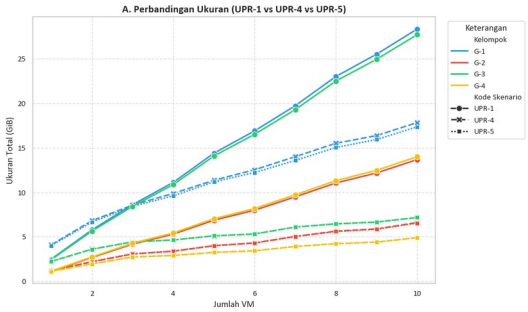
Pada **G-3** dengan sepuluh VM, skenario UPR-3 berhasil menekan penggunaan ruang secara drastis hingga mencapai penghematan 74,10% (ukuran sisa 7,17 GiB dari Ukuran Dasar 27,69 GiB). Tingginya angka ini membuktikan bahwa deduplikasi level berkas (*file-level*) sangat efektif; alat deduplikasi dapat dengan mudah mengidentifikasi file sistem operasi yang identik antar-VM. Sebaliknya, **G-1** hanya mencapai penghematan 29,66%. Rendahnya efisiensi ini disebabkan oleh masalah *alignment*. Perubahan kecil di dalam sistem berkas guest dapat menggeser posisi data di dalam file citra `.img`, menyebabkan blok data 128 KiB menjadi berbeda secara biner (*checksum mismatch*) meskipun isinya secara logis sama. Selisih penghematan yang masif sebesar 44,4% antara **G-3** dan **G-1** ini menjadi bukti kuat keunggulan arsitektur *white-box* dalam mendukung deduplikasi.

Sinergi terbaik terlihat pada kombinasi *white-box*, kompresi, dan deduplikasi. Skenario UPR-3 pada **G-4** mencapai total penghematan tertinggi dalam penelitian ini, yaitu 82,28% (hanya menggunakan 4,91 GiB untuk 10 VM). Sementara itu, **G-2** tertahan di angka 73,78%, menunjukkan selisih ~21% lebih rendah.

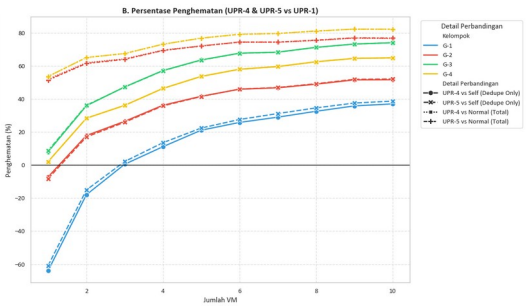
Penting untuk mencatat perbandingan terhadap basis yang setara. Jika dibandingkan terhadap UPR-1 terkompresi (bukan normal), deduplikasi pada **G-4** memberikan penghematan sebesar 64,93%, sedangkan pada **G-2** sebesar 45,68%. Meskipun selisih efisiensi mengecil menjadi ~19% (dibandingkan ~44% pada volume normal), keunggulan **G-4** tetap signifikan. Fenomena kelandaian juga terlihat jelas di sini: pada **G-4**, kurva penghematan deduplikasi mencapai titik jenuh lebih cepat (mulai mendatar di 5-6 VM), sementara pada **G-2** kurvanya masih sedikit menanjak, mengindikasikan bahwa proses deduplikasi pada **G-2** masih berjuang menemukan pola redundansi yang lebih sulit diidentifikasi.

Sebuah temuan menarik terlihat pada kedekatan nilai penghematan antara skenario lintas-kelompok. Total penghematan UPR-3 pada **G-2** (73,78%) tercatat hampir setara dengan UPR-3 pada **G-3** (74,10%). Kedekatan nilai ini (selisih < 0,4%) mengimplikasikan bahwa arsitektur *black-box* dengan lapisan kompresi dapat menyamai efisiensi yang dapat dicapai oleh *white-box* tanpa kompresi.

5.2.4 Pengaruh UPR-4 dan UPR-5



Gambar 5.7 Ukuran VM UPR-1 dibanding UPR-4 dan UPR-5



Gambar 5.8 Persentase Penghematan UPR-4 dan UPR-5 Terhadap UPR-1

Sub-bab ini mengevaluasi apakah teknik pemindaian parsial (memecah *extent* menjadi *chunk* lebih kecil) dapat meningkatkan efisiensi, dengan membandingkan skenario deduplikasi parsial 128 KiB (UPR-4) dan 64 KiB (UPR-5). Tren data ditunjukkan pada Gambar 5.7 dan Gambar 5.8.

Teknik ini terbukti sangat bermanfaat bagi penyelamatan efisiensi **G-1**. Pada volume normal (10 VM), penghematan meningkat dari 29,66% (UPR-3) menjadi 37,06% pada skenario UPR-4 dan 38,75% pada skenario UPR-5. Peningkatan ini mengonfirmasi hipotesis bahwa banyak data redundan pada **G-1** yang "tersembunyi" di dalam blok yang tidak selaras atau terfragmentasi. Meskipun demikian, terlihat adanya hukum hasil yang semakin berkurang (*diminishing returns*): transisi dari metode standar ke parsial 128 KiB memberikan lonjakan penghematan besar (~7,4%), namun transisi lebih lanjut ke 64 KiB hanya memberikan tambahan marginal (~1,7%).

Sebaliknya, pada **G-3**, teknik ini tidak memberikan dampak tambahan apa pun. Penghematan pada skenario UPR-4 dan UPR-5 identik dengan UPR-3, yaitu stagnan di angka 74,10%. Hal ini menunjukkan bahwa redundansi pada *white-box* bersifat struktural (berbasis file utuh). Memecah file yang sudah identik menjadi potongan-potongan kecil tidak menambah nilai penghematan, melainkan hanya menambah beban komputasi tanpa hasil yang berarti.

Pada volume terkompresi, pola yang sama berulang. Jika dibandingkan terhadap UPR-1 terkompresi, deduplikasi parsial pada **G-2** mampu meningkatkan efisiensi dari 45,68% (UPR-3) menjadi 52,11% (UPR-5). Peningkatan ini memperkecil selisih ketertinggalan terhadap **G-4** menjadi sekitar ~12%. Namun, upaya ekstra ini tetap tidak mampu menandingi efisiensi inheren dari **G-4** yang mencapai saturasi penghematan maksimal tanpa memerlukan teknik parsial yang kompleks.

Deduplikasi parsial UPR-4 pada **G-2** mampu meningkatkan efisiensi total menjadi 76,67%. Angka ini sedikit melampaui efisiensi UPR-3 pada **G-3** (74,10%), namun dicapai dengan biaya komputasi yang jauh lebih tinggi (kompresi + *chunking*). Peningkatan efisiensi dari UPR-4 ke UPR-5 pada **G-2** juga sangat tipis (76,67% ke 76,88%), memperkuat bukti terjadinya saturasi efisiensi di mana

upaya pemindaian yang lebih agresif tidak lagi sebanding dengan penghematan ruang yang diperoleh.

5.2.5 Tren dan Perbandingan Menyeluruh

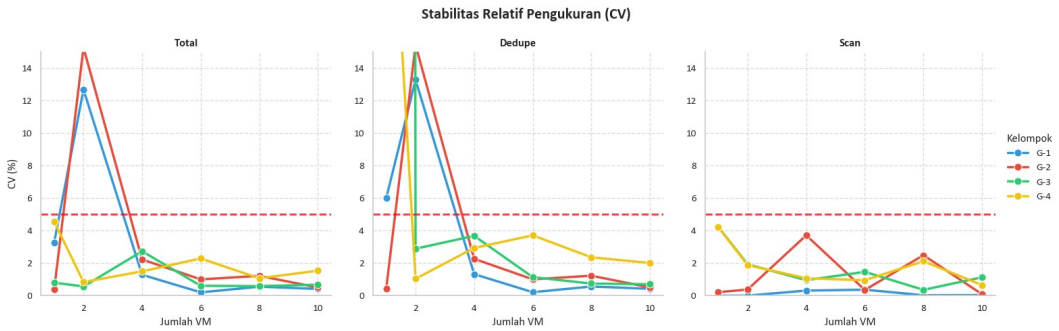
Secara keseluruhan, apabila diperhatikan, mayoritas kurva persentase penghematan pada seluruh kelompok eksperimen (**G-1** hingga **G-4**) menunjukkan pola logaritmik yang melambat setelah mencapai jumlah VM tertentu. Tren ini mencerminkan karakteristik profil pertumbuhan VM yang digunakan dalam penelitian. Meskipun setiap VM mengalami pertumbuhan data melalui penulisan baru (*overwrite*) yang memecah tautan CoW pada skenario UPR-2, data baru tersebut—berkat homogenitas sistem operasi dan kemiripan beban kerja (*workload similarity*)—sebenarnya identik di seluruh VM. Oleh karena itu, lonjakan efisiensi pada UPR-3 menegaskan bahwa deduplikasi berhasil "menangkap kembali" redundansi baru yang hilang oleh mekanisme CoW, hingga akhirnya efisiensi mencapai titik jenuh (*plateau*) ketika mayoritas data kembar telah teridentifikasi.

Dalam hal hierarki efisiensi akhir, **G-4** menempati posisi teratas sebagai arsitektur paling efisien, diikuti dengan sangat ketat oleh **G-2** yang memanfaatkan deduplikasi parsial. Hal ini menarik karena menunjukkan bahwa kelemahan struktural *black-box* dapat dikompensasi secara efektif melalui kombinasi kompresi dan deduplikasi agresif, hingga nyaris menyamai efisiensi *white-box*. Sebaliknya, **G-1** menempati posisi terendah dengan selisih yang signifikan, membuktikan bahwa tanpa bantuan kompresi atau *chunking*, penyimpanan *black-box* sangat rentan terhadap inefisiensi akibat isu *alignment* dan fragmentasi.

5.3 Analisis Kinerja waktu

Analisis kinerja waktu bertujuan untuk membandingkan efisiensi proses deduplikasi antara model penyimpanan *black-box* dan *white-box* dalam berbagai konfigurasi volume Btrfs. Parameter yang diukur mencakup waktu total eksekusi, waktu pemindaian, waktu deduplikasi, serta stabilitas kinerja sistem seiring peningkatan jumlah VM.

5.3.1 Stabilitas Kinerja



Gambar 5.9 CV Data Waktu

Stabilitas kinerja sistem dievaluasi dengan menghitung Koefisien Variasi (CV) dari waktu eksekusi yang diukur dalam tiga kali pengulangan (*triplo*) untuk setiap konfigurasi. CV mengukur dispersi relatif data di sekitar rata-rata, di mana nilai rendah menunjukkan konsistensi yang tinggi, sedangkan nilai tinggi mengindikasikan variabilitas atau ketidakstabilan kinerja.

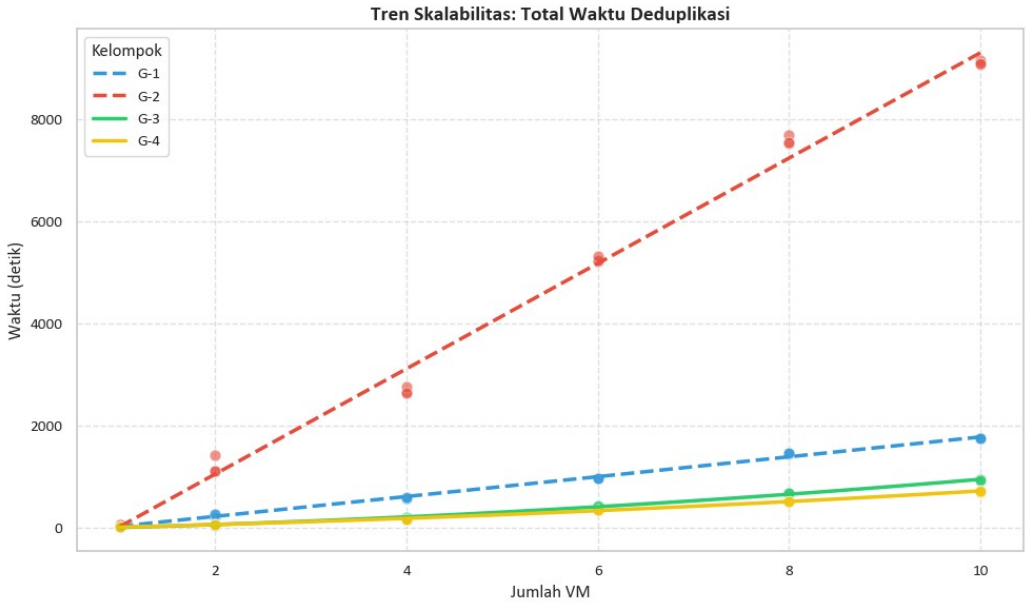
Berdasarkan data yang divisualisasikan pada Gambar 5.9, dapat ditarik beberapa temuan kunci mengenai pola stabilitas keempat kelompok eksperimen (G-1 hingga G-4). Secara umum, data waktu eksekusi yang diperoleh menunjukkan stabilitas yang memadai di seluruh rentang pengujian, dengan sebagian besar nilai CV berada di bawah 5%.

Namun, terdapat perbedaan perilaku yang signifikan antara kelompok *black-box* (G-1 dan G-2) dan *white-box* (G-3 dan G-4), terutama pada fase transisi dari beban kerja tunggal ke paralel (dari 1 VM ke 2 VM). Kelompok G-1 (*Black-box Normal*) dan G-2 (*Black-box Terkompresi*) mengalami lonjakan nilai CV yang cukup tajam pada konfigurasi 2 VM, masing-masing mencapai >12% dan >15%. Lonjakan ini mengindikasikan adanya ketidakstabilan atau *jitter* saat sistem *black-box* menginisiasi proses deduplikasi paralel untuk pertama kalinya. Fenomena ini diduga terkait dengan overhead tambahan dalam mengelola beberapa *citra diska jarang* citra disk secara bersamaan, termasuk potensi *contention* pada I/O dan manajemen metadata Btrfs.

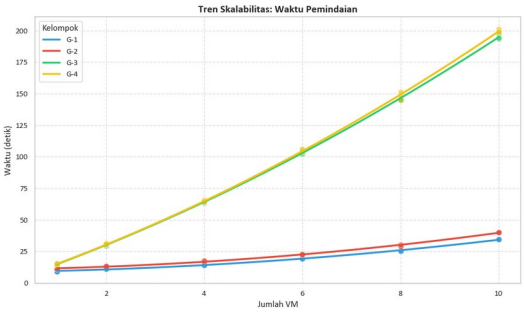
Sebaliknya, kelompok **G-3** dan **G-4** menunjukkan karakteristik kinerja yang jauh lebih stabil di seluruh rentang pengujian. Bahkan pada fase transisi kritis 1 VM ke 2 VM, kenaikan CV pada kedua kelompok ini relatif kecil dan secara konsisten jauh lebih rendah dibandingkan kelompok *black-box*. Stabilitas tinggi ini berlanjut hingga beban puncak (10 VM), di mana G-3 dan G-4 mencatat variabilitas waktu yang minimal. Pola ini membuktikan bahwa arsitektur *white-box* memiliki mekanisme manajemen I/O dan akses data yang lebih deterministik dan *robust*, berkat visibilitas penuh terhadap struktur berkas *guest* yang memungkinkan penjadwalan dan caching yang lebih efisien, baik dalam kondisi normal maupun saat menangani kompleksitas data terkompresi.

Perlu dicatat bahwa analisis stabilitas ini didasarkan pada tiga kali pengulangan percobaan ($n=3$). Meskipun jumlah sampel ini memadai untuk mengidentifikasi pola umum dan perbandingan relatif antar kelompok, nilai CV yang dihasilkan mungkin belum sepenuhnya menggambarkan variabilitas jangka panjang atau menangkap *outlier* yang sangat jarang terjadi. Oleh karena itu, interpretasi data CV lebih tepat sebagai indikator kecenderungan relatif stabilitas antar arsitektur, bukan sebagai ukuran presisi statistik absolut.

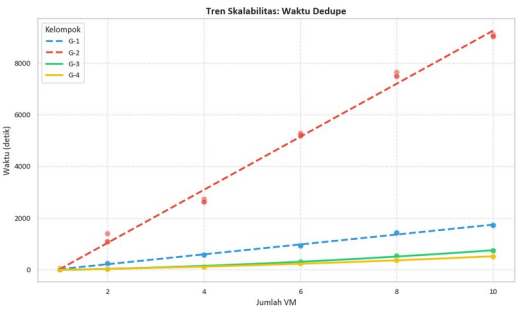
5.3.2 Tren Waktu Eksekusi dan Analisis Model Prediktif



Gambar 5.10 Tren Waktu Total



Gambar 5.11 Tren Waktu Pemindaian



Gambar 5.12 Tren Waktu Deduplikasi (Total - Pemindaian)

Untuk memahami skalabilitas proses deduplikasi, dianalisis tren peningkatan waktu eksekusi seiring pertambahan jumlah VM (1, 2, 4, 6, 8, 10). Analisis dilakukan terhadap tiga komponen waktu: Total Waktu, Waktu Pemindaian, dan Waktu Deduplikasi (diperoleh dari pengurangan Total Waktu dengan Waktu Pemindaian).

Gambar 5.10, 5.11 dan 5.12 masing-masing menggambarkan tren untuk ketiga komponen tersebut. Terlihat jelas perbedaan drastis dalam skala waktu antara kelompok *black-box* (G-1, G-2) dan *white-box* (G-3, G-4). Waktu eksekusi untuk kelompok *black-box* secara konsisten satu hingga dua orde magnitudo lebih lama dibandingkan *white-box*, dengan G-2 (*Black-box Terkompresi*) menjadi yang paling lambat.

Untuk memodelkan hubungan antara jumlah VM (x) dan waktu eksekusi (y), serta memperkirakan kompleksitas waktu (*time complexity*) dari proses deduplikasi, dilakukan *fitting* data dengan dua model: Linear ($y = ax + b$) dan Polinomial Kuadratik ($y = ax^2 + bx + c$). Kualitas setiap model dievaluasi

menggunakan Koefisien Determinasi (R^2). Model terbaik untuk setiap kelompok dan komponen disajikan dalam Tabel 5.1**.**

Tabel 5.1 Model Regresi dan Skor Determinasi UPR-3

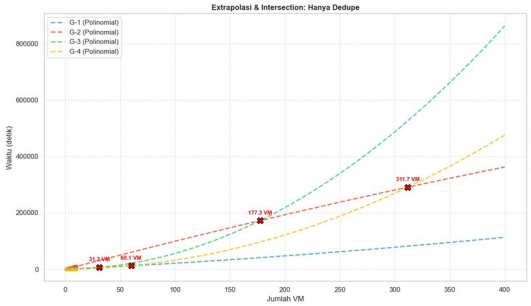
Kelompok	Komponen	Model Terbaik	R^2 (Model Terbaik)	Implikasi Kompleksitas
G-1	Total	$y = 0.43x^2 + 189.65x - 163.73$	0,9966	Mendekati $O(n^2)$ untuk n kecil, didominasi komponen linier.
G-1	Pemindaian	$y = 0.20x^2 + 0.55x + 8.71$	0,9983	Sangat rendah, hampir konstan.
G-1	Dedupe	$y = 0.23x^2 + 189.10x - 172.45$	0,9965	Didominasi O(n), dengan komponen kuadrat kecil.
G-2	Total	$y = -0.07x^2 + 1031.62x - 1017.10$	0,9943	Dominasi kuat O(n) linier.
G-2	Pemindaian	$y = 0.23x^2 + 0.56x + 10.76$	0,9975	Sangat rendah, hampir konstan.
G-2	Dedupe	$y = -0.31x^2 + 1031.06x - 1027.85$	0,9942	Dominasi kuat O(n) linier.
G-3	Total	$y = 5.92x^2 + 39.82x - 47.31$	0,9978	Gabungan O(n) dan $O(n^2)$, dengan koefisien kuadrat lebih signifikan.
G-3	Pemindaian	$y = 0.59x^2 + 13.52x + 0.64$	0,9999	O(n) dominan.
G-3	Dedupe	$y = 5.33x^2 + 26.30x - 47.95$	0,9965	Komponen kuadrat lebih dominan.
G-4	Total	$y = 3.56x^2 + 39.67x - 37.54$	0,9986	Mirip G-3, dengan koefisien kuadrat lebih kecil.
G-4	Pemindaian	$y = 0.65x^2 + 13.33x + 1.05$	0,9999	O(n) dominan.
G-4	Dedupe	$y = 2.91x^2 + 26.34x - 38.59$	0,9973	Komponen kuadrat signifikan.

Analisis model mengungkap perbedaan mendasar dalam kompleksitas proses:

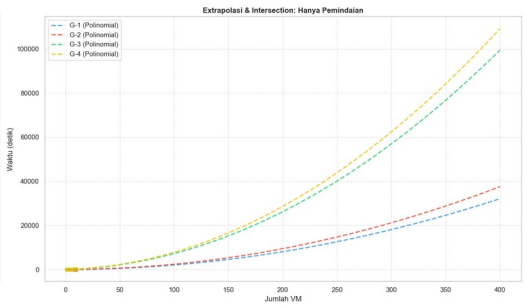
1. ***Black-box* (G-1 & G-2):** Waktu eksekusi didominasi oleh komponen linier ($O(n)$) yang sangat besar, terutama pada fase deduplikasi. Hal ini merefleksikan biaya komputasi tinggi untuk memindai dan membandingkan blok-blok biner dalam file citra yang monolitik, di mana setiap byte tambahan dari VM baru harus diproses. Koefisien linier yang jauh lebih besar pada G-2 (1031) dibanding G-1 (189) menunjukkan bahwa penambahan lapisan kompresi secara signifikan memperberat beban pemrosesan pada model *black-box*.
2. ***White-box* (G-3 & G-4):** Meski waktu absolut jauh lebih cepat, model untuk *white-box* menunjukkan kontribusi komponen kuadratik ($O(n^2)$)

yang lebih nyata, terutama pada fase deduplikasi. Ini mengindikasikan bahwa saat jumlah VM bertambah, overhead untuk membandingkan dan mengelola referensi antar berkas yang terduplikasi (misalnya, pembaruan metadata *reflink*) tumbuh lebih cepat karena jumlah berkas lebih banyak. Namun, karena konstanta a pada komponen kuadratik relatif kecil (2.91 hingga 5.92), dampak negatif skalabilitas ini baru akan signifikan pada jumlah VM yang sangat besar. Koefisien yang lebih rendah pada G-4 dibanding G-3 menunjukkan bahwa kompresi justru sedikit meredam kompleksitas kuadratik pada model *white-box*.

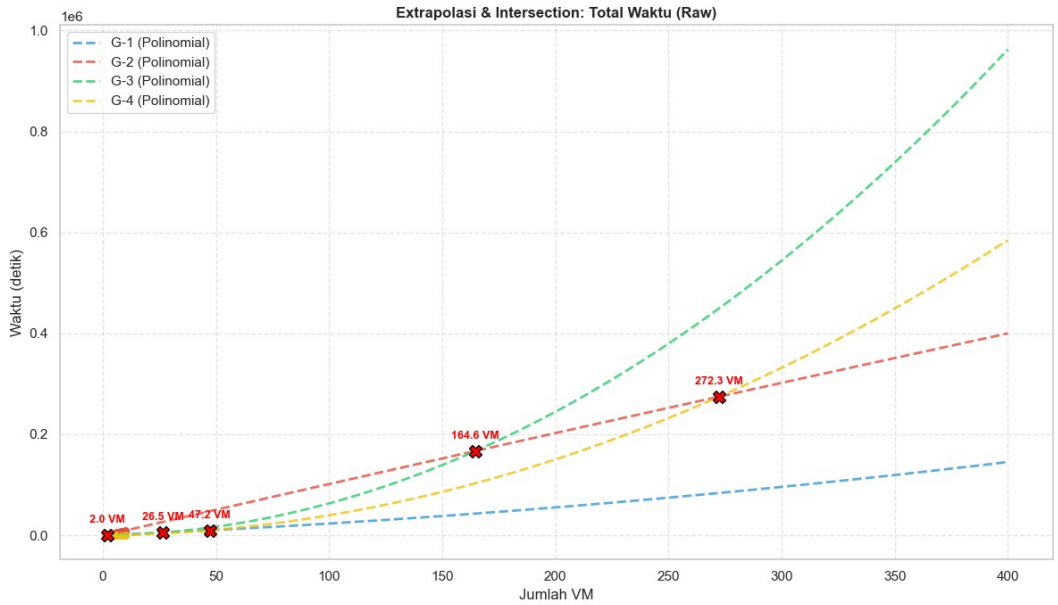
5.3.3 Ekstrapolasi dan Titik Potong Kinerja



Gambar 5.13 Ekstrapolasi Waktu Deduplikasi



Gambar 5.14 Ekstrapolasi Waktu Pemindaian



Gambar 5.15 Ekstrapolasi Waktu Total (Pemindaian + Dedupe)

Model regresi polinomial terbaik kemudian digunakan untuk melakukan ekstrapolasi guna memproyeksikan perilaku sistem pada skala VM yang lebih besar dan untuk menemukan titik potong (*intersection*) antar kelompok. Proyeksi untuk Total Waktu, Waktu Pemindaian, dan Waktu Deduplikasi divisualisasikan pada Gambar 5.13, 5.14 dan 5.15.

Ekstrapolasi memperkuat temuan sebelumnya: kelompok *black-box* (terutama G-2) menunjukkan kurva waktu yang melonjak sangat curam, sementara *white-box* tumbuh dengan laju yang lebih lunak. Titik potong antar kurva, yang menunjukkan jumlah VM di mana waktu eksekusi dua kelompok menjadi sama, dihitung dan disajikan dalam **Tabel 5.2**.

Tabel 5.2 Titik Potong Antarkelompok

Metrik Perbandingan	Pasangan Kelompok	Titik Potong (Jumlah VM)	Waktu pada Titik Potong (detik)
Total Waktu	G-1 vs G-3	26,5	~5,166
Total Waktu	G-1 vs G-4	47,2	~9,753
Total Waktu	G-2 vs G-3	164,6	~166,798
Total Waktu	G-2 vs G-4	272,3	~274,449
Total Waktu	G-3 vs G-4	2	~56

Titik potong memberikan wawasan praktis yang berharga:

- **G-1 vs G-3/G-4:** Pada skala kecil (<30 VM), *white-box* tanpa kompresi (G-3) lebih cepat. Namun, setelah sekitar 27 VM, *black-box* normal (G-1) menjadi lebih cepat karena pertumbuhan kuadratik pada G-3 mulai berpengaruh. G-4 (*white-box* terkompresi) baru akan disusul oleh G-1 setelah sekitar 47 VM.
- **G-2 vs G-3/G-4:** *Black-box* terkompresi (G-2) membutuhkan skala yang sangat besar (>160 VM) untuk menyamai kecepatan *white-box*. Ini mengonfirmasi bahwa meski kombinasi kompresi dan deduplikasi parsial dapat mendekati efisiensi ruang *white-box* (seperti dibahas di 5.2), biaya komputasi yang harus dibayar sangatlah tinggi.
- **G-3 vs G-4:** Titik potong di 2 VM mengindikasikan bahwa untuk sebagian besar skenario praktis (VM > 2), G-3 selalu lebih lambat daripada G-4.

5.3.4 Kesimpulan Kinerja Waktu

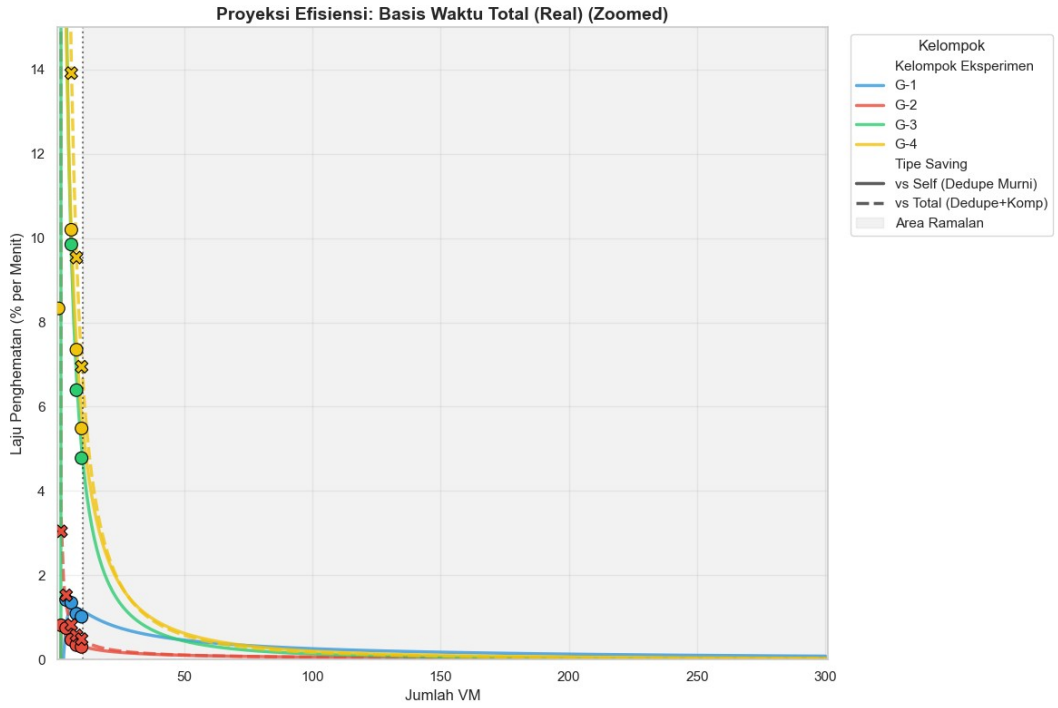
Secara keseluruhan, analisis kinerja waktu secara tegas mendemonstrasikan keunggulan arsitektur *white-box* (G-3 dan G-4) atas *black-box* (G-1 dan G-2) dalam hal kecepatan dan stabilitas proses deduplikasi pada skala percobaan (hingga 10 VM). Keunggulan ini terutama berasal dari efisiensi pemindaian berbasis berkas yang menghindari overhead pemrosesan blok biner buta.

Namun, analisis model prediktif mengungkap perbedaan karakteristik skalabilitas jangka panjang. Model *black-box*, meski sangat lambat, cenderung tumbuh secara mendekati linier (O(n)), sedangkan model *white-box* mulai menunjukkan komponen pertumbuhan kuadratik ($O(n^2)$) yang kecil. Implikasinya, pada lingkungan dengan ratusan VM yang homogen, gap kinerja mutlak mungkin akan tetap besar mendukung *white-box*, tetapi gap *relatif* dalam hal skalabilitas mungkin tidak sebesar yang terlihat pada skala kecil.

Pilihan antara G-3 (*white-box* normal) dan G-4 (*white-box* terkompresi) menjadi pertimbangan antara kecepatan optimal (G-3) dan penghematan ruang

optimal (G-4), di mana G-4 mengenakan "pajak kinerja" untuk kompresi. Sementara itu, penggunaan model *black-box* (terutama G-2) hanya dapat dipertimbangkan jika kendala ruang penyimpanan sangat kritis dan biaya komputasi serta waktu pemrosesan yang sangat lama dapat diterima.

5.4 Perbandingan Persentase Penghematan Terhadap Waktu



Gambar 5.16 Proyeksi % hemat per menit

Analisis sebelumnya telah menguraikan keunggulan *white-box* dalam hal efisiensi ruang absolut dan kecepatan deduplikasi. Namun, dalam praktiknya, administrator sistem seringkali perlu mempertimbangkan *trade-off* antara besarnya ruang yang dihemat dan waktu serta sumber daya komputasi yang dibutuhkan untuk mencapainya. Oleh karena itu, bagian ini memperkenalkan metrik baru: persentase penghematan ruang per menit pemrosesan, yang dihitung dengan membagi persentase penghematan total (skenario UPR-3) dengan waktu deduplikasi total (dalam menit). Metrik ini mengkuantifikasi seberapa "cepat" suatu strategi penyimpanan menghasilkan penghematan ruang, sehingga dapat menjadi pertimbangan praktis dalam memilih arsitektur yang optimal untuk skala tertentu.

Perhitungan dilakukan dengan menggunakan data penghematan dari skenario UPR-3—yang mewakili kondisi setelah proses deduplikasi *out-of-band* selesai—dan waktu total deduplikasi dari skenario G-1. Hasil perhitungan untuk 10 VM kemudian diproyeksikan ke skala yang lebih besar menggunakan model regresi yang telah diperoleh sebelumnya untuk penghematan dan waktu. Proyeksi ini divisualisasikan dalam Gambar 5.16.

Berdasarkan analisis proyeksi, dapat diidentifikasi dua fase kinerja yang berbeda:

1. **Fase Skala Kecil (1–30 VM):** Pada fase ini, arsitektur G-4 dan G-3 memegang keunggulan efisiensi yang jelas, menghasilkan penghematan ruang per menit yang jauh lebih tinggi dibandingkan pendekatan *black-box*. Hal ini sejalan dengan temuan sebelumnya di mana *white-box* menghasilkan penghematan ruang maksimal (G-4) dengan waktu pemrosesan yang relatif singkat. Nilai metrik untuk G-4 dan G-3 pada fase ini masing-masing stabil di kisaran 0.85–0.95 %/menit dan 0.90–1.05 %/menit, mengindikasikan bahwa setiap menit proses deduplikasi dapat menghemat sekitar 1% dari total ruang baseline.
2. **Fase Skala Menengah-Besar (>30 VM):** Seiring peningkatan jumlah VM, keunggulan efisiensi *white-box* mulai melandai (*plateau*), sementara kinerja *black-box* menunjukkan peningkatan relatif. Fenomena ini terutama disebabkan oleh karakteristik skalabilitas waktu eksekusi yang berbeda, seperti yang terungkap dalam analisis model di sub-bab 5.3. Model *white-box* (G-3 dan G-4) mulai menunjukkan komponen pertumbuhan kuadratik dalam waktu eksekusinya, sedangkan model *black-box* (terutama G-1) tumbuh secara hampir linier. Akibatnya, kurva "penghematan per menit" untuk G-3 dan G-4 mengalami penurunan gradual, sementara kurva untuk G-1 (*Black-box Normal*) justru menunjukkan tren yang relatif datar atau sedikit meningkat.

Proyeksi jangka panjang mengungkap titik konvergensi yang menarik. Gambar 5.16 menunjukkan bahwa pada skala sekitar >100 VM, efisiensi G-1 (dalam hal penghematan per menit) mulai mendekati dan bahkan dapat melampaui G-3. Namun, penting untuk diingat bahwa konvergensi ini terjadi pada tingkat penghematan ruang absolut yang berbeda secara signifikan. Pada skala 200 VM, meski G-1 mungkin menawarkan efisiensi waktu yang sebanding, total penghematan ruang yang dicapai (dalam persentase) tetap jauh di bawah G-3 dan G-4, seperti yang diprediksi oleh model logaritmik penghematan ruang. Dengan kata lain, *black-box* mungkin menjadi "lebih cepat" dalam menghasilkan setiap persen penghematan pada skala sangat besar, tetapi "jarak" total yang harus ditempuh untuk menghemat ruang (dari baseline) jauh lebih pendek pada *white-box*.

Implikasi praktis dari analisis ini adalah adanya skala optimal untuk setiap arsitektur:

- Untuk lingkungan dengan kurang dari 50 VM: Arsitektur *white-box* merupakan pilihan terbaik, karena menawarkan kombinasi penghematan ruang yang sangat baik dan kecepatan pemrosesan yang tinggi, sehingga memaksimalkan metrik penghematan per satuan waktu.
- Untuk lingkungan dengan puluhan hingga ratusan VM yang homogen: G-4 tetap menjadi pilihan untuk optimasi ruang tertinggi, meski dengan sedikit penalti waktu. Sementara itu, G-1 dapat dipertimbangkan jika beban komputasi untuk deduplikasi perlu dijaga seminimal mungkin dan

penghematan ruang bukan prioritas utama, mengingat skalabilitas waktunya yang lebih linear.

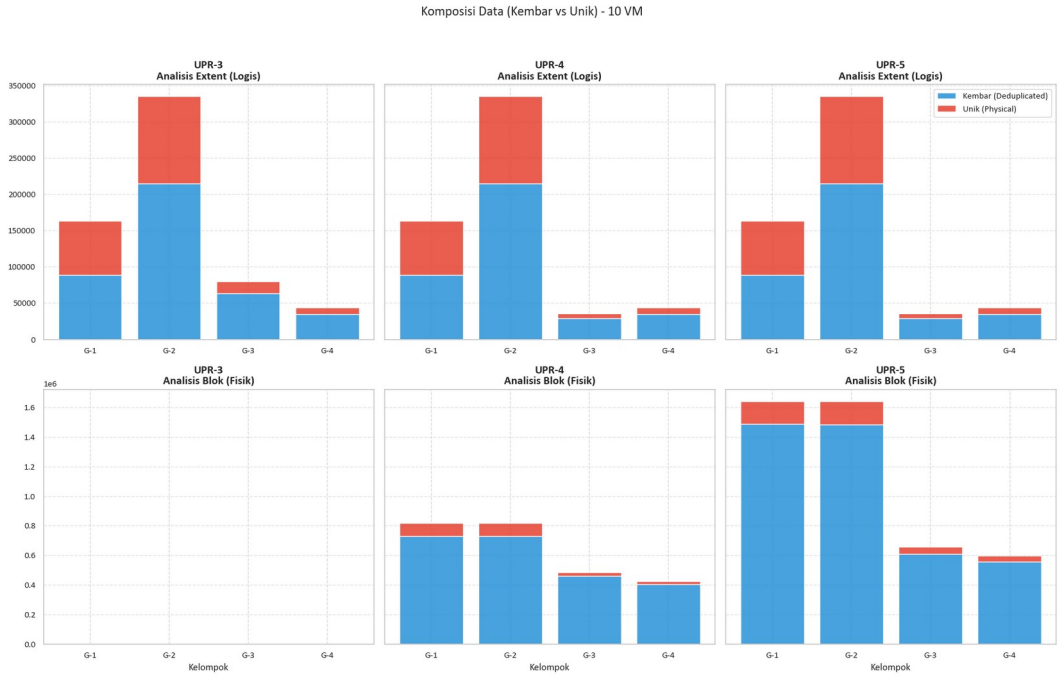
- Arsitektur G-2 (*Black-box* Terkompresi): Meski mampu mencapai penghematan ruang yang mendekati *white-box* (setelah deduplikasi parsial), kinerja waktunya yang sangat buruk membuat metrik "penghematan per menit"-nya selalu berada di posisi terendah di semua skala. Oleh karena itu, arsitektur ini hanya direkomendasikan dalam skenario yang sangat spesifik di mana kendala ruang penyimpanan sangat kritis dan waktu pemrosesan yang sangat lama dapat ditoleransi.

Kesimpulannya, meskipun *white-box* secara absolut lebih unggul dalam menghemat ruang dan waktu, metrik gabungan "penghematan per menit" mengungkap bahwa pada skala deployment yang sangat besar, keunggulan relatifnya dalam hal kecepatan menghasilkan penghematan dapat menyusut akibat pola pertumbuhan waktu eksekusi yang berbeda. Keputusan akhir dalam memilih arsitektur harus mempertimbangkan triad antara target penghematan ruang, ketersediaan waktu/jendela pemeliharaan, dan skala VM yang dikelola.

5.5 Mikroanalisis Statistik Duperemove

Analisis mendalam terhadap basis data hash (*hashfile*) yang dihasilkan oleh `duperemove` memberikan wawasan mikro tentang karakteristik alokasi data dan mekanisme deduplikasi pada kedua model penyimpanan. Data untuk analisis ini diambil dari lampiran A.4, yang berisi statistik terperinci dari proses deduplikasi. Analisis ini melengkapi pemahaman makro tentang penghematan ruang dan kinerja waktu dengan mengungkap hubungan antara pola fragmentasi, distribusi ukuran *extent*, dan beban kerja proses deduplikasi.

5.5.1 Komposisi Data dan Karakteristik Pindaian



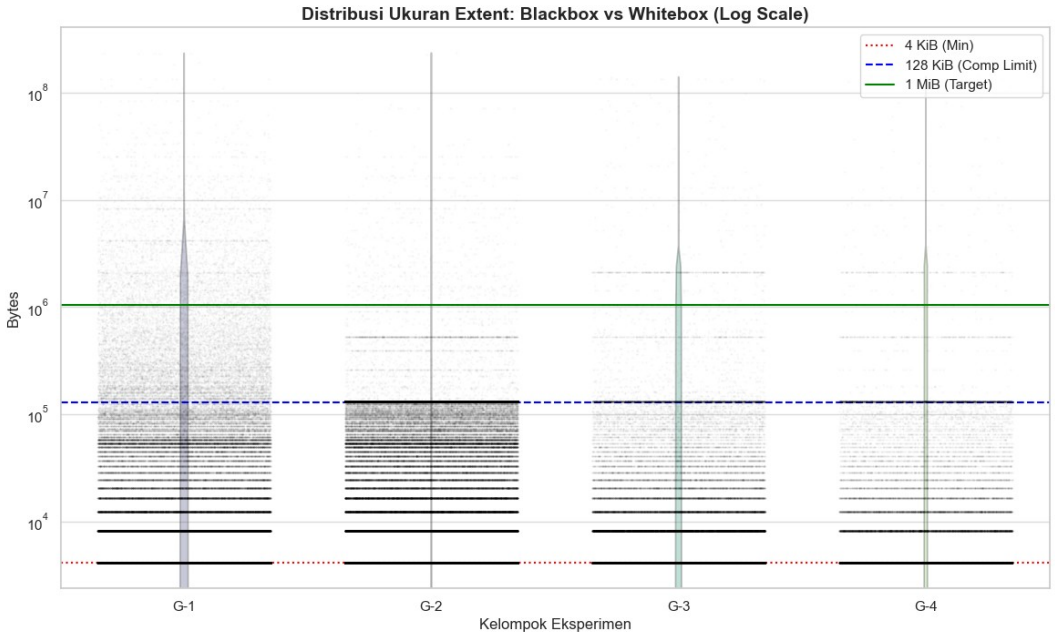
Gambar 5.17 Komposisi Data VM (10 VM)

Gambar 5.17 memberikan gambaran awal yang jelas tentang perbedaan mendasar dalam komposisi objek yang diproses oleh `duperemove` pada setiap kelompok. Perbedaan ini secara langsung mempengaruhi fase pemindaian (*scan*) dan deduplikasi.

Pada model *Black-box* (G-1 dan G-2), objek pindaian sedikit, namun kompleks. `Duperemove` hanya melihat 10 berkas—yaitu 10 file citra disk (`.img`) yang bersifat buram (*opaque*). Namun, setiap file citra ini dipetakan ke dalam ratusan ribu *extent*. Hal ini mencerminkan fragmentasi internal yang parah di dalam setiap citra, di mana data *guest* yang kontinu terpecah menjadi banyak potongan (*extent*) pada sistem berkas *host* akibat pola tulis dan *trim* yang tidak selaras.

Pada model *White-box* (G-3 dan G-4), Objek pindaian banyak, namun terstruktur. `Duperemove` melihat langsung jutaan berkas individual dari sistem operasi *guest*. Meski jumlah berkas sangat banyak, total *extent* yang dihasilkan justru lebih sedikit dibandingkan *black-box*. Ini menunjukkan bahwa sebagian besar berkas berukuran kecil dapat disimpan dalam satu atau sedikit *extent*. Kompresi pada G-4 bahkan mengurangi jumlah berkas yang terdeteksi hampir setengahnya.

5.5.2 Distribusi Ukuran Extent dan Pola Fragmentasi



Gambar 5.18 Violin Plot Distribusi Ukuran Eksten

Statistik deskriptif dari distribusi ukuran *extent* yang disajikan dalam Gambar 5.18 mengungkap pola alokasi yang berbeda. Data dari yang dilampirkan pada A.4 menunjukkan bahwa pada model *Black-box* (G-1 dan G-2), distribusi sangat heterogen. Nilai Koefisien Variasi (CV) yang sangat tinggi mengindikasikan variasi ukuran *extent* yang ekstrem. Modus adalah 4 KiB (ukuran blok dasar), tetapi terdapat *extent* berukuran sangat besar (>200 MiB). Pola ini adalah ciri khas fragmentasi internal pada objek monolitik. Sedangkan pada model *white-box* (G-3 dan G-4), distribusi terkonsentrasi pada ukuran kecil. Modus dan median sama-sama 4 KiB, menunjukkan bahwa sebagian besar *extent* berukuran sangat kecil. Ini sesuai dengan karakteristik sistem berkas yang terdiri dari ribuan berkas kecil.

5.5.3 Hubungan antara Struktur Data dan Kinerja Pemrosesan

Data agregat untuk 10 VM dari dat yang terlampir pada A.2 dan A.3 dianalisis untuk menemukan korelasi. Tabel 5.3 memvisualisasikan hubungan multivarian ini.

Tabel 5.3 Ringkasan Statistik dan Kinerja untuk 10 VM (Skenario UPR-3)

Kelompok	Total Berkas	Total Extent	Jml. Extent Kembar	Rasio Kembar	Waktu Total (s)	Waktu Scan (s)	Waktu Dedupe (s)
G-1	10	163635	88771	54,2%	1749,1	34,4	1714,7
G-2	10	334973	214934	64,2%	9102,2	40,1	9062,0

G-3	~1,18 juta	79525	63644	80,0%	930,1	195,1	735,1
G-4	~0,59 juta	43777	34491	78,8%	710,4	199,7	510,6

Beberapa hubungan kunci teridentifikasi:

1. **Jumlah Berkas vs. Waktu Pemindaian (Scan):** Terdapat korelasi positif yang kuat. *White-box* yang memiliki jutaan berkas membutuhkan waktu pemindaian yang jauh lebih lama dibandingkan *black-box* yang hanya memindai 10 file.
2. **Jumlah Extent vs. Waktu Deduplikasi:** Waktu deduplikasi berkorelasi dengan jumlah *extent* yang harus dibandingkan. G-2, dengan total *extent* tertinggi, memiliki waktu deduplikasi terlama.
3. **Rasio Extent Kembar vs. Efisiensi Ruang:** Rasio *extent* kembar merupakan indikator langsung efektivitas deduplikasi. Kelompok *white-box* memiliki rasio kembar yang sangat tinggi (≈80%), yang menjelaskan tingkat penghematan ruang maksimal yang mereka capai.

5.5.4 Implikasi Alokator Btrfs dan Mekanisme Deduplikasi

Perbedaan pola di atas berakar pada interaksi antara model penyimpanan dan alokator Btrfs:

- **Pada *Black-box*:** Btrfs melihat file citra `.img` sebagai objek buram tunggal. Alokasi dan pembebasan blok di dalamnya oleh *guest* menyebabkan fragmentasi internal yang parah dan tidak terprediksi.
- **Pada *White-box*:** Btrfs mengalokasikan setiap berkas *guest* secara terpisah. `Duperemove` dapat melihat dan memproses setiap berkas ini secara individual. Berkas sistem yang identik antar VM secara alami berbagai *extent* fisik yang sama.

5.5.5 Analisis Deduplikasi Parsial: Extent vs Blok

Seperti ditunjukkan dalam Gambar 5.17, komposisi data antara skenario deduplikasi penuh (UPR-3) dan parsial (UPR-4, UPR-5) memiliki perbedaan mendasar terkait granularitas pemindaian.

Pada deduplikasi parsial, `duperemove` tidak hanya mengidentifikasi duplikasi pada level *extent*, tetapi juga memecah *extent* menjadi blok-blok yang lebih kecil (128 KiB untuk UPR-4 dan 64 KiB untuk UPR-5) untuk mencari redundansi sub-*extent*. Data dari lampiran A.4 menunjukkan fenomena menarik:

1. **Jumlah *Extent* Tidak Berubah:** Untuk kelompok *black-box* (G-1 dan G-2), jumlah total *extent* yang dipindai tetap sama persis antara UPR-3, UPR-4, dan UPR-5. Hal ini membuktikan bahwa proses *chunking* parsial tidak mengubah cara Btrfs memetakan file citra ke dalam *extent*, melainkan hanya menambahkan lapisan analisis di bawah level *extent* tersebut.

2. **Penambahan Analisis Level Blok:** Meski jumlah *extent* tetap, deduplikasi parsial menambahkan sejumlah besar blok ke dalam proses analisis. Sebagai contoh, pada G-1, analisis UPR-4 menambahkan pemrosesan terhadap 819.200 blok (dengan 730.572 di antaranya teridentifikasi kembar), dan UPR-5 terhadap 1.638.400 blok (dengan 1.486.032 kembar). Dengan ini penghematan lebih dapat diraih seperti dibahas di 5.2.4.
3. **Efisiensi *White-box*:** Pada kelompok *white-box*, deduplikasi parsial justru mengurangi jumlah *extent* yang diproses (karena hanya memindai subset berkas), namun tetap mempertahankan tingkat penghematan ruang yang hampir sama. Ini menunjukkan bahwa redundansi pada arsitektur *white-box* sudah teridentifikasi dengan sangat baik pada level *extent*, sehingga analisis level blok yang lebih granular tidak diperlukan.

5.5.6 Kesimpulan dari Mikroanalisis

Analisis mikro terhadap data hash `duperemove` mengonfirmasi bahwa keunggulan *white-box* bersifat sistemik. Visibilitas penuhnya terhadap struktur berkas *guest* memungkinkan identifikasi duplikasi yang sangat akurat dan efisien pada level *extent*, menghasilkan rasio kembar tinggi (>80%) dan waktu deduplikasi total yang lebih singkat. Sebaliknya, *black-box* bergulat dengan fragmentasi internal yang menyebabkan jumlah *extent* membengkak dan rasio kembar yang lebih rendah. Deduplikasi parsial, meski menemukan banyak blok kembar, tidak mampu meningkatkan penghematan ruang lebih lanjut pada *black-box*, mengindikasikan bahwa masalah utamanya adalah keselarasan (*alignment*) dan fragmentasi, bukan redundansi tersembunyi di dalam *extent*. Harmoni antara model penyimpanan, alokator Btrfs, dan algoritma `duperemove` pada *white-box* terbukti menjadi kombinasi yang unggul.

BAB 6 PENUTUP

6.1 Kesimpulan

Penelitian ini telah melakukan evaluasi praktis terhadap penghematan ruang penyimpanan VM pada sistem berkas Btrfs dengan membandingkan dua pendekatan fundamental: model penyimpanan *white-box* yang memanfaatkan akses langsung ke struktur berkas melalui Virtio-FS, dan model penyimpanan *black-box* tradisional yang bekerja pada level blok data buram. Berdasarkan analisis eksperimen yang komprehensif, dapat disimpulkan bahwa pendekatan *white-box* menawarkan keunggulan yang signifikan dalam hal efektivitas penghematan ruang, efisiensi waktu pemrosesan, dan stabilitas kinerja.

Pendekatan *white-box* terbukti mampu merealisasikan konsep deduplikasi sadar semantik (semantics-aware) secara praktis tanpa kompleksitas pemodelan grafik seperti pada sistem *Expelliarmus*. Dengan mengekspos struktur sistem berkas guest secara transparan ke host melalui Virtio-FS, Btrfs mampu mengidentifikasi redundansi pada level berkas dan paket perangkat lunak secara presisi. Hal ini menghasilkan reduksi penyimpanan hingga 82,28%, melampaui pendekatan *black-box* (73,78%) yang terkendala oleh sifat "buram" (*opaque*) dan masalah pergeseran batas blok (*boundary shifting*) pada berkas citra diska.

Hasil penelitian ini mengonfirmasi temuan Meyer dan Bolosky bahwa deduplikasi yang bekerja pada granularitas berkas atau *extent* berkas mampu memberikan penghematan ruang yang sangat kompetitif dengan *overhead* yang jauh lebih rendah. Model *white-box* tidak hanya unggul dalam ruang, tetapi juga memberikan **kinerja waktu** yang 6 hingga 12 kali lebih cepat (12–15 menit) dibandingkan model *black-box* (30 menit – 2,5 jam). Percepatan ini dicapai karena Btrfs dapat langsung memproses *extent* berkas yang relevan tanpa harus memindai keseluruhan blok biner besar pada citra diska tunggal yang rentan fragmentasi internal.

Temuan penelitian ini secara keseluruhan membuktikan bahwa implementasi teknik *white-box* melalui teknologi Virtio-FS dan Btrfs tidak hanya layak secara teknis, tetapi juga memberikan keunggulan komprehensif dibandingkan pendekatan tradisional. Harmoni antara visibilitas penuh terhadap struktur berkas, mekanisme alokasi data Btrfs, dan algoritma deduplikasi berorientasi berkas menciptakan sinergi optimal untuk manajemen penyimpanan VM yang efisien.

6.2 Saran

Berdasarkan temuan dan keterbatasan dalam penelitian ini, beberapa arah pengembangan dapat dipertimbangkan untuk penelitian lanjutan. Pertama, ekspansi skala eksperimen dengan jumlah VM yang lebih banyak dan variasi beban kerja yang lebih beragam akan memberikan pemahaman lebih mendalam tentang karakteristik skalabilitas masing-masing pendekatan. Pengujian dengan puluhan hingga ratusan VM serta kombinasi *workload* yang heterogen dapat

mengungkap pola perilaku sistem pada kondisi yang lebih mendekati lingkungan produksi.

Kedua, peningkatan validitas statistik melalui pengulangan eksperimen yang lebih banyak direkomendasikan untuk memperkuat reliabilitas temuan, khususnya dalam pengukuran kinerja waktu yang bersifat stokastik. Dengan jumlah iterasi yang memadai, analisis statistik dapat memberikan kepercayaan yang lebih tinggi terhadap kesimpulan yang dihasilkan, terutama dalam mengidentifikasi *outlier* dan variasi kinerja.

Ketiga, penelitian mendatang dapat menyelidiki efek operasi replikasi Btrfs seperti `btrfs send/receive` terhadap pola fragmentasi dan efektivitas deduplikasi, khususnya pada model penyimpanan *black-box*. Mengingat operasi ini umum digunakan dalam skenario backup dan migrasi VM, pemahaman tentang dampaknya terhadap efisiensi penyimpanan akan bernilai praktis yang tinggi.

Keempat, eksplorasi optimasi parameter deduplikasi secara spesifik untuk arsitektur *black-box* dapat menjadi fokus penelitian berikutnya. Penyelidikan terhadap kombinasi ukuran *chunk*, algoritma *hashing*, dan strategi *caching* yang optimal mungkin dapat mengurangi beban komputasi yang signifikan saat ini tanpa mengorbankan tingkat penghematan ruang yang dicapai.

Terakhir, studi perbandingan dengan teknologi penyimpanan alternatif seperti ZFS atau VDO serta platform virtualisasi lain dapat memperluas cakupan pengetahuan tentang solusi penghematan ruang penyimpanan VM. Analisis komparatif semacam ini akan memberikan perspektif yang lebih komprehensif bagi praktisi dalam memilih stack teknologi yang paling sesuai dengan kebutuhan spesifik mereka.

DAFTAR REFERENSI

Binu, A. dan Kumar, G.S., 2011. Virtualization Techniques: A Methodical Review of XEN and KVM. Dalam: A. Abraham, J. Lloret Mauri, J.F. Buford, J. Suzuki dan S.M. Thampi, ed. *Advances in Computing and Communications*. Berlin, Heidelberg: Springer. hlm.399–410. https://doi.org/10.1007/978-3-642-22709-7_40.

Borowski, A., 2025. *kilobyte/compsize*. Tersedia di: <https://github.com/kilobyte/compsize> [Diakses 25 November 2025].

Borra, P., 2024a. A Survey of Google Cloud Platform (GCP): Features, Services, and Applications. *International Journal of Advanced Research in Science, Communication and Technology*, hlm.191–199. https://doi.org/10.48175/IJARSCT-18922.

Borra, P., 2024b. Comprehensive Survey of Amazon Web Services (AWS): Techniques, Tools, and Best Practices for Cloud Solutions. 9, hlm.24–29.

Btrfs at Scale: How Meta Depends on Btrfs for Our Entire Infrastructure - Josef Bacik, Meta. 2023. The Linux Foundation. Tersedia di: <https://www.youtube.com/watch?v=8vE9_0cQweg> [Diakses 27 Januari 2026].

BTRFS Developers, 2025a. *Compression — BTRFS documentation*. [daring] Tersedia di: <https://btrfs.readthedocs.io/en/latest/Compression.html#incompressible-data> [Diakses 17 Oktober 2025].

BTRFS Developers, 2025b. *Deduplication — BTRFS documentation*. [daring] Tersedia di: <https://btrfs.readthedocs.io/en/latest/Deduplication.html> [Diakses 21 November 2025].

BTRFS Developers, 2025c. *Introduction — BTRFS documentation*. [daring] Tersedia di: <https://btrfs.readthedocs.io/en/latest/Introduction.html> [Diakses 25 November 2025].

BTRFS Developers, 2025d. *Reflink — BTRFS documentation*. [daring] Tersedia di: <https://btrfs.readthedocs.io/en/latest/Reflink.html> [Diakses 26 November 2025].

BTRFS Developers, 2025e. *Subvolumes — BTRFS documentation*. [daring] Tersedia di: <https://btrfs.readthedocs.io/en/latest/Subvolumes.html> [Diakses 26 November 2025].

Clements, A.T., Ahmad, I., Vilayannur, M. dan Li, J., 2009. Decentralized deduplication in SAN cluster file systems. Dalam: *Proceedings of the 2009 conference on USENIX Annual technical conference*, USENIX'09. USA: USENIX Association. hlm.8.

Czenczek, H. dan Maglione, G., 2023. virtio-fs: Present and Future. [daring] KVM Forum 2023. Brno, Ceko. Tersedia di: <https://kvm-forum.qemu.org/2023/talk_O5WLFhA.pdf>.

Fasheh, M., 2025. *markfasheh/duperemove*. Tersedia di: <https://github.com/markfasheh/duperemove> [Diakses 25 November 2025].

Foundries.io, Ltd., 2025. *LmP Root File-System Over NFS — FoundriesFactory 95 documentation*. [daring] Tersedia di: <https://docs.foundries.io/latest/reference-manual/linux/linux-nfs-boot.html> [Diakses 21 November 2025].

Hajnoczi, S., 2020. virtio-fs: A Shared File System for Virtual Machines. [daring] FOSDEM 2020. Brussels, Belgia. Tersedia di: <https://archive.fosdem.org/2020/schedule/event/vai_virtio_fs/>.

Hajnoczi, S., Szeredi, M., Goyal, V. dan Gilbert, D., 2019. virtio-fs, A Shared FIlesystem for Virtual Machines. KVM Forum 2019. Lyon, Prancis.

Jayaram, K.R., Peng, C., Zhang, Z., Kim, M., Chen, H. dan Lei, H., 2011. An empirical analysis of similarity in virtual machine images. Dalam: *Proceedings of the Middleware 2011 Industry Track Workshop*, Middleware '11. [daring] New York, NY, USA: Association for Computing Machinery. hlm.1–6. https://doi.org/10.1145/2090181.2090187.

Jin, K. dan Miller, E.L., 2009. The effectiveness of deduplication on virtual machine disk images. Dalam: *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, SYSTOR '09. [daring] New York, NY, USA: Association for Computing Machinery. hlm.1–12. https://doi.org/10.1145/1534530.1534540.

Jones, R., 2025. *virt-sparsify*. [daring] virt-sparsify - Make a virtual machine disk sparse. Tersedia di: <https://libguestfs.org/virt-sparsify.1.html> [Diakses 28 November 2025].

Meyer, D.T. dan Bolosky, W.J., 2012. A study of practical deduplication. *ACM Transactions on Storage*, 7(4), hlm.1–20. https://doi.org/10.1145/2078861.2078864.

Morton, D., 2013. *IBM Mainframe Operating Systems: Timeline and Brief Explanation for the IBM System/360 and Beyond*. [daring] Tersedia di: <https://web.archive.org/web/20140701185435/http://www.demorton.com/Tech/$OSTL.pdf>.

Noveck, D. dan Lever, C., 2020. *Network File System (NFS) Version 4 Minor Version 1 Protocol*. [Request for Comments] Internet Engineering Task Force. https://doi.org/10.17487/RFC8881.

QEMU Developers, 2025. *Documentation/9psetup - QEMU*. [daring] Tersedia di: <https://wiki.qemu.org/Documentation/9psetup> [Diakses 21 November 2025].

Rodeh, O., 2008. B-trees, shadowing, and clones. *ACM Transactions on Storage*, 3(4), hlm.1–27. https://doi.org/10.1145/1326542.1326544.

Salter, J., 2021. *Examining btrfs, Linux's perpetually half-finished filesystem*. [daring] Ars Technica. Tersedia di: <https://arstechnica.com/gadgets/2021/09/examining-btrfs-linuxs-perpetually-half-finished-filesystem/> [Diakses 25 November 2025].

Saurabh, N., Remmers, J., Kimovski, D., Prodan, R. dan Barbosa, J.G., 2019. Semantics-Aware Virtual Machine Image Management in IaaS Clouds. Dalam: *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. [daring] 2019 IEEE International Parallel and

Distributed Processing Symposium (IPDPS). Rio de Janeiro, Brazil: IEEE. hlm.418–427. https://doi.org/10.1109/IPDPS.2019.00052.

The kernel development community, 2025. *Mounting the root filesystem via NFS (nfsroot) — The Linux Kernel documentation*. [daring] Tersedia di: <https://docs.kernel.org/admin-guide/nfs/nfsroot.html> [Diakses 21 November 2025].

Thwel, T.T. dan Sinha, G.R. ed., 2021. *Data deduplication approaches: concepts, strategies, and challenges*. London, United Kingdom: Academic Press, an imprint of Elsevier.

Tran, M.D., Nguyen, L.A., Lee, H.T., Paek, J., Cho, S. dan Son, Y., 2024. A Survey on Copy-on-Write File Systems. Dalam: *2024 15th International Conference on Information and Communication Technology Convergence (ICTC)*. [daring] 2024 15th International Conference on Information and Communication Technology Convergence (ICTC). hlm.436–441. https://doi.org/10.1109/ICTC62082.2024.10826721.

Virtio-FS Developers, 2024. *Virtiofs Design Document*. [Dokumentasi] Virtio-fs Home Page. Tersedia di: <https://virtio-fs.gitlab.io/design.html> [Diakses 25 September 2024].

Zygo, 2025. *Zygo/bees*. Tersedia di: <https://github.com/Zygo/bees> [Diakses 26 November 2025].

LAMPIRAN A DATASET YANG DIPEROLEH

A.1 Data Ukuran-Ukuran VM

Kelompok Eksperimen	Kode Skenario	Jumlah VM	Jenis Penyimpanan	Volume Pendukung	Penggunaan Ruang (Bytes)
G-1	UPR-1	1	*black-box*	Normal	2686275584
G-1	UPR-1	2	*black-box*	Normal	6206902272
G-1	UPR-1	3	*black-box*	Normal	9241391104
G-1	UPR-1	4	*black-box*	Normal	11927883776
G-1	UPR-1	5	*black-box*	Normal	15448256512
G-1	UPR-1	6	*black-box*	Normal	18134515712
G-1	UPR-1	7	*black-box*	Normal	21169381376
G-1	UPR-1	8	*black-box*	Normal	24689709056
G-1	UPR-1	9	*black-box*	Normal	27375980544
G-1	UPR-1	10	*black-box*	Normal	30410743808
G-1	UPR-2	1	*black-box*	Normal	3760570368
G-1	UPR-2	2	*black-box*	Normal	6527094784
G-1	UPR-2	3	*black-box*	Normal	8779784192
G-1	UPR-2	4	*black-box*	Normal	10772398080
G-1	UPR-2	5	*black-box*	Normal	13514498048
G-1	UPR-2	6	*black-box*	Normal	15437922304

G-1	UPR-2	7	*black-box*	Normal	17729695744
G-1	UPR-2	8	*black-box*	Normal	20587249664
G-1	UPR-2	9	*black-box*	Normal	22519050240
G-1	UPR-2	10	*black-box*	Normal	24756555776
G-1	UPR-3	1	*black-box*	Normal	3797721088
G-1	UPR-3	2	*black-box*	Normal	6456008704
G-1	UPR-3	3	*black-box*	Normal	8580935680
G-1	UPR-3	4	*black-box*	Normal	10298454016
G-1	UPR-3	5	*black-box*	Normal	12644737024
G-1	UPR-3	6	*black-box*	Normal	14248665088
G-1	UPR-3	7	*black-box*	Normal	16154148864
G-1	UPR-3	8	*black-box*	Normal	18213515264
G-1	UPR-3	9	*black-box*	Normal	19733336064
G-1	UPR-3	10	*black-box*	Normal	21389557760
G-1	UPR-4	1	*black-box*	Normal	4404076544
G-1	UPR-4	2	*black-box*	Normal	7320440832
G-1	UPR-4	3	*black-box*	Normal	9195429888
G-1	UPR-4	4	*black-box*	Normal	10599247872
G-1	UPR-4	5	*black-box*	Normal	12185821184

G-1	UPR-4	6	*black-box*	Normal	13451931648
G-1	UPR-4	7	*black-box*	Normal	15026540544
G-1	UPR-4	8	*black-box*	Normal	16645857280
G-1	UPR-4	9	*black-box*	Normal	17568198656
G-1	UPR-4	10	*black-box*	Normal	19139620864
G-1	UPR-5	1	*black-box*	Normal	4322004992
G-1	UPR-5	2	*black-box*	Normal	7143473152
G-1	UPR-5	3	*black-box*	Normal	9042063360
G-1	UPR-5	4	*black-box*	Normal	10311294976
G-1	UPR-5	5	*black-box*	Normal	11982274560
G-1	UPR-5	6	*black-box*	Normal	13125124096
G-1	UPR-5	7	*black-box*	Normal	14565232640
G-1	UPR-5	8	*black-box*	Normal	16140419072
G-1	UPR-5	9	*black-box*	Normal	17092689920
G-1	UPR-5	10	*black-box*	Normal	18627919872
G-2	UPR-1	1	*black-box*	Terkompresi	1204060160
G-2	UPR-1	2	*black-box*	Terkompresi	2878992384
G-2	UPR-1	3	*black-box*	Terkompresi	4491177984
G-2	UPR-1	4	*black-box*	Terkompresi	5695897600

G-2	UPR-1	5	*black-box*	Terkompresi	7371333632
G-2	UPR-1	6	*black-box*	Terkompresi	8575504384
G-2	UPR-1	7	*black-box*	Terkompresi	10187632640
G-2	UPR-1	8	*black-box*	Terkompresi	11863605248
G-2	UPR-1	9	*black-box*	Terkompresi	13067149312
G-2	UPR-1	10	*black-box*	Terkompresi	14679031808
G-2	UPR-2	1	*black-box*	Terkompresi	1286746112
G-2	UPR-2	2	*black-box*	Terkompresi	2611646464
G-2	UPR-2	3	*black-box*	Terkompresi	3861192704
G-2	UPR-2	4	*black-box*	Terkompresi	4702855168
G-2	UPR-2	5	*black-box*	Terkompresi	6017601536
G-2	UPR-2	6	*black-box*	Terkompresi	6858895360
G-2	UPR-2	7	*black-box*	Terkompresi	8108204032
G-2	UPR-2	8	*black-box*	Terkompresi	9425485824
G-2	UPR-2	9	*black-box*	Terkompresi	10266861568
G-2	UPR-2	10	*black-box*	Terkompresi	11515375616
G-2	UPR-3	1	*black-box*	Terkompresi	1287467008
G-2	UPR-3	2	*black-box*	Terkompresi	2407854080
G-2	UPR-3	3	*black-box*	Terkompresi	3397578752
G-2	UPR-3	4	*black-box*	Terkompresi	3790934016
G-2	UPR-3	5	*black-box*	Terkompresi	4552044544
G-2	UPR-3	6	*black-box*	Terkompresi	5058387968
G-2	UPR-3	7	*black-box*	Terkompresi	5971472384

G-2	UPR-3	8	*black-box*	Terkompresi	6697271296
G-2	UPR-3	9	*black-box*	Terkompresi	7117422592
G-2	UPR-3	10	*black-box*	Terkompresi	7973257216
G-2	UPR-4	1	*black-box*	Terkompresi	1289957376
G-2	UPR-4	2	*black-box*	Terkompresi	2363985920
G-2	UPR-4	3	*black-box*	Terkompresi	3301781504
G-2	UPR-4	4	*black-box*	Terkompresi	3632377856
G-2	UPR-4	5	*black-box*	Terkompresi	4299255808
G-2	UPR-4	6	*black-box*	Terkompresi	4635058176
G-2	UPR-4	7	*black-box*	Terkompresi	5420171264
G-2	UPR-4	8	*black-box*	Terkompresi	6052741120
G-2	UPR-4	9	*black-box*	Terkompresi	6325104640
G-2	UPR-4	10	*black-box*	Terkompresi	7095246848
G-2	UPR-5	1	*black-box*	Terkompresi	1303023616
G-2	UPR-5	2	*black-box*	Terkompresi	2389192704
G-2	UPR-5	3	*black-box*	Terkompresi	3325779968
G-2	UPR-5	4	*black-box*	Terkompresi	3653541888
G-2	UPR-5	5	*black-box*	Terkompresi	4311691264
G-2	UPR-5	6	*black-box*	Terkompresi	4629450752
G-2	UPR-5	7	*black-box*	Terkompresi	5402542080
G-2	UPR-5	8	*black-box*	Terkompresi	6019547136
G-2	UPR-5	9	*black-box*	Terkompresi	6280765440
G-2	UPR-5	10	*black-box*	Terkompresi	7029981184
G-3	UPR-1	1	*white-box*	Normal	2621004434
G-3	UPR-1	2	*white-box*	Normal	6062506341

G-3	UPR-1	3	*white-box*	Normal	9040268137
G-3	UPR-1	4	*white-box*	Normal	11661161981
G-3	UPR-1	5	*white-box*	Normal	15102758107
G-3	UPR-1	6	*white-box*	Normal	17723692909
G-3	UPR-1	7	*white-box*	Normal	20701536631
G-3	UPR-1	8	*white-box*	Normal	24143489098
G-3	UPR-1	9	*white-box*	Normal	26764161754
G-3	UPR-1	10	*white-box*	Normal	29742202084
G-3	UPR-2	1	*white-box*	Normal	2378738322
G-3	UPR-2	2	*white-box*	Normal	4921688421
G-3	UPR-2	3	*white-box*	Normal	6998309737
G-3	UPR-2	4	*white-box*	Normal	8722281981
G-3	UPR-2	5	*white-box*	Normal	11285937371
G-3	UPR-2	6	*white-box*	Normal	13007189869
G-3	UPR-2	7	*white-box*	Normal	15083802999
G-3	UPR-2	8	*white-box*	Normal	17646803018
G-3	UPR-2	9	*white-box*	Normal	19368063706
G-3	UPR-2	10	*white-box*	Normal	21444140260
G-3	UPR-3	1	*white-box*	Normal	2341034642

G-3	UPR-3	2	*white-box*	Normal	3787333989
G-3	UPR-3	3	*white-box*	Normal	4694096745
G-3	UPR-3	4	*white-box*	Normal	4918814205
G-3	UPR-3	5	*white-box*	Normal	5420494043
G-3	UPR-3	6	*white-box*	Normal	5645887341
G-3	UPR-3	7	*white-box*	Normal	6479655287
G-3	UPR-3	8	*white-box*	Normal	6929672266
G-3	UPR-3	9	*white-box*	Normal	7151694554
G-3	UPR-3	10	*white-box*	Normal	7702531300
G-3	UPR-4	1	*white-box*	Normal	2410330770
G-3	UPR-4	2	*white-box*	Normal	3884970341
G-3	UPR-4	3	*white-box*	Normal	4761668457
G-3	UPR-4	4	*white-box*	Normal	4986385917
G-3	UPR-4	5	*white-box*	Normal	5488065755
G-3	UPR-4	6	*white-box*	Normal	5713459053
G-3	UPR-4	7	*white-box*	Normal	6547226999
G-3	UPR-4	8	*white-box*	Normal	6929672266
G-3	UPR-4	9	*white-box*	Normal	7151694554
G-3	UPR-4	10	*white-box*	Normal	7702531300
G-3	UPR-5	1	*white-box*	Normal	2392095378
G-3	UPR-5	2	*white-box*	Normal	3866734949
G-3	UPR-5	3	*white-box*	Normal	4773497705
G-3	UPR-5	4	*white-box*	Normal	4998215165
G-3	UPR-5	5	*white-box*	Normal	5499895003
G-3	UPR-5	6	*white-box*	Normal	5725288301
G-3	UPR-5	7	*white-box*	Normal	6559056247

G-3	UPR-5	8	*white-box*	Normal	6941501514
G-3	UPR-5	9	*white-box*	Normal	7151694554
G-3	UPR-5	10	*white-box*	Normal	7702531300
G-4	UPR-1	1	*white-box*	Terkompresi	1240807567
G-4	UPR-1	2	*white-box*	Terkompresi	2951943547
G-4	UPR-1	3	*white-box*	Terkompresi	4597228788
G-4	UPR-1	4	*white-box*	Terkompresi	5837299094
G-4	UPR-1	5	*white-box*	Terkompresi	7548357313
G-4	UPR-1	6	*white-box*	Terkompresi	8788829016
G-4	UPR-1	7	*white-box*	Terkompresi	10434081495
G-4	UPR-1	8	*white-box*	Terkompresi	12145238001
G-4	UPR-1	9	*white-box*	Terkompresi	13386250362
G-4	UPR-1	10	*white-box*	Terkompresi	15031736310
G-4	UPR-2	1	*white-box*	Terkompresi	1229117489
G-4	UPR-2	2	*white-box*	Terkompresi	2593129083
G-4	UPR-2	3	*white-box*	Terkompresi	3917521302
G-4	UPR-2	4	*white-box*	Terkompresi	4835596762
G-4	UPR-2	5	*white-box*	Terkompresi	6199374947
G-4	UPR-2	6	*white-box*	Terkompresi	7120542876
G-4	UPR-2	7	*white-box*	Terkompresi	8445582269
G-4	UPR-2	8	*white-box*	Terkompresi	9810675253
G-4	UPR-2	9	*white-box*	Terkompresi	10729152096
G-4	UPR-2	10	*white-box*	Terkompresi	12069322110

G-4	UPR-3	1	*white-box*	Terkompresi	1214810161
G-4	UPR-3	2	*white-box*	Terkompresi	2114421371
G-4	UPR-3	3	*white-box*	Terkompresi	2935456150
G-4	UPR-3	4	*white-box*	Terkompresi	3128760794
G-4	UPR-3	5	*white-box*	Terkompresi	3495785571
G-4	UPR-3	6	*white-box*	Terkompresi	3688373404
G-4	UPR-3	7	*white-box*	Terkompresi	4210449341
G-4	UPR-3	8	*white-box*	Terkompresi	4547196469
G-4	UPR-3	9	*white-box*	Terkompresi	4737326688
G-4	UPR-3	10	*white-box*	Terkompresi	5271580030
G-4	UPR-4	1	*white-box*	Terkompresi	1214810161
G-4	UPR-4	2	*white-box*	Terkompresi	2114380411
G-4	UPR-4	3	*white-box*	Terkompresi	2935034262
G-4	UPR-4	4	*white-box*	Terkompresi	3128338906
G-4	UPR-4	5	*white-box*	Terkompresi	3495363683
G-4	UPR-4	6	*white-box*	Terkompresi	3687951516
G-4	UPR-4	7	*white-box*	Terkompresi	4210027453
G-4	UPR-4	8	*white-box*	Terkompresi	4546774581
G-4	UPR-4	9	*white-box*	Terkompresi	4736904800
G-4	UPR-4	10	*white-box*	Terkompresi	5271158142
G-4	UPR-5	1	*white-box*	Terkompresi	1214875697
G-4	UPR-5	2	*white-box*	Terkompresi	2114658939
G-4	UPR-5	3	*white-box*	Terkompresi	2935202198
G-4	UPR-5	4	*white-box*	Terkompresi	3128506842
G-4	UPR-5	5	*white-box*	Terkompresi	3495531619
G-4	UPR-5	6	*white-box*	Terkompresi	3688119452

G-4	UPR-5	7	*white-box*	Terkompresi	4210195389
G-4	UPR-5	8	*white-box*	Terkompresi	4546815541
G-4	UPR-5	9	*white-box*	Terkompresi	4736945760
G-4	UPR-5	10	*white-box*	Terkompresi	5271199102

A.2 Data Waktu Eksekusi Dupremove (Total)

Kelompok Eksperimen	Jumlah VM	Repetisi (Run)	Waktu Real (s)	Waktu User (s)	Waktu Sys (s)
G-1	1	1	19,47	1,15	16,31
G-1	1	2	19,56	1,24	16,36
G-1	1	3	20,64	1,28	17,19
G-1	2	1	207,77	4,68	336,74
G-1	2	2	260,56	4,81	333,03
G-1	2	3	261,76	4,75	333,68
G-1	4	1	571,58	13,35	1012,59
G-1	4	2	578,58	13,4	1012,14
G-1	4	3	586,4	13,47	1021,11
G-1	6	1	951,69	26,95	1758,87
G-1	6	2	955,24	26,8	1771,1
G-1	6	3	952,31	27,12	1747,04
G-1	8	1	1439,14	46,08	2694,06
G-1	8	2	1451,73	47,2	2745,85
G-1	8	3	1453,77	45,13	2767,79
G-1	10	1	1752,97	52,59	3343,47
G-1	10	2	1740,81	48,92	3315,88
G-1	10	3	1753,54	49,44	3348,74
G-2	1	1	45,11	1,66	40,79

G-2	1	2	45,28	1,59	41,12
G-2	1	3	44,94	1,59	40,82
G-2	2	1	1100,01	10,58	1821,1
G-2	2	2	1100,67	10,63	1817,88
G-2	2	3	1419,09	10,49	1803,88
G-2	4	1	2653,24	27,07	4700,96
G-2	4	2	2749,84	27,77	4705,72
G-2	4	3	2641,12	28	4713,69
G-2	6	1	5198,34	57,71	9570,22
G-2	6	2	5233,75	60,13	9606,59
G-2	6	3	5301,23	58,98	9631,3
G-2	8	1	7521,37	98,86	14271,32
G-2	8	2	7687,58	102,88	14571,56
G-2	8	3	7536,01	95,28	14283,3
G-2	10	1	9153,94	111,92	17526,22
G-2	10	2	9069,99	109,03	17373,48
G-2	10	3	9082,56	109,08	17357,69
G-3	1	1	14,48	3,99	13,1
G-3	1	2	14,5	4,07	13,39
G-3	1	3	14,69	3,98	13,11
G-3	2	1	43,48	9,66	46,87
G-3	2	2	43,73	9,7	46,94
G-3	2	3	43,97	9,98	47,07
G-3	4	1	179,62	22,71	188,73
G-3	4	2	187	21,74	192,15
G-3	4	3	189,29	22,8	193,26

G-3	6	1	413,22	38,38	402,24
G-3	6	2	414,41	38,25	412,27
G-3	6	3	418,03	37,96	409,14
G-3	8	1	666,12	54,09	605,79
G-3	8	2	673,59	54,05	614,96
G-3	8	3	667,98	52,89	611,72
G-3	10	1	935,89	71,2	864,22
G-3	10	2	931,06	71,45	822,51
G-3	10	3	923,49	71,51	829,67
G-4	1	1	14,52	4,13	13,45
G-4	1	2	14,89	4,06	13,73
G-4	1	3	15,85	4,2	13,8
G-4	2	1	48,7	9,88	52,71
G-4	2	2	48,01	9,97	52,37
G-4	2	3	48	10,16	52,58
G-4	4	1	163,96	22,17	181,44
G-4	4	2	162,45	22,65	185,32
G-4	4	3	167,25	22,61	185,76
G-4	6	1	347,85	37,1	401,39
G-4	6	2	332,77	37,81	394,36
G-4	6	3	343,76	37,47	396,45
G-4	8	1	510,7	52,84	612,89
G-4	8	2	505,07	53,66	607,88
G-4	8	3	515,93	53,32	609,47
G-4	10	1	718,47	70,93	832,28
G-4	10	2	697,98	70,94	825,45

G-4	10	3	714,62	70,47	822,85

A.3 Data Waktu Eksekusi Dupremove (Pemindaian)

Kelompok Eksperimen	Jumlah VM	Repetisi (Run)	Waktu Real (s)	Waktu User (s)	Waktu Sys (s)
G-1	1	1	9,05	0,76	5,86
G-1	1	2	9,05	0,76	5,87
G-1	1	3	9,05	0,74	5,82
G-1	2	1	11,09	3,18	14,28
G-1	2	2	11,09	3,18	14,23
G-1	2	3	11,09	3,25	14,17
G-1	4	1	14,24	9,8	35,23
G-1	4	2	14,16	9,84	35,94
G-1	4	3	14,17	9,74	35,46
G-1	6	1	19,37	21,12	71,73
G-1	6	2	19,24	21,3	70,72
G-1	6	3	19,26	21,3	71,92
G-1	8	1	25,34	39,9	122,91
G-1	8	2	25,34	39,2	123,73
G-1	8	3	25,33	39,82	122,94
G-1	10	1	34,41	43,23	138,22
G-1	10	2	34,41	42,78	138,49
G-1	10	3	34,43	41,71	139,3
G-2	1	1	11,07	1,53	6,8
G-2	1	2	11,11	1,54	6,68
G-2	1	3	11,07	1,59	6,65
G-2	2	1	13,27	5,23	15,73

G-2	2	2	13,19	5,01	15,49
G-2	2	3	13,18	4,84	15,24
G-2	4	1	17,37	14,52	38,78
G-2	4	2	16,35	14,53	38,46
G-2	4	3	17,51	14,67	39,22
G-2	6	1	22,6	31,56	74,92
G-2	6	2	22,59	33,11	76,99
G-2	6	3	22,73	32,8	76,67
G-2	8	1	28,83	57,97	131,04
G-2	8	2	28,85	57,83	132,1
G-2	8	3	30,1	57,9	132,07
G-2	10	1	40,13	62,02	147,98
G-2	10	2	40,14	63,15	148,63
G-2	10	3	40,08	62,02	149,92
G-3	1	1	15,21	4,66	12,58
G-3	1	2	14,16	4,56	12,21
G-3	1	3	14,14	4,32	11,82
G-3	2	1	29,58	10,31	25,81
G-3	2	2	30,59	10,6	26,14
G-3	2	3	30,59	10,61	26,47
G-3	4	1	63,42	25,59	53,05
G-3	4	2	64,42	24,9	54,5
G-3	4	3	64,52	24,94	54,47
G-3	6	1	105,25	43,95	84,77
G-3	6	2	102,36	43,24	81,44
G-3	6	3	103,03	43,37	82,91

G-3	8	1	145,83	62,41	114,39
G-3	8	2	145,64	62,38	113,37
G-3	8	3	144,86	62,22	113,43
G-3	10	1	197,53	87,88	147,38
G-3	10	2	193,33	87,97	143,02
G-3	10	3	194,31	88,3	144,7
G-4	1	1	14,14	4,58	12,87
G-4	1	2	14,15	4,54	12,87
G-4	1	3	15,2	4,67	13,32
G-4	2	1	31,61	10,96	28,07
G-4	2	2	30,6	10,77	27,22
G-4	2	3	30,61	10,69	27,17
G-4	4	1	64,26	24,25	55,77
G-4	4	2	65,56	24,81	56,57
G-4	4	3	64,51	24,89	55,75
G-4	6	1	104,18	42,81	87,41
G-4	6	2	105,94	42,56	88,69
G-4	6	3	104,28	42,6	86,69
G-4	8	1	148,93	63,28	120,93
G-4	8	2	151,11	64,21	121,06
G-4	8	3	144,93	64,02	115,66
G-4	10	1	201,2	89,92	153,94
G-4	10	2	199,13	89,63	151,82
G-4	10	3	198,84	89,45	151,43

A.4 Data Statistik Duperemove

Kelompok Eksperimen	Kode Skenario	Jumlah VM	Total Berkas	Total Extent	Total Blok	Rata-rata Panjang Extent (Bytes)	Jumlah Extent Kembar	Jumlah Blok Kembar
G-1	UPR-3	1	1	16084	0	167015,39	148	0
G-1	UPR-3	2	2	34329	0	180806,38	11879	0
G-1	UPR-3	3	3	49961	0	184972,1	18884	0
G-1	UPR-3	4	4	65867	0	181090,44	30089	0
G-1	UPR-3	5	5	83095	0	185910,78	39704	0
G-1	UPR-3	6	6	98565	0	183985,35	48707	0
G-1	UPR-3	7	7	114239	0	185307,83	59906	0
G-1	UPR-3	8	8	132068	0	186946,94	70709	0
G-1	UPR-3	9	9	147930	0	185060,37	79857	0
G-1	UPR-3	10	10	163635	0	185844,98	88771	0
G-2	UPR-3	1	1	31736	0	84752,33	323	0
G-2	UPR-3	2	2	70323	0	88353,98	30130	0
G-2	UPR-3	3	3	101214	0	91395,87	49216	0
G-2	UPR-3	4	4	132861	0	89869,22	71449	0
G-2	UPR-3	5	5	171455	0	90188,97	102795	0
G-2	UPR-3	6	6	203139	0	89363,33	123418	0
G-2	UPR-3	7	7	233954	0	90576,55	145703	0
G-2	UPR-3	8	8	272574	0	90689,88	171876	0
G-2	UPR-3	9	9	304152	0	90117,12	193675	0
G-2	UPR-3	10	10	334973	0	90892,9	214934	0
G-2	UPR-4	1	1	31736	81920	84752,33	323	60531

G-2	UPR-4	2	2	70323	163840	88353,98	30130	134277
G-2	UPR-4	3	3	101214	245760	91395,87	49216	204988
G-2	UPR-4	4	4	132861	327680	89869,22	71449	280677
G-2	UPR-4	5	5	171455	409600	90188,97	102795	356717
G-2	UPR-4	6	6	203139	491520	89363,33	123418	431538
G-2	UPR-4	7	7	233954	573440	90576,55	145703	507933
G-2	UPR-4	8	8	272574	655360	90689,88	171876	579682
G-2	UPR-4	9	9	304152	737280	90117,12	193675	656105
G-2	UPR-4	10	10	334973	819200	90892,9	214934	730236
G-1	UPR-4	1	1	16084	81920	167015,39	148	60576
G-1	UPR-4	2	2	34329	163840	180806,38	11879	134120
G-1	UPR-4	3	3	49961	245760	184972,1	18884	204627
G-1	UPR-4	4	4	65867	327680	181090,44	30089	281012
G-1	UPR-4	5	5	83095	409600	185910,78	39704	357514
G-1	UPR-4	6	6	98565	491520	183985,35	48707	433102
G-1	UPR-4	7	7	114239	573440	185307,83	59906	508150
G-1	UPR-4	8	8	132068	6553	186946,94	70709	580612

					60			
G-1	UPR-4	9	9	147930	737280	185060,37	79857	657033
G-1	UPR-4	10	10	163635	819200	185844,98	88771	730572
G-1	UPR-5	1	1	16084	163840	167015,39	148	121827
G-1	UPR-5	2	2	34329	327680	180806,38	11879	270052
G-1	UPR-5	3	3	49961	491520	184972,1	18884	412825
G-1	UPR-5	4	4	65867	655360	181090,44	30089	567791
G-1	UPR-5	5	5	83095	819200	185910,78	39704	724227
G-1	UPR-5	6	6	98565	983040	183985,35	48707	877834
G-1	UPR-5	7	7	114239	1146880	185307,83	59906	1030894
G-1	UPR-5	8	8	132068	1310720	186946,94	70709	1179513
G-1	UPR-5	9	9	147930	1474560	185060,37	79857	1335076
G-1	UPR-5	10	10	163635	1638400	185844,98	88771	1486032
G-2	UPR-5	1	1	31736	163840	84752,33	323	121739
G-2	UPR-5	2	2	70323	327680	88353,98	30130	270817
G-2	UPR-5	3	3	101214	491520	91395,87	49216	413815
G-2	UPR-5	4	4	132861	655360	89869,22	71449	567183

G-2	UPR-5	5	5	171455	819200	90188,97	102795	722128
G-2	UPR-5	6	6	203139	983040	89363,33	123418	875989
G-2	UPR-5	7	7	233954	1146880	90576,55	145703	1031108
G-2	UPR-5	8	8	272574	1310720	90689,88	171876	1178517
G-2	UPR-5	9	9	304152	1474560	90117,12	193675	1333665
G-2	UPR-5	10	10	334973	1638400	90892,9	214934	1484467
G-3	UPR-3	1	147521	7450	0	53829,14	2423	0
G-3	UPR-3	2	278475	15950	0	79885,1	6575	0
G-3	UPR-3	3	387818	23689	0	112548,96	10136	0
G-3	UPR-3	4	503542	31134	0	98514,1	18771	0
G-3	UPR-3	5	621612	39668	0	98342,41	29084	0
G-3	UPR-3	6	736016	47097	0	91344,01	35560	0
G-3	UPR-3	7	842571	54401	0	98241,66	43999	0
G-3	UPR-3	8	959124	62929	0	98187,75	50673	0
G-3	UPR-3	9	1072449	70374	0	93494,37	57202	0
G-3	UPR-3	10	1178330	79525	0	96048,92	63644	0

G-4	UPR-3	1	73762	3924	0	51260,54	18	0
G-4	UPR-3	2	139252	8613	0	71883,78	3284	0
G-4	UPR-3	3	193955	12931	0	91367,31	5174	0
G-4	UPR-3	4	251849	16862	0	82003,08	9388	0
G-4	UPR-3	5	310892	21563	0	83503,09	15000	0
G-4	UPR-3	6	368121	25483	0	78555,18	18614	0
G-4	UPR-3	7	421426	29394	0	85864,39	23027	0
G-4	UPR-3	8	479710	34078	0	86328,03	27357	0
G-4	UPR-3	9	536401	38007	0	82698,64	31000	0
G-4	UPR-3	10	589366	43777	0	84092,21	34491	0
G-3	UPR-4	1	73769	3526	59044	56687,66	16	15930
G-3	UPR-4	2	139233	7337	114616	89277,95	2657	92511
G-3	UPR-4	3	193873	10762	161762	137998,51	4315	132233
G-3	UPR-4	4	251703	14276	211054	118019,08	8773	186513
G-3	UPR-4	5	310730	18109	261363	116014,58	13793	246214
G-3	UPR-4	6	367905	21618	304484	106422,67	16551	288312
G-3	UPR-4	7	421155	25011	348514	112789,77	20647	329506

G-3	UPR-4	8	479 424	28855	3983 41	112195,76	22762	376994
G-3	UPR-4	9	536 058	32371	4409 67	106171,7	25737	418516
G-3	UPR-4	10	588 974	35752	4848 37	110690,99	28749	460627
G-3	UPR-5	1	737 69	3526	7972 3	56687,66	16	23354
G-3	UPR-5	2	139 233	7337	1569 06	89277,95	2657	121773
G-3	UPR-5	3	193 873	10762	2225 87	137998,51	4315	174463
G-3	UPR-5	4	251 703	14276	2897 90	118019,08	8773	247484
G-3	UPR-5	5	310 730	18109	3593 80	116014,58	13793	331372
G-3	UPR-5	6	367 905	21618	4144 09	106422,67	16551	384403
G-3	UPR-5	7	421 155	25011	4744 03	112789,77	20647	436874
G-3	UPR-5	8	479 424	28855	5433 88	112195,76	22762	501281
G-3	UPR-5	9	536 058	32371	5977 15	106171,7	25737	553464
G-3	UPR-5	10	588 974	35752	6575 21	110690,99	28749	609831
G-4	UPR-5	1	737 62	3924	7231 7	51260,54	18	21110
G-4	UPR-5	2	139 252	8613	1434 91	71883,78	3284	110065
G-4	UPR-5	3	193 955	12931	2042 97	91367,31	5174	159168
G-4	UPR-5	4	251	16862	2656	82003,08	9388	224596

			849		37			
G-4	UPR-5	5	310892	21563	329472	83503,09	15000	301996
G-4	UPR-5	6	368121	25483	378466	78555,18	18614	349399
G-4	UPR-5	7	421426	29394	433303	85864,39	23027	402904
G-4	UPR-5	8	479710	34078	496553	86328,03	27357	461821
G-4	UPR-5	9	536401	38007	549149	82698,64	31000	512478
G-4	UPR-5	10	589366	43777	599546	84092,21	34491	555787
G-4	UPR-4	1	73762	3924	51884	51260,54	18	14197
G-4	UPR-4	2	139252	8613	101398	71883,78	3284	81393
G-4	UPR-4	3	193955	12931	143485	91367,31	5174	117008
G-4	UPR-4	4	251849	16862	186792	82003,08	9388	163788
G-4	UPR-4	5	310892	21563	231330	83503,09	15000	216729
G-4	UPR-4	6	368121	25483	268403	78555,18	18614	252978
G-4	UPR-4	7	421426	29394	307259	85864,39	23027	291796
G-4	UPR-4	8	479710	34078	351337	86328,03	27357	333681
G-4	UPR-4	9	536401	38007	390082	82698,64	31000	371427
G-4	UPR-4	10	589366	43777	426661	84092,21	34491	404425

LAMPIRAN B KODE-KODE

B.1 Templat Jinja2 untuk VM

```
<domain type='{{ vm_config.hypervisor | default ('kvm') }}'>


  <metadata>
    {% if 'blockless' in vm_config.tags %}
    <blockless:info
xmlns:blockless="urn:rushingalien:blockless:1.0">
      <blockless:module source="{{ vm_config.os.kmod_src }}"
path="{{ (vm_config.filesystems | selectattr('tag', 'equalto',
'rootfs') | first).path }}/lib/modules/{{ vm_config.os.kmod_src |
basename }}"/>
    </blockless:info>
    {% endif %}
    {% if vm_config.cloudinit_cdrom is defined %}
    <cloudinit:cloudinit
xmlns:cloudinit="urn:rushingalien:cloudinit:1.0">
      <cloudinit:cdrom path='{{ vm_config.cloudinit_cdrom }}'/>
    </cloudinit:cloudinit>
    {% endif %}
    <tag:tag xmlns:tag="urn:rushingalien:tag:1.0">
    {% for tag in vm_config.tags %}
      <tag:name value="{{ tag }}"/>
    {% endfor %}
      <tag:fstype value="{{ vm_config.fstype }}"/>
    </tag:tag>
  </metadata>


  <features>
    <acpi/>
    <apic/>
    <vmport state="off"/>
    <smm state="on"/>
  </features>


  <name>{{ vm_config.name }}</name>
  <memory unit='MiB'>{{ vm_config.mem.size | default (1024)
}}</memory>


  {% if vm_config.mem.shmem | default (false) %}
  <memoryBacking>
    <source type='memfd'/>
    <access mode='shared'/>
  </memoryBacking>
  {% endif %}


  <vcpu placement='static'>{{ vm_config.cpu.vcpus | default
(2) }}</vcpu>
```

```
  <os firmware='{{ vm_config.os.firmware | default ('efi') }}'>
    <type arch='{{ vm_config.arch | default ('x86_64') }}' machine='{{ vm_config.machine | default ('pc-q35-9.2') }}'>hvm</type>
    {% if vm_config.os.direct_boot | default (false) %}
    <kernel>{{ vm_config.os.kernel }}</kernel>
    <initrd>{{ vm_config.os.initrd }}</initrd>
    <cmdline>{{ vm_config.os.cmdline }}</cmdline>
    {%endif%}
  </os>
  {% if vm_config.cpu.mode | default('host-passthrough') == 'custom' %}
    {{ vm_config.cpu.xml }}
  {% else %}
  <cpu mode='{{ vm_config.cpu.mode | default ('host-passthrough') }}' check='none' {% if vm_config.cpu.mode | default('host-passthrough') == 'host-passthrough' %}migratable='{% if vm_config.cpu.migratable | default (true) %}on{% else %}off{% endif %}'{% endif %}/>
  {% endif %}
  <devices>
    {% if vm_config.guest_agent | default (true) %}
    <channel type="unix">
      <source mode="bind"/>
      <target type="virtio" name="org.qemu.guest_agent.0"/>
    </channel>
    {% endif %}
    <emulator>{{ vm_config.emulator | default ('/usr/bin/qemu-system-x86_64') }}</emulator>
    {% if vm_config.cloudinit_cdrom is defined %}
    <disk type='file' device='cdrom'>
      <driver name='qemu' type='raw'/>
      <source file="{{ vm_config.cloudinit_cdrom }}"/>
      <target dev='sda' bus='sata'/>
      <readonly/>
    </disk>
    {% endif %}
    {% if vm_config.disks is defined %}
    {% for disk in vm_config.disks %}
    <disk type='{{ disk.type }}' device='{{ disk.device | default ('disk') }}'>
      <driver name='qemu' type='{{ disk.driver.type }}' io='{{ disk.driver.io | default ('io_uring') }}' discard='{{ disk.driver.discard | default ('ignore')}}'/>
      <source file='{{ disk.path }}'/>
      <target dev='{{ disk.target.dev }}' bus='{{ disk.target.bus | default("virtio") }}' {% if disk.target.emulate_ssd | default (false) %}rotation_rate='1'{% endif %}/>
      {% if disk.boot_order is defined %}
      <boot order='{{ disk.boot_order }}'/>
      {% endif %}
    </disk>
    {% endfor %}
```

```
80      {% endif %}
81      {% if vm_config.filesystems is defined %}
82      {% for fs in vm_config.filesystems %}
83      <filesystem type='{{ fs.type | default ('mount') }}'
accessmode='{{ fs.accessmode | default ('passthrough') }}'>
84        <driver type='{{ fs.driver.type | default ('virtiofs') }}'
{% if fs.driver.queue is defined %}queue='{{ fs.driver.queue}}'{%
endif %}/>
85        <binary path='{{ fs.virtiofsd.binary | default
('/usr/libexec/virtiofsd') }}' xattr='{{ fs.virtiofsd.xattr |
default ('on') }}'/>
86          {# use implied defaults if not defined #}
87          {% if fs.virtiofsd.cache is defined %}
88          <cache mode='{{ fs.virtiofsd.cache }}'/>
89          {% endif %}
90          {% if fs.virtiofsd.sandbox is defined %}
91          <sandbox mode='{{ fs.virtiofsd.sandbox }}'/>
92          {% endif %}
93          {% if fs.virtiofsd.lock is defined %}
94          <lock posix='{{ fs.virtiofsd.lock.posix }}'
flock='{{ fs.virtiofsd.lock.flock }}'/>
95          {% endif %}
96          {% if fs.virtiofsd.thread_pool is defined %}
97          <thread_pool size='{{ fs.virtiofsd.thread_pool }}'/>
98          {% endif %}
99        <source dir="{{ fs.path }}"/>
100       <target dir='{{ fs.tag }}'/>
101     </filesystem>
102     {% endfor %}
103     {% endif %}
104     {% if vm_config.interfaces is defined %}
105     {% for iface in vm_config.interfaces %}
106     <interface type='network'>
107       <source network='{{ iface.network }}'/>
108       <model type='{{ iface.model }}'/>
109       {% if iface.vlan_tag is defined %}
110       <vlan>
111         <tag id='{{ iface.vlan_tag }}'/>
112       </vlan>
113       {% endif %}
114     </interface>
115     {% endfor %}
116     {% endif %}
117     {% if vm_config.insert_devices is defined %}
118     {% for device in vm_config.insert_devices %}
119       {{ device }}
120     {% endfor %}
121     {% endif %}
122     <serial type='pty'>
123       <target type='isa-serial' port='0'>
```

```
124         <model name='isa-serial'/>
125       </target>
126     </serial>
127     <console type='pty'>
128       <target type='serial' port='0'/>
129     </console>
130     <console type="pty">
131       <target type="virtio"/>
132     </console>
133     <channel type='spicevmc'>
134       <target type='virtio' name='com.redhat.spice.0'/>
135       <address type='virtio-serial' controller='0' bus='0'
    port='1'/>
136     </channel>
137     <input type='tablet' bus='usb'>
138       <address type='usb' bus='0' port='1'/>
139     </input>
140     <input type='mouse' bus='ps2'/>
141     <input type='keyboard' bus='ps2'/>
142     <graphics type='spice'>
143       <listen type='socket'/>
144       <image compression='off'/>
145     </graphics>
146     <sound model='ich9'>
147     </sound>
148     <audio id='1' type='spice'/>
149     <video>
150       <model type='virtio' heads='1' primary='yes'/>
151     </video>
152     <redirdev bus='usb' type='spicevmc'>
153     </redirdev>
154     <redirdev bus='usb' type='spicevmc'>
155     </redirdev>
156     <watchdog model='itco' action='reset'/>
157     <memballoon model='virtio'>
158     </memballoon>
159     <rng model='virtio'>
160       <backend model='random'>/dev/urandom</backend>
161     </rng>
162   </devices>
163 </domain>
```

B.2 Profil Pertumbuhan

```
1   # vars/growth_simulation.yml
2
3   # --- Define three distinct server archetypes with your requested
    actions ---
4   growth_profiles:
```

```
5
6       # Profile A: Web/API Server
7       A:
8         daily_schedule:
9           - days: "1-2" # Initial Setup
10            actions:
11              - { op: 'install_package', name: ['nginx', 'git',
    'fio'] }
12              - { op: 'git_clone', repo:
    'https://github.com/containers/podman.git', dest:
    '/root/podman_src' }
13          - days: "3-5" # Active Workload
14            actions:
15              - { op: 'log_simulation', path:
    '/var/log/nginx/access.log', size: '10M' } # Simulate log growth
16              - { op: 'user_uploads', path: '/srv/www/uploads/',
    total_size: '100M', num_files: 50 } # Simulate user file uploads
17          - days: "5-7" # Maintenance & Continued Use
18            actions:
19              - { op: 'dnf_update' } # Run package updates
20              - { op: 'log_simulation', path:
    '/var/log/nginx/access.log', size: '15M' } # More logs
21
22      # Profile B: CI/CD Build Server
23      B:
24        daily_schedule:
25          - days: "1-2" # Initial Setup
26            actions:
27              - { op: 'install_package', name: ['java-21-openjdk-
    devel', 'maven', 'git'] }
28              - { op: 'git_clone', repo: 'https://github.com/spring-
    projects/spring-petclinic.git', dest: '/root/spring-petclinic' }
29          - days: "3" # Active Workload
30            actions:
31              - { op: 'build_java_project', path: '/root/spring-
    petclinic' } # Creates many .class and .jar files
32          - days: "4" # Maintenance & Rerunning Builds
33            actions:
34              - { op: 'build_java_project', path: '/root/spring-
    petclinic' } # Re-running a build changes some artifacts
35          - days: "3-7"
36            actions:
37              - { op: 'dnf_update' }
38              - { op: 'random_data', path:
    '/opt/build_artifacts/test_data.dat', size: '250M' } # Simulate
    large test datasets
39
40
41      # Profile C: General Purpose / Utility Server
42      C:
43        daily_schedule:
44          - days: "1-3" # Initial Setup & Light Use
```

```
45          actions:
46            - { op: 'install_package', name: ['python3-pip',
    'htop', 'tmux', 'git', 'fio'] }
47            - { op: 'git_clone', repo:
    'https://github.com/ansible/ansible.git', dest:
    '/root/ansible_src' } # Different git repo
48            - { op: 'log_simulation', path: '/root/log/messages',
    size: '20M' }
49        - days: "3-7" # Maintenance & Data Storage
50          actions:
51            - { op: 'dnf_update' }
52            - { op: 'random_data', path:
    '/data/backups/random_archive.dat', size: '400M' }
53
54
55  # --- Map the 10 VM suffixes to the profiles ---
56  # This simulates a realistic environment with multiple servers of
    the same type
57  profile_map:
58    '01': 'A' # Web Server 1
59    '02': 'B' # Build Server 1
60    '03': 'C' # Utility Server 1
61    '04': 'A' # Web Server 2
62    '05': 'B' # Build Server 2
63    '06': 'A' # Web Server 3
64    '07': 'C' # Utility Server 2
65    '08': 'B' # Build Server 3
66    '09': 'A' # Web Server 4
67    '10': 'C' # Utility Server 3
```

B.3 Playbook Provisi VM

```
1   ---
2   - name: Provision Libvirt VMs
3     tags: provision
4     hosts: main
5     become: true
6     vars:
7       jinja_render_dir: /var/lib/vm-jinja
8       kernel_ver: 6.14.0-63.fc42.x86_64
9
10      image_pool: /mnt/virt-dedupe/images
11      fs_pool: /mnt/virt-dedupe/filesystems
12      kernel_pool: /mnt/virt-dedupe/kernels
13
14      image_pool_comp: /mnt/virt-dedupe-comp/images
15      fs_pool_comp: /mnt/virt-dedupe-comp/filesystems
16      kernel_pool_comp: /mnt/virt-dedupe-comp/kernels
17
18      libvirt_uri: qemu:///system
19      # base_image_name_lvm: fedoraserver42-lvm.img
20      # base_image_name_xfs: fedoraserver42-xfs.img
```

```
21      base_image_name_ext4: fedoraserver42-ext4.img
22      base_subvol_name: fedoraserver42
23
24      # base_image_lvm: "{{ image_pool }}/{{ base_image_name_lvm }}"
25      # base_image_xfs: "{{ image_pool }}/{{ base_image_name_xfs }}"
26      base_image_ext4: "{{ image_pool }}/{{ base_image_name_ext4 }}"
27      base_subvol: "{{ fs_pool }}/{{ base_subvol_name }}"
28
29      # base_image_lvm_comp: "{{ image_pool_comp }}/{{ base_image_name_lvm }}"
30      # base_image_xfs_comp: "{{ image_pool_comp }}/{{ base_image_name_xfs }}"
31      base_image_ext4_comp: "{{ image_pool_comp }}/{{ base_image_name_ext4 }}"
32      base_subvol_comp: "{{ fs_pool_comp }}/{{ base_subvol_name }}"
33
34      tmp_dir_base: /tmp/ansible_cidata
35      vm_template: '../resources/templates/vms/golden.xml.j2'
36
37      # --- Cloudinit data vars ---
38      common_cloudinit_data:
39        iface: enp2s0
40        dhcp4: true
41        username: labuser
42        user_groups: "wheel,adm"
43        shell: /bin/bash
44        ssh_pub_key: "{{ ssh_secret }}"
45        password: "{{ password_secret }}"
46
47      # default_cloudinit_iface: enp2s0
48      # default_cloudinit_dhcp4: true
49      # default_cloudinit_username: labuser
50      # default_cloudinit_user_groups: "wheel,adm"
51      # default_cloudinit_shell: /bin/bash
52      # default_cloudinit_ssh_pub_key: "{{ ssh_secret }}"
53      # default_cloudinit_password: "{{ password_secret }}"
54
55      # --- VM Definitions (New High-Compatibility Syntax) ---
56      # Step 1: Generate the list of VM names using stable filters
57      blockfull_vm_names:
58        - blockfull-vm-01
59        - blockfull-vm-02
60        - blockfull-vm-03
61        - blockfull-vm-04
62        - blockfull-vm-05
63        - blockfull-vm-06
64        - blockfull-vm-07
65        - blockfull-vm-08
66        - blockfull-vm-09
67        - blockfull-vm-10
68      blockless_vm_names:
69        - blockless-vm-01
70        - blockless-vm-02
71        - blockless-vm-03
```

```
72        - blockless-vm-04
73        - blockless-vm-05
74        - blockless-vm-06
75        - blockless-vm-07
76        - blockless-vm-08
77        - blockless-vm-09
78        - blockless-vm-10
79
80      blockfull_vm_names_comp:
81        - blockfull-vm-compressed-01
82        - blockfull-vm-compressed-02
83        - blockfull-vm-compressed-03
84        - blockfull-vm-compressed-04
85        - blockfull-vm-compressed-05
86        - blockfull-vm-compressed-06
87        - blockfull-vm-compressed-07
88        - blockfull-vm-compressed-08
89        - blockfull-vm-compressed-09
90        - blockfull-vm-compressed-10
91      blockless_vm_names_comp:
92        - blockless-vm-compressed-01
93        - blockless-vm-compressed-02
94        - blockless-vm-compressed-03
95        - blockless-vm-compressed-04
96        - blockless-vm-compressed-05
97        - blockless-vm-compressed-06
98        - blockless-vm-compressed-07
99        - blockless-vm-compressed-08
100       - blockless-vm-compressed-09
101       - blockless-vm-compressed-10
102
103     # --- Common VM config
104     common_config:
105       cpu:
106         vcpus: 2
107       guest-agent: true
108       mem:
109         size: 1024
110         shmem: true
111       interfaces:
112         - network: default
113           model: virtio
114       template: ../resources/templates/vms/golden.xml.j2
115
116     common_fs_config:
117       tag: rootfs
118       driver:
119         xattr: "on"
120       virtiofsd:
121         binary: /usr/local/bin/virtiofsd
122
123     common_disk_config:
```

```
      type: file
      driver:
        type: raw
        io: io_uring
        discard: unmap
      target:
        dev: vda
        bus: virtio

    # --- Config Generation
    vms_config: |-
      {# --- BLOCKFULL VMs ------------------------------------- #}
      {% set blockfull_vms = [] %}
      {% for vm_name in blockfull_vm_names %}
      {% set _ = blockfull_vms.append({
          "name": vm_name,
          "fstype": "ext4",
          "type": "blockfull",
          "tags": ["blockfull"],
          "base_image": base_image_ext4,
          "cloudinit_cdrom": image_pool ~ '/cinit/' ~ vm_name ~ '-cinit.iso',
          "disks": [
            {
              "path": image_pool ~ '/instances/vm-' ~ (loop.index | format('%02d')) ~ '.img',
              "boot_order": 1
            } | combine(common_disk_config)
          ],
          "os": {
            "direct_boot": true,
            "kernel": kernel_pool ~ '/lib/modules/' ~ kernel_ver ~ '/vmlinuz',
            "initrd": kernel_pool ~ '/' ~ kernel_ver ~ '/initramfs.img',
            "cmdline": "rootfstype=ext4 root=UUID=03974df5-6efc-4500-97e3-27911fb96971 rw selinux=0"
          },
          "cloudinit_data":{
              'instance_id': vm_name,
              'hostname': vm_name,
          } | combine( common_cloudinit_data )
        } | combine(common_config, recursive=true)) %}
      {% endfor %}
      {# --- VIRTIOFS VMs ---------------------------------- #}
      {% set blockless_vms = [] %}
      {% for vm_name in blockless_vm_names %}
      {% set _ = blockless_vms.append({
          "name": vm_name,
          "fstype": "virtiofs",
          "type": "blockless",
          "tags": ["blockless"],
          "cloudinit_cdrom": image_pool ~ '/cinit/' ~ vm_name ~ '-cinit.iso',
          "base_image":  base_subvol,
          "filesystems": [
```

```
            {
              "path": fs_pool ~ '/instances/vm-' ~ (loop.index |
format('%02d'))
            } | combine(common_fs_config)
          ],
          "os": {
            "direct_boot": true,
            "kmod_src": kernel_pool ~ '/lib/modules/' ~
kernel_ver,
            "kernel": kernel_pool ~ '/lib/modules/' ~ kernel_ver ~
'/vmlinuz',
            "initrd": kernel_pool ~ '/' ~ kernel_ver ~
'/initramfs.img',
            "cmdline": "rootfstype=virtiofs root=rootfs rw
selinux=0"
          },
          "cloudinit_data":{
              'instance_id': vm_name,
              'hostname': vm_name,
          } | combine( common_cloudinit_data )
        } | combine(common_config, recursive=true)) %}
      {% endfor %}
      {# --- EXT4 VMs ---------------------------------------- #}
      {% set blockfull_vms_comp = [] %}
      {% for vm_name in blockfull_vm_names_comp %}
      {% set _ = blockfull_vms_comp.append({
          "name": vm_name,
          "fstype": "ext4",
          "type": "blockfull",
          "tags": ["blockfull"],
          "base_image": base_image_ext4_comp,
          "cloudinit_cdrom": image_pool_comp ~ '/cinit/' ~ vm_name
~ '-cinit.iso',
          "disks": [
            {
              "path": image_pool_comp ~ '/instances/vm-' ~
(loop.index | format('%02d')) ~ '.img',
              "boot_order": 1
            } | combine(common_disk_config)
          ],
          "os": {
            "direct_boot": true,
            "kernel": kernel_pool_comp ~ '/lib/modules/' ~
kernel_ver ~ '/vmlinuz',
            "initrd": kernel_pool_comp ~ '/' ~ kernel_ver ~
'/initramfs.img',
            "cmdline": "rootfstype=ext4 root=UUID=03974df5-
6efc-4500-97e3-27911fb96971 rw selinux=0"
          },
          "cloudinit_data":{
              'instance_id': vm_name,
              'hostname': vm_name,
          } | combine( common_cloudinit_data )
        } | combine(common_config, recursive=true)) %}
      {% endfor %}
      {# --- VIRTIOFS VMs ------------------------------------ #}
      {% set blockless_vms_comp = [] %}
      {% for vm_name in blockless_vm_names_comp %}
      {% set _ = blockless_vms_comp.append({
```

```
223           "name": vm_name,
224           "fstype": "virtiofs",
225           "type": "blockless",
226           "tags": ["blockless"],
              "cloudinit_cdrom": image_pool_comp ~ '/cinit/' ~ vm_name
227   ~ '-cinit.iso',
228           "base_image":  base_subvol_comp,
229           "filesystems": [
230             {
                "path": fs_pool_comp ~ '/instances/vm-' ~
231   (loop.index | format('%02d'))
232             } | combine(common_fs_config)
233           ],
234           "os": {
235             "direct_boot": true,
              "kmod_src": kernel_pool_comp ~ '/lib/modules/' ~
236   kernel_ver,
              "kernel": kernel_pool_comp ~ '/lib/modules/' ~
237   kernel_ver ~ '/vmlinuz',
              "initrd": kernel_pool_comp ~ '/' ~ kernel_ver ~
238   '/initramfs.img',
              "cmdline": "rootfstype=virtiofs root=rootfs rw
239   selinux=0"
240           },
241           "cloudinit_data":{
242               'instance_id': vm_name,
243               'hostname': vm_name,
244           } | combine( common_cloudinit_data )
245         } | combine(common_config, recursive=true)) %}
246       {% endfor %}
          {{ blockless_vms + blockfull_vms + blockfull_vms_comp +
247   blockless_vms_comp }}
248
249     tasks:
250       - name: Get list of existing libvirt VMs
251         community.libvirt.virt:
252           command: list_vms
253           uri: "{{ libvirt_uri }}"
254         register: existing_vms_result
255         changed_when: false
256
257       - name: Prepare Cloud-Init data and generate ISOs
258         loop: "{{ vms_config }}"
259         loop_control:
260           loop_var: vm_item
261           label: "{{ vm_item.name }}"
262         become: true
263         vars:
264           vm_name: "{{ vm_item.name }}"
265           vm_tmp_dir: "{{ tmp_dir_base }}/{{ vm_name }}"
            iso_path: "{{ (image_pool_comp if 'compressed' in vm_name
266   else image_pool) ~ '/cinit/' ~ vm_name ~ '-cinit.iso' }}"
267
268           template_context: "{{ vm_item.cloudinit_data }}"
269
270           cloud_init_content:
              meta-data: "{{ lookup('template',
271   'resources/templates/cloud-init/meta-data.j2',
```

```
template_vars=template_context) }}"
          user-data: "{{ lookup('template',
'resources/templates/cloud-init/user-data.j2',
template_vars=template_context) }}"
          network-config: "{{ lookup('template',
'resources/templates/cloud-init/network-config.j2',
template_vars=template_context) }}"

      generate_cloudinit_iso:
        vm_name: "{{ vm_name }}"
        tmp_dir: "{{ vm_tmp_dir }}"
        iso_path: "{{ iso_path }}"
        file_content: "{{ cloud_init_content }}"
        cleanup: true

    - name: Make boot disks
      become: true
      loop: >-
        {{
          vms_config |
          selectattr("type", "equalto", "blockfull")
        }}
      loop_control:
        loop_var: vm_config
        label: "{{ boot_disk.path }}"
      vars:
        boot_disk: "{{ vm_config.disks | selectattr('boot_order',
'equalto', 1) | first }}"
      ansible.builtin.command: >
        cp --reflink=always {{ vm_config.base_image }}
{{ boot_disk.path }}
      args:
        creates: "{{ boot_disk.path }}"

    - name: Gather btrfs_info
      become: true
      community.general.btrfs_info:
      register: btrfs_info

    - name: Make rootfs
      become: true
      loop: >-
        {{
          vms_config |
          selectattr('type', 'equalto', 'blockless')
        }}
      loop_control:
        loop_var: vm_config
        label: "{{ rootfs.path }}"
      vars:
        rootfs: "{{ vm_config.filesystems | selectattr('tag',
'equalto', 'rootfs') | first }}"
      ansible.builtin.command:
        argv:
          - btrfs
          - subvolume
```

```
320           - snapshot
321           - "{{ vm_config.base_image }}"
322           - "{{ rootfs.path }}"
323         creates: "{{ rootfs.path }}"
324
325     - name: Make kernel module subvol
326       become: true
327       loop: >-
328         {{
329           vms_config |
330           selectattr('type', 'equalto', 'blockless')
331         }}
332       loop_control:
333         loop_var: vm_config
334         label: "{{ rootfs.path }}"
335       vars:
          rootfs: "{{ vm_config.filesystems | selectattr('tag',
336 'equalto', 'rootfs') | first }}"
337       ansible.builtin.command:
338         argv:
339           - btrfs
340           - subvolume
341           - snapshot
342           - "{{ vm_config.os.kmod_src }}"
343           - "{{ rootfs.path ~ '/lib/modules/' }}"
          creates: "{{ rootfs.path ~ '/lib/modules/' ~
344 kernel_ver }}"
345
346     - name: Provision VMs
347       loop: "{{ vms_config }}"
348       loop_control:
349         loop_var: vm_config
350         label: "{{ vm_config.name }}"
351       community.libvirt.virt:
352         command: define
353         uri: "{{ libvirt_uri }}"
354         xml: "{{ lookup('template', vm_config.template) }}"
```