



UNIVERSITAS INDONESIA

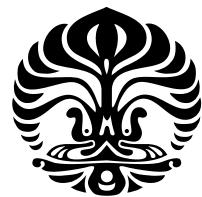
**ANALISIS KINERJA TUNING HYPERVISOR KVM DENGAN
CLOUDSTACK AGENT PADA LINGKUNGAN VIRTUAL MACHINE DI
CLOUD MANAGEMENT PLATFORM APACHE CLOUDSTACK**

SKRIPSI

ARIQ PRADIPA SANTOSO

2006527052

**FAKULTAS TEKNIK
PROGRAM STUDI TEKNIK KOMPUTER
DEPOK
2024**



UNIVERSITAS INDONESIA

**ANALISIS KINERJA TUNING HYPERVISOR KVM DENGAN
CLOUDSTACK AGENT PADA LINGKUNGAN VIRTUAL MACHINE DI
CLOUD MANAGEMENT PLATFORM APACHE CLOUDSTACK**

SKRIPSI

**Diajukan sebagai salah satu syarat untuk memperoleh gelar
Sarjana Teknik**

**ARIQ PRADIPA SANTOSO
2006527052**

**FAKULTAS TEKNIK
PROGRAM STUDI TEKNIK KOMPUTER
DEPOK
MEI 2024**

HALAMAN PERSETUJUAN

Judul : Analisis Kinerja Tuning Hypervisor KVM dengan *CloudStack Agent* pada Lingkungan *Virtual Machine* di *Cloud Management Platform* Apache CloudStack

Nama : ARIQ PRADIPA SANTOSO

NPM : 2006527052

Laporan Skripsi ini telah diperiksa dan disetujui.

Desember 2023

Yan Maraden S.T., M.T.

Pembimbing Skripsi

HALAMAN PERNYATAAN ORISINALITAS

**Skripsi ini adalah hasil karya saya sendiri,
dan semua sumber baik yang dikutip maupun dirujuk
telah saya nyatakan dengan benar.**

Nama : Ariq Pradipa Santoso

NPM : 2006527052

Tanda Tangan :

Tanggal : Desember 2023

HALAMAN PENGESAHAN

Skripsi ini diajukan oleh :

Nama : ARIQ PRADIPA SANTOSO
NPM : 2006527052
Program Studi : Teknik Komputer
Judul Skripsi : Analisis Kinerja Tuning Hypervisor KVM dengan *CloudStack Agent* pada Lingkungan *Virtual Machine* di *Cloud Management Platform* Apache CloudStack

Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana Teknik pada Program Studi Teknik Komputer, Fakultas Teknik, Universitas Indonesia.

DEWAN PENGUJI

Pembimbing : Yan Maraden S.T., M.T. ()

Penguji : ()

Penguji : ()

Penguji : ()

Ditetapkan di : Depok

Tanggal : Desember 2023

KATA PENGANTAR

Dengan mengucap syukur kehadirat Allah SWT, saya dapat menyelesaikan skripsi ini sebagai salah satu syarat untuk menyelesaikan program studi Teknik Komputer di Universitas Indonesia. Penulisan skripsi ini tidak lepas dari dukungan dan bantuan berbagai pihak, yang dengan tulus saya ucapkan terima kasih.

Ucapan terima kasih yang sebesar-besarnya saya persembahkan kepada kedua orang tua saya, yang telah memberikan doa, dukungan moral, dan finansial selama proses pembuatan skripsi ini. Tak lupa kepada dosen pembimbing, Bapak Yan Maraden, S.T., M.T. atas bimbingan, arahan, dan kritik yang membangun selama proses penulisan skripsi ini.

Saya juga ingin mengucapkan terima kasih kepada teman-teman di Teknik Komputer Universitas Indonesia yang telah memberikan semangat dan dukungan. Pengalaman dan ilmu yang saya peroleh selama proses pembelajaran di universitas ini sangat berharga dan menjadi motivasi dalam penyelesaian skripsi ini.

Skripsi ini berjudul Analisis Kinerja Tuning Hypervisor KVM dengan *Cloud-Stack Agent* pada Lingkungan *Virtual Machine* di *Cloud Management Platform* Apache CloudStack yang merupakan upaya saya untuk memberikan kontribusi dalam bidang cloud serta sebagai aplikasi dari teori yang telah saya pelajari. Saya berharap skripsi ini dapat bermanfaat bagi pengembangan ilmu pengetahuan, khususnya bagi para peneliti dan mahasiswa yang berkecimpung dalam bidang yang sama.

Akhir kata, semoga skripsi ini dapat memberikan manfaat dan inspirasi bagi kita semua. Kritik dan saran yang membangun selalu saya harapkan untuk perbaikan di masa yang akan datang.

Depok, 29 Desember 2023

Ariq Pradipa Santoso

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Universitas Indonesia, saya yang bertanda tangan di bawah ini:

Nama : Ariq Pradipa Santoso
NPM : 2006527052
Program Studi : Teknik Komputer
Fakultas : Teknik
Jenis Karya : Skripsi

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

Analisis Kinerja Tuning Hypervisor KVM dengan *CloudStack Agent* pada Lingkungan *Virtual Machine* di *Cloud Management Platform Apache CloudStack* beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Non-eksklusif ini Universitas Indonesia berhak menyimpan, mengalihmedia/formatkan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan memublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Depok
Pada tanggal : Desember 2023
Yang menyatakan

(Ariq Pradipa Santoso)

ABSTRAK

Nama : Ariq Pradipa Santoso
Program Studi : Teknik Komputer
Judul : Analisis Kinerja Tuning Hypervisor KVM dengan *Cloud-Stack Agent* pada Lingkungan *Virtual Machine* di *Cloud Management Platform* Apache CloudStack
Pembimbing : Yan Maraden S.T., M.T.

Penelitian ini bertujuan untuk menganalisis performa dari tuning Hypervisor Kernel-based Virtual Machine (KVM) dengan fokus pada Streaming SIMD Extensions (SSE) dalam konteks kompresi video pada Virtual Machines (VMs) yang berjalan di Apache CloudStack. Kajian ini secara khusus meneliti pengaruh penambahan flag SSE3, SSE4.1, dan SSE4.2 terhadap performa kompresi video. Dua aspek utama yang menjadi fokus adalah kecepatan kompresi dan kualitas video yang dihasilkan.

Untuk menilai kecepatan kompresi, penelitian ini membandingkan waktu kompresi antara Hypervisor KVM yang telah di-tuning dengan flag SSE tambahan dan Hypervisor KVM default. Metode yang digunakan untuk mengukur kecepatan ini adalah dengan menghitung waktu yang diperlukan untuk kompresi video pada kedua konfigurasi hypervisor tersebut.

Di sisi lain, kualitas video yang dikompresi dievaluasi menggunakan metrik Structural Similarity Index Measure (SSIM). SSIM digunakan untuk mengukur kesamaan visual antara video asli dan video hasil kompresi, memberikan gambaran tentang seberapa baik video dikompresi tanpa kehilangan kualitas visual.

Penelitian ini menghasilkan temuan penting mengenai efektivitas tuning SSE pada Hypervisor KVM dalam konteks Apache CloudStack, memberikan wawasan baru tentang optimisasi performa untuk kompresi video di lingkungan cloud. Hasil ini diharapkan dapat membantu dalam pengembangan praktik terbaik untuk konfigurasi Hypervisor KVM dalam meningkatkan efisiensi dan efektivitas kompresi video pada VM.

Kata kunci:

Tuning Hypervisor KVM, Kompresi Video, Cloud Computing, Apache CloudStack

ABSTRACT

Name : Arij Pradipa Santoso
Study Program : Teknik Komputer
Title : Performance Analysis of KVM Hypervisor Tuning with CloudStack Agent for Virtual Machine Environment on Cloud Management Platform Apache CloudStack
Counsellor : Yan Maraden S.T., M.T.

This study aims to analyze the performance of tuning the Kernel-based Virtual Machine (KVM) Hypervisor, focusing on Streaming SIMD Extensions (SSE) in the context of video compression on Virtual Machines (VMs) running in Apache CloudStack. Specifically, this research examines the impact of adding SSE3, SSE4.1, and SSE4.2 flags on video compression performance. The main focuses are the speed of compression and the quality of the resulting video.

To assess compression speed, this study compares the compression time between the KVM Hypervisor tuned with additional SSE flags and the default KVM Hypervisor. The method used to measure this speed involves calculating the time required for video compression on both hypervisor configurations.

On the other hand, the quality of the compressed video is evaluated using the Structural Similarity Index Measure (SSIM) metric. SSIM is used to measure the visual similarity between the original video and the compressed video, providing an insight into how well the video is compressed without losing visual quality.

This research yields significant findings regarding the effectiveness of SSE tuning on the KVM Hypervisor in the context of Apache CloudStack, offering new insights into performance optimization for video compression in cloud environments. These results are expected to assist in developing best practices for configuring the KVM Hypervisor to enhance efficiency and effectiveness in video compression on VMs.

Key words:

KVM Hypervisor Tuning, Video Compression, Cloud Computing, Apache CloudStack

DAFTAR ISI

HALAMAN JUDUL	i
LEMBAR PERSETUJUAN	ii
LEMBAR PERNYATAAN ORISINALITAS	iii
LEMBAR PENGESAHAN	iv
KATA PENGANTAR	v
LEMBAR PERSETUJUAN PUBLIKASI ILMIAH	v
ABSTRAK	vii
DAFTAR ISI	ix
DAFTAR GAMBAR	xii
DAFTAR TABEL	xv
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Permasalahan	2
1.2.1 Definisi Permasalahan	2
1.2.2 Batasan Permasalahan	2
1.3 Tujuan	3
1.4 Metodologi Penelitian	3
1.5 Sistematika Penulisan	5
2 TUNING HYPERVISOR KVM	6
2.1 <i>Cloud Computing</i>	6
2.2 Apache CloudStack	7
2.3 Hypervisor	8
2.4 KVM (<i>Kernel-Based Virtual Machine</i>)	9
2.5 <i>Virtual Machine</i>	10
2.6 <i>Advanced Encryption Standard (AES)</i>	11

2.7	<i>Cyclic Redundancy Check (CRC)</i>	12
2.8	<i>virsh</i>	13
2.9	<i>Streaming SIMD Extensions</i>	13
2.9.1	SSE2	14
2.9.2	SSE3	15
2.9.3	SSSE3	16
2.9.4	SSE4	16
3	METODE PENGUJIAN TUNING PARAMETR HYPERVISOR KVM DENGAN CLOUDSTACK AGENT	18
3.1	Tahapan Penelitian	18
3.2	Persiapan Lingkungan untuk Pengujian dan Analisis	19
3.2.1	Instalasi dan Konfigurasi Apache Cloudstack	20
3.2.1.1	Pengaturan Jaringan	20
3.2.1.2	Instalasi Cloudstack	22
3.2.1.3	Konfigurasi Database	22
3.2.1.4	Konfigurasi Storage	23
3.2.1.5	Konfigurasi KVM	24
3.2.1.6	Konfigurasi Firewall	27
3.2.1.7	Menjalankan CloudStack	29
3.2.2	Persiapan <i>Virtual Machine</i>	30
3.2.2.1	Image Ubuntu Server 22.04 LTS	30
3.2.2.2	Membuat Image Template	31
3.3	Tuning Hypervisor KVM	32
3.3.1	Konfigurasi Hypervisor KVM untuk Kompresi Video	36
3.3.2	Konfigurasi Hypervisor KVM untuk Validasi Integritas Data	37
3.3.3	Konfigurasi Hypervisor KVM untuk Enkripsi AES	38
3.4	Skenario Pengujian Tuning	38
3.4.1	Skenario Pengujian Kompresi Video	38
3.4.2	Skenario Pengujian Validasi Integritas Data	40
3.4.3	Skenario Pengujian Enkripsi AES	42
3.5	Analisis	44
3.5.1	Analisis Pengujian Kompresi Video	45
3.5.2	Analisis Pengujian Validasi Integritas Data	45
3.5.3	Analisis Pengujian Enkripsi AES	45
4	PENGUJIAN TUNING PARAMETER HYPERVISOR KVM DE-NGAN CLOUDSTACK AGENT	47

4.1	Hasil Pengujian Tuning Hypervisor KVM dengan Kompresi Video	47
4.1.1	Konfigurasi Default	47
4.1.2	Konfigurasi dengan SSSE3	49
4.1.3	Konfigurasi dengan SSE4.1	50
4.1.4	Konfigurasi dengan SSE4.2	52
4.1.5	Konfigurasi dengan SSE4a	53
4.1.6	Konfigurasi dengan SSSE3 + SSE4.1	55
4.1.7	Konfigurasi dengan SSSE3 + SSE4.2	56
4.1.8	Konfigurasi dengan SSSE3 + SSE4a	58
4.1.9	Konfigurasi dengan SSE4.1 + SSE4.2	59
4.1.10	Konfigurasi dengan SSE4.1 + SSE4a	61
4.1.11	Konfigurasi dengan SSE4.2 + SSE4a	62
4.1.12	Konfigurasi dengan SSSE3 + SSE4.1 + SSE4.2	64
4.1.13	Konfigurasi dengan SSSE3 + SSE4.1 + SSE4a	65
4.1.14	Konfigurasi dengan SSSE3 + SSE4.2 + SSE4a	67
4.1.15	Konfigurasi dengan SSE4.1 + SSE4.2 + SSE4a	68
4.1.16	Konfigurasi dengan SSSE3 + SSE4.1 + SSE4.2 + SSE4a	70
4.1.17	Analisis Pengujian Tuning Hypervisor KVM dengan Kom- presi Video	72
4.2	Hasil Pengujian Tuning Hypervisor KVM dengan Validasi Integri- tas Data	72
4.2.1	Konfigurasi Default	73
4.2.2	Konfigurasi dengan SSE4.2	74
4.2.3	Analisis Pengujian Tuning Hypervisor KVM dengan Vali- dasi Integritas Data	76
4.3	Hasil Pengujian Tuning Hypervisor KVM dengan Enkripsi AES	76
4.3.1	Konfigurasi Default	76
4.3.2	Konfigurasi dengan AES	79
4.3.3	Analisis Pengujian Tuning Hypervisor KVM dengan Enkripsi AES	82
5	KESIMPULAN	83
5.1	Saran	83
DAFTAR REFERENSI		85
LAMPIRAN		1

DAFTAR GAMBAR

Gambar 2.1	Logo Apache CloudStack	7
Gambar 2.2	Hypervisor Type 1 vs Type 2	8
Gambar 2.3	Arsitektur Hypervisor KVM	10
Gambar 2.4	Perbedaan Komputasi Tanpa Virtualisasi dengan Virtualisasi	11
Gambar 3.1	Tahapan Penelitian	18
Gambar 3.2	Persiapan Lingkungan Penelitian	19
Gambar 3.3	Seluruh flag instruksi di Host	33
Gambar 3.4	Flag default KVM	33
Gambar 3.5	Keluaran perintah virsh list	34
Gambar 3.6	Perintah virsh edit i-2-16-VM untuk melihat konfigurasi virtual machine i-2-16-VM	34
Gambar 3.7	Flag tuned dengan ssse3	36
Gambar 3.8	Flowchart Kode Pengujian Kompresi Video	40
Gambar 3.9	Flowchart Kode Pengujian Validasi Integritas Data	42
Gambar 3.10	Potongan Plaintext Data untuk AES Encryption Benchmark	42
Gambar 3.11	Kode Pengujian Enkripsi AES	44
Gambar 3.12	Flowchart Kode Pengujian Enkripsi dan Dekripsi AES	44
Gambar 4.1	lscpu Konfigurasi Default	47
Gambar 4.2	Test Kompresi Video dengan Konfigurasi Default	48
Gambar 4.3	Grafik Tes Kompresi Video dengan Konfigurasi Default	48
Gambar 4.4	lscpu Konfigurasi SSSE3	49
Gambar 4.5	Test Kompresi Video dengan Konfigurasi SSSE3	49
Gambar 4.6	Grafik Tes Kompresi Video dengan Konfigurasi SSSE3	50
Gambar 4.7	lscpu Konfigurasi SSE4.1	50
Gambar 4.8	Test Kompresi Video dengan Konfigurasi SSE4.1	51
Gambar 4.9	Grafik Tes Kompresi Video dengan Konfigurasi SSE4.1	51
Gambar 4.10	lscpu Konfigurasi SSE4.2	52
Gambar 4.11	Test Kompresi Video dengan Konfigurasi SSE4.2	52
Gambar 4.12	Grafik Tes Kompresi Video dengan Konfigurasi SSE4.2	53
Gambar 4.13	lscpu Konfigurasi SSE4a	53
Gambar 4.14	Test Kompresi Video dengan Konfigurasi SSE4a	54

Gambar 4.15 Grafik Tes Kompresi Video dengan Konfigurasi SSE4a	54
Gambar 4.16 lscpu Konfigurasi SSSE3 + SSE4.1	55
Gambar 4.17 Test Kompresi Video dengan Konfigurasi SSSE3 + SSE4.1	55
Gambar 4.18 Grafik Tes Kompresi Video dengan Konfigurasi SSSE3 + SSE4.1	56
Gambar 4.19 lscpu Konfigurasi SSSE3 + SSE4.2	56
Gambar 4.20 Test Kompresi Video dengan Konfigurasi SSSE3 + SSE4.2	57
Gambar 4.21 Grafik Tes Kompresi Video dengan Konfigurasi SSSE3 + SSE4.2	57
Gambar 4.22 lscpu Konfigurasi SSSE3 + SSE4a	58
Gambar 4.23 Test Kompresi Video dengan Konfigurasi SSSE3 + SSE4a	58
Gambar 4.24 Grafik Tes Kompresi Video dengan Konfigurasi SSSE3 + SSE4a	59
Gambar 4.25 lscpu Konfigurasi SSE4.1 + SSE4.2	59
Gambar 4.26 Test Kompresi Video dengan Konfigurasi SSE4.1 + SSE4.2	60
Gambar 4.27 Grafik Tes Kompresi Video dengan Konfigurasi SSE4.1 + SSE4.2	60
Gambar 4.28 lscpu Konfigurasi SSE4.1 + SSE4a	61
Gambar 4.29 Test Kompresi Video dengan Konfigurasi SSE4.1 + SSE4a	61
Gambar 4.30 Grafik Tes Kompresi Video dengan Konfigurasi SSE4.1 + SSE4a	62
Gambar 4.31 lscpu Konfigurasi SSE4.2 + SSE4a	62
Gambar 4.32 Test Kompresi Video dengan Konfigurasi SSE4.2 + SSE4a	63
Gambar 4.33 Grafik Tes Kompresi Video dengan Konfigurasi SSE4.2 + SSE4a	63
Gambar 4.34 lscpu Konfigurasi SSSE3 + SSE4.1 + SSE4.2	64
Gambar 4.35 Test Kompresi Video dengan Konfigurasi SSSE3 + SSE4.1 + SSE4.2	64
Gambar 4.36 Grafik Tes Kompresi Video dengan Konfigurasi SSSE3 + SSE4.1 + SSE4.2	65
Gambar 4.37 lscpu Konfigurasi SSSE3 + SSE4.1 + SSE4a	65
Gambar 4.38 Test Kompresi Video dengan Konfigurasi SSSE3 + SSE4.1 + SSE4a	66
Gambar 4.39 Grafik Tes Kompresi Video dengan Konfigurasi SSSE3 + SSE4.1 + SSE4a	66
Gambar 4.40 lscpu Konfigurasi SSSE3 + SSE4.2 + SSE4a	67

Gambar 4.41 Test Kompresi Video dengan Konfigurasi SSSE3 + SSE4.2 + SSE4a	67
Gambar 4.42 Grafik Tes Kompresi Video dengan Konfigurasi SSSE3 + SSE4.2 + SSE4a	68
Gambar 4.43 lscpu Konfigurasi SSE4.1 + SSE4.2 + SSE4a	68
Gambar 4.44 Test Kompresi Video dengan Konfigurasi SSE4.1 + SSE4.2 + SSE4a	69
Gambar 4.45 Grafik Tes Kompresi Video dengan Konfigurasi SSE4.1 + SSE4.2 + SSE4a	69
Gambar 4.46 lscpu Konfigurasi SSSE3 + SSE4.1 + SSE4.2 + SSE4a	70
Gambar 4.47 Test Kompresi Video dengan Konfigurasi SSSE3 + SSE4.1 + SSE4.2 + SSE4a	70
Gambar 4.48 Grafik Tes Kompresi Video dengan Konfigurasi SSSE3 + SSE4.1 + SSE4.2 + SSE4a	71
Gambar 4.49 Grafik <i>speedup</i> Tes Kompresi Video	72
Gambar 4.50 lscpu Konfigurasi Default	73
Gambar 4.51 Test Validasi Integritas Data Dengan Konfigurasi Default	73
Gambar 4.52 Grafik Tes Validasi Integritas Data Dengan Konfigurasi Default	74
Gambar 4.53 lscpu Konfigurasi SSE4.2	74
Gambar 4.54 Test Validasi Integritas Data Dengan Konfigurasi SSE4.2	75
Gambar 4.55 Grafik Tes Validasi Integritas Data Dengan Konfigurasi SSE4.2	75
Gambar 4.56 Grafik <i>speedup</i> Tes Validasi Integritas Data	76
Gambar 4.57 lscpu Konfigurasi Default	76
Gambar 4.58 Test Enkripsi AES Dengan Konfigurasi Default	77
Gambar 4.59 Grafik Tes Enkripsi AES Dengan Konfigurasi Default	78
Gambar 4.60 Grafik Tes Dekripsi AES Dengan Konfigurasi Default	78
Gambar 4.61 lscpu Konfigurasi AES	79
Gambar 4.62 Test Enkripsi AES Dengan Konfigurasi AES	80
Gambar 4.63 Grafik Tes Enkripsi AES Dengan Konfigurasi AES	81
Gambar 4.64 Grafik Tes Dekripsi AES Dengan Konfigurasi AES	81
Gambar 4.65 Grafik Rata-Rata Waktu Tes Enkripsi AES	82
Gambar 4.66 Grafik Rata-Rata Waktu Tes Dekripsi AES	82

DAFTAR TABEL

Tabel 2.1	Perbandingan SSE4.1 dan SSE4.2	17
------------------	--	----

DAFTAR KODE

Kode 3.1	Konfigurasi Netplan untuk Cloudbr0	21
Kode 3.2	Konfigurasi mysqld.cnf	23
Kode 3.3	Konfigurasi flag default	35
Kode 3.4	Konfigurasi flag ssse3	35
Kode 3.5	Konfigurasi Hypervisor KVM untuk Kompresi Video	37
Kode 3.6	Konfigurasi Hypervisor KVM untuk Validasi Integritas Data . .	37
Kode 3.7	Konfigurasi Hypervisor KVM untuk Enkripsi AES	38
Kode 3.8	Kode Pengujian Kompresi Video	39
Kode 3.9	Kode Pengujian Validasi Integritas Data	41

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Cloud Computing adalah sebuah model yang memungkinkan akses yang luas, nyaman, dan sesuai permintaan ke sumber daya komputasi bersama, seperti jaringan, server, dan penyimpanan. Model ini terdiri dari lima karakteristik utama, yakni layanan mandiri *on-demand*, akses jaringan yang luas, pengelolaan sumber daya (*resource pooling*), elastisitas yang cepat, dan layanan yang terukur. *Cloud Computing* juga melibatkan tiga model layanan, seperti *Software as a Service* (SaaS), *Platform as a Service* (PaaS), dan *Infrastructure as a Service* (IaaS). Selain itu, terdapat empat model implementasi: *private cloud*, *community cloud*, *public cloud*, dan *hybrid cloud*, masing-masing sesuai dengan kebutuhan dan kegunaan dari organisasi yang berbeda[1].

Kita dapat membuat sistem *Cloud Computing* sendiri dengan menggunakan Apache CloudStack. Apache CloudStack adalah perangkat lunak *open source* yang dirancang untuk implementasi dan administrasi jaringan *Virtual Machine*. CloudStack berfungsi sebagai platform *Cloud Computing Infrastructure as a Service* (IaaS) yang *reliable*, CloudStack juga dikenal karena *high availability* dan skalabilitasnya. Apache CloudStack digunakan oleh berbagai penyedia layanan untuk menyediakan layanan cloud publik, CloudStack juga digunakan oleh banyak perusahaan untuk membentuk solusi *private cloud* atau sebagai bagian dari konfigurasi *hybrid cloud*[2].

Untuk menjalankan Apache Cloudstack diperlukan sebuah hypervisor. Hypervisor adalah komponen virtualisasi yang mengawasi dan mengelola sistem operasi *guest* pada sistem *host*[3]. Hypervisor mengontrol komunikasi instruksi antara sistem operasi *guest* dan *hardware* dari *host*[3].

Pada penelitian ini Penulis menggunakan sistem operasi Ubuntu yang berbasis Linux dan menggunakan KVM sebagai hypervisornya. Hypervisor KVM (*Kernel-based Virtual Machine*) adalah sebuah hypervisor berbasis kernel yang memungkinkan virtualisasi pada sistem operasi Linux[4]. KVM memanfaatkan modul kernel untuk menyediakan dukungan langsung untuk virtualisasi dengan menggunakan instruksi *hardware* yang mendukung teknologi virtualisasi, seperti Intel VT atau AMD-V.

Penggunaan KVM sebagai hypervisor untuk menjalankan *Virtual Machine* pada Apache Cloudstack telah menjadi praktik umum. Namun, pertanyaan mendasar muncul: apakah hypervisor ini telah dikonfigurasi secara optimal untuk menjalankan tugas yang diberikan dengan efisien? Penelitian ini bertujuan untuk menganalisis apakah konfigurasi hypervisor KVM telah diatur dengan tepat dan efisien. Pada penelitian ini, penulis akan menguji performa hypervisor dengan melakukan beberapa tugas komputasi, seperti kompresi video, enkripsi, dan juga validasi integritas data. Selanjutnya, waktu yang diperlukan untuk menyelesaikan proses komputasi ini akan diukur dan dibandingkan dalam rangka evaluasi performa.

1.2 Permasalahan

Pada bagian ini akan dijelaskan mengenai definisi permasalahan yang Penulis hadapi dan ingin diselesaikan serta asumsi dan batasan yang digunakan dalam menyelesaiannya.

1.2.1 Definisi Permasalahan

Berdasarkan latar belakang yang telah dijelaskan pada bab 1.1 bahasan mengenai rumusah yang akan dibahas pada penelitian ini adalah sebagai berikut:

1. Apakah konfigurasi default dari Hypervisor KVM sudah dibuat dengan optimal dan efisien?
2. Apakah tuning Hypervisor KVM untuk *Virtual Machine* di Apache Cloudstack memberikan efek yang signifikan terhadap performa tugas komputasi seperti kompresi video, enkripsi dan dekripsi AES, dan validasi integritas data dengan CRC32?
3. Bagaimana konfigurasi Hypervisor KVM dapat diatur secara optimal untuk meningkatkan efisiensi atau kecepatan dalam menangani tugas komputasi tertentu pada Apache CloudStack?

1.2.2 Batasan Permasalahan

Karena berbagai kendala yang dihadapi pada skripsi ini dilakukan pembatasan masalah pada penelitian sebagai berikut:

1. Penelitian dilakukan menggunakan perangkat laptop dengan spesifikasi sebagai berikut:

- CPU: AMD Ryzen 7 4800H x64.
 - RAM: 16 GB.
 - Penyimpanan: 1TB pada SSD.
2. Sistem operasi yang digunakan adalah Ubuntu Server 22.04 LTS.
 3. Karena perangkat menggunakan CPU AMD, implementasi teknologi virtualisasi yang diadopsi adalah AMD-V.
 4. Parameter yang menjadi fokus analisis dalam penelitian ini adalah rata-rata perbedaan waktu kompresi video, waktu enkripsi dengan AES, dan waktu validasi integritas data dengan CRC32 antara hypervisor default dan hypervisor yang telah dituning.

1.3 Tujuan

Penelitian ini berfokus pada analisis performa dari tuning Hypervirisor KVM dengan menggunakan CloudStack Agent pada *Cloud Management Platform* Apache CloudStack. Dengan tujuan sebagai berikut:

- Evaluasi Konfigurasi Default Hypervisor KVM

Menilai apakah konfigurasi default hypervisor KVM telah diatur secara optimal dan efisien untuk mengerjakan tugas komputasi tertentu.

- Mengetahui Efek Performa dari Tuning Hypervisor KVM

Memahami apakah tuning hypervisor KVM dapat mempengaruhi performa dalam menjalankan tugas komputasi tertentu.

-

Hasil penelitian ini diharapkan tidak hanya memberikan panduan praktis tetapi juga kontribusi akademis dalam bidang optimalisasi infrastruktur cloud.

1.4 Metodologi Penelitian

Beberapa metodologi yang dilakukan pada penelitian ini, antara lain:

1. Studi Literatur

Studi literatur dalam penelitian ini dilakukan dengan melakukan pencarian, membaca, dan memahami jurnal serta sumber referensi yang terkait dengan *Cloud Computing*. Penelitian ini juga melibatkan eksplorasi literatur yang mendalam terkait dengan metode tuning hypervisor KVM, kondisi saat melakukan pemrosesan kompresi video, enkripsi dan dekripsi AES, dan juga mengenai validasi integritas data dengan CRC32 serta evaluasi keuntungan dan kerugian yang mungkin muncul akibat hasil tuning ini.

2. Pengumpulan Data

Pada penelitian ini, akan dilakukan tiga pengujian

(a) Kompresi Video

Kompresi video akan dilakukan menggunakan ffmpeg pada dua sistem, yaitu sistem yang menggunakan KVM default dan sistem yang menggunakan KVM yang sudah dituning. Pada masing-masing sistem, kompresi video akan dilakukan sebanyak 10 kali. Setelah itu, waktu rata-rata kompresi video pada kedua sistem akan dibandingkan dan dianalisis.

(b) Validasi Integritas Data CRC32

Validasi Integritas Data dengan menggunakan fungsi CRC32 akan dilakukan dengan program benchmark yang dibuat Penulis dengan python pada kedua sistem. Pada masing-masing sistem, validasi CRC32 akan dilakukan sebanyak 10 kali. Setelah itu, waktu rata-rata validasi integritas data dengan CRC32 pada kedua sistem akan dibandingkan dan dianalisis.

(c) Enkripsi dan Dekripsi AES

Enkripsi dan dekripsi AES akan dilakukan dengan program benchmark yang dibuat Penulis dengan python pada kedua sistem. Pada masing-masing sistem, enkripsi dan dekripsi AES akan dilakukan sebanyak 10 kali. Setelah itu, waktu rata-rata enkripsi dan dekripsi AES pada kedua sistem akan dibandingkan dan dianalisis.

3. Analisis

Setelah itu, dilakukan evaluasi terhadap hasil yang diperoleh dan kemudian menyusun kesimpulan.

1.5 Sistematika Penulisan

Pada penulisan skripsi ini terdiri dari 3 bab dengan pokok bahasan yang berbeda-beda untuk setiap bab.

- **BAB I PENDAHULUAN**

Bab I membahas tentang pendahuluan penelitian, yang meliputi latar belakang masalah, rumusan masalah, batasan masalah, tujuan penelitian, metode penelitian, dan sistematika penulisan.

- **BAB II TUNING HYPERVISOR KVM**

Bab II membahas tentang landasan teori yang digunakan untuk memahami dan menganalisis hasil penelitian.

- **BAB III METODE PENGUJIAN TUNING PARAMETER HYPERVISOR KVM DENGAN CLOUDSTACK AGENT**

Bab III membahas tentang tahapan dan bagaimana metode yang digunakan untuk melakukan tuning dan pengujian dari tuning parameter Hypervisor KVM, dan juga bagaimana analisis akan dilakukan.

- **BAB IV PENGUJIAN TUNING PARAMETER HYPERVISOR KVM DENGAN CLOUDSTACK AGENT**

Bab IV menjelaskan hasil penelitian yang terdiri dari pengujian tuning parameter hypervisor KVM, dan dilakukan analisis performa hypervisor KVM setelah proses tuning.

- **BAB V PENUTUP**

Bab V menjelaskan kesimpulan dari penelitian dilakukan dan saran.

BAB 2

TUNING HYPERVISOR KVM

Pada bab ini akan dijelaskan mengenai landasan teori yang digunakan dalam penelitian.

2.1 *Cloud Computing*

Cloud Computing, menurut definisi National Institute of Standards and Technology (NIST), merupakan suatu model komputasi yang menyediakan akses dan konfigurasi sumber daya komputasi seperti jaringan, server, penyimpanan, aplikasi, dan layanan secara on-demand, memungkinkan pelepasan yang cepat tanpa interaksi yang kompleks dengan penyedia layanan[1]. *Cloud Computing* bukanlah suatu teknologi spesifik, melainkan sebuah model yang menggambarkan operasional dan ekonomi untuk penyediaan serta penggunaan infrastruktur IT dan layanan terkait. Beberapa definisi cloud memiliki karakteristik umum yang sama, seperti *pay-per-use* (pembayaran sesuai dengan penggunaan), kapasitas yang elastis, layanan mandiri, dan abstraksi atau virtualisasi sumber daya[5]. NIST membagi model layanan cloud menjadi[1]:

1. Software As A Service (SaaS)

Konsumen dapat memanfaatkan kemampuan dengan menggunakan aplikasi penyedia yang beroperasi di dalam infrastruktur cloud. Aplikasi tersebut dapat diakses dari berbagai perangkat klien melalui antarmuka klien yang ringan, seperti browser web (contohnya, email berbasis web), atau antarmuka program. Pengguna tidak perlu mengelola atau mengontrol infrastruktur cloud yang mendasarinya, termasuk jaringan, server, sistem operasi, penyimpanan, atau bahkan kemampuan aplikasi individual, kecuali mungkin pada pengaturan konfigurasi aplikasi khusus pengguna yang terbatas.

2. Platform As A Service (PaaS)

Konsumen diberikan kemampuan untuk mendeploy aplikasi yang mereka buat atau peroleh ke infrastruktur cloud, menggunakan bahasa pemrograman, *library*, layanan, dan alat yang didukung oleh penyedia. Pengguna tidak perlu mengelola atau mengontrol infrastruktur cloud yang mendasarinya, termasuk jaringan, server, sistem operasi, atau penyimpanan. Meskipun begitu,

pengguna tetap memiliki kendali atas aplikasi yang diimplementasikan dan mungkin dapat mengatur konfigurasi lingkungan hosting aplikasi.

3. Infrastructure As A Service (IaaS)

Kemampuan yang diberikan kepada konsumen adalah untuk menyediakan sumber daya pemrosesan, penyimpanan, jaringan, dan sumber daya komputasi dasar lainnya di mana konsumen dapat mendeploy dan menjalankan perangkat lunak sembarang, yang dapat mencakup sistem operasi dan aplikasi. Konsumen tidak mengelola atau mengendalikan infrastruktur awan yang mendasari tetapi memiliki kendali atas sistem operasi, penyimpanan, dan aplikasi yang didistribusikan; dan mungkin kendali terbatas terhadap beberapa komponen jaringan tertentu (misalnya, firewall host).

2.2 Apache CloudStack



Gambar 2.1: Logo Apache CloudStack

Apache CloudStack adalah perangkat lunak *open source* yang dirancang untuk mendeploy dan mengelola jaringan besar mesin virtual, sebagai platform komputasi awan berbasis Infrastructure as a Service (IaaS) yang sangat tersedia dan dapat *scaleable*[2]. Pada Gambar 2.1 adalah logo dari Apache Cloudstack. CloudStack digunakan oleh sejumlah penyedia layanan untuk menawarkan layanan cloud publik, dan oleh banyak perusahaan untuk menyediakan penawaran cloud *on-premise* (pribadi), atau sebagai bagian dari solusi cloud hybrid[2].

CloudStack adalah solusi siap pakai yang mencakup seluruh "stack" fitur yang paling diinginkan oleh organisasi dengan IaaS cloud: orkestrasi komputasi, Jaringan sebagai Layanan, manajemen pengguna dan akun, dan antarmuka Pengguna (UI) yang baik[2].

CloudStack saat ini mendukung hypervisor paling populer: VMware, KVM, Citrix XenServer, Xen Cloud Platform (XCP), Oracle VM server, dan Microsoft Hyper-V[2].

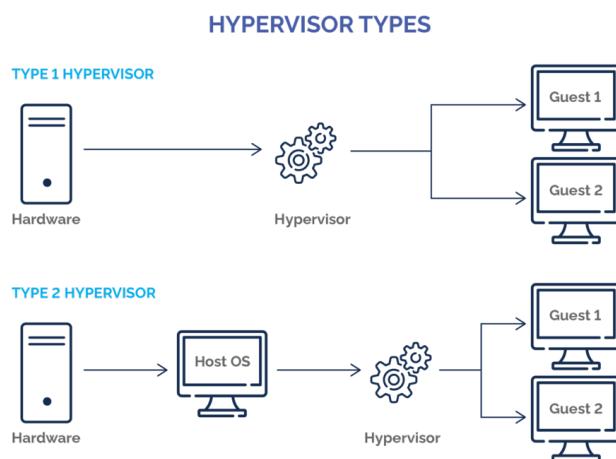
Pengguna dapat mengelola cloud mereka dengan antarmuka web yang mudah digunakan, alat *commandline*, dan/atau API RESTful yang lengkap. Selain itu,

CloudStack menyediakan API yang kompatibel dengan AWS EC2 dan S3 untuk organisasi yang ingin mendeploy cloud secara hybrid[2].

CloudStack, yang awalnya dikembangkan oleh Cloud.com, diakuisisi oleh Citrix pada tahun 2011 dan diserahkan kepada Apache Software Foundation pada tahun 2012. Pengembangan saat ini diatur oleh Apache Foundation dengan kode yang tersedia di bawah lisensi Apache 2.0[6].

2.3 Hypervisor

Sebuah hypervisor adalah perangkat lunak atau perangkat keras yang digunakan untuk menciptakan dan mengelola mesin virtual. Juga dikenal sebagai '*Virtual Machine Monitor*' atau VMM, hypervisor memungkinkan satu server fisik menjalankan beberapa mesin virtual, mengatasi keterbatasan satu sistem operasi pada satu server. Hypervisor dapat berupa perangkat keras atau program, dan fungsinya adalah menciptakan, memonitor, dan mengelola mesin-mesin virtual[7].



Gambar 2.2: Hypervisor Type 1 vs Type 2

Pada Gambar 2.2 diperlihatkan diagram sederhana perbedaan daripada Hypervisor tipe 1 dan Hypervisor tipe 2. Hypervisor tipe 1 dan tipe 2 merupakan perangkat lunak yang digunakan untuk menjalankan satu atau lebih *Virtual Machine* (VM) pada satu mesin fisik. *Virtual Machine* adalah replika digital dari mesin fisik, menciptakan lingkungan komputasi terisolasi yang pengguna alami sebagai sepenuhnya independen dari perangkat keras yang mendasarinya[8]. Hypervisor mengelola dan mengalokasikan sumber daya fisik ke VM dan berkomunikasi dengan perangkat keras di latar belakang. Hypervisor tipe 1 ditempatkan di atas server tanpa sistem operasi dan memiliki akses langsung ke sumber daya perangkat keras, sehingga

dikenal juga sebagai *bare metal* hypervisor. Sebaliknya, hypervisor tipe 2 adalah aplikasi yang diinstal pada sistem operasi host dan juga dikenal sebagai *hosted* atau *embedded* hypervisor[8].

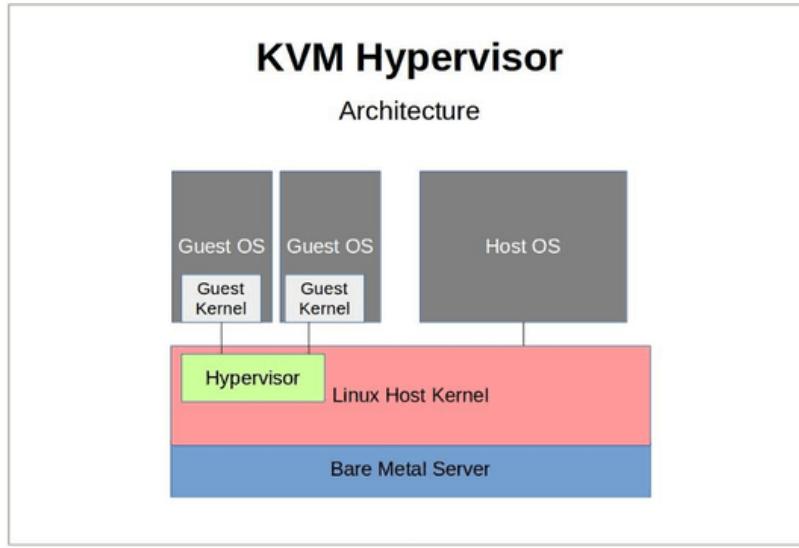
2.4 KVM (*Kernel-Based Virtual Machine*)

KVM (Kernel-based Virtual Machine) merupakan modul virtualisasi *open source* dan gratis dalam kernel Linux yang memungkinkan kernel berfungsi sebagai hypervisor. KVM adalah hypervisor tipe 1 atau biasa juga disebut sebagai hypervisor *bare metal*. KVM memungkinkan mesin host menjalankan beberapa lingkungan virtual terisolasi yang disebut sebagai guest atau *Virtual Machine* (VM). KVM (Kernel-based Virtual Machine) merupakan modul virtualisasi sumber terbuka dan gratis dalam kernel Linux yang memungkinkan kernel berfungsi sebagai hypervisor. Ini adalah tipe-1 (bare-metal) hypervisor yang memungkinkan mesin host menjalankan beberapa lingkungan virtual terisolasi yang disebut sebagai guest atau mesin virtual (VM).

KVM telah disatukan ke dalam kernel Linux utama pada versi 2.6.20, yang dirilis pada 5 Februari 2007. Untuk dapat berjalan, KVM memerlukan prosesor dengan ekstensi virtualisasi perangkat keras, seperti Intel VT atau AMD-V. KVM menyediakan abstraksi perangkat tetapi tidak ada emulasi prosesor. Modul ini mengekspos antarmuka /dev/kvm, yang dapat digunakan oleh mode pengguna untuk menyiapkan ruang alamat VM guest.

KVM mendukung virtualisasi perangkat keras untuk berbagai sistem operasi guest, termasuk BSD, Solaris, Windows, Haiku, ReactOS, Plan 9, dan lainnya. Sebagai bagian dari kernel Linux, KVM langsung mengambil manfaat dari setiap fitur baru, perbaikan, dan kemajuan Linux tanpa perlu rekayasa tambahan.

Dalam pengaturan KVM, CPU virtual dari VM diimplementasikan sebagai thread (disebut "virtual CPU" atau vcpu) yang dijadwalkan oleh *Linux Kernel Scheduler*. KVM pada dasarnya adalah pengemudi untuk ekstensi virtualisasi processor. Setiap kali *Linux Scheduler* memilih tugas untuk dijalankan pada CPU fisik, dan tugas tersebut dikerjakan oleh CPU virtual dari VM, KVM "dihubungi" untuk memastikan bahwa yang sebenarnya berjalan di perangkat keras adalah program dari OS guest[9].

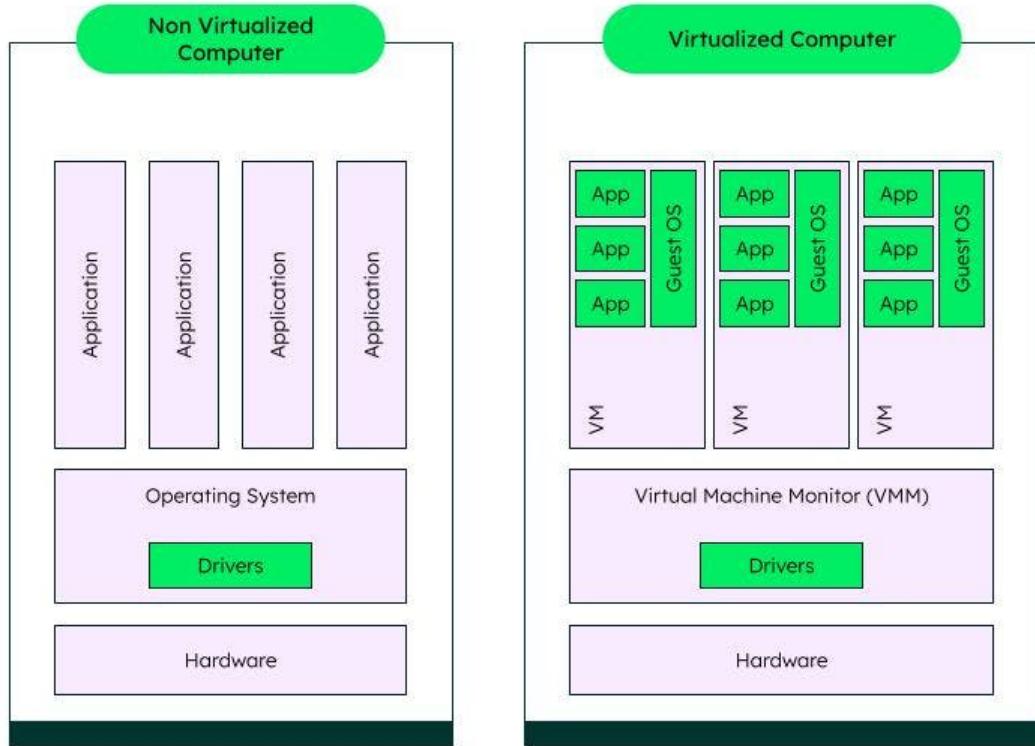


Gambar 2.3: Arsitektur Hypervisor KVM

Pada Gambar 2.3 adalah struktur daripada hypervisor KVM. KVM berjalan langsung pada kernel linux host dan membagikan resourcenyanya kepada guest *Virtual Machine*, sedangkan sistem operasi host berjalan langsung diatas kernel host. Sistem operasi host dapat melakukan konfigurasi kepada *Virtual Machine* yang menggunakan hypervisor miliknya.

2.5 Virtual Machine

Virtual Machine (VM) adalah sebuah lingkungan komputasi yang dibuat secara virtual di dalam sebuah komputer fisik (host)[10]. VM ini berfungsi seperti sebuah komputer independen yang dapat menjalankan sistem operasi dan aplikasi seperti komputer fisik biasa, tetapi semuanya berjalan dalam lingkungan virtual yang terisolasi di dalam host fisik. Setiap *Virtual Machine* (VM) menjalankan sistem operasi secara independen dan beroperasi terpisah dari VM lainnya, bahkan ketika semuanya berjalan pada host yang sama. Perangkat lunak yang dikenal sebagai hypervisor atau *Virtual Machine Monitor* (VMM) memungkinkan kita untuk menjalankan berbagai sistem operasi yang berbeda secara bersamaan pada *Virtual Machine* yang berbeda. VM memiliki beberapa keunggulan dibandingkan dengan mesin fisik, termasuk kemampuan untuk menjalankan beberapa lingkungan sistem operasi pada satu komputer fisik, mendukung aplikasi *legacy*, dan menyediakan opsi pemulihan dalam situasi bencana. VM digunakan untuk berbagai keperluan, seperti *Cloud Computing*, pengujian sistem operasi baru, dan menjalankan beberapa aplikasi pada satu mesin fisik[11].



Gambar 2.4: Perbedaan Komputasi Tanpa Virtualisasi dengan Virtualisasi

Terdapat perbedaan dalam cara berjalan antara komputer tanpa virtualisasi (host) dan komputer yang divirtualisasi (guest) atau *Virtual Machine*. Pada Gambar 2.4, dapat diperlihatkan bahwa pada komputer tanpa virtualisasi, sistem operasinya berjalan langsung di atas perangkat keras, dengan driver yang berfungsi sebagai jembatan, sehingga aplikasi dapat berjalan langsung di atas sistem operasi tersebut. Di sisi lain, pada komputer yang divirtualisasi, sistem operasinya berjalan di atas VMM atau Virtual Machine Monitor, di mana satu VMM dapat menjalankan banyak sistem operasi guest.

2.6 Advanced Encryption Standard (AES)

Advanced Encryption Standard atau disingkat AES pertama kali diumumkan oleh NIST (*National Institute Of Standards and Technology*) pada tahun 2001 sebagai pengganti algoritma DES yang semakin mudah diretas. AES merupakan hasil dari kompetisi yang diadakan oleh NIST pada tahun 1997 yang dimenangkan oleh tim dari belgium yang memiliki nama asli Rijndael. Rijndael dikembangkan oleh dua orang kriptografer dari belgium bernama Joan Daemen dan Vincent Rijmen. AES dipublikasikan oleh NIST pada tahun 2001 dan tersebar secara efektif pada bulan May 2002 setelah disetujui oleh Menteri Perdagangan Amerika, Donald Evans[12].

AES merupakan metode enkripsi data dengan teknik enkripsi simetris, yang berarti menggunakan kunci yang sama untuk proses enkripsi dan dekripsi. Kunci pada AES tersedia dalam tiga varian ukuran, yakni 128 bit, 192 bit, dan 256 bit. Meskipun ukuran kuncinya bervariasi, ukuran blok data yang dienkripsi tetap sama, yaitu 128 bit. Ukuran blok mengacu pada cara data dienkripsi dengan memecah dan menyusunnya. Data disusun menjadi array 4×4 berukuran 16 byte, di mana setiap byte terdiri dari 8 bit, sehingga total setiap blok berjumlah 128 bit[13].

2.7 Cyclic Redundancy Check (CRC)

Cyclic Redundancy Check (CRC) merupakan teknik yang digunakan untuk mendetect kesalahan dalam transmisi data digital. Teknik ini melibatkan penambahan kode pemeriksaan pada data untuk mengidentifikasi kesalahan yang mungkin terjadi selama proses transmisi. CRC didasarkan pada konsep pembagian polinomial, khususnya dengan menggunakan *Linear Feedback Shift Registers* (LFSRs).

1. Gambaran Teknik CRC[14]:

- CRC digunakan untuk mendeteksi kesalahan dalam data digital, terutama pada aplikasi yang beroperasi dengan kecepatan tinggi.
- CRC menggunakan *Linear Feedback Shift Registers* untuk deteksi kesalahan.
- Teknik ini melibatkan pembagian biner untuk menghasilkan sisa (CRC) yang kemudian ditambahkan ke data sebagai pemeriksaan kesalahan di penerima.

2. Prinsip CRC[15]:

- Teknik CRC menambahkan kode pemeriksaan r-bit ke kode informasi k-bit, membentuk kode (n, k) dengan panjang $n(k + r)$ bit.
- Setiap kode (n, k) memiliki polinomial $g(x)$ dengan kekuatan maksimum $n-k=r$ terkait dengannya, dikenal sebagai polinomial yang dihasilkan dari kode CRC.
- Kode CRC umumnya ditempatkan di akhir informasi untuk mendeteksi kesalahan selama transmisi.
- Di sisi penerima, perhitungan CRC dilakukan pada informasi yang diterima untuk memeriksa kesalahan.
- Implementasi seperti CRC-5, CRC-8, dan CRC-32 menunjukkan optimasi dan efisiensi yang spesifik untuk protokol dan sistem yang berbeda.

2.8 virsh

virsh adalah *command line tool* yang merupakan bagian dari *library* libvirt dan secara utama digunakan untuk mengelola dan berinteraksi dengan teknologi virtualisasi pada sistem Linux, khususnya KVM (Kernel-based Virtual Machine) dan QEMU (Quick Emulator), dan mendukung hypervisor lainnya seperti Xen, LXC, OpenVZ, VirtualBox, dan VMware ESX[16].

Secara sederhana penggunaan virsh adalah seperti ini

```
virsh [OPTION]... <command> <domain> [ARG] ...
```

Pada command ini, domain adalah ID domain numerik, atau nama domain, atau UUID domain; dan ARGS adalah opsi khusus dari command. Setiap command yang dimulai dengan # dianggap sebagai komentar dan diabaikan oleh sistem, semua perintah yang tidak dikenali akan didiagnosis.

2.9 *Streaming SIMD Extensions*

Streaming SIMD Extensions (SSE) adalah instruksi yang dibawa oleh Intel pada tahun 1999 dengan prosesor Pentium III dengan tujuan untuk meningkatkan kinerja prosesor arsitektur x86. Awalnya dikenal sebagai Katmai New Instructions (KNI) selama pengembangan, SSE secara signifikan meningkatkan kemampuan pemrosesan multimedia dan grafis dibanding pendahulunya, MMX. SSE mencakup 70 instruksi baru yang dirancang untuk meningkatkan tugas seperti pengolahan gambar, video 3D, audio dan video streaming, serta pengenalan suara[17].

Tidak seperti pendahulunya, MMX menggunakan unit floating-point standar yang sama, sedangkan SSE menggunakan unit terpisah dalam prosesor untuk perhitungan floating-point, yang memungkinkan pemrosesan yang lebih efisien. SSE mendukung operasi floating-point SIMD (Single Instruction Multiple Data) single precision, yang sangat penting untuk mengatasi bottleneck dalam pemrosesan grafis 3D. Teknologi ini memungkinkan satu instruksi untuk melakukan hingga empat operasi floating-point secara bersamaan, meningkatkan kecepatan dan efisiensi pemrosesan[17].

Manfaat SSE yang utama terlihat dalam decoding MPEG2, format standar untuk video DVD, memungkinkan prosesor yang dilengkapi SSE untuk menangani decoding tersebut di perangkat lunak dengan kecepatan penuh tanpa perangkat keras tambahan. SSE juga meningkatkan penggunaan CPU dan kinerja perangkat lunak pengenalan suara, memberikan akurasi yang lebih tinggi dan waktu respons

yang lebih cepat. Untuk sepenuhnya memanfaatkan kemampuan SSE, aplikasi harus diset secara khusus agar SSE-aware, sebuah fitur yang sekarang umum dalam banyak aplikasi grafis dan berhubungan dengan suara. SSE adalah perluasan dari MMX, yang berarti bahwa prosesor yang mendukung SSE juga kompatibel dengan instruksi MMX, hal ini memastikan *backward compatibility* dengan aplikasi yang mendukung MMX[17].

SSE telah berkembang seiring waktu dengan versi seperti SSE2, SSE3, dan SSE4, masing-masing membawa fitur baru dan meningkatkan kemampuan pemrosesan secara keseluruhan. Ekstensi ini telah banyak diadopsi dalam pengembangan perangkat lunak, terutama dalam bidang di mana pemrosesan paralel sangat penting.

2.9.1 SSE2

SSE2 adalah perluasan dari kemampuan SIMD (Single Instruction, Multiple Data) yang awalnya diperkenalkan dengan set instruksi SSE (Streaming SIMD Extensions). Pertama kali diperkenalkan oleh Intel dengan prosesor Pentium 4[18]. Perbedaan utama dan peningkatan dalam SSE2 dibandingkan dengan pendahulunya SSE meliputi:

1. Rentang Jenis Data yang Lebih Luas

SSE2 memperluas kemampuan SIMD untuk beroperasi pada data integer 64-bit dan data floating-point presisi ganda (SSE terutama berfokus pada operasi floating-point presisi tunggal).

2. Peningkatan Set Instruksi

SSE2 menambahkan instruksi dan kemampuan baru, meningkatkan pengolahan dan manipulasi berbagai jenis data, termasuk integer terpak (packed integers) dan angka floating-point presisi ganda.

3. Peningkatan Kinerja untuk Aplikasi Ilmiah dan Multimedia

Dengan menyediakan operasi pada tipe data yang lebih besar dan set operasi yang lebih luas, SSE2 meningkatkan efisiensi dan kinerja aplikasi yang berhubungan dengan perhitungan matematika kompleks, grafik 3D, pengkodean video, dan tugas multimedia lainnya.

Penggabungan SSE2 dalam arsitektur AMD64 menyoroti pentingnya dalam komputasi modern, terutama untuk aplikasi yang memerlukan kinerja tinggi dalam

pengolahan numerik dan multimedia. Dokumen ABI kemungkinan mengasumsikan keakraban dengan konsep-konsep ini, lebih berfokus pada detail implementasi dalam arsitektur AMD64 daripada menjelaskan perbedaan fundamental antara SSE dan SSE2[18].

2.9.2 SSE3

SSE3 (Streaming SIMD Extensions 3) merupakan ekstensi dari set instruksi SSE2, yang mana memperluas teknologi SSE (Streaming SIMD Extensions) asli. Perbedaan antara SSE3 dengan SSE2 antara lain:

1. 13 Set Instruksi Baru

SSE3 menambahkan tiga belas instruksi baru ke set instruksi SSE/SSE2 yang ada[19].

2. Dukungan untuk Model Eksekusi SIMD:

Dari 13 instruksi ini, sepuluh dirancang untuk mendukung model eksekusi SIMD (Single Instruction, Multiple Data). Model ini merupakan aspek penting dari komputasi paralel, di mana beberapa titik data diproses secara bersamaan menggunakan satu instruksi. Ini sangat bermanfaat untuk tugas-tugas seperti pengolahan grafis, perhitungan ilmiah, dan aplikasi multimedia, di mana operasi pada set data besar adalah hal yang umum[19].

3. Konversi Floating-Point ke Integer:

Salah satu instruksi SSE3 secara spesifik mempercepat pemrograman *x87-style*. Instruksi ini berfokus pada peningkatan efisiensi dalam mengkonversi nilai floating-point menjadi integer. Hal ini bisa sangat berguna dalam skenario di mana perhitungan melibatkan tipe data floating-point dan integer[19].

4. Akselerasi Sinkronisasi Thread:

Dua instruksi lain dalam set SSE3 ditujukan untuk meningkatkan sinkronisasi thread. Peningkatan ini sangat penting dalam aplikasi multi-thread, di mana pengelolaan eksekusi dan koordinasi beberapa thread adalah kunci untuk kinerja yang baik dan efisiensi[19].

Secara singkat, SSE3 memperluas kemampuan set instruksi SSE dan SSE2 dengan penekanan pada peningkatan eksekusi SIMD, konversi floating-point ke integer, serta sinkronisasi thread. Selain itu, SSE3 tetap mempertahankan kompatibilitas dengan jenis data dan lingkungan pemrograman yang sudah ada. Oleh karena

itu, SSE3 menjadi alat yang sangat berguna bagi pengembang yang bekerja pada tugas komputasi berkinerja tinggi, terutama yang melibatkan pemrosesan paralel dan jenis data yang kompleks.

2.9.3 SSSE3

SSSE3 (Supplemental Streaming SIMD Extensions 3) adalah ekstensi dari set instruksi SSE3 yang diperkenalkan oleh Intel pada tahun 2006. SSSE3 menambahkan 32 instruksi baru ke set instruksi SSE3 yang ada, yang dirancang untuk meningkatkan kinerja pemrosesan SIMD (Single Instruction, Multiple Data) pada aplikasi yang memerlukan operasi vektorisasi dan paralel. Pembaruan dari SSSE3 antara lain:

1. Dua belas instruksi yang melakukan operasi penambahan atau pengurangan horizontal.
2. Enam instruksi yang mengevaluasi nilai absolut.
3. Dua instruksi yang melakukan operasi perkalian-dan-penambahan dan mempercepat evaluasi *dot product*.
4. Dua instruksi yang mempercepat operasi perkalian bilangan bulat dan menghasilkan nilai bilangan bulat dengan penskalaan.
5. Dua instruksi yang melakukan *in-place shuffle* pada tingkat byte sesuai dengan operand kontrol pengacakan kedua.
6. Enam instruksi yang mengubah bilangan bulat menjadi negatif dalam operand tujuan jika elemen yang sesuai dalam operand sumber adalah negatif.
7. Dua instruksi yang menyelaraskan data dari gabungan dua operand.

2.9.4 SSE4

SSE4, yang merupakan singkatan dari Streaming SIMD Extensions 4, adalah serangkaian instruksi yang diperkenalkan oleh Intel dalam prosesor 64-bitnya yang dibuat dengan teknologi proses 45 nm. SSE4 terdiri dari dua subset: SSE4.1 dan SSE4.2[20].

Berdasarkan informasi dalam tabel 2.1, SSE4.1 dirancang untuk meningkatkan performa dalam aplikasi media, pencitraan, dan 3D, serta menghadirkan 47 instruksi baru. Instruksi-instruksi ini berkontribusi pada peningkatan vektorisasi kompiler dan memberikan dukungan untuk perhitungan *packed dword*. Di sisi lain,

Fitur	SSE4.1	SSE4.2
Pengenalan	Bagian dari prosesor Intel 45 nm	Diperkenalkan dengan arsitektur mikro Nehalem
Instruksi	47 set instruksi baru	7 instruksi tambahan
Fokus	Peningkatan kinerja pada media, pengolahan citra, dan beban kerja 3D	Pengolahan string dan teks
Penyempurnaan Utama	Peningkatan vektorisasi kompiler, dukungan untuk perhitungan <i>packed dword</i>	Kemampuan integer SIMD yang ditingkatkan, fokus pada operasi string
Kompatibilitas	Sepenuhnya kompatibel dengan versi SSE sebelumnya	Juga kompatibel dengan versi SSE sebelumnya
Dukungan OS	Tidak memerlukan dukungan OS baru	Sama seperti SSE4.1

Tabel 2.1: Perbandingan SSE4.1 dan SSE4.2

SSE4.2, yang pertama kali diperkenalkan dalam prosesor dengan mikroarsitektur Nehalem, menambahkan tujuh instruksi tambahan. Fokus utama dari instruksi-instruksi tambahan ini adalah pada pemrosesan string dan teks, serta peningkatan dalam kemampuan integer SIMD[20].

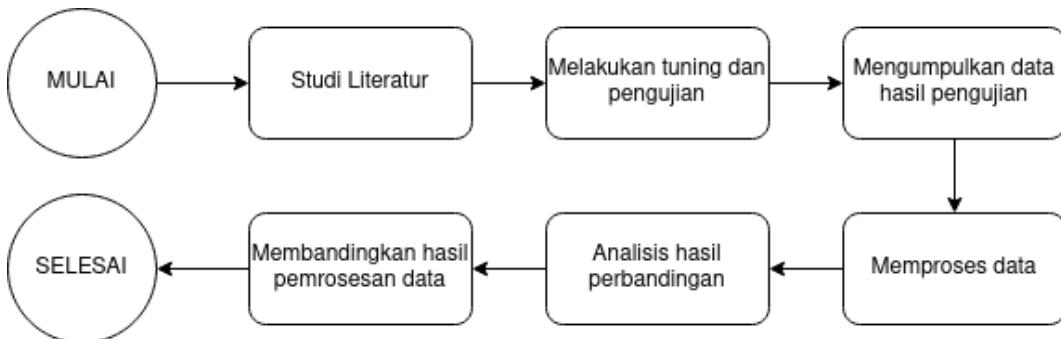
SSE4 tidak memperkenalkan tipe data baru tetapi mencakup berbagai tipe instruksi seperti perkalian dword, *floating-point dot-product*, *streaming load hint*, *packed blending*, dan instruksi MIN/MAX *packed integer*. Teknologi ini bertujuan untuk meningkatkan dukungan untuk perhitungan *packed dword* dan meningkatkan throughput memori untuk jenis memori tertentu. SSE4 sepenuhnya kompatibel dengan perangkat lunak dari generasi sebelumnya dari prosesor Intel dan tidak memerlukan dukungan OS baru di luar apa yang diperlukan untuk Streaming SIMD Extensions (SSE) sebelumnya[20].

BAB 3

METODE PENGUJIAN TUNING PARAMETER HYPERVISOR KVM DENGAN CLOUDSTACK AGENT

Penelitian ini bertujuan untuk mengevaluasi konfigurasi Hypervisor KVM yang disediakan secara langsung dalam konteks penggunaan *Virtual Machine* pada Apache Cloudstack. Tujuan utama adalah untuk memastikan bahwa *Virtual Machine* dapat mencapai kinerja yang optimal sesuai dengan spesifikasi sistem yang digunakan. Untuk mencapai tujuan ini, Penulis akan melakukan penyesuaian konfigurasi atau tuning pada Hypervisor KVM dan kemudian membandingkannya dengan kondisi awal yang tidak mengalami tuning. Hasil pengujian akan dianalisis untuk menentukan apakah ada perbedaan signifikan dalam kinerja antara Hypervisor KVM yang telah dituning dengan yang belum dituning.

3.1 Tahapan Penelitian



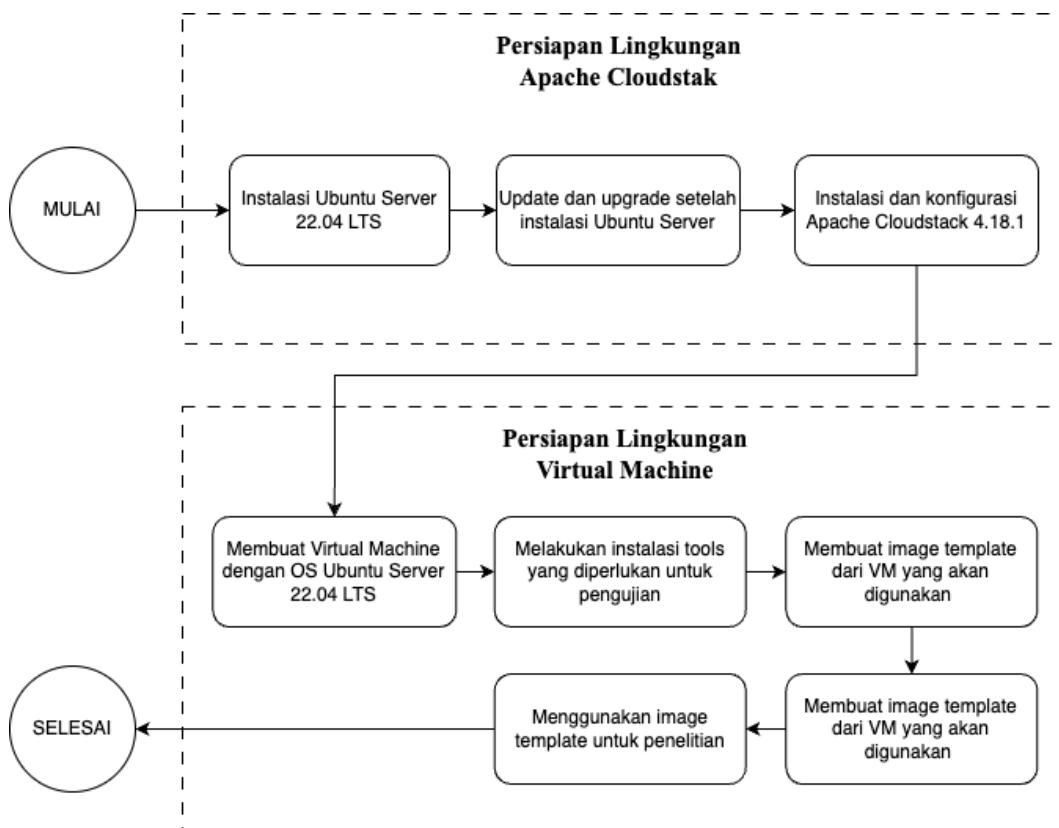
Gambar 3.1: Tahapan Penelitian

Penelitian ini akan dilakukan dalam beberapa tahap sesuai dengan diagram di gambar 3.1. Tahap pertama melibatkan studi literatur untuk mengumpulkan semua informasi yang diperlukan. Studi literatur ini penting karena memberikan dasar pengetahuan yang kuat tentang topik yang akan diteliti, memungkinkan peneliti untuk memahami konteks dan kerangka kerja yang relevan.

Selanjutnya, penelitian melibatkan tuning terhadap Hypervisor KVM, dan pengujian dilakukan dengan tiga skenario, yaitu kompresi video, enkripsi data dengan AES, dan validasi integritas data menggunakan CRC32. Hasil dari pengujian ini akan memberikan gambaran mengenai konfigurasi yang paling optimal dan sesuai,

serta membantu dalam pemahaman lebih lanjut tentang pengaruh tuning terhadap performa Hypervisor KVM.

3.2 Persiapan Lingkungan untuk Pengujian dan Analisis



Gambar 3.2: Persiapan Lingkungan Penelitian

Proses persiapan lingkungan untuk pengujian dan analisis yang tergambar dalam gambar 3.2 menggambarkan serangkaian langkah sistematis yang penting untuk memastikan integritas hasil penelitian. Proses ini dimulai dengan tahap inisiasi yang meliputi instalasi Ubuntu Server 22.04 LTS, sebuah sistem operasi yang stabil dan sering digunakan untuk keperluan server. Instalasi ini menjadi landasan dasar bagi infrastruktur yang akan dikembangkan.

Setelah sistem operasi terinstal, dilakukan pembaruan dan peningkatan sistem operasi tersebut untuk memastikan semua komponen sistem berada pada versi terkini serta keamanan sistem terjaga. Langkah ini krusial untuk menutup potensi kerentanan yang mungkin ada pada sistem. Selanjutnya, Apache Cloudstack versi 4.18.1 diinstal dan dikonfigurasikan. Apache Cloudstack berfungsi sebagai platform manajemen cloud yang memungkinkan pembuatan dan pengelolaan infrastruktur cloud yang kompleks, yang merupakan elemen kunci dalam penelitian ini.

Dalam pembuatan lingkungan virtual, dibuat *Virtual Machine* yang menggunakan sistem operasi yang sama, yaitu Ubuntu Server 22.04 LTS. Pembuatan VM ini memungkinkan simulasi berbagai skenario penelitian dalam lingkungan yang terisolasi. Selanjutnya, untuk keperluan analisis dan pengujian, berbagai perangkat lunak yang akan digunakan untuk melakukan pengujian akan di install.

Tahapan selanjutnya merupakan pembuatan image template dari VM yang telah dikonfigurasikan. Proses ini memungkinkan duplikasi VM dengan cepat dan efisien untuk pengujian berulang atau skenario analisis yang berbeda, menjamin konsistensi lingkungan penelitian. Image template ini kemudian digunakan sebagai standar untuk penelitian lebih lanjut, memastikan bahwa setiap VM yang dibuat untuk tujuan pengujian memiliki konfigurasi yang identik.

Setelah semua dilakukan hal ini menandakan bahwa lingkungan penelitian telah siap untuk diuji dan dianalisis. Kesiapan lingkungan ini esensial untuk memastikan bahwa pengujian yang dilakukan dapat diulang dengan cepat dan efisien.

3.2.1 Instalasi dan Konfigurasi Apache Cloudstack

Dalam instalasi ini akan digunakan *Home Network* di IP 192.168.0.0/24. Pertama pastikan bahwa *tools* yang dibutuhkan sudah terinstall dengan perintah:

```
sudo apt-get install -y openntpd openssh-server vim
htop tar
sudo apt-get install -y intel-microcode
```

Perintah ini akan menginstall *tools* openntpd, openssh-server, vim, htop, tar, dan intel-microcode. Setelah melakukan instalasi *tools* ini kemudian Penulis akan mengganti password dari user root dengan perintah:

```
sudo passwd root
```

3.2.1.1 Pengaturan Jaringan

Melakukan pengaturan *Linux Bridge* yang akan menghandle jaringan CloudStack *public, guest, management* dan *storage*. Untuk penyederhanaan Penulis akan menggunakan satu *bridge* yang bernama **cloudbr0** yang akan digunakan untuk semua jaringan ini. Untuk menginstall *tools* untuk melakukan *bridge* digunakan perintah:

```
sudo apt-get install bridge-utils
```

Setelah menginstall *tools* ini, Penulis akan mengkonfigurasi *bridge* cloudbr0 dengan menggunakan netplan. Pertama pastikan bahwa seleuruh konfigurasi netplan

sekarang di rename dengan menambahkan extensi .bak sebagai backup. Setelah itu buat file konfigruasi netplan dengan perintah:

```
sudo nano /etc/netplan/01-netcfg.yaml
```

Dengan perintah ini akan membuka text editor nano dengan file 01-netcfg.yaml. Isi file konfigurasi tersebut dengan:

```

1  network:
2    version: 2
3    renderer: networkd
4    ethernets:
5      enp2s0:
6        dhcp4: false
7        dhcp6: false
8        optional: true
9    bridges:
10      cloudbr0:
11        addresses: [192.168.0.10/24]
12        routes:
13          - to: default
14            via: 192.168.0.1
15        nameservers:
16          addresses: [1.1.1.1, 8.8.8.8]
17        interfaces: [enp2s0]
18        dhcp4: false
19        dhcp6: false
20        parameters:
21          stp: false
22          forward-delay: 0

```

Kode 3.1: Konfigurasi Netplan untuk Cloudbr0

Dalam konfigurasi ini IP dari *bridge* cloudbr0 adalah 192.168.0.10/24 dan akan menggunakan DNS 1.1.1.1 dan 8.8.8.8. Setelah melakukan konfigurasi ini Penulis akan mengaktifkan *bridge* cloudbr0 dengan perintah:

```
sudo netplan generate
sudo netplan apply
sudo reboot
```

Dengan perintah ini akan dilakukan netplan generate dan apply, setelah itu sistem melakukan reboot. Sampai sini pembuatan *bridge* cloudbr0 sudah selesai.

3.2.1.2 Instalasi Cloudstack

Pertama kita akan melakukan instalasi cloudstack-management dan juga mysql-server. Database yang akan digunakan adalah MySQL. Kita akan menggunakan perintah:

```
sudo mkdir -p /etc/apt/keyrings
wget -O- http://packages.shapeblue.com/release.asc
| gpg --dearmor | sudo tee /etc/apt/keyrings/
cloudstack.gpg > /dev/null

echo deb [signed-by=/etc/apt/keyrings/cloudstack.
gpg] http://packages.shapeblue.com/cloudstack/
upstream/debian/4.18 / > /etc/apt/sources.list.d
/cloudstack.list \
sudo apt-get update -y
sudo apt-get install cloudstack-management mysql-
server
```

Perintah tersebut digunakan untuk menginstal CloudStack Management Server versi 4.18 dan MySQL Server dari repositori tertentu. Pertama, perintah membuat direktori untuk menyimpan kunci GPG yang digunakan untuk mengotentikasi paket dari repositori CloudStack. Kemudian, kunci GPG tersebut diunduh, didekompresi, dan disimpan dalam direktori yang telah dibuat. Selanjutnya, konfigurasi ditambahkan ke file sources.list.d untuk menandatangani paket-paket dari repositori CloudStack. Setelah itu, daftar paket diperbarui dan CloudStack Management Server serta MySQL Server diinstal menggunakan perintah apt-get. Selain ini juga akan dilakukan instalasi *tools* cloudstack usage dan billing yang bersifat opsional, dengan perintah:

```
sudo apt-get install cloudstack-usage
```

3.2.1.3 Konfigurasi Database

Setelah mysql server selesai di install akan dilakukan konfigurasi setting InnoDB di mysql server dengan perintah:

```
sudo nano /etc/mysql/mysql.conf.d/mysqld.cnf
```

Hapus semua isi file tersebut dan isi dengan:

```

1 [mysqld]
2
3 server_id = 1
4 sql-mode=]
5   ↳ "STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION,ERROR_FOR_DIVISION_BY_ZERO
6 NO_ZERO_DATE,NO_ZERO_IN_DATE,NO_ENGINE_SUBSTITUTION"
7 innodb_rollback_on_timeout=1
8 innodb_lock_wait_timeout=600
9 max_connections=1000
10 log-bin=mysql-bin
11 binlog-format = 'ROW'

```

Kode 3.2: Konfigurasi mysqld.cnf

Simpan konfigurasi dan setelah itu restart mysql server. Hal ini dilakukan agar mysql menggunakan setting yang terbaru. Untuk melakukan restart mysql server dapat dilakukan dengan perintah:

```
systemctl restart mysql
```

Setelah mysql server selesai di restart langkah selanjutnya adalah deploy database untuk cloudstack. Deploy server ini harus dilakukan sebagai user root. Deploy dilakukan dengan perintah:

```
cloudstack-setup-databases cloud:cloud@localhost --
deploy-as=root:password -i 192.168.0.10
```

IP yang digunakan adalah IP dari sistem host. Tunggu sampai deployment selesai, langkah ini akan selesai dalam beberapa menit.

3.2.1.4 Konfigurasi Storage

Pada tahap ini Penulis akan melakukan konfigurasi untuk penyimpanan dari cloudstack. Pertama kita akan melakukan instalasi storage driver yang akan digunakan dengan perintah:

```
sudo apt-get install nfs-kernel-server quota
echo "/export *(rw,async,no_root_squash,
no_subtree_check)" > /etc/exports
sudo mkdir -p /export/primary /export/secondary
sudo exportfs -a
```

Perintah `sudo apt-get install nfs-kernel-server quota` digunakan untuk menginstal perangkat lunak server NFS dan modul

Quota pada sistem Linux. Selanjutnya, perintah echo "/export * (rw,async,no_root_squash,no_subtree_check)" > /etc/exports digunakan untuk menetapkan konfigurasi eksport NFS dengan opsi akses tertentu. Kemudian, perintah sudo mkdir -p /export/primary /export/secondary membuat direktori yang akan diakses oleh klien NFS. Terakhir, perintah sudo exportfs -a mengekspor semua direktori yang telah dikonfigurasi sebelumnya untuk diakses oleh klien NFS. Dengan demikian, rangkaian perintah ini mempersiapkan server NFS untuk berbagi data melalui jaringan. Setelah itu akan dilakukan konfigurasi NFS server dengan perintah:

```
sed -i -e 's/^RPCMOUNTDOPTS="--manage-gids"$/  
RPCMOUNTDOPTS="--p 892 --manage-gids"/g' /etc/  
default/nfs-kernel-server  
sed -i -e 's/^STATDOPTS=$/STATDOPTS="--port 662 --  
outgoing-port 2020"/g' /etc/default/nfs-common  
echo "NEED_STATD=yes" >> /etc/default/nfs-common  
sed -i -e 's/^RPCRQUOTADOPTS=$/RPCRQUOTADOPTS="--p  
875"/g' /etc/default/quota  
service nfs-kernel-server restart
```

Perintah-perintah tersebut digunakan untuk mengubah konfigurasi pada server NFS dan Quota pada sistem Linux. Pertama, menggunakan sed, konfigurasi NFS Mount Daemon dan NFS Stat Daemon diubah untuk menggunakan port yang ditentukan dan mengaktifkan manajemen GID. Selanjutnya, layanan Stat Daemon diaktifkan dengan menambahkan opsi yang sesuai ke file konfigurasi NFS Common. Kemudian, konfigurasi RPC Quota Daemon diubah untuk menggunakan port yang ditentukan. Terakhir, layanan NFS Kernel Server di-restart untuk menerapkan perubahan konfigurasi yang telah dilakukan.

3.2.1.5 Konfigurasi KVM

Tahap selanjutnya adalah konfigurasi KVM dan cloudstack agent. Pertama install kvm dan cloudstack agent dengan perintah:

```
sudo apt-get install qemu-kvm cloudstack-agent
```

Setelah KVM dan cloudstack agent terinstall langkah selanjutnya adalah menjalankan perintah ini:

```
sudo sed -i -e 's/^\# vnc_listen.*$/vnc_listen =  
"0.0.0.0"/g' /etc/libvirt/qemu.conf
```

```

sudo echo LIBVIRTD_ARGS="--listen \" >> /etc/
    default/libvirtd
sudo systemctl mask libvirtd.socket libvirtd-rot.
    socket libvirtd-admin.socket libvirtd-tls.socket
    libvirtd-tcp.socket
sudo systemctl restart libvirtd

```

Perintah tersebut digunakan untuk mengkonfigurasi libvirt, sebuah toolkit untuk mengelola virtualisasi di Linux. Pertama, perintah sed digunakan untuk mengubah pengaturan dalam file konfigurasi qemu.conf agar libvirt dapat menerima koneksi VNC dari semua interface jaringan. Kemudian, argumen --listen ditambahkan ke konfigurasi libvirtd melalui file /etc/default/libvirtd, yang membuat libvirt mendengarkan koneksi dari interface jaringan. Selanjutnya, perintah systemctl mask digunakan untuk menonaktifkan unit-unit systemd yang terkait dengan libvirtd, seperti socket-soket yang tidak diperlukan, untuk menghindari aktivasi yang tidak diinginkan. Terakhir, layanan libvirtd di-restart agar perubahan konfigurasi dapat diterapkan.

Selanjutnya akan dilakukan konfigurasi default untuk libvirtd dengan perintah:

```

echo 'listen_tls=0' >> /etc/libvirt/libvirtd.conf
echo 'listen_tcp=1' >> /etc/libvirt/libvirtd.conf
echo 'tcp_port = "16509"' >> /etc/libvirt/libvirtd.
    conf
echo 'mdns_adv = 0' >> /etc/libvirt/libvirtd.conf
echo 'auth_tcp = "none"' >> /etc/libvirt/libvirtd .
    conf
systemctl restart libvirtd

```

Perintah-perintah tersebut digunakan untuk mengkonfigurasi default pada file konfigurasi libvirt (libvirtd.conf). Pertama, dengan menambahkan listen_tls=0, libvirt diminta untuk tidak mendengarkan koneksi TLS, sehingga mematikan mode tersebut. Kemudian, listen_tcp=1 mengaktifkan libvirt untuk mendengarkan koneksi TCP/IP. Penambahan tcp_port = "16509" menetapkan port TCP yang akan digunakan oleh libvirt. Selanjutnya, dengan mdns_adv = 0, mDNS advertise untuk penemuan otomatis layanan libvirt dimatikan. Terakhir, auth_tcp = "none" mengatur libvirt untuk menerima koneksi TCP/IP tanpa melakukan otentikasi. Setelah perubahan-perubahan ini diterapkan, layanan libvirt di-restart agar konfigurasi baru dapat mulai berlaku. Penting untuk dicatat bahwa konfigurasi default ini hanya diperlukan saat setup awal, karena saat menambahkan host KVM dalam CloudStack, konfigurasi yang lebih aman menggunakan TLS akan diterap-

kan secara otomatis.

Pada host tertentu ditempat menjalankan docker dan service lainnya, dimungkinkan perlu untuk menambahkan konfigurasi berikut ini di /etc/sysctl.conf lalu jalankan sysctl -p

```
nano /etc/sysctl.conf
echo 'net.bridge.bridge-nf-call-arptables = 0' >> /
etc/sysctl.conf
echo 'net.bridge.bridge-nf-call-iptables = 0' >> /
etc/sysctl.conf
sysctl -p
```

Pada perintah ini pertama, akan dibuka file /etc/sysctl.conf menggunakan editor teks nano. Kemudian, dengan menambahkan baris net.bridge.bridge-nf-call-arptables = 0 dan net.bridge.bridge-nf-call-iptables = 0 ke dalam file tersebut, Ini akan mematikan fungsi kernel yang biasanya dipanggil saat menggunakan Docker dan jembatan jaringan (bridge). Ini penting karena beberapa layanan, seperti Docker, kadang memerlukan konfigurasi khusus agar berjalan dengan lancar di lingkungan host yang kompleks. Terakhir, dengan menjalankan perintah sysctl -p, perubahan konfigurasi yang telah dibuat akan diterapkan tanpa harus me-reboot sistem, sehingga memungkinkan kernel untuk memuat ulang konfigurasi yang baru diterapkan.

Setelah itu kita akan menggenerate host id dengan uuid, hal ini dapat dilakukan dengan perintah:

```
sudo apt-get install uuid
UUID=$(uuid)
echo host_uuid = \"\$UUID\" >> /etc/libvirt/libvirtd.conf
systemctl restart libvirtd
```

Perintah ini digunakan untuk meng-generate UUID (Universally Unique Identifier) untuk host dan mengintegrasikannya ke dalam konfigurasi libvirtd. Perintah `UUID=$(uuid)` digunakan untuk meng-generate UUID secara acak dan menyimpannya dalam variabel `UUID`. `UUID` adalah string panjang yang unik secara global dan digunakan untuk mengidentifikasi entitas secara unik di sistem. Kemudian, perintah `echo host_uuid = \$UUID >> /etc/libvirt/libvirtd.conf` menambahkan baris `host_uuid = "<nilai UUID>"` ke dalam file konfigurasi libvirtd.conf. Fungsi baris ini adalah untuk menyimpan UUID host di kon-

figurasi libvирtd, yang akan digunakan oleh libvirt untuk mengidentifikasi host secara unik dalam lingkungan virtualisasi. Terakhir, perintah systemctl restart libvирtd digunakan untuk merestart layanan libvирtd agar perubahan konfigurasi yang telah dilakukan dapat diterapkan.

3.2.1.6 Konfigurasi Firewall

Untuk melakukan konfigurasi firewall kita dapat menggunakan perintah:

```
# configure firewall rules to allow useful ports
NETWORK=192.168.0.0/24
iptables -A INPUT -s $NETWORK -m state --state NEW
-p udp --dport 111 -j ACCEPT
iptables -A INPUT -s $NETWORK -m state --state NEW
-p tcp --dport 111 -j ACCEPT
iptables -A INPUT -s $NETWORK -m state --state NEW
-p tcp --dport 2049 -j ACCEPT
iptables -A INPUT -s $NETWORK -m state --state NEW
-p tcp --dport 32803 -j ACCEPT
iptables -A INPUT -s $NETWORK -m state --state NEW
-p udp --dport 32769 -j ACCEPT
iptables -A INPUT -s $NETWORK -m state --state NEW
-p tcp --dport 892 -j ACCEPT
iptables -A INPUT -s $NETWORK -m state --state NEW
-p tcp --dport 875 -j ACCEPT
iptables -A INPUT -s $NETWORK -m state --state NEW
-p tcp --dport 662 -j ACCEPT
iptables -A INPUT -s $NETWORK -m state --state NEW
-p tcp --dport 8250 -j ACCEPT
iptables -A INPUT -s $NETWORK -m state --state NEW
-p tcp --dport 8080 -j ACCEPT
iptables -A INPUT -s $NETWORK -m state --state NEW
-p tcp --dport 9090 -j ACCEPT
iptables -A INPUT -s $NETWORK -m state --state NEW
-p tcp --dport 16514 -j ACCEPT

apt-get install iptables-persistent
```

Perintah tersebut bertujuan untuk mengonfigurasi aturan firewall pada sistem

menggunakan iptables, dengan fokus pada pembukaan port-port yang umum digunakan untuk layanan-layanan tertentu. Pertama, baris `NETWORK=192.168.0.0/24` mendefinisikan jaringan yang diizinkan melalui aturan firewall. Selanjutnya, setiap perintah `iptables -A INPUT ... -j ACCEPT` menambahkan aturan iptables untuk mengizinkan koneksi baru pada port tertentu dari jaringan yang telah ditentukan sebelumnya. Misalnya, perintah `iptables -A INPUT -s $NETWORK -m state --state NEW -p udp --dport 111 -j ACCEPT` mengizinkan koneksi UDP baru pada port 111 dari jaringan yang telah ditentukan.

Aturan-aturan tersebut dirancang untuk membuka akses ke port-port yang umum digunakan oleh layanan-layanan seperti NFS, RPC, dan layanan manajemen tertentu. Setelah semua aturan ditambahkan, perintah `apt-get install iptables-persistent` digunakan untuk menginstal paket `iptables-persistent`, yang bertujuan agar konfigurasi aturan firewall yang telah dibuat akan dipertahankan dan diterapkan secara otomatis setiap kali sistem di-boot. Dengan demikian, langkah-langkah ini membantu menjaga keamanan sistem dengan memperbolehkan akses hanya ke port-port yang diperlukan dan memastikan bahwa aturan firewall tidak hilang setelah reboot.

Setelah itu kita harus menonaktifkan apparmor pada libvirtd, dikarenakan cloudstack melakukan berbagai macam hal yang mana bisa terblokir oleh mekanisme pertahanan seperti apparmor[21]. Untuk mematikan apparmor dapat dilakukan dengan perintah:

```
ln -s /etc/apparmor.d/usr.sbin.libvirtd /etc/
      apparmor.d/disable /
ln -s /etc/apparmor.d/usr.lib.libvirt.virt-aa-
      helper /etc/apparmor.d/disable /
apparmor_parser -R /etc/apparmor.d/usr.sbin.
      libvirtd
apparmor_parser -R /etc/apparmor.d/usr.lib.libvirt.
      virt-aa-helper
```

Perintah-perintah tersebut bertujuan untuk menonaktifkan AppArmor pada layanan libvirtd (Libvirt daemon). Langkah pertama adalah membuat symlink (tautan simbolis) dari file konfigurasi AppArmor yang berkaitan dengan libvirtd dan helpernya ke dalam direktori `/etc/apparmor.d/disable/`. Dengan melakukan ini, AppArmor diarahkan untuk mengabaikan atau menonaktifkan aturan-aturan yang telah didefinisikan untuk libvirtd. Selanjutnya, perintah `apparmor_parser -R` digunakan untuk me-refresh aturan-aturan AppArmor yang terkait dengan libvirtd dan helpernya, sehingga aturan-aturan yang telah dinonaktifkan dengan sym-

link dapat diaplikasikan secara efektif.

Namun, penting untuk dicatat bahwa menonaktifkan AppArmor dapat meningkatkan risiko keamanan sistem karena menghilangkan lapisan proteksi yang diberikan oleh AppArmor terhadap layanan tersebut.

3.2.1.7 Menjalankan CloudStack

Sampai tahap ini instalasi cloudstack sudah selesai dan untuk menjalankan cloudstack dapat dengan menjalankan perintah ini:

```
cloudstack-setup-management
```

Setelah server manajemen sudah UP, cloudstack dapat diakses di <http://192.168.0.10:8080> (IP dari cloudbr0) dan masuk dengan kredensial default yaitu admin/admin. Saat masuk pertama kali akan dilanjutkan dengan pengaturan Apache CloudStack. Dikonfigurasi ini dapat mengatur Zone, Network, Pod, Cluster, Host, Primary Storage, dan Secondary Storage. Pada setiap tahapnya dapat dikonfigurasikan dengan:

1. Zone

Name	nama bebas
Public DNS 1	8.8.8.8
Public DNS 2	1.1.1.1
Internal DNS 1	192.168.0.1

2. Network

Gateway	192.168.0.1
Netmask	255.255.255.0
VLAN/VNI	(kosong untuk vlan://untagged, VXLAN untuk vxlan://untagged)
Start IP	192.168.0.20
End IP	192.168.0.50

3. Pod

Name	nama bebas
Gateway	192.168.0.1
Start/end reserved system IPs	192.168.0.51 - 192.168.0.80

4. Guest Traffic

VLAN/VNI range	700-900
----------------	---------

5. Cluster

Name	nama bebas
Hypervisor	KVM

6. Host

Hostname	192.168.0.10
Username	root
Password	password user root

7. Primary Storage

Name	nama bebas
Scope	zone-wide
Protocol	NFS
Server	192.168.0.10 [ubuntu server static IP]
Path	/export/primary

8. Secondary Storage

Provider	NFS
Name	nama bebas
Server	192.168.100.10 [ubuntu server static ip]
Path	/export/secondary

Setelah konfigurasi selesai, pergi ke Navigation>Infrasturcture>Summary dan pastikan seluruh komponen sudah dinyalakan. Sekarang pengguna dapat menambahkan file iso yang diinginkan untuk membuat instance.

3.2.2 Persiapan *Virtual Machine*

Pada tahap ini akan dijelaskan bagaimana cara membuat *Virtual Machine* yang akan digunakan untuk melakukan pengujian. *Virtual Machine* yang akan dibuat menggunakan image Ubuntu Server 22.04 LTS yang telah diinstal sebelumnya.

3.2.2.1 Image Ubuntu Server 22.04 LTS

Langkah pertama yang harus dilakukan adalah mengunduh file image ISO Ubuntu Server 22.04 LTS dari situs resmi Ubuntu dan menyimpannya di sistem host. Agar dapat digunakan oleh CloudStack, file ISO ini perlu di-host agar dapat diakses melalui alamat IP host. Hosting ini akan dilakukan menggunakan Apache2, sebuah aplikasi web server. Setelah konfigurasi Apache2 selesai dan file image ISO dapat diakses melalui IP, langkah selanjutnya adalah membuka CloudStack dan

menuju ke Configuration>Global Settings>All Settings. Di sini, isi konfigurasi "Secstorage allow internal sites" dengan nilai 192.168.0.0/24 (Home network) dan simpan pengaturan tersebut. Setelah itu, navigasikan ke Images>ISOs dan pilih opsi "register ISO". Tambahkan file ISO Ubuntu Server 22.04 LTS yang telah diunduh sebelumnya dengan memasukkan alamat URL image ISO dari sistem host. Tunggu hingga file image siap digunakan.

3.2.2.2 Membuat Image Template

Setelah ISO image sudah siap digunakan langkah selanjutnya adalah membuat image template. Pergi ke Compute>Instances dan pilih Add Instances. Konfigurasi *Virtual Machine* yang akan digunakan adalah seperti ini:

1. Deployment Infrastructure

- Pod: Default
- Cluster: Default
- Host: Default

2. Template/ISO

Klik tab ISOs dan pilih image Ubuntu Server 22.04 LTS yang telah ditambahkan sebelumnya.

3. Compute Offering

Pilih High Instance dengan spesifikasi:

- CPU: 4 vCPU
- Clock Speed: 2.0 GHz
- Memory: 4096 MB

4. Data Disk

Pada penelitian ini *Virtual Machine* akan diberikan data disk berukuran 20GB.

5. Networks

Di bagian network, pilih "Create New Network" dan pilih tab Isolated. Beri nama untuk jaringan tersebut, pilih zona yang sesuai, dan untuk bagian Network Offering pilih "Network Offering used

for CloudStack Kubernetes Service". Isi bagian Gateway dengan 192.168.0.1 (Gateway Home Network) dan Netmask dengan 255.255.255.0. Untuk DNS 1, gunakan 1.1.1.1 (Cloudflare DNS), dan untuk DNS 2, gunakan 8.8.8.8 (Google DNS). Setelah itu, klik OK dan centang jaringan yang baru saja dibuat.

Setelah itu, klik "Launch Instance" dan tunggu sampai instance siap digunakan. Jika instance sudah siap digunakan, buka konsol dari instance dan lakukan instalasi standar Ubuntu Server 22.04 LTS. Setelah instalasi Ubuntu Server selesai, lanjutkan dengan instalasi program-program yang akan digunakan untuk penelitian. Program-program yang digunakan termasuk ffmpeg untuk melakukan kompresi video, python untuk melakukan benchmark enkripsi, dan validasi integritas data.

Setelah selesai menginstal semua program yang diperlukan, matikan *Virtual Machine* dan pergi ke Storage>Volumes. Klik pada titik tiga dari volume yang digunakan oleh *Virtual Machine* yang telah dibuat sebelumnya, lalu pilih opsi Create template from volume. Tunggu hingga proses pembuatan template selesai. Template ini akan sangat berguna untuk memudahkan pembuatan *Virtual Machine* baru dengan konfigurasi dan aplikasi yang sudah disiapkan jika terjadi masalah atau jika dibutuhkan pembuatan *Virtual Machine* baru.

3.3 Tuning Hypervisor KVM

Untuk melakukan tuning pada Hypervisor KVM, digunakan tools virsh, sebuah command line tool yang disediakan oleh virtualization API libvirt untuk mengelola hypervisor seperti KVM, Xen, VMware ESXi, dan lain-lain. Tuning Hypervisor KVM dilakukan dengan menambahkan flag atau instruksi yang tidak terdapat pada konfigurasi KVM default.

```
fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx
mmxext fxsr_opt pdpe1gb rdtscp lm constant_tsc rep_good nopl
nonstop_tsc cpuid extd_apicid aperfmpf rapl pni pclmulqdq
monitor ssse3 fma cx16 sse4_1 sse4_2 movbe popcnt aes xsave avx
f16c rdrand lahf_lm cmp_legacy svm extapic cr8_legacy abm sse4a
misalignsse 3dnowprefetch osvw ibs skinit wdt tce topoext
perfctr_core perfctr_nb bpext perfctr_llc mwaitx cpb cat_l3
cdp_l3 hw_pstate ssbd mba ibrs ibpb stibp vmmcall fsgsbbase
bmi1 avx2 smep bmi2 cqmq rdt_a rdseed adx smap clflushopt clwb
sha_ni xsaveopt xsavec xgetbv1 cqmq_llc cqmq_occup_llc cqmq_mbm_total
cqmq_mbm_local clzero irperf xsaveerptr rdpru wbnoinvd cppc arat
npt lbrv svm_lock nrip_save tsc_scale vmcb_clean flushbyasid
decodeassists pausefilter pfthreshold avic v_vmsave_vmload vgif
v_spec_ctrl umip rdpid overflow_recov succor smca
```

Gambar 3.3: Seluruh flag instruksi di Host

Terlihat pada Gambar 3.3 adalah daftar lengkap flag instruksi yang tersedia di sistem host, dengan teks yang ditebalkan menunjukkan flag instruksi yang sudah diaktifkan secara default oleh hypervisor KVM. Namun, tampak bahwa KVM tidak mengaktifkan semua flag instruksi meskipun sistem host mendukung flag-flag tersebut. Untuk melihat flag yang sudah diaktifkan pada virtual machine dapat dilakukan dengan menggunakan perintah lscpu.

```
fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 cl
flush mmx fxsr sse sse2 ht syscall nx lm rep_good nopl cpuid extd_apic
id tsc_known_freq pni cx16 x2apic hypervisor lahf_lm cmp_legacy svm 3d
nowprefetch vmmcall
```

Gambar 3.4: Flag default KVM

Terlihat pada gambar 3.4 sama dengan Gambar 3.3 di text yang ditebalkan. Dalam penelitian ini, flag yang akan diuji adalah ssse3, sse4.1, sse4.2, sse4a, dan aes. Semua flag SSE akan digunakan untuk percobaan kompresi video, sementara sse4.2 akan digunakan untuk validasi integritas data dengan CRC32, dan flag AES akan digunakan untuk enkripsi menggunakan AES. Harapannya, tuning ini akan memberikan peningkatan kinerja yang signifikan dibandingkan dengan konfigurasi default dari KVM.

Untuk melakukan tuning flag pada Hypervisor KVM, langkah pertama yang harus dilakukan adalah melihat daftar lengkap virtual machine yang sedang berjalan

di sistem. Hal ini dapat dilakukan dengan menggunakan perintah:

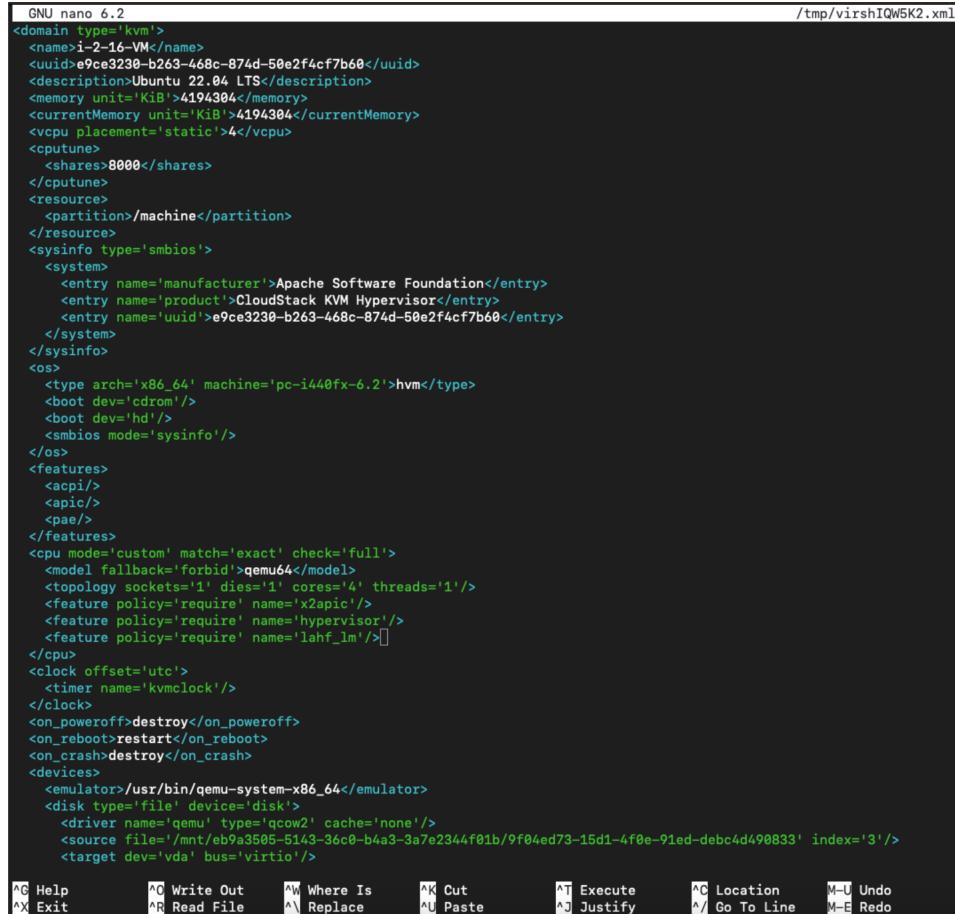
```
virsh list
```

Keluaran dari perintah ini adalah:

Id	Name	State
2	v-1-VM	running
3	s-2-VM	running
20	r-12-VM	running
31	i-2-16-VM	running

Gambar 3.5: Keluaran perintah virsh list

Pada Gambar 3.5 terlihat daftar virtual machine yang sedang berjalan di sistem. Dari daftar ini, virtual machine yang akan di-tuning adalah i-2-16-VM. Setelah mendapatkan nama virtual machine yang akan di-tuning (i-2-16-VM), gunakan perintah virsh edit i-2-16-VM untuk melihat konfigurasi virtual machine tersebut.



```

GNU nano 6.2
<domain type='kvm'>
  <name>i-2-16-VM</name>
  <uuid>e9ce3230-b263-468c-874d-50e2f4cf7b60</uuid>
  <description>Ubuntu 22.04 LTS</description>
  <memory unit='KiB'>4194304</memory>
  <currentMemory unit='KiB'>4194304</currentMemory>
  <vcpu placement='static'>4</vcpu>
  <cputune>
    <shares>8000</shares>
  </cputune>
  <resource>
    <partition>/machine</partition>
  </resource>
  <sysinfo type='smbios'>
    <system>
      <entry name='manufacturer'>Apache Software Foundation</entry>
      <entry name='product'>CloudStack KVM Hypervisor</entry>
      <entry name='uuid'>e9ce3230-b263-468c-874d-50e2f4cf7b60</entry>
    </system>
  </sysinfo>
  <os>
    <type arch='x86_64' machine='pc-i440fx-6.2'>hvm</type>
    <boot dev='cdrom' />
    <boot dev='hd' />
    <smbios mode='sysinfo' />
  </os>
  <features>
    <acpi />
    <apic />
    <pae />
  </features>
  <cpu mode='custom' match='exact' check='full'>
    <model fallback='forbid'>qemu64</model>
    <topology sockets='1' dies='1' cores='4' threads='1' />
    <feature policy='require' name='x2apic' />
    <feature policy='require' name='hypervisor' />
    <feature policy='require' name='lahf_lm' />
  </cpu>
  <clock offset='utc'>
    <timer name='kvmclock' />
  </clock>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>destroy</on_crash>
  <devices>
    <emulator>/usr/bin/qemu-system-x86_64</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2' cache='none' />
      <source file='/mnt/e9ce3230-5143-36c0-b4a3-3a7e2344f01b/9f04ed73-15d1-4f0e-91ed-debc4d490833' index='3' />
      <target dev='vda' bus='virtio' />
    </disk>
  </devices>
</domain>

```

The terminal window shows the XML configuration of the virtual machine. The configuration includes details such as the domain type (kvm), name (i-2-16-VM), UUID, memory, vcpu count, cputune, resource, sysinfo (smbios), os (type hvm, arch x86_64, machine pc-i440fx-6.2), features (acpi, apic, pae), and a specific CPU model (qemu64) with topology and feature requirements. It also defines a clock with a timer named 'kvmclock', power management (on_poweroff, on_reboot, on_crash), and device configuration (emulator, disk).

Gambar 3.6: Perintah virsh edit i-2-16-VM untuk melihat konfigurasi virtual machine i-2-16-VM

Pada gambar 3.6 terlihat konfigurasi virtual machine i-2-16-VM. Dari konfigurasi ini kita akan fokus pada bagian:

```

1 ...
2
3 <cpu mode='custom' match='exact' check='full'>
4   <model fallback='forbid'>qemu64</model>
5   <topology sockets='1' dies='1' cores='4' threads='1' />
6   <feature policy='require' name='x2apic' />
7   <feature policy='require' name='hypervisor' />
8   <feature policy='require' name='lahf_1m' />
9 </cpu>
10 ...
11 ...

```

Kode 3.3: Konfigurasi flag default

Kode 3.3 menampilkan konfigurasi flag default dari hypervisor KVM. Langkah berikutnya adalah menambahkan flag instruksi yang tidak termasuk dalam konfigurasi KVM default. Untuk melakukan ini, dilakukan duplikasi konfigurasi tersebut ke dalam sebuah file baru, seperti dalam penelitian ini yang akan disimpan di file tuned_kvm.xml. Setelah disimpan di file tuned_kvm.xml, selanjutnya dilakukan penambahan fitur yang diinginkan.

```

1 ...
2
3 <cpu mode='custom' match='exact' check='full'>
4   <model fallback='forbid'>qemu64</model>
5   <topology sockets='1' dies='1' cores='4' threads='1' />
6   <feature policy='require' name='x2apic' />
7   <feature policy='require' name='hypervisor' />
8   <feature policy='require' name='lahf_1m' />
9   <feature policy='require' name='ssse3' />
10 </cpu>
11 ...
12 ...

```

Kode 3.4: Konfigurasi flag ssse3

Pada kode 3.4, fitur ssse3 ditambahkan dengan menambahkan baris <feature policy='require' name='ssse3' /> pada kode tersebut. Setelah mengubah file

tuned_kvm.xml, simpan perubahan pada file tersebut. Selanjutnya, matikan mesin virtual dengan menggunakan perintah `destroy`:

```
virsh destroy i-2-16-VM
```

Setelah virtual machine dimatikan, langkah selanjutnya adalah membuat virtual machine baru berdasarkan konfigurasi yang telah diubah sebelumnya. Hal ini dapat dilakukan dengan menggunakan perintah:

```
virsh create tuned_kvm.xml
```

Dengan perintah ini maka *Virtual Machine* i-2-16-VM akan dibuat menggunakan konfigurasi `tuned_kvm.xml`. Kita dapat melihat perubahan pada *Virtual Machine* i-2-16-VM dengan menggunakan perintah `lscpu` pada console virtual machine.

```
fpu de pse tsc msr pae mce cx8 apic sep mttr pge mca cmov pat pse36 cl
flush mmx fxsr sse sse2 ht syscall nx lm rep_good nopl cpuid extd_apic
id tsc_known_freq pni ssse3 cx16 x2apic hypervisorlahf_lm cmp_legacy
svm 3dnowprefetch vmmcall
```

Gambar 3.7: Flag tuned dengan ssse3

Dengan ini menandakan bahwa flag ssse3 pada *Virtual Machine* i-2-16-VM sudah diaktifkan dan Hypervisor KVM sudah berhasil dituning.

3.3.1 Konfigurasi Hypervisor KVM untuk Kompresi Video

Pengujian kompresi video akan menggunakan konfigurasi seperti terlihat pada kode 3.5 yaitu dengan menggunakan flag ssse3, sse4.1, sse4.2, dan sse4a. Pemilihan flag-flag ini didasarkan pada instruksi SIMD (Single Instruction, Multiple Data) yang digunakan untuk operasi vektor pada data. Instruksi SIMD memungkinkan prosesor melakukan operasi secara bersamaan pada beberapa elemen data, sehingga mempercepat proses kompresi video dengan memanfaatkan kemampuan tersebut.

```

1 ...
2
3 <cpu mode='custom' match='exact' check='full'>
4   <model fallback='forbid'>qemu64</model>
5   <topology sockets='1' dies='1' cores='4' threads='1' />
6   <feature policy='require' name='x2apic' />
7   <feature policy='require' name='hypervisor' />
8   <feature policy='require' name='lahf_1m' />
9   <feature policy='require' name='ssse3' />
10  <feature policy='require' name='sse4.1' />
11  <feature policy='require' name='sse4.2' />
12  <feature policy='require' name='sse4a' />
13 </cpu>
14 ...
15 ...

```

Kode 3.5: Konfigurasi Hypervisor KVM untuk Kompresi Video

3.3.2 Konfigurasi Hypervisor KVM untuk Validasi Integritas Data

Untuk melakukan pengujian dengan validasi integritas data akan digunakan konfigurasi seperti pada kode 3.6. Flag yang akan digunakan adalah sse4.2. Flag ini dipilih dikarenakan salah satu kemampuan yang dimiliki oleh flag ini adalah mendukung instruksi CRC32 yang akan digunakan untuk validasi integritas data.

```

1 ...
2
3 <cpu mode='custom' match='exact' check='full'>
4   <model fallback='forbid'>qemu64</model>
5   <topology sockets='1' dies='1' cores='4' threads='1' />
6   <feature policy='require' name='x2apic' />
7   <feature policy='require' name='hypervisor' />
8   <feature policy='require' name='lahf_1m' />
9   <feature policy='require' name='sse4.2' />
10 </cpu>
11 ...
12 ...

```

Kode 3.6: Konfigurasi Hypervisor KVM untuk Validasi Integritas Data

3.3.3 Konfigurasi Hypervisor KVM untuk Enkripsi AES

Untuk melakukan pengujian dengan enkripsi AES akan digunakan konfigurasi seperti pada kode 3.7. Flag yang akan digunakan adalah aes. Flag ini dipilih dikarenakan salah satu kemampuan yang dimiliki oleh flag ini adalah mendukung instruksi AES yang akan digunakan untuk enkripsi AES.

```

1 ...
2
3 <cpu mode='custom' match='exact' check='full'>
4   <model fallback='forbid'>qemu64</model>
5   <topology sockets='1' dies='1' cores='4' threads='1' />
6   <feature policy='require' name='x2apic' />
7   <feature policy='require' name='hypervisor' />
8   <feature policy='require' name='lahf_lm' />
9   <feature policy='require' name='aes' />
10 </cpu>
11 ...
12 ...

```

Kode 3.7: Konfigurasi Hypervisor KVM untuk Enkripsi AES

3.4 Skenario Pengujian Tuning

Pada bagian ini akan dijelaskan skenario pengujian yang akan dilakukan untuk mengukur dampak tuning yang dilakukan pada Hypervisor KVM terhadap kinerja virtual machine. Pengujian akan dilakukan dengan menggunakan tiga skenario yang berbeda, yaitu kompresi video, validasi integritas data, dan enkripsi AES. Setiap skenario akan dijalankan pada virtual machine yang telah dituning dengan konfigurasi yang sesuai.

3.4.1 Skenario Pengujian Kompresi Video

Pada pengujian kompresi video akan dilakukan dengan menggunakan perangkat lunak ffmpeg untuk melakukan kompresi video. Pengujian ini akan mengukur waktu yang diperlukan untuk melakukan kompresi video dengan menggunakan konfigurasi tuning yang telah dilakukan pada Hypervisor KVM. Pengujian ini akan dilakukan dengan menggunakan video yang berjudul Times Square Day Wide yang memiliki durasi 10 detik dan resolusi 4k. Video ini akan dikompresi menggunakan codec h265 dengan preset medium. Pengujian akan dijalankan sebanyak 10 kali dan

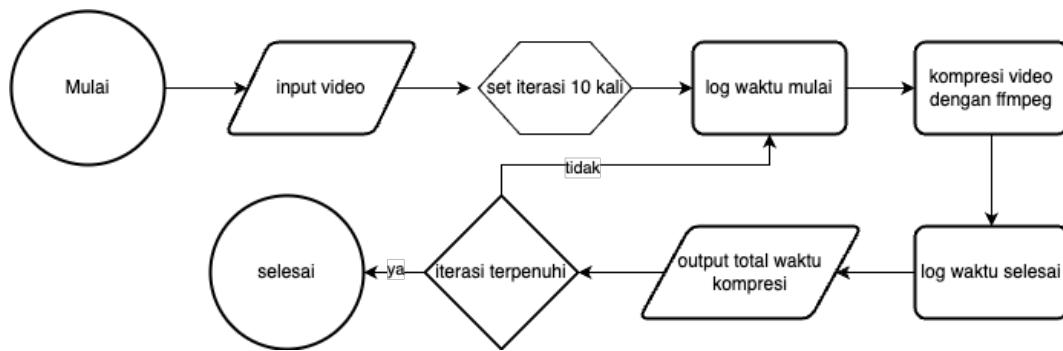
diukur waktu yang diperlukan untuk menyelesaikan proses kompresi video. Skrip bash yang digunakan untuk melakukan pengujian kompresi video dapat dilihat pada kode 3.8.

```

1  #!/bin/bash
2
3  # Specify the path to your input video file
4  input_file="timessquaredaywide.mp4"
5
6  # Specify the output directory
7  output_directory="output"
8
9  # Specify the number of times to run the benchmark
10 num_runs=10
11
12 # Loop to run the benchmark command multiple times
13 for i in $(seq 1 $num_runs)
14 do
15     start_time=$(date +%s.%N)
16     log_file="$output_directory/log_$i.txt"
17
18     echo "Running benchmark $i..."
19
20     # Add your ffmpeg compression command here
21     ffmpeg -i "$input_file" -c:v libx265 -preset medium -crf 32 -c:a aac
22     ↳ -strict -2 "$output_directory/output_$i.mp4" > "$log_file" 2>&1
23
24     end_time=$(date +%s.%N)
25     elapsed_time=$(awk "BEGIN {print $end_time - $start_time}")
26
27     echo "Benchmark $i completed in $elapsed_time seconds. Log saved to $"
     ↳ log_file.
done

```

Kode 3.8: Kode Pengujian Kompresi Video



Gambar 3.8: Flowchart Kode Pengujian Kompresi Video

3.4.2 Skenario Pengujian Validasi Integritas Data

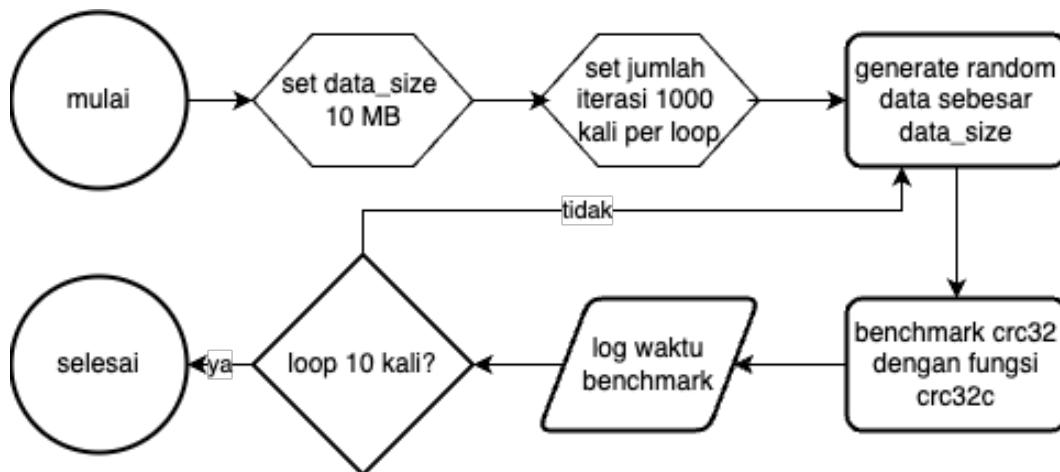
Pada pengujian validasi integritas data akan dilakukan dengan menggunakan library crc32c dari python untuk melakukan perhitungan CRC32 pada file. Pengujian ini akan mengukur waktu yang diperlukan untuk melakukan perhitungan CRC32 pada file dengan ukuran 10 MB. Pengujian ini akan dijalankan sebanyak 10 kali dan diukur waktu yang diperlukan untuk menyelesaikan proses perhitungan CRC32. Skrip python yang digunakan untuk melakukan pengujian validasi integritas data dapat dilihat pada kode 3.9.

```

1 import os
2 import time
3 import crc32c
4
5 def benchmark_crc32c(data, iterations):
6     # Perform CRC32 calculation
7     start_time = time.time()
8     for _ in range(iterations):
9         crc32c.crc32c(data)
10    end_time = time.time()
11
12    # Calculate average time per iteration
13    avg_time = (end_time - start_time) / iterations
14    return avg_time
15
16 def main():
17     # Define parameters
18     data_size = 10 * 1024 * 1024 # Size of the data (10 MB)
19     num_iterations = 1000 # Number of iterations to average the time
20
21     for i in range(10):
22         # Generate random data
23         data = bytearray(os.urandom(data_size))
24
25         # Run the benchmark
26         avg_time_crc32c = benchmark_crc32c(data, num_iterations)
27         print(
28             f"Run {i +
29                 1} Average time for CRC32 calculation: {avg_time_crc32c
30                 * 1000} ms"
31     )
32
33 main()

```

Kode 3.9: Kode Pengujian Validasi Integritas Data

**Gambar 3.9:** Flowchart Kode Pengujian Validasi Integritas Data

3.4.3 Skenario Pengujian Enkripsi AES

Pada pengujian enkripsi AES akan dilakukan dengan menggunakan library cryptography dari python untuk melakukan enkripsi AES. Pengujian ini akan mengukur waktu yang diperlukan untuk melakukan enkripsi AES pada data teks Gambar 3.10. Pengujian ini akan dijalankan sebanyak 10 kali dan diukur waktu yang diperlukan untuk menyelesaikan proses enkripsi AES. Skrip python yang digunakan untuk melakukan pengujian enkripsi AES dapat dilihat pada kode 3.11.

Gambar 3.10: Potongan Plaintext Data untuk AES Encryption Benchmark

```

1  from cryptography.hazmat.primitives.ciphers import Cipher,
   ↪ algorithms, modes
2  from cryptography.hazmat.primitives import padding
3  from cryptography.hazmat.backends import default_backend
4  import time
5
6  # Generate a random AES key and plaintext
7  key = b"sixteenbytekey12"
8  plaintext = # Gambar Potongan Plaintext Data untuk AES Encryption
   ↪ Benchmark
9
10 # Function to perform AES encryption benchmark
11 def aes_encrypt_benchmark(key, plaintext):
12     backend = default_backend()
13     cipher = Cipher(algorithms.AES(key), modes.ECB(), backend=backend)
14     encryptor = cipher.encryptor()
  
```

```

15
16     padder = padding.PKCS7(algorithms.AES.block_size).padder()
17     padded_plaintext = padder.update(plaintext) + padder.finalize()
18
19     start_time = time.time()
20     ciphertext = encryptor.update(padded_plaintext) + encryptor.finalize()
21     end_time = time.time()
22
23     return ciphertext, end_time - start_time
24
25 # Function to perform AES decryption benchmark
26 def aes_decrypt_benchmark(key, ciphertext):
27     backend = default_backend()
28     cipher = Cipher(algorithms.AES(key), modes.ECB(), backend=backend)
29     decryptor = cipher.decryptor()
30
31     start_time = time.time()
32     padded_plaintext = decryptor.update(ciphertext) + decryptor.finalize()
33     end_time = time.time()
34
35     unpadder = padding.PKCS7(algorithms.AES.block_size).unpadder()
36     plaintext = unpadder.update(padded_plaintext) + unpadder.finalize()
37
38     return end_time - start_time
39
40 def run_benchmark():
41     # Benchmark AES encryption
42     ciphertext, aes_time_encrypt = aes_encrypt_benchmark(key, plaintext)
43     # print(f"AES Encryption Time: {(aes_time_encrypt * 1000):.5f} ms")
44
45     # Benchmark AES decryption
46     aes_time_decrypt = aes_decrypt_benchmark(key, ciphertext)
47     # print(f"AES Decryption Time: {(aes_time_decrypt * 1000):.5f} ms")
48     return aes_time_encrypt, aes_time_decrypt
49
50 def main():
51     iterations = 1000
52
53     for i in range(10):
54         encrypt_time = 0
55         decrypt_time = 0
56         for _ in range(iterations):
57             encrypt_time_iter, decrypt_time_iter = run_benchmark()
58             encrypt_time += encrypt_time_iter

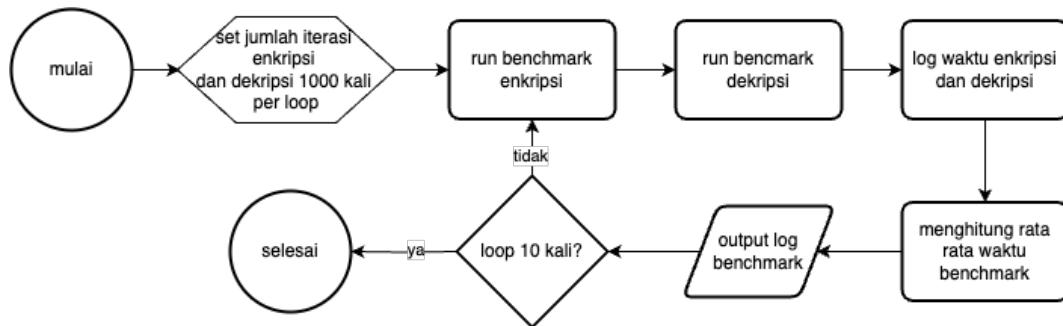
```

```

59     decrypt_time += decrypt_time_iter
60
61     avg_encrypt_time = encrypt_time / iterations
62     avg_decrypt_time = decrypt_time / iterations
63
64     print(f"Run {i+1} Average AES Encryption Time: {(avg_encrypt_time |)
65         e *
66         1000):.5f} ms")
67     print(
68         f"Run {i+1} Average AES Decryption Time: {(avg_decrypt_time |
69         e *
69         1000):.5f} ms\n"
69 )

```

Gambar 3.11: Kode Pengujian Enkripsi AES



Gambar 3.12: Flowchart Kode Pengujian Enkripsi dan Dekripsi AES

3.5 Analisis

Analisis akan dilakukan dengan melakukan perhitungan menggunakan formula speedup. Diasumsikan bahwa $T(1)$ adalah waktu eksekusi Test dengan konfigurasi default, dan $T(p)$ adalah waktu eksekusi Test dengan konfigurasi yang telah dituning[22].

Formula speedup dinyatakan sebagai berikut:

$$S(p) = \frac{T(1)}{T(p)} \quad (3.1)$$

Berdasarkan rumus 3.1, jika nilai $S(p)$ semakin besar, maka konfigurasi yang dituning semakin bagus. Hal ini menandakan bahwa tuning Hypervisor dapat mempercepat proses yang dilakukan. Sebaliknya, jika $S(p)$ semakin kecil, maka konfig-

urasi yang dituning semakin lambat. Jika nilai $S(p)$ sama, maka efek dari konfigurasi yang dituning sama dengan konfigurasi default.

3.5.1 Analisis Pengujian Kompresi Video

Dalam analisis pengujian dengan kompresi video, akan dibandingkan waktu eksekusi antara konfigurasi Hypervisor yang tidak dituning (default) dan konfigurasi yang telah dituning. Perbandingan ini dilakukan dengan menggunakan formula speedup. Konfigurasi yang optimal adalah konfigurasi yang memiliki nilai speedup tertinggi, yang menunjukkan peningkatan kinerja yang signifikan dalam proses kompresi video.

3.5.2 Analisis Pengujian Validasi Integritas Data

Untuk analisis pengujian validasi integritas data, akan digunakan pendekatan yang serupa dengan analisis kompresi video. Akan dibandingkan waktu eksekusi antara konfigurasi Hypervisor yang tidak dituning (default) dan konfigurasi yang telah dituning menggunakan formula speedup. Nilai $S(p)$ yang lebih besar menunjukkan bahwa konfigurasi yang dituning lebih efisien dalam memverifikasi integritas data dibandingkan dengan konfigurasi default. Konfigurasi yang optimal adalah konfigurasi yang memiliki nilai speedup tertinggi, yang mengindikasikan peningkatan kinerja yang signifikan dalam proses validasi integritas data.

3.5.3 Analisis Pengujian Enkripsi AES

Dalam analisis pengujian enkripsi AES, kami akan menggunakan metode yang sama seperti analisis sebelumnya, yaitu dengan membandingkan waktu eksekusi antara konfigurasi Hypervisor yang tidak dituning (default) dan konfigurasi yang telah dituning menggunakan formula speedup. Nilai $S(p)$ yang lebih besar menunjukkan bahwa konfigurasi yang dituning lebih efisien dalam melakukan enkripsi AES dibandingkan dengan konfigurasi default. Konfigurasi yang optimal adalah konfigurasi yang memiliki nilai speedup tertinggi, yang mengindikasikan peningkatan kinerja yang signifikan dalam proses enkripsi AES.

Dengan menganalisis nilai speedup, kami dapat mengevaluasi dan membandingkan kinerja berbagai konfigurasi Hypervisor dalam melakukan tugas-tugas seperti kompresi video, validasi integritas data, dan enkripsi AES. Konfigurasi dengan nilai speedup tertinggi akan menjadi pilihan yang optimal untuk

meningkatkan kinerja dan efisiensi dalam setiap tugas yang diuji.

BAB 4

PENGUJIAN TUNING PARAMETER HYPERVISOR KVM

DENGAN CLOUDSTACK AGENT

Pada bab ini akan dijelaskan hasil dari pengujian tuning Hypervisor KVM. Pengujian dilakukan dengan menggunakan program benchmark yang telah dijelaskan pada bab 3.

4.1 Hasil Pengujian Tuning Hypervisor KVM dengan Kompresi Video

Pada bagian ini akan dijelaskan hasil pengujian tuning Hypervisor KVM dengan menggunakan program benchmark yang telah dibuat sebelumnya dengan melakukan kompresi video.

4.1.1 Konfigurasi Default

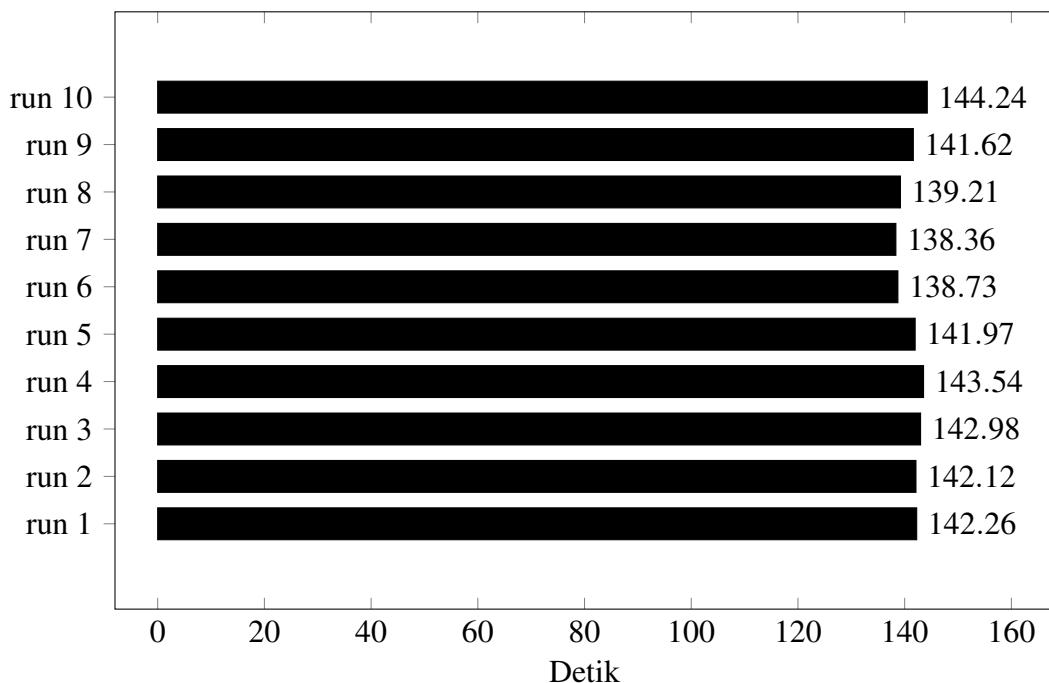
```
fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat  
pse36 clflush mmx fxsr sse sse2 ht syscall nx lm rep_good nopl  
cpuid extd_apicid tsc_known_freq pni cx16 x2apic hypervisor  
lahf_lm cmp_legacy svm 3dnowprefetch vmmcall
```

Gambar 4.1: lscpu Konfigurasi Default

Gambar 4.1 menunjukkan hasil dari perintah `lscpu` pada konfigurasi default. Konfigurasi ini tidak memiliki perubahan flag cpu apapun.

```
ubuntu@ubuntu-server:~$ sh ./compress.sh
Running benchmark 1...
Benchmark 1 completed in 142.257 seconds. Log saved to output/log_1.txt.
Running benchmark 2...
Benchmark 2 completed in 142.116 seconds. Log saved to output/log_2.txt.
Running benchmark 3...
Benchmark 3 completed in 142.983 seconds. Log saved to output/log_3.txt.
Running benchmark 4...
Benchmark 4 completed in 143.54 seconds. Log saved to output/log_4.txt.
Running benchmark 5...
Benchmark 5 completed in 141.966 seconds. Log saved to output/log_5.txt.
Running benchmark 6...
Benchmark 6 completed in 138.732 seconds. Log saved to output/log_6.txt.
Running benchmark 7...
Benchmark 7 completed in 138.355 seconds. Log saved to output/log_7.txt.
Running benchmark 8...
Benchmark 8 completed in 139.209 seconds. Log saved to output/log_8.txt.
Running benchmark 9...
Benchmark 9 completed in 141.622 seconds. Log saved to output/log_9.txt.
Running benchmark 10...
Benchmark 10 completed in 144.236 seconds. Log saved to output/log_10.txt.
ubuntu@ubuntu-server:~$ [compress-0:bash*
```

Gambar 4.2: Test Kompresi Video dengan Konfigurasi Default



Gambar 4.3: Grafik Tes Kompresi Video dengan Konfigurasi Default

Program benchmark dengan kode 3.8 dijalankan pada konfigurasi default Hypervisor KVM. Hasil pengujian dapat dilihat pada gambar 4.2. Dari log yang dihasilkan saat menjalankan program benchmark, waktu yang diperlukan untuk melakukan setiap run benchmark diambil dan digunakan untuk membuat grafik seperti yang terlihat pada gambar 4.3. Grafik tersebut menunjukkan bahwa waktu

yang diperlukan untuk melakukan benchmark pada konfigurasi default berkisar antara 138 detik hingga 144 detik. Rata-rata dari 10 run yang dilakukan adalah 141.5 detik.

4.1.2 Konfigurasi dengan SSSE3

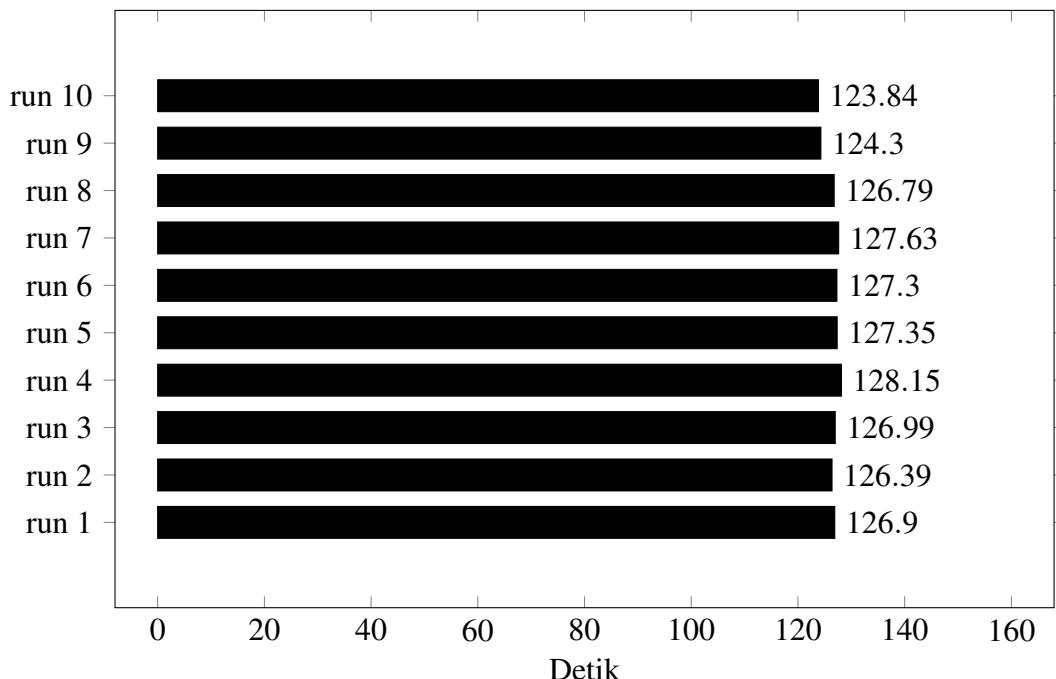
```
fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 ht syscall nx lm rep_good nopl
cpuid extd_apicid tsc_known_freq pni ssse3 cx16 x2apic hypervisor
lahf_lm cmp_legacy svm 3dnowprefetch vmmcall
```

Gambar 4.4: lscpu Konfigurasi SSSE3

Gambar 4.4 menunjukkan hasil dari perintah lscpu pada konfigurasi SSSE3. Konfigurasi ini memiliki perubahan flag cpu ssse3.

```
ubuntu@ubuntu-server:~$ sh ./compress.sh
Running benchmark 1...
Benchmark 1 completed in 126.9 seconds. Log saved to output/log_1.txt.
Running benchmark 2...
Benchmark 2 completed in 126.385 seconds. Log saved to output/log_2.txt.
Running benchmark 3...
Benchmark 3 completed in 126.985 seconds. Log saved to output/log_3.txt.
Running benchmark 4...
Benchmark 4 completed in 128.149 seconds. Log saved to output/log_4.txt.
Running benchmark 5...
Benchmark 5 completed in 127.348 seconds. Log saved to output/log_5.txt.
Running benchmark 6...
Benchmark 6 completed in 127.303 seconds. Log saved to output/log_6.txt.
Running benchmark 7...
Benchmark 7 completed in 127.631 seconds. Log saved to output/log_7.txt.
Running benchmark 8...
Benchmark 8 completed in 126.788 seconds. Log saved to output/log_8.txt.
Running benchmark 9...
Benchmark 9 completed in 124.297 seconds. Log saved to output/log_9.txt.
Running benchmark 10...
Benchmark 10 completed in 123.838 seconds. Log saved to output/log_10.txt.
ubuntu@ubuntu-server:~$
```

Gambar 4.5: Test Kompresi Video dengan Konfigurasi SSSE3



Gambar 4.6: Grafik Tes Kompresi Video dengan Konfigurasi SSSE3

Program benchmark dengan kode 3.8 dijalankan pada konfigurasi SSSE3. Hasil pengujian dapat dilihat pada gambar 4.5. Dari log yang dihasilkan saat menjalankan program benchmark, waktu yang diperlukan untuk melakukan setiap run benchmark diambil dan digunakan untuk membuat grafik seperti yang terlihat pada gambar 4.6. Grafik tersebut menunjukkan bahwa waktu yang diperlukan untuk melakukan benchmark pada konfigurasi SSSE3 berkisar antara 123 detik hingga 128 detik. Rata-rata dari 10 run yang dilakukan adalah 126.5 detik.

4.1.3 Konfigurasi dengan SSE4.1

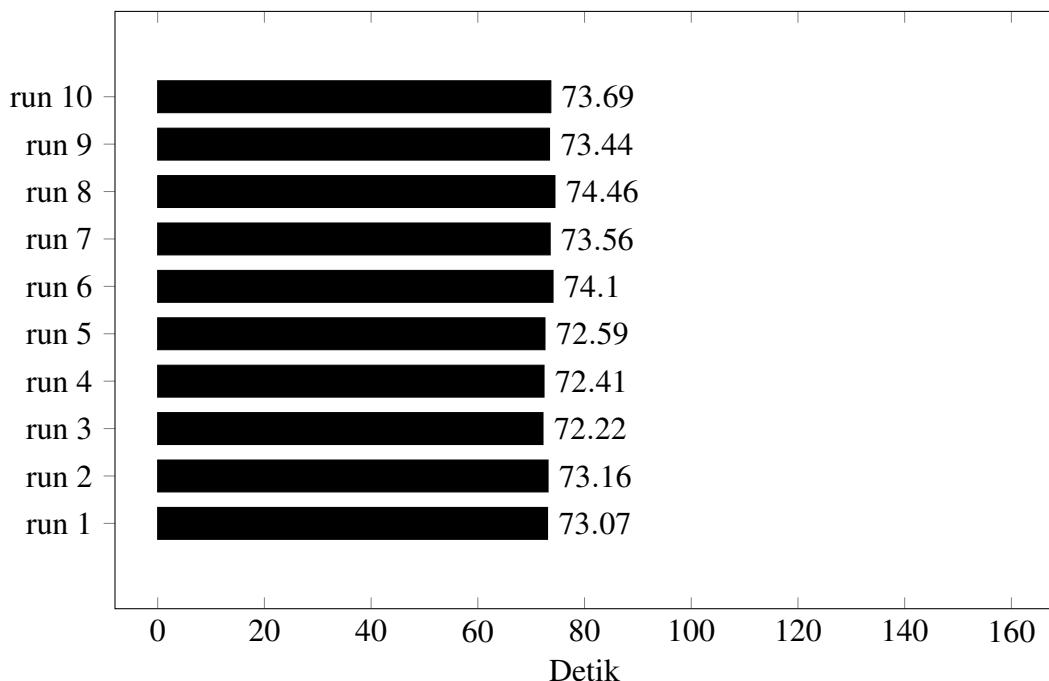
```
fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 ht syscall nx lm rep_good
nopl cpuid extd_apicid tsc_known_freq pni cx16 sse4_1 x2apic
hypervisorlahf_lm cmp_legacy svm 3dnowprefetch vmmcall
```

Gambar 4.7: `lscpu` Konfigurasi SSE4.1

Gambar 4.7 menunjukkan hasil dari perintah `lscpu` pada konfigurasi SSE4.1. Konfigurasi ini memiliki perubahan flag `cpu sse4_1`.

```
ubuntu@ubuntu-server:~$ sh ./compress.sh
Running benchmark 1...
Benchmark 1 completed in 73.0689 seconds. Log saved to output/log_1.txt.
Running benchmark 2...
Benchmark 2 completed in 73.1632 seconds. Log saved to output/log_2.txt.
Running benchmark 3...
Benchmark 3 completed in 72.2159 seconds. Log saved to output/log_3.txt.
Running benchmark 4...
Benchmark 4 completed in 72.4091 seconds. Log saved to output/log_4.txt.
Running benchmark 5...
Benchmark 5 completed in 72.5864 seconds. Log saved to output/log_5.txt.
Running benchmark 6...
Benchmark 6 completed in 74.0975 seconds. Log saved to output/log_6.txt.
Running benchmark 7...
Benchmark 7 completed in 73.5585 seconds. Log saved to output/log_7.txt.
Running benchmark 8...
Benchmark 8 completed in 74.457 seconds. Log saved to output/log_8.txt.
Running benchmark 9...
Benchmark 9 completed in 73.4418 seconds. Log saved to output/log_9.txt.
Running benchmark 10...
Benchmark 10 completed in 73.6852 seconds. Log saved to output/log_10.txt.
ubuntu@ubuntu-server:~$
```

Gambar 4.8: Test Kompresi Video dengan Konfigurasi SSE4.1



Gambar 4.9: Grafik Tes Kompresi Video dengan Konfigurasi SSE4.1

Program benchmark dengan kode 3.8 dijalankan pada konfigurasi SSE4.1. Hasil pengujian dapat dilihat pada gambar 4.8. Dari log yang dihasilkan saat menjalankan program benchmark, waktu yang diperlukan untuk melakukan setiap run benchmark diambil dan digunakan untuk membuat grafik seperti yang terlihat pada gambar 4.9. Grafik tersebut menunjukkan bahwa waktu yang diperlukan untuk

melakukan benchmark pada konfigurasi SSE4.1 berkisar antara 72 detik hingga 74 detik. Rata-rata dari 10 run yang dilakukan adalah 73.2 detik.

4.1.4 Konfigurasi dengan SSE4.2

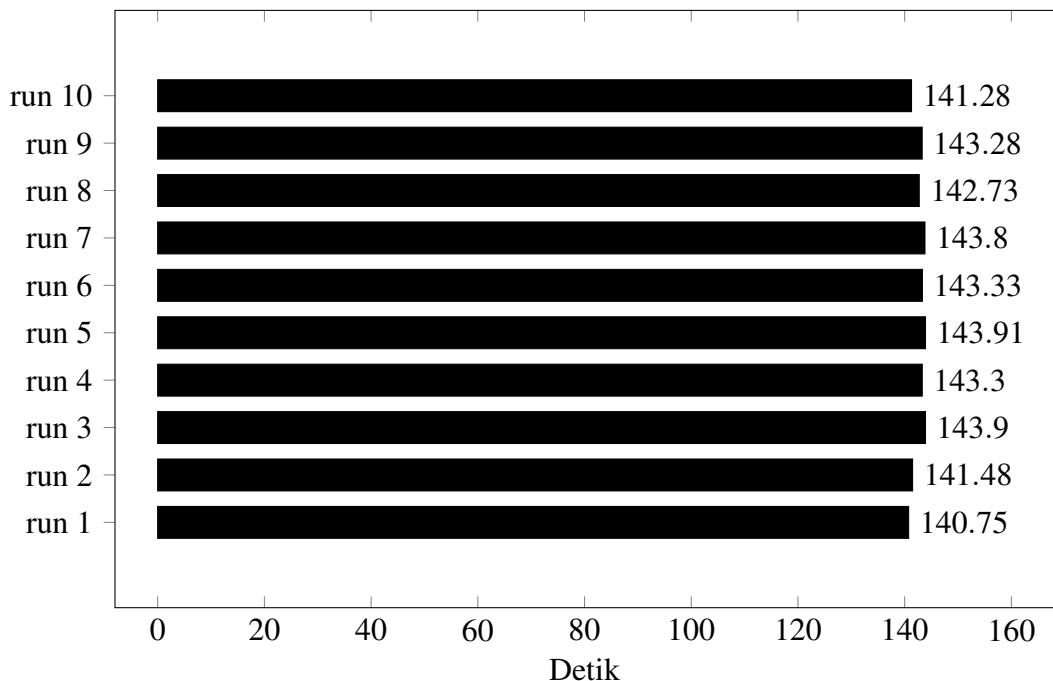
```
fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 ht syscall nx lm rep_good
nopl cpuid extd_apicid tsc_known_freq pni cx16 sse4_2 x2apic
hypervisorlahf_lm cmp_legacy svm 3dnowprefetch vmmcall
```

Gambar 4.10: lscpu Konfigurasi SSE4.2

Gambar 4.10 menunjukkan hasil dari perintah lscpu pada konfigurasi SSE4.2. Konfigurasi ini memiliki perubahan flag cpu sse4.2.

```
ubuntu@ubuntu-server:~$ sh ./compress.sh
Running benchmark 1...
Benchmark 1 completed in 140.747 seconds. Log saved to output/log_1.txt.
Running benchmark 2...
Benchmark 2 completed in 141.484 seconds. Log saved to output/log_2.txt.
Running benchmark 3...
Benchmark 3 completed in 143.902 seconds. Log saved to output/log_3.txt.
Running benchmark 4...
Benchmark 4 completed in 143.299 seconds. Log saved to output/log_4.txt.
Running benchmark 5...
Benchmark 5 completed in 143.905 seconds. Log saved to output/log_5.txt.
Running benchmark 6...
Benchmark 6 completed in 143.325 seconds. Log saved to output/log_6.txt.
Running benchmark 7...
Benchmark 7 completed in 142.803 seconds. Log saved to output/log_7.txt.
Running benchmark 8...
Benchmark 8 completed in 142.732 seconds. Log saved to output/log_8.txt.
Running benchmark 9...
Benchmark 9 completed in 143.276 seconds. Log saved to output/log_9.txt.
Running benchmark 10...
Benchmark 10 completed in 141.284 seconds. Log saved to output/log_10.txt.
ubuntu@ubuntu-server:~$ _
```

Gambar 4.11: Test Kompresi Video dengan Konfigurasi SSE4.2



Gambar 4.12: Grafik Tes Kompresi Video dengan Konfigurasi SSE4.2

Program benchmark dengan kode 3.8 dijalankan pada konfigurasi SSE4.2. Hasil pengujian dapat dilihat pada gambar 4.11. Dari log yang dihasilkan saat menjalankan program benchmark, waktu yang diperlukan untuk melakukan setiap run benchmark diambil dan digunakan untuk membuat grafik seperti yang terlihat pada gambar 4.12. Grafik tersebut menunjukkan bahwa waktu yang diperlukan untuk melakukan benchmark pada konfigurasi SSE4.2 berkisar antara 140 detik hingga 143 detik. Rata-rata dari 10 run yang dilakukan adalah 142.7 detik.

4.1.5 Konfigurasi dengan SSE4a

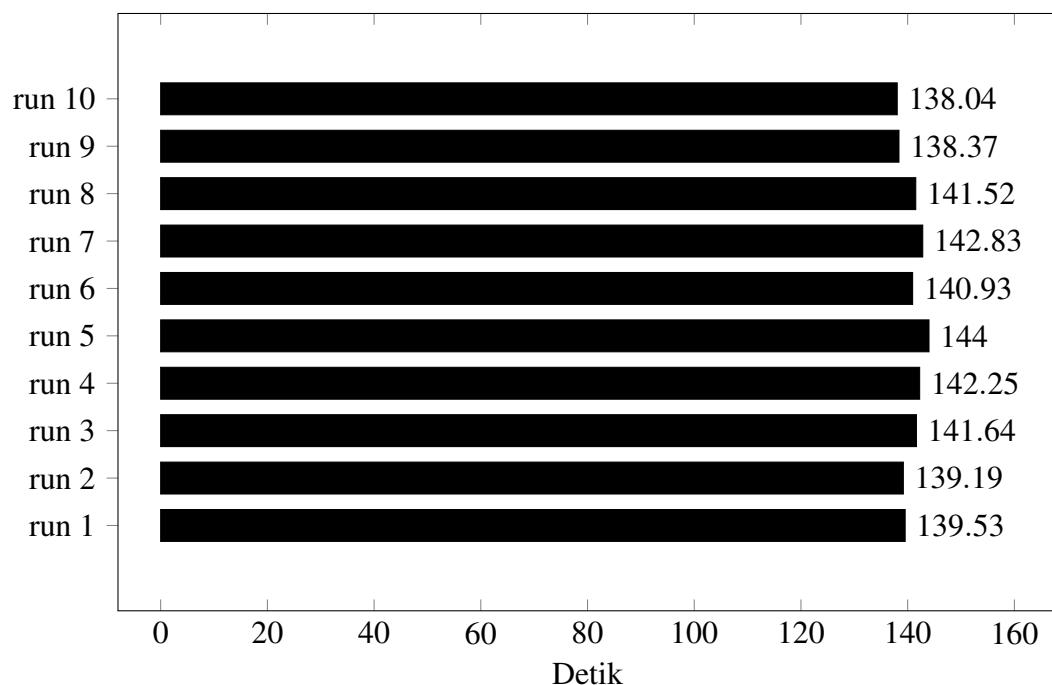
```
fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 ht syscall nx lm rep_good nopl
cpuid extd_apicid tsc_known_freq pni cx16 x2apic hypervisor
lahf_lm cmp_legacy svm sse4a 3dnowprefetch vmmcall
```

Gambar 4.13: lscpu Konfigurasi SSE4a

Gambar 4.13 menunjukkan hasil dari perintah `lscpu` pada konfigurasi SSE4a. Konfigurasi ini memiliki perubahan flag `cpu sse4a`.

```
ubuntu@ubuntu-server:~$ sh ./compress.sh
Running benchmark 1...
Benchmark 1 completed in 139.528 seconds. Log saved to output/log_1.txt.
Running benchmark 2...
Benchmark 2 completed in 139.188 seconds. Log saved to output/log_2.txt.
Running benchmark 3...
Benchmark 3 completed in 141.641 seconds. Log saved to output/log_3.txt.
Running benchmark 4...
Benchmark 4 completed in 142.246 seconds. Log saved to output/log_4.txt.
Running benchmark 5...
Benchmark 5 completed in 143.997 seconds. Log saved to output/log_5.txt.
Running benchmark 6...
Benchmark 6 completed in 140.927 seconds. Log saved to output/log_6.txt.
Running benchmark 7...
Benchmark 7 completed in 142.825 seconds. Log saved to output/log_7.txt.
Running benchmark 8...
Benchmark 8 completed in 141.517 seconds. Log saved to output/log_8.txt.
Running benchmark 9...
Benchmark 9 completed in 138.368 seconds. Log saved to output/log_9.txt.
Running benchmark 10...
Benchmark 10 completed in 138.04 seconds. Log saved to output/log_10.txt.
ubuntu@ubuntu-server:~$
```

Gambar 4.14: Test Kompresi Video dengan Konfigurasi SSE4a



Gambar 4.15: Grafik Tes Kompresi Video dengan Konfigurasi SSE4a

Program benchmark dengan kode 3.8 dijalankan pada konfigurasi SSE4a. Hasil pengujian dapat dilihat pada gambar 4.14. Dari log yang dihasilkan saat menjalankan program benchmark, waktu yang diperlukan untuk melakukan setiap run benchmark diambil dan digunakan untuk membuat grafik seperti yang terlihat pada gambar 4.15. Grafik tersebut menunjukkan bahwa waktu yang diperlukan untuk

melakukan benchmark pada konfigurasi SSE4a berkisar antara 138 detik hingga 143 detik. Rata-rata dari 10 run yang dilakukan adalah 140.8 detik.

4.1.6 Konfigurasi dengan SSSE3 + SSE4.1

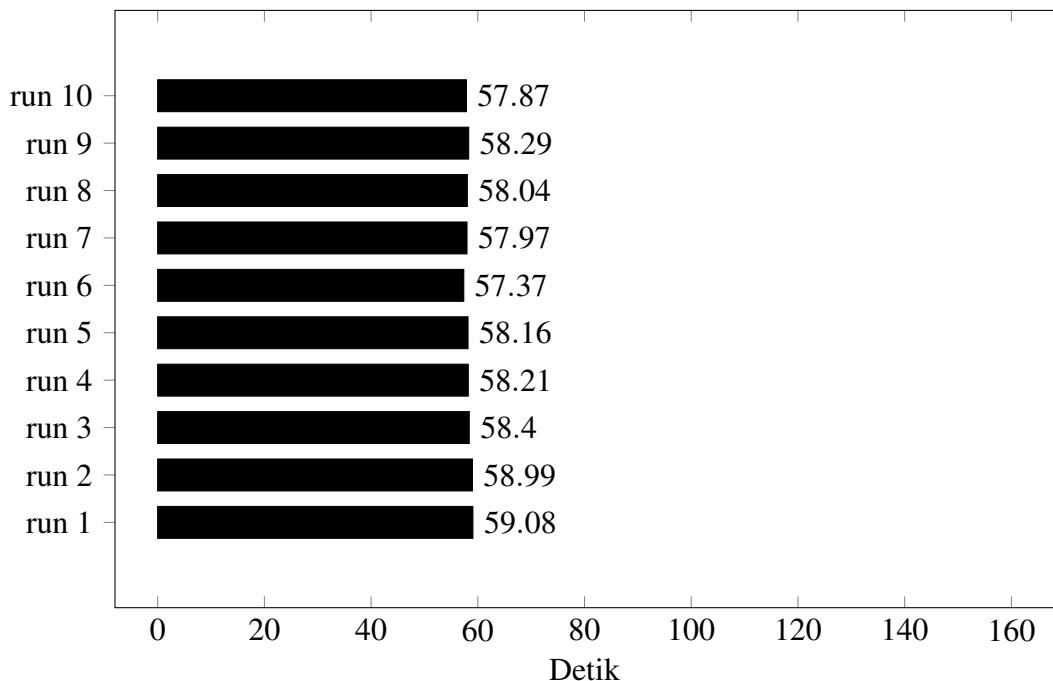
```
fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 ht syscall nx lm rep_good nopl
cpuid extd_apicid tsc_known_freq pni ssse3 cx16 sse4_1 x2apic
hypervisorlahf_lm cmp_legacy svm 3dnowprefetch vmmcall
```

Gambar 4.16: lscpu Konfigurasi SSSE3 + SSE4.1

Gambar 4.16 menunjukkan hasil dari perintah lscpu pada konfigurasi SSSE3 + SSE4.1. Konfigurasi ini memiliki perubahan flag cpu ssse3 dan sse4.1.

```
ubuntu@ubuntu-server:~$ sh ./compress.sh
Running benchmark 1...
Benchmark 1 completed in 59.0792 seconds. Log saved to output/log_1.txt.
Running benchmark 2...
Benchmark 2 completed in 58.9872 seconds. Log saved to output/log_2.txt.
Running benchmark 3...
Benchmark 3 completed in 58.3989 seconds. Log saved to output/log_3.txt.
Running benchmark 4...
Benchmark 4 completed in 58.2131 seconds. Log saved to output/log_4.txt.
Running benchmark 5...
Benchmark 5 completed in 58.1641 seconds. Log saved to output/log_5.txt.
Running benchmark 6...
Benchmark 6 completed in 57.3711 seconds. Log saved to output/log_6.txt.
Running benchmark 7...
Benchmark 7 completed in 57.9736 seconds. Log saved to output/log_7.txt.
Running benchmark 8...
Benchmark 8 completed in 58.0353 seconds. Log saved to output/log_8.txt.
Running benchmark 9...
Benchmark 9 completed in 58.2901 seconds. Log saved to output/log_9.txt.
Running benchmark 10...
Benchmark 10 completed in 57.8719 seconds. Log saved to output/log_10.txt.
ubuntu@ubuntu-server:~$ _
```

Gambar 4.17: Test Kompresi Video dengan Konfigurasi SSSE3 + SSE4.1



Gambar 4.18: Grafik Tes Kompresi Video dengan Konfigurasi SSSE3 + SSE4.1

Program benchmark dengan kode 3.8 dijalankan pada konfigurasi SSSE3 + SSE4.1. Hasil pengujian dapat dilihat pada gambar 4.17. Dari log yang dihasilkan saat menjalankan program benchmark, waktu yang diperlukan untuk melakukan setiap run benchmark diambil dan digunakan untuk membuat grafik seperti yang terlihat pada gambar 4.18. Grafik tersebut menunjukkan bahwa waktu yang diperlukan untuk melakukan benchmark pada konfigurasi SSSE3 + SSE4.1 berkisar antara 57 detik hingga 59 detik. Rata-rata dari 10 run yang dilakukan adalah 58.2 detik.

4.1.7 Konfigurasi dengan SSSE3 + SSE4.2

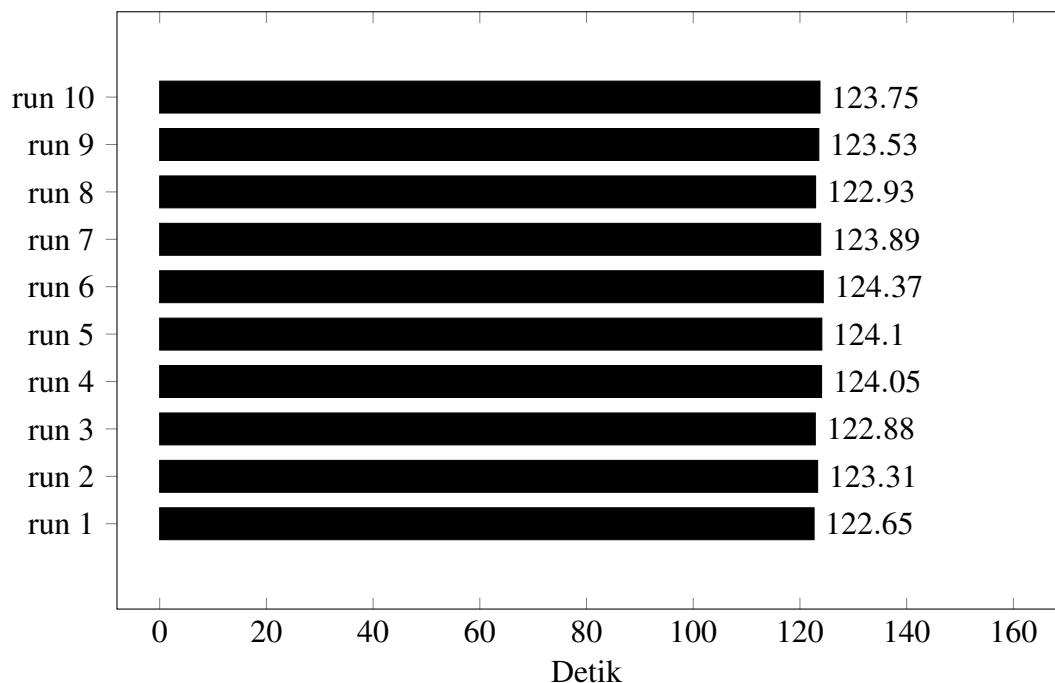
```
fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 ht syscall nx lm rep_good nopl
cpuid extd_apicid tsc_known_freq pni ssse3 cx16 sse4_2 x2apic
hypervisor lahf_lm cmp_legacy svm 3dnowprefetch vmmcall
```

Gambar 4.19: lscpu Konfigurasi SSSE3 + SSE4.2

Gambar 4.19 menunjukkan hasil dari perintah lscpu pada konfigurasi SSSE3 + SSE4.2. Konfigurasi ini memiliki perubahan flag cpu ssse3 dan sse4.2.

```
ubuntu@ubuntu-server:~$ sh ./compress.sh
Running benchmark 1...
Benchmark 1 completed in 122.649 seconds. Log saved to output/log_1.txt.
Running benchmark 2...
Benchmark 2 completed in 123.313 seconds. Log saved to output/log_2.txt.
Running benchmark 3...
Benchmark 3 completed in 122.875 seconds. Log saved to output/log_3.txt.
Running benchmark 4...
Benchmark 4 completed in 124.051 seconds. Log saved to output/log_4.txt.
Running benchmark 5...
Benchmark 5 completed in 124.097 seconds. Log saved to output/log_5.txt.
Running benchmark 6...
Benchmark 6 completed in 124.366 seconds. Log saved to output/log_6.txt.
Running benchmark 7...
Benchmark 7 completed in 123.885 seconds. Log saved to output/log_7.txt.
Running benchmark 8...
Benchmark 8 completed in 122.93 seconds. Log saved to output/log_8.txt.
Running benchmark 9...
Benchmark 9 completed in 123.534 seconds. Log saved to output/log_9.txt.
Running benchmark 10...
Benchmark 10 completed in 123.751 seconds. Log saved to output/log_10.txt.
ubuntu@ubuntu-server:~$
```

Gambar 4.20: Test Kompresi Video dengan Konfigurasi SSSE3 + SSE4.2



Gambar 4.21: Grafik Tes Kompresi Video dengan Konfigurasi SSSE3 + SSE4.2

Program benchmark dengan kode 3.8 dijalankan pada konfigurasi SSSE3 + SSE4.2. Hasil pengujian dapat dilihat pada gambar 4.20. Dari log yang dihasilkan saat menjalankan program benchmark, waktu yang diperlukan untuk melakukan setiap run benchmark diambil dan digunakan untuk membuat grafik seperti yang terlihat pada gambar 4.21. Grafik tersebut menunjukkan bahwa waktu yang diperlukan

untuk melakukan benchmark pada konfigurasi SSSE3 + SSE4.2 berkisar antara 122 detik hingga 124 detik. Rata-rata dari 10 run yang dilakukan adalah 123.5 detik.

4.1.8 Konfigurasi dengan SSSE3 + SSE4a

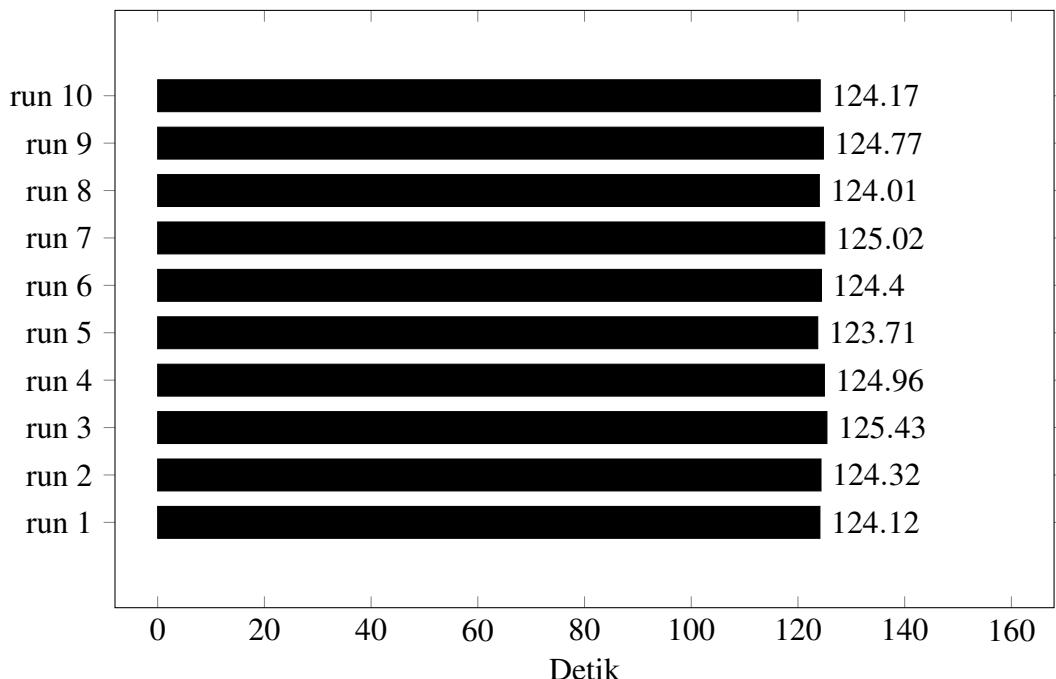
```
fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 ht syscall nx lm rep_good nopl
cpuid extd_apicid tsc_known_freq pni ssse3 cx16 x2apic hypervisor
lahf_lm cmp_legacy svm sse4a 3dnowprefetch vmmcall
```

Gambar 4.22: lscpu Konfigurasi SSSE3 + SSE4a

Gambar 4.22 menunjukkan hasil dari perintah lscpu pada konfigurasi SSSE3 + SSE4a. Konfigurasi ini memiliki perubahan flag cpu ssse3 dan sse4a.

```
ubuntu@ubuntu-server:~$ sh ./compress.sh
Running benchmark 1...
Benchmark 1 completed in 124.115 seconds. Log saved to output/log_1.txt.
Running benchmark 2...
Benchmark 2 completed in 124.316 seconds. Log saved to output/log_2.txt.
Running benchmark 3...
Benchmark 3 completed in 125.429 seconds. Log saved to output/log_3.txt.
Running benchmark 4...
Benchmark 4 completed in 124.956 seconds. Log saved to output/log_4.txt.
Running benchmark 5...
Benchmark 5 completed in 123.711 seconds. Log saved to output/log_5.txt.
Running benchmark 6...
Benchmark 6 completed in 124.402 seconds. Log saved to output/log_6.txt.
Running benchmark 7...
Benchmark 7 completed in 125.019 seconds. Log saved to output/log_7.txt.
Running benchmark 8...
Benchmark 8 completed in 124.005 seconds. Log saved to output/log_8.txt.
Running benchmark 9...
Benchmark 9 completed in 124.767 seconds. Log saved to output/log_9.txt.
Running benchmark 10...
Benchmark 10 completed in 124.171 seconds. Log saved to output/log_10.txt.
ubuntu@ubuntu-server:~$
```

Gambar 4.23: Test Kompresi Video dengan Konfigurasi SSSE3 + SSE4a



Gambar 4.24: Grafik Tes Kompresi Video dengan Konfigurasi SSSE3 + SSE4a

Program benchmark dengan kode 3.8 dijalankan pada konfigurasi SSSE3 + SSE4a. Hasil pengujian dapat dilihat pada gambar 4.23. Dari log yang dihasilkan saat menjalankan program benchmark, waktu yang diperlukan untuk melakukan setiap run benchmark diambil dan digunakan untuk membuat grafik seperti yang terlihat pada gambar 4.24. Grafik tersebut menunjukkan bahwa waktu yang diperlukan untuk melakukan benchmark pada konfigurasi SSSE3 + SSE4a berkisar antara 123 detik hingga 125 detik. Rata-rata dari 10 run yang dilakukan adalah 124.4 detik.

4.1.9 Konfigurasi dengan SSE4.1 + SSE4.2

```
fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 ht syscall nx lm rep_good nopl
cpuid extd_apicid tsc_known_freq pni cx16 sse4_1 sse4_2 x2apic
hypervisor lahf_lm cmp_legacy svm 3dnowprefetch vmmcall
```

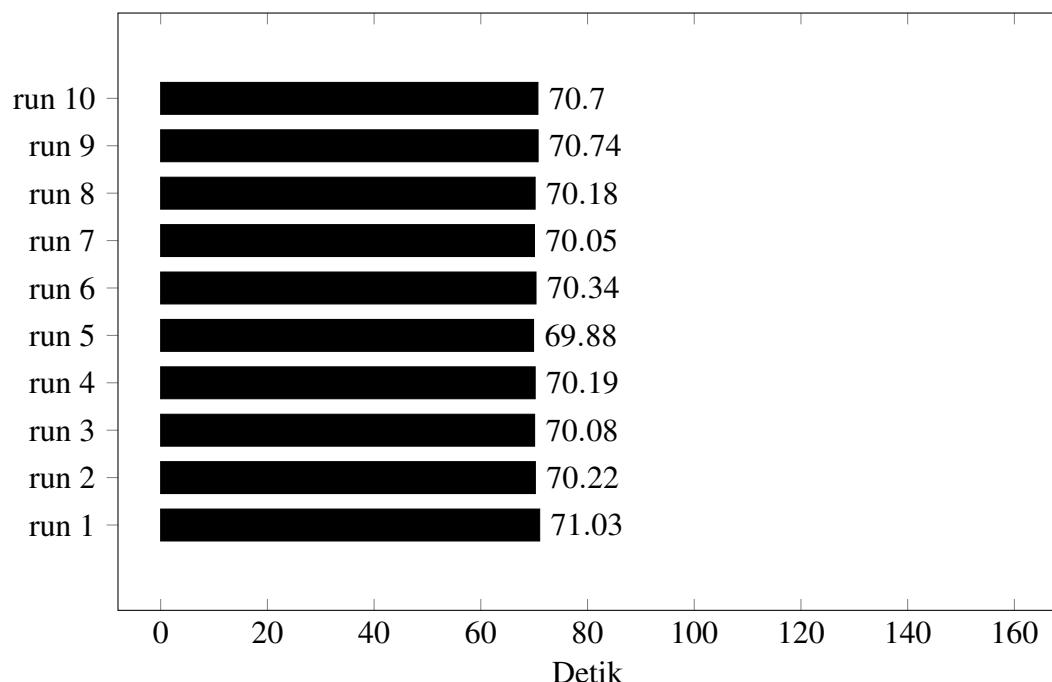
Gambar 4.25: lscpu Konfigurasi SSE4.1 + SSE4.2

Gambar 4.25 menunjukkan hasil dari perintah `lscpu` pada konfigurasi SSE4.1 + SSE4.2. Konfigurasi ini memiliki perubahan flag `cpu sse4.1` dan `sse4.2`.

```

Running benchmark 1...
Benchmark 1 completed in 71.0228 seconds. Log saved to output/log_1.txt.
Running benchmark 2...
Benchmark 2 completed in 70.22 seconds. Log saved to output/log_2.txt.
Running benchmark 3...
Benchmark 3 completed in 70.0831 seconds. Log saved to output/log_3.txt.
Running benchmark 4...
Benchmark 4 completed in 70.186 seconds. Log saved to output/log_4.txt.
Running benchmark 5...
Benchmark 5 completed in 69.8824 seconds. Log saved to output/log_5.txt.
Running benchmark 6...
Benchmark 6 completed in 70.3427 seconds. Log saved to output/log_6.txt.
Running benchmark 7...
Benchmark 7 completed in 70.0528 seconds. Log saved to output/log_7.txt.
Running benchmark 8...
Benchmark 8 completed in 70.1771 seconds. Log saved to output/log_8.txt.
Running benchmark 9...
Benchmark 9 completed in 70.7355 seconds. Log saved to output/log_9.txt.
Running benchmark 10...
Benchmark 10 completed in 70.7029 seconds. Log saved to output/log_10.txt.
ubuntu@ubuntu-server:~$
```

Gambar 4.26: Test Kompresi Video dengan Konfigurasi SSE4.1 + SSE4.2



Gambar 4.27: Grafik Tes Kompresi Video dengan Konfigurasi SSE4.1 + SSE4.2

Program benchmark dengan kode 3.8 dijalankan pada konfigurasi SSE4.1 + SSE4.2. Hasil pengujian dapat dilihat pada gambar 4.26. Dari log yang dihasilkan saat menjalankan program benchmark, waktu yang diperlukan untuk melakukan setiap run benchmark diambil dan digunakan untuk membuat grafik seperti yang terlihat pada gambar 4.27. Grafik tersebut menunjukkan bahwa waktu yang diperlukan untuk melakukan benchmark pada konfigurasi SSE4.1 + SSE4.2 berkisar antara 69

detik hingga 71 detik. Rata-rata dari 10 run yang dilakukan adalah 70.34 detik.

4.1.10 Konfigurasi dengan SSE4.1 + SSE4a

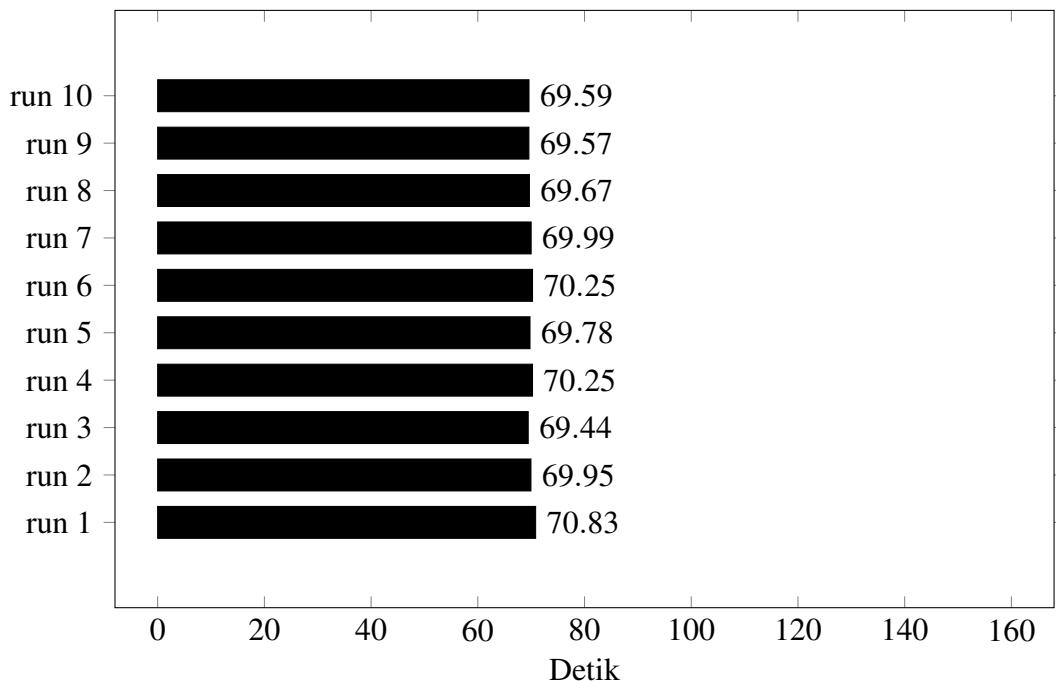
```
fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 ht syscall nx lm rep_good
nopl cpuid extd_apicid tsc_known_freq pni cx16 sse4_1 x2apic
hypervisor lahf_lm cmp_legacy svm sse4a 3dnowprefetch vmmcall
```

Gambar 4.28: lscpu Konfigurasi SSE4.1 + SSE4a

Gambar 4.28 menunjukkan hasil dari perintah lscpu pada konfigurasi SSE4.1 + SSE4a. Konfigurasi ini memiliki perubahan flag cpu sse4.1 dan sse4a.

```
ubuntu@ubuntu-server:~$ sh ./compress.sh
Running benchmark 1...
Benchmark 1 completed in 70.8344 seconds. Log saved to output/log_1.txt.
Running benchmark 2...
Benchmark 2 completed in 69.9521 seconds. Log saved to output/log_2.txt.
Running benchmark 3...
Benchmark 3 completed in 69.4431 seconds. Log saved to output/log_3.txt.
Running benchmark 4...
Benchmark 4 completed in 70.249 seconds. Log saved to output/log_4.txt.
Running benchmark 5...
Benchmark 5 completed in 69.7786 seconds. Log saved to output/log_5.txt.
Running benchmark 6...
Benchmark 6 completed in 70.2531 seconds. Log saved to output/log_6.txt.
Running benchmark 7...
Benchmark 7 completed in 69.9923 seconds. Log saved to output/log_7.txt.
Running benchmark 8...
Benchmark 8 completed in 69.6707 seconds. Log saved to output/log_8.txt.
Running benchmark 9...
Benchmark 9 completed in 69.5722 seconds. Log saved to output/log_9.txt.
Running benchmark 10...
Benchmark 10 completed in 69.5885 seconds. Log saved to output/log_10.txt.
ubuntu@ubuntu-server:~$ _
```

Gambar 4.29: Test Kompresi Video dengan Konfigurasi SSE4.1 + SSE4a



Gambar 4.30: Grafik Tes Kompresi Video dengan Konfigurasi SSE4.1 + SSE4a

Program benchmark dengan kode 3.8 dijalankan pada konfigurasi SSE4.1 + SSE4a. Hasil pengujian dapat dilihat pada gambar 4.29. Dari log yang dihasilkan saat menjalankan program benchmark, waktu yang diperlukan untuk melakukan setiap run benchmark diambil dan digunakan untuk membuat grafik seperti yang terlihat pada gambar 4.30. Grafik tersebut menunjukkan bahwa waktu yang diperlukan untuk melakukan benchmark pada konfigurasi SSE4.1 + SSE4a berkisar antara 69 detik hingga 70 detik. Rata-rata dari 10 run yang dilakukan adalah 69.9 detik.

4.1.11 Konfigurasi dengan SSE4.2 + SSE4a

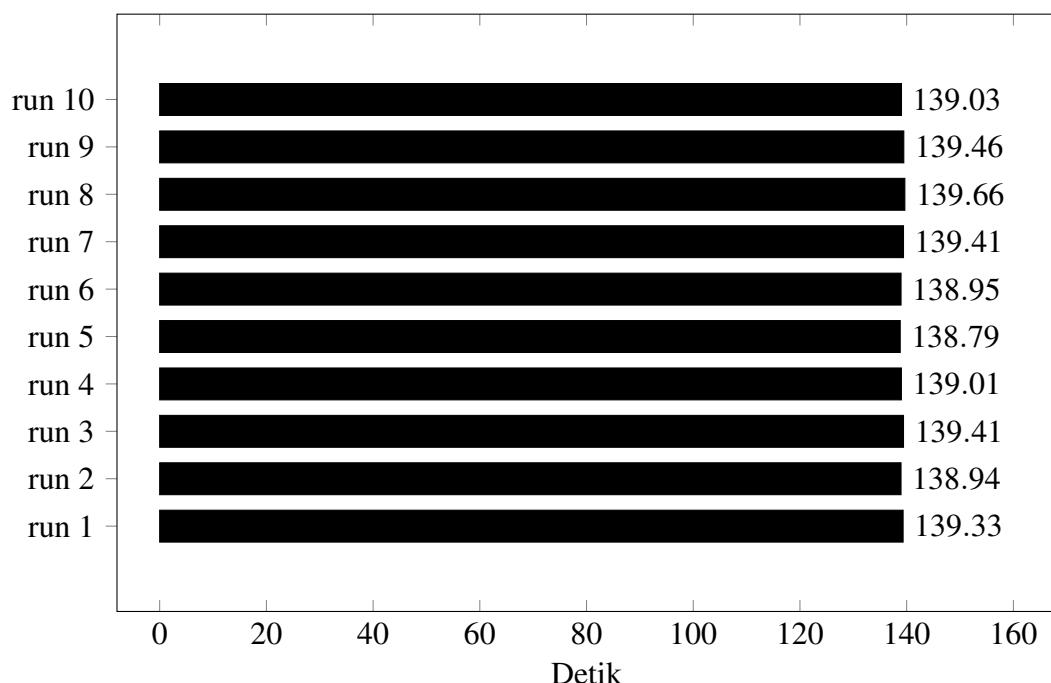
```
fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 ht syscall nx lm rep_good
nopl cpuid extd_apicid tsc_known_freq pni cx16 sse4_2 x2apic
hypervisor lahf_lm cmp_legacy svm sse4a 3dnowprefetch vmmcall
```

Gambar 4.31: lscpu Konfigurasi SSE4.2 + SSE4a

Gambar 4.31 menunjukkan hasil dari perintah `lscpu` pada konfigurasi SSE4.2 + SSE4a. Konfigurasi ini memiliki perubahan flag `cpu sse4_2` dan `sse4a`.

```
ubuntu@ubuntu-server:~$ sh ./compress.sh
Running benchmark 1...
Benchmark 1 completed in 139.333 seconds. Log saved to output/log_1.txt.
Running benchmark 2...
Benchmark 2 completed in 138.942 seconds. Log saved to output/log_2.txt.
Running benchmark 3...
Benchmark 3 completed in 139.409 seconds. Log saved to output/log_3.txt.
Running benchmark 4...
Benchmark 4 completed in 139.012 seconds. Log saved to output/log_4.txt.
Running benchmark 5...
Benchmark 5 completed in 138.791 seconds. Log saved to output/log_5.txt.
Running benchmark 6...
Benchmark 6 completed in 138.947 seconds. Log saved to output/log_6.txt.
Running benchmark 7...
Benchmark 7 completed in 139.408 seconds. Log saved to output/log_7.txt.
Running benchmark 8...
Benchmark 8 completed in 139.66 seconds. Log saved to output/log_8.txt.
Running benchmark 9...
Benchmark 9 completed in 139.464 seconds. Log saved to output/log_9.txt.
Running benchmark 10...
Benchmark 10 completed in 139.028 seconds. Log saved to output/log_10.txt.
ubuntu@ubuntu-server:~$ _
```

Gambar 4.32: Test Kompresi Video dengan Konfigurasi SSE4.2 + SSE4a



Gambar 4.33: Grafik Tes Kompresi Video dengan Konfigurasi SSE4.2 + SSE4a

Program benchmark dengan kode 3.8 dijalankan pada konfigurasi SSE4.2 + SSE4a. Hasil pengujian dapat dilihat pada gambar 4.32. Dari log yang dihasilkan saat menjalankan program benchmark, waktu yang diperlukan untuk melakukan setiap run benchmark diambil dan digunakan untuk membuat grafik seperti yang terlihat pada gambar 4.33. Grafik tersebut menunjukkan bahwa waktu yang diperlukan

untuk melakukan benchmark pada konfigurasi SSE4.2 + SSE4a berkisar antara 138 detik hingga 139 detik. Rata-rata dari 10 run yang dilakukan adalah 139.1 detik.

4.1.12 Konfigurasi dengan SSSE3 + SSE4.1 + SSE4.2

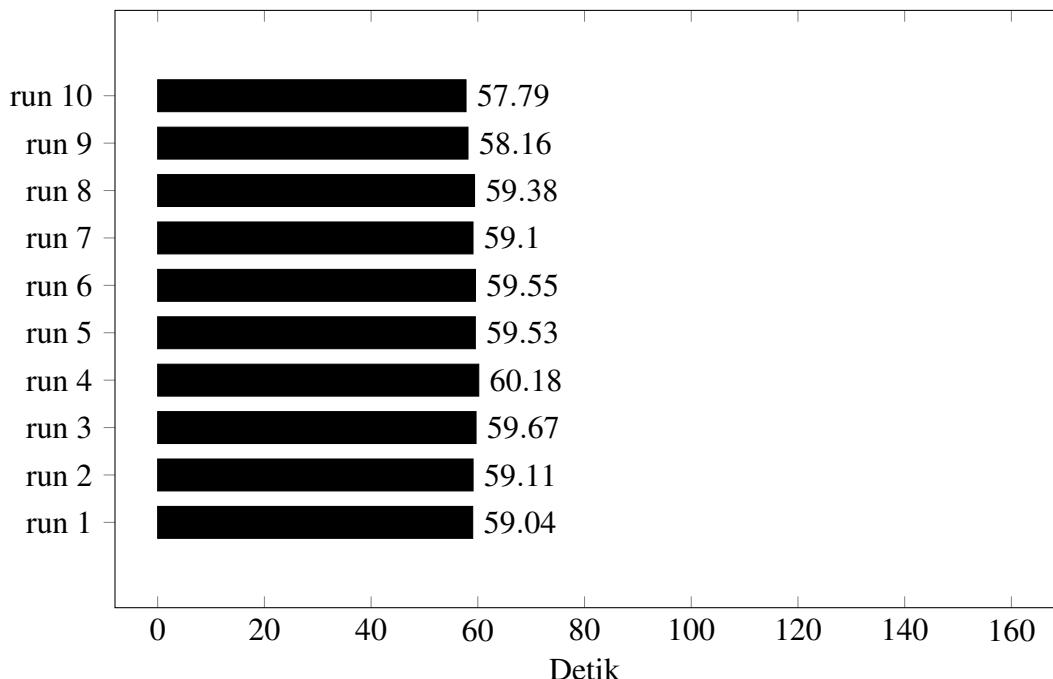
```
fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 ht syscall nx lm rep_good nopl
cpuid extd_apicid tsc_known_freq pni ssse3 cx16 sse4_1 sse4_2
x2apic hypervisor lahf_lm cmp_legacy svm 3dnowprefetch vmmcall
```

Gambar 4.34: lscpu Konfigurasi SSSE3 + SSE4.1 + SSE4.2

Gambar 4.34 menunjukkan hasil dari perintah lscpu pada konfigurasi SSSE3 + SSE4.1 + SSE4.2. Konfigurasi ini memiliki perubahan flag cpu ssse3, sse4.1, dan sse4.2.

```
ubuntu@ubuntu-server:~$ sh ./compress.sh
Running benchmark 1...
Benchmark 1 completed in 59.0406 seconds. Log saved to output/log_1.txt.
Running benchmark 2...
Benchmark 2 completed in 59.1088 seconds. Log saved to output/log_2.txt.
Running benchmark 3...
Benchmark 3 completed in 59.6655 seconds. Log saved to output/log_3.txt.
Running benchmark 4...
Benchmark 4 completed in 60.1753 seconds. Log saved to output/log_4.txt.
Running benchmark 5...
Benchmark 5 completed in 59.6311 seconds. Log saved to output/log_5.txt.
Running benchmark 6...
Benchmark 6 completed in 59.5534 seconds. Log saved to output/log_6.txt.
Running benchmark 7...
Benchmark 7 completed in 59.097 seconds. Log saved to output/log_7.txt.
Running benchmark 8...
Benchmark 8 completed in 59.3849 seconds. Log saved to output/log_8.txt.
Running benchmark 9...
Benchmark 9 completed in 58.1526 seconds. Log saved to output/log_9.txt.
Running benchmark 10...
Benchmark 10 completed in 57.789 seconds. Log saved to output/log_10.txt.
ubuntu@ubuntu-server:~$ _
```

Gambar 4.35: Test Kompresi Video dengan Konfigurasi SSSE3 + SSE4.1 + SSE4.2



Gambar 4.36: Grafik Tes Kompresi Video dengan Konfigurasi SSSE3 + SSE4.1 + SSE4.2

Program benchmark dengan kode 3.8 dijalankan pada konfigurasi SSSE3 + SSE4.1 + SSE4.2. Hasil pengujian dapat dilihat pada gambar 4.35. Dari log yang dihasilkan saat menjalankan program benchmark, waktu yang diperlukan untuk melakukan setiap run benchmark diambil dan digunakan untuk membuat grafik seperti yang terlihat pada gambar 4.36. Grafik tersebut menunjukkan bahwa waktu yang diperlukan untuk melakukan benchmark pada konfigurasi SSSE3 + SSE4.1 + SSE4.2 berkisar antara 57 detik hingga 60 detik. Rata-rata dari 10 run yang dilakukan adalah 59.1 detik.

4.1.13 Konfigurasi dengan SSSE3 + SSE4.1 + SSE4a

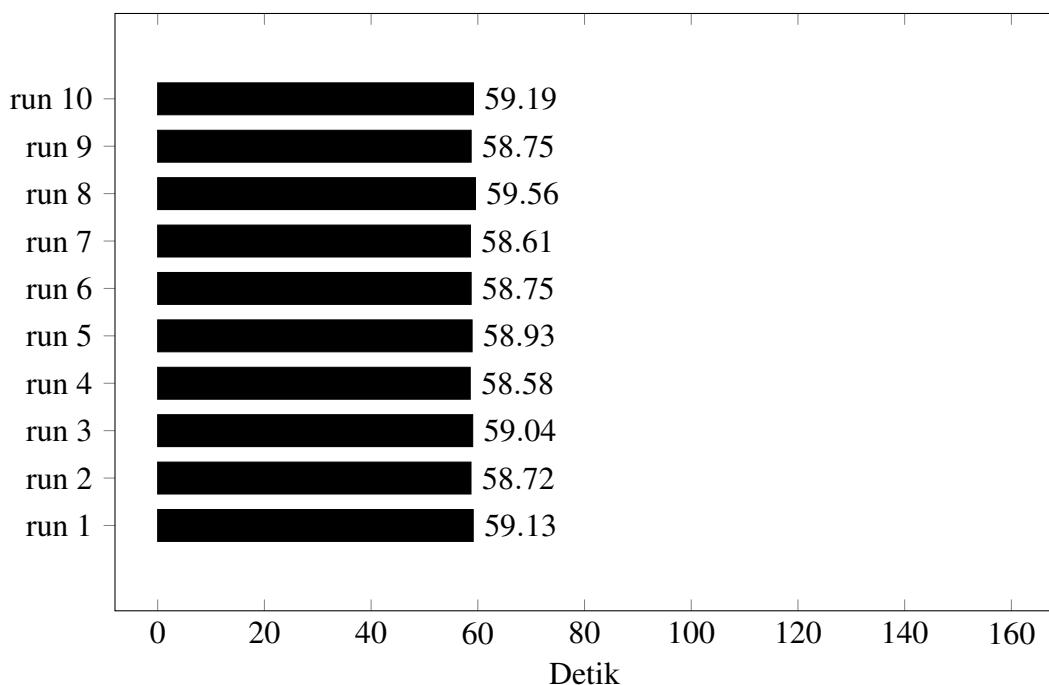
```
fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 ht syscall nx lm rep_good nopl
cpuid extd_apicid tsc_known_freq pni ssse3 cx16 sse4_1 x2apic
hypervisorlahf_lm cmp_legacy svm sse4a 3dnowprefetch vmmcall
```

Gambar 4.37: 1scpu Konfigurasi SSSE3 + SSE4.1 + SSE4a

Gambar 4.37 menunjukkan hasil dari perintah `1scpu` pada konfigurasi SSSE3 + SSE4.1 + SSE4a. Konfigurasi ini memiliki perubahan flag `cpu ssse3, sse4.1, dan sse4a`.

```
ubuntu@ubuntu-server:~$ sh ./compress.sh
Running benchmark 1...
Benchmark 1 completed in 59.1333 seconds. Log saved to output/log_1.txt.
Running benchmark 2...
Benchmark 2 completed in 58.7183 seconds. Log saved to output/log_2.txt.
Running benchmark 3...
Benchmark 3 completed in 59.0377 seconds. Log saved to output/log_3.txt.
Running benchmark 4...
Benchmark 4 completed in 58.5786 seconds. Log saved to output/log_4.txt.
Running benchmark 5...
Benchmark 5 completed in 58.926 seconds. Log saved to output/log_5.txt.
Running benchmark 6...
Benchmark 6 completed in 58.7468 seconds. Log saved to output/log_6.txt.
Running benchmark 7...
Benchmark 7 completed in 58.6098 seconds. Log saved to output/log_7.txt.
Running benchmark 8...
Benchmark 8 completed in 59.5553 seconds. Log saved to output/log_8.txt.
Running benchmark 9...
Benchmark 9 completed in 58.7498 seconds. Log saved to output/log_9.txt.
Running benchmark 10...
Benchmark 10 completed in 59.185 seconds. Log saved to output/log_10.txt.
ubuntu@ubuntu-server:~$ _
```

Gambar 4.38: Test Kompresi Video dengan Konfigurasi SSSE3 + SSE4.1 + SSE4a



Gambar 4.39: Grafik Tes Kompresi Video dengan Konfigurasi SSSE3 + SSE4.1 + SSE4a

Program benchmark dengan kode 3.8 dijalankan pada konfigurasi SSSE3 + SSE4.1 + SSE4a. Hasil pengujian dapat dilihat pada gambar 4.38. Dari log yang dihasilkan saat menjalankan program benchmark, waktu yang diperlukan untuk melakukan setiap run benchmark diambil dan digunakan untuk membuat grafik seperti yang terlihat pada gambar 4.39. Grafik tersebut menunjukkan bahwa waktu

yang diperlukan untuk melakukan benchmark pada konfigurasi SSSE3 + SSE4.1 + SSE4a berkisar antara 58 detik hingga 59 detik. Rata-rata dari 10 run yang dilakukan adalah 58.9 detik.

4.1.14 Konfigurasi dengan SSSE3 + SSE4.2 + SSE4a

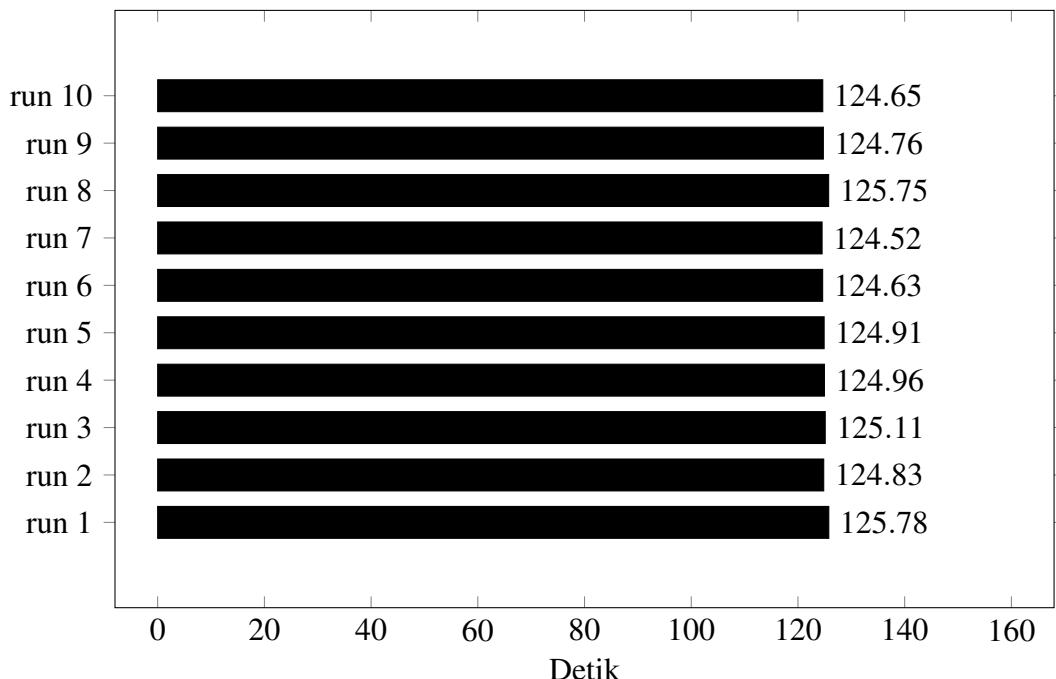
```
fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 ht syscall nx lm rep_good nopl
cpuid extd_apicid tsc_known_freq pni ssse3 cx16 sse4_2 x2apic
hypervisor lahf_lm cmp_legacy svm sse4a 3dnowprefetch vmmcall
```

Gambar 4.40: lscpu Konfigurasi SSSE3 + SSE4.2 + SSE4a

Gambar 4.40 menunjukkan hasil dari perintah lscpu pada konfigurasi SSSE3 + SSE4.2 + SSE4a. Konfigurasi ini memiliki perubahan flag cpu ssse3, sse4.2, dan sse4a.

```
ubuntu@ubuntu-server:~$ sh ./compress.sh
Running benchmark 1...
Benchmark 1 completed in 125.78 seconds. Log saved to output/log_1.txt.
Running benchmark 2...
Benchmark 2 completed in 124.83 seconds. Log saved to output/log_2.txt.
Running benchmark 3...
Benchmark 3 completed in 125.111 seconds. Log saved to output/log_3.txt.
Running benchmark 4...
Benchmark 4 completed in 124.961 seconds. Log saved to output/log_4.txt.
Running benchmark 5...
Benchmark 5 completed in 124.907 seconds. Log saved to output/log_5.txt.
Running benchmark 6...
Benchmark 6 completed in 124.631 seconds. Log saved to output/log_6.txt.
Running benchmark 7...
Benchmark 7 completed in 124.521 seconds. Log saved to output/log_7.txt.
Running benchmark 8...
Benchmark 8 completed in 125.754 seconds. Log saved to output/log_8.txt.
Running benchmark 9...
Benchmark 9 completed in 124.759 seconds. Log saved to output/log_9.txt.
Running benchmark 10...
Benchmark 10 completed in 124.65 seconds. Log saved to output/log_10.txt.
ubuntu@ubuntu-server:~$ _
```

Gambar 4.41: Test Kompresi Video dengan Konfigurasi SSSE3 + SSE4.2 + SSE4a



Gambar 4.42: Grafik Tes Kompresi Video dengan Konfigurasi SSSE3 + SSE4.2 + SSE4a

Program benchmark dengan kode 3.8 dijalankan pada konfigurasi SSSE3 + SSE4.2 + SSE4a. Hasil pengujian dapat dilihat pada gambar 4.41. Dari log yang dihasilkan saat menjalankan program benchmark, waktu yang diperlukan untuk melakukan setiap run benchmark diambil dan digunakan untuk membuat grafik seperti yang terlihat pada gambar 4.42. Grafik tersebut menunjukkan bahwa waktu yang diperlukan untuk melakukan benchmark pada konfigurasi SSSE3 + SSE4.2 + SSE4a berkisar antara 124 detik hingga 125 detik. Rata-rata dari 10 run yang dilakukan adalah 124.9 detik.

4.1.15 Konfigurasi dengan SSE4.1 + SSE4.2 + SSE4a

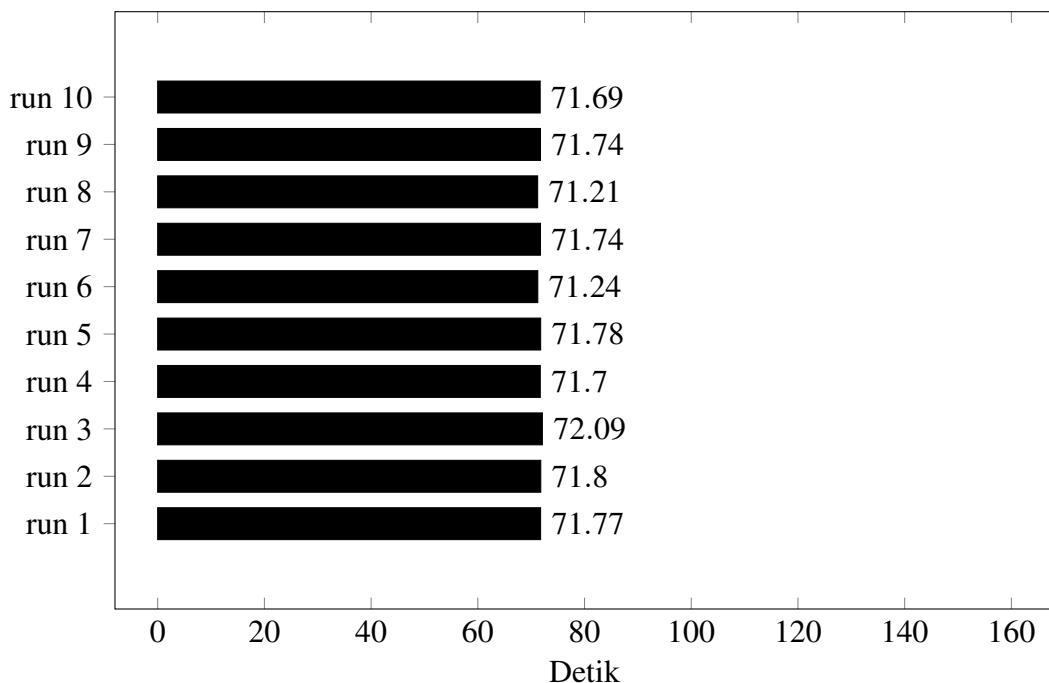
```
fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 ht syscall nx lm rep_good nopl
cpuid extd_apicid tsc_known_freq pni cx16 sse4_1 sse4_2 x2apic
hypervisorlahf_lm cmp_legacy svm sse4a 3dnowprefetch vmmcall
```

Gambar 4.43: `lscpu` Konfigurasi SSE4.1 + SSE4.2 + SSE4a

Gambar 4.43 menunjukkan hasil dari perintah `lscpu` pada konfigurasi SSE4.1 + SSE4.2 + SSE4a. Konfigurasi ini memiliki perubahan flag `cpu sse4.1, sse4.2, dan sse4a`.

```
ubuntu@ubuntu-server:~$ sh ./compress.sh
Running benchmark 1...
Benchmark 1 completed in 71.7658 seconds. Log saved to output/log_1.txt.
Running benchmark 2...
Benchmark 2 completed in 71.7952 seconds. Log saved to output/log_2.txt.
Running benchmark 3...
Benchmark 3 completed in 72.0926 seconds. Log saved to output/log_3.txt.
Running benchmark 4...
Benchmark 4 completed in 71.7037 seconds. Log saved to output/log_4.txt.
Running benchmark 5...
Benchmark 5 completed in 71.7758 seconds. Log saved to output/log_5.txt.
Running benchmark 6...
Benchmark 6 completed in 71.24 seconds. Log saved to output/log_6.txt.
Running benchmark 7...
Benchmark 7 completed in 71.7426 seconds. Log saved to output/log_7.txt.
Running benchmark 8...
Benchmark 8 completed in 71.2123 seconds. Log saved to output/log_8.txt.
Running benchmark 9...
Benchmark 9 completed in 71.7412 seconds. Log saved to output/log_9.txt.
Running benchmark 10...
Benchmark 10 completed in 71.6859 seconds. Log saved to output/log_10.txt.
ubuntu@ubuntu-server:~$ _
```

Gambar 4.44: Test Kompresi Video dengan Konfigurasi SSE4.1 + SSE4.2 + SSE4a



Gambar 4.45: Grafik Tes Kompresi Video dengan Konfigurasi SSE4.1 + SSE4.2 + SSE4a

Program benchmark dengan kode 3.8 dijalankan pada konfigurasi SSE4.1 + SSE4.2 + SSE4a. Hasil pengujian dapat dilihat pada gambar 4.44. Dari log yang dihasilkan saat menjalankan program benchmark, waktu yang diperlukan untuk melakukan setiap run benchmark diambil dan digunakan untuk membuat grafik seperti yang terlihat pada gambar 4.45. Grafik tersebut menunjukkan bahwa waktu

yang diperlukan untuk melakukan benchmark pada konfigurasi SSE4.1 + SSE4.2 + SSE4a berkisar antara 71 detik hingga 72 detik. Rata-rata dari 10 run yang dilakukan adalah 71.6 detik.

4.1.16 Konfigurasi dengan SSSE3 + SSE4.1 + SSE4.2 + SSE4a

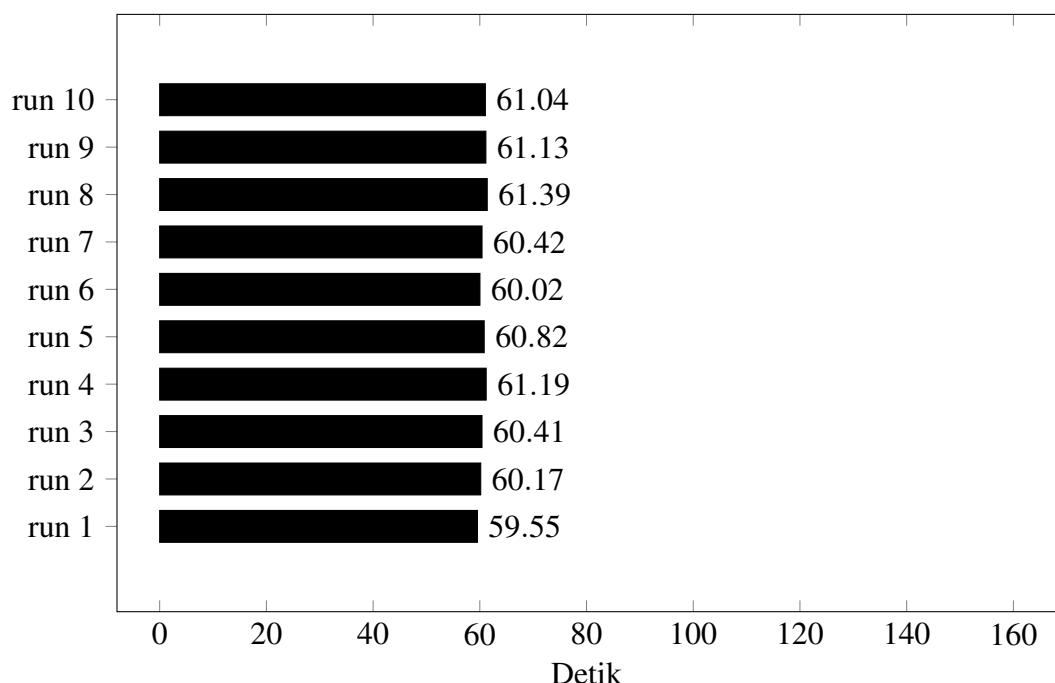
```
fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 ht syscall nx lm rep_good nopl
cpuid extd_apicid tsc_known_freq pni ssse3 cx16 sse4_1 sse4_2
x2apic hypervisor lahf_lm cmp_legacy svm sse4a 3dnowprefetch
vmmcall
```

Gambar 4.46: lscpu Konfigurasi SSSE3 + SSE4.1 + SSE4.2 + SSE4a

Gambar 4.37 menunjukkan hasil dari perintah lscpu pada konfigurasi SSSE3 + SSE4.1 + SSE4.2 + SSE4a. Konfigurasi ini memiliki perubahan flag cpu ssse3, sse4.1, sse4.2, dan sse4a.

```
ubuntu@ubuntu-server:~$ sh ./compress.sh
Running benchmark 1...
Benchmark 1 completed in 59.5519 seconds. Log saved to output/log_1.txt.
Running benchmark 2...
Benchmark 2 completed in 60.1732 seconds. Log saved to output/log_2.txt.
Running benchmark 3...
Benchmark 3 completed in 60.4103 seconds. Log saved to output/log_3.txt.
Running benchmark 4...
Benchmark 4 completed in 61.193 seconds. Log saved to output/log_4.txt.
Running benchmark 5...
Benchmark 5 completed in 60.82 seconds. Log saved to output/log_5.txt.
Running benchmark 6...
Benchmark 6 completed in 60.0236 seconds. Log saved to output/log_6.txt.
Running benchmark 7...
Benchmark 7 completed in 60.4153 seconds. Log saved to output/log_7.txt.
Running benchmark 8...
Benchmark 8 completed in 61.3896 seconds. Log saved to output/log_8.txt.
Running benchmark 9...
Benchmark 9 completed in 61.1348 seconds. Log saved to output/log_9.txt.
Running benchmark 10...
Benchmark 10 completed in 61.0414 seconds. Log saved to output/log_10.txt.
ubuntu@ubuntu-server:~$
```

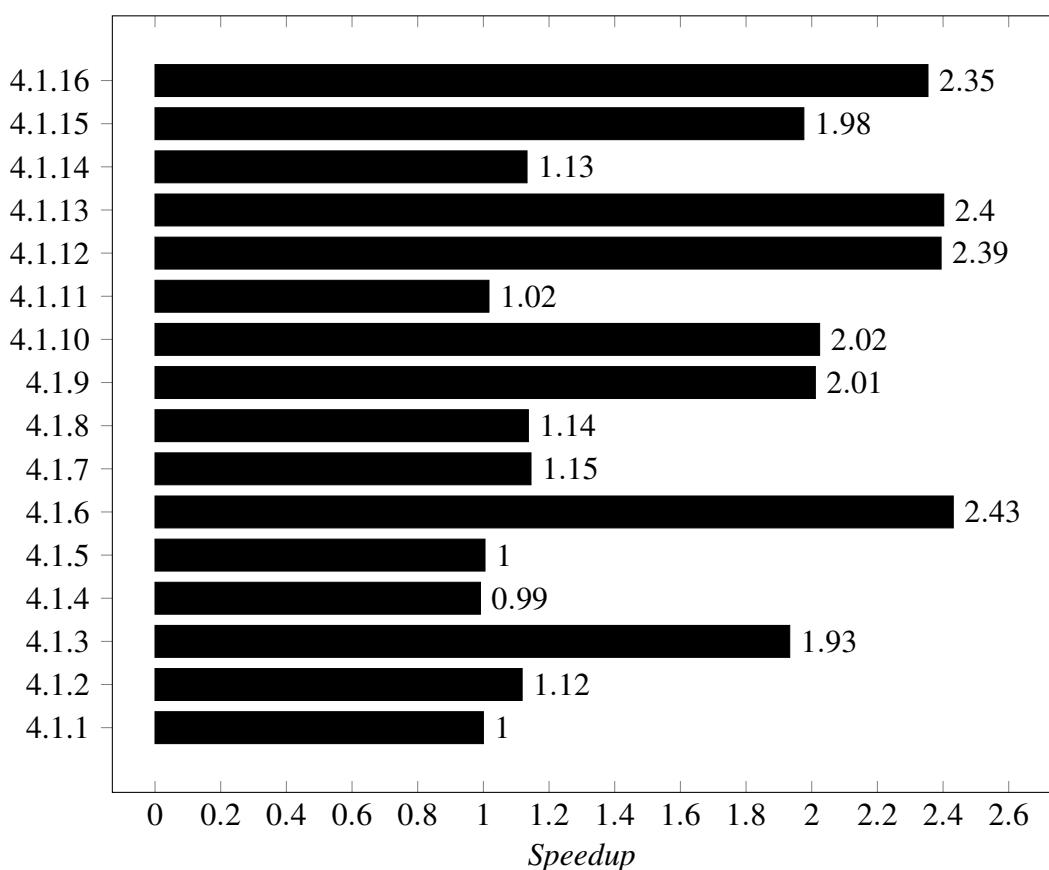
Gambar 4.47: Test Kompresi Video dengan Konfigurasi SSSE3 + SSE4.1 + SSE4.2 + SSE4a



Gambar 4.48: Grafik Tes Kompresi Video dengan Konfigurasi SSSE3 + SSE4.1 + SSE4.2 + SSE4a

Program benchmark dengan kode 3.8 dijalankan pada konfigurasi SSSE3 + SSE4.1 + SSE4.2 + SSE4a. Hasil pengujian dapat dilihat pada gambar 4.47. Dari log yang dihasilkan saat menjalankan program benchmark, waktu yang diperlukan untuk melakukan setiap run benchmark diambil dan digunakan untuk membuat grafik seperti yang terlihat pada gambar 4.48. Grafik tersebut menunjukkan bahwa waktu yang diperlukan untuk melakukan benchmark pada konfigurasi SSSE3 + SSE4.1 + SSE4.2 + SSE4a berkisar antara 59 detik hingga 61 detik. Rata-rata dari 10 run yang dilakukan adalah 60.1 detik.

4.1.17 Analisis Pengujian Tuning Hypervisor KVM dengan Kompresi Video



Gambar 4.49: Grafik speedup Tes Kompresi Video

Grafik 4.49 menunjukkan *speedup* dari setiap konfigurasi yang diuji. Dari grafik tersebut, dapat dilihat bahwa konfigurasi yang memiliki *speedup* tertinggi adalah konfigurasi SSSE3 + SSE4.1 dengan nilai *speedup* 2.431, dengan rata-rata waktu untuk melakukan kompresi selama 58.2 detik.

4.2 Hasil Pengujian Tuning Hypervisor KVM dengan Validasi Integritas Data

Pada bagian ini akan dijelaskan hasil pengujian tuning hypervisor KVM dengan validasi integritas data dengan menggunakan program benchmark kode 3.9.

4.2.1 Konfigurasi Default

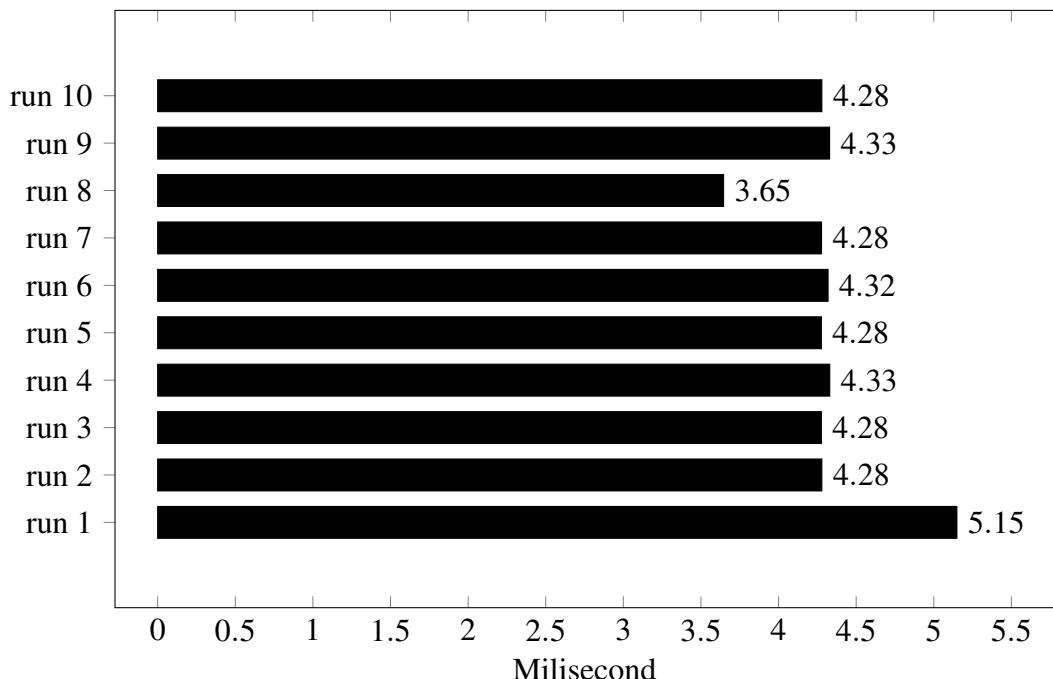
```
fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 ht syscall nx lm rep_good nopl
cpuid extd_apicid tsc_known_freq pni cx16 x2apic hypervisor
lahf_lm cmp_legacy svm 3dnowprefetch vmmcall
```

Gambar 4.50: lscpu Konfigurasi Default

Gambar 4.50 menunjukkan hasil dari perintah `lscpu` pada konfigurasi default. Konfigurasi ini tidak memiliki perubahan flag cpu.

```
ubuntu@ubuntu-server:~/crc32$ ls
crc32.py original.txt sse4.1.txt sse4.2.txt ssse3.txt
ubuntu@ubuntu-server:~/crc32$ cat original.txt
Run 1 Average time for CRC32 calculation: 5.149031162261963 ms
Run 2 Average time for CRC32 calculation: 4.279698133468628 ms
Run 3 Average time for CRC32 calculation: 4.276137113571167 ms
Run 4 Average time for CRC32 calculation: 4.330532550811768 ms
Run 5 Average time for CRC32 calculation: 4.276708364486694 ms
Run 6 Average time for CRC32 calculation: 4.319846153259277 ms
Run 7 Average time for CRC32 calculation: 4.277328014373779 ms
Run 8 Average time for CRC32 calculation: 3.646852970123291 ms
Run 9 Average time for CRC32 calculation: 4.329800605773926 ms
Run 10 Average time for CRC32 calculation: 4.279640197753906 ms
ubuntu@ubuntu-server:~/crc32$
```

Gambar 4.51: Test Validasi Integritas Data Dengan Konfigurasi Default



Gambar 4.52: Grafik Tes Validasi Integritas Data Dengan Konfigurasi Default

Program benchmark dengan kode 3.9 dijalankan pada konfigurasi default. Hasil pengujian dapat dilihat pada gambar 4.51. Dari log yang dihasilkan saat menjalankan program benchmark, waktu yang diperlukan untuk melakukan setiap run benchmark diambil dan digunakan untuk membuat grafik seperti yang terlihat pada gambar 4.52. Grafik tersebut menunjukkan bahwa waktu yang diperlukan untuk melakukan benchmark pada konfigurasi default berkisar antara 3.6 ms hingga 5.1 ms. Rata-rata dari 10 run yang dilakukan adalah 4.3 ms.

4.2.2 Konfigurasi dengan SSE4.2

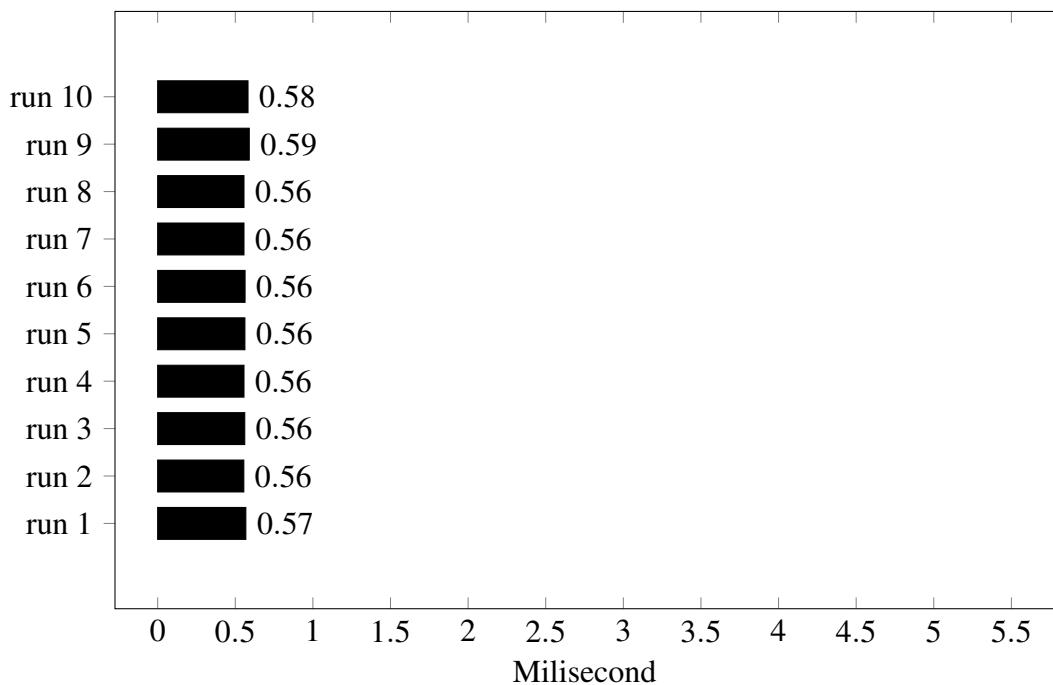
```
fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 ht syscall nx lm rep_good
nopl cpuid extd_apicid tsc_known_freq pni cx16 sse4_2 x2apic
hypervisorlahf_lm cmp_legacy svm 3dnowprefetch vmmcall
```

Gambar 4.53: lscpu Konfigurasi SSE4.2

Gambar 4.53 menunjukkan hasil dari perintah `lscpu` pada konfigurasi SSE4.2. Konfigurasi ini memiliki perubahan flag `cpu sse4_2`.

```
ubuntu@ubuntu-server:~/crc32$ cat sse4.2.txt
Run 1 Average time for CRC32 calculation: 0.5689413547515869 ms
Run 2 Average time for CRC32 calculation: 0.5555415153503418 ms
Run 3 Average time for CRC32 calculation: 0.5618934631347656 ms
Run 4 Average time for CRC32 calculation: 0.5566246509552002 ms
Run 5 Average time for CRC32 calculation: 0.5627317428588867 ms
Run 6 Average time for CRC32 calculation: 0.5632684230804443 ms
Run 7 Average time for CRC32 calculation: 0.556835412979126 ms
Run 8 Average time for CRC32 calculation: 0.5568451881408691 ms
Run 9 Average time for CRC32 calculation: 0.5905630588531494 ms
Run 10 Average time for CRC32 calculation: 0.5825207233428955 ms
ubuntu@ubuntu-server:~/crc32$
```

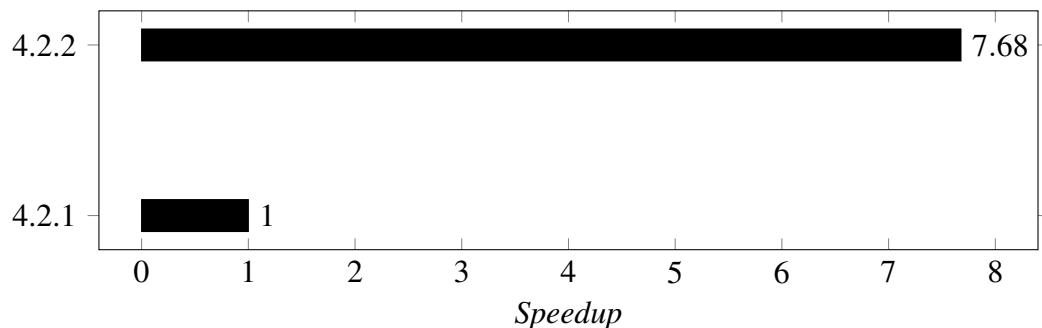
Gambar 4.54: Test Validasi Integritas Data Dengan Konfigurasi SSE4.2



Gambar 4.55: Grafik Tes Validasi Integritas Data Dengan Konfigurasi SSE4.2

Program benchmark dengan kode 3.9 dijalankan pada konfigurasi SSE4.2. Hasil pengujian dapat dilihat pada gambar 4.54. Dari log yang dihasilkan saat menjalankan program benchmark, waktu yang diperlukan untuk melakukan setiap run benchmark diambil dan digunakan untuk membuat grafik seperti yang terlihat pada gambar 4.55. Grafik tersebut menunjukkan bahwa waktu yang diperlukan untuk melakukan benchmark pada konfigurasi SSE4.2 berkisar antara 0.55 ms hingga 0.59 ms. Rata-rata dari 10 run yang dilakukan adalah 0.56 ms.

4.2.3 Analisis Pengujian Tuning Hypervisor KVM dengan Validasi Integritas Data



Gambar 4.56: Grafik speedup Tes Validasi Integritas Data

Grafik 4.56 menunjukkan *speedup* dari setiap konfigurasi yang diuji. Dari grafik tersebut, dapat dilihat bahwa konfigurasi yang memiliki *speedup* tertinggi adalah konfigurasi SSE4.2 dengan nilai *speedup* 7.678, dengan rata-rata waktu untuk melakukan validasi integritas data selama 0.56 ms.

4.3 Hasil Pengujian Tuning Hypervisor KVM dengan Enkripsi AES

Pada bagian ini akan dijelaskan hasil pengujian tuning hypervisor KVM dengan enkripsi AES dengan menggunakan program benchmark kode 3.11.

4.3.1 Konfigurasi Default

```
fpu de pse tsc msr pae mce cx8 apic sep mttrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 ht syscall nx lm rep_good nopl
cpuid extd_apicid tsc_known_freq pnpi cx16 x2apic hypervisor
lahf_lm cmp_legacy svm 3dnowprefetch vmmcall
```

Gambar 4.57: lscpu Konfigurasi Default

Gambar 4.57 menunjukkan hasil dari perintah `lscpu` pada konfigurasi default. Konfigurasi ini tidak memiliki perubahan flag cpu.

```
ubuntu@ubuntu-server:~/aes$ python aes.py > log_original.txt
ubuntu@ubuntu-server:~/aes$ cat log_original.txt
Run 1 Average AES Encryption Time: 0.01718 ms
Run 1 Average AES Decryption Time: 0.02083 ms

Run 2 Average AES Encryption Time: 0.01713 ms
Run 2 Average AES Decryption Time: 0.02087 ms

Run 3 Average AES Encryption Time: 0.01713 ms
Run 3 Average AES Decryption Time: 0.02079 ms

Run 4 Average AES Encryption Time: 0.01732 ms
Run 4 Average AES Decryption Time: 0.02097 ms

Run 5 Average AES Encryption Time: 0.01723 ms
Run 5 Average AES Decryption Time: 0.02098 ms

Run 6 Average AES Encryption Time: 0.01723 ms
Run 6 Average AES Decryption Time: 0.02099 ms

Run 7 Average AES Encryption Time: 0.01727 ms
Run 7 Average AES Decryption Time: 0.02095 ms

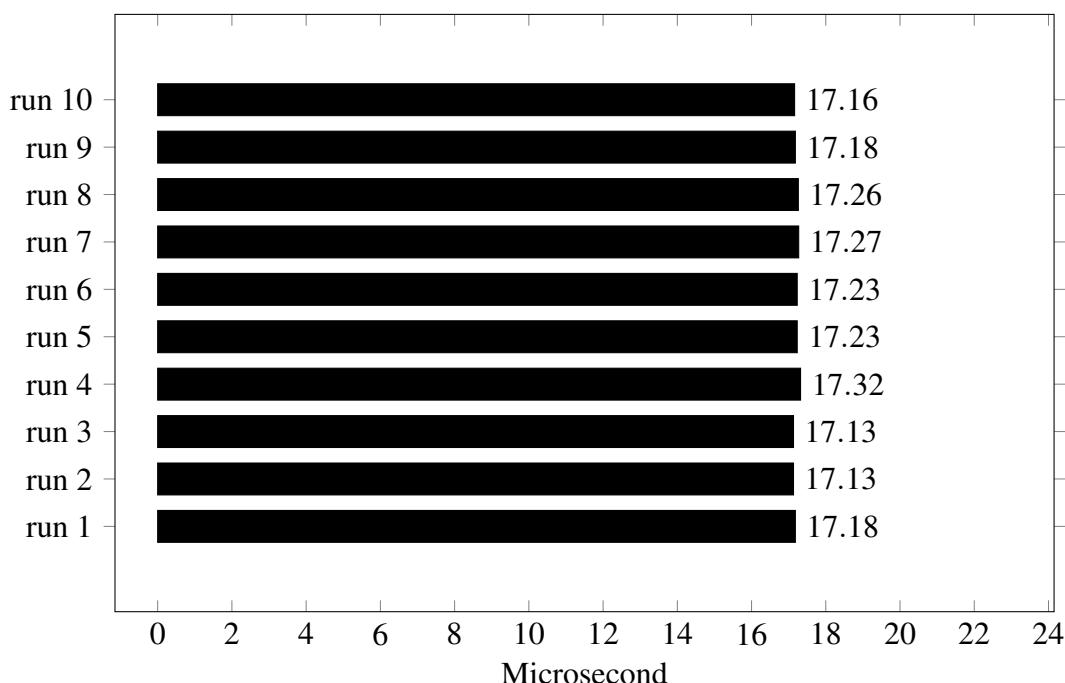
Run 8 Average AES Encryption Time: 0.01726 ms
Run 8 Average AES Decryption Time: 0.02095 ms

Run 9 Average AES Encryption Time: 0.01718 ms
Run 9 Average AES Decryption Time: 0.02090 ms

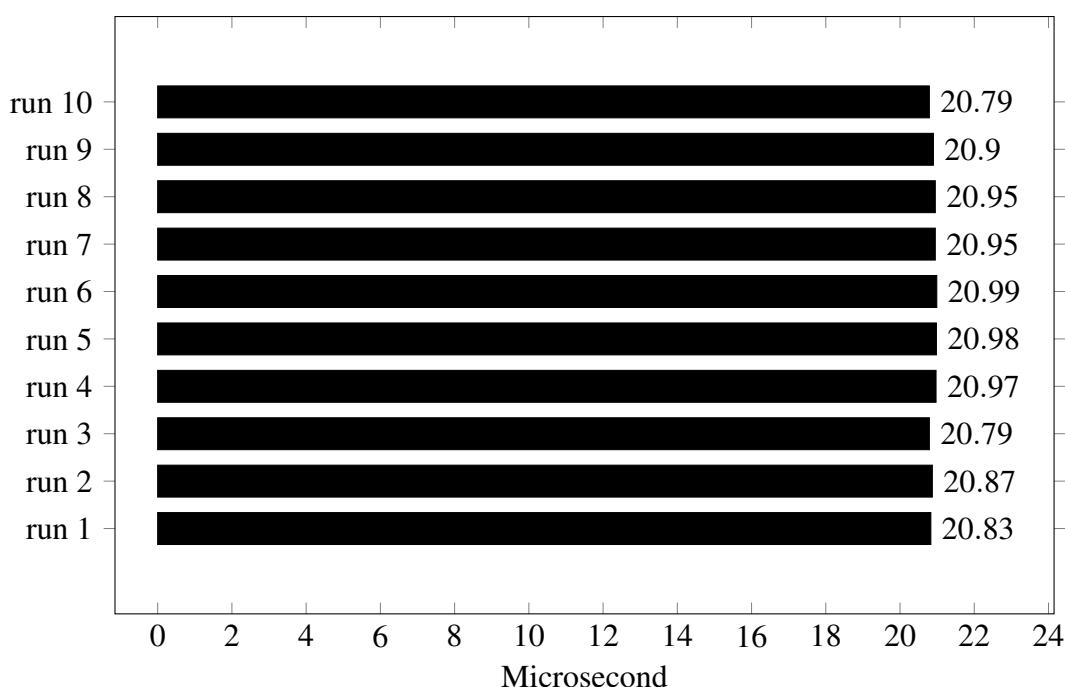
Run 10 Average AES Encryption Time: 0.01716 ms
Run 10 Average AES Decryption Time: 0.02079 ms

ubuntu@ubuntu-server:~/aes$
```

Gambar 4.58: Test Enkripsi AES Dengan Konfigurasi Default



Gambar 4.59: Grafik Tes Enkripsi AES Dengan Konfigurasi Default



Gambar 4.60: Grafik Tes Dekripsi AES Dengan Konfigurasi Default

Program benchmark dengan kode 3.11 dijalankan pada konfigurasi default. Hasil pengujian dapat dilihat pada gambar 4.58. Dari log yang dihasilkan saat menjalankan program benchmark, waktu yang diperlukan untuk melakukan setiap run benchmark diambil dan digunakan untuk membuat grafik seperti yang terlihat pada

gambar 4.59 dan 4.60. Grafik tersebut menunjukkan bahwa waktu yang diperlukan untuk melakukan benchmark pada konfigurasi default berkisar untuk melakukan enkripsi berkisar antara $17 \mu\text{s}$ hingga $17.3 \mu\text{s}$ dan untuk melakukan dekripsi berkisar antara $20.7 \mu\text{s}$ hingga $21 \mu\text{s}$. Rata-rata dari 10 run yang dilakukan adalah $17.2 \mu\text{s}$ untuk enkripsi dan $20.9 \mu\text{s}$ untuk dekripsi.

4.3.2 Konfigurasi dengan AES

```
fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat  
pse36 clflush mmx fxsr sse sse2 ht syscall nx lm rep_good nopl  
cpuid extd_apicid tsc_known_freq pni cx16 x2apic aes hypervisor  
lahf_lm cmp_legacy svm 3dnowprefetch vmmcall
```

Gambar 4.61: lscpu Konfigurasi AES

Gambar 4.61 menunjukkan hasil dari perintah `lscpu` pada konfigurasi AES. Konfigurasi ini memiliki perubahan flag `cpu aes`.

```
ubuntu@ubuntu-server:~/aes$ python aes.py > log_aes.txt
ubuntu@ubuntu-server:~/aes$ cat log_aes.txt
Run 1 Average AES Encryption Time: 0.00815 ms
Run 1 Average AES Decryption Time: 0.00805 ms

Run 2 Average AES Encryption Time: 0.00799 ms
Run 2 Average AES Decryption Time: 0.00797 ms

Run 3 Average AES Encryption Time: 0.00797 ms
Run 3 Average AES Decryption Time: 0.00789 ms

Run 4 Average AES Encryption Time: 0.00801 ms
Run 4 Average AES Decryption Time: 0.00787 ms

Run 5 Average AES Encryption Time: 0.00804 ms
Run 5 Average AES Decryption Time: 0.00783 ms

Run 6 Average AES Encryption Time: 0.00797 ms
Run 6 Average AES Decryption Time: 0.00778 ms

Run 7 Average AES Encryption Time: 0.00795 ms
Run 7 Average AES Decryption Time: 0.00784 ms

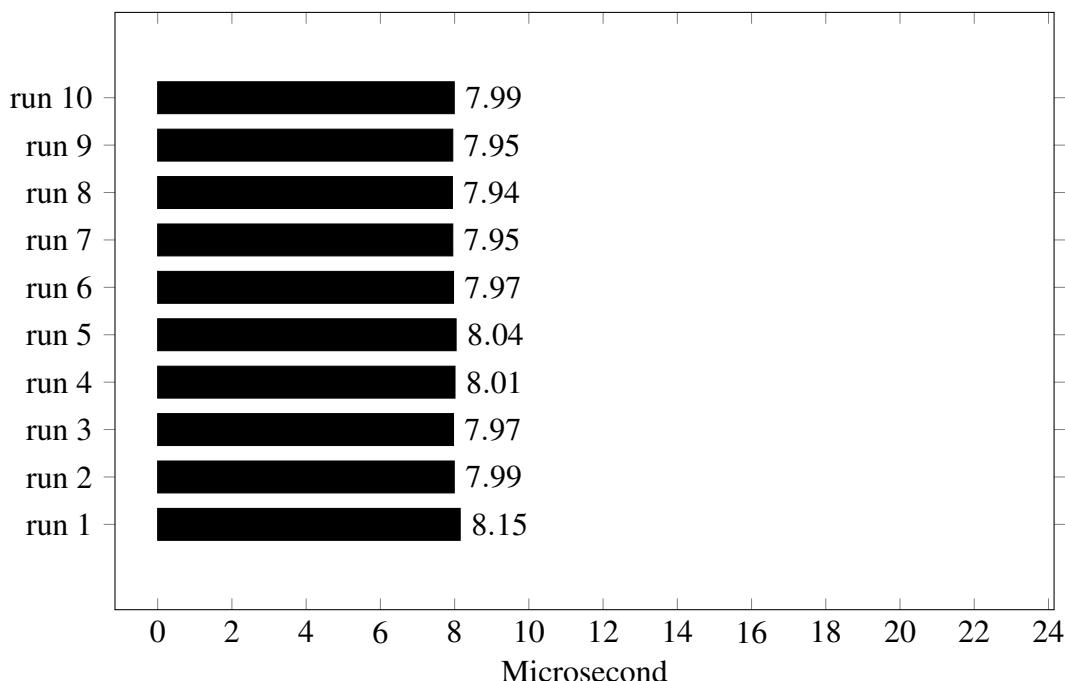
Run 8 Average AES Encryption Time: 0.00794 ms
Run 8 Average AES Decryption Time: 0.00776 ms

Run 9 Average AES Encryption Time: 0.00795 ms
Run 9 Average AES Decryption Time: 0.00780 ms

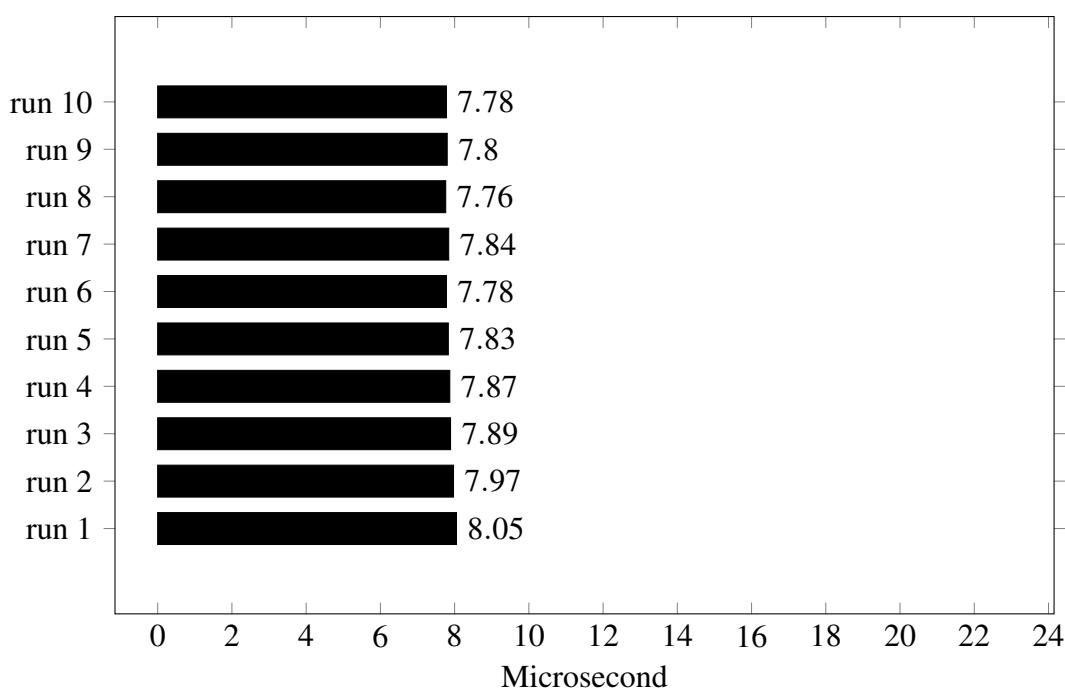
Run 10 Average AES Encryption Time: 0.00799 ms
Run 10 Average AES Decryption Time: 0.00778 ms

ubuntu@ubuntu-server:~/aes$ _
```

Gambar 4.62: Test Enkripsi AES Dengan Konfigurasi AES



Gambar 4.63: Grafik Tes Enkripsi AES Dengan Konfigurasi AES

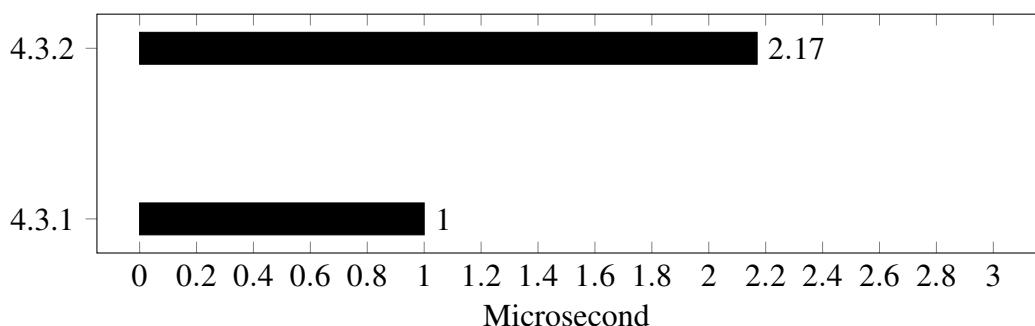


Gambar 4.64: Grafik Tes Dekripsi AES Dengan Konfigurasi AES

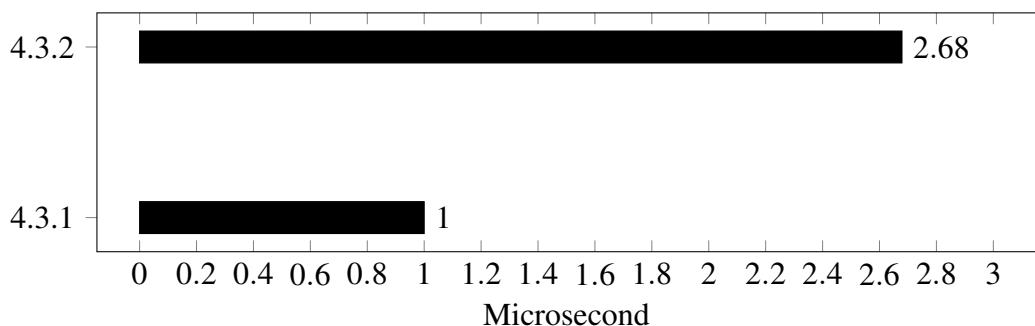
Program benchmark dengan kode 3.11 dijalankan pada konfigurasi AES. Hasil pengujian dapat dilihat pada gambar 4.62. Dari log yang dihasilkan saat menjalankan program benchmark, waktu yang diperlukan untuk melakukan setiap run benchmark diambil dan digunakan untuk membuat grafik seperti yang terlihat pada

gambar 4.63 dan 4.64. Grafik tersebut menunjukkan bahwa waktu yang diperlukan untuk melakukan benchmark pada konfigurasi AES berkisar untuk melakukan enkripsi berkisar antara $7.8 \mu\text{s}$ hingga $8.15 \mu\text{s}$ dan untuk melakukan dekripsi berkisar antara $7.76 \mu\text{s}$ hingga $8.05 \mu\text{s}$. Rata-rata dari 10 run yang dilakukan adalah $7.9 \mu\text{s}$ untuk enkripsi dan $7.8 \mu\text{s}$ untuk dekripsi.

4.3.3 Analisis Pengujian Tuning Hypervisor KVM dengan Enkripsi AES



Gambar 4.65: Grafik Rata-Rata Waktu Tes Enkripsi AES



Gambar 4.66: Grafik Rata-Rata Waktu Tes Dekripsi AES

Grafik 4.65 dan 4.66 menunjukkan *speedup* dari setiap konfigurasi yang diuji. Dari grafik tersebut, dapat dilihat bahwa konfigurasi yang memiliki *speedup* tertinggi adalah konfigurasi AES dengan nilai *speedup* 2.17 untuk enkripsi dan 2.679 untuk dekripsi. Dengan masing masing rata-rata waktu untuk melakukan enkripsi dan dekripsi adalah $7.9 \mu\text{s}$ dan $7.8 \mu\text{s}$.

BAB 5

KESIMPULAN

Berdasarkan hasil pengujian hasil tuning hypervisor dari tiga skenario berbeda yang telah dilakukan, dapat disimpulkan berbagai hal:

1. Flag default yang disediakan oleh hypervisor bersifat minimal, sehingga dapat dilakukan tuning untuk menambahkan fitur-fitur yang lebih spesifik sesuai dengan kebutuhan pengguna.
2. Dari hasil pengujian tuning hypervisor dengan kompresi video dengan menggunakan tools ffmpeg, kombinasi feature flag yang paling baik untuk digunakan adalah SSSE3 dan SSE4.1 dengan kecepatan kompresi video sebesar 2.43 kali lebih cepat dibandingkan dengan flag default.
3. Dari hasil pengujian tuning hypervisor dengan validasi integritas data menggunakan fungsi CRC32, penambahan feature flag sse4.2 dapat mempercepat proses validasi integritas file sebesar 7.68 kali lebih cepat dibandingkan dengan flag default.
4. Dari hasil pengujian tuning hypervisor dengan enkripsi AES, penambahan feature flag aes dapat mempercepat proses enkripsi AES sebesar 2.17 kali lebih cepat dibandingkan dengan flag default.
5. Dari hasil pengujian tuning hypervisor dengan dekripsi AES, penambahan feature flag aes dapat mempercepat proses dekripsi AES sebesar 2.68 kali lebih cepat dibandingkan dengan flag default.

5.1 Saran

Setelah dilakukan penelitian dan pengujian terhadap tuning hypervisor untuk *Virtual Machine* di Apache Cloudstack, Penulis menemukan beberapa saran yang bisa menjadi pertimbangan untuk meningkatkan penelitian pada topik ini kedepannya:

- Menggunakan jenis CPU yang berbeda untuk melakukan tuning hypervisor, karena setiap jenis CPU memiliki karakteristik dan fungsi yang berbeda.
- Melakukan evaluasi pengaruh tuning hypervisor terhadap kinerja aplikasi yang berbeda, misalnya, aplikasi web, basis data, atau aplikasi real-time.

- Memberikan contoh kasus penggunaan yang lebih kompleks, seperti penggunaan hypervisor untuk kebutuhan *machine learning* atau *big data* dan menganalisis efek dari tuning hypervisor pada kasus tersebut.
- Untuk konteks penelitian tuning hypervisor, dapat digunakan jenis Cloud Management Platform (CMP) yang berbeda, seperti OpenStack, ManageIQ, atau Cloudify.
- Membandingkan efektivitas tuning hypervisor pada lingkungan virtualisasi yang berbeda, seperti (*full virtualization*), paravirtualisasi, atau kontainerisasi.

DAFTAR REFERENSI

- [1] P. Mell and T. Grance. The nist definition of cloud computing. *National Institute of Standards and Technology, Information Technology Laboratory*, 2011.
- [2] Apache Software Foundation. Apache cloudstack: About. <https://cloudstack.apache.org/about.html>. Accessed: 2023-11-19.
- [3] M. Scarfone, K. Souppaya and P. Hoffman. Guide to security for full virtualization technologies. *National Institute of Standards and Technology, Information Technology Laboratory*, 2011.
- [4] Amazon Web Services Inc. What is kvm (kernel-based virtual machine)? <https://aws.amazon.com/what-is/kvm/>. Accessed: 2023-11-19.
- [5] Rajkumar Buyya, James Broberg, and Andrzej M. Goscinski. *Cloud computing: Principles and Paradigms*. Wiley, March 2011.
- [6] TechTarget. Cloudstack. <https://www.techtarget.com/whatis/definition/CloudStack>, July 2019. Accessed: 2023-11-20.
- [7] Jordan MacPherson. What is a hypervisor? - types, benefits and how does it work? <https://www.parkplacetechologies.com/blog/what-is-hypervisor-types-benefits/>, April 2023. Accessed: 2023-11-20.
- [8] Amazon Web Services Inc. What's the difference between type 1 and type 2 hypervisors? <https://aws.amazon.com/compare/the-difference-between-type-1-and-type-2-hypervisors/>. Accessed: 2023-11-20.
- [9] Luca Abeni and Dario Faggioli. Using xen and kvm as real-time hypervisors. *Journal of Systems Architecture*, 106:101709, June 2020.
- [10] J.V. Pradilla and C.E. Palau. *Micro Virtual Machines (MicroVMs) for Cloud-assisted Cyber-Physical Systems (CPS)*, page 125–142. Elsevier, 2016.
- [11] IBM. What are virtual machines? — IBM — ibm.com. <https://www.ibm.com/topics/virtual-machines>. [Accessed 30-11-2023].

- [12] Faiz Muqorrir Kaffah, Yana Aditia Gerhana, Ihsan Miftahul Huda, Ali Rahaman, Khaerul Manaf, and Beki Subaeki. E-mail message encryption using advanced encryption standard (aes) and huffman compression engineering. In *2020 6th International Conference on Wireless and Telematics (ICWT)*, pages 1–6, 2020.
- [13] National Institute of Standards and Technology (U.S.). Advanced encryption standard (aes). May 2023.
- [14] Xinmiao Zhang and Yok Jye Tang. Low-complexity parallel cyclic redundancy check. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2021.
- [15] Nandivada Sridevi, K. Jamal, and Kiran Mannem. Implementation of cyclic redundancy check in data recovery. In *2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC)*, pages 17–24, 2021.
- [16] libvirt: virsh — libvirt.org. <https://libvirt.org/manpages/virsh.html>. [Accessed 06-12-2023].
- [17] SSE (Streaming SIMD Extensions) — Microprocessor Types and Specifications — InformIT — informit.com. <https://www.informit.com/articles/article.aspx?p=130978&seqNum=8>. [Accessed 17-12-2023].
- [18] Michael Matz, Jan Hubicka, Andreas Jaeger, and Mark Mithcell. System v application binary interface amd64 architecture processor supplement. https://refspecs.linuxbase.org/elf/x86_64-abi-0.99.pdf, 2010. [Accessed 17-12-2023].
- [19] M. Hassaballah, S. Omran, and Y. B. Mahdy. A review of simd multimedia extensions and their usage in scientific and engineering applications. *The Computer Journal*, 51(6):630–649, January 2008.
- [20] Intel. Sse4 programming reference, 2007. Intel 64 and IA-32 Architectures.
- [21] Apache Software Foundation. Host kvm installation c 2014; apache cloudstack 4.15.0.0 documentation — docs.cloudstack.apache.org. <https://docs.cloudstack.apache.org/en/4.15.0.0/installguide/hypervisor/kvm.html>. [Accessed 24-03-2024].
- [22] Cetin, Nurhan and Nagel, Kai. Large scale transportation simulations on beowulf clusters. Technical report, 2001.

LAMPIRAN

LAMPIRAN 1

The Advanced Encryption Standard (AES) is a symmetric-key encryption algorithm that has become the de facto standard for securing sensitive data worldwide. Developed by Belgian cryptographers Joan Daemen and Vincent Rijmen, AES was adopted by the U.S. National Institute of Standards and Technology (NIST) in 2001 after a rigorous evaluation process involving numerous submissions from around the globe. AES replaced the aging Data Encryption Standard (DES) and its variants, which were becoming increasingly vulnerable to brute-force attacks due to advances in computing power. AES operates on fixed block sizes of 128 bits and supports key lengths of 128, 192, or 256 bits, with the latter providing the highest level of security. Its sophisticated mathematical structure, based on substitution-permutation networks, makes it highly resistant to various cryptanalytic attacks, including differential and linear cryptanalysis. AES employs multiple rounds of substitution, permutation, and key mixing operations to scramble the plaintext data, making it virtually impossible to decrypt without the correct key. One of the key strengths of AES is its versatility and flexibility. It can be implemented in hardware or software, making it suitable for a wide range of applications, from securing communication channels to protecting data at rest. AES has been extensively studied and scrutinized by cryptographers worldwide, and its strength lies in its ability to withstand known attacks while maintaining high performance. Consequently, AES has been widely adopted in various industries, including government, finance, healthcare, and telecommunications, as well as in protocols and standards such as TLS/SSL, IPsec, and wireless encryption standards like WPA2.