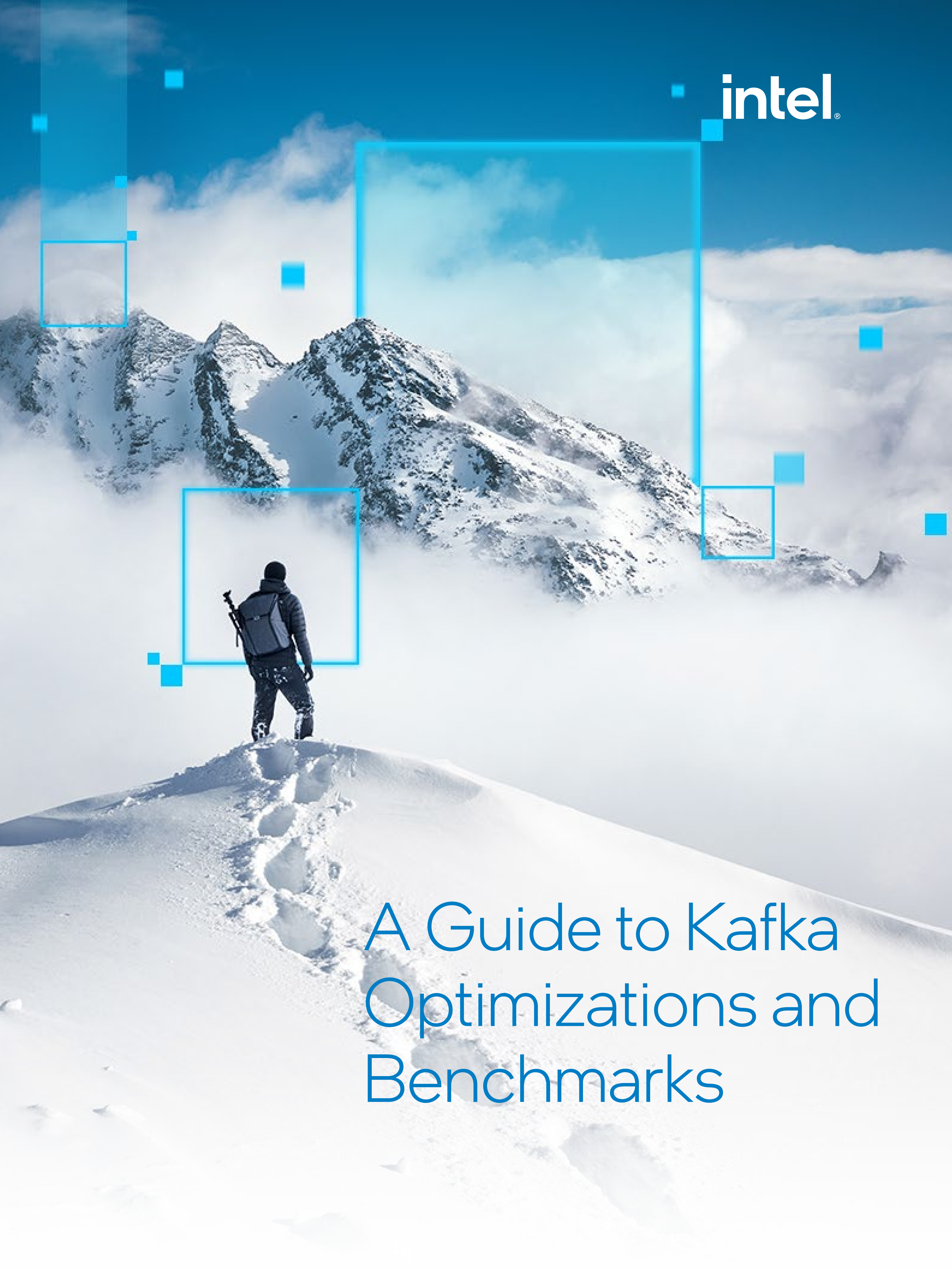


A Guide to Kafka Optimizations and Benchmarks

Authors

Debashis Paul
Cloud Solutions Engineer

Roberto Baturoni
Cloud Solutions Engineer

Collaborators

David Shade
Cloud Solutions Architect

William Fowler
Cloud Solutions Architect

Jun Chen
Cloud Software Development Engineer

Sunny Wang
Cloud Software Development Engineer

Acknowledgments

Marco Carlo Changho
Performance Marketing Engineer

Padma Apparao
Principal Engineer, AI and Cloud Performance Architect

Andres Mejia
AI Software Development Engineer

Murali Madhanagopal
Cloud and AI Architect

Suleyman Sair, PhD
Principal Engineer, Cloud Software

Table of Contents

Overview

Apache Kafka is an open-source Distributed Event streaming platform used for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications. It is a Publish-Subscribe real-time messaging system to process data in a resilient, fault tolerant, horizontally scalable way.

Because Kafka is a high volume and low latency message broker, we need a fast (but still secure) encryption algorithm capable of encrypting an arbitrary amount of data. Kafka Producer must encrypt the messages before pushing them over the network into Kafka Consumer, which then needs to decrypt them upon retrieval. Kafka supports the encryptions using Transport Layer Security (TLS). Enabling TLS causes performance impact due to encryption overhead. Apache Kafka does not directly support any form of encryption-at-rest for data stored at a broker.

This handbook will dive into several use cases to show Kafka workload optimizations in 3rd Generation Intel Xeon Scalable CPUs to accelerate the encryption process through hardware across different compression methods against different JDK versions.

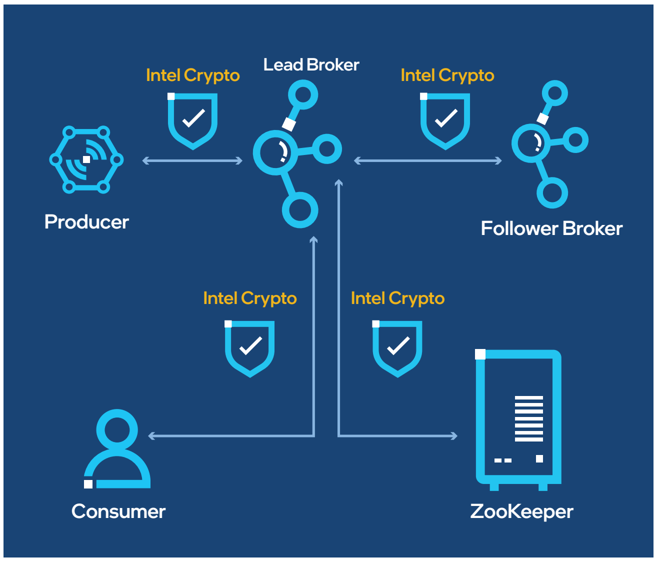
Intel® Crypto for Kafka Encryption Acceleration

New 3rd Gen Intel Xeon Scalable processors, introduce enhanced cryptographic operations called Crypto accelerations contributing to improved performance for Apache Kafka workloads where encryption and decryption are enabled. The new Crypto instructions set supports implementation of stronger encryption protocols without compromising performance by reducing compute cycles allocated for cryptography processing.

As high volumes of data will be encrypted, symmetric key encryption is the natural choice for efficiently ensuring the confidentiality of stored Kafka topic messages. The Advanced Encryption Standard (AES) is an efficient symmetric encryption algorithm. Intel® AES instructions are supported by 3rd Gen Intel Xeon Scalable processor (Code name: Ice Lake) Vector Advanced Encryption Standard (VAES) for faster processing of cryptographic algorithms, constant time encryption, and resilience to certain side-channel attacks. The new AES-NI instruction set is comprised of six new instructions that perform several compute intensive parts of the AES algorithm. These instructions can execute using significantly less clock cycles than a software solution.

Galois/Counter Mode (GCM) is an authenticated encryption mode for block ciphers. AES-GCM is not only efficient and secure, but hardware implementations can achieve high speeds with low cost and low latency, because the mode can be pipelined. Applications that require high data Throughput can benefit from these high-speed implementations. AES-GCM is optimized with newest JDK software e.g., OpenJDK 11.0.11 and later which leverages power of VAES and VPCLMULQDQ instructions from Intel® AVX-512 instruction set family to accelerate the Kafka streaming performance while reducing the CPU overhead due to encryptions. TLS Cryptography protocols uses AES-GCM cipher suites to optimize Kafka Broker Throughput performance and reducing encryption overhead without impacting the latency SLA.

Intel has also developed compression plug-in solution as an extension of certain compression algorithm, improving Throughput, latency, and compression ratios for Kafka workload.



Optimize Apache Kafka Streaming

Optimizing the Kafka streaming performance is a key challenge for any enterprise for better SLA, increased TCO, better user experience, and satisfying the compliance requirements.

Below are several ways to achieve better Kafka performance on Intel® platforms:

- Switching to the latest generation Intel Xeon processor to leverage Advanced Crypto accelerations
- Scaling to choose higher vCPU Intel-based instances
- Use optimized JDK versions to benefit from Crypto upstream features
- Use best Compression methods based on use cases
- Use OpenSSL TLS or advanced JDK SSL features
- Use Intel Libraries for better Compressions based on use cases

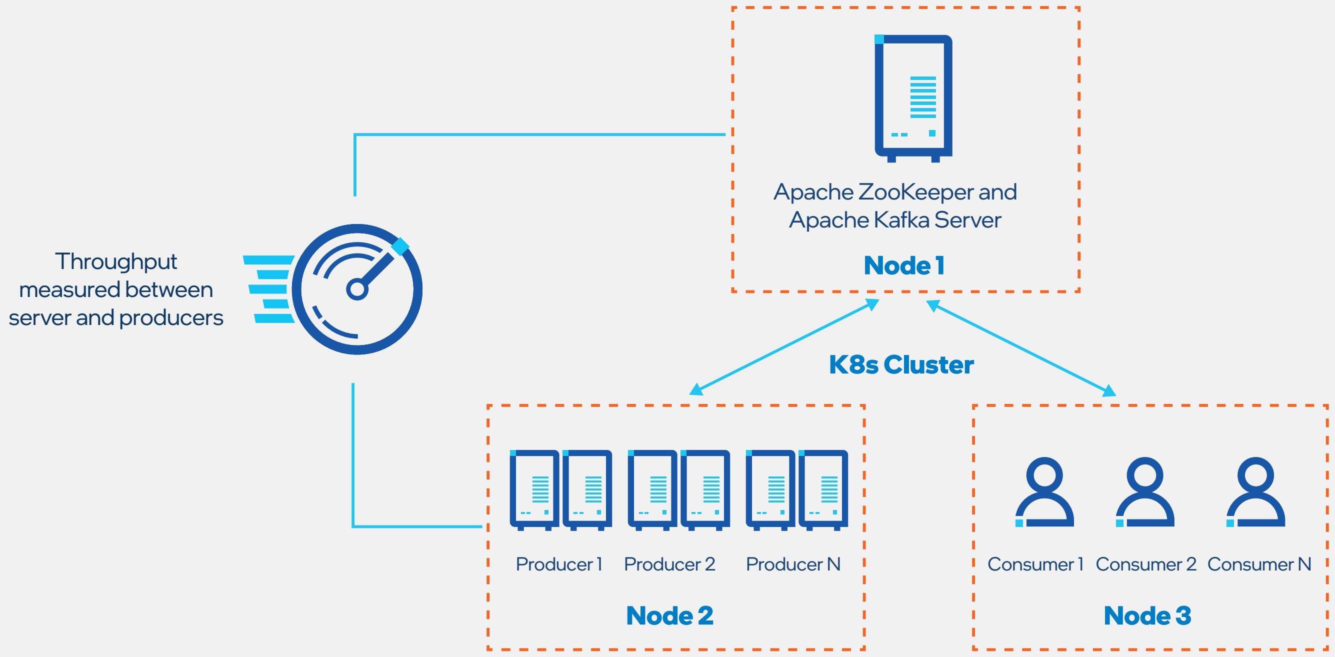
Performance Benchmark Test

For capacity planning tied to SLA requirement, it is important to run the benchmark testing to achieve the optimized throughput and best user experience as different environments, workloads, and use-cases have specific needs.

The below benchmark test cases were performed across different Intel Xeon-SP generations CPUs, JDK versions, compression methods, and encryption scenarios in AWS cloud instances. For test cases executed on specific AWS instances and related Hardware configurations, Kafka software, and workload configurations refer to Appendix A.

Workload Architecture

This workload is measuring Apache Kafka's streaming performance by utilizing the built-in standard application tool. Currently, the test case measures Apache Kafka Producer and Consumer performance. Intel has used the diagram below as Kafka benchmarking framework for testing. The workload executed using standard embedded scripts 'kafka-producer-perf-test.sh' and 'kafka-consumer-perf-test.sh' for performance harness.



Process/Methodology

Benchmark process executed in Intel internal framework with below mentioned steps:

- Perform Kafka Producer – publish millions of messages per thread to Broker Kafka server
- Perform Kafka Consumer – read millions of subscribed messages per thread
- The workload contains three docker images:
  - Producer (generate and send messages to Kafka and ZooKeeper server)
  - Kafka-ZooKeeper-server (receive messages from Producer and send messages to Consumer)
  - Consumer (get messages from Kafka and ZooKeeper server)
- Three Kubernetes worker nodes used for this test case to host Producer, Broker, and Consumer containers in PODs.
- Each POD is assigned to each Kubernetes node using Anti-affinity setup. Producer has 1 POD, Broker has 1 POD, and Consumer has 1 POD. Testing is done with Replication factor 1 with 1 partition.
- Median value of three Runs taken for Max Throughput and P99 Latency to avoid outliers.
- Measure the p99 Latency and aggregate transmitted Throughput Producer to Broker.

Encryption ON

33%

Throughput improvement with 3rd Gen Ice Lake instances (m6i)

12%

Latency improvement vs. older generation (m5)

KPI – Key Performance Indicators

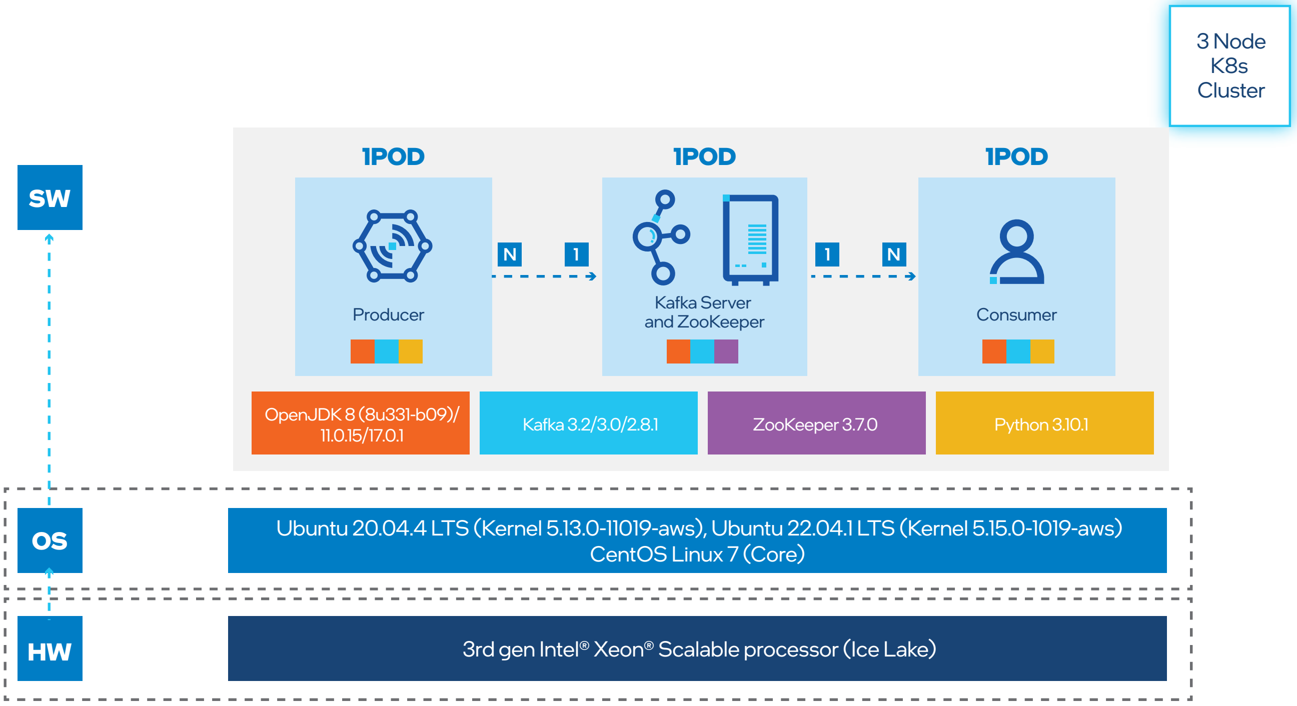
This benchmark results focuses on two KPIs:

#1 – Max Throughput (in MB/second) which measures the sum of Producer messages that arrive to Broker within a specific amount of time.

#2 – Producer P99 Latency—the time it takes for a record produced to Kafka to be fetched by the Consumer. P99 Latency is standard tail latency measures how much end-to-end latency 99th percentile of time.

Software Stack

Any change in configurations have been called out in individual optimizations area to override:


3 Node K8s Cluster
SW
1POD
Producer
N
1
1POD
Kafka Server and ZooKeeper
1
N
1POD
Consumer
OpenJDK 8 (8u331-b09)/ 11.0.15/17.0.1
Kafka 3.2/3.0/2.8.1
ZooKeeper 3.7.0
Python 3.10.1
OS
Ubuntu 20.04.4 LTS (Kernel 5.13.0-11019-aws), Ubuntu 22.04.1 LTS (Kernel 5.15.0-1019-aws) CentOS Linux 7 (Core)
HW
3rd gen Intel® Xeon® Scalable processor (Ice Lake)

1. Intel Gen-to-Gen Kafka Performance Comparison

Both Throughput and Latency KPI depends on choice of hardware or cloud provider so it is important to understand what hardware acceleration and software can help to achieve your specific latency goals in your unique environment.

In the below test Kafka performance comparison done between 3rd Gen Intel Xeon Scalable processor instances and 2nd Gen Intel Xeon Scalable processor instances in Amazon AWS cloud m5, m6i, and i4i instances. It shows the performance difference across storage, compute, and memory optimized AWS instances. In the sections below, any changes made to the baseline configurations are called out.

1.1: Intel Gen-2-Gen AWS m6i (3rd gen) vs. m5 (2nd gen) – Encryption OFF

The performance test is done on 2nd Gen Intel Xeon Scalable processor m5.4xl (16 vCPU) vs. 3rd Gen Intel Xeon Scalable processor-based m6i.4xl (16 vCPU) in Open JDK 11.0.15 version while Kafka Encryption config is turned off.

Performance Summary

3rd Gen Intel Xeon Scalable processor instances (m6i) shows 30% Throughput improvement vs. older generation (m5); for Throughput: higher is better and for P99 Latency: lower is better. See Table 1.1 and 2.1 from the Appendix A for configuration details.

1.2: Intel Gen-2-Gen AWS m6i (3rd gen) vs. m5 (2nd gen) – Encryption ON

The performance test is done on 2nd Gen Intel Xeon Scalable processor m5.4xl (16 vCPU) vs. 3rd Gen Intel Xeon Scalable processor-based m6i.4xl (16 vCPU) in Open JDK 11.0.15 version while Kafka Encryption config is turned on.

Performance Summary

3rd Gen Intel Xeon Scalable processor instances (m6i) shows 33% Throughput improvement and 12% Latency improvement vs. older generation (m5). Improvement because of Intel VAES crypto instructions for 3rd gen; for Throughput: higher is better and for P99 Latency: lower is better. See Table 1.1 and 2.1 from the Appendix A for configuration details.

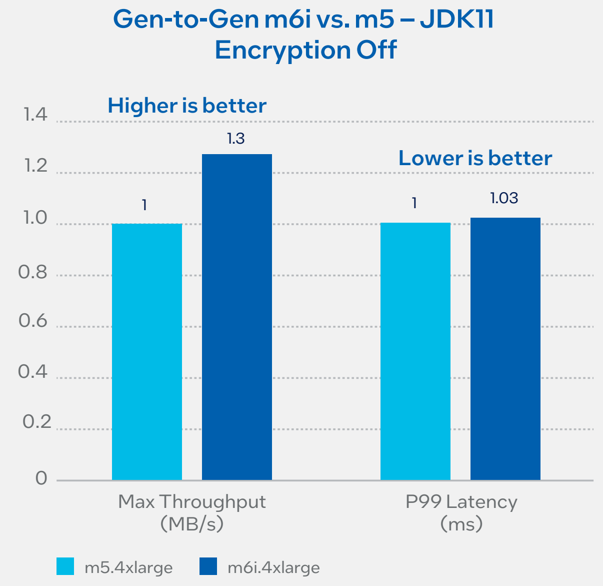


Figure 1: Intel® AWS m6i (3rd Gen) vs. m5 (2nd Gen) – Encryption OFF.

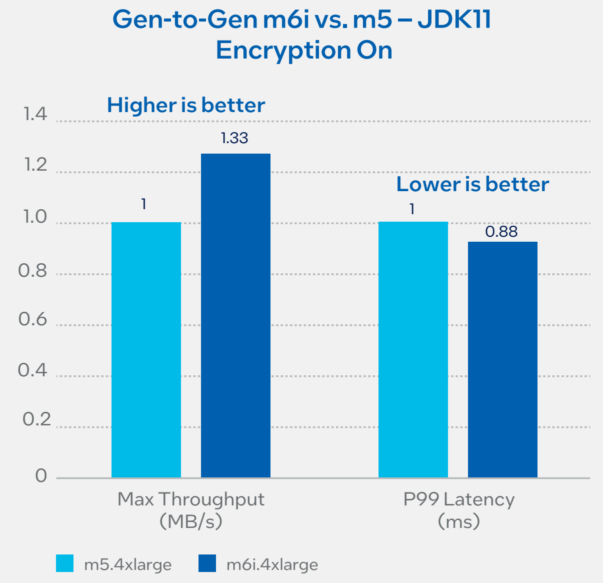


Figure 2: Intel® AWS m6i (3rd Gen) vs. m5 (2nd Gen) – Encryption ON.

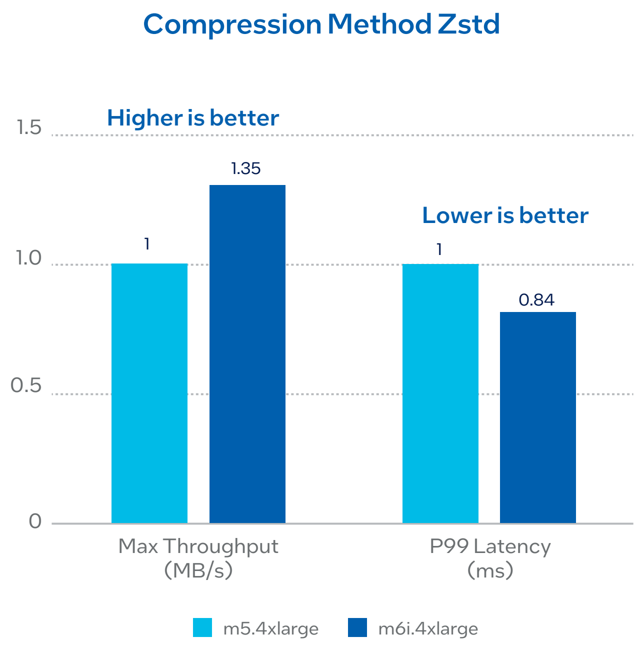
1.3: Intel Gen-2-Gen AWS m6i (3rd Gen) vs. m5 (2nd Gen) with Compression

The performance test is done on 2nd Gen Intel Xeon Scalable processor m5.4xl (16 vCPU) vs. 3rd Gen Intel Xeon Scalable processor m6i.4xl (16 vCPU) in Open JDK 11.0.15 version.

The following images will show the optimization on Throughput and latency across two different compression methods Zstd and LZ4.

Performance Summary

- AWS m6i.4xlarge instance shows 35% Throughput improvement in Zstd and 30% Throughput improvement in LZ4 against AWS m5.4xl instance.
- AWS m6i.4xlarge instance shows 12% Latency improvement in Zstd and 36% Latency improvement in LZ4 against AWS m5.4xl instance; for Throughput: higher is better and for P99 Latency: lower is better. See Table 1.1 and 2.1 from Appendix A for configuration details.



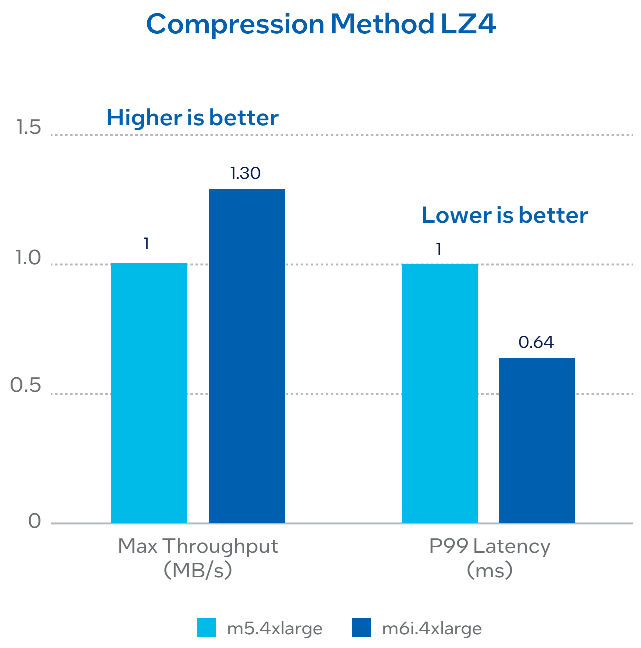


Figure 3: Intel® AWS m6i (3rd Gen) vs. m5 (2nd Gen) – Zstd / LZ4 compression.

2. Kafka Performance – CPU Scaling AWS i4i

Kafka performance comparison done across multiple 3rd Gen Intel Xeon Scalable processor instances in Amazon AWS cloud 'i4i' instances. In the test below, number of Producers, number of Brokers, number of Consumers, and total partitions also increased linearly (4x).

For e.g., the i4i.4xlarge instance with 16 vCPU having 32 Brokers, Consumers, Producers, partitions.

2.1: Intel® AWS i4i Instances (LZ4 compression)

The performance test is done on Intel Xeon Ice Lake i4i.xlarge(4vCPU), i4i.2xlarge(8vCPU), i4i.4xlarge(16vCPU) in Open JDK 11.0.15 version.

The following images will show the optimization on Throughput and Latency for compression methods LZ4.

Performance Summary

- 3rd Gen Intel Xeon Scalable processor AWS i4i instance CPUs scaling shows linear % Max Throughput improvement with LZ4 compression. Brokers, Consumers, and partitions also scaled along with instances. For Throughput: higher is better and for P99 Latency: lower is better. See Table 1.2 and 2.2 from Appendix A for configuration details.

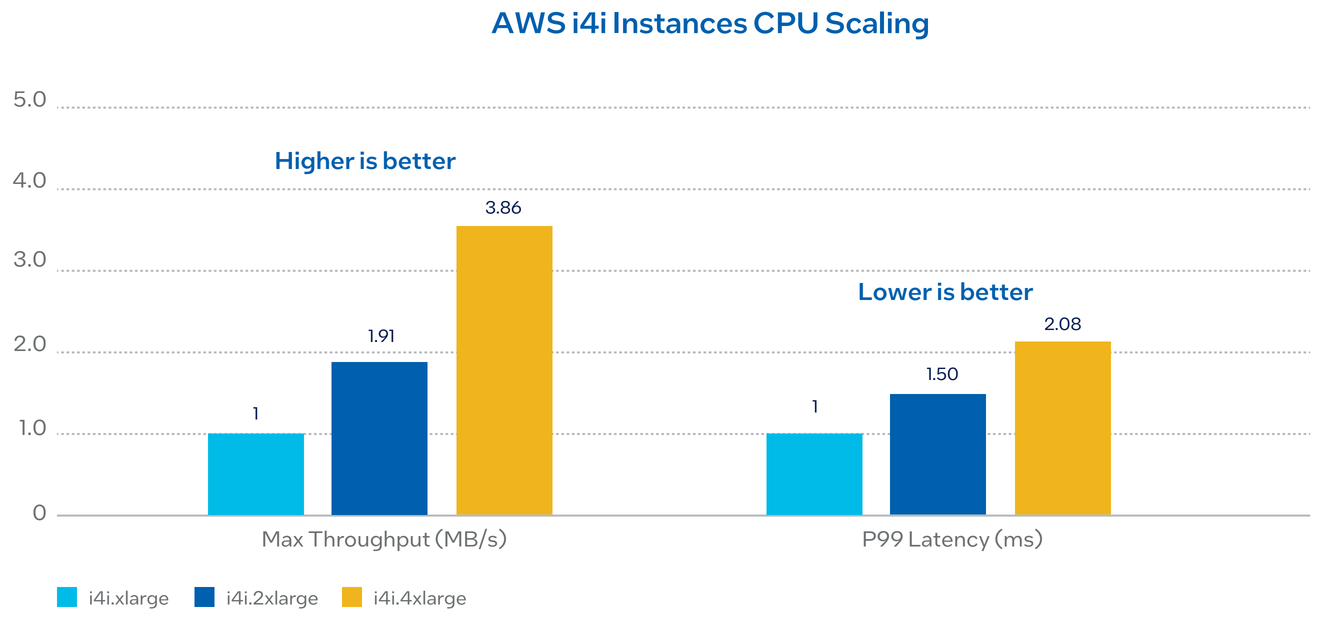


Figure 4: Intel® AWS i4i Instances scaling (LZ4 compression).

3. Kafka Encryption Performance Across Java Versions

The transparent end-to-end encryption in Kafka done via Java serializer and de-serializer implementation utilizes Intel VAES Crypto instructions set. 3rd Gen Intel Xeon Scalable processor instructions supports the operations of Crypto algorithms for simultaneous execution and a method allowing parallel processing of multiple independent databuffers giving the Crypto acceleration boost of Kafka stream processing performance.

Intel team has upstreamed several Crypto instructions set supports in OpenJDK version 11.0.11+ via Open-source Java community. This document will provide the details on Kafka performance comparison across different Java versions.

3.1 Kafka Throughput in Intel AWS m6i (3rd Gen) vs. m5 (2nd Gen) Across JDK and Encryptions

Kafka performance comparison done across 3rd Gen Intel Xeon Scalable processor in Amazon AWS cloud 'm6i.4xlarge' and 'm5.4xlarge' instances across JDK 8 vs. JDK 11 versions for different Encryption setting. JDK 11 and higher version provides the Intel® Crypto acceleration against no Crypto support for JDK 8 hence attributed to significant performance improvements.

Performance Summary

3rd Gen Intel Xeon Scalable processors AWS instance shows ~25-30% Throughput improvement against 2nd Gen Intel Xeon Scalable processors; for Throughput: higher is better and for P99 Latency: lower is better. See Table 1.1 and Table 2.1 from the Appendix A for configuration details.

3.2 Kafka Throughput and Latency on Intel AWS i4i.4xlarge Across JDK

Kafka performance comparison done across 3rd Gen Intel Xeon Scalable processor in Amazon AWS cloud 'i4i.4xlarge' instances across JDK 8 vs. JDK 11 versions for compression Zstd while Encryptions are turned on. JDK 11 and higher provides the Intel Crypto acceleration against no Crypto support for JDK 8 hence attributed to significant performance improvements.

Performance Summary

3rd Gen Intel Xeon Scalable Processors AWS instance i4i.4xlarge shows 26% Throughput and 39% Latency improvement JDK 8 to JDK 11 with Encryption ON; for Throughput: higher is better and for P99 Latency: lower is better. See Table 1.2 and Table 2.2 from Appendix A for configuration details.

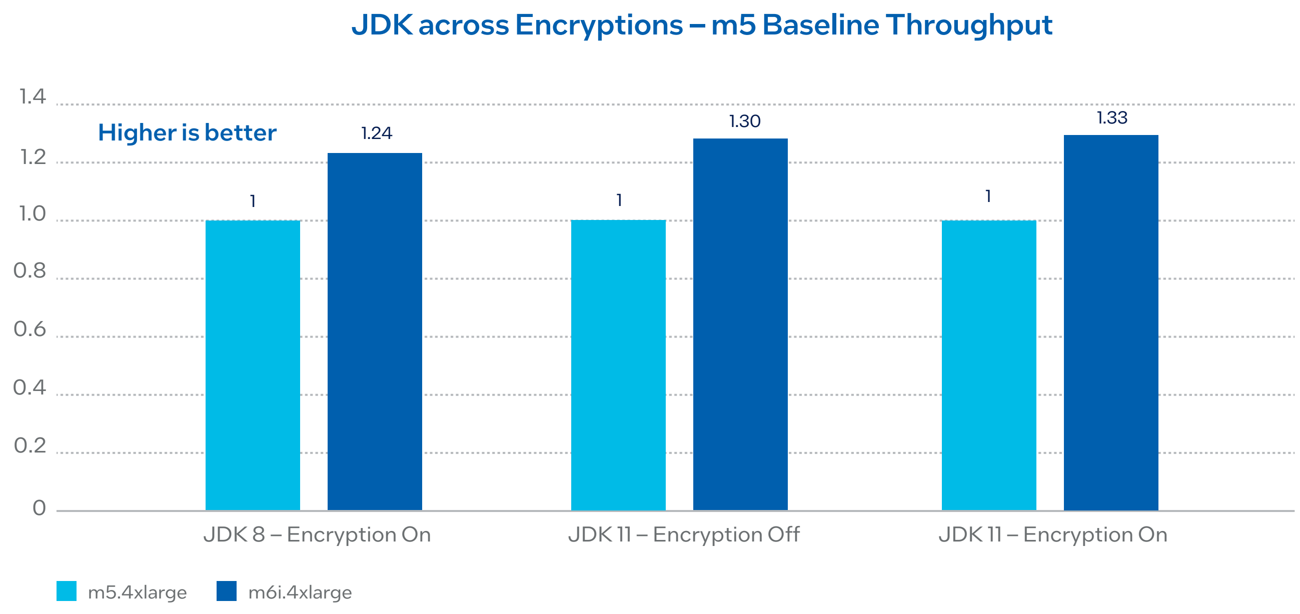


Figure 5: Intel® AWS m6i 3rd Gen vs. m5 2nd Gen – No compression.

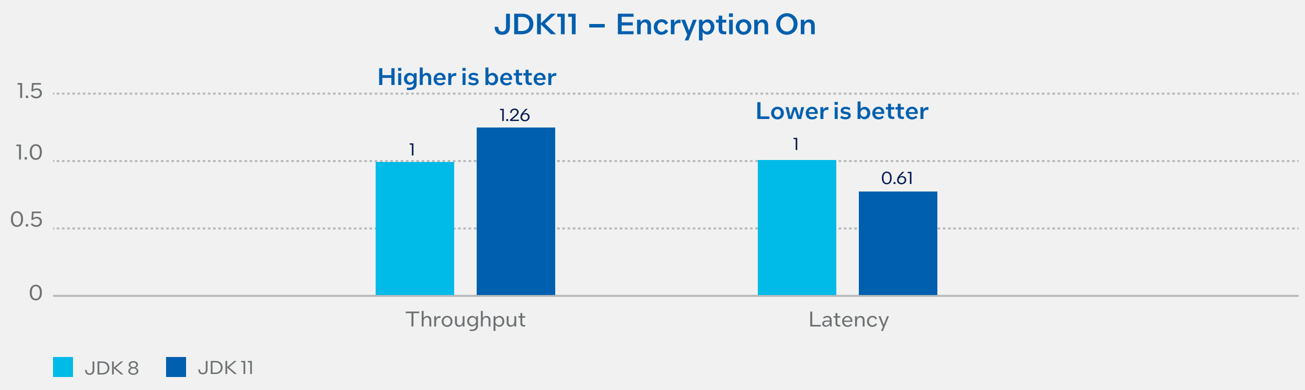


Figure 6: Intel® AWS i4i.4xlarge Latency and Throughput for JDK 8 vs. JDK 11.

4. Kafka Compression in Intel AWS Instances

Compression has a huge significance in Kafka workload performance. By default, Kafka messages are not compressed, compressing data batches improves Throughput and reduces the load on physical storage (with replication it would be even more) plus data transmitted over the network will be reduced. Message compression adds latency in the Producer (CPU time spent compressing the messages) but it is not always suitable for low-latency applications where the cost of compression or decompression has zero tolerance.

From Producers to Broker Throughput with different compression algorithms inhibits significant difference vs. no compression.

In the below graph Throughput and latency impact in a compressed and non-compressed data is outlined. How compression varies across different algorithms and performance impact is shown. Intel has also developed a plugin solution on top of 'gzip' compression to improve Latency for Max Throughput.

4.1 Kafka Compression Performance Comparison AWS m6i.4xlarge

Performance on Kafka data across different compression algorithms is shown in the below chart. All tests on m6i.4xlarge have been done with 32 (double the vCPUs) Producers/Brokers/Consumers and partitions.

Performance Summary

P99 Latency is better when 'gzip' compression is applied comparing other compression methods. Excluding 'gzip' other compression methods are showing better Throughput; for Throughput: higher is better and for P99 Latency: lower is better. See Table 1.1 and 2.3 from Appendix A for configuration details.

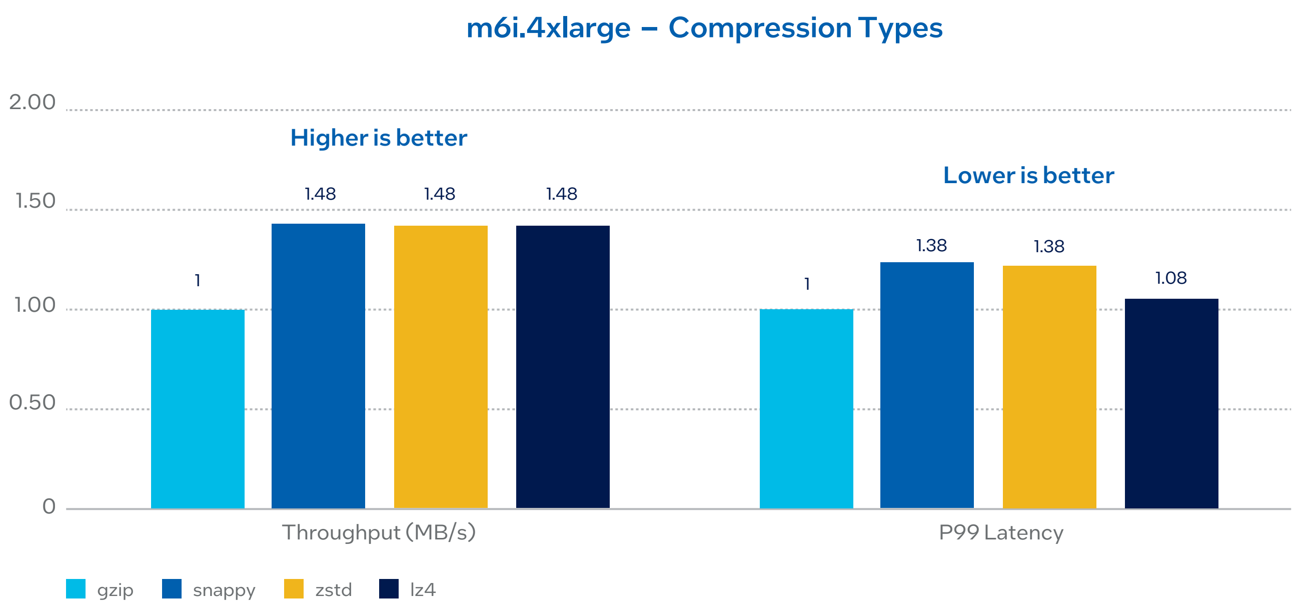


Figure 7: Intel® AWS m6i.4xlarge across compression types – Encryption OFF.

4.2 Kafka Compression Optimizations on Intel Libraries

Kafka compression process helps to achieve two things: Reducing network bandwidth usage and saving disk space on Kafka Brokers. However, the tradeoff would be dispatch latency because of higher CPU utilization due to compression. Gzip is known to have the highest compression ratios with high CPU usage but slowest compression speed (latency). In certain use cases gzip is more desirable cost optimize solution against LZ4, Zstd, or Snappy.

Intel® Ingenuity Partner Program (Intel® IPP) multithreaded software library with Zlib interface improve the default gzip latency. And Intel Intelligent Storage acceleration library (ISA-L) optimizes the storage Throughput with functions for RAID, erasure code, cyclic redundancy check (CRC) functions, cryptographic hash, encryption, and compression. The below graph presents Intel solution over native Java gzip to show performance boost.

Performance Summary

Intel® Storage Acceleration Library (Intel® ISA-L) improves Throughput by 1.47x and Intel IPP improves Throughput by 1.15x comparing Java native gzip Intel ISA-L improves Latency by 34% and IPP improves Latency by 8% comparing Java native gzip; for Throughput: higher is better and for P99 Latency: lower is better. See Table 1.3 and Table 2.4 from Appendix A for configuration details.

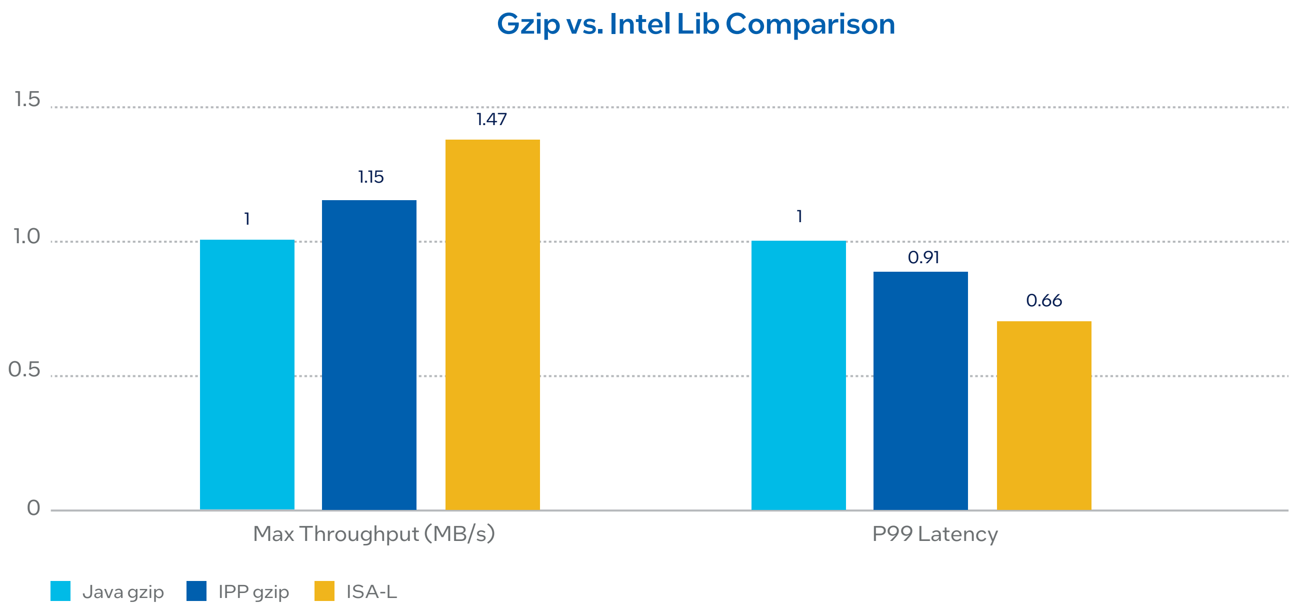


Figure 8: 3rd Gen Intel® Xeon® Scalable processor with Intel Compression library.

5. Intel's Contributions on Open Source Optimization

OpenJDK – Upstream/Backport Support for VAES Crypto

Intel team has contributed to OpenJDK community so that Java can leverage performance acceleration Crypto features support from Intel AVX-512 VAES (Vectorized Advanced Encryption Standard) instruction set in 3rd Gen Intel Xeon Scalable processors.

Intel team also backported several Crypto/Hash acceleration support features from future JDK versions (JDK 12+) to JDK 11 LTS and JDK 11.0.15 which differentiate the overall Java performance in JDK 11 to boost performance for numerous Java dependent workloads including Kafka. This enhancement is contributed by Intel and sponsored by the hotspot compiler team.

Kafka Community – TLS Regression

Kafka supports TLS for both encryption and authentication. TLS cryptographic protocol uses AES-GCM which can be CPU intensive. If a server has negotiated TLS 1.3 it must terminate the connection with an "unexpected message" alert. TLS 1.3 On Kafka 2.7 doesn't support renegotiation creating intermittent disconnections in Brokers before read/write is completed impacting p99 Latency.

While working with a customer, Intel engineers found the issue for JDK 11 and TLS 1.3 and suggested a fix, customer applied the fix to resolve the issue which has been requested to upstream (https://issues.apache.org/jira/browse/KAFKA-13418) to Kafka community.

OpenSSL – SSL & TLS with Several Cryptographic Functions Including AES

The OpenSSL project provides an open-source implementation of the SSL/TLS protocols and is a commonly deployed library for SSL/TLS world-wide which can be used in Kafka clients and Broker communication. Confluent Kafka broadly adopted OpenSSL for TLS. OpenSSL implementation can have better performance comparing to JDK SSL.

Asynchronous OpenSSL is a non-blocking approach that supports a parallel-processing model at the cryptographic level for SSL/TLS protocols, which in turn allows for other types of optimizations. This capability allows cryptographic transformations to be processed on dedicated hardware engines or on separate logical cores. Intel® QuickAssist Technology (Intel® QAT) Engine on Open SSL can boost the overall TLS performance.

Intel QAT OpenSSL Engine (QAT_Engine) supports acceleration for both hardware as well as optimized software based on vectorized instructions.

Intel® Storage Acceleration Library (Intel® ISA-L)

Intel ISA-L provides tools to minimize disk space use and maximize storage Throughput, security, and resilience. Intel ISA-L is a collection of optimized low-level functions targeting storage applications. Intel ISA-L helps improve compression and Throughput performance and reduce latency for a storage application with erasure coding that uses Reed-Solomon error correction. Intel ISA-L proves increase gzip compression performance (better Throughput) using Intel implementation called IGZIP. Intel ISA-L Crypto accelerates multi-buffer cryptography hashes providing better Throughput leveraging vector SIMD instructions set and improved AES ciphers. This can optimize Kafka performance.

Intel® Integrated Performance Primitives (Intel® IPP) Cryptography

Intel IPP Cryptography is a secure, fast, and lightweight library of building blocks for cryptography, highly optimized for various Intel CPUs that includes 3rd Gen Intel Xeon Scalable processor. It optimizes hardware cryptography instructions support using several Intel® Streaming SIMD versions and various AVX Instructions sets. Intel Integrated Performance Primitives which is multi-threaded software library (part of Intel OneAPI toolkit) which shows better Kafka performance using IPP gzip which is the Intel patched version of native Java gzip solution.

6. Intel's Continued Innovation to Optimize Apache Kafka

- Better Kafka performance using JDK 18 optimized CRC32, interleaved GCM functions on Intel hardware.
- New 4th Gen Intel Xeon Scalable Processor QAT accelerator engine improves Crypto acceleration and data De/compression while offloading the CPU.
- JDK 18 improved Java array copy/clear to use 512-bit wide vector width instruction for 4th Gen Intel Xeon Scalable processor.
- Intel Granulate (https://granulate.io/solutions/intel/) application and workload performance optimization solution saves CPU utilization without any code changes. It can reduce costs by up to 60% while saving CPU up to 25-40%.

Appendix A Configurations

The following tables show the full configuration details for the test environment, platforms, and software.
All performance results are based on these configurations and tests by Intel on April 12, 2022 - Sept. 21, 2022.

Table 1.1: Hardware Configuration Used for this Testing

	m6i.4xlarge	m5.4xlarge
Manufacturer	Amazon EC2	Amazon EC2
Product Name	m6i.4xlarge	m5.4xlarge
BIOS Version	1	1
Microcode	0xd000331	0x500320a
IRQ Balance	Enabled	Enabled
CPU Model	Intel® Xeon® Platinum 8375C CPU @ 2.90 GHz	Intel® Xeon® Platinum 8259CL CPU @ 2.50 GHz
Base Frequency	2.9 GHz	2.5 GHz
Maximum Frequency	3.5 GHz	3.5 GHz
All-Core Maximum Frequency	3.5 GHz	3.1 GHz
CPU(s)	16	16
Thread(s) per Core	2	2
Core(s) per Socket	8	8
Socket(s)	1	1
NUMA Node(s)	1	1
Prefetchers	DCU HW, DCU IP, L2 HW, L2 Adj.	DCU HW, DCU IP, L2 HW, L2 Adj.
Turbo	Enabled	Enabled
Frequency	2,899 MHz	2.5 GHz
Max C-State	9	9
Installed Memory	64 GB (1x64 GB DDR4 3,200 MT/s [Unknown])	64 GB (1x64 GB DDR4 2,933 MT/s [Unknown])
Huge Pages Size	2,048 kB	2,048 kB
Transparent Huge Pages	madvise	madvise
Automatic NUMA Balancing	Disabled	Disabled
NIC Summary	1x Elastic Network Adapter (ENA)	1x Elastic Network Adapter (ENA)
Drive Summary	1x 500G Amazon Elastic Block Store	1x 500G Amazon Elastic Block Store

Table 1.2: Hardware Configuration Used for this Testing

	i4i.xlarge	i4i.2xlarge	i4i.4xlarge
Manufacturer	Amazon EC2	Amazon EC2	Amazon EC2
Product Name	i4i.xlarge	i4i.2xlarge	i4i.4xlarge
BIOS Version	1	1	1
Microcode	0xd000331	0xd000331	0xd000331
IRQ Balance	Enabled	Enabled	Enabled
CPU Model	Intel® Xeon® Platinum 8375C CPU @ 2.90 GHz	Intel® Xeon® Platinum 8375C CPU @ 2.90 GHz	Intel® Xeon® Platinum 8375C CPU @ 2.90 GHz
Base Frequency	2.9 GHz	2.9 GHz	2.9 GHz
Maximum Frequency	3.5 GHz	3.5 GHz	3.5 GHz
All-Core Maximum Frequency	3.5 GHz	3.5 GHz	3.5 GHz
CPU(s)	4	8	16
Thread(s) per Core	2	2	2
Core(s) per Socket	2	4	8
Socket(s)	1	1	1
NUMA Node(s)	1	1	1
Prefetchers	DCU HW, DCU IP, L2 HW, L2 Adj.	DCU HW, DCU IP, L2 HW, L2 Adj.	DCU HW, DCU IP, L2 HW, L2 Adj.
Turbo	Enabled	Enabled	Enabled
Frequency	2.9 GHz	2.9 GHz	2.9 GHz
Max C-State	9	9	9
Installed Memory	32 GB (1x64 GB DDR4 3,200 MT/s [Unknown])	64 GB (641x64 GB DDR4 3,200 MT/s [Unknown])	128 GB (1x64 GB DDR4 3,200 MT/s [Unknown])
Huge Pages Size	2,048 kB	2,048 kB	2,048 kB
Transparent Huge Pages	madvise	madvise	madvise
Automatic NUMA Balancing	Disabled	Disabled	Disabled
NIC Summary	1x Elastic Network Adapter (ENA)	1x Elastic Network Adapter (ENA)	1x Elastic Network Adapter (ENA)
Drive Summary	1x 500 G Amazon Elastic Block Store	1x 500 G Amazon Elastic Block Store	1x 500 G Amazon Elastic Block Store, 1x 3.4T Amazon EC2 NVMe Instance Storage

Table 1.3: Configurations of the Intel Bare metal 3rd Gen Intel® Xeon® Scalable Processor	
HW / SW Configuration for IA Testing (ISA-L and IPP)	
OS	CentOS Linux 7 (Core)
Kernel	5.13.0+
CPU model	ICELAKE – Intel® Xeon® Gold 6348 CPU @ 2.60 GHz
Sockets, Total CPU(s), NUM Count	2, 112, 2
HT, Turbo Boost	YES, YES
Memory	1,024 GB (32x32 GB DDR4 3,200 MT/s [3,200 MT/s])
Disk	Nvme0n1: 3.7T
Network	loopback
BIOS Version	05.01.01
Microcode	0xd0002a0
FWVersion	02.01.00.1127
Kafka	3.0.0
Java	JDK 11.0.15
ISA-L	2.30
IPP	2021.4.0

Table 2.1: Software and Workload Used for this Testing		
Attribute	**m6i.4xlarge**	**m5.4xlarge**
OS_VER	20.04.4	22.04.1
OS_IMAGE	Ubuntu 22.04.1 LTS	Ubuntu 22.04.1 LTS
OPENJDK_VER	jdk-11.0.15	jdk-11.0.15
OPENJDK_PACKAGE	openJDK11U-jdk_x86_linux_hotspot_11.0.15_10.tar.gz	openJDK11U-jdk_x86_linux_hotspot_11.0.15_10.tar.gz
PYTHON_VER	Python-3.10.2	Python-3.10.2
PYTHON_PACKAGE	Python-3.10.2.tgz	Python-3.10.2.tgz
ZooKeeper	3.7.0	3.7.0
ZOOKEEPER_PACKAGE	apache-zookeeper-3.7.0-bin.tar.gz	apache-zookeeper-3.7.0-bin.tar.gz
KAFKA	3.2	3.2
KAFKA_PACKAGE	kafka_2.12-3.2.0.tgz	kafka_2.12-3.2.0.tgz
Kafka Configuration Used for this Testing		
REPLICATION_FACTOR	1	
PARTITIONS	*Twice # of vCPUs	
# OF PRODUCERS	*Twice # of vCPUs	
# OF CONSUMERS	*Twice # of vCPUs	
NUM_RECORDS	5,000,000	
ENCRYPTION	TRUE	
RECORD_SIZE	1,000	
COMPRESSION_TYPE	Zstd/LZ4	
MESSAGES	10,000,000	
CONSUMER_TIMEOUT	600,000	
BATCH_SIZE	524,288	
LINGER_MS	100	

Table 2.2: Software and Workload Used for this Testing

Attribute	i4i.xlarge, i4i.2xlarge, i4i.4xlarge
OS_VER	22.04.1
OS_IMAGE	Ubuntu 22.04.1 LTS
OPENJDK_VER	jdk-11.0.15
OPENJDK_PACKAGE	openJDK11U-jdk_x86_linux_hotspot_11.0.15_10.tar.gz
PYTHON_VER	Python-3.10.2
PYTHON_PACKAGE	Python-3.10.2.tgz
ZooKeeper	3.7.0
ZOOKEEPER_PACKAGE	apache-zookeeper-3.7.0-bin.tar.gz
KAFKA	3.2
KAFKA_PACKAGE	kafka_2.12-3.2.0.tgz
Kafka Configuration Used for this Testing	
REPLICATION_FACTOR	1
PARTITIONS	*Twice # of vCPUs
# OF PRODUCERS	*Twice # of vCPUs
# OF CONSUMERS	*Twice # of vCPUs
NUM_RECORDS	5,000,000
ENCRYPTION	TRUE
RECORD_SIZE	1,000
COMPRESSION_TYPE	Zstd/LZ4
MESSAGES	10,000,000
CONSUMER_TIMEOUT	600,000
BATCH_SIZE	524,288
LINGER_MS	100

Table 2.3: Software and Workload Used for this Testing	
Attribute	m6i.4xlarge
OS_VER	20.04.4
OS_IMAGE	Ubuntu 20.04.4 LTS
OPENJDK_VER	jdk-17.0.1
OPENJDK_PACKAGE	openjdk-17.0.1_linux-x64_bin.tar.gz
PYTHON_VER	Python-3.10.2
PYTHON_PACKAGE	Python-3.10.2.tgz
ZooKeeper	3.7.0
ZOOKEEPER_PACKAGE	apache-zookeeper-3.7.0-bin.tar.gz
KAFKA	2.8.1*
KAFKA_PACKAGE	kafka_2.12-2.8.1.tgz
Kafka Configuration Used for this Testing	
REPLICATION_FACTOR	1
PARTITIONS	*Twice # of vCPUs
# OF PRODUCERS	*Twice # of vCPUs
# OF CONSUMERS	*Twice # of vCPUs
NUM_RECORDS	3,000,000
ENCRYPTION	OFF
RECORD_SIZE	1,000
COMPRESSION_TYPE	gzip/Zstd/Snappy/LZ4
MESSAGES	2,000,000
CONSUMER_TIMEOUT	600,000
BATCH_SIZE	Default
LINGER_MS	Default

2.4: Kafka Configuration Used for this Testing	
REPLICATION_FACTOR	1
PARTITIONS	1
# OF PRODUCERS	112
# OF CONSUMERS	1
# OF BROKERS	1
NUM_RECORDS	5,000,000
MESSAGES	10000000.0
ENCRYPTION	No
RECORD_SIZE	2048
COMPRESSION_TYPE	Gzip/IPP Gzip
BATCH_SIZE	524288.0
LINGER_MS	100 ms