White Paper

Data Center & Artificial Intelligence
Open-Source Cloud Software Databases

Transparent Hardware-Accelerated Compression for zlib on Intel® Xeon® Processors

zlib-accel, a drop-in replacement for zlib, provides higher throughput and lower CPU utilization.

Authors

Olasoji Denloye
Cloud Software Development Engineer

Mohamed Issa
Cloud Software Development Engineer

Shylaja Kokoori
Cloud Software Development Engineer

Mulugeta Mammo
Cloud Software Development Engineer

Tony Ruiz
Cloud Software Development Engineer

Matt Welch
Cloud Software Development Engineer

Abstract

Compression operations consume a substantial portion of compute resources in modern data center applications, creating a critical need for performance optimization. While the zlib compression algorithm remains ubiquitous due to its excellent compression ratio, its CPU-intensive nature limits overall system performance. The zlib-accel library addresses this challenge by leveraging Intel's hardware acceleration capabilities available on 4th Gen Intel® Xeon® processors and later, specifically Intel® QuickAssist Technology (Intel® QAT) and the Intel® In-Memory Analytics Accelerator (Intel® IAA). These built-in hardware accelerators enhance compression, encryption, and analytics, offering significantly higher performance than traditional on-core compression.

Building upon Intel's comprehensive acceleration ecosystem, which includes Intel® Query Processing Library (Intel® QPL), QATzip, and QAT-Java libraries, zlib-accel provides a unique, transparent software shim approach for seamless integration with existing applications. While other Intel acceleration solutions offer powerful capabilities for developers willing to modify their applications, zlib-accel complements this ecosystem by serving as a drop-in replacement that requires no code changes. This transparent approach automatically routes compression workloads to available hardware accelerators, making hardware acceleration accessible to a broader range of applications. By offloading compression tasks from CPU cores to dedicated hardware, zlib-accel frees up valuable compute resources for business logic while simultaneously boosting compression performance. The library represents an ideal solution for organizations seeking to optimize compression-heavy applications without the complexity and development overhead, adding another valuable option to Intel's acceleration toolkit.

Table of Contents

Introduction

The zlib library is open-source, general-purpose software designed for lossless data compression. It was authored by Jean-loup Gailly and Mark Adler and is widely recognized for its portability, efficiency, and patent-free status. The library is implemented in C and supports almost all hardware platforms and operating systems, providing a compact and fast compression format primarily useful in memory and communication channels.[1, 2, 3, 4]

Formats Supported

There are three main data formats standardized by RFCs issued in 1996 that are used by zlib:

- zlib format (RFC 1950): This is the default wrapper format used by the library for in-memory compression and decompression. It encapsulates Deflate-compressed data with a lightweight header and a checksum for integrity.[2]
- Deflate format (RFC 1951): The core compression algorithm used by zlib, which combines LZ77 sliding window compression with Huffman coding.[3]
- gzip format (RFC 1952): zlib can also read and write the gzip format, commonly used for file compression. Gzip includes more metadata for file handling and uses a CRC32 checksum.[1,4]

These formats enable zlib to work effectively with individual data streams or files, with gzip optimized for file systems and the zlib format suited for network transmission and memory management.[1]

Compression Levels

Zlib provides 10 compression levels, from 0 (no compression) to 9 (maximum compression). Lower levels, such as 1, offer the fastest speed with the lowest compression ratio, while higher levels, such as 9, achieve maximum data reduction at the expense of speed.[1, 5, 6, 7] The default level, usually set at 6, balances efficiency and speed for general use.[6]

Extensive Usage in Software Applications

The widespread adoption of zlib is reflected in the vast number of applications that either directly link to or embed it:

- zlib is integral in major operating systems, file compressing utilities (gzip and ZIP formats), and networking protocols for content encoding.[8]
- It is embedded in programming languages and development libraries such as Python, Ruby, Java, and Node.js, enabling compression features in software easily.[1, 6, 7]
- Many popular file formats, including PNG images, PDF documents, and software installation packages, use zlib for internal data compression.[2, 8]
- Numerous games, embedded devices, and data communication platforms depend on zlib for dependable, high-speed compression.[8]

Compression Ratio Comparison with Other Formats

Since zlib was standardized in 1996, several newer non-Deflate-based compression libraries have been released. Two of the most prominent ones are zstd and LZ4. Each core library algorithm has a unique trade-off between compression ratio and speed.

The performance of zstd, zlib (Deflate), and LZ4 compression algorithms has been extensively analyzed in several studies. Zstandard (zstd) significantly outperforms zlib in both compression ratio and speed. For example, at compression level 1, zstd is approximately 3.4 times faster than zlib at the same level while achieving better compression than zlib at level 9. Additionally, zstd decompresses about 4 times faster than zlib on a single core, making it a highly efficient choice for applications that require inline compression and decompression.[9, 10, 11]

LZ4 is known for its very high speed in both compression and decompression, often outperforming both zstd and zlib. However, this speed advantage comes at the cost of much lower compression ratios compared to zstd and zlib. LZ4 is best suited for scenarios where fast compression and decompression are crucial, such as real-time data processing or high-speed network transfers. Moreover, LZ4 requires lower CPU usage than zstd, which prioritizes compression quality by using more CPU resources.[10, 12]

The zlib library offers moderate compression performance but is generally slower and less effective than both zstd and LZ4. While it achieves a reasonable balance between compression ratio and speed, zstd typically outperforms it in most benchmarks.[9, 10, 11]

Benchmark results on a dataset of approximately 4 GiB illustrate these differences clearly: LZ4 was the fastest, achieving a compression ratio of about 1.13X; zstd achieved about 1.48X with moderate compression time, and zlib was slower, with a compression ratio around 1.43X.[10]

Algorithm	Compression Speed	Decompression Speed
zstd	Moderate to Fast	Fast
zlib	Slow	Moderate
LZ4	Fastest	Fastest

Table 1. Comparison of algorithm speeds.

Algorithm	Compression Ratio	CPU Usage
zstd	High	Moderate
zlib	Moderate to High	High
LZ4	Low to Moderate	Low

Table 2. Comparison of algorithm ratios and CPU utilization.

Overall, zstd is widely recommended as a versatile algorithm that provides an excellent balance of speed and compression ratio; LZ4 is preferred when speed is the top priority; and zlib is typically the least efficient among the three algorithms.[9, 10, 11] These findings are summarized in Tables 1 and 2.

With that said, it's important to remember that zlib adoption is still significantly greater than zstd and LZ4. Moreover, compatibility and maintenance concerns mean that not every application or framework will use a different compression method, even if it offers additional performance. This is why accelerating zlib can still provide value.

Shim Layer

The zlib-accel library provides a shim layer that transparently intercepts zlib compression and decompression calls and offloads them to hardware accelerators when available, enhancing performance without requiring modifications to the application. It allows existing zlib-dependent applications to benefit from compression acceleration easily and with minimal configuration.

Intel IAA is a hardware accelerator built into Intel Xeon Scalable processors. Intel IAA is designed to accelerate database queries and analytics workloads by offloading CPU-intensive tasks, such as compression and decompression, from the CPU. It is optimized for in-memory databases and can reduce memory bandwidth usage, improve throughput, and lower power consumption. Intel IAA is particularly effective for accelerating columnar compression and database query performance.[13, 14]

Intel QAT is another hardware accelerator integrated into Intel processors, designed to accelerate bulk data compression and encryption. While Intel IAA targets database and analytics compression, Intel QAT accelerates storage and networking applications with enhanced data compression and cryptography performance.[15]

The zlib-accel library offers much easier deployment compared to directly integrating compression acceleration plugins into applications. Direct integration often requires modifying application code to incorporate plugin APIs, whereas the shim layer intercepts zlib calls system-wide, providing hardware acceleration transparently without changes to the existing software stack. This makes zlib-accel an accessible option for leveraging Intel hardware acceleration in legacy and new applications alike.

After compiling the shim layer into a shared library, an application's zlib API calls can be intercepted. This can be done by adding LD_PRELOAD=<path_to_shim> before the executable program name on the command line. For further details, consult the zlib-accel GitHub repository online.[16]

The zlib-accel library is designed to work with applications that use the standard zlib API for compression and decompression. The shim layer specifically targets zlib function calls and accelerates the most common compression levels and window sizes. However, certain zlib configurations and advanced features may not be fully supported by the underlying hardware accelerators, in which case the library gracefully falls back to software-based compression to ensure compatibility and correctness.

Performance benefits from zlib-accel are most pronounced in scenarios with large data payloads and sustained compression workloads, where the overhead of hardware-accelerator initialization can be effectively amortized. Applications with very small compression tasks or infrequent compression operations may not see significant performance improvements due to the setup costs of hardware acceleration. Additionally, acceleration availability depends on compatible Intel hardware (Intel IAA or Intel QAT) and a proper system configuration, including appropriate driver installation and device accessibility. The library automatically detects available accelerators and adapts its behavior accordingly, ensuring robust operation across different hardware configurations while maximizing performance where acceleration is available.

In summary, zlib-accel leverages Intel IAA and Intel QAT technologies to provide high-performance, transparent compression acceleration with minimal integration effort, benefiting in-memory databases, analytics workloads, and general compression tasks.

Benchmarking

To evaluate the performance benefits of zlib-accel across real-world applications, comprehensive benchmarking was conducted using three widely deployed database and storage systems: RocksDB, Apache Cassandra, and PostgreSQL. These applications represent diverse compression workloads and usage patterns commonly found in enterprise environments, making them ideal candidates for assessing the practical impact of hardware-accelerated compression.

The zlib-accel library, leveraging Intel IAA and Intel QAT hardware accelerators, was benchmarked against a comprehensive set of compression alternatives to provide a thorough performance comparison. The baseline comparisons included standard zlib (the ubiquitous software implementation), zstd (a modern high-performance compression algorithm widely adopted in storage systems), LZ4 (a fast compression algorithm optimized for speed), and zlib-ng. The zlib-ng library is a high-performance fork of the original zlib library that maintains full API compatibility while incorporating modern CPU optimizations, including vectorized implementations and improved algorithms, to deliver enhanced performance on contemporary processor architectures without requiring hardware accelerators.

To ensure fair and accurate comparisons, datasets were generated individually for each compression algorithm being tested. This approach accounts for the different compression characteristics and optimal data patterns of each algorithm, preventing bias that could arise from using datasets optimized for a single compression method. By generating algorithm-specific datasets, the benchmarking methodology ensures that each compression solution is evaluated under conditions that reflect its intended use case and optimal performance characteristics.

The benchmarking methodology focused on four critical performance indicators that comprehensively assess compression effectiveness and system impact. Throughput measurements capture the volume of data that can be compressed or decompressed per unit time, typically expressed in megabytes per second (MB/s) or gigabytes per second (GB/s), and directly correlate with system capacity and efficiency. Latency measurements assess the time required to complete individual compression or decompression operations, expressed in microseconds or milliseconds, and are crucial for applications that require low response times and real-time processing. Compression ratio measures the effectiveness of each algorithm in reducing data size, expressed as the ratio of the original to the compressed data size, ensuring that performance gains do not compromise storage efficiency. Finally, CPU cycle reduction quantifies the computational savings achieved by offloading compression tasks to dedicated hardware accelerators, measured as the decrease in CPU cycles consumed relative to software-only implementations, directly translating into freed compute resources for application logic. Together, these metrics provide a comprehensive view of compression performance, encompassing bulk data processing efficiency, interactive application responsiveness, storage optimization, and computational resource utilization.

RocksDB

RocksDB is an open-source, embeddable key-value store that uses a log-structured merge-tree (LSM) architecture. It stores data in Sorted String Table (SST) files, which support configurable compression algorithms. The compression affects both storage size and performance, with common methods including zlib and zstd.

Test Setup

- **Hardware:** AWS EC2 instance m7i.metal-24xl with 1 GB/s EBS and 6000 IOPS
- **Software:** Ubuntu 24.04.2 with RocksDB v9.11.0
- **Compression libraries tested:** zlib, zlib-ng (optimized zlib), zstd, QATzip (Intel QAT acceleration), and QPL (Intel IAA acceleration)
- **Compression levels:** Default levels, generally level 6 for zlib-based
- **Workload:** Benchmark tool db_bench running a workload of 80% reads, and 20% writes
- **Block sizes:** 4 KB and 16 KB blocks
- **Key size:** 16 bytes; Value sizes: 32 bytes and 256 bytes
- Block cache was disabled to isolate compression effects
- Roughly 100 GB of input file data when fully uncompressed

Please note that compressed blocks will usually reside in the OS page cache because the system has plenty of memory. This means that filesystem access is avoided in most cases.

Configuration

The zlib-accel shim enables transparent hardware-accelerated compression/decompression by preloading zlib-accel and selecting zlib compression in RocksDB, without any application code changes. It supports Intel IAA (QPL) and Intel QAT acceleration.

Performance Results

Compared to zlib-ng (an optimized software zlib library):

- Intel QAT acceleration increased throughput by 23%-47%
- Intel QAT decreased p99 read latency by 1%-24%
- Compression size tradeoff increased by ~1%-2%

When examining Intel IAA acceleration (using QPL) against the same zlib-ng baseline:

- Throughput gain of 92%-205%
- p99 read latency reduction of 53%-74%
- Compression size tradeoff increase of 5%-9%

Compared to zstd (popular modern software compression), Intel IAA acceleration provided:

- Throughput improvement of 34%-95%
- p99 read latency reduction of 23%-51%
- Compressed size increase of 2%-9%

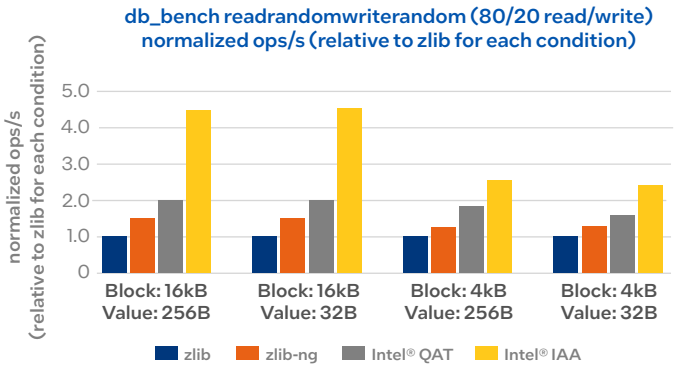

Figure 1. RocksDB throughput with zlib, zlib-ng, Intel QAT, and Intel IAA.

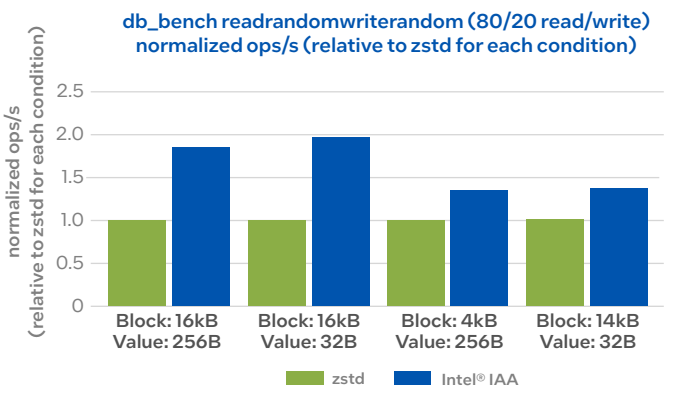

Figure 2. RocksDB throughput with zstd and Intel IAA.

All this data is visualized in the bar charts above. These results show that hardware compression acceleration via zlib-accel can significantly improve RocksDB performance, especially with Intel IAA acceleration, though at the cost of a larger compressed size compared to software zlib or zstd. For applications that can use non-Deflate compression, zstd remains a good alternative, but using hardware-accelerated zlib Deflate offers significant performance gains on systems that support Intel acceleration.

To understand the effect on CPU resources, consider three aspects. First, the average CPU frequency varies by at most 1% across all benchmarks. Second, CPU utilization is almost identical across all cases because the accelerators use busy polling to minimize latency. Third, the runtime is fixed in all scenarios. All of this taken together means that the number of CPU clock cycles is invariant. Therefore, to calculate the approximated normalized clock cycles per operation metric, one only needs to take the inverse of the normalized throughput values shown earlier.

Summary

RocksDB performance is enhanced by leveraging hardware-accelerated compression (zlib-accel) without code changes, delivering substantially higher throughput and lower latency than typical software compression methods, at the cost of a modestly larger compressed data size. Intel IAA provides the highest acceleration benefits in these tests.

Apache Cassandra

Apache Cassandra is an open-source, highly scalable NoSQL database designed to handle massive amounts of data across multiple servers with no single point of failure. It is known for its distributed architecture, which provides high availability and fault tolerance. Cassandra supports software compression libraries like LZ4, zstd, and Deflate (zlib). The zlib-accel shim improves the performance of Deflate compression/decompression operations by leveraging Intel QAT and Intel IAA hardware accelerations, without any Cassandra code changes.

Test Setup

- **Hardware:** Giga Computing R284-A90 with Dual Intel® Xeon® 6980P processor, 1.5 TB DDR5 Memory and 6 x 3.4 TB NVMe direct-attached storage
- **Software:** Ubuntu 24.04.3, OpenJDK 17.0.16, Cassandra 5.0.5 and zlib-accel v1.0.0
- **Compression libraries used:** zlib, zlib-ng (optimized zlib), zstd, QATzip (Intel QAT acceleration), and QPL (Intel IAA acceleration)
- **Compression levels in testing:** default levels for software-based compression, "fast" for LZ4, level 3 for zstd, level 6 for zlib-based; level 9 (max compression) for Intel QAT; level 1 (default) for Intel IAA

Please note that the maximum compression level (9) was chosen for zlib-accel QAT because no significant performance difference was observed between the two levels.

Configuration

Six independent Cassandra servers/databases were created, with compression enabled. All Cassandra servers were running on one socket while the corresponding benchmarks were running on the second socket to isolate the client/server tasks and avoid network bottlenecks. The number of Intel QAT and Intel IAA hardware acceleration engines used was 4, the maximum supported by a CPU socket. The NoSQLBench benchmark was used with the default cql timeseries schema using 80% reads and 20% writes mixture on the initialized database. The benchmark client thread count was increased until the average 99th SLA reached 3ms. The total Cassandra throughput was measured in Ops/sec and recorded at the end of the mix workload run.

Performance Results

The chart in Figure 3 summarizes the normalized throughput to zlib and compression ratio obtained for each compression library.

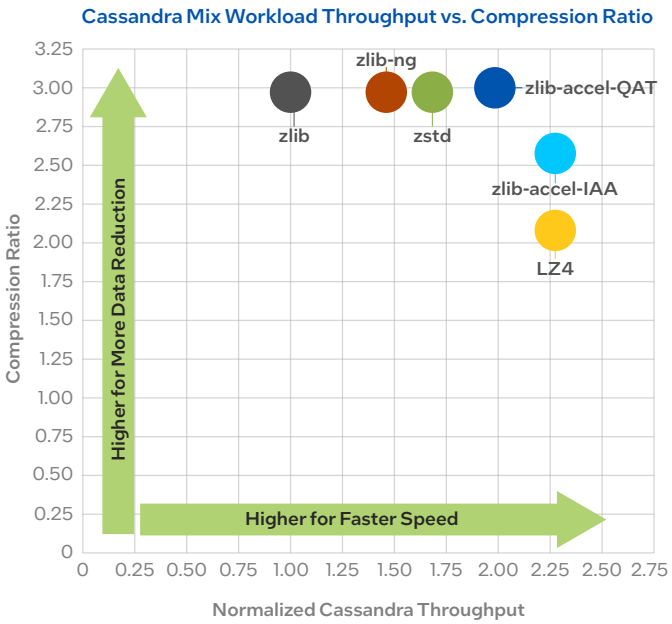

Figure 3. Cassandra throughput vs. compression ratio.

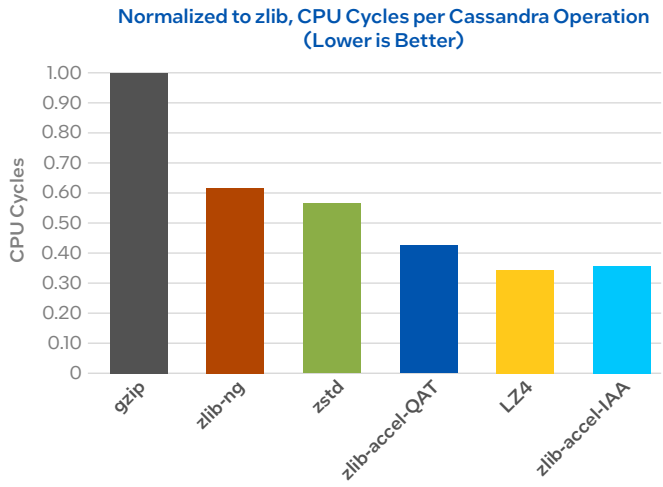

Figure 4. Cassandra CPU cycles per operation.

To prioritize maximum speed, the throughput scale shows LZ4 and zlib-accel-IAA having similar high throughput with less than 1% difference between them. Choosing between the compression libraries may come down to which has better compression ratios. For instance, zlib-accel-IAA has a 24.8% higher compression ratio than LZ4.

To prioritize data reduction, the compression ratio scale shows multiple compression algorithms with the highest compression ratios (between 2.95 and 3.00). These are zstd and Deflate (zlib, zlib-ng, and zlib-accel-QAT). Given similar compression ratios, zlib-accel-QAT is the better choice, as it has the highest throughput, 18% higher than zstd, 98% higher than zlib, and 36% higher than zlib-ng.

Another factor to consider when selecting a compression library is the number of CPU resources it requires. Figure 4 shows the normalized number of CPU cycles per Cassandra Op on a modern Intel Xeon CPU, with lower values indicating better performance. As shown in the figure, zlib consumes the most cycles, which correlates with the lowest throughput. The highest throughput and lowest cycles are both LZ4 and zlib-accel-IAA. If getting the highest compression ratio is a priority, zlib-accel-QAT has the lowest cycles per Cassandra Operation with compression ratios between 2.95 and 3.00.

PostgreSQL

PostgreSQL is a powerful, open-source relational database management system (RDBMS). It's known for its robustness, scalability, and support for advanced data types and performance optimization features.

Test Setup

- **Hardware:** AWS EC2 instance i7i.metal-24xl with 6 x 3.4 TB NVMe direct-attached storage, configured in a 20.4 TB RAID0 array
- **Software:** Ubuntu 24.04.3 with PostgreSQL 17.4 and zlib-accel v1.0.0
- **Compression libraries tested:** zlib, zlib-ng (optimized zlib), zstd, QATzip (Intel QAT acceleration), and QPL (Intel IAA acceleration)
- **Compression levels:** Default levels for software-based compression, generally level 6 for zlib-based; level 9 (max compression) for Intel QAT; level 1 (default) for Intel IAA
- **Workload:** PostgreSQL backup using pg_dump with 4 jobs

Configuration

zlib-accel enables transparent hardware-accelerated compression or decompression by preloading the zlib-accel library and selecting gzip compression in pg_dump, without any application code changes. It supports Intel IAA (QPL) and Intel QAT acceleration. Because the pg_dump backup process compresses data continuously into the archive, the setup cost is easily amortized across the backup operation.

Performance Results

In Figure 5, we show the relative backup throughput of the various algorithms plotted against their compression ratio (uncompressed/compressed size).

Compared to zlib using Deflate compression:

- Intel QAT acceleration increased throughput by up to 29.3X
- Compression size tradeoff of 2.8%
- Intel IAA acceleration (using QPL) increased throughput by up to 23.4X
- Compression size tradeoff of 5.4%

Compared to zlib-ng (an optimized software zlib library):

- Intel QAT acceleration increased throughput by up to 7.6X
- Compression size tradeoff of 3.1%
- Intel IAA acceleration increased throughput by up to 6.1X
- Compression size tradeoff of 5.7%

Compared to zstd, a high-performance software alternative,

- Intel QAT acceleration increased throughput by up to 4.9X
- Compression size decreases marginally over zstd by 0.4%
- Intel IAA acceleration increased throughput by up to 3.9X
- Compression size tradeoff of 2.1%

In Figure 6, we plot the CPU cycle cost of backups for each algorithm. All values are normalized to those measured for gzip.

These results show that hardware-accelerated compression can significantly improve the performance of PostgreSQL backup operations, especially with Intel QAT accelerators, with minimal tradeoffs to compression size. Even against optimized software algorithms like zstd, acceleration of decompression with zlib-accel can offer significant performance improvements with no increase to compressed archive size.

Summary

PostgreSQL backup performance improves by enabling hardware-accelerated compression via zlib-accel, with no code changes. Both Intel QAT and Intel IAA accelerators are shown to improve performance over leading software-based methods with minimal increases in compressed archive size.

Related Materials

Although this whitepaper focuses on transparent zlib acceleration, other approaches are worth mentioning.

The Intel IAA and Intel QAT compression plugins for RocksDB are two examples. These customized components were developed to integrate directly into RocksDB. Currently, they are out of date because a different

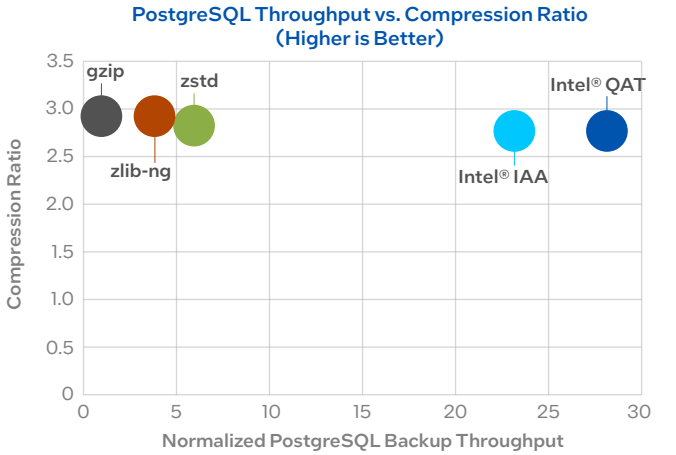

Figure 5. PostgreSQL backup throughput vs. compression ratio.

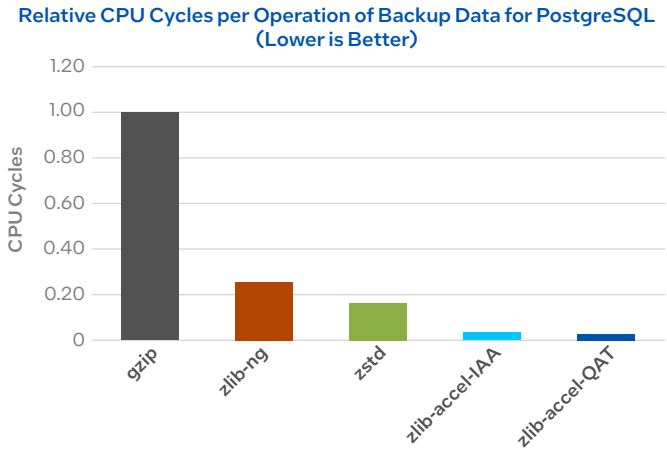

Figure 6. PostgreSQL relative CPU cycles per operation.

pluggable compression framework was introduced into the core RocksDB GitHub repository. However, they do provide a reference point for what level of performance is possible.[17,18]

The Intel QAT Zstandard (zstd) plugin is an external sequence producer added in zstd version 1.5.4 that accelerates compression by offloading sequence production. Integration with Intel QAT maintains API compatibility with existing zstd interfaces, facilitating transparent acceleration for applications through plug-in registration functions. While currently limited to compression (not decompression), future improvements aim to enhance features like dictionary support.[19]

There is also a plugin that integrates Intel IAA with Cassandra. A prior study found higher throughput compared to zstd and a lower compressed data footprint relative to LZ4. It's important to note that results aren't directly comparable with those found in this paper because of differences in hardware, software, and benchmarking methodologies.[20]

Conclusion

The zlib-accel library represents a significant advancement in making hardware compression acceleration accessible to a broad range of applications. By providing a transparent shim layer that requires no code modifications, zlib-accel removes the traditional barriers to adopting hardware acceleration, enabling organizations to immediately benefit from Intel's compression technologies without the complexity and development overhead typically associated with hardware integration. This ease of deployment makes it an ideal solution for both legacy applications and new deployments seeking to optimize compression performance.

Performance evaluations across critical database and storage systems demonstrate that both Intel IAA and Intel QAT deliver substantial improvements over software-based compression implementations. Testing with Apache Cassandra, PostgreSQL, and RocksDB reveals significant improvements in both throughput and latency, with the hardware accelerators consistently outperforming not only standard zlib implementations but also modern, optimized compression algorithms such as zstd and LZ4. These results validate that Intel's compression hardware provides compelling performance advantages while maintaining competitive compression ratios, ensuring that performance gains do not compromise storage efficiency across diverse real-world workloads.

The demonstrated improvements in production-relevant database systems underscore the practical value of zlib-accel in enterprise environments. The combination of transparent integration, substantial performance improvements across multiple application domains, and maintained compression efficiency makes zlib-accel a valuable addition to Intel's acceleration ecosystem. As data center applications continue to demand higher performance and efficiency, the ability to seamlessly leverage dedicated compression hardware becomes increasingly critical. The zlib-accel shim successfully bridges the gap between high-performance hardware capabilities and practical application deployment, enabling organizations to unlock the full potential of their Intel infrastructure while freeing up valuable CPU resources for core business logic and computational tasks.

[1] zlib.net. (2024). zlib 1.3.1 Manual. https://zlib.net/manual.html

[2] Gailly, J., & Adler, M. (1995). RFC 1950 - ZLIB Compressed Data Format Specification version 3.3. https://www.rfc-editor.org/rfc/rfc1950

[3] Gailly, J., & Adler, M. (1995). RFC 1951 - DEFLATE Compressed Data Format Specification version 1.3. https://www.rfc-editor.org/rfc/rfc1951

[4] Gailly, J., & Adler, M. (1995). RFC 1952 - GZIP File Format Specification version 4.3. https://www.rfc-editor.org/rfc/rfc1952

[5] Erlang Documentation Team. (2006). zlib. In Erlang/OTP documentation. https://erlang.org/documentation/doc-5.6/lib/kernel-2.12/doc/html/zlib.html

[6] Python Software Foundation. (2025). zlib — Compression compatible with gzip. https://docs.python.org/3/library/zlib.html

[7] Node.js Documentation. (2025). zlib. https://nodejs.org/api/zlib.html

[8] Roelofs, G., Gailly, J.-l., & Adler, M. (n.d.). zlib applications. zlib.net. https://zlib.net/apps.html

[9] Databento Blog. (2024, November 11). Zstd vs. zlib: Market data compression. https://databento.com/blog/zstd-vs-zlib

[10] Jain, M. (2022, December 5). Comparing compression algorithms for moving big data. https://manishrjain.com/compression-algo-moving-data

[11] Szorc, G. (2017, March 7). Better compression with Zstandard. Gregory Szorc's Digital Home. https://gregoryszorc.com/blog/2017/03/07/better-compression-with-zstandard/

[12] Facebook Engineering. (2016, August 31). Smaller and faster data compression with Zstandard. https://engineering.fb.com/2016/08/31/core-infra/smaller-and-faster-data-compression-with-zstandard/

[13] Intel Corporation. (2024, April 18). Intel® In-Memory Analytics Accelerator (Intel® IAA). https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/in-memory-analytics-accelerator.html

[14] Introduction — Intel® QPL v1.8.0 Documentation. (n.d.). https://intel.github.io/qpl/documentation/introduction_docs/introduction.html

[15] Intel Corporation. (2024, November 27). What is Intel® QuickAssist Technology (Intel® QAT)? Intel. https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/what-is-intel-qat.html

[16] intel/zlib-accel - GitHub. (2025, April 7). https://github.com/intel/zlib-accel

[17] Ravindran, B., Giacchino, L., Jee, K., & Babu, N. (2023, September 14). Intel® In-Memory Analytics Accelerator Plugin for RocksDB Storage Engine (Intel® IAA Plugin for RocksDB Storage Engine). https://www.intel.com/content/www/us/en/content-details/788607/intel-in-memory-analytics-accelerator-plugin-for-rocksdb-storage-engine-intel-iaa-plugin-for-rocksdb-storage-engine.html

[18] Issa, M., Giacchino, L., Kumar, S., Qian, D., & Will, B. (2024, January 4). Accelerating RocksDB Compression with Intel® QuickAssist Technology. https://www.intel.com/content/www/us/en/content-details/812751/accelerating-rocksdb-compression-with-intel-quickassist-technology.html

[19] Will, B., Qian, D., Khade, A., & Schuetze, J. (2023, August 16). Intel® QuickAssist Technology Zstandard Plugin, an External Sequence Producer for Zstandard. Intel AI Community. https://community.intel.com/t5/Blogs/Tech-Innovation/Artificial-Intelligence-AI/Intel-QuickAssist-Technology-Zstandard-Plugin-an-External/post/1509818

[20] Ravindran, B., Kokoori, S., Jee, K., Babu, N., Reddy, S., & Ruiz, T. (2024, June 14). Apache Cassandra with Intel® In-Memory Analytics Accelerator. https://www.intel.com/content/www/us/en/content-details/825354/apache-cassandra-with-intel-in-memory-analytics-accelerator-intel-iaa.html