A Comparative Analysis of DEFLATE and Snappy for E-Commerce Data Compression

Team Members:

Pradeep.M – 22MID0152

Dharanishvar.R – 22MID0170

Mohanakrishnan.J – 22MID0175

Abstract

Context: Modern e-commerce platforms are defined by their capacity to process high-velocity, massive-scale data. This data, which includes real-time user session logs, large JSON/CSV product catalogs, and high-volume transaction histories, must be stored and transmitted efficiently. While data compression is a standard solution for reducing storage and network costs, it introduces a critical trade-off: CPU overhead vs. compression ratio. In a latency-sensitive e-commerce environment, where site responsiveness is directly tied to user retention and revenue, a slow compression algorithm can become a critical performance bottleneck.

Problem: The incumbent, traditional compression standard is DEFLATE (the algorithm powering zlib, gzip, and .zip files). DEFLATE is optimized for maximum compression ratio, not speed. It achieves this by employing a computationally "heavy" two-stage process: first, a "greedy" LZ77-based search for optimal string matching, followed by a complex Huffman entropy coding stage that rewrites the data as a dense, CPU-intensive bit-stream. This process, while creating very small files, introduces significant latency, making it unsuitable for real-time databases, in-memory caches, or live data pipelines where performance is paramount.

Proposed Solution: This project proposes and evaluates Snappy, a modern compression library from Google, as a high-speed alternative specifically engineered for this "speed-first" workload. Snappy's design philosophy fundamentally inverts the traditional trade-off. It is a one-stage, byte-oriented algorithm that implements a highly optimized, "good-enough" LZ77-style search. Crucially, Snappy intentionally skips the entire Huffman coding stage, outputting a simple, CPU-friendly byte-stream. This design allows for "good-enough" compression at a fraction of the computational cost, prioritizing raw throughput and low latency.

Methodology: To quantify this trade-off, we conducted a direct comparative benchmark. The DEFLATE algorithm was implemented in Python (using the zlib library at its maximum setting, level=9), and the

Snappy algorithm was implemented in C++ (using the snappy library). Both implementations were benchmarked on a large-scale, multi-gigabyte CSV dataset, representing a typical e-commerce product catalog. Key performance metrics—compression time, decompression time, and final compression ratio—were recorded and analyzed.

Results: The benchmark results were definitive. DEFLATE achieved a superior compression ratio of 3.10:1, proving its effectiveness at minimizing file size. In stark contrast, Snappy achieved a more moderate 2.13:1 ratio. However, this came at a severe and predictable performance cost for DEFLATE, which took 11.41 seconds to compress the file. Snappy completed the same compression task in just 0.67 seconds, making it 17 times faster. The decompression speeds showed a similar result, proving Snappy's clear advantage in high-speed, read-write scenarios.

Conclusion: For a latency-sensitive e-commerce environment, DEFLATE's superior compression ratio is a poor trade-off for its critical performance bottleneck. The 17x speed advantage of Snappy demonstrates its suitability for real-time systems. The minor cost in additional storage is a negligible price to pay for the significant reduction in CPU load and the critical gain in user-facing responsiveness. We therefore recommend Snappy as the superior choice for compressing data in our platform's real-time caches, databases, and log-shipping pipelines.