Dictionary Codec

Yunhua Fang

November 2024

1 Introduction

Dictionary encoding has become a cornerstone in contemporary data analytics systems, particularly for compressing datasets with relatively low cardinality. This technique operates by first scanning the data to identify all unique elements, which are then compiled into a "dictionary." Each original data item is subsequently replaced with a corresponding dictionary identifier (ID), effectively reducing the overall data footprint. To facilitate rapid retrieval and efficient management of the dictionary, sophisticated indexing data structures such as hash tables or B-trees are often employed. These structures ensure that look-up operations remain swift, even as the size of the dictionary grows. Beyond mere compression, dictionary encoding significantly enhances the performance of scan and search operations by enabling the utilization of Single Instruction, Multiple Data (SIMD) instructions. SIMD allows for parallel processing of multiple data points in a single operation, thereby accelerating computational tasks. Additionally, the compact representation of data through dictionary encoding not only minimizes storage requirements but also optimizes cache utilization, further boosting system efficiency. This combination of reduced data size and accelerated processing makes dictionary encoding an invaluable tool for real-time analytics, large-scale data warehousing, and various applications that demand both high performance and efficient storage.

In this report, we will examine the effectiveness of dictionary encoding and assess the efficiency of its query mechanisms compared to a traditional (vanilla) baseline approach. Our investigation will focus on two primary areas:

Dictionary Encoding Performance: We will evaluate how dictionary encoding compresses data with low cardinality and accelerates scan and search operations. To optimize the construction of the dictionary, our implementation

supports multi-threading, allowing users to specify the number of threads based on their system's capabilities. This multithreaded approach aims to efficiently build the dictionary, reducing the time required for data compression.

Query Mechanism Efficiency: We will analyze the performance of query operations when utilizing dictionary encoding. Specifically, we will compare queries executed with Single Instruction, Multiple Data (SIMD) instructions against those without SIMD support. SIMD enables parallel processing of multiple data points, potentially speeding up query execution significantly.

To provide a comprehensive evaluation, we will conduct experiments that benchmark the performance of dictionary encoding and its query mechanisms against the vanilla baseline. Metrics such as compression ratio, query execution time, and resource utilization will be measured and analyzed. Additionally, we will explore how varying the number of threads affects the dictionary construction process and how SIMD integration influences query performance.

Through this analysis, the report aims to demonstrate the advantages of dictionary encoding in reducing data footprint and enhancing query efficiency. The findings will offer valuable insights into the suitability of dictionary encoding for real-world data analytics systems, highlighting its potential to improve both storage efficiency and computational performance.

2 Performance

The raw data we use is a 1GB text file. Our self-designed dictionary encoding algorithm performs the encoding process under multi-threading. For search and scan, we have 3 algorithms for comparison: Query with SIMD, Query without SIMD, and vanilla baseline.

In our research, we systematically evaluate the performance of dictionary encoding and its query mechanisms by recording and analyzing various metrics across three key areas. First, Encoding Speed Performance is assessed by running our encoding program with different thread counts—1, 2, 4, and 8—and comparing the latency to demonstrate the efficiency gains from multithreading. Second, Single Data Search Performance involves benchmarking the speed of three distinct search algorithms, including our optimized query

method and a traditional vanilla search, to highlight the performance differences and the advantages of dictionary encoding over the baseline. Lastly, Prefix Scan Performance examines how these three algorithms handle prefix scan operations, revealing the relative strengths and weaknesses in more complex search scenarios. By meticulously measuring encoding latency, search speeds, and prefix scan efficiency under varying conditions, our study provides comprehensive insights into how multi-threading and SIMD instructions enhance dictionary encoding. This holistic analysis underscores the benefits of dictionary encoding in reducing data footprint and accelerating query operations, offering valuable guidance for implementing efficient data compression and retrieval strategies in real-world analytics systems.

2.1 Encoding Speed Performance

The dictionary encoding implementation effectively harnesses the power of multi-threading to accelerate the encoding of large datasets, such as a 1GB text file. By distributing the workload across multiple threads, the encoding process is partitioned into concurrent tasks, each handling a distinct subset of the raw data. This parallelization allows for the simultaneous identification and mapping of unique strings to their corresponding unique identifiers, significantly reducing the overall processing time. Specifically, the encode_from_memory function in the DictionaryEncoder class initializes multiple threads based on the available hardware concurrency, dividing the input data into equal chunks that each thread processes independently. Each thread scans its assigned data segment, extracts unique strings, and updates a thread-local dictionary to avoid contention and ensure thread safety. Once all threads complete their execution, the local dictionaries are merged into a global dictionary, consolidating the unique string mappings. This approach not only maximizes CPU utilization by keeping all cores active but also minimizes bottlenecks associated with single-threaded execution, thereby facilitating a swift and efficient encoding process. The seamless integration of multi-threading within the dictionary encoding framework ensures that large-scale data can be processed rapidly, laying a robust foundation for subsequent query operations.

Thread Num	1	2	4	8
Latency / s	68.74	32.33	27.44	24.52

Table 1: Speed Performance of Dictionary Encoding

2.2 Single Data Search Performance & Prefix Scan Performance

SIMD (Single Instruction, Multiple Data) optimizations are strategically integrated into the query processing component to enhance performance and efficiency, particularly during prefix search operations. By leveraging SIMD instructions, such as those provided by the SSE4.2 instruction set, the system can perform parallel comparisons of multiple characters simultaneously, significantly accelerating the matching process. In the prefix search functionality, SIMD enables the simultaneous loading and comparison of prefix segments from both the query string and the dataset entries. This parallelism reduces the number of required instructions and minimizes latency, especially when dealing with large volumes of data. Additionally, the implementation includes a fallback to scalar operations for cases where SIMD is either not supported or the prefix length exceeds the optimal threshold for SIMD processing. This dual-path approach ensures both high performance through SIMD where applicable and reliable functionality through traditional methods otherwise. Consequently, the utilization of SIMD within the query engine not only accelerates search operations but also contributes to the overall scalability and responsiveness of the system when handling extensive datasets.

Search	Query + SIMD	Query	Vanilla
Latency / s	0	0	0.086792

Table 2: Speed Performance of Search

Scan	Query + SIMD	Query	Vanilla
Latency / s	0.0248743	0.050126	0.0654666

Table 3: Speed Performance of Prefix Scan

3 Discussion

The speed performance of dictionary encoding across varying thread counts demonstrates a clear trend of reduced latency with increased parallelism. Specifically, utilizing a single thread results in a latency of 68.74 seconds, which significantly decreases to 32.33 seconds when doubled to two threads. Further scaling to four threads continues this improvement, albeit at a reduced rate, bringing latency down to 27.44 seconds. Extending the thread

count to eight yields a marginal reduction, achieving a latency of 24.52 seconds. This diminishing return on performance enhancement can be attributed to several factors inherent in multi-threaded processing. Initially, the distribution of tasks across multiple threads effectively leverages the computational resources, thereby halving the processing time when moving from one to two threads. However, as the number of threads increases, the benefits become less pronounced due to overhead associated with thread management, such as context switching and synchronization delays. Additionally, certain portions of the encoding algorithm may not be fully parallelizable, limiting the extent to which performance can scale with additional threads. Consequently, while multi-threading substantially accelerates the dictionary encoding process by utilizing available CPU cores more efficiently, the incremental gains diminish beyond a certain threshold. This analysis underscores the importance of optimizing both parallel and serial components of the encoding process to maximize scalability and performance in multi-threaded environments.

The speed performance results for exact query searches reveal a notable disparity between the different query strategies employed. Specifically, both the "Query + SIMD" and "Query" methods report a latency of 0 seconds, whereas the "Vanilla" approach exhibits a measurable latency of 0.086792 seconds. We try to use ns as unit to measure the latency of Query + SIMD and Query, but we still cannot get the difference between them. A possible reason the latency is too small is that we did not count the time of loading encoded columns. When searching on loaded encoded columns with the Query, the speed will be really fast so the timer cannot present the difference. We would affirm the substantial performance gains achieved through optimization. Alternatively, this result could indicate a potential issue in the timing implementation, such as not accurately capturing the start and end times of the query operations.

Comparatively, the "Vanilla" method serves as a baseline, demonstrating a non-trivial latency of approximately 0.086792 seconds. This latency is expected as the vanilla approach likely involves a straightforward, unoptimized search mechanism that processes each entry sequentially without leveraging advanced optimizations. In contrast, the "Query" and "Query + SIMD" strategies are designed to enhance performance through dictionary encoding and SIMD (Single Instruction, Multiple Data) optimizations. The dictionary encoding reduces the search space by mapping unique strings to unique identifiers, thereby minimizing redundant comparisons. When combined with SIMD, the search operations can process multiple data points in parallel, sig-

nificantly accelerating the matching process.

The performance results for the prefix scan operations distinctly illustrate the efficacy of various query strategies in optimizing search latency. Notably, the "Query + SIMD" approach achieves the lowest latency at 0.0248743 seconds, significantly outperforming both the "Query" method with 0.050126 seconds and the "Vanilla" baseline at 0.0654666 seconds. This substantial reduction in latency can be primarily attributed to the synergistic combination of dictionary encoding and SIMD (Single Instruction, Multiple Data) optimizations. The "Query + SIMD" strategy leverages SIMD instructions to execute multiple data comparisons in parallel, thereby accelerating the prefix matching process and minimizing the time required to traverse and evaluate large datasets. In contrast, the "Query" method, while still benefiting from dictionary encoding, does not utilize SIMD, resulting in longer processing times due to sequential data handling. The "Vanilla" approach, devoid of both dictionary encoding and SIMD optimizations, relies on straightforward, unoptimized sequential scans, leading to the highest latency among the three methods. The observed performance improvements underscore the critical role of SIMD in enhancing computational efficiency and reducing search times, especially in operations involving extensive data comparisons such as prefix scans. Additionally, the progressive decrease in latency from "Vanilla" to "Query" and further to "Query + SIMD" highlights the cumulative benefits of integrating multi-threading with advanced vectorization techniques. These findings demonstrate that adopting optimized query strategies, particularly those utilizing SIMD, can significantly enhance the responsiveness and scalability of search operations within large-scale data processing environments.

4 Conclusion

The comprehensive performance evaluations conducted on the dictionary encoding and query operations elucidate the substantial benefits of employing multi-threading and SIMD (Single Instruction, Multiple Data) optimizations within large-scale data processing systems.

Firstly, the dictionary encoding process demonstrated a clear reduction in latency as the number of threads increased from one to eight. Specifically, latency decreased from 68.74 seconds with a single thread to 24.52 seconds with eight threads. This significant improvement underscores the effective-

ness of multi-threading in leveraging available CPU cores to parallelize the encoding workload, thereby enhancing overall processing speed. However, the observed diminishing returns beyond four threads suggest that factors such as thread management overhead and resource contention begin to limit further performance gains. Nonetheless, the ability to scale encoding performance through multi-threading is pivotal for handling extensive datasets efficiently.

In the realm of exact query searches, the optimized strategies utilizing both dictionary encoding and SIMD optimizations exhibited markedly superior performance compared to the vanilla approach. While the "Vanilla" method recorded a latency of 0.086792 seconds, both the "Query + SIMD" and "Query" methods reported negligible latencies of 0 seconds. This discrepancy likely arises from the high efficiency of the optimized methods in processing exact matches, potentially coupled with the limitations of the timing mechanism used. The optimized approaches benefit from reduced search spaces through dictionary encoding and accelerated comparison operations via SIMD, enabling rapid retrieval of query results. The stark contrast in performance highlights the critical role of advanced optimization techniques in minimizing search latencies.

Similarly, the prefix scan operations further validated the advantages of combining multi-threading with SIMD optimizations. The "Query + SIMD" strategy achieved the lowest latency of 0.0248743 seconds, outperforming the "Query" method at 0.050126 seconds and the "Vanilla" approach at 0.0654666 seconds. This progression reflects the enhanced capability of SIMD to perform parallel comparisons, thereby expediting the identification of matching prefixes within large datasets. The incremental performance improvements from "Vanilla" to "Query" and further to "Query + SIMD" illustrate the cumulative impact of optimizing both the algorithmic structure and the underlying hardware utilization.

In conclusion, the integration of multi-threading and SIMD optimizations significantly elevates the performance of dictionary encoding and query operations in data-intensive applications. Multi-threading effectively harnesses the computational power of multi-core processors to expedite encoding processes, while SIMD accelerates query executions by enabling parallel data processing. These optimizations not only reduce latency but also enhance the scalability and responsiveness of the system when managing vast datasets. Moving forward, further refinements in parallel processing techniques and more precise timing mechanisms will continue to unlock additional perfor-

mance gains, solidifying the foundation for efficient large-scale data management and retrieval.