HocoPG: A Database System with Homomorphic Compression for Text Processing

Jiawei Guan, Feng Zhang, Yuxin Tang, Weitang Ye, Xiaoyong Du
Key laboratory of Data Engineering and Knowledge Engineering (MOE), and School of Information,
Renmin University of China
{ guanjw, fengzhang, 2020201596, 2021201613, duyong }@ruc.edu.cn

ABSTRACT

Databases employ out-of-line storage and compression strategies to manage extensive text data. However, the growth in both the size of individual data items and overall data volume has significantly increased the burden of decompression, adversely affecting query performance. To address this challenge, we develop HocoPG, an innovative system that incorporates homomorphic compression theory within RDBMS, enabling direct computation on compressed data to enhance query efficiency and system usability. HocoPG performs homomorphic evaluations across a suite of basic text operations, enabling the execution of intricate text queries by combining these operations flexibly. In this demonstration, we showcase the deployment and usage of HocoPG through a database terminal. Additionally, we introduce the HocoPG Admin, a tool that provides insights to DBAs and general users for choosing the optimal compression scheme based on their individual needs.

PVLDB Reference Format:
Jiawei Guan, Feng Zhang, Yuxin Tang, Weitang Ye, Xiaoyong Du . HocoPG: A Database System with Homomorphic Compression for Text Processing. PVLDB, 17(12): XXX-XXX, 2024.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Guan-JW/HocoPG-Demo.

1 INTRODUCTION

Databases are crucial for managing, storing, and analyzing extensive text data, providing scalable solutions for data access and organization [8]. The adoption of *out-of-line disk storage* [5, 7] is significant for handling large objects without embedding them directly in table structures. Instead, databases store references to the data, with actual contents located in distinct disk storage areas. For example, PostgreSQL employs TOAST (The Oversized-Attribute Storage Technique) to manage large values in a specialized tablespace efficiently. This strategy effectively mitigates main table bloat and maintains query performance.

Databases commonly employ *compression* techniques alongside out-of-line storage to optimize storage and performance [1, 2]. Prior to the persistence of text data on disk, they utilize general-purpose compression algorithms such as Gzip, LZ4, or Zstd [6] to minimize data volume. This compression significantly reduces disk space requirements, thus speeding up I/O operations and improving overall database performance.

Proceedings of the VLDB Endowment, Vol. 17, No. 12 ISSN 2150-8097.
doi:XX.XX/XXX.XX

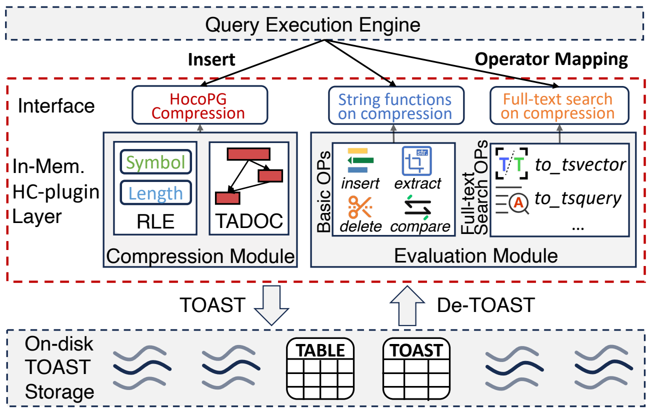


Figure 1: The HocoPG Architecture.

Despite achieving significant compression ratios, general-purpose compression algorithms come with notable drawbacks. Specifically, these algorithms are known to incur considerable computational overheads during (de)compression cycles. These inefficiencies can slow down data processing and potentially delay result retrieval, as compressed data must first be decompressed.

In this paper, we present HocoPG (shown in Figure 1), a database system leveraging homomorphic compression theory [4] for the efficient management of large-scale text objects within RDBMS. HocoPG significantly enhances processing efficiency, surpassing traditional text query performance by 2.86× and achieving up to 2.48× storage savings. It streamlines the execution of a wide range of text operations by translating standard operations into homomorphic equivalents, facilitating computation on compressed data.

At the algorithm level, HocoPG introduces two homomorphic compression schemes (RLE and TADOC), each with different compression ratios and processing capabilities. These schemes are adept at performing a variety of operations on large compressed texts, including basic string manipulations and full-text search functionalities. At the system level, HocoPG enhances traditional RDBMS by integrating a homomorphic compression plugin (HC-plugin) and a Database Administrator (DBA) analysis tool. The HC-plugin serves as an intermediate layer, interfacing between the database execution engine and the disk storage units. The DBA tool aids DBAs and developers in choosing compression schemes, facilitating more efficient data management and maintenance within database systems. Through modular design, HocoPG exhibits portability, facilitating its adaptation to various database systems. Moreover, it is inherently extensible, allowing users to incorporate additional homomorphic compression schemes.

We demonstrate HocoPG via two approaches: 1) using the database *terminal*, and 2) deploying an interactive DBA analysis platform, *HocoPG Admin*, designed for managing large text objects. In the database terminal, users engage HocoPG features by executing standard SQL queries with default or additional HocoPG compression options. HocoPG is then responsible for the entire lifecycle of the data, including its compression, decompression, storage, and processing. HocoPG Admin provides an enhanced interactive experience, allowing DBAs or general users to interact with the database through a user-friendly interface. It emphasizes the out-of-line storage format of tables, providing users with a detailed perspective on their storage arrangements. Additionally, it conducts comparative analyses of various compression mechanisms using data samples, demonstrating the efficiency and effectiveness of different compression options. These analyses assist users in identifying the most suitable compression algorithm for their specific needs, thus optimizing their data management strategy.

The demonstrations of HocoPG underscore the efficacy of homomorphic compression technology in improving the efficiency of storage and processing for extensive text data.

2 THEORETICAL BASIS

Homomorphic compression theory [4] provides a theoretical basis for direct computation on compressed data. Specifically, homomorphic compression (HC) is defined as a mapping that follows:

$$\varphi\big[\sigma_u(u_1,\ldots,u_n)\big] = \sigma_c\big(\varphi(u_1),\ldots,\varphi(u_n)\big),$$

where $u_1,\ldots,u_n \in \mathbb{U}$ represent uncompressed texts, $\sigma_u \in \Pi$ and $\sigma_c \in \Theta$ denote the basic operations for uncompressed and compressed text, respectively.

Homomorphic compression theory distinguishes compression algorithms through key properties like *directness* and *strong homomorphism*, which reflect an algorithm's capacity for direct computation. These properties determine the scope and frequency of operations an algorithm supports. Utilizing this theoretical foundation, one can develop a homomorphic compression scheme (HC scheme) that includes algorithms for compression, decompression, and evaluation, enabling efficient operations on compressed data. Empirical evidence suggests that deploying HC schemes in text data management significantly enhances text processing efficiency.

3 HOCOPG OVERVIEW

HocoPG is a database system designed to enhance the management and processing of large objects through the application of homomorphic compression techniques. This section presents its overall architecture and key features.

3.1 Architecture

Figure 1 shows the architecture of HocoPG, highlighting the integration of the HC-plugin within RDBMS, employing PostgreSQL as the base system.

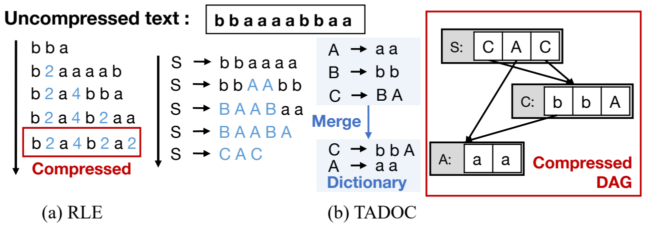


Figure 2: Compression Algorithms in the HC-Plugin.

In-Memory HC-Plugin Layer. The HC-plugin is constituted by two principal components: the *compression module* (detailed in Sec. 3.2) and the *evaluation module* (described in Sec. 3.3). The compression module activates upon the insertion of text data, applying designated compression algorithms to detect patterns and reduce data size. For text query execution, HocoPG efficiently translates operations for uncompressed text into those compatible with compressed data, utilizing its operator mapping feature. The evaluation module supports an extensive array of operations on text data stored within the database, encompassing both string functions and full-text search capabilities targeted on compressed data.

On-disk TOAST Storage (Sec. 3.4). HocoPG functions autonomously, without affecting the native out-of-line storage mechanisms of the database. The compression process yields text data that is seamlessly integrated into TOAST tables through the existing TOAST mechanism as if it were standard uncompressed data. This approach significantly reduces the overhead typically associated with TOAST operations, thereby improving the efficiency of queries. We next introduce the key features and implementation details of HocoPG.

3.2 Compression Module

HC Schemes. Motivated by homomorphic compression, HocoPG incorporates two HC schemes, RLE [3] and TADOC [9], into its compression module. The workflows of the two compression algorithms are depicted in Figure 2. Specifically, RLE operates by detecting and counting sequential repetitions of characters, whereas TADOC employs a method of replacing recurrent character sequences with dictionary references to form a directed acyclic graph (DAG).

Properties. Within the theoretical framework of homomorphic compression [4], RLE is characterized as a fully homomorphic compression scheme (FHC), allowing for both unrestricted direct access to and updates on compressed data. In contrast, TADOC is identified as a partially homomorphic compression scheme (PHC), which permits arbitrary direct access to compressed data, albeit with constraints on the updates that can be performed.

System Configurations. Within the configuration of HocoPG, initial parameters for the HC-plugin are established as system variables, enabling homomorphic compression and evaluation by default. Following the activation of HocoPG, the system seamlessly incorporates the two HC schemes as elective attributes for the `TEXT` property during the processes of `TABLE CREATE` and `UPDATE`. This integration allows for the tailored application of compression strategies in accordance with the specific operational requirements. To gain insights into the effectiveness of different HC schemes, DBAs and users are encouraged to utilize our HocoPG Admin (see Sec. 4.2).

3.3 Evaluation Module

Text Operations in RDBMS. Operations on large text objects in RDBMS can be broadly classified into two categories: string functions and full-text search.

- **String Functions**: This category encompasses a suite of operators dedicated to the examination and modification of string values. Key operations include string matching and replacement, which are vital for precise text data manipulation and analysis.
- **Full-text Search**: This functionality offers advanced search capabilities, such as stemming analysis, to identify word roots and word occurrences. This enables users to conduct intricate searches across large text collections.

Enhancing Text Manipulation with HocoPG. HocoPG systematically deconstructs text operations within databases into four main functions: `extract`, `insert`, `delete`, and `compare`, supplemented by tokenization functions `to_tsvector` and `to_tsquery`. This set of basic functions provides a foundation for conducting complex text processing on compressed data.

HocoPG enables the direct execution of all basic operations on RLE- and TADOC-compressed data, ensuring data integrity and processing efficiency. To further boost query performance, HocoPG incorporates several optimization strategies. These include aligning offsets between uncompressed and compressed data and reusing intermediate results in compressed patterns, such as dictionary entries. These optimizations significantly reduce I/O overhead and enhance processing speed.

Executing Operations on Compressed Text. During query processing, the execution engine decomposes queries into their constituent operations. HocoPG routes operations pertinent to large text objects through its operator mapping layer. This layer translates operations into calls to specific APIs within the evaluation module, thus allowing for the direct manipulation of compressed data. This ensures a seamless and optimized user experience.

3.4 On-Disk TOAST Storage

TOAST is a critical feature in PostgreSQL designed for handling large data objects via out-of-line storage. It works by breaking down large objects into smaller chunks and compressing them for optimized storage, thus reducing storage and transmission costs.

Secondary Compression with TOAST. To enhance the TOAST mechanism with advanced compression capabilities, we develop the HC-plugin in HocoPG as an in-memory layer that interfaces with TOAST storage on disk. HocoPG applies sophisticated compression techniques at the time of data insertion, leveraging pattern recognition in data for initial compression. This pre-compressed data is subsequently managed and stored by TOAST, which may apply additional compression based on its configuration.

Empirical analyses reveal that HocoPG significantly improves data loading efficiency and reduces storage demands, regardless of TOAST's secondary compression actions. Specifically, under TOAST's *plain* or *external* settings, which bypass additional compression, HocoPG's pre-compression can diminish data size, leading to reduced bandwidth needs and faster disk writes. When TOAST is configured to *extended* or *main* modes, which activate PGLZ or LZ4 compression, applying a secondary compression to HocoPG-pre-compressed data further boosts efficiency in both space and time. Moreover, the decreased data volume needed for reconstitution in the De-TOAST process, combined with the ability to run queries directly on compressed data, further enhances query efficiency.

By operating independently from the underlying storage mechanisms, HocoPG treats its data as standard input to TOAST for possible secondary processing. This design enables HocoPG to support a wide array of compression algorithms and functionalities without disrupting the database's storage infrastructure.

4 DEMONSTRATION

The demonstration consists of two parts: 1) Executing text queries using HocoPG within a database *terminal*; 2) Presenting text management results and insights using *HocoPG Admin*. We employ Pizza&Chili Corpus, a benchmark commonly used for evaluating compression algorithms and compressed data structures, for showcasing HocoPG's capabilities.

4.1 Scenario 1: Database Terminal

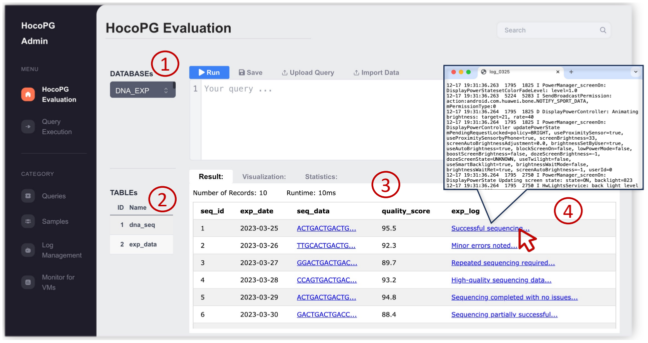
HocoPG extends the `TEXT` data type in RDBMS, integrating support for RLE and TADOC schemes and enabling basic operations on compressed data. To illustrate HocoPG's ability to handle both symbolic sequences and semantic data, we employ the DNA sequencing dataset for demonstration. Users can manage their text data using HocoPG through the following steps in a database terminal:

Step1: HocoPG Setup (Optional). HocoPG is activated by default via environmental variables configured in the system, initially employing the lightweight RLE scheme for handling `TEXT` data types. Users, however, retain the autonomy to tailor the management of their text data according to specific needs. As shown in the subsequent code snippet, it is feasible to incorporate HC scheme options for `TEXT` fields when creating new database tables. In the provided dna_seq table, the seq_data field is set up to use RLE for efficient processing of repetitive sequence data, while the exp_log field is configured to apply TADOC to exploit textual semantic redundancy.

```
-- Create table with HocoPG options.
CREATE TABLE dna_seq (seq_id SERIAL PRIMARY KEY,
               exp_date DATE NOT NULL,
               seq_data TEXT COMPRESSION RLE,
               exp_log TEXT COMPRESSION TADOC);
```

Step2: Query Execution. We proceed with text processing on table dna_seq. The code snippets below showcase data loading, substring extraction and replacement within the seq_data field, and full-text search on the exp_log field. Note that HocoPG allows unrestricted text manipulations. Upon insertion into the table, HocoPG automatically compresses text data using predefined HC schemes. All subsequent query operations are performed directly on this compressed data.

```
-- 1. Load data into table
INSERT INTO dna_seq VALUES ('2024-03-31',
    'ACTGACTGACTG...', pg_read_file('log_0331.txt'));
-- 2. Substring extraction
SELECT SUBSTRING(seq_data from 1 for 10) FROM dna_seq;
```



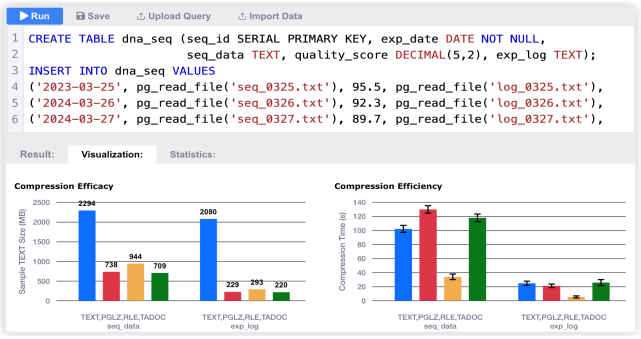
ID	Name
1	dna_seq
2	exp_data

Number of Records: 10 Runtime: 10ms

seq_id	exp_date	seq_data	quality_score	exp_log
1	2023-03-25	ACTGACTGACTG...	95.5	Successful sequencing...
2	2023-03-26	TTGCACTGACTG...	92.3	Minor errors noted...
3	2023-03-27	GGACTGACTGAC...	89.7	Repeated sequencing required...
4	2023-03-28	CCAGTGACTGAC...	93.2	High-quality sequencing data...
5	2023-03-29	ACTGACTGACTG...	94.8	Sequencing completed with no issues...
6	2023-03-30	GACTGACTGACC...	88.4	Sequencing partially successful...

```
12-17 19:31:36.263  1795  1825 I PowerManager_screenOn:
DisplayPowerStatesetColorFadeLevel: level=1.0
12-17 19:31:36.263  5224  5283 I SendBroadcastPermission:
action:android.com.huawei.bone.NOTIFY_SPORT_DATA,
mPermissionType:0
12-17 19:31:36.264  1795  1825 D DisplayPowerController: Animating
brightness: target=21, rate=40
12-17 19:31:36.264  1795  1825 I PowerManager_screenOn:
DisplayPowerController updatePowerState
mPendingRequestLocked=policy=BRIGHT, useProximitySensor=true,
useProximitySensorbyPhone=true, screenBrightness=33,
screenAutoBrightnessAdjustment=0.0, brightnessSetByUser=true,
useAutoBrightness=true, blockScreenOn=false, lowPowerMode=false,
boostScreenBrightness=false, dozeScreenBrightness=-1,
dozeScreenState=UNKNOWN, useTwilight=false,
useSmartBacklight=true, brightnessWaitMode=false,
brightnessWaitRet=true, screenAutoBrightness=-1, userId=0
12-17 19:31:36.264  1795  2750 I PowerManager_screenOn:
DisplayPowerState Updating screen state: state=ON, backlight=823
12-17 19:31:36.264  1795  2750 I HwLightsService: back light level
```

Figure 3: Out-of-Line Table Visualization.



```
CREATE TABLE dna_seq (seq_id SERIAL PRIMARY KEY, exp_date DATE NOT NULL,
                     seq_data TEXT, quality_score DECIMAL(5,2), exp_log TEXT);
INSERT INTO dna_seq VALUES
('2023-03-25', pg_read_file('seq_0325.txt'), 95.5, pg_read_file('log_0325.txt'),
('2024-03-26', pg_read_file('seq_0326.txt'), 92.3, pg_read_file('log_0326.txt'),
('2024-03-27', pg_read_file('seq_0327.txt'), 89.7, pg_read_file('log_0327.txt'),
```

Figure 4: Comparison of Compression Effect.

```
-- 3. Substring replacement
UPDATE dna_seq SET seq_data =
          OVERLAY(seq_data placing 'new_str' from 1);
-- 4. Full-text search
SELECT to_tsvector(exp_log) FROM dna_seq WHERE seq_id=1;
```

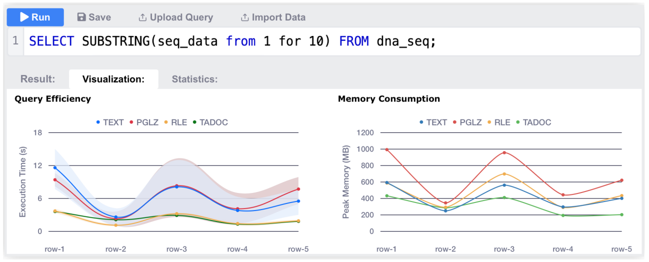
4.2 Scenario 2: HocoPG Admin

Managing large text objects through a database terminal presents several challenges: 1) The data scale is substantial, making it impractical to display effectively within a terminal; 2) Users are isolated from the underlying data storage format; 3) Comparing and selecting compression methods demands the development of specialized code to assess the compatibility of diverse storage solutions with specific user workloads. Given these challenges, most DBAs and users tend to select storage options based on intuition or default to the database's predetermined choices.

In recognition of these obstacles, we develop HocoPG Admin to facilitate both DBAs and general users in profiling the efficiency of text processing. This tool aims to provide valuable insights into text storage solutions, thereby reducing the complexity associated with text management and enhancing overall efficiency. The primary user interface of HocoPG Admin is shown in Figure 3. We next introduce two of its key features.

Feature 1: Out-of-Line Table Visualization. Figure 3 illustrates HocoPG Admin's approach to display database tables, closely mirroring the out-of-line storage format used by the underlying storage. Users begin by choosing a database ① and the table ② they wish to view, which then shows up in the "Result" panel on the right ③. Columns in out-of-line storage are hyperlinked, allowing users to click through and view stored text data on a separate web page ④.

Feature 2: Scheme Comparison. HocoPG Admin enhances the SQL editing and execution process by enabling query input into a SQL editor and result visualization in the "Result" panel. More significantly, its fundamental goal transcends simple query assistance; it endeavors to equip DBAs and users with analytical insights for the selection of optimal compression schemes. Upon selecting a database table, the system commences a row sampling process to prepare the data. It then applies various compression strategies to this sample dataset to evaluate both compression efficiency and query performance, as illustrated in Figures 4 and 5. The analysis presents users with data on average performance and variability over multiple iterations. By offering a framework for comparative analysis, HocoPG Admin provides a foundation for DBAs and general users to discern the most suitable compression algorithm tailored to their unique requirements and application contexts.



```
SELECT SUBSTRING(seq_data from 1 for 10) FROM dna_seq;
```

Figure 5: Comparison of Query Execution.

ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (No. 62322213 and 62172419), Beijing Nova Program (No. 20230484397 and 20220484137), and the Outstanding Innovative Talents Cultivation Funded Programs 2023 of Renmin University of China. Feng Zhang is the corresponding author of this paper.

REFERENCES

[1] Bishwaranjan Bhattacharjee, Lipyeow Lim, Timothy Malkemus, George Mihaila, Kenneth Ross, Sherman Lau, Cathy McArthur, Zoltan Toth, and Reza Sherkat. 2009. Efficient index compression in DB2 LUW. *PVLDB* 2, 2 (2009), 1462–1473.
[2] Peter Boncz, Thomas Neumann, and Viktor Leis. 2020. FSST: fast random access string compression. *PVLDB* 13, 12 (2020), 2649–2661.
[3] Solomon Golomb. 1966. Run-length encodings (corresp.). *IEEE transactions on information theory* 12, 3 (1966), 399–401.
[4] Jiawei Guan, Feng Zhang, Siqi Ma, Kuangyu Chen, Yihua Hu, Yuxing Chen, Anqun Pan, and Xiaoyong Du. 2023. Homomorphic Compression: Making Text Processing on Compression Unlimited. *SIGMOD* 1, 4 (2023), 1–28.
[5] Krishna Kunchithapadam, Wei Zhang, Amit Ganesh, and Niloy Mukherjee. 2011. Oracle database filesystem. In *SIGMOD*. 1149–1160.
[6] Chunwei Liu, Anna Pavlenko, Matteo Interlandi, and Brandon Haynes. 2023. A deep dive into common open formats for analytical dbmss. *PVLDB* 16, 11 (2023), 3044–3056.
[7] Sorin Stancu-Mara and Peter Baumann. 2008. A comparative benchmark of large objects in relational databases. In *Proceedings of the 2008 international symposium on Database engineering & applications*. 277–284.
[8] Jiajia Wang, Weizhong Zhao, Xinhui Tu, and Tingting He. 2023. A novel dense retrieval framework for long document retrieval. *FCS* 17, 4 (2023).
[9] Feng Zhang, Jidong Zhai, Xipeng Shen, Onur Mutlu, and Wenguang Chen. 2018. Efficient document analytics on compressed data: Method, challenges, algorithms, insights. *PVLDB* 11, 11 (2018), 1522–1535.