Enabling Efficient NVM-Based Text Analytics without Decompression

Xiaokun Fang[◇], Feng Zhang[◇], Junxiang Nong[◇], Mingxing Zhang[+], Puyun Hu[◇], Yunpeng Chai[◇], Xiaoyong Du[◇]
[◇]*Key Laboratory of Data Engineering and Knowledge Engineering (MOE), and School of Information, Renmin University of China*
[+]*Department of Computer Science and Engineering, Tsinghua University*
{fangxiaokun, fengzhang, junxiangnong}@ruc.edu.cn, zhang_mingxing@mail.tsinghua.edu.cn,
{ceerrep, ypchai, duyong}@ruc.edu.cn

2024 IEEE 40th International Conference on Data Engineering (ICDE) | 979-8-3503-1715-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICDE60146.2024.00286

***Abstract*—Text analytics directly on compression (TADOC) is a promising technology designed for handling big data analytics. However, a substantial amount of DRAM is required for high performance, which limits its usage in many important scenarios where the capacity of DRAM is limited, such as memory-constrained systems. Non-volatile memory (NVM) is a novel storage technology that combines the advantage of reading performance and byte addressability of DRAM with the durability of traditional storage devices like SSD and HDD. Unfortunately, no research demonstrates how to use NVM to reduce DRAM utilization in compressed data analytics. In this paper, we propose N-TADOC, which substitutes DRAM with NVM while maintaining TADOC's analytics performance and space savings. Utilizing an NVM block device to reduce DRAM utilization presents two challenges, including poor data locality in traversing datasets and auxiliary data structure reconstruction on NVM. We develop novel designs to solve these challenges, including a pruning method with NVM pool management, bottom-up upper bound estimation, correspondent data structures, and persistence strategy at different levels of cost. Experimental results show that on four real-world datasets, N-TADOC achieves 2.04× performance speedup compared to the processing directly on the uncompressed data and 70.7% DRAM space saving compared to the original TADOC.**

I. Introduction

The advent of the big data era has amplified the need for large-scale data analytics. Text analytics directly on compression (TADOC) [1]–[9] emerges as an efficient tool for addressing this need. TADOC's unique feature of directly processing compressed data, circumventing the need for decompression, translates into significant savings in storage space. Additionally, TADOC can also optimize processing time by reusing input and intermediary data [2], [3]. Studies have demonstrated that TADOC is capable of reducing processing time by 50% and storage space by 90.8% [2], [3]. However, traditional TADOC utilizes dynamic random-access memory (DRAM) extensively, leading to high costs.

On the other hand, non-volatile memory (NVM), a next-gen storage technology with a density superior to DRAM and promising performance in various real-world applications [10]–[13], offers a potential remedy. By offloading data onto NVM, one can decrease DRAM usage and handle larger datasets due to NVM's greater density. Furthermore, NVM's lower equipment and power costs compared to DRAM can reduce the total cost of ownership for data analysts [14]. In addition, we argue that an efficient implementation of TADOC on NVM aligns perfectly with NVM's unique characteristics. NVM's lower bandwidth compared to DRAM aligns with TADOC's ability to process data under compression, reducing bandwidth demand. TADOC can further augment this alignment by utilizing text redundancy to decrease update frequencies during analytics, thereby minimizing NVM write operations and enhancing its durability. As such, this paper explores the potential of integrating NVM into TADOC, aiming to deliver a cost-effective text data analytics solution.

Despite the benefits of building TADOC on NVM devices, establishing a compressed data direct processing framework based on NVM is a challenging task. TADOC's requirement for random access to compressed datasets can lead to poor data locality when directly applied to NVM. In addition, when the capacity of traditional data structures is insufficient, violent reconstruction is required. This can lead to a large number of redundant access to NVM, which can further decrease the efficiency of TADOC on NVM. These challenges necessitate the invention of new techniques to minimize the impact of poor data locality and superfluous NVM access on NVM-based TADOC processing.

Currently, none of the previous works can be applied to enable TADOC on NVM efficiently. Zhang *et al*. [1], [2] first proposed TADOC and it is designed in sequential execution. There are prior works [5], [8], [15] focused on accelerating TADOC through GPU or porting TADOC to other accelerator. However, due to the differences in hardware architecture, they cannot be utilized efficiently by NVMs.

We design N-TADOC, the first **N**VM-based **t**ext **a**nalytics **d**irectly **o**n **c**ompression. N-TADOC enables us to utilize nonvolatile memory to perform text analytics on the data in the compressed state. It integrates three novel designs to overcome the above challenges for NVM-oriented compressed data direct processing. First, we introduce a pruning method to eliminate redundancy in compressed data, and then carefully restructure the compressed data in the NVM pool for better cache utilization. Second, we design a bottom-up method to estimate the upper bound of variable length of data structures on NVM, reducing redundant NVM access due to insufficient

- Feng Zhang is the corresponding author.

2375-026X/24/$31.00 ©2024 IEEE
DOI 10.1109/ICDE60146.2024.00286

space for refactoring. Third, we design data structures such as *vector* and *hashtable* to adapt to NVM pool management and the upper bound summation method of data object length. Finally, we utilize various frameworks to achieve different levels of persistence cost for text analytics. These collective design decisions have allowed us to create a robust, efficient NVM-based text analytics framework capable of surmounting these challenges. N-TADOC represents a crucial advancement in harnessing NVM's full potential in data analytics applications.

We conduct a comprehensive evaluation of N-TADOC on the Intel Optane platform utilizing four diverse real-world datasets, including the Yelp COVID-19 data [16], NSF Research Award Abstracts [17], and Wiki dump data with different sizes [18]. To validate the effectiveness of N-TADOC, we assess the performance of six operations commonly used in text analytics, as outlined in previous studies [19]–[21]. The results indicate that N-TADOC not only reduces the average DRAM occupancy by more than 70% compared to the original TADOC but also significantly boosts the efficiency of typical data analytics tasks by 2.04×. These promising outcomes demonstrate the potential of N-TADOC to efficiently enable text analytics on NVM-based devices.

To the best of our knowledge, N-TADOC is the first work to provide efficient text analytics on NVM without the need for decompression. In this paper, we make the following contributions.

- We present N-TADOC, which is the first work enabling efficient NVM-based text analytics directly on compressed data.
- We propose a novel pruning method with NVM pool management and maintain data locality for text analytics.
- We design a bottom-up summation technique to estimate the upper bound of the size of data objects, eliminating redundant NVM access for data object reconstruction.
- We develop novel data structures to adapt to NVM pool management and summation technique.
- We conduct comprehensive experiments of N-TADOC on NVM, demonstrating its significant benefits compared to uncompressed text analytics.

II. Background

In this section, we introduce TADOC and non-volatile memory (NVM) devices, which constitute the background and premises of our work.

TADOC. TADOC [1]–[9] is a novel lossless compression method that can process compressed data directly without decompression. To be more specific, TADOC performs a digital encoding of the original data input employing a dictionary conversion. It then recursively describes the digital data that has been transformed using context-free grammar (CFG) and translates this description into rules. Figure 1 illustrates how TADOC compresses data using CFG representation. Figure 1 (a) shows the original two files of *file A* and *file B*, where wi represents a word. TADOC extends Sequitur [22]–[24] as core algorithm to transfer input data to the CFG, as shown in Figure 1 (b). The first rule R0 represents compressed *file A* and *file B* including delimiters, and the following rules R1, R2 represent frequently occurring patterns in compressed files. The CFG can be further transferred to Figure 1 (c) with the help of the dictionary shown in Figure 1 (d). After repeated data pieces have been converted into a number of rules using context-free grammar, the process of data analytics is then represented as rule interpretation. TADOC inserts one segmentation symbol for the file boundary so that we can utilize the redundant information that exists between files. Moreover, these rules can be further represented as a directed acyclic graph (DAG), as shown in Figure 1 (e), so the data analytics tasks can be transformed into a DAG traversal problem. The latest TADOC technology [1]–[4] can now cut down on the amount of storage space required for compressed text analytics by an average of 90.8%.

Example. We take *word count* as an example to show how to perform text analytics through DAG traversal in Figure 1 (e). In Step 1, R0 is weighted as 1, while R1 and R2 are given the weight of 0. In Step 2, R0 passes weight to subrules R1 and R2, so R1's weight reaches 2 and R2 reaches 2. Besides, R2 receives weight from R1 in the next iteration, which makes its weight reach 6. In Step 3, we accumulate the weighted word frequencies of all three rules, which constitutes the final result.

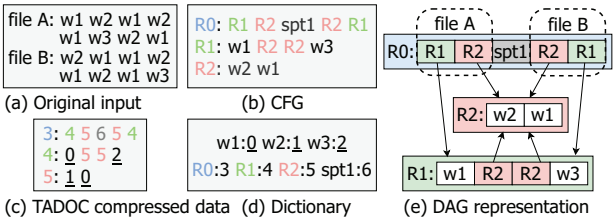

Fig. 1. A compression example with TADOC.

NVM device. As a promising storage technology, NVM has prospective features [25]–[29]. First, unlike hard-disk drivers (HDDs) or solid-state drives (SSDs) that only support block-based data access, NVM is byte addressable, which is similar to DRAM. Second, NVM has a high storage density, higher than DRAM, and comparable to SSD and HDD. Third, it is non-volatile, which means that all data written to NVM will not be lost when the machine is suddenly powered off or the program is interrupted. However, compared with DRAM and disk, programming on NVM requires special considerations. For example, the read and write latency of NVM is unbalanced. In particular, NVM has a read latency similar to DRAM, but its write latency is higher than DRAM. Since non-volatile memory combines the advantages of DRAM, SSD, and HDD, it is capable of taking their place in the fields of file systems, databases, and other applications [11], [30], but special optimizations are required.

Intel Optane. We conduct experiments on the Intel Optane platform [10]. It has three modes: 3) *Direct access mode*. In this mode, applications can access NVM directly through the load/store instruction, which bypasses the interface of

the traditional file system and maximizes the performance of NVM. 2) *File mode*. In this mode, NVM can be recognized by the file system as an external storage device. 1) *Memory mode*. As an additional memory device, NVM is not different from DRAM and does not guarantee the persistence of data. The work of this paper is designed in the direct access mode. Moreover, please note that our work is not limited to Intel Optane. Other platforms with similar characteristics such as ReRAM [31] and PCM [32] can also benefit from our design.

III. Motivation

This section initially outlines the obstacles that arise when integrating NVM into the direct analysis of compressed text. Following this, it briefly revisits prior studies to clarify why their methodologies do not offer solutions to these challenges. Lastly, it details the practical applications of N-TADOC.

A. Challenges

In this section, we discuss two major challenges of enabling efficient text analytics on NVM without decompression.

Poor data locality in traversing DAG on NVM. The first critical challenge comes from the management of compressed datasets on NVM [33]. The characteristics of compressed data necessitate random access to various nodes during the traversal of the DAG formed by the CFG. For instance, as depicted in Figure 1 (e), after initially traversing R0, the grammatical representations of R1 and R2 are retrieved. However, NVM's relatively large access granularity and the scattered distribution of data objects on NVM result in poor data locality. In particular, the physical 3D-Xpoint media access granularity is 256 bytes. If R0, R1, and R2 are distributed randomly on the NVM device with poor data locality, then it becomes difficult to leverage this feature. This lack of data locality leads to frequent redundant access operations and significantly impacts the efficiency of the analytics process. Therefore, the task of maintaining a DAG representation in an NVM pool with good data locality remains a challenging task. This lack of data locality combined with the larger access granularity of NVM (256 bytes in Intel Optane NVM) exaggerates the problem of access amplification and hence considerably hampering the efficiency of the analytics process. Thus, managing a DAG representation with improved data locality in an NVM pool is a tough task.

Redundant access and overhead due to dynamic data object growth on NVM. Another significant challenge faced by N-TADOC is the management of data object capacity limitations during the analytics process on NVM [34]. As the analytics progresses, intermediate results and data objects grow dynamically, potentially surpassing the capacity of the allocated space on NVM. This situation necessitates frequent reconstruction of data objects, resulting in a substantial number of redundant access operations on NVM. The process of dynamic data object growth involves time-consuming read-modify-write operations, as data objects need to be resized or relocated to accommodate the expanding data. Addressing this challenge requires the development of efficient strategies to manage dynamic data object growth on NVM and minimize the overhead associated with frequent reconstructions. By addressing the challenge of redundant access and overhead due to dynamic data object growth on NVM, N-TADOC can achieve better efficiency and performance, enabling faster and more scalable text analytics on NVM-based systems.

B. Revisiting Previous Methods

In this part, we explore past traversal-based methods to explain motivation for a new NVM-based TADOC architecture.

Why NVM-based STL data structure does not apply? In the realm of NVM-based TADOC, the utilization of sophisticated data structures such as vectors and hash tables is unavoidable. First, the data structures available in the standard library do not inherently support the allocation of space on NVM devices, which makes it difficult to implement the TADOC algorithm. Second, after exploring and experimenting with the PMDK (Persistent Memory Development Toolkit) [35], we have found that the effectiveness of many data structures is significantly lower than that of the STL. Therefore, we believe that it is crucial to construct a collection of efficient libraries that can be used to store results generated by text analytics on NVM.

Why previous TADOC method does not apply? In our preliminary explorations, we attempt to apply existing TADOC methods [1]–[4] to NVM environments for direct text analytics. We overloaded the allocator of the data structures from previous work to point to NVM while keeping methods unchanged. Directly applying Optane PM to TADOC incurs 13.37× performance overhead compared to the original version. These existing methods are optimized for DRAM and fail to fully leverage the characteristics of NVM. The non-volatile nature of NVM requires a different approach to data storage and management compared to DRAM. Therefore, exploring new algorithms and data management strategies is essential to fully exploit the potential of NVM and achieve high-performance text analytics.

C. Application Scenarios of N-TADOC

In this section, we explore potential application scenarios for N-TADOC. In conventional big data applications, disks and DRAM have always been central components, with the former being used for persistent data storage and the latter for faster access and disk cache. However, with the advent of NVM technology, it is anticipated to play a significant role in future designs [36]. Implementing TADOC on NVM could benefit a multitude of big data applications, and we highlight three key examples:

- **Distributed system** with mixed computing and storage resources is a potential application scenario [37]. In such systems, the real-time analysis and processing of text data are vital. Employing NVM as a medium for direct analytics of compressed text could lead to notable advantages. The quick access time and reduced latency of NVM can enhance system responsiveness and performance, particularly beneficial when multiple distributed components

need to concurrently access and analyze compressed text. In addition, the decreased dependency on traditional DRAM storage improves cost-effectiveness. Leveraging NVM's high-density storage and non-volatility can allow text analytics tasks to be carried out more efficiently within distributed systems, thereby enhancing scalability and reducing operational costs.

- **Data mining** is an essential method for a collection of modern information extraction tasks [38] and often involves large-scale text datasets. By utilizing NVM for direct analytics of compressed text, significant improvements can be achieved in terms of search efficiency and storage requirements. For instance, in data mining applications, the ability to perform fast searches and build indexes directly on compressed text stored in NVM enables quicker identification of patterns and insights within the data. The combination of efficient text compression techniques and direct analytics on NVM contributes to more efficient and scalable data mining processes.
- **Embedded Systems** such as IoT networks and wireless sensor nodes often operate under more stringent power constraints [39]. The data stored in memory are often written to an SSD as the standard procedure. However, as a result of the speed gap that exists between DRAM and SSD, NVM devices attract an increasing amount of attention. By enabling support for compressed data direct processing on NVM, this emerging technology can serve as an intermediate buffer layer, significantly enhancing the overall effectiveness of data analytics while simultaneously conserving energy. The ability to directly analyze compressed data on NVM not only reduces the need for frequent data movement between DRAM and storage devices but also exploits NVM's non-volatile nature, enabling energy-efficient processing and reducing data transfer overhead.

IV. N-TADOC Design

In this section, we introduce the design of N-TADOC architecture, including the pruning method with NVM pool management, the bottom-up summation for NVM space, the N-TADOC data structures, and different persistence strategies. Next, we show the overview of N-TADOC.

A. Overview

We introduce N-TADOC, a framework for enabling efficient TADOC on NVM. The overview of N-TADOC is shown in Figure 2. The system takes TADOC compressed data as input and produces the required text analytics results. N-TADOC leverages pruning to eliminate redundancy and NVM pool management to manage the DAG structure efficiently. The framework employs a bottom-up approach to estimate the upper bound of data structure lengths on NVM and uses novel data structures for NVM pool management and bottom-up summation respectively. Additionally, N-TADOC provides phase-level and operation-level persistence for text analytics to ensure data integrity and availability.

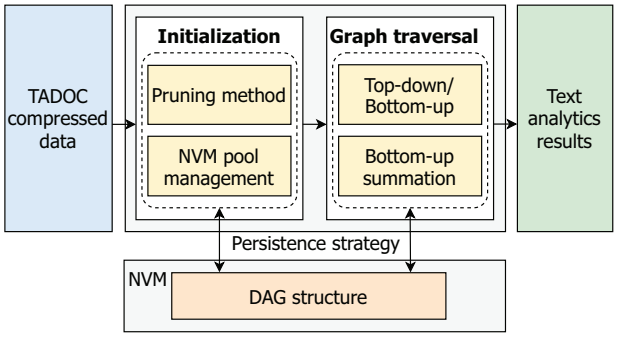

Fig. 2. N-TADOC overview.

Workflow. The workflow of N-TADOC consists of two phases: initialization and graph traversal. During the initialization phase, N-TADOC performs the necessary setup and data preparation for the graph traversal phase. This includes constructing the DAG for the dataset, initializing the hash table or vector for storing the rule information, and allocating memory for the graph traversal process. The initialization phase also involves reading any persisted data if available. In the graph traversal phase, N-TADOC traverses the DAG and propagates the weight of the rules to each file in the dataset. The traversal process is dependent on the chosen traversal strategy, which can be either top-down or bottom-up. During the traversal phase, N-TADOC accesses the NVM-resident data structures for rule and frequency lookup. N-TADOC provides flexibility in data persistence, allowing for the results to be persisted after each operation or at the end of the phase. Once the graph traversal and result collection is complete, N-TADOC returns the results such as wordcount output $\langle \text{word1}, \text{count1} \rangle, \langle \text{word2}, \text{count2} \rangle, \ldots$ to the user for further analytics.

Solutions to challenges. N-TADOC optimizes compressed dataset arrangement using pruning and reorganization techniques to reduce redundancy and improve cache utilization. It employs a bottom-up method to estimate NVM's capacity for variable-length data structures of intermediate analytical results, minimizing the need for frequent reconstructions. Additionally, tailored data structures and various frameworks in N-TADOC optimize NVM pool management and persistence costs, resulting in an efficient and robust text analytics framework that harnesses NVM's potential.

Difference from previous works. TADOC [1] targets the data structure design, which is a lossless compression method that employs CFG to translate data into rules. After rule representation, the analytics tasks can be transformed into rule analysis, as discussed in Section II. TADOC involves only data structure design without any architectural optimization. G-TADOC [5] targets the computation, which provides the GPU acceleration for the computation part of rule processing in TADOC, including the parallelization with a large number of threads, dependency elimination in rule parallel processing, and order maintenance among massive threads.

As to this work, N-TADOC targets the storage, which utilizes NVM block devices to improve storage efficiency. Specifically, we design a pruning method with NVM pool management, bottom-up upper bound summation, and persistence strategy at different levels. These hardware storage designs from a storage architectural perspective have never been considered before. N-TADOC is a holistic system that significantly reduces both latency and space costs. It achieves this by utilizing advanced techniques to optimize data organization and management, reducing redundancy and improving locality.

B. Pruning Method with NVM Pool Management

We next show the detailed design of rearranging rules in CFG with the management of the NVM pool. Similar ideas have also been used in recent academic explorations [40]–[42]. We redesign the DAG structure generated by compressing the original text stored in NVM. Our proposed structure can maintain good data locality during DAG traversal, thus reducing the redundant reads and writes of NVM due to frequent cache misses, and improving the efficiency of data analytics.

General design. We introduce a pruning approach that is based on two key observations. First, we observe that generated rules can contain numerous duplicate subrules. For instance, in Figure 1, the rule R0 has two subrules, R1s, which are identical. Second, the internal structure of rules can be random, where subrules and words are arranged in a disordered manner. This duplication pattern is not leveraged further, and the structural disorder leads to non-cache-friendly DAG traversal. To address this issue, we introduce a pruning technique that trims the grammar representations of the rules for text analytics tasks. In our pruning technique, we traverse the rule to obtain the subrules and words it contains along with their frequencies. We then write all subrules and their frequencies to the DAG pool, followed by the frequencies of words. After pruning, the IDs and frequencies of the subrules and words in the rule are arranged consecutively on the NVM, which sets the foundation for the locality of the traversal on the DAG. To maximize the utilization of the large read/write granularity on NVM devices, we arrange the pruned representations of different rules adjacently in the DAG pool. Furthermore, we organize the metadata of rules in the NVM pool, including the position of subrules and words (referring to the offset at which the rule is stored in the DAG pool), the out/in degree (representing the number of outgoing/incoming edges for the rule in the compressed file's DAG representation), word list size (indicating the number of different word types included in the rule), and the weight of the rule (representing the frequency or occurrence of the rule in the dataset). This allows for efficient retrieval and traversal of the pruned DAG. The NVM pool also contains a traversal queue and a frequency counter. The traversal queue is used to record the progress of a task during a top-down traversal process, take out the rule being traversed, and add its subrules to the queue. For example, in Figure 3, we take rule R1 out of the queue and add its subrules R2, R3, and R4 to the queue. The counter records the frequencies of words or sequences based on the requirements of the task. It consists of vectors or hash tables, which are covered in Section IV-D. At the end of the traversal, we update the counter based on the weight of each rule and their word list.

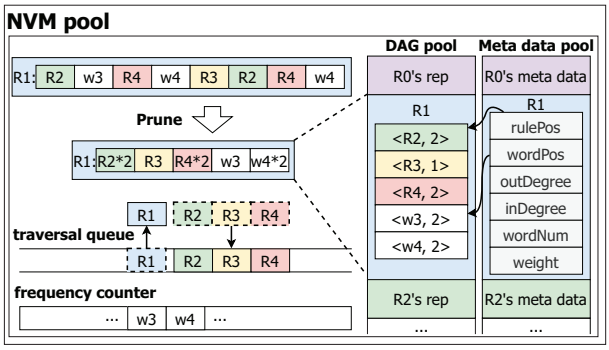

Fig. 3. Pruning method with NVM pool management.

Detailed algorithm. We present our NVM pool management-based pruning approach in Algorithm 1. The algorithm takes as input the rules to be pruned, denoted as R, and the pointer to the NVM pool, $pool\_top$. Initially, the algorithm calls the $getSubrules$ and $getWords$ functions to obtain the IDs and frequencies of the subrules and words in R, and stores them in a bucket data structure, as shown in Lines 2 and 3. Subsequently, N-TADOC writes each tuple $\langle \text{id}, \text{freq} \rangle$ to the NVM pool by moving the pointer accordingly, as indicated in Lines 5 to 8.

Algorithm 1 Pruning Method with NVM Management

Input: Rule that needs to be pruned: R; NVM pool pointer: $pool\_top$
Output: R's new management in NVM pool

```
1: function pruneNVMManage(r, pool_top)
2:     sub_rules = getSubrules(R);
3:     words = getWords(R);
4:
5:     write sub_rules at pool_top;
6:     pool_top ← pool_top + sub_rules.size;
7:     write words at pool_top;
8:     pool_top ← pool_top + words.size;
```

Example. We can demonstrate the pruning process with NVM pool management using the following example. Let us consider the raw grammar "R1→R2 w3 R4 w4 R3 R2 R4 w4". After executing Algorithm 1, R1 is pruned to the simplified version "R1→R2×2 R3 R4×2 w3 w4×2", as shown in Figure 3. It is worth noting that R1's representation is arranged adjacent to the next rule, R2, in the NVM pool. This arrangement eliminates at most 50.2% of the grammar redundancy on NVM. This arrangement not only saves NVM space but also reduces internal fragmentation, making it easier for the traversal on the DAG to achieve good data locality.

Overall, the pruning method effectively trims the grammar representation of rules, making it more efficient for text analytics tasks.

Complexity analysis. We denote the number of rules in CFG as N_r and the number of words as N_w. We denote the original grammar length of rule R as $L_{R_{raw}}$, and the trimmed length is expressed as $L_{R_{prune}}$. Because we need to traverse the grammar of R and use buckets to count the frequency of subrules and words, the time complexity of this process is $O(L_{R_{raw}})$, and the space complexity is $O(N_r+N_w)$. Then we traverse buckets to harvest and write the pruned grammar to NVM, with the time complexity of $O(L_{R_{prune}}+N_r+N_w)$ and space complexity of $O(L_{R_{prune}})$. Therefore, the overall time complexity of Algorithm 1 is $O(L_{R_{raw}}+L_{R_{prune}}+N_r+N_w)$, and its space complexity is $O(N_r + N_w + L_{R_{prune}})$.

The pruning method can also support sequence analytics tasks by performing a lightweight bottom-up preprocessing step to obtain the head and tail structure of all rules, which is also mentioned in Section IV-D. Then, using this head and tail structure, pruning operations can be performed, avoiding analytics errors caused by rearranging subrules and sequences within rules. Therefore, it can be argued that the pruning method keeps TADOC's functional scope, making it suitable for both sequence-dependent and sequence-independent analytics tasks. Similarly, the "Bottom-up Summation" method explained below remains unaffected by the task's dependency on sequence information.

C. Bottom-Up Summation

This part shows the detailed design of the bottom-up summation technique for the length of the data structure. Traditional TADOC traverses the DAG structure in a top-down or bottom-up manner for different datasets. In the top-down method, nodes' weights propagate from top down to bottom in topological order. In the bottom-up method, nodes pass word lists in reverse topological order from bottom up to top. To adapt to the growing word list in bottom-up traversal, TADOC uses variable length data structures such as *unordered_map*. On NVM, when the capacity of variable length data structure is insufficient, the cost of reconstruction is high. Therefore, we design a method that can estimate the length of a data structure based on the bottom-up traversal method of TADOC.

General design. In this section, we introduce our proposed technique for estimating the length of data structures, which we refer to as the "bottom-up summation" technique. This technique is closely related to the bottom-up traversal method, where the word list of the bottom rules in the DAG graph is passed up level by level to update the word list of all rules. Our approach focuses on estimating the upper bound of the word list length for each rule in a lightweight traversal from the bottom to the top of the DAG. To begin, we define a rule as "determined" once an upper bound has been established for its word list length. Initially, if a rule does not contain any subrules, we set its upper bound as equal to its length. We then perform a summation on the rule R, and if we discover that R contains a subrule R' that has not yet been determined, we repeat the same process for R' until all of the subrules in R have been determined. Once all subrules have been determined, we calculate the upper bound of rule R as the sum of the upper bounds of all of its internal subrules plus the length of its original word list. After the traversal of R is completed, its determination is made. Finally, we allocate space for the word list based on its upper bound, which helps to eliminate the overhead of redundant read and write operations during data structure reconstruction on NVM. This process improves the efficiency of the data structure reconstruction process and reduces the access latency of the NVM, as it avoids unnecessary read and write operations that can negatively impact the overall performance.

Detailed algorithm. Algorithm 2 presents our bottom-up summation method, which helps determine the upper bound of the word list length for each rule in the DAG graph. This algorithm takes as input the rule R for which the upper bound needs to be determined and all the determined rules' upper bounds, denoted as L. To begin, the algorithm traverses all subrules of rule R, as shown in Line 2. If any subrule has not been determined, the algorithm recursively determines the upper bound of that subrule by calling itself with the subrule as input and updating L, as shown in Lines 3 to 5. Then, the algorithm calculates the upper bound of R by summing the upper bounds of all the internal subrules and adding the length of its original word list, as shown in Lines 6 to 8. Once the algorithm has computed the upper bound of R's word list, it sets R to be determined, as shown in Line 9. This marks the completion of the bottom-up summation process for R, and the upper bound can be used to allocate space for its word list.

Algorithm 2 Bottom-Up Summation

Input: Rule that needs to determine the upper bound: R; Upper bounds of all rules: L
Output: Upper bound of rule R: l

```
1: function bottomUpSummate(R, L)
2:     l ← 0;
3:     for r in R.subrules do
4:         if r is not determined then
5:             bottomUpSummate(r, L);
6:         l ← l + L[r];
7:     l ← l + R.wordNum;
8:     L[R] ← l;
9:     set R determined;
```

Example. We use an example to demonstrate the bottom-up summation process. To obtain the upper bound of the word list length of R0 in Figure 1 (e), we need to recursively determine the upper bounds of R1 and R2, respectively. R2 does not contain any subrules, so the upper bound is equal to its word list's length, which is 2. R1 only contains subrule R2 and two words itself, so its upper bound is 2+2, which is 4. Finally, we obtain R0's upper bound in this bottom-up method, which is 6 (4+2).

Complexity analysis. The time complexity of Algorithm 2 is important to consider since it affects the efficiency of the approach. In this regard, we can analyze the complexity of the bottom-up summation process. Let us denote the generated subgraph of node R and all nodes it can reach in the DAG as $\mathcal{G}$. We respectively denote the number of nodes and edges in graph $\mathcal{G}$ as V and E. Since the bottom-up summation process needs to access all nodes once in graph $\mathcal{G}$, the complexity of the process can be expressed as $O(V)$. Additionally, the upper bound information of nodes is transmitted from bottom to top along the edge once, and the complexity of this process can be expressed as $O(E)$. Hence, the overall complexity of Algorithm 2 is $O(V + E)$.

D. N-TADOC Data Structures

This section shows the data structure redesigned on N-TADOC to adapt to the NVM pool management technique proposed in Section IV-B and the bottom-up summation method proposed in Section IV-C. This section first introduces the implementation of traditional data structures such as the hash table on NVM, and then introduces the head and tail structure designed for rules in order to support efficient sequence operation.

Basic structures for analytics. In this part, we explain the data structure of allocating space on the memory pool for N-TADOC, which is designed to efficiently process compressed data in NVM. The most important data structure for TADOC is the hash table, which can be used to store results locally or globally, and the same structure is used in the design of N-TADOC. Figure 4 shows the hash table data structure used in N-TADOC. In the first step, we read the compressed graph dataset and establish a DAG pool on NVM. In the second step, the metadata is fed to the bottom-up estimator to generate the estimated length of each rule data object. In the third step, NVM allocator allocates space for data objects on NVM using estimated results. The original state of the hash table is shown in Figure 4 (a), where the status buffer is used to check the status of each node, the key buffer is used to store keys, and the value buffer is used to store corresponding values. The hash table length is adjusted upward to the power of 2 for alignment to improve the hit rate of the cache. Since the upper bound of the size of each hash table is known, all hash tables can be stored adjacent in a memory pool to reduce the external fragmentation of NVM and save space. Figure 4 (b) shows the state after inserting $\langle 2, 5\rangle$ into the hash table, assuming that the key-value pair is mapped to the first node. In case of hash collision, the final mapping location of the new node is determined by pseudo-random detection and hashing, which ensures that the new node is inserted into the table with minimal disruption to the existing data structure.

Head and tail structures for sequence support. The head and tail structures are integral to improving the efficiency of sequence analytics such as *sequence count*. Traditionally, TADOC utilizes function recursive calls to expand rules, which results in coarse-grained expansion and consequent inefficiencies. However, to address this limitation and enhance

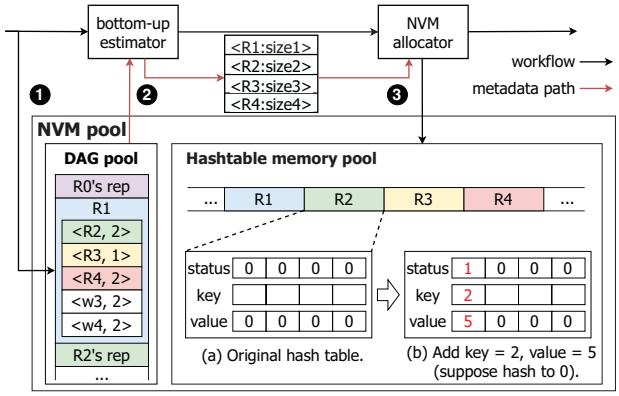

Fig. 4. Hash table design.

the sequence support, the GPU-based TADOC [5] proposes the use of head and tail data structures for each rule, which store the content of the beginning and end of the rule. By leveraging these data structures, it becomes possible to avoid multi-rule scanning and instead focus on only the head and tail buffers of different subrules, thereby increasing the speed of sequence analytics.

E. Persistence Strategy

This part shows several persistence strategies we employ in N-TADOC. To fully leverage the persistence features of non-volatile memory, we develop N-TADOC persistence cost at both the operation level and phase level to cater to different requirements.

Operation-level persistence cost. For operation-level persistence cost, we utilize the Persistent Memory Development Kit (PMDK) [35] *libpmemobj-cpp*, which implements transaction and logging mechanisms to ensure the atomicity of operations. However, it should be noted that the transaction mechanism may cause a significant issue of write amplification, which can lead to degraded efficiency in N-TADOC data analytics. To address this issue, we propose another persistence strategy, which is referred to as phase-level persistence cost.

Phase-level persistence cost. Regarding phase-level persistence cost, we directly map the space on NVM to memory by utilizing *libpmem*, and explicitly flush back to NVM at the end of each N-TADOC phase. As the data is only persisted at the end of each phase, the cost of persistence is considerably reduced. Specifically, in the initialization phase, N-TADOC first establishes the DAG structure, which is then flushed back to NVM at the end of the phase. In the graph traversal phase, N-TADOC obtains the result and persists it to NVM. In the event of failure, N-TADOC returns to the previous checkpoint, but the cost of persistence is amortized, resulting in better performance. Moreover, if the process is interrupted by external factors such as power failure, we restart analytics from the current phase, which significantly enhances its ability to prevent potential risks. The data generated in different phases is always isolated from the persistent data of the

previous phase, the persistent data of the previous stage will only be read in the next phase. This means that when a failure occurs in the current phase, the recovery process can directly overwrite the dirty data. This requires identifying and reverting to the previous checkpoint, discarding the intermediate results of the current phase. As a result, the recovery process needs to recompute certain portions of the analytics, potentially leading to longer recovery times.

In summary, the trade-off lies in the balance between persistent performance and recovery performance. By reducing the frequency of data persistence, N-TADOC achieves better overall performance during normal execution. However, in the event of a failure, the recovery process may incur additional computational overhead to recompute certain portions of the analytics. The specific trade-off between these two aspects depends on the characteristics of the workload, the frequency of failures, and the recovery requirements of the system.

V. Implementation

The primary focus of this paper is to implement the N-TADOC framework for direct processing of compressed files on non-volatile memory. N-TADOC comprises a set of text analytics tasks that are typically found in various existing applications. The main framework is developed using C/C++, including the data structure component, compressed file reading components, and graph traversal component.

In addition, *libpmemobj-cpp* is utilized to achieve operation-level persistence while *libpmem* is used for phase-level persistence. The operation-level persistence provided by PMDK guarantees the atomicity of operations using transaction and logging mechanisms, which helps to maintain data consistency. On the other hand, phase-level persistence using *libpmem* enables us to flush data to NVM at the end of each phase, thereby reducing the persistence cost.

The N-TADOC framework's implementation allows for the efficient processing of compressed files on NVM while providing data consistency guarantees through persistence mechanisms.

VI. Evaluation

In this section, we first introduce our experimental setup, then measure the performance of N-TADOC on NVM, and compare it with uncompressed text analytics on NVM, TADOC, N-TADOC on SSD and HDD for evaluation. We also study the space savings of N-TADOC on DRAM, break down analytics time, and point out the impact of different workloads on N-TADOC efficiency.

A. Experimental Setup

In this part, we show our experimental setup.

Methodology. In our evaluation, we compare the performance of our method, which enables TADOC on NVM devices, denoted as "N-TADOC", with the baseline of analyzing uncompressed text on NVM devices. In the baseline configuration, the text analysis task was performed on NVM. No specialized compression techniques or methods designed for NVM were employed, except for the dictionary conversion of the original text into numerical representations. To provide a theoretical upper bound of efficiency, we also implement TADOC on a pure DRAM platform. We measure the performance of N-TADOC separately for phase-level persistence and operation-level persistence and compare the space savings of N-TADOC on DRAM. Furthermore, we include a comparison with TADOC implemented on SSD (Solid State Disk) and HDD (Hard-Disk Driver). This allows us to evaluate the performance of N-TADOC in comparison to an alternative NVM-based storage solution. All datasets used in our evaluation are assumed to be stored on disk and the time measurement includes IO time. We begin timing from the initialization phase of loading the dataset to writing the analytics results back to disk after the graph traversal. Following this methodology, we aim to provide a comprehensive evaluation of the performance and efficiency of N-TADOC on NVM devices compared to the baseline approach, the pure DRAM platform, N-TADOC on SSD and HDD. Additionally, we consider space savings achieved by N-TADOC on DRAM and hard disk. To provide further insights into the performance of N-TADOC, we break down the phase-level time taken by N-TADOC, highlighting the specific tasks and operations involved in the process. Furthermore, we optimize the traversal process of N-TADOC under different workloads to evaluate its performance under varying conditions.

Benchmark. We use the following six common text analytics tasks to measure the performance of N-TADOC and the baseline. These analytics tasks have been widely used in previous studies [19]–[21].

- *Word count* [19], [43], [44] is a fundamental algorithm in text analytics, widely employed in applications such as document classification, clustering, and theme identification. It calculates the total occurrences of each word in a given dataset, which may comprise multiple compressed files.
- *Sort* [19], [21], [45] is a common preprocessing step in various data analytics tasks and plays a crucial role in text analytics for organizing and arranging data systematically. The sort benchmark involves sorting the words of compressed files based on alphabetical order.
- *Term vector* [19], [20], [46] focuses on representing documents as vectors, where each term corresponds to a dimension in the vector space. This benchmark involves constructing term vectors for the content of compressed files by their most frequent words.
- *Inverted index* [19], [47], [48] is a crucial data structure used in search engines and text retrieval systems. This benchmark task involves building word-to-document indexing from the compressed files.
- *Sequence count* [19], [49], [50] is vital for applications such as identifying frequent n-grams, pattern recognition, and extracting sequential patterns from textual data. The sequence count benchmark aims to identify and count specific sequences of words in compressed text data.
- *Ranked inverted index* [19], [20], [51] is valuable for

efficient ranked retrieval in information retrieval systems, enabling quick access to relevant documents based on occurrence frequencies. The ranked inverted index benchmark utilizes the output of the sequence-count benchmark and produces a list per word-sequence in decreasing order of occurrence in the respective documents.

Platforms. In this paper, experiments are carried out on a single machine. This machine is configured with two Intel Xeon gold 5320 CPUs, 2TB Intel Optane persistent memory 200 series, 400GB Intel Optane SSD DC P5800X series SSD and 160TB DELL SAS RPM HDD. DRAM speed is 3200MT/s. The operating system is Ubuntu 20.04. We set the memory budget to 20% of the uncompressed dataset size to avoid full memory caching of NVM data.

Datasets. The datasets utilized in our evaluation are listed in Table I. These datasets have been obtained from various real-world workloads and have been extensively used in previous studies [1], [5]. Specifically, Dataset A contains COVID-19 data collected from Yelp [16]. Dataset B, consisting of a large number of small files, contains the NSF Research Award Abstracts (NSFRAA) and has been downloaded from the UCI Machine Learning Repository [17]. Dataset C comprises four web documents retrieved from Wikipedia [18], while Dataset D is a considerably large Wikipedia dataset.

TABLE I
DATASETS.

Dataset	File#	Rule#	Vocabulary Size
A	1	36,882	240,552
B	134,631	2,771,880	1,864,902
C	4	2,095,573	6,370,437
D	109	57,394,616	99,239,057

B. Performance

This part shows the N-TADOC overall speedup over uncompressed text analytics with the same persistence strategy, the gap between N-TADOC and the theoretical efficiency upper bound, speedups over SSD and HDD compressed text analytics.

Speedups over NVM uncompressed text analytics. In Figure 5, we demonstrate the speedup that N-TADOC provides in comparison to NVM uncompressed text analytics on four different datasets. Figure 5 (a) demonstrates that N-TADOC with phase-level persistence achieves 2.04× speedup, whereas Figure 5 (b) demonstrates that N-TADOC with operation-level persistence achieves a speedup of 1.40×. This is because the persistence strategy at the operation level introduces more overhead than the persistence at the phase level. For simplicity of discussion, the N-TADOC mentioned later is based on phase-level persistence strategy.

We have the following findings. First, the fact that N-TADOC is capable of achieving a large performance boost in the majority of cases demonstrates that data analytics directly on compressed files can be efficiently conducted on NVM. Second, when applied to dataset B, which is comprised of a large number of small files, N-TADOC achieves a moderate performance on the benchmark associated with file information such as *term vector* and *inverted index*. This is mainly due to that we use the bottom-up traversal method instead of the top-down traversal method to perform text analytics on dataset B. We preprocess to extract the word lists of all rules and cache those word lists on NVM, which introduces additional overhead. On the other hand, this is a significant improvement over the top-down traversal strategy, which is covered in the Section VI-E. This is because the top-down traversal strategy needs to traverse the DAG when processing each file, and the efficiency can be very low when the number of files is large.

Discrepancy to DRAM compressed text analytics. In Figure 6, we demonstrate that N-TADOC with phase-level persistence cost still has a certain discrepancy to the theoretical efficiency upper bound, which is TADOC running on a pure DRAM platform. As shown in Figure 6, N-TADOC is 1.59× slower than TADOC on average. We also make notable findings during the evaluation. First, N-TADOC's *word count* has the lowest performance, with an average slowdown of 2.26× when compared to TADOC. *Word count* is a simple benchmark with efficient implementation on DRAM involving basic counting operations. In contrast, in the case of developing TADOC on NVM, it introduces the complexity of memory management. This additional complexity results in a significant performance gap between N-TADOC and the theoretical upper bound in *word count*. Second, the slowdown effect is most noticeable on the smallest dataset A, with an average slowdown of 1.55×. Due to NVM's superior performance in handling large-scale data, the minimal size of the dataset may not fully leverage the performance advantages of NVM. As a result, N-TADOC might not achieve the theoretical performance on the smallest dataset. Third, the performance gap between N-TADOC and TADOC diminishes as the dataset size increases. In addition to the advantages of NVM over large dataset, N-TADOC is more likely to benefit from improved cache utilization for larger dataset. As the dataset size grows, the cache hit rate improves, leading to reduced memory latency and enhanced overall performance.

Speedups over SSD and HDD compressed text analytics. To evaluate the performance of N-TADOC on HDD compressed text analytics, we compare its speedups with respect to SSD-based and HDD-based approaches. We switch the file path from NVM to SSD/HDD while keeping the compression strategy intact. The following results demonstrate the performance improvements achieved by N-TADOC over SSD and HDD compressed text analytics. Figure 7 presents the speedup achieved by N-TADOC compared to SSD and HDD compressed text analytics using four different datasets. As shown in Figure 7, N-TADOC with phase-level persistence achieves an average speedup of 1.87× and 2.92× over N-TADOC for SSD and HDD. These results indicate that N-TADOC significantly outperforms SSD-based and HDD-based approaches in terms of speed. The utilization of NVM devices in N-TADOC enables faster data access and processing, resulting in improved performance for compressed text analytics

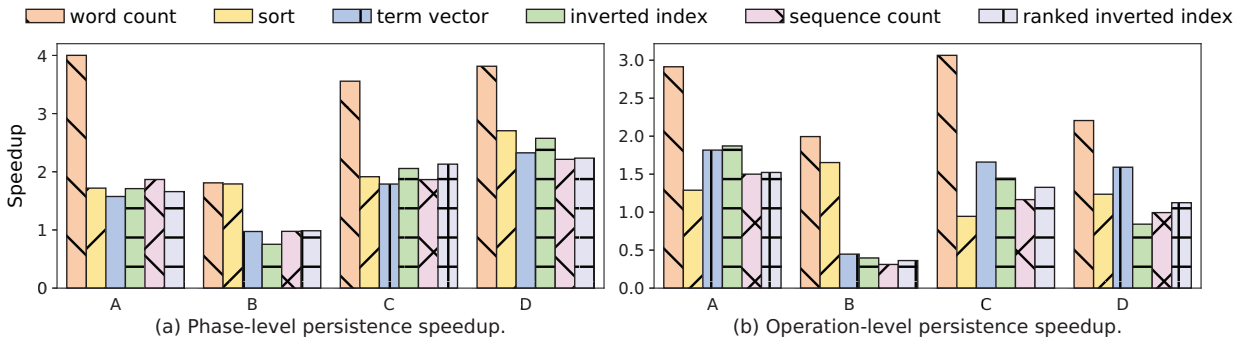

Fig. 5. Performance speedups over uncompressed text analytics.

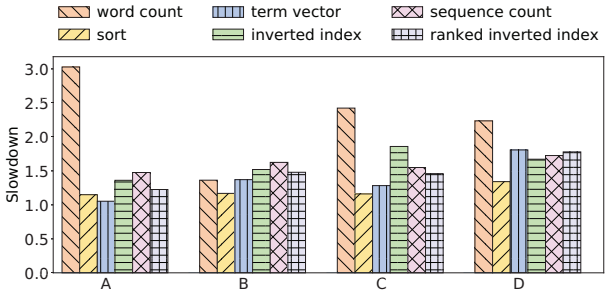

Fig. 6. Performance discrepancy to TADOC.

tasks. By leveraging the benefits of NVM, such as reduced latency and higher data transfer rates, N-TADOC demonstrates substantial speedups over SSD and HDD compressed text analytics.

These findings highlight the efficiency and performance advantages of N-TADOC on NVM devices compared to traditional NVM-based uncompressed and HDD-based compressed approaches. By leveraging the speed and reliability of NVM, N-TADOC offers substantial improvements in processing time and overall performance for compressed text analytics workloads.

C. DRAM Space Savings

In this part, we show the space savings achieved by N-TADOC on DRAM.

DRAM space savings were evaluated by measuring the difference in DRAM usage between TADOC and N-TADOC, with RSS (Resident Set Size) being used to indicate the physical memory size occupied by the process. RSS was measured using the *smem* command line tool, and the experiments conducted on all four datasets show significant DRAM space savings, with an average space saving of 70.7%. Furthermore, the DRAM space-saving ratio for datasets A, B, C, and D was found to be 65.6%, 70.7%, 72.2%, and 74.3%, respectively. The findings of this paper indicate that larger datasets tend to have a greater proportion of DRAM space savings as compared to smaller ones. This can be attributed to the fact that larger datasets occupy more NVM, which makes it more advantageous to use NVM resources instead of DRAM. Additionally, different benchmarks show varying DRAM space saving rates, with the *word count* task showing the highest rate at 79.8% and the *sequence count* task showing the lowest rate at only 60.7%. This observation provides an explanation for the largest difference in efficiency between N-TADOC and TADOC in the *word count* task, as shown in Figure 6. Overall, this trade-off between using more NVM to save DRAM and the resultant performance degradation needs to be considered while conducting data analytics directly on compressed files on NVM.

D. Time Breakdown

In this part, we analyze the time breakdown of the N-TADOC analytics process, focusing on the proportion of the initialization phase and the graph traversal phase, as well as the speedups achieved under different phases.

Time breakdown. Table II presents a detailed breakdown of the analytics time for datasets C and D. The results indicate that, for both datasets, the initialization phase takes up a larger proportion of the total time on average as the dataset size increases. For instance, on large dataset D, the initialization phase accounts for 73.4% of the total time on average for benchmarks such as *sequence count* and *ranked inverted index*. This is mainly due to the increased dataset size and the associated persistence cost. Moreover, the time proportion of the graph traversal phase is significantly higher for benchmark tests such as *sort* than *word count* on both datasets. This is because sorting the results by dictionary introduces additional overhead. The same reason applies to the difference in the proportion of phase time between benchmarks such as *inverted index* and *term vector*.

Speedups under different phases. The initialization phase starts with allocating space on the NVM and ends with reading the dictionary of compressed data. In contrast, the graph traversal phase starts with graph traversing and ends with merging results. The results show that the initialization phase and graph traversal phase achieve an average of 1.96× and 2.53× speedup, respectively, under different benchmark tests

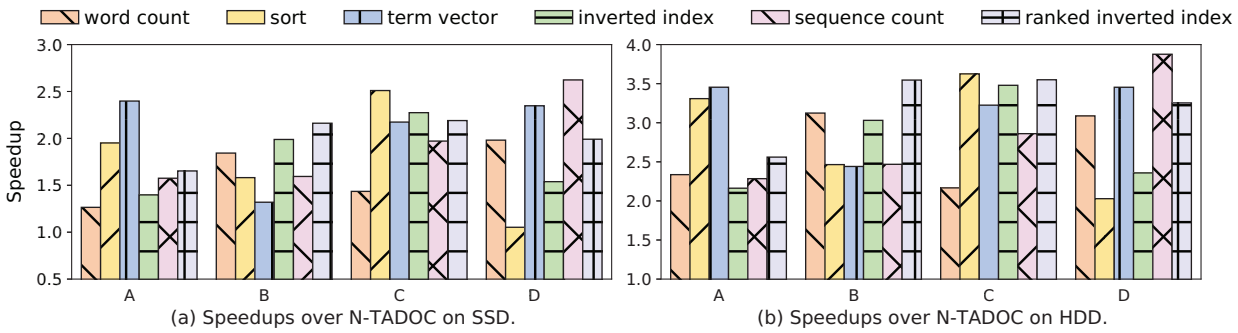

Fig. 7. Performance speedups over N-TADOC on SSD and HDD.

for dataset C. On the other hand, for dataset D, the initialization phase and graph traversal phase achieve an average of 1.23× and 2.87× speedup, respectively. Moreover, the graph traversal phase's speedup ratio is higher than the overall speedup, indicating that the acceleration effect of N-TADOC is mostly achieved in this phase.

TABLE II
TIME BREAKDOWN FOR DATASETS C AND D (SECONDS).

Dataset	Benchmark	Initial phase	Traversal phase
C	word count	2.70	1.36
	sort	6.47	9.39
	term vector	7.60	7.11
	inverted index	7.30	2.86
	sequence count	7.48	17.51
	ranked inverted index	7.45	19.49
D	word count	225.31	23.76
	sort	271.43	295.83
	term vector	126.40	273.77
	inverted index	125.61	168.38
	sequence count	1107.51	308.08
	ranked inverted index	1187.58	544.78

E. Traversal Optimization under Different Workloads

We continue our investigation into how N-TADOC performs in a variety of different workloads. The impact of workload on performance improvement is mostly comprised of two types of graph traversal: top-down traversal and bottom-up traversal. The top-down traversal of a graph is more appropriate for datasets that have a limited number of rules or files before compression. The bottom-up traversal of a graph is more appropriate for datasets that have a limited number of components in the root node after compression. When using the top-down traversal strategy, the program is required to traverse the DAG in order to retrieve the weight of rules for each file. During the graph traversal phase, the performance of the algorithm can suffer severely if the number of rules or documents is high and the top-down traversal method is used, as is the case with dataset B. In dataset B, we evaluate how effective the top-down and bottom-up approaches are in comparison to one another. On average, the efficiency of the top-down method is approximately 1000× lower than that of the bottom-up method. As a consequence of this, the top-down method does not make effective use of NVM to cache the rule's word list. Instead, it chooses to traverse the DAG for each file individually for weight propagation, which results in low efficiency. After acquiring the word list for each rule during the bottom-up traversal procedure, the program then scans the root node to collect the final results. This is done in order to complete the process. Bottom-up traversal has the potential to achieve higher performance than top-down traversal in certain circumstances. These scenarios include the root node having a short length or there being an excessive number of files and rules.

F. Discussion

This part discusses N-TADOC's specialized optimization, application scope, limitations and vision for the future.

Specialized optimization. N-TADOC is specifically designed and optimized for NVM, taking advantage of its non-volatile persistence characteristic. As to persistence, it does not need to be considered on DRAM, but is a critical and unique concern on NVM. We also run the code originally designed for DRAM directly on NVM and find that the efficiency is not high. These techniques cannot be directly applicable to disk-based storage media such as HDD or SSD too. This limitation primarily arises from the heavy reliance of N-TADOC on random access, which aligns with the performance characteristics and access patterns of NVM but does not exhibit the same level of efficiency on HDD or SSD.

Application scope. N-TADOC is designed to benefit text analytics workloads that heavily rely on dynamic data structures and operations. These workloads can include tasks such as document indexing and query processing. By leveraging NVM's characteristics, N-TADOC aims to optimize the performance and efficiency of these operations for text analytics applications.

Limitations. In small-scale text analytics tasks, the size of the input text can limit the effectiveness of N-TADOC due to insufficient data to fully utilize the NVM resources. The limited amount of data stored in NVM can result in decreased

performance improvements in small-scale tasks. Additionally, the cost of initial data transmission and data organization on NVM can become a bottleneck in small-scale applications, which can diminish the potential performance gains of using NVM-based solutions. This is because the time and resources required to transfer data and organize it on NVM can become a significant portion of the overall analytics time, potentially offsetting any performance gains achieved by using NVM. Therefore, while N-TADOC has demonstrated its effectiveness in large-scale text analytics tasks, it may not be the optimal solution for small-scale applications.

Cross-evaluation. In order to thoroughly evaluate the performance and effectiveness of N-TADOC in comparison to TADOC, it is crucial to conduct cross-evaluation. In our experiments, N-TADOC on NVM achieves a 5× speedup over TADOC on NVM. Comparing the results of N-TADOC and TADOC in the NVM environment, we observe the advantages and improvements offered by N-TADOC.

Vision for the future. The discontinuation of Intel Optane [52] is undoubtedly disappointing news. However, it is worth noting that Intel Optane is not the only non-volatile memory, NVM-based architecture, available in the market. There are other NVM architectures that offer similar or even better features than Optane. While the experiments conducted so far have focused on Intel Optane due to its large-scale commercial availability, there is a plan to migrate N-TADOC to other NVM-based architectures in the future. Resistive Random Access Memory (ReRAM) [31] and Phase Change Memory (PCM) [32] are among the potential candidates for this migration. The migration of N-TADOC to different NVM architectures will enable researchers to explore and compare the performance of N-TADOC on different platforms. This can not only help in identifying the strengths and weaknesses of each architecture but also provide an opportunity for optimization and improvement of N-TADOC. The exploration of N-TADOC on other NVM architectures could also lead to the discovery of new opportunities for improvement and optimization. For example, a different architecture may provide better compression or faster access times, leading to more efficient and faster processing of large text corpora. Additionally, techniques such as data compression and caching can be explored to further optimize the storage and retrieval of compressed text data in NVM, balancing the trade-off between space efficiency and analytics speed.

VII. Related Work

According to our understanding, N-TADOC is the first work that enables NVM-based text analytics without decompression. In this section, we show the related work of grammar compression, NVM data analytics, and embedded NVM systems.

Grammar compression. There has been a significant amount of research dedicated to compressing grammatical structures [1]–[9], [53]–[55]. For example, Text Analytics Directly on Compression (TADOC) [3] uses grammatical structures for compression, and can be used to perform a variety of document analytics. Pan et al. [5] enable TADOC on GPUs, resulting in a significant improvement in the performance of TADOC. CompressDB [9] is a storage engine that supports data processing on compressed data without decompression, achieving improved throughput and latency reduction for various database systems. CompressGraph [6] leverages redundancy in repeated neighbor sequences to achieve performance boost and space reduction in graph applications. These works has further expanded the potential uses for TADOC, making it a more efficient and scalable solution for processing large volumes of data.

NVM data analytics. NVMs have found applications in several subfields of data analytics, including structured data analytics [56]–[60], graph analytics [12], and machine learning analytics [61]–[63]. Alibaba's storage engine X-engine [56] in cloud-native database [57] uses LSM-tree design and adaptive separation of hot data records and cold data records; they store hot data on NVM/SSD and cold data on SSD/HDD, improving write throughput and read throughput. Malicevic et al. [12] investigate the effect that switching from DRAM to NVM has on the most recent and cutting-edge graph processing frameworks. A lightweight version of random shuffling known as LIRS [61] has been developed for use in machine learning analytics in order to shuffle training set using NVM devices.

Embedded NVM systems. Embedded NVM systems have gained significant popularity in recent times due to their high efficiency and low cost. To overcome the challenges associated with NVM embedded systems, researchers have been conducting extensive research. Two key issues that researchers have been focusing on are write endurance and state retention [64], [65]. N-TADOC utilizes a dual-layer structure consisting of DRAM and NVM. By leveraging NVM reads and performing certain computations, N-TADOC reduces the write operations on NVM during text analytics tasks to improve write endurance. It also reduces state retention overhead through different persistence strategies.

VIII. Conclusion

In this paper, we have presented N-TADOC, which enables efficient NVM-based text analytics without decompression. We show the challenges of data locality and redundant access overhead of data objects growth on NVM , and present a series of solutions to solve these challenges. N-TADOC is 2.04× faster than text analytics on the uncompressed datasets with NVM. With a moderate performance decrease, N-TADOC saves 70.7% of DRAM space compared with TADOC.

ACKNOWLEDGMENTS

This work is supported by the National Key R&D Program of China under Grant (No. 2023YFB4503603), National Natural Science Foundation of China (62172419, U1911203, and 62322213), and Beijing Nova Program (20220484137 and 20230484397). Feng Zhang is the corresponding author of this paper.

REFERENCES

[1] F. Zhang, J. Zhai, X. Shen, D. Wang, Z. Chen, O. Mutlu, W. Chen, and X. Du, "TADOC: Text analytics directly on compression," *The VLDB Journal*, 2021.

[2] F. Zhang, J. Zhai, X. Shen, O. Mutlu, and W. Chen, "Zwift: A Programming Framework for High Performance Text Analytics on Compressed Data," in *ICS*, 2018.

[3] F. Zhang, J. Zhai, X. Shen, O. Mutlu, and W. Chen, "Efficient Document Analytics on Compressed Data: Method, Challenges, Algorithms, Insights," *Proc. VLDB Endow.*, 2018.

[4] F. Zhang, J. Zhai, X. Shen, O. Mutlu, and X. Du, "Enabling Efficient Random Access to Hierarchically-Compressed Data," in *ICDE*, 2020.

[5] F. Zhang, Z. Pan, Y. Zhou, J. Zhai, X. Shen, O. Mutlu, and X. Du, "G-TADOC: Enabling Efficient GPU-Based Text Analytics without Decompression," in *ICDE*, 2021.

[6] Z. Chen, F. Zhang, J. Guan, J. Zhai, X. Shen, H. Zhang, W. Shu, and X. Du, "Compressgraph: Efficient parallel graph analytics with rule-based compression," *Proc. ACM Manag. Data*, vol. 1, no. 1, pp. 4:1–4:31, 2023.

[7] Y. Zhang, F. Zhang, H. Li, S. Zhang, and X. Du, "Compressstreamdb: Fine-grained adaptive stream processing without decompression," in *ICDE*. IEEE, 2023, pp. 408–422.

[8] Y. Liu, F. Zhang, Z. Pan, X. Guo, Y. Hu, X. Zhang, and X. Du, "Compressed data direct computing for chinese dataset on dcu," *CCF Transactions on High Performance Computing*, pp. 1–15, 2023.

[9] W. Wan, F. Zhang, C. Zhang, M. Zhang, J. Zhai, Y. Chai, H. Zhang, W. Lu, Y. Chen, H. Li *et al.*, "Compressed data direct computing for databases," *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[10] "Intel Optane Memory," https://www.intel.com/content/www/us/en/products/details/memory-storage/optane-memory.html.

[11] M. Dong, H. Bu, J. Yi, B. Dong, and H. Chen, "Performance and protection in the ZoFS user-space NVM file system," in *SOSP*, 2019.

[12] J. Malicevic, S. Dulloor, N. Sundaram, N. Satish, J. Jackson, and W. Zwaenepoel, "Exploiting NVM in large-scale graph analytics," in *INFLOW@SOSP*, 2015.

[13] A. Eisenman, D. Gardner, I. AbdelRahman, J. Axboe, S. Dong, K. M. Hazelwood, C. Petersen, A. Cidon, and S. Katti, "Reducing DRAM footprint with NVM in facebook," in *EuroSys*, 2018.

[14] V. Mironov, I. G. Chernykh, I. M. Kulikov, A. A. Moskovsky, E. Epifanovsky, and A. Kudryavtsev, "Performance Evaluation of the Intel Optane DC Memory With Scientific Benchmarks," in *MCHPC@SC*, 2019.

[15] Z. Pan, F. Zhang, Y. Zhou, J. Zhai, X. Shen, O. Mutlu, and X. Du, "Exploring Data Analytics Without Decompression on Embedded GPU Systems," *IEEE Trans. Parallel Distributed Syst.*, 2022.

[16] "COVID-19 Data from Yelp Open Dataset," https://www.yelp.com/dataset/, 2019.

[17] A. Asuncion and D. Newman, "UCI machine learning repository," 2007.

[18] "Wikipedia HTML data dumps," https://dumps.wikimedia.org/enwiki/, 2017.

[19] F. Ahmad, S. Lee, M. Thottethodi, and T. Vijaykumar, "Puma: Purdue mapreduce benchmarks suite," 2012.

[20] F. Zhang and M. F. Sakr, "Performance variations in resource scaling for mapreduce applications on private and public clouds," in *IEEE CLOUD*. IEEE Computer Society, 2014, pp. 456–465.

[21] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. N. Vijaykumar, "Tarazu: optimizing mapreduce on heterogeneous clusters," in *ASPLOS*. ACM, 2012, pp. 61–74.

[22] C. G. Nevill-Manning and I. H. Witten, "Inferring sequential structure," 1996.

[23] C. G. Nevill-Manning and I. H. Witten, "Identifying Hierarchical Structure in Sequences: A linear-time algorithm," *CoRR*, vol. cs.AI/9709102, 1997.

[24] C. G. Nevill-Manning and I. H. Witten, "Linear-time, incremental hierarchy inference for compression," in *Data Compression Conference*. IEEE Computer Society, 1997.

[25] C. J. Xue, Y. Zhang, Y. Chen, G. Sun, J. J. Yang, and H. Li, "Emerging non-volatile memories: opportunities and challenges," in *CODES+ISSS*, 2011.

[26] A. Shanbhag, N. Tatbul, D. Cohen, and S. Madden, "Large-scale in-memory analytics on Intel Optane DC persistent memory," in *DaMoN*, 2020.

[27] O. Patil, L. Ionkov, J. Lee, F. Mueller, and M. Lang, "Performance characterization of a DRAM-NVM hybrid memory architecture for HPC applications using intel optane DC persistent memory modules," in *MEMSYS*, 2019.

[28] K. Wu, F. Ober, S. Hamlin, and D. Li, "Early Evaluation of Intel Optane Non-Volatile Memory with HPC I/O Workloads," *CoRR*, vol. abs/1708.02199, 2017.

[29] M. I. Seltzer, V. J. Marathe, and S. Byan, "An NVM carol: Visions of NVM past, present, and future," in *ICDE*. IEEE Computer Society, 2018, pp. 15–23.

[30] J. Arulraj and A. Pavlo, "How to Build a Non-Volatile Memory Database Management System," in *SIGMOD*, 2017.

[31] Y. Chen, "Reram: History, status, and future," *IEEE Transactions on Electron Devices*, vol. 67, no. 4, pp. 1420–1433, 2020.

[32] R. Bez, "Chalcogenide pcm: A memory technology for next decade," in *2009 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2009, pp. 1–4.

[33] J. Arulraj, "Data management on non-volatile memory," in *SIGMOD Conference*. ACM, 2019, p. 1114.

[34] I. Oukid and W. Lehner, "Data structure engineering for byte-addressable non-volatile memory," in *SIGMOD Conference*. ACM, 2017, pp. 1759–1764.

[35] "PMDK," https://pmdk.io/, 2022.

[36] K. Kambatla, G. Kollias, V. Kumar, and A. Grama, "Trends in big data analytics," *J. Parallel Distributed Comput.*, vol. 74, no. 7, pp. 2561–2573, 2014.

[37] T. E. Anderson, M. Canini, J. Kim, D. Kostic, Y. Kwon, S. Peter, W. Reda, H. N. Schuh, and E. Witchel, "Assise: Performance and Availability via Client-local NVM in a Distributed File System," in *OSDI*, 2020.

[38] M. Chen, J. Han, and P. S. Yu, "Data Mining: An Overview from a Database Perspective," *IEEE Trans. Knowl. Data Eng.*, 1996.

[39] A. Chen, "A review of emerging non-volatile memory (NVM) technologies and applications," *Solid-State Electronics*, 2016.

[40] A. S. Memaripour and S. Swanson, "Breeze: User-level access to non-volatile main memories for legacy software," in *ICCD*. IEEE Computer Society, 2018, pp. 413–422.

[41] W. Cai, H. Wen, H. A. Beadle, C. Kjellqvist, M. Hedayati, and M. L. Scott, "Understanding and optimizing persistent memory allocation," in *ISMM*. ACM, 2020, pp. 60–73.

[42] D. Bittman, P. Alvaro, P. Mehra, D. D. E. Long, and E. L. Miller, "Twizzler: A *Data-centric* OS for non-volatile memory," *ACM Trans. Storage*, vol. 17, no. 2, pp. 11:1–11:31, 2021.

[43] J. E. Blumenstock, "Size matters: word count as a measure of quality on wikipedia," in *WWW*. ACM, 2008, pp. 1095–1096.

[44] J. W. Pennebaker, M. E. Francis, and R. J. Booth, "Linguistic inquiry and word count: Liwc 2001," *Mahway: Lawrence Erlbaum Associates*, vol. 71, no. 2001, p. 2001, 2001.

[45] N. K. Govindaraju, N. Raghuvanshi, M. Henson, and D. Manocha, "A cache-efficient sorting algorithm for database and data mining computations using graphics processors," *University of North Carolina, Tech. Rep*, 2005.

[46] R. Stata, K. Bharat, and F. Maghoul, "The term vector database: fast access to indexing terms for web pages," *Computer Networks*, vol. 33, no. 1-6, pp. 247–255, 2000.

[47] H. Yan, S. Ding, and T. Suel, "Inverted index compression and query processing with optimized document ordering," in *WWW*. ACM, 2009, pp. 401–410.

[48] D. R. Cutting and J. O. Pedersen, "Optimizations for dynamic inverted index maintenance," in *SIGIR*. ACM, 1990, pp. 405–411.

[49] L. Lebart, "Classification problems in text analysis and information retrieval," in *Advances in Data Science and Classification: Proceedings of the 6th Conference of the International Federation of Classification Societies (IFCS-98) Università "La Sapienza", Rome, 21–24 July, 1998*. Springer, 1998, pp. 465–472.

[50] U. Zernik, *Lexical acquisition: exploiting on-line resources to build a lexicon*. Psychology Press, 1991.

[51] J.-F. Yeh, S.-F. Li, M.-R. Wu, W.-Y. Chen, and M.-C. Su, "Chinese word spelling correction based on n-gram ranked inverted index list," in *Proceedings of the Seventh SIGHAN Workshop on Chinese Language Processing*, 2013, pp. 43–48.

[52] "Intel Optane Business Update: What Does This Mean for Warranty and Support," https://www.intel.com/content/www/us/en/support/articles/000091826/memory-and-storage.html, 2022.

[53] S. Maneth and F. Peternek, "Compressing graphs by grammars," in *ICDE*. IEEE Computer Society, 2016, pp. 109–120.
[54] P. Ferragina, G. Manzini, T. Gagie, D. Köppl, G. Navarro, M. Striani, and F. Tosoni, "Improving matrix-vector multiplication via lossless grammar-compressed matrices," *Proc. VLDB Endow.*, vol. 15, no. 10, pp. 2175–2187, 2022.
[55] X. Huang, Y. Dong, G. Ye, and Y. Shi, "Meaningful image encryption algorithm based on compressive sensing and integer wavelet transform," *Frontiers Comput. Sci.*, vol. 17, no. 3, p. 173804, 2023.
[56] G. Huang, X. Cheng, J. Wang, Y. Wang, D. He, T. Zhang, F. Li, S. Wang, W. Cao, and Q. Li, "X-engine: An optimized storage engine for large-scale e-commerce transaction processing," in *SIGMOD Conference*. ACM, 2019, pp. 651–665.
[57] F. Li, "Cloud native database systems at alibaba: Opportunities and challenges," *Proc. VLDB Endow.*, vol. 12, no. 12, pp. 2263–2272, 2019. [Online]. Available: http://www.vldb.org/pvldb/vol12/p2263-li.pdf
[58] P. Götze, S. Baumann, and K. Sattler, "An nvm-aware storage layout for analytical workloads," in *ICDE Workshops*. IEEE Computer Society, 2018, pp. 110–115.
[59] Y. Luo, P. Jin, Q. Zhang, and B. Cheng, "Tlbtree: A read/write-optimized tree index for non-volatile memory," in *ICDE*. IEEE, 2021, pp. 1889–1894.
[60] T. Wang, J. J. Levandoski, and P. Larson, "Easy lock-free indexing in non-volatile memory," in *ICDE*. IEEE Computer Society, 2018, pp. 461–472.
[61] Z. Ke, H. Cheng, and C. Yang, "LIRS: enabling efficient machine learning on NVM-based storage via a lightweight implementation of random shuffling," *CoRR*, vol. abs/1810.04509, 2018.
[62] L. Chen and S. Chen, "How does updatable learned index perform on non-volatile main memory?" in *ICDE Workshops*. IEEE, 2021, pp. 66–71.
[63] Z. Zhang, Z. Chu, P. Jin, Y. Luo, X. Xie, S. Wan, Y. Luo, X. Wu, P. Zou, C. Zheng, G. Wu, and A. Rudoff, "PLIN: A persistent learned index for non-volatile memory with high performance and instant recovery," *Proc. VLDB Endow.*, vol. 16, no. 2, pp. 243–255, 2022.
[64] Z. Wang, Z. Gu, M. Yao, and Z. Shao, "Endurance-aware allocation of data variables on nvm-based scratchpad memory in real-time embedded systems," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 34, no. 10, pp. 1600–1612, 2015.
[65] T. D. Verykios, D. Balsamo, and G. V. Merrett, "Selective policies for efficient state retention in transiently-powered embedded systems: Exploiting properties of NVM technologies," *Sustain. Comput. Informatics Syst.*, vol. 22, pp. 167–178, 2019.