electronics

Article

ClusteredLog: Optimizing Log Structures for Efficient Data Recovery and Integrity Management in Database Systems

Mariha Siddika Ahmad *,† and Brajendra Nath Panda *,†

Department of Electrical Engineering and Computer Science, University of Arkansas, Fayetteville, AR 72701, USA
* Correspondence: ma135@uark.edu (M.S.A.); bpanda@uark.edu (B.N.P.)
† These authors contributed equally to this work.

Abstract: In modern database systems, efficient log management is crucial for ensuring data integrity and facilitating swift recovery from potential data corruption or system failures. Traditional log structures, which store operations sequentially as they occur, often lead to significant delays in accessing and recovering specific data objects due to their scattered nature across the log. ClusteredLog addresses the limitations of traditional logging methods by implementing a novel logical organization of log entries. Instead of simply storing operations sequentially, it groups related operations for each data item into clusters. As a result, ClusteredLog enables faster identification and recovery of damaged data items and thus reduces the need for extensive log scanning, improving overall efficiency in database recovery processes. We introduce data structures and algorithms that facilitate the creation of these clustered logs, which also track dependencies and update operations on data items. Simulation studies demonstrate that our clustered log method significantly accelerates damage assessment and recovery times compared to traditional sequential logs, particularly as the number of transactions and data items increases. This optimization is pivotal for maintaining data integrity and operational efficiency in databases, especially in scenarios involving potential malicious modifications.

Keywords: cluster; database management; log optimization

Citation: Ahmad, M.S.; Panda, B.N. ClusteredLog: Optimizing Log Structures for Efficient Data Recovery and Integrity Management in Database Systems. *Electronics* **2024**, *13*, 4723. https://doi.org/10.3390/electronics13234723

Academic Editors: Boni García, Mario Munoz-Organero, Patricia Callejo and Miguel Ángel Hombrados

Received: 1 October 2024
Revised: 19 November 2024
Accepted: 21 November 2024
Published: 29 November 2024

1. Introduction

In modern database systems, efficient log management is crucial for ensuring data integrity and facilitating swift recovery from system failures or malicious attacks. Traditional log structures, which store operations sequentially, often lead to significant delays in accessing and recovering specific data objects due to their scattered nature across the log. This paper introduces ClusteredLog, a novel logical organization of log structures that clusters related operations for each data item, thereby enhancing the speed and efficiency of data recovery processes.

The importance of efficient damage assessment and recovery in database systems cannot be overstated, particularly in critical environments such as financial institutions, healthcare facilities, or air traffic control systems. In these scenarios, quickly identifying and isolating damaged data is paramount, as prolonged downtime or full system shutdowns are often not viable options. Traditional methods of addressing data integrity issues frequently involve scanning entire log files. This approach becomes increasingly time-intensive and resource-demanding as databases scale and complexity escalates [1–3]

ClusteredLog addresses these challenges by introducing an innovative logical organization that groups related operations for each data item, creating virtual clusters within the log structure. This method reduces the need for extensive log scanning and enables quicker identification and recovery of damaged data items. By grouping related operations together, our approach addresses the limitations of traditional sequential logs. This approach proves particularly effective in scenarios involving potential malicious modifications or system failures.

Furthermore, ClusteredLog's design takes into account the realities of modern database systems, where multiple processes responsible for various tasks append information to logs concurrently. Our approach maintains the physical sequential nature of log writing while providing a logical structure that facilitates rapid access to related operations. This balance ensures that the benefits of clustered organization are achieved without compromising the fundamental principles of log management in database systems.

The key contributions of this paper include the following:

1. A detailed description of the ClusteredLog data structure and its implementation.
2. Algorithms for creating clustered log structures and performing efficient damage assessment.
3. A comprehensive simulation study demonstrating the performance benefits of ClusteredLog compared to traditional sequential logs.

Our results show that ClusteredLog significantly accelerates damage assessment and recovery times, especially as the number of transactions and data items increases. This optimization is crucial for ensuring data integrity and efficiency in modern databases, where quick recovery from failures or attacks is vital. ClusteredLog enables targeted recovery without full system shutdowns, making it especially valuable in environments that require continuous operation.

In the following sections, we present the literature review, detail the ClusteredLog method, explain the damage assessment process, and provide an in-depth analysis of our simulation results. We conclude by discussing the implications of our findings and potential future directions for research in this area. This includes considerations for implementation in various database environments and potential optimizations for specific use cases.

2. Literature Review

The use of logs as a primary data source for anomaly detection in large-scale distributed systems has been extensively researched. This approach is particularly valuable because logs provide a rich source of real-time data about system operations, making them ideal for analysis and anomaly detection. Logs capture detailed runtime information, which can be processed and analyzed using machine learning techniques to detect anomalies and reduce manual inspection [4]. For instance, metaheuristic optimization algorithms like Gray Wolf Optimization and Glowworm Optimization have been employed for clustering log data, efficiently identifying patterns and anomalies in database queries [5]. Furthermore, the DIMAQS (Dynamic Identification of Malicious Query Sequences) system employs real-time monitoring of query sequences to detect ransomware attacks, using Colored Petri Nets and deep neural networks to model and detect malicious activities [6].

Transaction-level and inter-transaction-level detection methods are vital for enhancing database security. These methods help identify malicious activities that traditional methods may overlook. Transaction-level detection employs specification-based approaches to flag deviations from predefined transaction rules, while inter-transaction-level detection uncovers unusual patterns across multiple transactions through anomaly detection techniques. This hybrid system reduces false positives and negatives, offering comprehensive protection [7]. Similar hybrid methods employing role- and behavior-based risk assessments have been demonstrated to improve database security [8].

Log-based anomaly detection has also been improved through techniques such as source code analysis and pattern mining. Drill, for example, enhances anomaly detection in large-scale storage systems by using source code analysis to extract context from log statements and applying pattern mining to detect deviations from normal behavior [9]. Another study utilizes expectation maximization clustering and sequential pattern mining to detect intrusive transactions in databases, uncovering recurring patterns that help differentiate between normal and malicious behavior [10].

Transformer models have shown promise in log anomaly detection for large-scale storage systems by leveraging pre-trained models on large-scale unlabeled data. These models improve anomaly detection accuracy by effectively identifying abnormal behaviors

in complex systems [11]. Additionally, LogGPT uses natural language models like GPT to model normal log behavior and capture complex patterns and dependencies for more accurate anomaly detection [12]. In a related development, automated log labeling and root cause analysis have been advanced through techniques like tokenization, template extraction, and embedding, enabling systems to process logs more efficiently [13].

Real-time log analysis systems such as LogLens and LogCluster employ clustering techniques to discover patterns and facilitate efficient log-to-pattern mappings. These systems cluster similar logs and generate representative log sequences to enhance log parsing and anomaly detection [14]. Moreover, techniques such as LSTM models have been used to capture contextual features in log sequences, further improving anomaly detection accuracy by modeling both sequential and quantitative relationships within logs [15,16].

Several other systems have advanced log analysis for cybersecurity and cloud environments. For example, CLP (Compressed Log Processor) uses pattern mining and two-level variable dictionaries to enhance log compression and search efficiency [17]. Systems like Log2Graphs use graph-based approaches for unsupervised anomaly detection, representing log sequences' events as nodes in a graph and capturing temporal and causal dependencies as edges between events [18]. Another study models web application vulnerabilities using graph-based techniques, integrating log analysis with attack pattern detection through subgraph matching [19]. This graph-based methodology has also been extended to tasks like template recognition using large log datasets [20].

Tools such as LogAnomaly and LogAssist have contributed to improving log analysis by employing sophisticated models and pattern extraction techniques. LogAnomaly uses LSTM networks and Transformer models for both sequential and quantitative anomaly detection, while LogAssist applies log summarization and workflow compression techniques to reduce the volume of logs and enable efficient analysis [16,21]. The Iterative Structure Extraction (ISE) algorithm, used in Logzip, enhances log compression by clustering and discovering hidden log patterns, providing efficient processing of large log datasets [22].

User behavior risk assessment in Zero Trust environments has also emerged as a critical area for anomaly detection. One study [23] uses log analysis and pattern mining to detect outliers in user behaviors, grouping similar activities to identify security risks. Similarly, duplicate logging statements and their relationship to code clones have been studied to improve logging practices.This research identifies anti-patterns in log design that can impact maintainability and security [24].

Graph-based incident aggregation for large-scale online service systems focuses on tracking failure impacts across interconnected services. Dependency tracking is used to capture relationships between services to identify cascading effects of system failures [25]. In cloud systems, tools like Prism employ clustering strategies to reveal hidden functional clusters. This enhances observability by processing large volumes of cloud instance data and uncovering hidden relationships [26].

Multi-source log clustering methods have also been applied to distributed systems, enabling the clustering of related log statements from different sources. These methods compare log variables and static elements to cluster logs across distributed systems. This process identifies common patterns, independent of log format or source [27].

Clustering-based methods like LogMine continue to push the boundaries of log analysis by extracting templates and patterns from large datasets using hierarchical clustering, tokenization, and type detection [28]. This enables faster and more scalable pattern recognition, which is crucial for real-time anomaly detection in distributed environments. Other clustering approaches focus on cybersecurity applications, grouping log sequences to detect outliers that may indicate security risks. For example, a survey on log clustering methods highlighted various techniques for detecting anomalies in cybersecurity environments, providing insight into handling evolving log patterns [29].

Several hybrid methods have emerged to address the challenge of log signature generation and anomaly detection. A hybrid approach using density-based spatial clustering and Named Entity Recognition (NER) efficiently extracts log signatures from heterogeneous

log data [30]. Other methods, like CloudRAID, focus on detecting concurrency bugs in distributed systems by mining logs to identify problematic message orderings, demonstrating the versatility of log mining techniques for identifying complex system issues [31]. Another method is TRACEMESH, which uses streaming sampling techniques to cluster distributed traces in real time [32]. These systems focus on detecting bugs and performance bottlenecks in large-scale environments, where traditional methods fall short.

Logs play a vital role in damage assessment and recovery, as they provide a detailed record of transactions that allows for the precise identification of compromised data. A log-based damage assessment model for fog computing systems is presented in [33] that utilizes transaction logs to trace the lineage of data affected by blind writes. By analyzing logs, the model constructs a Blind Write Set (BWSi) and Children Data Set (CDSi) to identify compromised data items efficiently, enabling rapid recovery. Further exploration of this log-centric approach is presented in [34], where the blind write lineage model is tested through a simulation that analyzes transaction logs under various parameters, such as the number of transactions and blind writes. The study demonstrates the model's effectiveness in using logs to quickly assess damage and initiate recovery, outperforming traditional methods.

In [35], two models are presented for preventing insider threats: a log-based model and a dependency graph-based model. The log-based model tracks all operations on data items and employs a threshold mechanism to limit modifications. If changes exceed these thresholds, the model triggers a verification process to examine the transactions. And this enables real-time detection of suspicious activities. Together, these models enhance security by mitigating risks from malicious updates and improving accountability within the database system.

Additionally, ref. [36] utilizes multiple agents for parallel damage assessment and recovery using versioned data items based on timestamps. By linking update expressions to specific data item versions, the approach minimizes log access and focuses on identifying only damaged versions for recovery. This targeted method avoids unnecessary operations on unaffected transactions, enhancing efficiency and precision in damage assessment and recovery while reducing system downtime and maintaining data integrity.

Furthermore, ref. [37] involves creating three graph representations of the system (Possible Paths Graph, Active Paths Graph, and Damage Spread Graph), using adjacency and time–frequency matrices to model data dependencies and update frequencies. The approach employs algorithms to estimate damage spread based on minimum update frequencies, assess actual damage, and prioritize data object repairs using criticality, repair time, and centrality metrics. This model allows for quick damage assessment, efficient recovery scheduling, and the ability to make unaffected parts of the system accessible during the recovery process.

Recent research has explored novel approaches to optimize damage assessment and recovery processes in large-scale cloud databases. Ref. [38] proposes a model that accelerates damage assessment and recovery of large, interdependent datasets in cloud systems by quickly categorizing data items into three zones: damaged, undamaged, and gray (uncertain). This is achieved through a directed graph representation of data dependencies, analysis of update timestamps, and a zoning procedure. The model first identifies and releases undamaged data for immediate use, then determines damaged items, and finally subjects items in the gray zone to detailed assessment. This approach allows for faster resumption of critical operations while damage assessment and recovery continue, optimizing the process compared to traditional methods that require comprehensive assessment of all data items from the start.

Ref. [39] provides a high-performance transition-dependency-based damage assessment technique that saves the dependencies between transactions using a matrix. They simply used one matrix; therefore, no additional data structures were required. According to the experimental data, the transaction dependence strategy using a matrix performs well. In order to reduce time, they sacrifice a small amount of memory in their algorithm. The

attacker's purpose is usually denial of service. As a result, they concentrate on shortening the recuperation procedure.

Additionally, the authors of [40] proposed a hash-based assessment and recovery model for healthcare management systems, offering a novel approach to minimizing service disruption caused by malicious transactions or data corruption. By utilizing a hash table as the primary data structure, the model achieves fast access and search times, significantly improving the efficiency of damage assessment and recovery processes. The damage assessment algorithm employs a transaction dependency paradigm to categorize affected transactions, while the recovery algorithm strategically undoes malicious transactions and re-executes affected ones.

On the other hand, ARIES (Algorithm for Recovery and Isolation Exploiting Semantics) [41] is a pioneering transaction recovery method using write-ahead logging, fine-grained locking, and partial rollbacks. It employs LSNs to track page states and logged updates, enabling a "repeating history" recovery approach. Unlike ARIES, our approach optimizes log structures specifically for faster data recovery and integrity management in database systems, utilizing advanced clustering and pattern mining techniques to improve efficiency and reduce recovery time.

In summary, advancements in log analysis techniques, such as clustering, deep learning, and graph-based modeling, have significantly improved the detection of anomalies, malicious transactions, system vulnerabilities, and damage assessment and recovery. These methods form the backbone of modern database intrusion detection systems, offering scalable, accurate, and efficient solutions for ensuring database security.

3. ClusteredLog Method

The model we intend to develop focuses on organizing logs to facilitate swift access to operations performed on data objects. Without an effective logging mechanism, it becomes exceedingly difficult to determine precisely what occurred and how to recover from any issues. Therefore, it is imperative to store detailed information about the program logic to ensure accurate recovery of affected systems.

However, logs are typically sequential files, and multiple processes responsible for various tasks append information to them. As a result, details regarding operations on specific data objects tend to be scattered throughout the log. Consequently, retrieving a particular set of (potentially damaged) data objects requires accessing different parts of the log, leading to significant delays.

To address this challenge, we plan to implement a logical organization of the log. This approach involves logically structuring the log so that all related information is stored together. The log is physically stored on a disk, with information about a specific object possibly fragmented across the log. Our goal is to create a logical structure that allows related operations to be collected and stored sequentially. This optimization will facilitate faster access to relevant information when needed.

Figure 1a,b illustrate the concept. Figure 1a displays the original physical structure of a log, indicating the locations of operations on three objects: x, y and z. It is worth noting that operations on a particular data object are not conducted in a single transaction but rather by multiple transactions at different points in time.

Our plan involves designing a logical structure like the one depicted in Figure 1b, which organizes all related operations sequentially. Importantly, our objective is not to duplicate the log structure, which would require additional space and time. Instead, we aim to devise the necessary data structures and algorithms to establish the logical structure, thereby facilitating a faster recovery process.

In an operating system, finding information and starting to write wherever space is available does not guarantee an increasing or sequential order of data storage. When a page becomes full, the system searches for an available page to write data, but these available pages may not be in increasing order. For instance, if a file is deleted from page number 6, that page becomes available for use. When new data need to be written, the system looks

for available pages wherever they may be and jumps there. Availability of pages is not in a particular sequence because users delete files in various orders. For example, a user might delete a file created a month ago, then one created today, and later one created 10 days ago. However, it is important to note that the page numbers mentioned here are logical identifiers, not physical page addresses. Therefore, mapping is necessary to determine the exact physical page addresses corresponding to logical page numbers. This mapping ensures that the system can locate the appropriate physical page for data storage based on its logical identifier. Fortunately, creating this mapping is relatively straightforward.

X	Y	Z	Y	Z	Y	Z	X

(**a**)

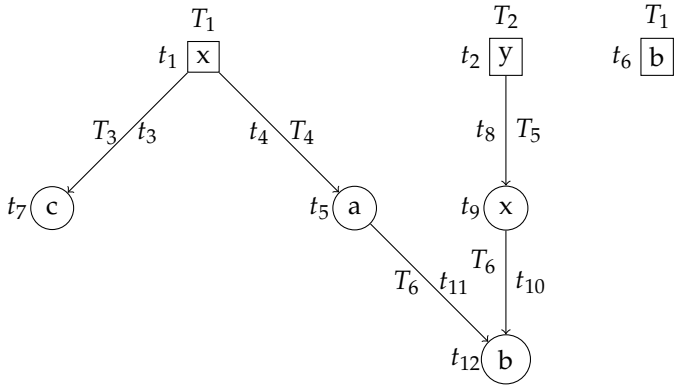
X	X	Y	Y	Y	Z	Z	Z

(**b**)

Figure 1. (**a**) Physical log structure showing locations of related operations on various objects (**b**) Logical log structure showing related operations on various objects.

So let us consider the following log structure based on the operations that occurred in the graph.

The graph in Figure 2a shows that transaction T_1 wrote x at time t_1. Then, T_4 read x at time t_4, wrote a at time t_5, and so on. In the log structure below, the write operation on data item x in T_1 is represented as $W_1[x]$. Similarly, the read operation on x and the write operation on a in T_4 are represented as $R_4[x]\ W_4[a]$, and so on.



(**a**)

Transactions' operations	$W_1[x]\ W_2[y]\ R_3[x]$	$R_4[x]\ W_4[a]\ W_1[b]\ W_3[c]$	$R_5[y]\ W_5[x]\ R_6[x]\ R_6[a]\ W_6[b]$
page	P1	P2	P3

(**b**)

Figure 2. (**a**) Transactions in a graph structure. (**b**) Transactions in traditional log structure.

The log structure operates such that each transaction is sequentially recorded as they are executed, irrespective of their sequence. For instance, in T_1, data object x is written blindly, meaning it updates x without reading any other data item. Similarly, transaction T_2 executes a blind write operation on data object y. These and subsequent operations are all recorded within the first page of the log structure. Conversely, after page 1 is filled up, on page 2, transaction T_4 involves reading data object x to update a, and then, again transaction T_1 occurs to blind write b, demonstrating a sequential flow of operations in the log which are logged in sequential ordered logical pages. These operations can be better

understood by using the graph structure in Figure 2a. These sequential logical pages are mapped to a physical page address.

Clustered data in the context of databases refers to the way data are physically organized and stored on disk to enhance performance for certain types of queries and operations. This paper employs a similar approach, where our primary focus is to create a cluster for each written data item. As transactions progress, this cluster details every update operation on the item, including the data items read to make each update on the clustered data item and when they were read. Additionally, it records the other data items that were updated based on those versions, along with the timestamp when those versions were read. For example, as shown in Figure 2a, the cluster of data item x will include all the reading operations that have been used to update x. For instance, data item x was written at time t_9 and appears on page 3 of the log. Since a transaction wrote x after reading y, x is a function of y, that is, $x = f(y)$, which means the value of y was used to update x. However, this version of x was used to update data item b after reading x at time t_{10}. All this information is included in this data structure.

This clustering technique is pivotal for tracking the updates and dependencies of data items, especially in scenarios involving potential malicious modifications. By organizing data in clusters, we can efficiently monitor and manage the integrity and recovery of data items within the database.

We will also include the page number where the cluster data item was written. The page number will be useful in understanding the formula used for the update. This is important because, while we record that a data item was written after reading data item x, the specific formula used for the update will be found in the log. Thus, having the page number will allow us to directly locate and verify the formula, enabling us to recalculate with the newly recovered values. It must be noted that there would be multiple cluster structures for each written data item.

Before it is explained with an example, let us check the cluster structure.

Read data: data items which are used to update cluster data item (could be multiple)	**Cluster data:** data items that have been written	**Written data:** data items reading the cluster data item (could be multiple)
Time when they were read	Time when it was written	Time when the cluster data was read
Ps. If cluster data item is blindly written then this part will be empty	Page number where it is written	

Keeping this in mind, in the previous example, if we consider x as the cluster data item then for x, the cluster structure will be as follows:

Read data	**Cluster data**	**Written data**
	x time: t_1 page: *1*	c time: t_3 a time: t_4
y time: t_8	x time: t_9 page: *3*	b time: t_{10}

This list is created in real time as transactions occur. When a transaction reads a data item to write another data item, it is added to the table in the above format. Here, in the above example, it is seen that x was written at time t_1 and appears on page 1 since it was a blind written data item, so nothing was read to modify x. But that version of x was used to update c after reading x at time t_3, and for data item a, the reading time of x is t_4.

The cluster data structure can be persisted to non-volatile storage at regular intervals, e.g., hourly, to facilitate efficient recovery in case of system failure. In the event of data loss, the cluster state can be reconstructed using two approaches:

1. Restoration from the most recent persistent snapshot;
2. Complete regeneration from the initial state.

The periodic persistence of data clusters to non-volatile storage represents a critical strategy for ensuring data resilience and facilitating efficient recovery in distributed database systems. This approach implements a hybrid recovery mechanism that combines snapshot-based restoration with incremental log replay. At predefined intervals, typically hourly, the system captures and persists the current state of data clusters to disk storage, creating recovery points. In the event of a system failure, the recovery process initiates by restoring the most recent persisted cluster state, which serves as a baseline. Subsequently, the system applies an incremental replay of log entries generated since the last persistence point, effectively reconstructing the delta of changes.

For instance, if cluster snapshots are saved hourly and a failure occurs at 3:30, the system can restore the cluster state from the 3:00 snapshot. Subsequently, it can replay and recluster and regenerate the log entries generated between 3:00 and 3:30 to reconstruct the lost 30 min of cluster data. This approach balances recovery speed with data recency, allowing for rapid restoration of the bulk of the cluster state while ensuring minimal data loss through incremental reconstruction of the most recent time period.

This method optimizes the trade-off between recovery time and data loss by leveraging both bulk state restoration and fine-grained log replay. The frequency of cluster persistence can be tuned based on system requirements, balancing storage overhead against potential data loss. Furthermore, this approach enables parallel processing during recovery, as the bulk of the data can be quickly restored from the persisted state while the incremental changes are reconstructed concurrently. The technique also facilitates point-in-time recovery capabilities, allowing administrators to restore the system to any state between persistence points by selectively applying log entries. Overall, this hybrid persistence and recovery strategy enhances system resilience, minimizes downtime, and provides flexible recovery options in large-scale distributed database environments.

For creating those clustered data structures, Algorithm 1 will now be explored.

Notations used in Algorithm 1

- i-th transaction (T_i): A specific transaction that occurred at the i-th position in the sequence of transactions.
- T_j: A specific transaction that occurred at the j-th position where $j \neq i$.
- Cluster (C_x): A data structure (likely a hash table or key-value store) that stores data items and their associated metadata. So, for each data item x, $C_x = \{(x, wt_x) | x$ is the clustered data and $wt[x]$ is the timestamp of the write operation on $x\}$.
- $R_i[x]$: Reading operation on data item x in transaction T_i.
- $W_i[y]$: Writing operation on data item y in transaction T_i.
- ReadingList (RL_{T_i}): A list (or similar data structure) associated with each transaction to store read data items and their timestamps. That is, the i-th transaction T_i, $RL_{T_i} = \{(x, rt[x]) | x$ is the data item that is read to update another data item y and $rt[x]$ is the timestamp of $R_i[x]\}$.
- WritingList (WL_{T_i}): A list (or similar data structure) associated with each transaction to store written data items and their timestamps. That is, for the i-th transaction T_i, $WL_{T_i} = \{(y, wt[y], p) | y$ is the data item that is modified, $wt[y]$ is the timestamp of $W_i[y]$, and p is the page number of the log where the modification happened$\}$.
- Cluster structure:
  - Read Data Column (RD)
  - Cluster Data Column (CD)
  - Written Data Column (WD)

Algorithm 1 Creating Clustered Data Structures

1: Initialize *Cluster* as an empty dictionary
2: Initialize *ReadingList* (RL_{T_i}) for each transaction
3: Initialize *WritingList* (WL_{T_i}) for each transaction
4: **for** each page in log **do**
5: **for** each operation **do**
6: **if** the operation is $R_i[x]$ **then**
7: **if** RL_{T_i} does not exist **then**
8: Create RL_{T_i}
9: Append $[x, rt[x]]$ to the RL_{T_i}
10: **else**
11: Append $[x, rt[x]]$ to the RL_{T_i}
12: **end if**
13: **end if**
14: **if** operation is $W_i[y]$ **then**
15: **if** WL_{T_i} exists **then**
16: Go to step 20
17: **else**
18: Create WL_{T_i}
19: **end if**
20: Append $[y, wt[y]]$ to WL_{T_i}
21: **if** cluster C_y exists **then**
22: Go to step 26
23: **else**
24: Create C_y in *Cluster*
25: **end if**
26: Append RL_{T_i} to RD
27: Append WL_{T_i} to CD
28: **for** each data item in RL_{T_i} **do**
29: **if** cluster C_x exists **then**
30: **if** $t_{CD} < rt[x] < wt[y]$ **then**
31: Append $[y, rt[x]]$ to WD
32: **end if**
33: **end if**
34: **end for**
35: $RL_{T_i} = \emptyset$
36: $WL_{T_i} = \emptyset$
37: **end if**
38: **if** operation is a commit operation **then**
39: Delete RL_{T_i} and WL_{T_i} for committed T_i
40: Finalize and store clusters for the transaction
41: **end if**
42: **end for**
43: **end for**
44: Return Cluster

Comment: From the example in Figure 2a,b, it can be observed that the clustering process begins by processing Page 1 (P1) transactions sequentially. For the initial writing operation in Transaction 1, a $\boldsymbol{WL}_{T_1}$ is created and populated with $[x, t_1]$ according to step 14. Then, the Algorithm 1 , according to step 21, checks for the existence of C_x. Since C_x does not exist, it is created (step 24), and $\boldsymbol{RL}_{T_1}$ (step 26) is appended which is an empty list since x was blindly written here; $\boldsymbol{WL}_{T_1}$ is checked, resulting in $[x, t_1]$ being added to the Cluster Data (CD) column. Subsequently, for $W_2[y]$, another Writing List $\boldsymbol{WL}_{T_2}$ is established, and $[y, t_2]$ is appended. The next transaction prompts the creation of Reading List $\boldsymbol{RL}_{T_3}$ according to step 7, which includes $[x, t_3]$. Moving to Page 2 (P2), Transaction 4 is processed by creating both $\boldsymbol{RL}_{T_4}$ and $\boldsymbol{WL}_{T_4}$, incorporating $[x, t_4]$ and $[a, t_5]$, respectively.

This systematic process continues for each subsequent transaction, gradually forming the desired clusters based on the read and write operations performed on various data items across different transactions.

In step 30, the timestamp of the existing Clustered Data (CD) is compared with two other time values: the read time of data item x, denoted as $rt[x]$, and the update time of another data item y that occurred after reading x, denoted as $wt[y]$. The most recent CD timestamp that is still earlier than both $rt[x]$ and $wt[y]$ is selected. And then, the pair $[y, rt[x]]$ is added in the corresponding Write Dependency (WD) column of the clustered data.

This process ensures that the correct sequence of events is captured, as a data item can only be read and subsequently used to update another item after the clustered item has been updated. Consequently, the most recent clustered data item will have a timestamp that precedes both its read time and the update time of any data item that depends on it.

4. Damage Assessment

Definition 1. ***(Damaged data item list):*** *Whenever we encounter a new data item that has been written using a damaged data item, we will add it to our damaged data item list. For example, in the cluster structure of Figure 2a, if we find that* $\boldsymbol{x}$ *in transaction* T_1 *is malicious,* $\boldsymbol{a}$ *and* $\boldsymbol{c}$ *are consequently affected. And in sequence, when we jump to both* $\boldsymbol{a}$ *and* $\boldsymbol{c}$ *cluster, we see* $\boldsymbol{b}$ *is dependent on* $\boldsymbol{a}$*. So, in turn, b is also affected. Thus, the damaged data item list would be* $\{\boldsymbol{x, c, a, b}\}$.

It is important to note that we are not including the later updated $\boldsymbol{x}$ in T_5. As from the cluster structure, we see that the later $\boldsymbol{x}$ was recovered by being updated using $\boldsymbol{y}$ which was not damaged. Let us check all the cluster structures for all the cluster data items. And for visual representation, we color damaged data items grey.

x Cluster structure:

Read data	Cluster data	Written data	
	x time: t_1 page: *1*	c time: t_3	a time: t_4
y time: t_8	x time: t_9 page: *3*	b time: t_{10}	

y Cluster structure:

Read data	Cluster data	Written data
	y time: t_2 page: *1*	c time: t_3

a Cluster structure:

Read data	Cluster data	Written data
x time: t_4	a time: t_5 page: *2*	b time: t_{11}

b Cluster structure:

Read data		Cluster data	Written data
		b time: t_6 page: *2*	
x time: t_{10}	a time: t_{11}	b time: t_{12} page: *3*	

c Cluster structure:

Read data	Cluster data	Written data
x time: t_3	c time: t_7 page: *2*	b time: t_{11}

Graph representation is used here to understand this better. A graph has been shown to understand and build the table for cluster structure. But the graph representation is not

used for various reasons. Firstly, because the graph representation is not available, only the log is accessible. The question arises as to why the graph should not be constructed from the log instead of building this cluster structure. This is because although graphs are easier to understand, implementing graphs is harder and finding the exact location of a damaged data item is more time-consuming than our method. And from the new structure, finding the summary of information is much easier. For instance, it allows us to quickly determine the number of different versions of data item $\boldsymbol{x}$ that were affected, along with the specific time intervals during which the damage occurred. Additionally, we can readily identify when $\boldsymbol{x}$ was recovered, when it was initially compromised, and the duration of its damaged state. All this critical information is easily accessible within our proposed structure, enhancing the overall efficiency of data integrity management and recovery processes.

Now, we delve into the next Algorithm 2 which is for the damage assessment.

Notations used in Algorithm 2

- *DDL (DamagedDataList)*: List to store damaged data items and their timestamps. So,

$$DDL = \{[\boldsymbol{u}, \boldsymbol{wt}[\boldsymbol{u}]] \mid u \text{ is the damaged data item and } wt[u] \text{ is the write time stamp}\}$$

- *PDL (PropagatedDamagedList)*: List to store propagated damaged data items and related information. So,

$$PDL = \{[\boldsymbol{v}, \boldsymbol{rt}[\boldsymbol{u}]] \mid v \text{ is the propagated damaged data item and } rt[u] \text{ is the time stamp when data item} u \text{ was read to update data item } v\}$$

- *Cluster (C_x)*: A data structure (likely a hash table or key-value store) that stores data items and their associated metadata. So, for each data item x,

$$C_x = \{(\boldsymbol{x}, \boldsymbol{wt}[\boldsymbol{x}]) \mid x \text{ is the clustered data and } wt[x] \text{ is the time stamp of the write operation of } x\}$$

- T_m: malicious transaction identified by IDS.
- Cluster structure:
  - Read Data Column (RD)
  - Cluster Data Column (CD)
  - Written Data Column (WD)
  - p = page number

Algorithm 2 Damage Assessment

```
Find the W_m[x] in the transaction T_m
Append the [x, wt[x]] in the DDL
if cluster C_x exists then
    Find [x, wt[x]] in CD
    if WD exists then
        Append [v, rt[x]] in the PDL
        if cluster C_v exists then
            Check RD for [x, rt[x]]
            if matched then
                From column CD find [v, wt[v]]
                Append [v, wt[v]] in the DDL
                if WD exists for [v, wt[v]] then
                    Go to step 6
                end if
            end if
        end if
    end if
end if
Return DDL
```

Comment: In Figure 2a,b, if the malicious transaction is T_1, identified by the intrusion detection system (IDS), the process begins by reviewing the log to locate data items x and b that were written in T_1 (step 1). According to step 2, $[x, t_1]$ and $[b, t_6]$ are added to the DDL. These items are then traversed one by one, starting with $[x, t_1]$. The cluster C_x is checked, and $[x, t_1]$ is searched within the Cluster Data (CD). The Written Data (WD) for $[x, t_1]$ are also reviewed, which reveal that data items c and a were written after reading x. As a result, $[c, t_3]$ and $[a, t_4]$ are added to the Propagated Damage List (PDL) according to step 6. Next, cluster C_c is checked in Step 8, and the Reading Data (RD) of cluster C_c is searched for $[x, t_3]$. Upon finding this in RD, the CD is checked and cluster data $[c, t_7]$ is found, which is then added to the DDL. For data item c, step 6 through step 13 is not repeated because the WD of $[c, t_7]$ is empty. This process tracks how the malicious transaction's effects propagate through the cluster dependencies, identifying the impacted data items along the way.

5. Simulations Results

In our simulation study, we considered three variables, which are as follows:

1. Number of Transactions: This represents the quantity of transactions executed per experiment.
2. Number of Data Items: This denotes the total count of data items utilized per experiment.
3. Maximum Number of Operations per Transaction: This parameter defines the highest number of individual operations (such as read or write) that can be performed within a single transaction.

For consistency, we maintained the following base values throughout the experiments:

- Number of Transactions = 2000
- Number of Data Items = 500
- Maximum Number of Operations per Transaction = 5

In each scenario, one variable was manipulated while the others were kept constant. The program was run 25 times for each scenario, and the average number of operations was determined from the log. This calculation was performed for both the clustered log method and the traditional log method after the identification of the malicious data item.

In a typical database scenario, operations are stored in a log file. However, the sequential nature of the log file introduces a bottleneck, which significantly slows down operations. This slowdown occurs because files and logs are stored on secondary disks, and as the log is sequential, disk access is required for reading operations, resulting in a slower process. The objective is to minimize the number of such readings, as reducing these operations leads to quicker recovery times. To demonstrate that the clustered log accelerates damage assessment compared to the normal sequential log, a comparison is made between the average number of data items checked and the average number of pages examined in both the clustered log and the traditional log.

In our analysis, a logarithmic scale has been applied to the Y-axis of the chart. This chart plots the average number of data items to be read and the average number of pages to check in both clustered and traditional logs. The analysis considers varying numbers of transactions, varying numbers of data items, and varying maximum numbers of operations per transaction. The use of a logarithmic scale, where each increment represents a tenfold increase, allows for a clearer visualization of the proportional differences and growth patterns between the two logging methods since the differences might not be as evident on a linear scale.

5.1. Varying the Number of Transactions

In this scenario, the number of transactions is varied, ranging from 500 to 4000, while keeping the other variables (number of data items and maximum number of operations per transaction) constant for both cases of average number of data items to be read and average number of pages to be accessed.

5.1.1. Average Number of Data Items to Check

The comparison highlights the efficiency of the clustered log approach in managing dependencies between data items (Figure 3). While the traditional log method shows a gradual increase in the amount of data items read as the number of transactions increases, the clustered log method needs to read a much lower number of data items. This trend is attributed to the rising number of transactions, which results in a greater number of dependencies among data items. Consequently, this leads to increasing number of data items in each cluster. This efficiency is due to the ability of the clustered log to quickly identify dependencies, avoiding the need to sift through the entire log, which is a limitation of the traditional approach.

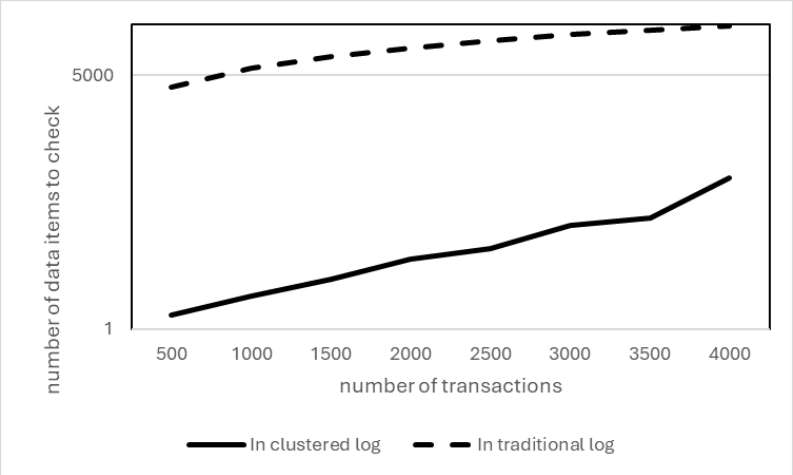


Figure 3. Numberof data items to check when the number of transactions increase.

5.1.2. Average Number of Pages to Check

Similar to the pattern observed with the average number of data items, the average number of pages examined follows a similar trend in Figure 4. In the clustered log method, data items are grouped together, and each cluster includes the page number where the items are recorded. This is the reason why our method allows for direct access to specific pages identified in the clustered log, eliminating the need to scan each page to find dependent data items. This targeted approach greatly accelerates the damage assessment process, as the traditional method of navigating through multiple pages is time-consuming.

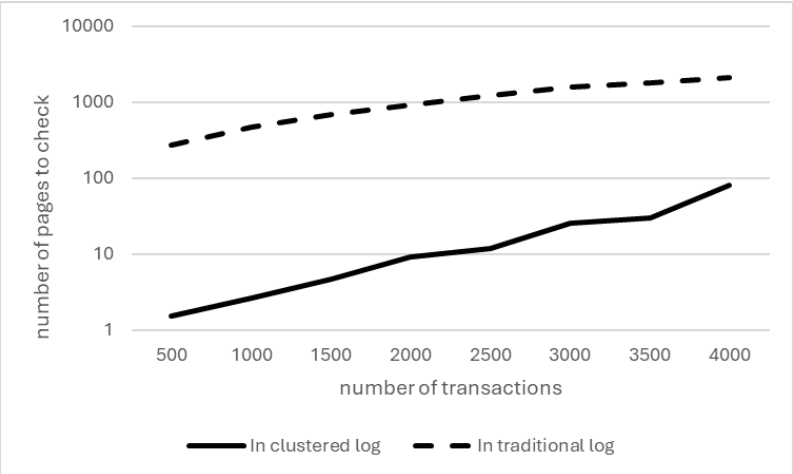


Figure 4. Number of pages to access as number of transactions increase.

5.2. Varying the Number of Data Items

In this scenario, the number of data items is adjusted, ranging from 200 to 1000, while the other variables (Number of transactions and Maximum number of operations per transaction) are kept constant.

5.2.1. Average Number of Data Items to Check

As the number of data items increases, the data-item-to-transaction ratio becomes higher. Consequently, interdependencies among data items decrease, which is reflected in the downward trend observed in the graph in Figure 5 for our clustered method. In contrast, in the traditional method, the trend remains constant, but the time taken is significantly longer due to the greater number of data items that need to be checked compared to our method.

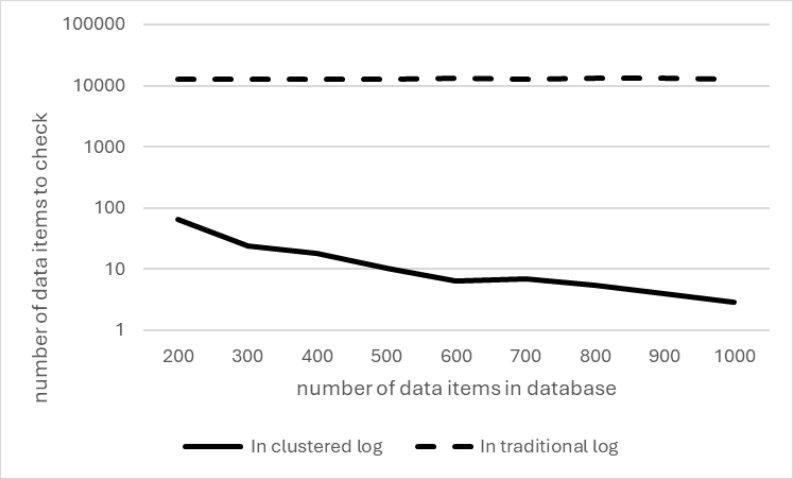


Figure 5. Number of data items to check as the number of data items in the database increases.

5.2.2. Average Number of Pages to Check

Similar to the previous case, a downward trend is observed in Figure 6 as well. As the number of data items increases, interdependencies among them decrease, leading to a reduction in the number of affected data items. Consequently, this results in fewer pages needing to be traversed.

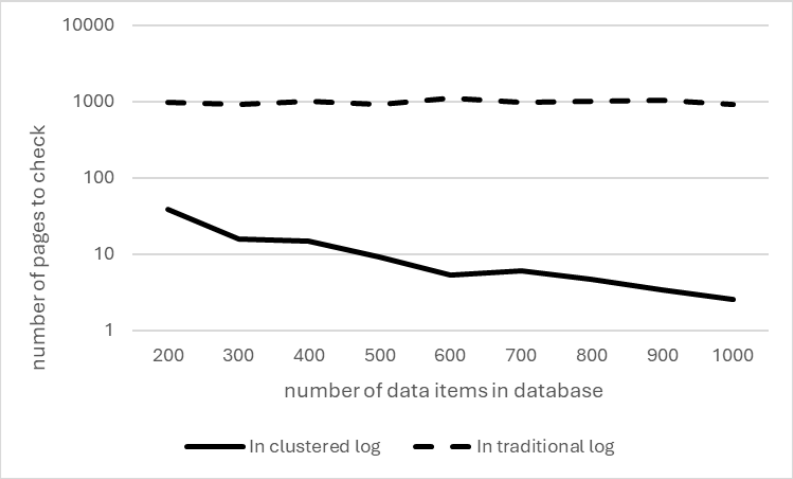


Figure 6. Number of pages to access as number of data items in database increases.

5.3. Varying the Maximum Number of Operations per Transaction

In this scenario, the maximum number of operations per transaction are adjusted, ranging from 2 to 12, while keeping the other variables (number of transactions and number of data items) constant.

5.3.1. Average Number of Data Items to Check

Increased dependencies result in larger clusters, which means more data items are included within each cluster. Consequently, a greater number of data items are affected by the damaged items, leading to the observed upward trend in the graph in Figure 7. In traditional logs, the increase is much less pronounced compared to our log structure. This is because, in traditional logs, one must scan through the entire log, resulting in less variation in the number of data items processed. In contrast, our method shows a more noticeable upward trend due to its efficient handling of dependencies, which leads to a more pronounced increase in the number of affected data items.

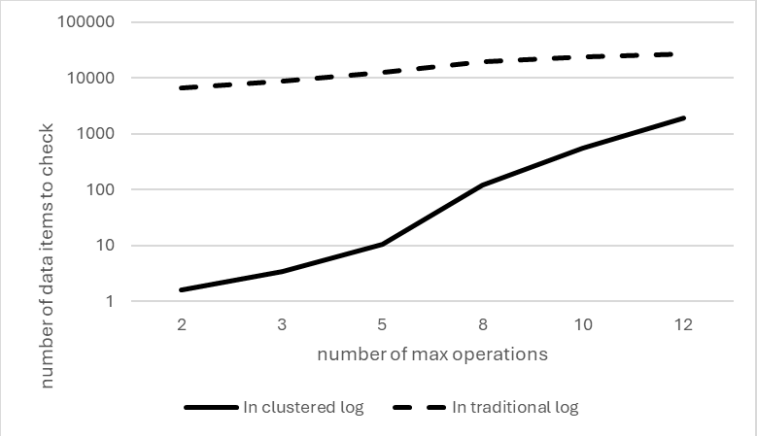


Figure 7. Number of data items to check as number of max operations per transaction increases.

5.3.2. Average Number of Pages to Check

The average number of pages to check shows a similar trend (Figure 8) to the previous case. As the maximum number of operations increases, more dependencies are created, necessitating the traversal of additional pages. This is because a higher number of data items and dependencies often means that these items are spread across different pages, requiring more pages to be checked. In traditional logs, the number of pages checked remains relatively stable due to the constant number of data items, but it is still significantly higher compared to the clustered log structure.

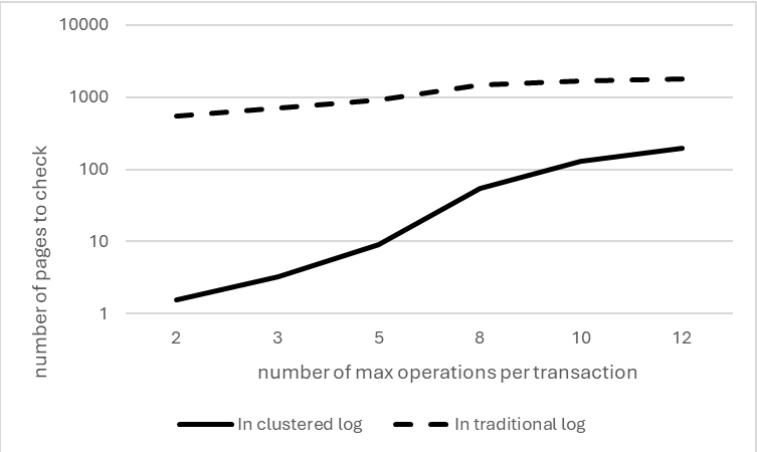


Figure 8. Number of pages to access as number of max operations per transaction increases.

6. Discussion

The ClusteredLog approach introduced in this paper represents a significant advancement in log management for database systems, particularly in scenarios requiring fast efficient data recovery and integrity management. Our research demonstrates several key findings and implications.

6.1. Efficiency in Data Recovery

ClusteredLog's logical organization of log structures significantly reduces the time required for damage assessment and recovery. By clustering related operations for each data item, our method enables quicker identification and recovery of damaged data items compared to traditional sequential logging methods.

6.2. Scalability

The simulation results show that ClusteredLog's performance advantage becomes more pronounced as the number of transactions and data items increases. This scalability is crucial for modern database systems that handle large volumes of data and transactions.

6.3. Dependency Management

ClusteredLog's ability to track dependencies between data items proves particularly effective in scenarios involving complex transaction structures. This feature allows for more accurate and efficient propagation of damage assessment, which is critical in maintaining data integrity.

6.4. Trade-Offs and Considerations

While ClusteredLog offers significant improvements in recovery time and efficiency, it is important to note potential trade-offs:

- **Implementation Complexity**: The logical clustering approach may require more complex implementation compared to traditional sequential logging.
- **Storage Overhead**: Although not explicitly measured in our study, the clustered structure might introduce some storage overhead, which should be evaluated in future work.

7. Conclusions

The ClusteredLog approach introduced in this paper represents a significant advancement in log management for database systems, particularly in scenarios requiring efficient data recovery and integrity management. Our research demonstrates that by implementing a logical organization that clusters related operations for each data item, we can significantly reduce the time required for damage assessment and recovery. This efficiency is crucial in modern, mission-critical database environments where system downtime or data integrity issues can have severe consequences. The simulation results show that ClusteredLog's performance advantage becomes more pronounced as the number of transactions and data items increases, highlighting its scalability for modern database systems handling large volumes of data and transactions. The method's ability to track dependencies between data items proves particularly effective in scenarios involving complex transaction structures, allowing for more accurate and efficient propagation of damage assessment. While ClusteredLog offers significant improvements in recovery time and efficiency, it is important to note potential trade-offs, such as implementation complexity and possible storage overhead, which should be evaluated in future work. Overall, ClusteredLog represents a promising solution to the challenges of efficient log management in modern database systems, paving the way for more robust and responsive data recovery mechanisms. In this study, we employed simulation-based evaluation methods, necessitated by constraints in computational resources and time. However, we acknowledge the significant advantages that could be gained from leveraging standardized benchmark datasets to enhance the robustness and

generalizability of our findings. Although integrating such datasets lies beyond the scope of this work, it represents a promising direction for future research initiatives.

Author Contributions: Conceptualization/Validation/methodology/writing—review and editing, M.S.A. and B.N.P.; software/visualization/writing—original draft preparation, M.S.A.; supervision, B.N.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: No datasets were generated during the current study.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Ammann, P.; Jajodia, S.; McCollum, C.D.; Blaustein, B.T. Surviving information warfare attacks on databases. In Proceedings of the IEEE Symposium on Security and Privacy (Cat. No. 97CB36097), Oakland, CA, USA, 4–7 May 1997; IEEE: Piscataway, NJ, USA, 1997; pp. 164–174.
2. Liu, P.; Ammann, P.; Jajodia, S. Rewriting Histories: Recovering from Malicious Transactions. In *Security of Data and Transaction Processing*; Atluri, V., Samarati, P., Eds.; Springer: Boston, MA, USA, 2000. [CrossRef]
3. Patnaik, S.; Panda, B. Transaction-relationship oriented log division for data recovery from information attacks. *J. Database Manag. (JDM)* **2003**, *14*, 27–41. [CrossRef]
4. He, S.; Zhu, J.; He, P.; Lyu, M.R. Experience Report: System Log Analysis for Anomaly Detection. In Proceedings of the 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), Ottawa, ON, Canada, 23–27 October 2016; pp. 207–218. [CrossRef]
5. Idrisi, M.S.; Khan, M.S.; Tripathi, M.; Singh, I. MSPMO: Maximal Sequential Pattern and Metaheuristic Optimization based Database Intrusion Detection System. In Proceedings of the 2022 2nd International Conference on Intelligent Technologies (CONIT), Hubli, India, 24–26 June 2022; pp. 1–5. [CrossRef]
6. Sendner, C.; Iffländer, L.; Schindler, S.; Jobst, M.; Dmitrienko, A.; Kounev, S. Ransomware Detection in Databases through Dynamic Analysis of Query Sequences. In Proceedings of the 2022 IEEE Conference on Communications and Network Security (CNS), Austin, TX, USA, 3–5 October 2022; pp. 326–334. [CrossRef]
7. Doroudian, M.; Shahriari, H.R. A hybrid approach for database intrusion detection at transaction and inter-transaction levels. In Proceedings of the 2014 6th Conference on Information and Knowledge Technology (IKT), Shahrood, Iran, 27–29 May 2014; pp. 1–6. [CrossRef]
8. Singh, I.; Kumar, N.; Srinivasa, K.G.; Sharma, T.; Kumar, V.; Singhal, S. Database intrusion detection using role and user behavior based risk assessment. *J. Inf. Secur. Appl.* **2020**, *55*, 102654. ISSN 2214-2126. [CrossRef]
9. Zhang, D.; Egersdoerfer, C.; Mahmud, T.; Zheng, M.; Dai, D. Drill: Log-based Anomaly Detection for Large-scale Storage Systems Using Source Code Analysis. In Proceedings of the 2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS), St. Petersburg, FL, USA, 15–19 May 2023; pp. 189–199. [CrossRef]
10. Singh, I.; Jindal, R. Expectation maximization clustering and sequential pattern mining based approach for detecting intrusive transactions in databases. *Multimed. Tools Appl.* **2021**, *80*, 27649–27681. [CrossRef]
11. Yan, S.; Shi, L.; Ren, J.; Wang, W.; Liu, Y.; Sun, L. Log-Based Anomaly Detection with Transformers Pre-Trained on Large-Scale Unlabeled Data. In Proceedings of the ICC 2024—IEEE International Conference on Communications, Denver, CO, USA, 9–13 June 2024; pp. 1–6. [CrossRef]
12. Han, X.; Yuan, S.; Trabelsi, M. LogGPT: Log Anomaly Detection via GPT. In Proceedings of the 2023 IEEE International Conference on Big Data (BigData), Sorrento, Italy, 15–18 December 2023; pp. 1117–1122. [CrossRef]
13. Wittkopp, T.; Acker, A.; Kao, O. Progressing from Anomaly Detection to Automated Log Labeling and Pioneering Root Cause Analysis. In Proceedings of the 2023 IEEE International Conference on Data Mining Workshops (ICDMW), Shanghai, China, 1–4 December 2023; pp. 1231–1239. [CrossRef]
14. Debnath, B.; Solaimani, M.; Gulzar, M.A.G.; Arora, N.; Lumezanu, C.; Xu, J. LogLens: A Real-Time Log Analysis System. In Proceedings of the 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), Vienna, Austria, 2–6 July 2018; pp. 1052–1062. [CrossRef]
15. Du, M.; Li, F.; Zheng, G.; Srikumar, V. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17). Association for Computing Machinery, New York, NY, USA, 21–25 August 2017; pp. 1285–1298. [CrossRef]
16. Meng, W.; Liu, Y.; Zhu, Y.; Zhang, S.; Pei, D.; Liu, Y.; Chen, Y.; Zhang, R.; Tao, S.; Sun, P.; et al. Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. *IJCAI* **2019**, *19*, 4739–4745.
17. Rodrigues, K.; Luo, Y.; Yuan, D. CLP: Efficient and scalable search on compressed text logs. In Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21), Virtual, 14–16 July 2021; pp. 183–198.

18. Wang, C.; Xu, D.; Li, Z. Log2graphs: An Unsupervised Framework for Log Anomaly Detection with Efficient Feature Extraction. *arXiv* **2024**, arXiv:2409.11890.
19. Bongkodmalee, T.; Vongvipaporn, S.; Siripanich, P.; Fugkeaw, S.; Vorakulpipat, C. A Design and Development of Web Application Vulnerability Detection Using Graph-Based Modeling Analysis Over CAPEC. In Proceedings of the 2024 21st International Joint Conference on Computer Science and Software Engineering (JCSSE), Phuket, Thailand, 19–24 June 2024; pp. 28–34. [CrossRef]
20. Julio, C.C.; Khwaja, A.S.; Woungang, I.; Anpalagan, A. A Graph-based Data Mining Approach for Template Recognition using Large Log Datasets in Software Systems. In Proceedings of the 2024 21st International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), Khon Kaen, Thailand, 27–30 May 2024; pp. 1–5. [CrossRef]
21. Locke, S.; Li, H.; Chen, T.-H.P.; Shang, W.; Liu, W. LogAssist: Assisting Log Analysis Through Log Summarization. *IEEE Trans. Softw. Eng.* **2022**, *48*, 3227–3241. [CrossRef]
22. Liu, J.; Zhu, J.; He, S.; He, P.; Zheng, Z.; Lyu, M.R. Logzip: Extracting Hidden Structures via Iterative Clustering for Log Compression. In Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), San Diego, CA, USA, 11–15 November 2019; pp. 863–873. [CrossRef]
23. Yunanto, W.; Pao, H.-K. User Behaviour Risk Evaluation in Zero Trust Architecture Environment. In Proceedings of the 2022 IEEE 8th World Forum on Internet of Things (WF-IoT), Yokohama, Japan, 26 October–11 November 2022; pp. 1–6. [CrossRef]
24. Li, Z.; Chen, T.-H.; Yang, J.; Shang, W. Studying Duplicate Logging Statements and Their Relationships With Code Clones. *IEEE Trans. Softw. Eng.* **2022**, *48*, 2476–2494. [CrossRef]
25. Chen, Z.; Liu, J.; Su, Y.; Zhang, H.; Wen, X.; Ling, X.; Yang, Y.; Lyu, M.R. Graph-based Incident Aggregation for Large-Scale Online Service Systems. In Proceedings of the 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), Melbourne, Australia, 15–19 November 2021; pp. 430–442. [CrossRef]
26. Liu, J.; Jiang, Z.; Gu, J.; Huang, J.; Chen, Z.; Feng, C. Prism: Revealing Hidden Functional Clusters from Massive Instances in Cloud Systems. In Proceedings of the 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), Luxembourg, 11–15 September 2023; pp. 268–280. [CrossRef]
27. Raffety, J.; Stone, B.; Svacina, J.; Woodahl, C.; Cerny, T.; Tisnovsky, P. Multi-source log clustering in distributed systems. In *Information Science and Applications: Proceedings of ICISA 2020*; Springer: Singapore, 2021; pp. 31–41.
28. Hamooni, H.; Debnath, B.; Xu, J.; Zhang, H.; Jiang, G.; Muee, A. LogMine: Fast Pattern Recognition for Log Analytics. In Proceedings of the 25th ACM International on Conference on Information and Knowledge Management (CIKM '16), Indianapolis, IN, USA, 24–28 October 2016; pp. 1573–1582. [CrossRef]
29. Landauer, M.; Skopik, F.; Wurzenberger, M.; Rauber, A. System log clustering approaches for cyber security applications: A survey. *Comput. Secur.* **2020**, *92*, 101739. ISSN 0167-4048. [CrossRef]
30. Pokharel; Prabhat; Pokhrel, R.; Joshi, B. A hybrid approach for log signature generation. *Appl. Comput. Inform.* **2023**, *19*, 108–121. [CrossRef]
31. Lu, J.; Li, F.; Liu, C.; Li, L.; Feng, X.; Xue, J. CloudRaid: Detecting Distributed Concurrency Bugs via Log Mining and Enhancement. *IEEE Trans. Softw. Eng.* **2022**, *48*, 662–677. [CrossRef]
32. Chen, Z.; Jiang, Z.; Su, Y.; Lyu, M.R.; Zheng, Z. TraceMesh: Scalable and Streaming Sampling for Distributed Traces. *arXiv* **2024**, arXiv:2406.06975.
33. Ahmad, M.S.; Panda, B. Damage Assessment in Fog Computing Systems: A Blind Write Lineage Approach. In Proceedings of the 5th Workshop on Secure IoT, Edge and Cloud systems (SIoTEC) 2024, Philadelphia, PA, USA, 6–9 May 2024.
34. Ahmad, M.S.; Panda, B. Validating Damage Assessment: A Simulation-Based Analysis of Blind Write Lineage in Fog Computing. In Proceedings of the Eighteenth International Conference on Emerging Security Information, Systems and Technologies SECURWARE 2024, Nice, France, 3–7 November 2024.
35. Ragavan, H.; Panda, B. Mitigating Malicious Updates: Prevention of Insider Threat to Databases. In Proceedings of the 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, Melbourne, VIC, Australia, 16–18 July 2013; pp. 781–788. [CrossRef]
36. Kurra, K.; Panda, B.; Li, W.-N.; Hu, Y. An Agent Based Approach to Perform Damage Assessment and Recovery Efficiently after a Cyber Attack to Ensure E-government Database Security. In Proceedings of the 2015 48th Hawaii International Conference on System Sciences, Kauai, HI, USA, 5–8 January 2015; pp. 2272–2279. [CrossRef]
37. Panda, B.; Burns, J. Assessing Damage and Recovery of Critical Data in Unsecure Cloud Systems. In Proceedings of the 2023 Congress in Computer Science, Computer Engineering, & Applied Computing (CSCE), Las Vegas, NV, USA, 24–27 July 2023; pp. 1191–1198.
38. Panda, B.; Shruthi, R. Optimized Damage Assessment in Large Datasets in Cloud. In Proceedings of the 2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC), Vancouver, WA, USA, 6–9 December 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 384–390.
39. Haraty, R.A.; Kaddoura, S.; Zekri, A. Transaction dependency based approach for database damage assessment using a matrix. *Int. J. Semant. Web Inf. Syst.* **2017**, *13*, 74–86. [CrossRef]

40. Haraty, R.A.; Boukhari, B.; Kaddoura, S. An effective hash-based assessment and recovery algorithm for healthcare systems. *Arab. J. Sci. Eng.* **2021**, *47*, 1523–1536. [CrossRef]
41. Mohan, C.; Haderle, D.; Lindsay, B.; Pirahesh, H.; Schwarz, P. ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM Trans. Database Syst. (TODS)* **1992**, *17*, 94–162. [CrossRef]