A Survey on Epoch-based In-Memory Database Systems

Lan Anh Nguyen, Sangjin Lee, Hyung Tae Lee, and Yongseok Son
Department of Computer Science and Engineering, Chung-Ang University

***Abstract*—In-memory databases has emerged as the most favorable modern database replacing with traditional ones to sufficiently handle with storing, retrieving, and processing large collections of data in big data applications. High speed in storing, retrieving, processing of in-memory databases makes itself outperform classic databases and becomes the most prominent database nowadays. However, along with rapid growth of collected data from intensive-workload applications, in-memory databases should achieve an exceptional scalability to deal with the huge amounts of received data. In addition, they must overcome their internal disadvantages regarding to durability and persistence. Epoch-based mechanisms are suitable solutions for scalability, durability, and persistence issues, while maintaining excellent performance in speed of in-memory databases. This study provides a survey of epoch-based in-memory databases in terms of their database management architecture, classification of epoch-based mechanisms used in, and current developments of epoch-based in-memory database systems.**

***Index Terms*—In-memory database, big data, storage engine, epoch-based mechanisms,**

I. Introduction

In data-driven world todays, databases play a critical role, storing, retrieving, processing massive data with low-latency. In-memory database has become the best candidate for the database position in data-driven or big data applications. This type of database is a modern data management system, storing data in the system's main memory (*e.g.,* most often RAM) [1]. It works as the next generation of databases and opposes to traditional databases that use disks for storing. Popular in-memory databases used in data-driven applications are MongoDB Atlas [2], Redis [3], Aerospike [4], Silo [5], InluxDB [6], KeyDB [7], VoltDB [8], *etc.*

Basically, unlike on-disk databases, there is no need to perform disk I/O to query and update data in in-memory databases, helping to access data faster. Additionally, it also provides flexibility, reliability, high availability, and instantaneous ACID (*e.g.,* Atomic, consistent, isolated, and durable) adherence [1]. Therefore, in-memory databases are ideal for applications demanding real-time responses, such as IoT applications, gaming, session management, read-heavy applications, edge AI/ML, fraud detection, *etc.* However, it faces with some drawbacks, such as high cost, lack of data persistence, risk of data loss, and architecture complexity. A comparison between in-memory and on-disk database management systems (DBMS) is provided in Table I.

As shown in Table I, scalability, durability, and persistence are serious issues in in-memory database management systems (I-DBMS), which have to be addressed carefully. These issues

TABLE I
A COMPARISON OF IN-MEMORY AND ON-DISK DBMS

	In-Memory DBMS	**On-Disk DBMS**
Primary storage medium	Memory	Disk
Programming	Simpler	Quite difficult
Performance	Extremely fast	Slower
Scalability	Limited by available memory or limited if not use distributed nodes	Possible
Durability and Persistence	Not well guaranteed	Guaranteed
Data volume	Limited by available memory	Much larger
Cost	Higher	Lower
Use cases	Applications require low-latency or real-time access	Applications require to handle large datasets

are directly related to two critical units in the storage engine of I-DBMS, transaction and recovery managers. To be specific, the transaction manager schedules transactions to guarantee a logically consistent state of the database [9]. The recovery manager keeps the operation log and is responsible for restoring the system in case of a failure or crash [9].
Besides, big data applications generate massive concurrent operations (*i.e.,* reading, writing, renaming, deleting, *etc.*). To control the concurrency in this case, transaction and lock managers in the storage engine of I-DBMS cooperate to guarantee logical and physical data integrity while ensuring efficient executions of concurrent operations [9]. Like transaction and recovery managers holding particular responsibility in the storage engine, the lock manager has its own duty. It keeps locks on the database objects for running transactions, which ensures that concurrent operations can not violate physical data integrity [9].

To solve these issues in in-memory databases, epoch-based mechanisms are one of appropriate solutions. The epoch concept has been widely utilized in the context of operating systems and computational systems. An epoch refers to a period in which modifications take place. According to epoch-based mechanisms, operations of a system can be divided into distinct epochs, in other words, epoch works as a boundary

to ensure the serialization requirement of the system. Epoch-based mechanisms raise some potential advantages, such as efficient resource management, working as synchronization points in synchronization, deallocation points in garbage collection of memory management, and fault tolerance (*e.g.,* data can be checkpointed or replicated at epoch boundaries, making easier to recover from failure or crash).
There are various works recently employing epoch-based mechanisms to improve system performance (*e.g.,* popularly in databases and operating systems). In term of operating systems, [10] proposes and implements a portable and scalable synchronization framework for filesystems by designing an epoch-based synchronization mechanism. Also in operating systems, [11] introduces a barrier-enabled I/O stack, which consists of three key components, an epoch-based I/O scheduling, an order-preserving dispatch, and a dual-mode journaling, to eliminate the overhead from the storage order preserving in modern I/O stack. In term of databases, [12] addresses the log redundancy (*e.g.,* double-logging) in log-structured merge-tree based relational databases. In this study, the double-logging is removed and replaced with a passive data persistence. The persistence mechanism is based on epoch, parallelly commiting and flushing log items on multiple memory buffers according to epoch boundaries.

Epoch-based mechanisms have recently been also applied into in-memory databases. For more details, they are used in various cases, such as transactions in multicore databases [13], [14], logging for data persistence and recovery [15], [16], garbage collection in memory management [17], [18], and replication in distributed databases [19], *etc.* These epoch-based mechanisms will be considered and analyzed in this study.

Addressing the significance of epoch-based mechanisms and the lack of a survey on epoch-based fundamental approaches in in-memory database management systems, we contributes:

- First, we provide an architecture of an in-memory database management system (I-DBMS) based on conventional database management systems. Additionally, main steps in running of in-memory storage engine (I-SE) are also analyzed in this study
- Second, we specify three main approaches of epoch-based mechanisms used in I-SE
- Finally, we contribute a survey of recent studies on epoch-based in-memory databases

This study is organized as follows:
The section II shows an architectural overview and main steps of in-memory storage engine. Furthermore, it also provides a classification of epoch-based mechanisms in in-memory databases. A survey of recent studies on epoch-based in-memory databases is shown in section III. Conclusions are provided in the last section.

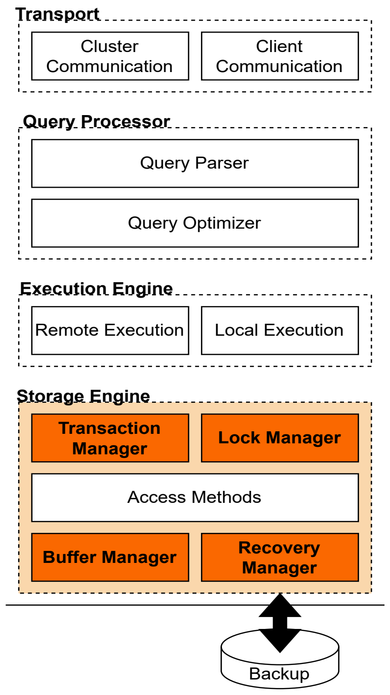

Fig. 1. Architecture of an in-memory database management system

II. An overview of epoch-based in-memory databases

A. *Architecture of epoch-based in-memory database management system (I-DBMS)*

To get an overview of epoch-based in-memory databases, we firstly provide the architecture of an in-memory database management system (I-DBMS, *e.g.,* sometimes called "main memory database management system") in Figure 1. Basically, I-DBMS maintains the common architecture of a DBMS explained in [9]. The architecture contains four main engines, such as transport, query processor, execution engine, and storage engine. Unlike on-disk DBMS using disks as the primary storage, I-DBMS uses disks as a backup device for persistence and recovery.
This study focuses on epoch-based mechanisms for storage, so we dive deeply only the in-memory storage engine (I-SE) of the architecture in Figure 1. I-SE in I-DBMS consists of five main components with dedicated responsibilities [9]:

- **Transaction manager:** schedules transactions to guarantee a logically consistent state of the database
- **Lock manager:** locks on the database objects for running transactions, ensuring that concurrent operations can not violate physical data integrity
- **Access methods:** manage accesses and organize data on main memory. They include files and storage structures
- **Buffer or resource manager:** manages data pages or resources in memory
- **Recovery manager:** keeps the operation log and is responsible for restoring the system in case of a failure or crash

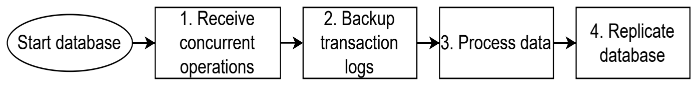

Fig. 2. Main steps in operation of storage engine in I-DBMS

However, considering the epoch-based concept, the I-SE in I-DBMS usually employs epoch-based mechanisms for transactions, recovery, and resource management, directly relating to the transaction manager, buffer or resource manager, and recovery manager. We consider these components as:

- Epoch-based transaction manager
- Epoch-based buffer or resource manager
- Epoch-based recovery manager

While the transaction manager along with the lock manager affect the concurrency control, synchronization, consistency, and scalability, the buffer manager is responsible for managing pages or resources in memory, relating to garbage collection, and reclamation. The recovery manager is responsible for logging to the backup device, and replicating to distributed nodes to guarantee the persistence and recovery in case of a failure or crash. Therefore, we highlight main components (*e.g.,* transaction manager, buffer manager, and recovery manager) that can employ the epoch concept to improve performance system in Figure 1.

Figure 2 shows four main steps of I-SE, which are managed by epoch-based components in I-SE as follows:

1) Receiving concurrent operations (*i.e.,* write, read, rename, delete, *etc.*), handled by the epoch-based transaction manager
2) Backuping transaction logs, handled by the epoch-based recovery manager
3) Processing data, handled by the epoch-based buffer or resource manager
4) Replicating data records, handled by the epoch-based recovery manager

In the following part, we provide a classification of epoch-based mechanisms used by components in I-SE to improve the system performance.

*B. **Epoch-based mechanisms used in in-memory storage engine (I-SE)***

According to the above analysis on the architecture of I-DBMS, main steps of I-SE, and epoch, we propose a classification of epoch-based mechanisms used in I-SE in Figure 3. In this classification, we consider which components utilizing the epoch concept to categorize epoch-based mechanisms in I-SE. Therefore, there are three main epoch-based approaches in our classification, such as epoch-based transaction management (*e.g.,* on epoch-based transaction manager), epoch-based buffer or resource management (*e.g.,* on epoch-based buffer or resource manager), and epoch-based recovery management approaches (*e.g.,* on epoch-based recovery manager).
In the following part, we provide an analysis on the three approaches.

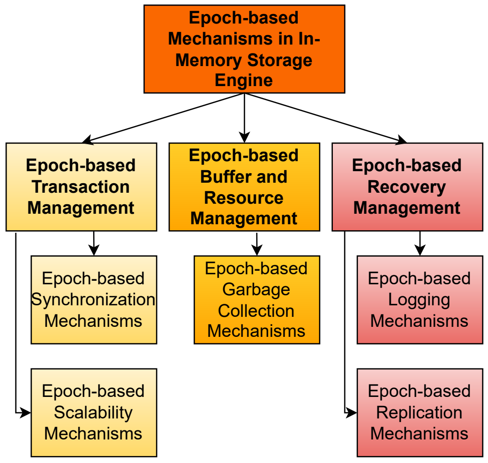
Epoch-based Mechanisms in In-Memory Storage Engine
Epoch-based Transaction Management
Epoch-based Buffer and Resource Management
Epoch-based Recovery Management
Epoch-based Synchronization Mechanisms
Epoch-based Garbage Collection Mechanisms
Epoch-based Logging Mechanisms
Epoch-based Scalability Mechanisms
Epoch-based Replication Mechanisms

Fig. 3. Classification of epoch-based mechanisms used in in-memory storage engine

*1) **Epoch-based transaction management:*** As explained in II-A, transactions should be coordinated, scheduled to maintain the consistent state of in-memory databases. Therefore, synchronization mechanisms guaranteeing the consistency are critical in databases.
Epoch-based synchronization mechanisms are appropriate and efficient to guarantee the consistency requirement of in-memory databases. In these mechanisms, epochs work as boundaries or synchronization points for updates or modifications of data.
Besides, nowadays, to handle intensive workloads from big data applications, transactions in in-memory databases should be scalable to deal with the issue. Utilizing the potential of multi-core processing along with epoch-based mechanisms, transactions in in-memory databases can perform the scalability while guaranteeing the consistency of the system.
As a result, for the epoch-based transaction management approach, there are two main types of epoch-based mechanisms:

- Epoch-based synchronization mechanisms
- Epoch-based scalability mechanisms

*2) **Epoch-based buffer or resource management:*** As explained in II-A, the buffer or resource manager in in-memory databases is responsible for resource management for effective resource allocation and deallocation.
Also utilizing epochs as synchronization points or stability points, the buffer or resource manager can prevent resource leaks or inefficient resource allocation. There is a popular term in resource management in databases, called "garbage collection".
The garbage collection refers to the process in databases of identifying and removing data that is no longer used or referenced by databases. This process helps to reclaim resource space and guarantee efficient database performance.
Maintaining the above operation of the garbage collection, epoch-based mechanisms enable to collect or reclaim no longer used resources at epoch boundaries, ensuring that memory is efficiently managed without causing memory leaks.

TABLE II
RECENT STUDIES ON EPOCH-BASED IN-MEMORY DATABASES

	Silo [13]	**Scalable Replication** [14]	**LeanStore** [17]	**Low-latency Logging** [15]	**Instant Recovery** [16]	**Deuteronomy** [18]	**Commit and Replication** [19]
Scalability	✓	✓					
Synchronization	✓						✓
Garbage collection			✓			✓	
Logging				✓	✓		
Replication		✓					✓

As a result, for the epoch-based buffer or resource management approach, epoch-based mechanisms can be considered as:

- Epoch-based garbage collection mechanisms or epoch-based reclamation mechanisms

*3) **Epoch-based recovery management**:* As explained in II-A, the recovery manager manages operation logs or transaction logs for restoring the system in case of a failure or crash, ensuring the data persistence and durability of databases.
Additionally, this component in I-SE is also responsible for replicating critical data, system states to distributed nodes to guarantee the fault tolerance of databases.
Epoch-based logging and replication mechanisms are satisfactory, maintaining to efficiently log transaction logs and replicate data at epoch boundaries.
As a result, for the epoch-based recovery management approach, there are two main types of epoch-based mechanisms:

- Epoch-based logging mechanisms
- Epoch-based replication mechanisms

III. CURRENT DEVELOPMENTS IN EPOCH-BASED IN-MEMORY DATABASES

There are various studies utilizing the epoch concept to improve performance of in-memory databases, such as [13]–[19]. A summary of these works is shown in Table II, which classifies them into specific epoch-based mechanisms, such as scalabilty, synchronization, garbage collection, logging, and replication, as explained in II-B. For more details:

- Silo [13]: This in-memory database system uses epoch as a serialization point to identify transactions, guaranteeing the synchronization of system. In addition, the in-memory database uses periodically-updated epochs, improving it scalability.
- Scalable Replication [14]: This database system uses epochs to batch transactions in an actual commit order. Hence, batch of transactions in an epoch can be replayed on the back up database while guaranteeing the order of transaction. On the other hand, because a batch of transaction in each epoch is replayed to the back up, which is different from replaying one by one transaction, the scalability of this database is also improved.
- LeanStore [17]: This in-memory database designs an epoch-based reclamation. It has a global epoch growing periodically. Threads wanting to evict or delete a page will be assigned a local epoch. Before pages are actually evicted, they are assigned local epochs. A comparison between local epochs of the threads and pages will be considered to guarantee a safe page eviction.
- Low-latency Logging [15]: This study proposes a logging improvement for Silo database [13]. Epochs are used as persistence points for concurrent logging threads. As a result, the persistence is guaranteed via epoch-based logging. In addition, using concurrent logging scheme raises a low latency logging.
- Instant Recovery [16]: This study addresses the time bottleneck during the recovery phase of in-memory database systems. Epoch is used to index checkpoints without waiting, making the recovery phase to be instant.
- Deuteronomy [18]: Epochs in this study work as stability points to guarantee entries in the garbage list safely reused. Entries, added to the garbage list, are assigned local epochs. Running threads are also assigned local epochs. According to their local epochs, entries in the garbage list can be considered as unlinked or not.
- Commit and Replication [19]: Epochs in this study are used as commit points for transactions in the systems. Additionally, within an epoch, transactions are replicated to distributed nodes for an efficient replication.

IV. CONCLUSIONS

In this study, we firstly provide an architecture of in-memory database management system based on conventional database management system and main steps in running of in-memory storage engine, which utilize the epoch concept to accelerate the system performance. Additionally, we propose a classification of epoch-based mechanisms used in in-memory databases. Finally, we show a summary of recent studies on epoch-based in-memory databases.
Eventhough in-memory database performs as a next-generation database for big data applications and can be boosted by epoch-based mechanisms, it still struggles with some challenges, such as official epoch-based in-memory databases and a shortage of comprehensive survey of epoch-based mechanisms for in-memory databases. Future studies

on epoch-based in-memory databases should deal with these challenges carefully, promoting a profound development of in-memory databases.

Acknowledgement

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT), KIAT grant funded by the Korea Government (MOTIE), and Korea Institute of Science and Technology Information (KISTI). (No. NRF-2021R1C1C1010861, NRF-2022R1A4A5034130, KIAT-P0012724) (Corresponding Author: Yongseok Son).

References

[1] What is an in-memory database? https://www.voltactivedata.com/blog/2021/06/whats-an-in-memory-database/. Accessed: 2023-08-13.
[2] Mongodb atlas. https://www.mongodb.com/atlas. Accessed: 2023-08-13.
[3] Redis. https://redis.io/. Accessed: 2023-08-13.
[4] Aerospike. https://aerospike.com/. Accessed: 2023-08-13.
[5] Silo. https://dbdb.io/db/silo. Accessed: 2023-08-13.
[6] Influxdb. https://www.influxdata.com/db/. Accessed: 2023-08-13.
[7] Keydb. https://docs.keydb.dev/. Accessed: 2023-08-13.
[8] Voltdb. https://dbdb.io/db/voltdb. Accessed: 2023-08-13.
[9] A. Petrov. *Database Internals*. O'Reilly Media, Incorporated, 2019.
[10] Woong Sul, Heon Y Yeom, and Hyuck Han. montage: Nvm-based scalable synchronization framework for crash-consistent file systems. *Cluster Computing*, 24(4):3573–3590, 2021.
[11] Youjip Won, Jaemin Jung, Gyeongyeol Choi, Joontaek Oh, Seongbae Son, Jooyoung Hwang, and Sangyeun Cho. {Barrier-Enabled}{IO} stack for flash storage. In *16th USENIX Conference on File and Storage Technologies (FAST 18)*, pages 211–226, 2018.
[12] Kecheng Huang, Zhaoyan Shen, Zhiping Jia, Zili Shao, and Feng Chen. Removing {Double-Logging} with passive data persistence in {LSM-tree} based relational databases. In *20th USENIX Conference on File and Storage Technologies (FAST 22)*, pages 101–116, 2022.
[13] Stephen Tu, Wenting Zheng, Eddie Kohler, Barbara Liskov, and Samuel Madden. Speedy transactions in multicore in-memory databases. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 18–32, 2013.
[14] Dai Qin, Angela Demke Brown, and Ashvin Goel. Scalable replay-based replication for fast databases. *Proceedings of the VLDB Endowment*, 10(13):2025–2036, 2017.
[15] Masahiro Tanaka and Hideyuki Kawashima. Stable low latency logging for epoch-based in-memory database. In *2022 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 167–170, 2022.
[16] Leon Lee, Siphrey Xie, Yunus Ma, and Shimin Chen. Index checkpoints for instant recovery in in-memory database systems. *Proceedings of the VLDB Endowment*, 15(8):1671–1683, 2022.
[17] Viktor Leis, Michael Haubenschild, Alfons Kemper, and Thomas Neumann. Leanstore: In-memory data management beyond main memory. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 185–196. IEEE, 2018.
[18] Justin Levandoski, David Lomet, Sudipta Sengupta, Ryan Stutsman, and Rui Wang. High performance transactions in deuteronomy. In *Conference on Innovative Data Systems Research (CIDR 2015)*, 2015.
[19] Yi Lu, Xiangyao Yu, Lei Cao, and Samuel Madden. Epoch-based commit and replication in distributed oltp databases. 2021.