MosaicDB: An Efficient Trusted / Untrusted Memory Management for Location Data in Database

Tomoya Suzuki*, Taisho Sasada†, Yuzo Taenaka‡ and Youki Kadobayashi§
Division of Information Science, Nara Institute of Science and Technology

Email: *suzuki.tomoya.sp9@is.naist.jp, †sasada.taisho.su0@is.naist.jp, ‡yuzo@is.naist.jp, §youki-k@is.naist.jp

***Abstract*—Location data has been used for various purposes in digitized society but includes waypoints directly related to personal privacy, such as home addresses. To hide such sensitive waypoints, some applications provide Endpoint Privacy Zones (EPZs) that keep a portion of the track secret. However, most service providers placing databases on a cloud face the potential risk of exposing sensitive waypoints to cloud service providers. Existing studies have proposed databases using Trusted Execution Environments (TEE) that protects sensitive data in a trusted and completely secure memory region. However, as TEE inevitably limits the size of the trusted memory, databases proposed in these studies cannot use the trusted memory efficiently due to the fundamental design that handles all data in the trusted memory. Moreover, the memory outside of the trusted memory, called untrusted memory, remains vacant even if the trusted memory is fully used, thereby leading to insufficient memory utilization on a whole system and decreased database performance. In this study, we propose MosaicDB, a memory-efficient and trusted database for location data, using both trusted and untrusted memory. To enhance memory utilization efficiency, MosaicDB handles only sensitive waypoints within the EPZ in the trusted memory while handling non-sensitive waypoints in the untrusted memory. Experimental results show that MosaicDB improves memory utilization efficiency, thereby achieving a 25% reduction in execution time for selection queries compared to the database that handles all data in the trusted memory.**

Keywords*—Database; Trusted Execution Environment; Intel SGX; Location data; Endpoint Privacy Zones; Cloud Computing.

I. Introduction

In recent years, the popularity of Fitness Tracking Social Networks (FTSNs) has expanded as the number of health-conscious people has increased. FTSNs allow users to track outdoor activities and share their routes with other users. By sharing routes, users can enhance the enjoyment of their activities and maintain their motivation. However, sharing routes also raises privacy concerns, as other users can browse routes that often include sensitive waypoints, such as homes or workplaces.

FTSNs enable users to designate Endpoint Privacy Zones (EPZs) to prevent privacy leakage from sharing routes. An EPZ allows users to hide some routes, as shown in Figure 1. Ongoing research is also being conducted to facilitate the establishment of more robust EPZs [1][2]. That is, sensitive waypoints are protected on an application. However, waypoints on a database still suffer from an exposure risk in a cloud environment. As modern application services, including database systems, are deployed in a cloud environment, waypoints in the database may be stolen by the Cloud Service Provider (CSP) with the highest privileges on the cloud system. Although existing databases offer disk-level encryption to protect sensitive data, a CSP may steal unencrypted data or encryption keys directly from memory, leading to significant privacy leakage, as shown in Figure 2.

To overcome these threats, databases using Trusted Execution Environments (TEE) have been proposed [3][4][5][6]. TEE creates a trusted region in memory using hardware-level security mechanisms. Any privileged software, such as an operating system and hypervisor, cannot directly read and write the confidential data managed by a database since the trusted memory is completely isolated from the main (untrusted) memory. However, as TEE severely restricts the size of the trusted memory, databases proposed in these studies are now facing a challenge of performance degradation due to a shortage of the trusted memory. This challenge arises from their fundamental design of handling all data in the trusted memory.

In this study, we propose MosaicDB, a memory-efficient and trusted database for location data. MosaicDB selectively handles location data (waypoints) in the trusted memory, following the necessity of data protection in the application context. As an application conceals all waypoints within EPZs and exposes the remaining waypoints, we only need to protect waypoints within EPZs. Moreover, the number of waypoints within EPZs is less than that outside the EPZs. We utilize this characteristics of location data and thus attempt to efficiently use both trusted and untrusted memory. That is, MosaicDB only protects waypoints within EPZs in the trusted memory so that they are not exposed. As other waypoints outside EPZs are already public on an application, MosaicDB handles them without any special treatment in the untrusted memory. This data management method based on application context allows us to synchronously protect sensitive data that users do not want to be exposed in the database, and to effectively utilize both trusted and untrusted memory on the database server.

The structure of this paper is as follows: In Section II, we provide an overview of the fundamental features of Intel Software Guard Extensions (SGX), TEE employed in this study. In Section III, we outline related works on databases utilizing TEE and their challenges. In Section IV, we explore the detailed design of MosaicDB and query processing flow. In Section V, we present the results of the experimental evaluation. In Section VI, we describe the limitations of MosaicDB and outline future directions. Finally, in Section VII, we conclude the paper by summarizing the contributions of this research.

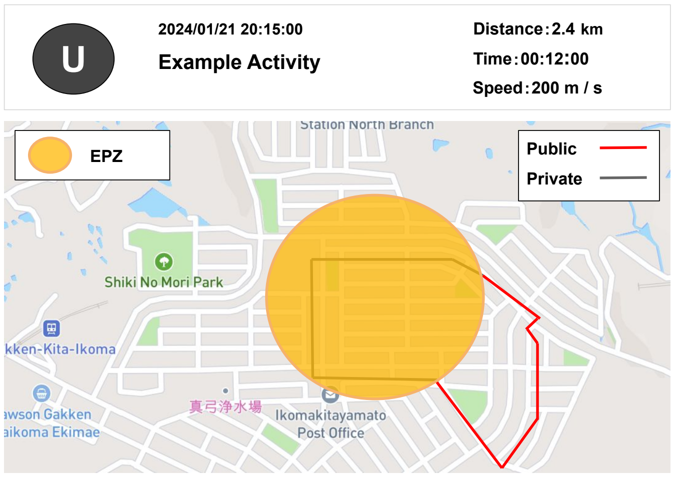

Figure 1. Example of EPZ

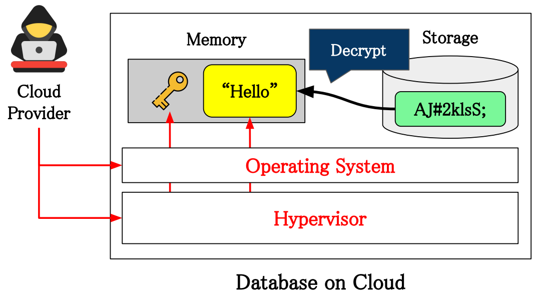

Figure 2. Threat of data theft by malicious cloud providers

II. Intel SGX

We protect the database using Intel SGX, the most widely used TEE in cloud environments. SGX is installed in 6th generation Intel CPUs and later, which creates a trusted region (known as enclaves) in memory. SGX achieves the confidentiality and integrity of sensitive processes using the encrypted and isolated physical memory known as the enclave page cache (EPC). CPUs fully control access to the enclave, thus preventing attacks on processes from privileged software, such as the operating system or hypervisor. This powerful protection provided by SGX enables cloud users to safely execute processes in untrusted cloud environments. The following subsections describe a limitation of SGX, the sealing feature for secure data persistence, and the remote attestation feature for secure communication between a database and an application server.

A. Hardware limitation of the enclave

One of the major limitations of Intel SGX is the size of enclave memory. Intel CPUs up to 10th generation support a maximum enclave memory size of 128 MB to 256 MB, while Xeon scalable processors from the 3rd generation and later support a maximum enclave memory size of 512 GB. When the enclave memory footprint exceeds this size limitation, it leads to highly inefficient EPC paging, significantly decreasing the performance of processes runnning in enclaves. Although the available enclave memory size has increased in 3rd generation and later Xeon scalable processors, the available enclave memory size will be smaller depending on the system configuration. For example, in Microsoft Azure's DCsv3 series, only 16 GB of enclave memory is available for a virtual machine with a total of 32 GB memory. Therefore, if the database uses only enclave memory, the non-enclave memory cannot be used, resulting in inefficient memory utilization.

B. Data persistence with sealing

Since the enclave is a memory region allocated to a process, the data in the enclave will be lost when the process is stopped. To address this, SGX provides sealing and unsealing functions to persist data in storage securely. These functions enable the encryption and decryption of protected data using an enclave-specific key. However, cryptographic operations in sealing incur significant overhead, requiring SGX applications to minimize the frequency of sealing operations whenever possible. In this study, we minimize this overhead by exclusively protecting location data within the EPZ through sealing.

C. Authentication and key exchange with remote attestation

Remote attestation [7] is an authentication protocol that mutually verifies the integrity of enclaves between SGX applications. It verifies the integrity of enclaves with which it communicates and exchanges the key used to encrypt the communication content, thereby ensuring secure communication between SGX applications. Remote attestation is available only from codes in the enclave. This study uses this protocol to secure communication between the application servers and databases.

III. Related work

Several databases using SGX have been proposed [3][4][5][6]. CryptSQLite [3] protects all the data in the enclave. Its design is straightforward; however, it is suitable only for cases involving small-scale data because the memory load in the enclave increases as the data size increases.

EnclaveDB [4] uses SGX to protect tables managed in memory. The simple design of protecting the entire table makes it easy to integrate into existing RDBMSs; however, this consumes a significant amount of the enclave memory while making it impractical to utilize untrusted memory efficiently.

StealthDB [5] addressed the problem of excessive enclave memory usage by protecting only primitive operators (e.g., $\geqq, \leq, +, *$) that process unencrypted data in the enclave. The amount of memory used by the operations remains almost constant, minimizing the utilization of the enclave memory in all database query processing and acheiving high memory utilization efficiency. However, an increase in the transitions between the enclave and untrusted memory significantly degrades database performance.

Yoshimura et al. [6] proposed an RDBMS designed to reduce enclave memory usage and transitions between the enclave and untrusted memory by protecting only specific columns in the enclave. This approach boosts the memory utilization

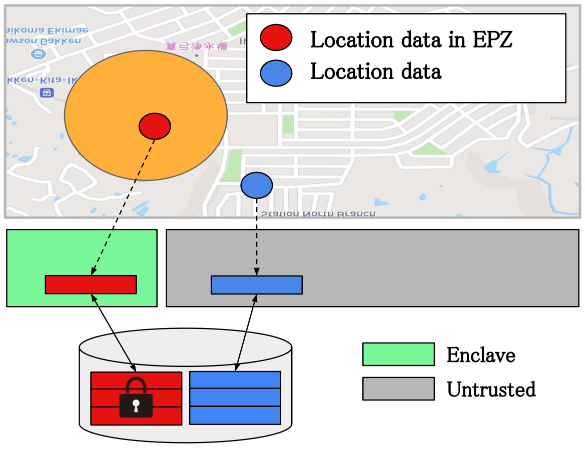

Figure 3. Concept of MosaicDB

efficiency as long as there are few columns under protection. However, when dealing with location data that includes columns such as latitude, longitude, and time, nearly all columns require protection, which can decrease the memory utilization efficiency. This method also does not allow selective protection of specific data (row) based on application context. This leads to unnecessary protection of non-sensitive data in the specific application context.

Based on the above considerations, we propose a memory-efficient and trusted database that uses both enclave and untrusted memory, focusing on location data.

IV. MosaicDB

In this section, we describe our threat model, fundamental design, and detailed query execution flow in MosaicDB.

A. Threat model

To design the architecture of MosaicDB, we define our threat model. Our threat model assumes a CSP as an main adversary able to access all the software (such as an operating system and hypervisor) and hardware except for enclaves. A CSP can continuously access the database's memory using memory dump or cold boot attacks. Since we assume that a CSP cannot tamper with the code and data in the enclave or the SGX hardware, attacks on SGX hardware [8][9][10] are beyond the scope of this study. Note that while we assume a malicious CSP is the most critical threat, any malicious software, such as malware in the database server, is also considered a threat in this study. Furthermore, we assume that FTSN users appropriately keep sensitive waypoints private using their defined EPZs. Therefore, attacks that attempt to identify sensitive waypoints using publicly disclosed waypoints discussed in [1][2] are beyond the scope of this study.

B. MosaicDB concept and archtecture

A concept of MosaicDB is shown in Figure 3. MosaicDB capitalizes on the fact that location data outside the EPZ is public in the application context, reducing the need to handle it as sensitive data in the database. When inserting location data, MosaicDB checks whether location data is included within the EPZ by calculating the geographical distance between the center of the EPZ stored in MosaicDB and location data. In all operations performed by MosaicDB, the location data within the EPZ are managed in the enclave, whereas other location data is managed in the untrusted memory. MosaicDB manages location data by utilizing both memory and storage, similar to typical RDBMSs. MosaicDB encrypts the location data loaded into the enclave by using the sealing before persistence.

The architecture of MosaicDB is shown in Figure 4. MosaicDB has general components in the typical RDBMSs, such as the parser, planner, executor, and storage engine. To realize location data management using both the enclave and untrusted memory, we duplicate the executor, responsible for query execution, and the storage engine, which accesses data on buffers and storage, respectively. The executor and storage engine within the enclave handle queries involving sensitive location data, whereas those in untrusted memory execute queries related to non-sensitive location data. The parser, which generates an abstract syntax tree from a query string; the planner, which generates an execution plan; and the preprocessor, which checks whether the location data is contained within the EPZ, are placed in the enclave because they handle unencrypted location data that may or may not be within the EPZ. The following subsections describe the query execution flow using these components.

C. Query execution flow

We will describe the query execution flow of MosaicDB by dividing it into three parts. Note that MosaicDB is currently designed to handle only simple CRUD (CREATE, INSERT, UPDATE, DELETE) queries, and supporting more complex queries is future work.

1) Query analysis and optimization

In the initial stage of query execution, MosaicDB analyzes queries and creates optimized execution plans. The network module (① in Figure 4) first receives the encrypted query string from the client, and the decryptor (② in Figure 4) decrypts the query string in the enclave. The keys used for encryption and decryption are exchanged via remote attestation when the client connects to the database. Then, the parser (③ in Figure 4) generates a query tree, which is an abstract syntax tree, from the query string and passes it to the planner. Finally, the planner (④ in Figure 4) generates a plan tree, which is an optimized execution plan, from the query tree and passes it to the preprocessor (⑤ in Figure 4).

2) Checking location data within the EPZ

If the query is `INSERT`, the preprocessor checks whether the location data in the plan tree is contained within the EPZ. If contained within the EPZ, it passes the plan tree to the trusted executor; otherwise, it passes it to the untrusted executor (⑥ in Figure 4). If the query is not `INSERT`, the checking process is ignored because the plan tree of a query like `SELECT * FROM locations;` does not contain location data, and the plan tree is passed to both the trusted and untrusted executors. EPZs are

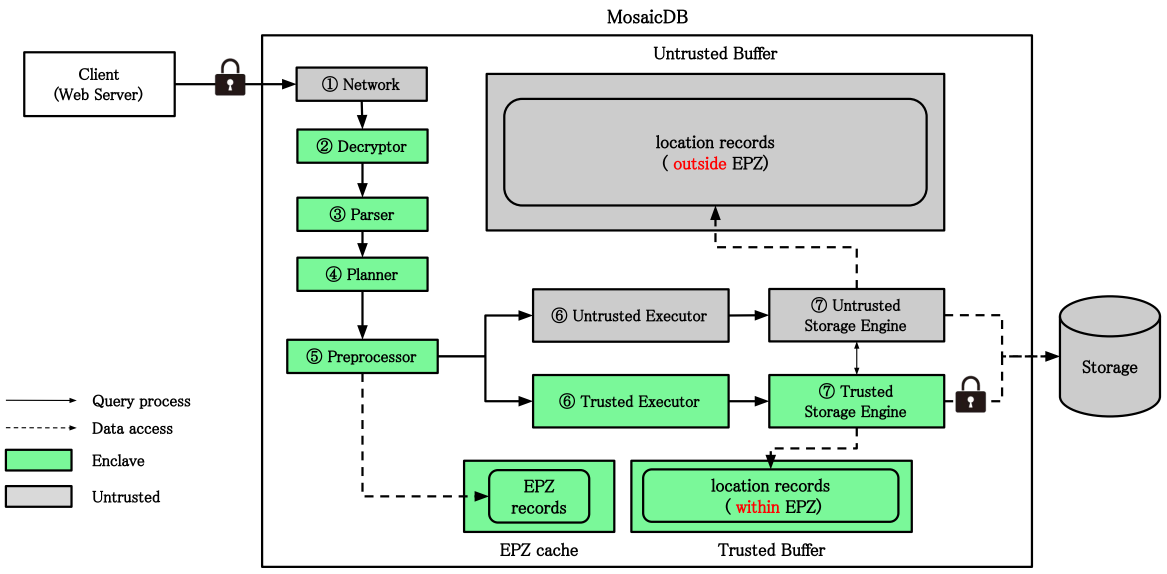

Figure 4. Overview of MosaicDB

cached in memory as an dictionary with user ID as a key (*EPZ cache* in Figure 4) because reading EPZ records on the storage has an extra cost because of sealing. Through an evaluation, we will confirm that the EPZ cache can effectively mitigate overhead, even with an increased number of EPZs.

*3) **Processing the plan tree in the enclave and untrusted memory***

The trusted and untrusted executors execute queries according to the plan tree. The execution process performed by the trusted and untrusted executors is largely similar to a typical RDBMS, but there is a difference in terms of the query execution results. In MosaicDB, if the execution of either the trusted or untrusted executor fails, the final query execution result is a failure. This allows data updates caused by failed transactions to be rolled back, thereby preventing data inconsistency.

The trusted and untrusted storage engines (⑦ in Figure 4) engines insert, scan, update, and delete location data in memory and storage according to requests from the executor. The location data is managed in buffers (*Trusted / Untrusted Buffer* in Figure 4), which are located in the enclave and untrusted memory respectively. They are organized into fixed-length blocks called pages. The location data within the EPZ are stored in a page in the enclave (*Secure page* in Figure 5), and other location data is stored in a page in the untrusted memory (*Normal page* in Figure 5). The secure page is encrypted using the sealing and is decrypted only in the enclave, so an attacker cannot steal the location data in the secure page. The metadata page in the normal buffer is a page that stores metadata such as the page IDs of secure / unsecure pages, and is referenced from both the enclave and untrusted memory. After query execution, the executors return the result to the client via the network module. In the case of queries that need to return records, the merged records obtained in the trusted and untrusted executors are encrypted and returned.

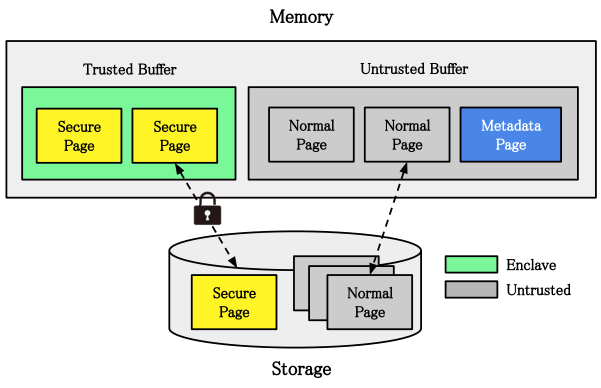

Figure 5. Page management in MosaicDB

V. Evaluation

To evaluate the overhead and memory utilization efficiency of MosaicDB, we compare the execution times and memory usage of INSERT and SELECT queries between MosaicDB and the baseline, where all the location data is managed in the enclave. In the current implementation, the execution flows of UPDATE and DELETE queries are almost the same as those of a typical RDBMS; therefore, we exclude these queries from the evaluation.

We used an Intel(R) NUC Kit NUC7PJYH as the experimental environment. The CPU is an Intel(R) Pentium(R) Silver J5005, the memory is 16 GB, and the storage is 256 GB. For the experiments, we used Geolife trajectory datasets [11] provided by Microsoft from which we extracted the amount of data required for each experiment. All EPZs in the experiment were set to be within 5 km of Peking University, where the location data in the dataset is concentrated.

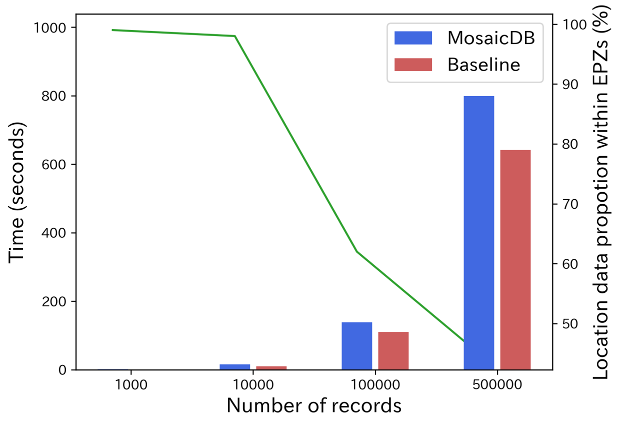

Figure 6. Execution time of INSERT

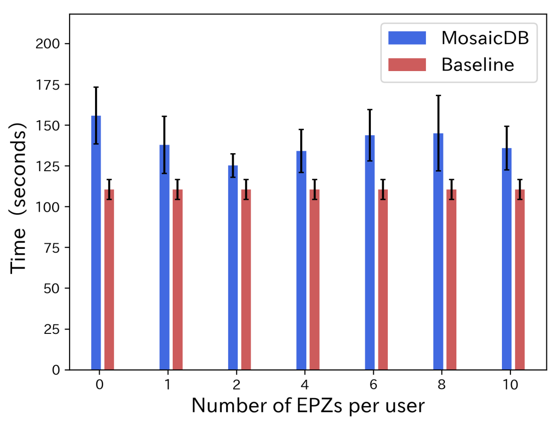

Figure 7. Execution time of INSERT while varying number of EPZs

A. Execution Time

We evaluated the execution times of INSERT and SELECT queries while varying the data size. We conducted two different experiments for INSERT queries. One experiment involved keeping the number of EPZs fixed while varying the data volume, and the other involved maintaining a fixed data volume while changing the number of EPZs. In the former experiment, we fixed the number of EPZs to one. In the latter experiment, the maximum number of EPZs was set to ten, which aligns with the typical number of EPZs expected in general FTSNs. We used 100,000 records in the latter experiment, and the execution times were averaged over ten runs in both experiments. Finally, the green line in Figure 6, 8 represents the proportion of location data within the EPZ.

Figure 6 shows that MosaicDB increases an overhead by 1.2 to 1.6 times compared to the baseline in INSERT query. On the other hand, an increase in the number of EPZs resulted in minimal additional overhead, as shown in Figure 7. This indicates that the increase in the number of EPZs can be reduced by EPZCache, while there is some overhead in MosaicDB. When there are no EPZs, MosaicDB's execution time is at its slowest because all location data is processed in the untrusted memory. Query processing in the untrusted memory involves additional overhead, such as plan tree serialization and deserialization, as well as additional transitions between the enclave and untrusted memory, making it more costly than executing queries in the enclave.

Figure 8 shows that MosaicDB can reduce SELECT query execution time by up to 25% compared to the baseline. MosaicDB achieves this performance improvement due to the reduced overhead of sealing, as it does not encrypt location data outside the EPZ, unlike the baseline.

B. Memory Usage

We estimated the memory usage of MosaicDB while varying the amount of data stored in the enclave and untrusted memory for INSERT and SELECT queries. Since MosaicDB determines whether to store location data in the enclave or untrusted memory during insertion, the amount of location data stored in the enclave or untrusted memory remains constant for INSERT and SELECT queries. Consequently, we calculate the memory usage as the product of the number of records and the size of a record.

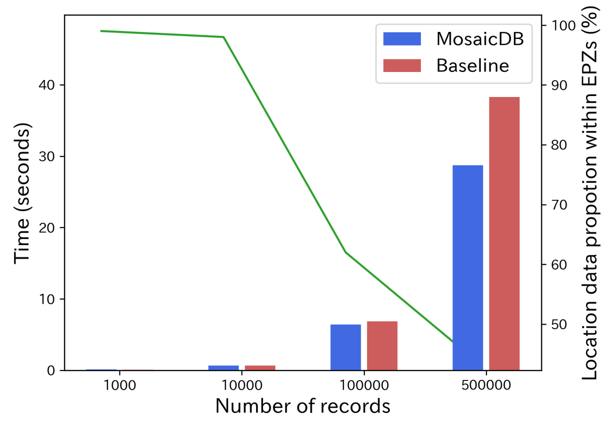

Figure 8. Execution time of SELECT

Figure 9 shows that both the enclave and untrusted memory are used when the number of records exceeds 100,000. This observation aligns with the trend in Figure 6 and Figure 8, where the proportion of location data within the EPZ decreases as the number of records increases. It suggests that if the proportion of location data within the EPZ is sufficiently small, MosaicDB can effectively utilize the untrusted memory while keeping the enclave memory usage in check. For instance, if the proportion of location data within the EPZ is around 10%, approximately 90% of location data can be accommodated in the untrusted memory. Thus, we conclude that MosaicDB allows for more efficient utilization of the server's memory compared to conventional methods that manage all data in the enclave.

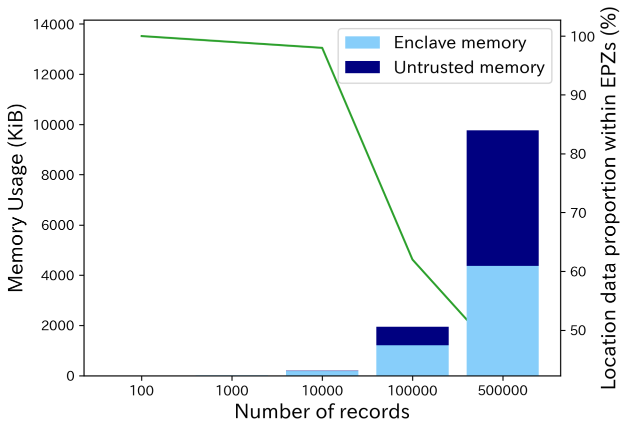

Figure 9. Enclave / untrusted memory usage in MosaicDB

VI. Discussion

In this section, we discuss the necessity of additional evaluation experiments to demonstrate the practicality of MosaicDB, as well as the limitations and future directions of MosaicDB.

A. Extended evaluation queries

In this paper, we evaluated the execution time and memory usage of the MosaicDB by using only simple INSERT and SELECT queries. However, the basic evaluation of these queries alone is insufficient to demonstrate the practicality of MosaicDB in FTSNs. For example, in real FTSNs, queries that simultaneously process location data and other data (e.g., queries that JOIN location data and other data) must be executed with high throughput; however, the performance of the MosaicDB in executing such queries has not been measured. In addition, such queries tend to consume a large amount of temporary buffer space in the RDBMS, which affects not only the execution time but also memory usage in the enclave. Therefore, it is necessary to confirm whether MosaicDB can improve the memory utilization efficiency for such queries in the future.

B. Evaluation of actual memory load

In our evaluation, we estimated the memory usage by quantifying the amount of location data stored in the enclave and untrusted memory. However, for a more accurate evaluation of the memory utilization efficiency of MosaicDB, it is necessary to measure the actual memory load on both the enclave and untrusted memory. In the first generation of SGX, the physical memory usage of the enclave is determined during enclave initialization, making the physical memory usage unsuitable for evaluation. Therefore, we plan to measure the enclave memory load by monitoring the SGX paging.

C. More flexible memory management

Considering memory efficiency, it is ideal to distribute the utilization of both the enclave and untrusted memory evenly. However, in MosaicDB, all location data outside the EPZ are stored in the untrusted memory. Consequently, there is a risk of overloading the untrusted memory when there is an extreme shortage of location data within the EPZ or an abundance of enclaves. A more flexible approach to data management tailored to the load conditions of the enclave and untrusted memory is necessary to address these variations in application conditions and database server memory setups.

VII. Conclusions

In this paper, we proposed MosaicDB, a memory-efficient and trusted database that manages location data using both the enclave and untrusted memory in SGX. MosaicDB utilizes the characteristics of FTSNs and protects only the location data within the EPZ in the enclave so that both the enclave and untrusted memory can be effectively utilized. Performance evaluation showed that MosaicDB can efficiently utilize the entire memory of the server. The extension of evaluation queries, evaluation of the actual memory load, and more flexible memory management are future works.

Acknowledgments

This work was partly supported by the ICS-CoE Program at the Information technology Promotion Agency, Japan.

References

[1] W. U. Hassan, S. Hussain, and A. Bates, “Analysis of privacy protections in fitness tracking social networks -or- you can run, but can you hide?,” in *27th USENIX Security Symposium (USENIX Security 18)*, (Baltimore, MD), pp. 497–512, USENIX Association, Aug. 2018.

[2] K. Dhondt, V. Le Pochat, A. Voulimeneas, W. Joosen, and S. Volckaert, “A run a day won’t keep the hacker away: Inference attacks on endpoint privacy zones in fitness tracking social networks,” in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pp. 801–814, Association for Computing Machinery, 2022.

[3] Y. Wang *et al.*, “Cryptsqlite: Sqlite with high data security,” *IEEE Transactions on Computers*, vol. 69, no. 5, pp. 666–678, 2019.

[4] C. Priebe, K. Vaswani, and M. Costa, “Enclavedb: A secure database using sgx,” in *2018 IEEE Symposium on Security and Privacy*, pp. 264–278, 2018.

[5] A. Gribov, D. Vinayagamurthy, and S. Gorbunov, “Stealthdb: a scalable encrypted database with full sql query support,” in *Proceedings on Privacy Enhancing Technologies Symposium*, pp. 370–388, 2019.

[6] M. Yoshimura, T. Sasada, Y. Taenaka, and Y. Kadobayashi, “Memory efficient data-protection for database utilizing secure/unsecured area of intel sgx,” in *DBKDA 2023, The Fifteenth International Conference on Advances in Databases, Knowledge, and Data Applications*, pp. 38–43, 2023.

[7] A. Ittai, G. Shay, J. Simon, and S. Vincent, “Innovative technology for cpu based attestation and sealing,” in *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy (HASP ’13)*, 2013.

[8] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller, “Cache attacks on intel sgx,” in *Proceedings of the 10th European Workshop on Systems Security*, pp. 1–6, Association for Computing Machinery, 2017.

[9] P. Borrello, A. Kogler, M. Schwarzl, M. Lipp, D. Gruss, and M. Schwarz, “ÆPIC leak: Architecturally leaking uninitialized data from the microarchitecture,” in *31st USENIX Security Symposium (USENIX Security 22)*, pp. 3917–3934, USENIX Association, 2022.

[10] J. V. Bulck *et al.*, “Foreshadow: Extracting the keys to the intel SGX kingdom with transient Out-of-Order execution,” in *27th USENIX Security Symposium (USENIX Security 18)*, pp. 991–1008, USENIX Association, 2018.

[11] Y. Zheng, H. Fu, X. Xie, W.-Y. Ma, and Q. Li, *Geolife GPS trajectory dataset - User Guide*, geolife gps trajectories 1.1 ed., 7 2011.