Secure Data Processing at Scale

Kajetan Maliszewski
supervised by Prof. Volker Markl
Technische Universitat Berlin
maliszewski@tu-berlin.de

ABSTRACT

Although the cloud is today a de-facto standard for scalable data processing, there are still many applications that cannot make use of the cloud due to data or computation privacy. Sensitive data, such as in the health domain; and computations, such as core-business AI pipelines, grew into valuable assets that made secure data processing a hot topic in industry and academia. On one hand, the existing data processing systems prioritize performance and, to a certain level, trade users' privacy. On the other hand, privacy-preserving data processing systems sacrifice performance. In this PhD thesis, we envision a fully secure general-purpose data processing system for the cloud. Overall, we aim at devising: (i) algorithms that are adequate to work with very limited memory, such as the one exposed by trusted execution environments; (ii) scalable state management techniques; (iii) oblivious data-access algorithms; and (iv) privacy-preserving query optimizations techniques to speed up query execution.

1. INTRODUCTION

Processing data on the cloud has become omnipresent in our days [2]. For example, services, such as Amazon AWS and Microsoft Azure, have made trivial for companies, researchers, and organizations (users for short) to set up and maintain compute nodes. The cloud has given unprecedented power to users: they can now run applications and analytics before they were not able to run. For instance, a small company can easily offer scalable data analytics without owning its data infrastructure [1].

However, there are still many applications from different domains that cannot fully benefit from the cloud. Among these, we mainly find users working with *sensitive data*, e. g., on medical or transactional data, and users performing *sensitive computations*, e. g., machine/deep learning pipelines defining the core business of a company. These users typically have to classify their data or computations and hence are subject to strict compliance rules that force them to not trade privacy. As most cloud solutions do not treat privacy as a first-class citizen, users end up working on their premises sacrificing scalability and efficiency. This, for example, is the case of most applications in the healthcare domain, which use in-house solutions and infrastructure [8].

Even though, hybrid-cloud has recently appeared as a possible solution to this problem [17, 18], sensitive data and computations still cannot be moved from the private cloud. This is because existing data processing systems lack features for preserving the privacy of data and computations. Although few systems work on encrypted data [14, 15], most cloud-based data processing systems, such as Spark and Flink, expose data and computations at the hardware level. The research community proposed using *trusted execution environments* (TEEs) [12] to provide solutions to this problem [7, 9, 11, 16, 19]. Other works have also used oblivious algorithms to provide stronger security for TEEs by hiding data access patterns [7, 19]. Nevertheless, all these solutions suffer from several of the following problems: they (i) lack basic performance optimizations; (ii) do not scale out; (iii) cannot support stateful operators nor large state information; and (v) are ad-hoc to specific cases.

Therefore, despite all these efforts, we are still missing a holistic solution that could be a panacea to all the aforementioned concerns. The Holy Grail would be to replicate the success of general-purpose distributed data processing systems for secure, scalable cloud data processing. Users should focus on the logic of their applications while the system should take care of running and scaling out such applications efficiently and without any data/computation leakage. To the best of our knowledge, there is no general-purpose system that provides support for fully secure and scalable cloud data processing.

Building such a system is particularly challenging for many reasons. First, users must be able to easily define privacy constraints over their data and computations. Second, we have to revisit data processing algorithms and state management techniques to work within secure environments. Third, the system must ensure data and computation privacy on public compute nodes without sacrificing performance. Fourth, it is not clear how the system optimizes queries in the cloud when privacy is a first-class citizen.

In this thesis, we plan to tackle the above research challenges and lay down the foundations of the foreseen general-purpose system. We plan to proceed as follows: we will first build a single node secure and efficient data processing engine; we will follow with making our data processing engine distributed and scalable, by considering public and trusted

Proceedings of the VLDB 2020 PhD Workshop, August 31st, 2020. Tokyo, Japan.

nodes; we will focus on devising a privacy-preserving query optimizer. In summary, we plan to make the following major contributions:

1. We will devise an efficient general-purpose data processing engine for TEEs. In particular, we will propose new data processing algorithms that are adequate to work with very limited memory (provided by TEEs environments).
2. We will extend our data processing engine to support stateful operators. We will particularly provide both oblivious data-access algorithms and support for large state information in TEEs environments.
3. We will propose bidirectional data anonymization algorithms as well as data processing algorithms being able to work over encrypted data.
4. We will then devise different privacy-preserving query optimizations techniques to speed up query execution.

In the remainder of this paper, we first define the problem we plan to tackle in this thesis in Section 2. We present related work in Section 3. In Section 4, we depict our envisioned solution. We follow, in Section 5, with different open challenges we must tackle to make our envisioned solution a reality. Lastly, we conclude this paper in Section 6.

2. PROBLEM STATEMENT

Efficient and fully secure data processing on the cloud is currently not possible at the terabyte scale. Guaranteeing robustness and consistent privacy level requires (i) usage of novel hardware technologies for low-level security, (ii) redesigning existing approaches for new environments, and (iii) efficient secure query processing engine. Data and computations can only be fully protected using technologies such as TEEs, combined with oblivious data access. Most of the existing execution approaches cannot be easily mapped to the new runtimes due to heavy limitations that TEEs enforce on the users. Data processing algorithms have to be redesigned having these limitations in mind. Most importantly, compute resources have to be wisely fully utilized to guarantee high query execution efficiency. Moreover, worker nodes need to be classified by the level of security they require; *private compute nodes* are considered secure, *trusted/secure compute nodes* can prove their security but require data anonymization for privacy, and *public compute nodes* need oblivious processing in a TEE.

Therefore, the problem, and main challenge, resides in how to *enable efficient and truly secure data processing jobs to hybrid compute environments.*

3. RELATED WORK

Efficient Execution & State Management. Sanctuary [16] is a distributed streaming system. The authors present a set of algorithms to manage large state, but they blindly spill to disk the state information. Additionally, the state is not managed obliviously, hence, the longer the job runs, the more information leaks out. SecureStreams [9] is a lightweight streaming platform running in SGX enclaves. The jobs are build using a set of simple operators but the system lacks optimizations (unnecessary encryption/decryption between each operator) and does not scale-out well (inter-operator communication quickly becomes a traffic bottleneck). It also does not protect against access-pattern attacks. TrustedDB [5] is a secure database built on a cryptographic coprocessor, an older trusted hardware architecture. It stores large state externally and accesses it using a Paging Module, which exposes the access patterns. EnclaveDB [11] is a database utilizing SGX that only handles state up to the size of the enclave memory (approximately 90 MB). The authors assume that the future releases of SGX would support much larger memory, however, up until the current release it has not happened.

The problem of tiny memory has been previously addressed in small footprint databases [6, 10]. They propose lightweight solutions, however, drastically limiting functionality for their very specific use cases, i. e., handheld computers, and smartcards.

Data Access Privacy. ObliDB [7] is an oblivious database core engine for general workloads. It hides access patterns using oblivious query processing algorithms that require a full table scan for each query. However, it runs only on a single node in an SGX environment. Opaque [19] executes encrypted Spark jobs using oblivious access. It proposes a query optimizer to mitigate the cost of obliviousness, however, it still reaches performance degradation of up to 46x.

Query Optimization. The existing systems utilizing hybrid-cloud [17, 18] perform the optimizations based on manual tagging the data by the users with its sensitivity. Later, only the insensitive data is processed in the public cloud. This is cumbersome for the users and harmful for workloads consisting mainly of sensitive data. In contrast to these works on hybrid-cloud, we aim at sending sensitive data to the public cloud and omit the tagging process by leveraging secure hardware.

4. OUR VISION

To overcome the problem stated in Section 2, we envision a system comprising a *master node* and three types of *compute nodes*: (i) *private compute nodes* (a fully trusted worker), (ii) *trusted/secure compute nodes* (guaranteed with certificates of trust), (iii) *public compute nodes* (a fully untrusted machine). We believe that these abstractions ideally fulfill the performance needs while maintaining strong privacy.

A user submits a query to the master node. In turn, the master node parses the query and optimizes it by exploiting the knowledge about the topology and available resources. It then compiles, executes, and monitors the query. Note that it is the compute nodes that carry out the actual execution.

Figure 1 depicts the execution of a query over the three types of machines, namely private, trusted/secure, and public compute nodes. Each node has a rigorous way of processing the query enforced by the query execution plan. Private compute nodes are allowed to process the input data in clear, i. e., neither encrypted nor anonymized (*green arrow*). Trusted/secure compute nodes are considered trustworthy. Hence, they can process data as a private node but might also be forced to process anonymized data depending on its trustworthy degree (*orange arrow*). Public compute nodes are simply considered insecure. They thus receive enclave-encrypted data and process it inside a TEE (*red arrow*). Executing a query in such environments is far from being trivial as data might be transferred from one kind of compute nodes to another. For example, Figure 1 illustrates

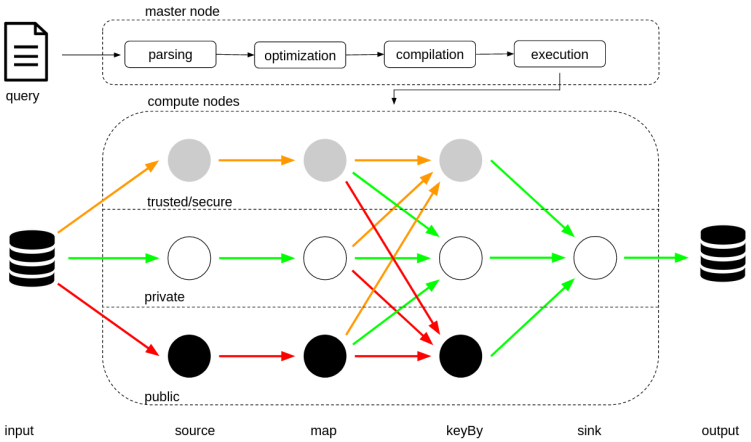

Figure 1: Overall architecture of our envisioned system: green arrows represent traffic in plain text, orange arrows represent anonymized data, and red arrows represent data encrypted with the enclave key.

such data transfers with different arrow colors: while passing from orange to green means data de-anonymization, red to green stands for decryption with the enclave key. At the end of the query, the data is aggregated and sent to output as defined by the query.

5. RESEARCH CHALLENGES

Building a system as described in Section 4 comes with several research challenges, mainly around *data access privacy*, *efficient query execution*, *state management*, and *query optimization*. We elaborate on each of these in the following.

5.1 Efficient query execution

Efficiently executing a query in TEEs is quite challenging because of the extremely small main memory capacity in TEEs, expensive CPU instruction set, and no system calls. For example, Intel's proprietary TEE technology (SGX) defines an enclave, a private and highly protected region in memory that cannot be accessed from outside of its process. It places the enclave code and data in a special memory area of 128 MB. Yet, excluding space for the SGX metadata, there is approximately only 90 MB left for the application.

As a result, the design of state-of-the-art data processing operations (e. g., a join operator) cannot simply be mapped to enclave-enabled versions. For example, consider the case of a hash-join operator. In the build phase, this operator takes the smaller table and builds the hash table for the selected key. Once the table reaches 90 MB in size, it will start spilling the records to the memory outside the enclave in an encrypted form. These data spilling operations cause expensive calls to the CPU due to the context-switching instructions and costly encryption.

Therefore, efficiently executing queries in TEEs requires a radical change in the design of data processing operators. We will investigate new techniques for cache management inside the enclaves and optimizations on CPU instruction set specifically for relational algebra. To speed up query execution even further, we will investigate the use of query compilation techniques to generate highly optimized code for TEEs. Additionally, we will examine parallel enclaves execution on multi-core CPUs. We will design data processing operators relying on these new techniques.

5.2 State Management

The state is an essential element of an operator. During execution, it is used for storing metadata and intermediate results, e. g., a rolling aggregation while scanning a table. Handling large state management efficiently for enclave-enabled operators is challenging because of the extremely small main memory capacity in TEEs (see Section 5.1).

Existing systems store large states in the memory outside of the enclave[5, 16]. However, constant paging, and thus, data encryption and decryption, imposes great performance deterioration. For example, TrustedDB [5] uses a custom-built Paging Module that stores all pages outside of the Secure Coprocessor. The pages are pulled on-demand as needed by the query processing engine. Thoma et al. in [16] propose stateful operators for stream processing using SGX's built-in paging mechanism. Yet, both systems are not sufficiently optimized for secure hardware. For example, in [16], a hash-join operator blindly spills the hash table to the outside memory, leading to expensive calls outside the enclave whenever it looks for a tuple.

Thus, new state management techniques are required for settings where the main memory is drastically limited. We will investigate new data structures and data indexes that allow for state management outside the enclave while at the same time reducing the unnecessary outside calls.

5.3 Data Access Privacy

Although TEEs (such as SGX) provide secure data processing via hardware, it is still possible to have data leakage by understanding the data access patterns done by the operators inside the enclave. There have thus been many proposals on how to achieve data access privacy in systems designed for specific use-cases, e. g., homomorphic encryption [14] and oblivious algorithms [7, 19].

However, it is a perpetuating problem how to glue these proposals together to provide a general-purpose data processing engine with oblivious data access and without hurting performance. For instance, Opaque [19] provides oblivious data processing on SGX, but comes with an overhead of up to 46x! Similarly, ObliDB [7] comes with a huge overhead as it performs a full table scan for each query to achieve obliviousness.

We will explore different ways of reducing such overhead incurred by current oblivious data access algorithms. Particularly, we plan to investigate how to set a lower-bound on the number of non-relevant tuples that are necessary to hide access patterns. Such a lower-bound guarantee is required to not compromise privacy while not harming performance. Another research direction we will explore is to store inside the enclave metadata about table values (similar to Oracle's Zone Maps [3]). Having such knowledge ahead of a query can even prevent scanning a table at all.

5.4 Privacy-Preserving Query Optimization

Intel SGX is notorious for its performance degradation and vulnerability to some attacks [4]. End-to-end encryption and data de/anonymization are known to be computationally intensive. Inter-cloud data transfers have been identified as a bottleneck in cloud computing [18]. These are some of the aspects the query optimizer will consider when deciding which data is to be executed on which nodes.

Existing systems reduce the problem space by reducing their functionality [7, 13]. However, while VC3 [13] is not using oblivious access and thus it is vulnerable to data access pattern attacks, ObliDB [7] runs only in an SGX-enabled environment. Essentially, all these systems are designed for homogeneous environments, i. e., they assume all compute nodes in the system provide a TEE environment. This is not the case in our envisioned system where multifarious machines with different requirements co-habit together in the same system. Enforcing a unified policy across all of them would end up in performance degradations.

To solve this challenge, we will study the performance of connecting operators with different privacy constraints. This will add a new dimension to the query optimization and will force us to rethink the optimization techniques for logical and physical query plans. Moreover, we will investigate new techniques for data anonymization and de-anonymization, data processing over encrypted data, and query optimization for heterogeneous secure environments.

6. CONCLUSION

We presented our plan towards a general-purpose secure and scalable data processing system for hybrid environments (i. e., composed of private, trusted, and public compute nodes). We showed that each of the existing systems solves only a piece of the problem. State-of-the-art solutions force users to work with sensitive data or computations only within a private environment, hence underutilizing the available computing resources. Moreover, some works trade users' privacy or drastically limit the use-case, e. g., to a centralized system not suitable for very large datasets. We showed that we are still missing a system that sees the bigger picture and solves the problem of truly secure data processing at scale. We presented our envisioned system to solve such a problem and discussed the main research challenges that we must tackle to make our vision a reality.

7. ACKNOWLEDGMENTS

This work was funded by the German Ministry for Education and Research as BIFOLD - Berlin Institute for the Foundations of Learning and Data (ref. 01IS18025A and ref. 01IS18037A).

8. REFERENCES

[1] AWS Startup Stories. https://aws.amazon.com /campaigns/aws-startups-stories/.

[2] Microsoft's Growth ReAzuring Under Nadella. https://markets.businessinsider.com /news/stocks/microsoft-s-growth-reazuring-under-nadella-1028372914.

[3] Oracle Database Concepts. https://docs.oracle.com.

[4] P. Antonopoulos, A. Arasu, K. Eguro, J. Hammer, R. Kaushik, D. Kossmann, R. Ramamurthy, and J. Szymaszek. Pushing the limits of encrypted databases with secure hardware. *arXiv preprint arXiv:1809.02631*, 2018.

[5] S. Bajaj and R. Sion. TrustedDB: A Trusted Hardware Based Database with Privacy and Data Confidentiality. In *SIGMOD*, 2011.

[6] C. Bobineau, L. Bouganim, P. Pucheral, and P. Valduriez. PicoDBMS: Scaling down database techniques for the smartcard. In *VLDB*, 2000.

[7] S. Eskandarian and M. Zaharia. ObliDB: oblivious query processing for secure databases. *PVLDB*, 2019.

[8] L. Griebel, H.-U. Prokosch, F. Köpcke, D. Toddenroth, J. Christoph, I. Leb, I. Engel, and M. Sedlmayr. A scoping review of cloud computing in healthcare. *BMC Med. Inf. & Decision Making*, 2015.

[9] A. Havet, R. Pires, P. Felber, M. Pasin, R. Rouvoy, and V. Schiavoni. Securestreams: A reactive middleware framework for secure data stream processing. In *DEBS*, 2017.

[10] J. S. Karlsson, A. Lal, C. Leung, and T. Pham. IBM DB2 everyplace: A small footprint relational database system. In *ICDE*, 2001.

[11] C. Priebe, K. Vaswani, and M. Costa. Enclavedb: A secure database using SGX. In *S&P*, 2018.

[12] M. Sabt, M. Achemlal, and A. Bouabdallah. Trusted Execution Environment: What It is, and What It is Not. In *Trustcom/BigDataSE/ISPA*, 2015.

[13] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich. VC3: Trustworthy data analytics in the cloud using SGX. In *S&P*, 2015.

[14] J. J. Stephen, S. Savvides, V. Sundaram, M. S. Ardekani, and P. Eugster. STYX: stream processing with trustworthy cloud-based execution. In *SoCC*, 2016.

[15] S. D. Tetali, M. Lesani, R. Majumdar, and T. Millstein. MrCrypt: Static analysis for secure cloud computations. In *OOPSLA*, 2013.

[16] C. Thoma, A. J. Lee, and A. Labrinidis. Behind enemy lines: Exploring trusted data stream processing on untrusted systems. In *CODASPY*, 2019.

[17] X. Xu and X. Zhao. A framework for privacy-aware computing on hybrid clouds with mixed-sensitivity data. In *HPCC/CSS/ICESS*, 2015.

[18] K. Zhang, X. Zhou, Y. Chen, X. Wang, and Y. Ruan. Sedic: privacy-aware data intensive computing on hybrid clouds. In *CCS*, 2011.

[19] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica. Opaque: An oblivious and encrypted distributed analytics platform. In *NSDI*, 2017.