International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2025.082
e ISSN: 2584-2854
Volume: 03
Issue:03 March 2025
Page No: 520 - 525

Efficiency Enhancement Architecture of SQL Queries Through Hierarchical Cache

Achal Vijay Navaloor[1], Kumudavalli M V[2], Aliya Hassan[3]
[1,2,3] PG- Department of Computer Applications, Dayananda Sagar College of Arts Science & Commerce, Bangalore, Karnataka, India.
Email ID: *achalnavaloor12@gmail.com[1], kumudamanju@gmail.com[2], aliyahassan0301@gmail.com[3]*

Abstract
Secure Multi-Party Computation (MPC) enables privacy-preserving collaborative data analysis but suffers from high computational overhead, communication latency, and inefficient query execution. Traditional caching mechanisms, such as SMPCache, attempt to optimize performance but face limitations due to high storage overhead, low cache hit rates, and inefficient cache lookup strategies. This study suggests a Hierarchical SMPCache (H-SMPCache) The approach to address these issues, inspired by CPU caching, introduces a multi-level caching structure (L1, L2, L3) to efficiently manage SQL queries in MPC environments. The proposed hierarchical caching model aims to classify queries based on execution cost, frequency, and computational complexity.

Keywords: *Secure Multi-Party Computation (MPC), Hierarchical Caching, SMPCache, SQL Query Optimization, L1-L2-L3 Cache, Privacy-Preserving Data Analysis.*

1. Introduction
Data protection is more crucial than ever during data analysis using today's data control apps. A key paradigm that enables many parties to compute functions on entries while maintaining the privacy of these inputs is Multi-party Calculation (MPC). Situations involving sensitive data, like healthcare, finance, and personal information management, benefit greatly from this. Nonetheless, MPCs have significant obstacles, chiefly related to compensating for computation, communication lags, and ineffective query processing. The computational complexity needed to run SQL queries in an MPC context can exacerbate these issues. This makes real-time performance challenging. SMPCache and other classic caching methods have been developed to increase the speed of these scenarios. The efficiency of the cache is significantly influenced by the ability to predict which queries occur most frequently. Poor cache strategies can result in decreased overall performance and inefficient resource utilization.

1.1. The Difficulties of Secure Multi-Party Computation
MPC protocols frequently have significant latency and poor performance because they demand a lot of processing power and communication capacity. The intricacy of the calculations required to run SQL queries in an MPC environment may make these problems worse and make achieving real-time performance challenging. SMPcache and other traditional caching methods have been developed to increase speed under these circumstances.9. Regretfully, people frequently encounter obstacles that keep them from progressing: Current caching solutions' large storage space requirements could be a major disadvantage in situations when storage is expensive or scarce. The efficacy of the cache is greatly impacted by predicting which queries will be executed most frequently. Poor cache hit rates may result in inefficient resource usage and a decline in overall performance.Traditional caching algorithms sometimes handle the search of cached queries inefficiently, increasing the latency while obtaining cached data.[1-2]

1.2. The Necessity of Hierarchical Caching
To address these issues, we propose a new method known as hierarchical smpcache(hsmpcache). The multilevel cache architecture of computer processors that improves processing efficiency and increases

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2025.082
e ISSN: 2584-2854
Volume: 03
Issue:03 March 2025
Page No: 520 - 525

access to data serves as a model for this approach. Level 1 (L1), Level 2 (L2), and Level 3 (L3) - Cutches are all included in the hierarchical cache model and we recommend that you improve the handling of SQL queries in your MPC settings.The fastest and smallest cache is level 1 (L1), whichis designed for the most frequently visited data and queries. Provides lightning access and significantly reduces immediate instructions and data latency.It is found on all CPU cores. Located inside or next to the CPU core, it acts as a connection between the ultraast L1 cache and the slow L3 cache thatdoes not provide the data that is needed.It's inside the chip, but outside the main storage, which increases system speed by reducing the need to access main memory.

1.3. Objectives of the Hierarchical SMPCache

To address these issues, we propose a new methodknown as hierarchical smpcache (h-smpcache). The multilevel cache architecture of computer processors that improves processing efficiency andincreases access to data serves as a model for this approach. Level 1 (L1), Level 2 (L2), and Level 3 (L3) - Cutches are all included in the hierarchical cache model and we recommend that you improve the handling of SQL queries in your MPC settings. The fastest and smallest cache is level 1 (L1), which is designed for the most frequently visited data and queries. Provides lightning access and significantly reduces immediate instructions and data latency. It is found on all CPU cores. Located inside or next to the CPU core, it acts as a connection between the ultrafast L1 cache and the slow L3 cache that does not provide the data that is needed. It's inside the chipbut outside the main storage, which increases system speed by reducing the need to access the main memory

1.4. Cache Management Tools

The Redis Using a computer's main memory (RAM), Redis is an open-source data structure memory that stores key-value pairs. Its maternal nature allows for quick access to regularly used data. Among the many data structures that Redis offers are strings, lists, sets, hashes, sort sets, and more. It is renowned for supporting high performance, low latency, and sophisticated features including PUB/submessage, replication, and persistence. Redis is frequently utilized as a database, message broker, or cache. Gives rapid access to commonly used data and keeps significant pairs in storage. By lowering database load, Memcached is frequently utilized to enhance web application performance. Because of its flexibility for horizontal scaling, numerous servers can collaborate to handle massive volumes of data. Memcached is renowned for its low memory overhead and ease of usage. Because it is based on LevelELDB, it is designed to read quickly. Features like column families, transactions, and snapshots are offered by ROCKSDB. Because of this, it can be used in a range of storage situations. Typically, it is employed for data caching. This is due to the fact that it provides an affordable balance between durability and quickness. To guarantee that periods are written to the hard drive, ROCKSDB writes data. Because of this, it's a dependable choice for programs that need permanent memory. In order to avoid double calculations and save huge and costly inquiries, it offers a hard drive. Caching, high-performance computation, and real-time analytics are just a few of the many applications that Apache Ignite offers. Among its characteristics are machine learning, distributed data structures, SQL support, and acid transactions. Even after a system restarts, data is accessible thanks to Apache Ignite hard drives. Because of this, it is a reliable option for applications that need fault tolerance and high availability. While RocksDB and Apache Ignite offer persistent store options for larger data records and costly computations, Redis and Memcached enable quick memory access to frequently used data. A balanced approach to speed, durability, and scalability is ensured by this combination.[3-4]

2. Literature Review

J. Shi et al. focus on improving the efficiency of SQL queries in multi-party collaborative data analysis using a cache-like optimization mechanism. The authors address the inefficiencies of existing secure multi-party computation (MPC) solutions. P. Alexander discusses the Rosetta system-level specification language, which is designed for complex, heterogeneous systems. The paper emphasizes the importance of standardization in system-level design and provides insights into the

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2025.082
e ISSN: 2584-2854
Volume: 03
Issue:03 March 2025
Page No: 520 - 525

development and application of the Rosetta language. R. Saha et al. explore the use of randomness to enhance security and privacy in multi-party computation (MPC). The authors propose a novel approach to address the limitations of existing MPC models by incorporating a random function generator. Y. -Z. Hu and H. Wang present an innovative approach to optimizing the performance of RocksDB, a key-value database, through an auto-tuning system based on Transformer models. The authors address the challenges of manually configuring RocksDB knobs and propose a solution to enhance performance across various workloads.

M. Sepehri, S. Cimato, and E. Damiani investigate techniques for executing privacy-preserving queries on partitioned databases using secure multi-party computation (SMC). The authors propose a novel approach to address the challenges of privacy and efficiency in query processing. C.-S. Stan et al. (2019) provides a comparative analysis of two popular data processing frameworks: Apache Spark and Apache Ignite. The authors evaluate the performance of these frameworks based on various metrics, including features, implementation, architecture, and performance. C. Yearn et al. (2024) explores the use of meta-learning techniques to optimize the configuration of RocksDB, a persistent key-value store, for various workloads. The authors propose a novel approach called MetaTune, which aims to address the challenges of tuning RocksDB configurations for different types of workloads.

Z. Ji et al. (2014) presents a framework designed to enhance the performance of data access in the Ubiquitous Consumer Wireless World (UCWW) by leveraging Redis, a NoSQL database. The authors propose a distributed Redis framework to address the limitations of a single Redis node and improve the system's performance in handling large volumes of requests from web applications. W. Wei, K. Namba, and F. Lombardi (2016) presents a detailed analysis of a hybrid cache memory design. The authors focus on evaluating the performance and architectural aspects of hybrid cache memory, which combines different types of memory technologies to achieve optimal performance. K. Kawabata and N. Hayashibara was presented at the 2024 IEEE 29th Pacific Rim International Symposium on Dependable Computing (PRDC) in Osaka, Japan. The paper discusses the implementation of Oblivious Random Access Memory (ORAM) using caching and prefetching techniques to improve performance.[5-8]

3. Methodology

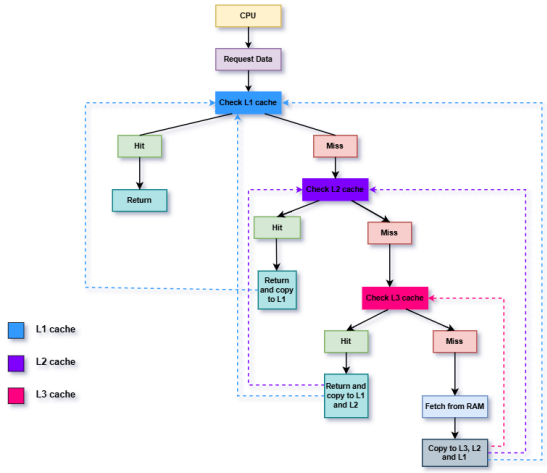

Figure 1 CPU and Information

CPU and Information Needs the main part of a computer system that is in charge of carrying out commands and computations is the Central Processing Unit (CPU). Data will be queried if the CPU needs it to carry out the instruction. Prior to arriving at main memory (RAM), this request travels in a hierarchical fashion through several cache storage tiers. In order to maintain effective CPU performance, the objective is to get your data as fast as feasible. There are two distinct caches within this cache: Data Cache (L1D): Holds the most often used information. The CPU can access the most commonly used data and instructions because of the close proximity to the CPU core, which guarantees reduced latency when accessing data. Cache, L2 cache, and the next cache level, receives the requirements. The L2 cache may be allocated to a single core or shared over many seeds, depending on the CPU architecture. Size and Speed: Compared to the L1 cache, the L2 cache is larger but operates more slowly. For frequently requested information and commands that don't fit into the L1 cache, it serves as secondary memory. in the L3 cache. It is typically

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2025.082
e ISSN: 2584-2854
Volume: 03
Issue:03 March 2025
Page No: 520 - 525

shared by the chip under all CPU cores and offers shared storage for any data that the kernel might want. Size and Speed: In the hierarchy, the L3 cache is the biggest and slowest. Although it keeps access to the data from the intermediate data acceptance phase, it archives the data and instructions that are utilized frequently. Hit: The requested data will return to the CPU and be used in a subsequent inquiry if it is located in the L3 cache. Information. from RAM, or main memory. This is where the data is kept if none of the cache layers are accessible. RAM is far slower than cache memory, but it is also much larger. Because RAM is physically farther from the CPU and is of the dynamic random access memory (DRAM) type, accessing data from it has a larger latency than cache memory. To speed up access in the future, the data is then copied into the L3, L2, and L1 caches. The CPU will be able to swiftly access data the next time it's needed. Fastest offers the quickest access to the information and commands that are utilized the most. The amount of time the CPU takes to output data is greatly decreased as a result. It saves the data and instructions kept in the middle and acts as a common resource for all CPU cores. Make sure you always have access to the data you require, even if the CPU demands a high delay, in case data is unavailable at any of the cache levels. Secondary memory for information and commands that are regularly accessed. Several cache storage levels (L1, L2, and L3) are used by the CPU to guarantee effective data calls prior to main memory (RAM). This multi-phase method ranks data access and storage according to CPU core proximity and usage frequency. This enhances CPU performance overall and lowers latency. You can comprehend complex designs that allow modern CPUs to manage data requirements efficiently by knowing the function and interactions of each cache level and RAM. [9]

4. Results and Discussion

Theoretical Performance Improvements Query Runtime: The main goal of H-smpcache is to use cache structures at several levels to reduce query execution time. In this structure, in many cases, accessed and calculated inexpensive queries are stored in the fastest cache (L1). By using L1 cache speed, these queries can be processed with minimal latency. Categorization of queries based on execution cost and frequency allows the system to prioritize faster cache levels forQuilly resolution. This theoretical approach ensures that the most common and simple queries are executed immediately, improving overall system performance. In the HSMPCache model, queries are categorized based on execution cost and fraud. This classification allows the system to store the queries that are most frequently accessed in the L1 cache, the fastest cache level. As a result, it is expected to have the highest hit rate for the L1 cache. Less frequently asked queries are stored in the slower but larger L2 and L3 caches. The approach seeks to increase the overall cache hit rate by strategically allocating queries among multiple cache layers. Larger, slower caches (L2 and L3) hold less important and infrequently used queries. By making sure that only necessary queries are stored in a small amount of space in the L1 cache, this method reduces storage efforts. Consequently, the system may optimize caching tools, make effective use of available storage resources, and guarantee that queries with higher priority are resolved promptly. Because only critical queries are retained at each cache level, this distribution lessens query storage redundancy. The model maximizes storage space by removing duplicate storage for queries, ensuring that each cache level has pertinent and understandable data. This improves query resolution efficiency and lowers the system's overall storage load. There is no thorough communication between the parties that can use a large range if the query is answered in the cache. The system can manage the majority of its query needs locally by keeping frequently accessed queries in the L1 cache. This boosts bandwidth efficiency and decreases confidence in the communication channel. Data transfer between the parties is not necessary if the query is resolved in the cache. Delays in communication are reduced as a result. This method lowers the possibility of data leaks during transmission while simultaneously speeding up query resolution. The concept guarantees the security of sensitive data and reduces communication efforts by limiting data transfer. Fast calculations are handled by the L1 cache, whereas more complex calculations

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2025.082
e ISSN: 2584-2854
Volume: 03
Issue:03 March 2025
Page No: 520 - 525

are handled by the L2 and L3 caches. The system can accommodate workloads at several cache levels and avoid a single cache level becoming a bottleneck thanks to this load distribution. The approach guarantees optimal use of resources and boosts overall system efficiency by allocating arithmetic jobs according to their complexity. Scalability: Scalability has a significant impact on a cache system's performance. Without causing appreciable performance deterioration, the H-SMPCache model is made to scale and handle fault load increases well. The system can consider more requests by distributing them across multiple cache levels by employing a hierarchical structure. This guarantees that even when the query load rises, the system will continue to respond swiftly and effectively. H-SMPCache is appropriate for a range of computer settings and workloads due to its scalable nature. MPC enables several parties to work together through entries and manage these entries directly. H-SMPCache guarantees the security of sensitive data while query processing by including MPC. Additionally, the use of Anhentious RAM (ORAM) continues to be protected from access patterns and leaks. Oram continuously mixes and reduces data and makes it difficult to access to it to make itaccessible. This combination of MPC and ORAM increases the general privacy and security of your cache system. Each of these frameworks provides unique features and capabilities for secure calculations.Rosetta: A free and open-source software framework called Rosetta enables the private and safe execution of SQL queries in a multi-party computing environment. This focuses on a multi-party SQL version that is secure and protects privacy when database queries are being executed. Caching systems are a flexible solution for safe data processing because of their flexibility to adapt to the data security and protection needs of different applications. With the use of a secure multi-party computing framework and a multi-level cache structure, H-SMPCache seeks to offer a reliable and effective query-processing solution in a secure computing environment. It is a promising method for enhancing distributed data processing systems' security and performance because of its scalable nature and adaptation to different MPC frameworks.

Conclusion

H-SMPCACHE (Hierarchical SMPCACHE): A Novel Approach and a different solution to the problems of a secure MPC environment (multi-party computing) is the hierarchical, secure, secure, multi-party computation cache (H-SMPCache). H-SMPCache seeks to minimize communication efforts, optimize memory utilization, and shorten query execution times by drawing inspiration from the CPU's multi-level cache design. With strong data protection and security standards, this approach enhances efficiency and performance. This guarantees that joint requests are processed rapidly, lowers latency, and improves system efficiency overall. This is because, with fewer requests, the L1 cache has the highest hit rate, followed by the L2 and L3 caches. Performance is enhanced and access time is decreased with this distribution. While crucial queries are not maintained in the L2 and L3 caches, important requests are stored in the L1 cache. This guarantees that memory usage is kept to a minimum and that queries are promptly and highly prioritized answered. Every cache level optimizes query resolution efficiency by containing distinct and pertinent material. This lessens reliance on communication lines and improves bandwidth efficiency. No data transmission is necessary if the query is resolved in the cache. This lowers the chance of data loss and connection lag. Fast calculations are handled by the L1 cache, whereas more complex calculations are handled by the L2 and L3 caches. This equilibrium boosts system efficiency and avoids bottlenecks. By allocating many cache layers and guaranteeing system response, the hierarchical structure incorporates extra queries. MPC guarantees the security of sensitive data while it is being processed by queries. Purster terminated the global Privation and Securition, and Vollische Ram (Oram) advanced even further with Si-kyung. This flexibility guarantees that your cache system satisfies security and data protection standards for a range of applications. H-SMPCache offers a reliable and effective solution for query processing in a secure computer environment by integrating multi-stage cache structure and MPC frameworks. It is a potential method for enhancing the security and performance

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2025.082
e ISSN: 2584-2854
Volume: 03
Issue:03 March 2025
Page No: 520 - 525

of distributed data processing systems because of its scalability and compatibility with several frameworks. This concept is an important tool for contemporary, secure computer applications since it uses a hierarchical caching structure to increase performance and efficiency while protecting privacy and secure data.

References

[1]. J. Shi et al., "SMPCache: Towards More Efficient SQL Queries in Multi-Party Collaborative Data Analysis," in IEEE Transactions on Knowledge and Data Engineering, doi: 10.1109/TKDE.2025.3535944.

[2]. P. Alexander, "Rosetta: Standardization at the System Level," in Computer, vol. 42, no. 1, pp. 108-110, Jan. 2009, doi: 10.1109/MC.2009.23

[3]. R. Saha, G. Kumar, G. Geetha, M. Conti, and W. J. Buchanan, "Application of Randomness for Security and Privacy in Multi-Party Computation," in IEEE Transactions on Dependable and Secure Computing, vol. 21, no. 6, pp. 5694-5705, Nov.-Dec. 2024, doi: 10.1109/TDSC.2024.3381959.

[4]. Y. -Z. Hu and H. Wang, "TATune: A RocksDB Knob Tuning System Based on Transformer," in IEEE Access, vol. 11, pp. 143589-143600, 2023, doi: 10.1109/ACCESS.2023.3343455.

[5]. M. Sepehri, S. Cimato, and E. Damiani, "Privacy-Preserving Query Processing by Multi-Party Computation," in The Computer Journal, vol. 58, no. 10, pp. 2195-2212, Oct. 2015, doi: 10.1093/comjnl/bxu093.

[6]. C. -S. Stan, A. -E. Pandelica, V. -A. Zamfir, R. -G. Stan and C. Negru, "Apache Spark and Apache Ignite Performance Analysis," 2019 22nd International Conference on Control Systems and Computer Science (CSCS), Bucharest, Romania, 2019, pp. 726-733, doi: 10.1109/CSCS.2019.00129.

[7]. C. Yearn, J. Lee, S. Seo and S. Park, "Towards Workload-Specific Configuration Tuning via Meta-Learning for RocksDB," 2024 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Kuching, Malaysia, 2024, pp. 4450-4457, doi: 10.1109/SMC54092.2024.10831422.

[8]. Z. Ji, I. Ganchev, M. O'Droma and T. Ding, "A Distributed Redis Framework for Use in the UCWW," 2014 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, Shanghai, China, 2014, pp. 241-244, doi: 10.1109/CyberC.2014.50.

[9]. W. Wei, K. Namba and F. Lombardi, "Design and comparative evaluation of a hybrid Cache memory at architectural level," 2016 International Great Lakes Symposium on VLSI (GLSVLSI), Boston, MA, USA, 2016, pp. 125-128, doi: 10.1145/2902961.2903002.