Journals Pub

International Journal of
Distributed Computing and Technology

ISSN: 2455-7307
Volume 9, Issue 2, 2023
DOI (Journal): 10.37628/IJDCT

http://computers.journalspub.info/index.php?journal=JDCT&page=index

Review IJDCT

A Comprehensive Survey on Optimizing Storage Models, Data Layouts, and System Catalogs

Md. Shifatul Ahsan Apurba*

Abstract

This survey compares four papers that propose solutions for optimizing the storage model, data layout, and system catalogs in data management systems for hybrid workloads, which consist of both analytical and transactional queries. These solutions include a main memory hybrid storage engine that separates the storage of analytical and transactional data and uses a sophisticated query optimizer, a hands-free adaptive store that adjusts the storage layout based on access patterns, a hybrid storage engine that combines the strengths of row- and column- store systems and a column layout optimization method that considers both analytical and transactional access patterns and uses ghost values to support updates. These papers highlight the importance of designing storage systems specifically for hybrid workloads and the need for further research in this area.

Keywords: Adaptive storage, dynamic operators, adaptive hybrids, main memory, workloads, hybrid storage, workloads

INTRODUCTION

In recent years, there has been a growing interest in the design of storage engines that can effectively support hybrid workloads, which are characterized by a mix of long-running analytical queries and short-lived transactions. These hybrid workloads are increasingly common in modern systems and pose unique challenges for storage engine design due to the need to balance the conflicting requirements of high-throughput analytical processing and low-latency transactional processing.

To address these challenges, several storage engines have been proposed that employ various techniques to optimize the performance of hybrid workloads. These techniques include the use of column-oriented layouts, data partitioning, compression, and multi-version concurrency control, among others. In this survey, we provide an overview of four such storage engines: HYRISE, H_2O, Bridging the Archipelago, and Optimal Column Layout. Each of these storage engines has its own unique approach to supporting hybrid workloads, and we will describe the key features and evaluation results of each in detail.

*Author for Correspondence
Md. Shifatul Ahsan Apurba
E-mail: md.shifatul.ahsan.apurba@g.bracu.ac.bd

Associate Software Engineer, Department of Computer Science and Engineering, BRAC University, Dhaka, Bangladesh

Received Date: July 20, 2023
Accepted Date: July 28, 2023
Published Date: August 17, 2023

Citation: Md. Shifatul Ahsan Apurba. A Comprehensive Survey on Optimizing Storage Models, Data Layouts, and System Catalogs. International Journal of Distributed Computing and Technology. 2023; 9(2): 1–8p.

Moreover, our survey aims to provide a comprehensive understanding of the state-of-the-art in storage engine design for hybrid workloads and to identify the key design trade-offs and challenges that need to be addressed in this area. By highlighting the strengths and weaknesses of different approaches, we hope to provide valuable

© JournalsPub 2023. All Rights Reserved

insights for future research on hybrid workloads and to guide the development of more effective storage engines for these demanding applications.

LITERATURE REVIEW

There has been a growing interest in optimizing the physical design of databases for hybrid workloads, which consist of both transactional and analytical queries. These queries can have significantly different access patterns and performance requirements, making it challenging to design a physical layout that can efficiently support both types of workloads.

In their paper, Grund *et al.* introduce HYRISE, a hybrid database system for main memory with an emphasis on optimizing cache performance for queries [1]. Based on the access trends of queries across tables, HYRISE builds vertical divisions of tables. The authors built a precise model of cache misses that can predict the number of misses that will occur for a given partitioning and access pattern, allowing them to calculate the optimal partitioning. Based on this approach, they introduced a database design technique that can scale to tables with many columns and identifies partitioning that result in the fewest possible unobserved variables for a given workload. On a practical benchmark drawn from a popular corporate application, the authors demonstrate that HYRISE can create designs that are 20 to 400% quicker than either the pure-column or pure-row method. They also demonstrate that their method outperforms the prior state-of-the-art hybrid database system, Data Morphing, in terms of physical design and scalability.

In another paper, Alagiannis *et al.* mentioned the Adaptive Store, H_2O [2]. This hybrid data store automatically adjusts the physical data structure and generates the evaluated various processing techniques and access code in response to varying workloads. H_2O was built to effectively accommodate fluctuating workloads, and its modular structure allows it to adapt to new conditions with little startup time. Scanning procedures, which include most of the data, are where hybrid data layouts really shine. C++ has been used to implement H_2O, and C++ macros are used to produce specialized code for various data structures [3]. Depending on the intricacy of the query, the compilation cost for H_2O might range from 10 to 150 ms.

Another paper by Arulraj *et al*. focused on Hybrid workloads, which involve both transactional and analytical processing, which have gained popularity in recent years with the rise of HTAP (Hybrid Transactional/Analytical Processing) applications [4]. Traditional row-based storage models are optimized for transactional workloads, while column-based storage models are better suited for analytical workloads [5]. However, switching between these storage models for different types of queries can be inefficient and require manual tuning. In this paper, Huang *et al.* propose a tile-based FSM (Fragmented Storage Model) architecture as a solution for bridging the gap between row- and column-based storage models in hybrid workloads. The FSM architecture allows for a flexible and dynamic storage layout that can adapt to the needs of different types of queries at runtime [6]. It stores tuples in a partitioned and fragmented manner, with each partition containing a group of attributes called a "tile group". The FSM architecture also includes a data reorganization process that can adapt the storage layout based on the workload.

A paper named "Optimal Column Layout for Hybrid Work-loads" by Athanassoulis *et al*. presents a storage engine called Casper, which is designed to handle hybrid workloads in column-store systems [7]. Hybrid workloads involve both long-running analytical queries and short-term transactions and require systems to keep these from impacting each other's performance or correctness [8]. Casper uses snapshot isolation through multi-version concurrency control (MVCC) to allow transactions to work on a snapshot of the database at the start of the transaction. It also introduces a new concept called "ghost values" to reduce contention during update operations by allowing them to affect only one partition when possible, avoiding the need to spread changes across multiple partitions. Casper also has compression techniques like a dictionary and frame-of-reference encoding and uses a column-chunk-

© *JournalsPub 2023. All Rights Reserved*

based layout for efficient updates. The developers created a benchmark called Hybrid Access Patterns (HAP) to evaluate Casper's performance, and also compared it to other storage engines using HAP and the ADAPT benchmark. The results show that Casper consistently performs better than the other storage engines in terms of throughput and latencies for both analytical and transactional queries.

USED ARCHITECTURE

There were multiple architectures used in the papers. Each architecture has its benefits and capabilities.

HYRISE Architecture

One of the papers we reviewed used HYRISE architecture [1]. HYRISE is a main memory hybrid database system that stores data in both row and column-oriented representations. It is designed to maximize cache performance by creating vertical partitions of tables with different widths, depending on the access patterns of queries over tables. The system stores data in a combination of narrow and wide partitions, with narrow partitions being used for attributes that are accessed together frequently and wide partitions being used for attributes that are accessed infrequently. To determine the optimal partitioning, HYRISE uses a cost model that estimates the number of cache misses for a given partitioning and access pattern. The system also includes a database design algorithm that finds the partitioning that minimizes the number of cache misses for a given workload as shown in Figure 1.

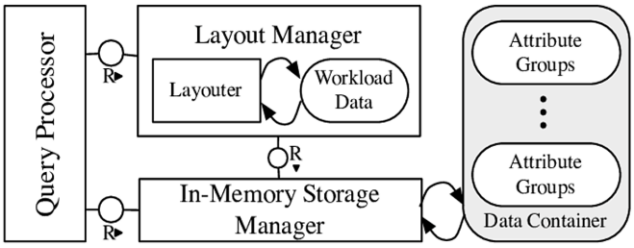

Figure 1. HYRISE Architecture.

H_2O Architecture

Another paper used the H_2O Architecture, which is implemented in C++ and uses a layer of C++ macros to generate tailored code for different data layouts [2]. The physical data layout for H_2O can be column-major, row-major, or a group of columns. The group of columns layout only includes the attributes accessed by the query and is the most efficient layout for select-project-aggregation queries. H_2O also uses custom operators that are tailored to the underlying data layout to further improve performance. The custom operators are generated on the fly to match the data layout and remove the interpretation overhead of the generic operator. Figure 2 depicts the H_2O architecture.

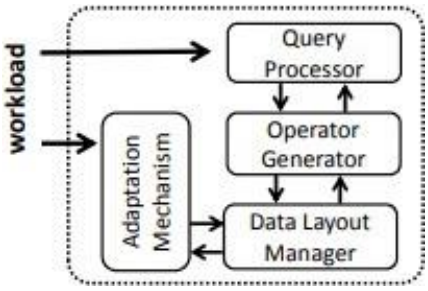

Figure 2. H_2O Architecture [2].

FSM Oriented DBMS Architecture

In another paper, we reviewed what works with Row-Stores and Column-Stores using FSM-oriented DBMS architecture. The FSM architecture was implemented in the Peloton DBMS, a multi-threaded in-

© JournalsPub 2023. All Rights Reserved

memory DBMS designed for HTAP work-loads. The FSM architecture consists of a runtime component for online query monitoring and data reorganization, as well as an execution engine and FSM storage manager. The execution engine supports all common relational operators and is implemented using the logical tile abstraction.

Casper Architecture

In the last paper, the Casper framework was used for our review [7]. Casper is written in C++ and intended to be a direct replacement for any relational scan operators that also support updates. Point queries, aggregate range queries, arithmetic range queries, inserts, deletions, and updates are all supported, along with the other four access patterns from the paper. The Frequency Model, keeps an accurate estimate of access permissions over the physical storage, the minimization component uses the Mosek method for solving to make physical configuration decisions, the partitioner incorporates these decisions, and the core storage engine implements update and access activities on the partitioned data, are the main components of Casper's architecture. Casper allows for concurrent processing and uses snapshot isolation or locking to guarantee the consistency of updates to various partitions as shown in Figure 3.

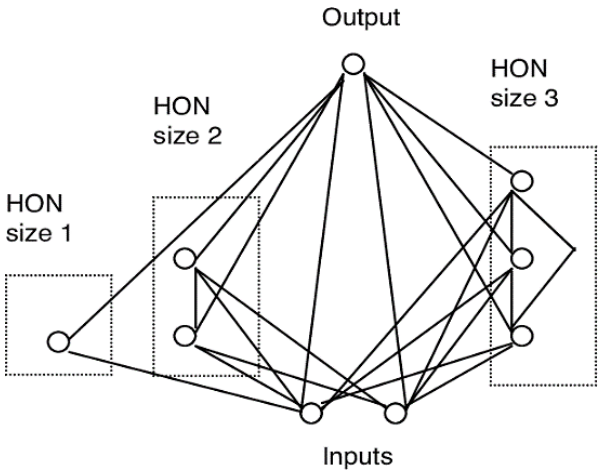

Figure 3. Casper Architecture.

USED MODELS

Several models were used in the four papers we reviewed.

Cost Model

To determine the optimal partitioning, in the first paper, HYRISE uses a cost model that estimates the number of cache misses for a given partitioning and access pattern [1]. The model takes into account the width of the partitions, the selectivity of the queries, and the distribution of the data. The model is able to accurately predict the number of cache misses for a variety of partitioning and access patterns. The authors, Grund *et al*. also developed a database design algorithm that finds the partitioning that minimizes the number of cache misses for a given workload. The algorithm is based on a divide-and- conquer approach that clusters primary partitions that are often accessed together and generates optimal sub-layouts for each cluster. The sub-layouts are then combined to form the final layout.

Machine Learning and Rule-based Model

In the second paper, the developers use a combination of machine learning and rule-based approaches to adapt to changing workloads. The system uses a model-based approach to select the most appropriate data layout for a given query based on the characteristics of the query and the available data layouts. The system also uses a machine learning model to predict the optimal physical data layout for a given query based on the characteristics of the query and the available data layouts.

© *JournalsPub 2023. All Rights Reserved*

ADAPT Bench-Marking Model

To evaluate the performance of the FSM architecture in the third paper, the authors Arulraj *et al*. developed the ADAPT benchmark, which consists of queries commonly found in enterprise work-loads [4]. The ADAPT database has two tables: a narrow table with 50 attributes and a wide table with 500 attributes. The workload includes insert, scan, aggregate, arithmetic, and join queries. The authors also compared the FSM architecture to traditional NSM (Non-Fragmented Storage Model) and DSM (Disjoint Storage Model) storage models as shown in Figure 4.

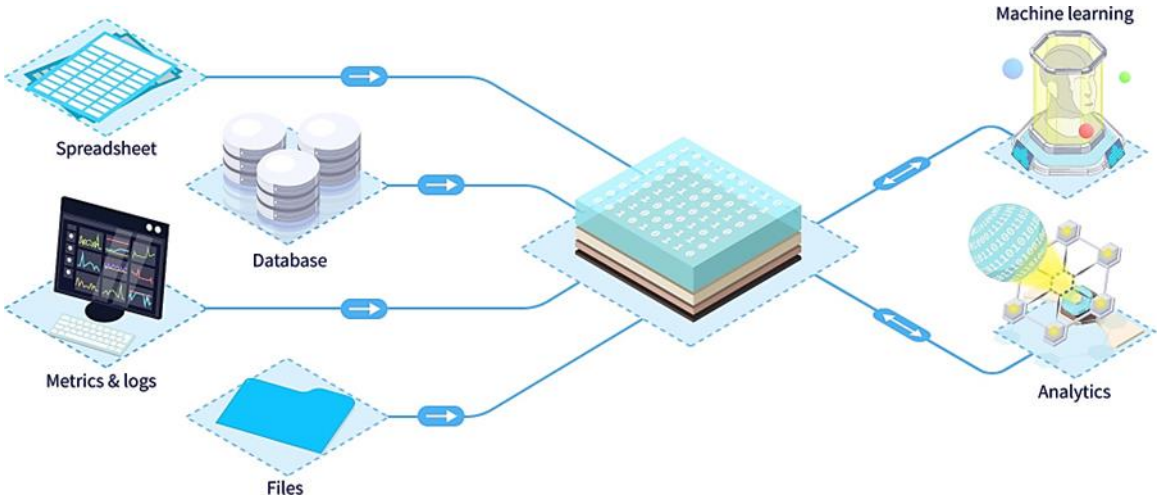

Figure 4. Machine Learning-based database model example.

Frequency, Optimization and Ghost-value Model

The fourth paper by Athanassoulis *et al*. indicated that Casper uses several models to optimize the physical layout of data for hybrid workloads [7]. The first model is the Frequency Model, which maintains an accurate estimate of the access patterns over the physical storage by tracking the number of times each partition is accessed. The second model is the optimization model, which formulates the physical layout decision as a mixed-integer linear program (MILP) and uses the Mosek solver to find the optimal solution. The optimization model takes into account the access frequencies, the cost of each operation, and the space constraints when deciding the size and number of partitions. The third model is the ghost value model, which determines the number of ghost values to be inserted into each partition in order to reduce contention in update operations.

RESULT ANALYSIS

In the first paper, the authors Grund *et al*. assessed the framework's efficiency against a customer-generated workload [1]. In order to verify the cost model and database designer, they compared the efficiency of their hybrid approach to that of all-row and all-column designs. According to the findings, HYRISE consistently matches or exceeds the best row and column performance. They further demonstrate that the model accurately predicts real-world cache misses and that cache failure are a strong predictor of performance. The study's authors discovered that HYRISE consumes four times fewer cycles than the all-row configuration and around 1.6 times faster than the all-column structure on OLTP queries, with essentially equal performance on analytical queries. They also discovered that HYRISE outperforms the prior leading hybrid database system, Data Morphing.

In the second paper by Alagiannis *et al*., H_2O has been evaluated using both micro-benchmarks and a real-life workload from the SkyServer project [2]. The results show that H_2O can gracefully adapt to changing workloads and always stays close to optimal performance. H_2O also performs better than a static data lay-out advisor and has a lower overhead compared to other hybrid systems. In a sensitivity analysis of various parameters that affect the behavior of H_2O, the group of columns layout was found to be the most efficient for select-project-aggregation queries and outperformed the row-major and

© JournalsPub 2023. All Rights Reserved

column-major layouts. H_2O also showed significant performance improvements when using custom operators tailored to the underlying data layout.

The results of the third paper by Arulraj *et al*. showed that the FSM architecture outperformed both NSM and DSM in terms of execution time for most query types and selectivity settings [4]. The FSM architecture was also able to adapt to changing workloads at runtime, converging to an optimal storage layout for each query segment. In comparison to H_2O, a state-of-the art adaptive storage manager, the FSM architecture demonstrated better performance in terms of data reorganization strategies and the ability to maintain multiple copies of data with different layouts. Overall, the FSM architecture proved to be a promising solution for bridging the archipelago between row- and column-based storage models in hybrid workloads. It offers a flexible and dynamic storage layout that can adapt to the needs of different types of queries at runtime, resulting in improved performance compared to traditional storage models. The fourth paper showed Athanassoulis *et al.* conducted several experiments to evaluate the performance of Casper using the HAP and ADAPT benchmarks [7]. In the HAP benchmark, Casper consistently outperformed state-of-the-art storage engines in terms of throughput and latencies for both analytical and transactional queries. The results also showed that Casper is able to meet performance constraints for insert operations while maintaining high overall system throughput, and is robust to a certain level of workload uncertainty. In the ADAPT bench- mark, Casper outperformed other storage engines in terms of throughput and latencies for both analytical and transactional queries. Overall, the results demonstrate that Casper is a highly effective storage engine for hybrid workloads, offering significant performance improvements for both analytical and transactional queries.

COMPARATIVE DISCUSSION

The four papers discuss different approaches to optimizing hybrid workloads in database systems.

HYRISE is a main memory hybrid storage engine that aims to optimize both transactional and analytical workloads by using a combination of row- and column-based storage [9]. It uses a modular architecture and a query optimizer that can choose the best storage layout for each query. HYRISE also supports multi-version concurrency control and snapshot isolation to enable efficient updates [1].

H_2O is a hands-free adaptive store that aims to optimize hybrid workloads by automatically adapting to changing work-load patterns. It uses a combination of row- and column-based storage and employs techniques such as compression and partitioning to improve performance. H_2O also supports transactional workloads through the use of a lock manager [2]. Bridging the Archipelago between Row-Stores and Column-Stores for Hybrid Workloads proposes a hybrid storage engine that uses a combination of row- and column-based storage to optimize hybrid workloads. It uses a query optimizer to choose the best storage layout for each query and employs techniques such as compression and partitioning to improve performance.

The paper also discusses the use of ghost values to reduce contention during updates.

Optimal Column Layout for Hybrid Workloads focuses on optimizing the column layout in a hybrid storage engine to improve performance for both transactional and analytical workloads [7]. It proposes a method called Casper that uses a frequency model to accurately estimate access patterns and an optimization component to choose the optimal column layout. Casper also supports compression and partitioning and uses ghost values to reduce contention during updates [7].

Overall, all four papers address the challenge of optimizing hybrid workloads in database systems by using a combination of row- and column-based storage and employing various techniques such as compression, partitioning, and ghost values. HYRISE and H_2O focus on adaptively changing the storage layout to optimize performance while Bridging the Archipelago between Row-Stores and Column-

© *JournalsPub 2023. All Rights Reserved*

Stores for Hybrid Workloads and Optimal Column Layout for Hybrid Workloads focus on optimizing the column layout specifically [7].

LIMITATIONS AND SCOPE

While reviewing the papers we have figured out some limitations and some scopes for the future.

HYRISE: A Main Memory Hybrid Storage Engine

The main limitation is that it only supports in-memory processing, which may not be feasible for very large datasets. In addition, it currently only supports a limited set of SQL features. Future work for HYRISE includes expanding the supported SQL features and potentially adding support for out-of-memory processing.

H_2O: A Hands-free Adaptive Store

It has the limitation of relying on heuristics to determine the optimal storage layout, which may not always result in the best performance. In addition, it does not currently support transactions. Future work for H_2O includes the incorporation of more sophisticated optimization techniques and the addition of transaction support.

Bridging the Archipelago between Row-Stores and Column-Stores for Hybrid Workloads

This paper has the limitation of only supporting a small number of workloads and queries in their evaluation. In addition, the proposed system has not yet been implemented and tested on real-world datasets. Future work for this paper includes a more extensive evaluation with a wider range of workloads and the implementation and testing of the proposed system.

Optimal Column Layout for Hybrid Workloads

It has the limitation of not being robust to high levels of workload uncertainty. In addition, it does not currently support the distribution of data across multiple servers. Future work for this paper includes the development of a new robust optimization paradigm for handling high levels of workload uncertainty and the extension of the proposed techniques to support distributed data storage.

CONCLUSION

In conclusion, our survey has examined the various approaches that have been proposed to address the challenges of optimizing storage engines for hybrid workloads. We have found that there is a trade-off between the performance of updates and reads, as well as between performance and memory consumption. Some approaches prioritize update performance, while others prioritize read performance. Some approaches rely on advanced compression techniques to reduce memory consumption, while others use partitioning or ghost values to improve performance. Our survey has also highlighted the importance of taking into account the access patterns of the workload, as well as the potential for uncertainty in the workload to impact the performance of the storage engine. Future research should focus on developing robust optimization techniques to handle high levels of workload uncertainty and on finding ways to further balance the trade-offs between performance and memory consumption.

REFERENCES

1. Grund M, Krüger J, Plattner H, Zeier A, Cudre-Mauroux P, Madden S. Hyrise: a main memory hybrid storage engine. Proc VLDB Endow. 2010 Nov 1; 4(2): 105–16.
2. Alagiannis I, Idreos S, Ailamaki A. H_2O: a hands-free adaptive store. In Proceedings of the 2014 ACM SIGMOD international conference on Management of data. 2014 Jun 18; 1103–1114.
3. Antcheva I, Ballintijn M, Bellenot B, Biskup M, Brun R, Buncic N, Canal P, Casadei D, Couet O, Fine V, Franco L. ROOT—A C++ framework for petabyte data storage, statistical analysis and visualization. Comput Phys Commun. 2009 Dec 1; 180(12): 2499–512.

© JournalsPub 2023. All Rights Reserved

4. Arulraj J, Pavlo A, Menon P. Bridging the archipelago between row-stores and column-stores for hybrid workloads. In Proceedings of the 2016 International Conference on Management of Data. 2016 Jun 14; 583–598.
5. Huang D, Liu Q, Cui Q, Fang Z, Ma X, Xu F, Shen L, Tang L, Zhou Y, Huang M, Wei W. TiDB: a Raft-based HTAP database. Proc VLDB Endow. 2020 Aug 1; 13(12): 3072–84.
6. Backasch R, Hempel G, Blochwitz C, Werner S, Groppe S, Pionteck T. An architectural template for composing application specific datapaths at runtime. In 2015 IEEE International Conference on ReConFigurable Computing and FPGAs (ReConFig). 2015 Dec 7; 1–6.
7. Athanassoulis M, Bøgh KS, Idreos S. Optimal column layout for hybrid workloads. Proc VLDB Endow. 2019 Sep 1; 12(13): 2393–407.
8. Wang JC, Ding D, Wang H, Christensen C, Wang Z, Chen H, Li J. Polyjuice: High-Performance Transactions via Learned Concurrency Control. In 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI). 2021 Jul 14; 198–216.
9. Pinnecke M, Broneske D, Durand GC, Saake G. Are databases fit for hybrid workloads on GPUs? A storage engine's perspective. In 2017 IEEE 33rd International Conference on Data Engineering (ICDE). 2017 Apr 19; 1599–1606.

© *JournalsPub 2023. All Rights Reserved*