International Journal of Electrical and Computer Engineering (IJECE)
Vol. 11, No. 2, April 2021, pp. 1780~1795
ISSN: 2088-8708, DOI: 10.11591/ijece.v11i2.pp1780-1795

Growth of relational model: Interdependence and complementary to big data

Sucharitha Shetty[1], B. Dinesh Rao[2], Srikanth Prabhu[3]
[1,3]Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, India
[2]Manipal School of Information Science, Manipal Academy of Higher Education, India

Article Info

Article history:

Received Mar 20, 2020
Revised Jul 27, 2020
Accepted Nov 6, 2020

Keywords:

Big data
Hybrid transaction/analytical processing
Online analytical processing
Online transactional processing

ABSTRACT

A database management system is a constant application of science that provides a platform for the creation, movement, and use of voluminous data. The area has witnessed a series of developments and technological advancements from its conventional structured database to the recent buzzword, bigdata. This paper aims to provide a complete model of a relational database that is still being widely used because of its well known ACID properties namely, atomicity, consistency, integrity and durability. Specifically, the objective of this paper is to highlight the adoption of relational model approaches by bigdata techniques. Towards addressing the reason for this in corporation, this paper qualitatively studied the advancements done over a while on the relational data model. First, the variations in the data storage layout are illustrated based on the needs of the application. Second, quick data retrieval techniques like indexing, query processing and concurrency control methods are revealed. The paper provides vital insights to appraise the efficiency of the structured database in the unstructured environment, particularly when both consistency and scalability become an issue in the working of the hybrid transactional and analytical database management system.

This is an open access article under the CC BY-SA license.

Corresponding Author:

Sucharitha Shetty
Department of Computer Science and Engineering
Manipal Institute of Technology
Manipal Academy of Higher Education, Manipal, India
Email: sucha.shetty@manipal.edu

1. INTRODUCTION

Internet computing, smartphones have made every layman, academician, industrialist, developer a part of the database management system. In light of this awareness, there is a pressing demand to provide the best of knowledge in the least time to these customers. Many different types of databases are in use today. However, for many years, the relational database in various splendiferous models captured in the form of row-store, column-store and hybrid store has been around. The traditional model of a database of single machines can no longer meet the growing demands. The information must be put away on disk or in a remote server as the measure of information increases. Therefore, the queries used to fill the visualization may take minutes, hours or longer to return, bringing about long holding up times between the co-operations of every user and diminishing the capacity of the user to investigate the information rapidly [1].

Today, the trend is moving away from a "one-size-fits-all" structure [2]. During the early 1990s, companies wanted to collect data from multiple operating databases into a business intelligence data ware-

house. This information had to be processed to deliver the best results in the shortest response time resulting in the distinction between online analytical processing (OLAP) and online transactional processing (OLTP). OLAP systems provided answers to multi-dimensional queries that contribute to decision-making, planning and problem-solving. On the other hand, OLTP mainly concentrates on the CRUD operations (create, read, update and delete). Different column-oriented OLAP frameworks, for example, BLU [3], Vertica [4], Vectorvise, as well as many in-memory OLTP frameworks, including VoltDB [5], Hekaton [6] and many others, have seen a rise in the database sector. Hardware developments have been the biggest push for new database designs. Over the years, disk space has undergone an increase in Moore's law [7]. The second generation of OLAP and OLTP systems makes better use of multi-core and large memories.

All of these innovations have contributed to the rise of a new database paradigm known as big data that does not provide all of the relational model's operations but do scale to big data sizes. NoSQL or key-value stores, such as Voldemort [8], Cassandra [9], RocksDB [10], offer fast write throughput and lookups on the primary key [11], and very high scale-out, but lack in their query capabilities, and provide loose transactional guarantees. SQL-on-Hadoop solutions [12] (e.g., Hive, Impala, Big SQL [13], and Spark SQL [14]) use OLAP model templates and column formats to run OLAP queries over large static data volumes. These frameworks are not arranged as real-time operational databases albeit appropriate for batch processing.

The amalgamation of simultaneous transactional and analytical workloads is a subject that is now looked upon by database vendors and industry analysts. It is known as hybrid transactional and analytical processing (HTAP). This system must provide answers to simple transactional queries to complex analytical queries. Applications of HTAP include banks and trading firms which have transactions that raise the volume of data almost every second, and they also need to evaluate their fund at the same time. In industries related to telecom services, energy-related companies need to analyze their past and present states of their networks. HTAP allows decision-makers to become more conscious of the past and present state of the company and where it is going. It is also the fact that not all companies could benefit from this system as one of the factors could be a delay in the process itself rather than software. HTAP tries to address the few major drawbacks of traditional approaches [15]:

a. Architectural and technical complexity: Earlier data had to go through the process of extraction, transformation, and loading (ETL) phase i.e. the data had to be extracted from the operational database, then transformed to be loaded into the analytical database.
b. Analytical latency: Prior it would take a long time to turn live data into analytical data. This could be appropriate for certain applications like month-end combinations. But this is not feasible for real-time analyses.
c. Synchronization: If both analytical and transactional databases are segregated and the researcher attempts to drill down to the origin at some point during the analysis, then there is a risk of "out-of-synch" error messages
d. Data duplication: For keeping up predictable information all through, the conventional methodology had various duplicates of similar information that had to be controlled and overseen.
e. Reader-writer problem: Conventional DBMS had the trouble executing both OLAP and OLTP together as readers would block writers and writers would block readers [16].

As a result, the demand for mixed workloads resulted in several hybrid data architectures, as demonstrated in the "lambda" architecture, where multiple solutions were grouped, but this increased complexity, time-consumption, and cost. The lambda architecture is divided into three layers: a) the batch layer for han- dling and pre-processing append-only collection of raw data; b) the serving layer which indexes batch views so that we can accommodate low latency ad-hoc queries; and c) the speed layer which manages all low latency requirements using quick and incremental algorithms over recent data only [17, 18]. The study covers the below objectives:

a. The paper illustrates development and innovation done towards the refinement of the relational model in terms of data storage layout and efficient data retrieval methods.
b. It acknowledges the complexities involved in the simultaneous usage of both transactional and analytical processing in a relational model with the introduction of the current approach known as hybrid transactional and analytical processing.
c. The paper incorporates an understanding of big data models adopting a structured database.

2. DATABASE

The database is the fundamental heart for the working of huge information banks which are ensured by not uncovering how the information is sorted out in the framework. While structuring a database, the fundamental three conditions should be taken care of: Ordering dependency (whether the data should be stored in an ordered way or not, keeping into account insertion and deletion as a bottleneck), indexing dependency

(the selection of attributes for indexes should be such that they can be built and removed from time to time) and access path dependency (the path to be chosen to fetch the data either sequentially or by indices) [19].

Among the various data models for the database management system proposed over a while, this paper focuses on the relational model which is the most prominent. Early referencing strategies in programming languages didn't distinguish the logical structure of the information from its physical structure. However, the relational model which was introduced in the mid-1970s gave a clear division [20]: information is organized as tables and it is up to the database management framework to choose how these tables are taken care of in the memory [21].

Later, as the size of data increased with the advent of the internet, the need for fast retrieval of data became the need of the hour. This lead to the division of a single relational model to separate transactional and analytical data models maintaining the structured database. On an everyday premise, the mix of addition, deletion, update and basic queries on data is named as online transactional processing (OLTP). However, a typical issue with the database is it appears to be very broad, and the techniques of query processing frequently search the whole set repeatedly. So one of the preliminary approaches used was sampling. In contrast to customary information, be that as it may, sampling information is inalienably questionable, for example, it doesn't speak to the whole populace information. It is along these lines beneficial to return the results of the query as well as the confidence intervals that show the outcomes' accuracy [22]. Besides, there may be no or too few samples in a certain section in a multidimensional space [23]. It needs some further research to produce trustworthy findings. The conventional astuteness says that since (1) numerous samples must be taken to accomplish sensible accuracy, and (2) sampling algorithms must do an I/O disk per tuple inspected, while query assessment algorithms can amortize the expense of an I/O disk on a page containing all tuples, and (3) in sampling, the overhead of starting an activity is n times when n is the number of samples taken; however if the query is analyzed, the overhead is just once brought about, the cost of evaluating the size of the query by sampling is too high to ever be effective [24]. This eventually leads to online analytical processing (OLAP) becoming attractive. Further, as velocity, assortment and volume of information have expanded the requirement for a mix of both transactional and explanatory databases is unavoidable for constant real-time decision making [25]. This platform which empowers to deal with an assortment of information and executes the two exchanges and complex inquiries with low latencies are termed as hybrid transactional and analytical processing (HTAP). Figure 1 represents a block diagram that summarizes the key points covered in the paper.

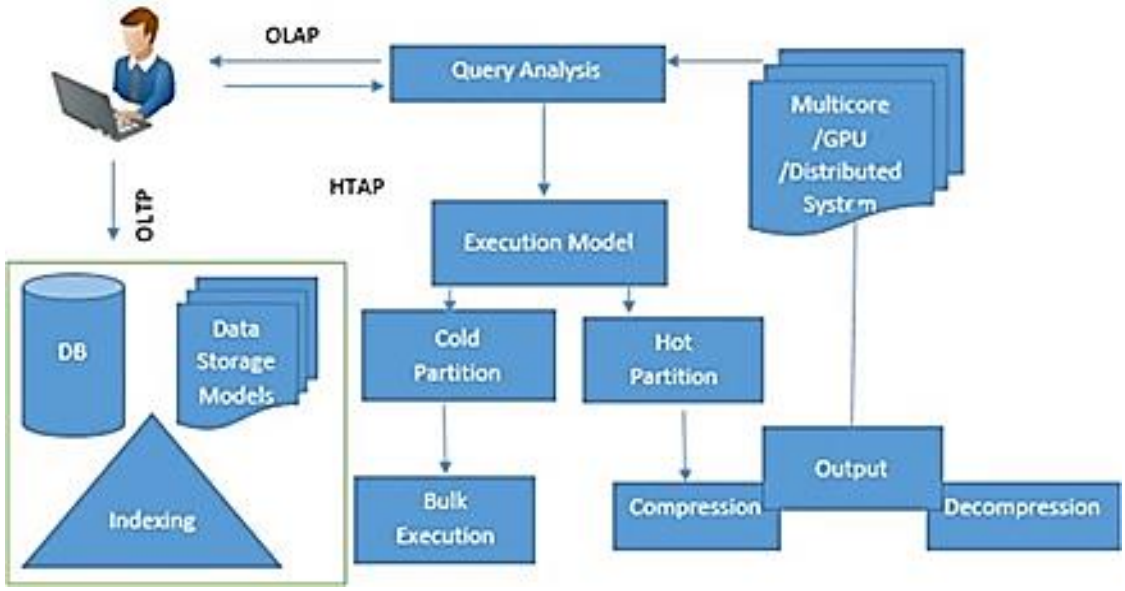

Figure 1. Block diagram of HTAP

3. DATA STORAGE MODELS

The structure of the database can be seen as the process of representation, index creation, processing, and maintenance. Nelson *et al*. [26] proposed the evolutionary list file [ELF] structure which was adopted in the design of the relational model. ELF structure emphasis the key elements of file structure as entities, lists,

and links. The earliest implementation of the relational approach can be seen as a research project launched in the summer of 1968 at the Massachusetts Institute of Technology known as Mac advanced internet management system (MacAIMS) [27], incorporating reference numbers now known as primary keys, user access rights, and B-tree concept. Thereafter, several relational data models were proposed of which a few are mentioned below.

3.1. N-ary storage model (NSM)

This model is also well-known as a row-store where N stands for the number of columns in the table. Most of this DBMS followed the volcano-style processing model in which one tuple (or block) of information is processed at a time. Row stores claim to stake for workloads that query an increased number of attributes from wide tables. These workloads include business (e.g., network performance and management applications) as well as in scientific fields (e.g., neuroscience, chemical, and biological applications). Several important datasets, related to the medical field are made of more than 7000 attributes. One of the solutions being, constantly increasing support for large tables, for instance, SQL Server today permits 30K columns per row but at the maximum, only 4096 columns are allowed in the SELECT query. This storage layout shows slower performance for OLAP queries where there is a need for a limited number of attributes, as row-stores generate disk and memory bandwidth overhead [28, 29].

3.2. Decomposition storage model (DSM)

DSM [30] partitions an n-property relationship vertically into n sub-relations, which are accessed based on the interest for the corresponding values of the attribute. In each sub-relation of the component, a surrogate record-id field is required with the goal that records can be arranged together. Sybase-IQ incorporates vertical partitioning for data warehouse applications within conjunction with bitmap indices [31]. DSM needs consid- erably less I/O than NSM for table scans with just a few attributes. Queries involving multiple attributes from a relationship need to spend extra time bringing the participating sub-relations together; this extra time can be important. There is also space wastage for the extra record-id field in each of the sub-tables.

3.3. Partition attributes across (PAX)

PAX [32] addresses the issue of low cache utilization in NSM. Unlike NSM, all values of a specific attribute are grouped by PAX on a mini page within each page. PAX makes full use of the cache resources during sequential access to data, as only the values of the required attribute are loaded into the cache. PAX does not optimize the I/O between disk and memory relative to DSM. Unlike NSM, PAX loads all the pages that belong to the relationship for scans into the memory, regardless of whether the request needs all or only a few of the attributes.

3.4. Multi-resolution block storage model (MBSM)

MBSM [33] is responsible for handling both I/O performance and main memory cache use. It shares the positive cache behavior of PAX where attributes are stored in physical contiguous segments. Disk blocks are grouped in super-blocks with a single-record stored in a partitioned manner among the blocks and then these blocks in-turn are organized in fixed-size mega-blocks on the disk. MBSM only loads pages from the disk with the values of related attributes. Better performance while inserting/updating data and does not need a join to rebuild records. They work faster for sequential scanning and slower for random. They are optimized for un-compressed fixed-width data.

3.4. Column-stores

Column-store systems [34] partition a database vertically into a collection of separately stored individual columns. Here, instead of the primary keys, virtual ids are used as tuple identifiers, as ids reduce the width of the information stored on the disk. The offset location is used as the virtual identifier. It is useful for OLAP operations. It offers better cache usage bypassing cache-line sized blocks of tuples among operators and working on different values one after another, instead of utilizing an ordinary tuple-at-a-time iterator. It also provides late materialization, compression and decompression techniques, Database cracking and Adaptive indexing features. Druid [35] is an open-source real-time analytical data store which is column oriented in structure. It supports two subsystems one in the historical node for read-optimization and others in the real-time nodes for write-optimization. However, the real-time system is used to ingest high data but does not support data updates. The drawbacks include frequent multiple inserts and updates are slow.

3.5. Hybrid stores

If one knows a priori the workload for a given application, then it is possible to use a sophisticated hybrid system that can be precisely calibrated for the given workload. Through the analysis of query workloads, a cache-efficient attribute layout can be designed. This is the concept followed in data morphing [36]. This layout can then be used as a template for storing data in the cache. As a consequence, fewer cache misses occur during query processing when compared to NSM and DSM under all types of workloads. The drawback of this technique is that, if the query workload changes dynamically reorganizing data according to new data layout can take time and make performance decrease. Along these lines, below are mentioned few such hybrid systems [37].

a. In clothe storage model, separate data layouts are designed for in-memory and non-volatile storage. Clotho creates in-memory pages individually tailored for compound and dynamically changing workloads, and enables efficient use of different storage technologies (e.g., disk arrays or MEMS-based storage devices) [38]. The layout on the disk follows the PAX model and the in-memory pages contain attributes according to queries' demands.
b. Fractured mirrors retain the best of the two data models namely NSM and DSM. Fractured mirrors create data layouts of both NSM and DSM in the two disks but they are not identically mirrored [39]. If the NSM table is divided into two equal segments and similarly DSM is divided into two equal segments. Then, one combination will be on one disk and so on the other. Since, if there is a query skew then there will be maximum disk utilization. Also, due to fractured mirrors, better query plans can be formulated to minimize the L1 and L2 cache misses. The drawback of Fractured Mirrors design is the duplication of space
c. H2O [40] presents two new concepts. One is data access patterns and multiple storage layout for a single-engine. The system decides an adaptive hybrid system during query processing that which partitioning model is best for that query class and its corresponding data pieces. As the load changes based on the query, the storage, and the access patterns will continuously change to it accordingly. Vertical partitioning is done based on creating groups of columns that are frequently accessed together.
d. Dynamic vertical partitioning [41] uses a set of rules for the development of an active system called dynamic vertical partitioning-DYVEP which dynamically partitions distributed database in the vertical order. In the case of static vertical partitioning, some attributes may not be used by queries hence wastage of space. So, in this approach DYVEP monitors queries and collects and evaluates relevant statistics for the vertical partitioning process to decide if a new partitioning is necessary, and if so, it fires the Vertical Partitioning algorithm. If the vertical partitioning scheme is better than the already existing one, then the system reorganizes the scheme. The architecture of DYVEP has three modules: Statistic collector, partitioning processor, and partitioning reorganizer. Acceptable query response time was observed when experimented on a benchmark database TPC-H.

Narrow partitions perform better for columns accessed as a major aspect of analytical queries (for example through successive scans). Conversely, wider partitions perform better for columns accessed as a major aspect of OLTP-style query, because such exchanges frequently insert, delete, update, or access huge numbers of a row's fields. It was seen that performance improvement of 20 percent to 400 percent over pure all column or all row structures is both increasingly adaptable and delivers better plans for main memory frameworks than past vertical partitioning approaches. Database device vendors have different storage engines to serve workloads with different characteristics effectively under the same software suite. For example, MySQL supports multiple storage engines (e.g. InnoDB, MyISAM), but it is difficult to communicate on the storage layer between different data formats. More specifically, a different execution engine is required for each storage engine.

4. RELATED WORKS ON DATA RETRIEVAL

This section is divided into third subsections. The first section presents various strategies adopted for data indexing. The second subsection describes the common query processing techniques used to speed up the data retrieval process. The third subsection describes the concurrency control methods used to prevent transaction mismatches.

4.1. Indexing

Relational model indexing [42] is a blessing for the database management system for fast and quick retrieval of data through bypassing the traversal of every row. An index may include a column or more, and sometimes the size of an index may grow larger than the table it is being created for, but eventually, the index will provide a rapid lookup and fast access to the data. This compensates for the overhead of having indexes. Table 1 provides a glimpse of various indexing strategies followed so far.

Table 1. Various indexing methods

Indexing Strategy	Ref	Properties	Challenges
B-Tree	[43]	to manage situations where records were expected to surpass the size of the main storage and disk operations were required	Simultaneous updates
Log Structured Merge-tree	[44]	insertion efficiency	their query performance is comparatively worse than a B- Tree
Bloom Filter	[45]	negligible space	Storage savings are at the cost of false positives. However, this drawback does not impact information processing if the likelihood of an error is small enough
R-tree	[46]	manage this multidimensional array queries effectively	Index consumes more memory space
Minmax indexes or block scope indexes	[47]	quick scanning of extremely large tables	the need to update each page range's stored min/max values as tuples are inserted into them
Inverted index	[48]	Recommended for fast keyword and full-text searches. Here, the query is resolved by looking at the word ID in the inverted index	large storage and mainte nance cost for insert, update and delete operation
Function-Based Indexes	[49]	built on expressions-works on lin guistic sorts, case insensitive sorts	can be used only on cost-based optimization
Application domain in- dexes	[49]	applies to specific application domain-used mainly in applica tions related to spatial data, video clips	application-specific software related to indexing has to be incorporated

4.2. Query processing

A query is a language expression that identifies information from a database to be retrieved. To address a query, a database optimization algorithm must choose from several known access paths [50]. The point of query optimization is to discover a query assessment technique that limits the most significant performance metric, which can be the CPU, I/O, and network time, the time-space result of locked database objects and their lock span storage costs, complete resource use, even energy utilization (for example for battery-controlled laptops), a mix of the above-mentioned, or some other performance measures [51]. Table 2 presents several related studies dealing with query optimizations. The internal specifics of implementing these access paths and deriving the associated cost functions are beyond the scope of this paper.

Table 2. Studies related to query optimization

Ref.	Objectives	Method
[52]	Proposed a method of dynam ically adjusting database per formance in a computer system based on the request made by the database query.	Access plans are generated for incoming process query and for each of the plans the desired memory is calculated and the estimated cost for the same is calculated. A comparison is done with the estimated cost for the current memory.
[53]	Addresses the problem of cleaning the stale data from Materialized Views	This clean sample is used to estimate aggregate query results. The approach is a maintenance view strategy that is employed periodically for a given materialized view. During this periodic interval, the materialized view is marked stale and an estimated up-to-date view is developed.
[54]	Proposes a caching method whereby the compiler is optimized to perform caching.	Every cache is assigned a Web variable. If a query is run with no WHERE clause, then no free Web variable is assigned. However, if a query has WHERE clause, then that condition field becomes the Web variable. Based on the insert and update on the web variable, accordingly, the cache is refilled
[55]	Design a web cache with im proved data freshness	The main memory database is used to cache result sets of previous queries for subsequent reusing. Invalidation of the cache works by replacing or deleting obsolete data directly in a caching system. For single table queries, the parser extracts the selection predicates from the where clause and maintains a list of predicates and its corresponding rowsets in the cache.
[56]	Extends traditional lazy evaluation	In lazy evaluation, by contrast, the evaluation of a statement does not cause the statement to execute; instead, the evaluation produces a single batch of all the queries involved in building the model until the data from any of the queries in the batch is needed either because the model needs to be displayed, or because the data is needed to construct a new query.
[57]	Analyses of multiple queries and suggests frequent queries and query components.	Mining query plans for finding candidate queries and sub-queries for materialized views will, in turn, optimize all the queries of these components. These frequent components may represent frequent sub-queries. Here, importance is given to the analysis of query execution plan than to the query text since most of the time query is the same but the representation of it in text form is different.
[58]	Dynamically adjusts the materialized view set using Clustering	With the help of the similarity threshold, a cluster of SQL queries is found and any new query whose similarity is below the threshold, then a new cluster is formed.

4.3. Concurrency control

OLTP DBMS supports the part that communicates with end-users. End users send requests for some purpose (e.g. buy or sell an item) to the application. These requests are processed by the application and then the transactions are executed in the database to read or write to the database. Concurrency control has a long and rich history that goes back to the start of database systems. Two-phase locking and time-stamp ordering are the two classes of algorithms used in concurrency control. The first method of ensuring the proper execution of simultaneous DBMS transactions was the two-phase locking (2PL). Under this scheme, transactions must obtain locks in the database for a particular element before a read or write operation on that element can be performed. 2PL is considered a negative approach because it believes that transactions overlap may occur and therefore need to be locked. However, if this lock is uncontrolled, it may lead to deadlock. Another approach is the Timestamp ordering (T/O) concurrency control scheme where an increase in timestamp is produced for serialization of transactions. These timestamps are used during conflicting operations such as reading and writing operations on a similar component, or two different write operations on a similar element [16].

Multi-core systems are now omnipresent [59], conventional DBMSs, however, still scale poorly beyond a few cores. The main weakness of OLTP databases is their concurrency control algorithm. Previous work showed that concurrency control algorithms suffer from bottlenecks of both fundamental and artificial scalability [60, 61]. Threads spend a significant amount of time waiting for a global lock to be obtained by a large number of cores. This is a fundamental bottleneck inherent in the way these algorithms work. The two major scalability bottlenecks: (1) conflict detection and (2) allocation of timestamps. The conflict detection bottleneck is detecting and resolving deadlocks is a burdensome operation that consumes most of the transaction's processing time as the number of cores increases, even for relatively low contention levels. Although recent work improves some artificial weaknesses, fundamental bottlenecks remain.

In conventional databases, the function of concurrency control mechanisms is to decide the interleaving order of operations between simultaneous transactions over shared data. But there is no fundamental reason to rely on concurrency control logic during the actual execution, nor is it necessary to force the same thread to be responsible for the execution of both the transaction and concurrency control logic. This significant in- sight was found in subsequent studies [62] which could lead to a complete paradigm shift in how we think about transactions [63, 64]. It is important to note that the two tasks of placing the order for accessing shared data and implementing the logic of the transaction are completely independent. Such functions can therefore tech- nically be carried out through separate threads in different phases of execution. For instance, Ren *et al*. [65] propose ORTHRUS which is based on pessimistic concurrency control, where transaction executing threads delegate locking usefulness to devoted lock manager threads. ORTHRUS depends on explicit message-passing to convey among threads, which can acquaint pointless overhead with transaction execution regardless of the accessible shared memory model of a single machine.

Multi-version concurrency control (MVCC) is a commonly used method for concurrency control, as it enables modes of execution where readers never block writers. MVCC methods have a long history as well. Rather than refreshing data objects, each update makes another version of that data object, with the end goal that simultaneous readers can, in any case, observe the old version while the update transaction continues simultaneously. Therefore, read-only transactions never need to pause and don't need to utilize locking. This is an amazingly desiring property and the motivation behind why numerous DBMSs actualize MVCC [66]. The optimistic approach to concurrency control came from Kung and Robinson, but only one-version databases were considered [67]. The optimistic techniques [68] are used in applications where transaction conflicts are very rare. In this approach, the transactions are allowed to perform as desired until a write phase is encountered. Thereby, increases the degree of concurrency.

Snapshot isolation (SI) [69] is a multi-versioning scheme used by many database systems. To isolate read-only transactions from updates, many commercial database systems support snapshot isolation which not serializable (Oracle [70], SQL Server [71] and others). However, many articles have addressed the conditions under which SI can be serialized or how it can be serialized. In 2008, Cahill *et al* released a detailed and realistic solution [72]. Their technique requires transactions to check for dependencies of read-write. Techniques such as validating read and predicate repeatability checks have already been used in the past [73]. Oracle TimesTen [70] and solidDB [74] from IBM employ multiple lock types.

4.4. RUM conjecture

The fundamental challenges faced by every researcher, programmer or network architect when designing a new access method is how to reduce read times (R), upgrade costs (U), and memory/storage overhead (M). It is observed that while optimizing in any two areas would negatively impact third. Figure 2 presents an overview of RUM conjecture.

The three criteria that people working on database systems are seeking to design for are outlined by researchers from Harvard DB lab: Read overhead, update overhead, and memory overhead [75]. Tree-based

data structures [76] are mainly targeted for improving the read requests, traded for space overhead and increase in update time as the nodes get shifted and new nodes get added. Utilizing adaptive data models gives better read execution at a greater expense of support. Adding metadata to permit traversals, (for example, fragmentary falling) may influence write time and space yet may expand read time.

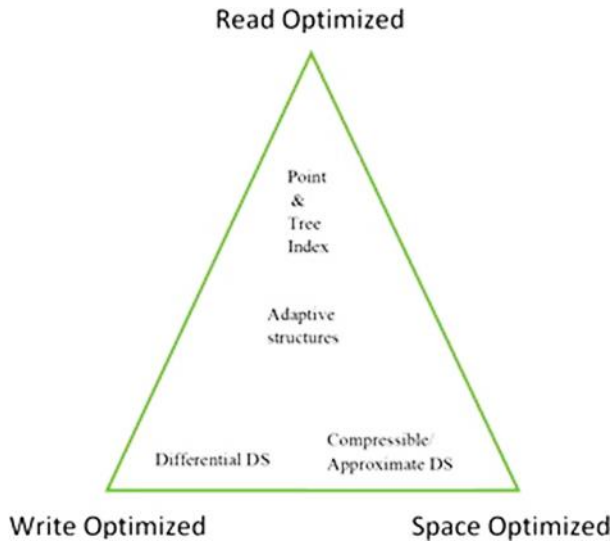

Figure 2. Indexes in the RUM space

LSM-Trees [44], the Partitioned B-tree [77] and the Stepped Merge algorithm [78] optimize the performance of write. All updates and deletions in LSM Trees do not require searching for disk data and guarantee sequential writings by deferring and buffering all insert, modifying and deleting operations. This comes at a price of higher maintenance costs and compaction requirements and more expensive reads. At the same time, LSM Trees help improve memory performance by avoiding reserving space and enabling block compression.

Compression (for instance, algorithms, for example, Gorilla compression [79], delta encoding, and numerous others) could be used to maximize memory capacity, adding some read and write overhead. Heap files and hash indexes, for example, will give great performance guarantees and lower overhead storage due to the simplicity of file format. Trade efficiency reliability can be tested by using approximate data structures such as Bloom Filter [45], HyperLogLog [80], Count Min Sketch [81], Sparse Indexes such as ZoneMaps [82], Column Imprints [83] and many others. Choosing which overhead(s) to streamline for and to what degree, stays a noticeable piece of the way toward planning another access method, particularly as hardware and workload changes after some time [84].

5. STORAGE-TIERING TECHNIQUES

Storage tiering [85], or transferring data between different storage types to suit data I/O characteristics to that of storage, is a strategy that administrators have been using for many years. Data aging is the easiest and most common reason for this. That is, as sections of the data age, they change their access patterns, and are less likely to be accessed: the data will cool down and eventually get cold. Cold data have very different I/O patterns from warm and hot data, and thus storage needs. Also, hot data may have specific storage needs for a largely random I/O pattern for an OLTP server workload. For example, SQL Server can use a design based on a per-node or a CPU. Even latches and memory objects can be dynamically partitioned by SQL Server to minimize a hot spot.

There are different customary strategies for surveying whether the data is viewed as significant: (i) frequencies analysis because of the workload, (ii) predefined business rules dependent on expert examination or functional expertise, and (iii) storing (cache) approach. The principle behind the access frequency analysis is simple: the more data is accessed, the more important it is and hence the higher the likelihood of repeated accesses. There are various approaches to quick and resource-efficient monitoring of data accesses. One example is the logging scheme proposed by Levandoski *et al*. [86] using techniques of sampling and smoothing to minimize memory usage. By contrast, rule-based approaches require application experts to work manually

[87]. An example might be: "All payments made older than a year are cold." Business rules have many advantages: knowledge of the domain. High precision can be introduced, especially given the time parameter. But at the same time, experts can devise rules that are too weak and lose the potential for optimization. Also, current online transactional or analytical processing (OLxP) frameworks are getting progressively unpredictable be- cause of the combination of different platforms (for example operational reporting, warehousing, transaction processing, information mining applications, and so forth.) which make rule-driven experts move towards impossible in the long haul. Caching is another way of finding important data. For example, page buffer monitoring using least recently used or most recently used policy [88]. Such methods can be faster than frequency approaches and consume less memory. On the other hand, they may be less precise and have high bookkeeping overhead. Developers need to update their cache configuration constantly as the workload of their application varies [89]. In reality, however, it is difficult to identify changes in workload for large applications [90]. Despite understanding the improvements in the workflow, developers still have to make a lot of effort to understand the new workload and re-configure the caching system manually.

The technique for query filtering and tuple reconstruction was suggested by Boissier *et al*. [91]. All the attributes used to evaluate the query are stored in DRAM. This info gives enough data to rebuild the information separating it into two segments: (i) tuples kept in main memory and (ii) tuples moved to disk. Two methods were presented to compute the hotness of a table cell and discussed the advantages and disadvantages of both approaches. To do as such, they presented two kinds of bit-vectors: columnar and row vectors, containing data about the hotness of a column or a row. They at that point proposed to cluster segments with comparative hotness behavior and to consolidate columnar vectors of each cluster into a single vector. This helped to reduce the complexity of sorting large datasets and tuple reconstruction.

5.1. Big data models-interdependence and complementary

5.1.1. Data storage model

SQL in Map-reduce systems: Mapreduce, well-known for processing very large datasets in a parallel fashion using computer clusters is a highly distributed and scalable program. However, when it comes to joining different datasets or finding the top-k records, it is comparatively slow. To overcome this, the relational approach needs to be adopted. As such two strategies are employed (1) Usage of SQL-like language on top of Map-reduce, e.g., Apache Pig and Apache Hive (2) inte- grate Map-reduce computation with SQL e.g., Spark.

Not only SQL (NoSQL): NoSQL provides higher availability and scalability over traditional databases by using distributed and fault-tolerant architecture [92]. With these computers can be added easily, replicated and even support consistency over a while. The various types of NoSQL being Key-value store (e.g Cassandra, Riak), Document-store (e.g, Couch DB, Mongo DB), Tabular data- stores (e.g, HBase, BigTable) and Graph Database. However, in most of these, creating custom views, normalization and working with multiple relations is difficult. Though originally SQL was not supported by NoSQL, present observations indicate the need for highly expressive and declar-ative query language like SQL. This is observed in a couple of systems like SlamData, Impala and Presto that support SQL Queries on MongoDB, HBase and Cassandra [93].

Graph database is widely explored nowadays to analyze massive online semantic web, social networks, road networks and biological networks. Reachability and shortest path queries are time- consuming to be performed in relational databases using SQL. Writing recursive and chains of joins is unpleasant. However, since RDBMS stores many relations that can be related to graphs in the real application, combination of the two, could produce better results. AgensGraph [94] is a multi-model graph database based on PostgreSQL RDBMS. It enables developers to integrate the legacy relational data model and the flexible graph data model in one database.

NewSQL: This class of database incorporates both the scalability and availability of NoSQL systems as well as the ACID properties of the relational model. Systems with SQL engines like InfoBright and MySQL Cluster support the same interface like SQL with better scalability. Sharding is done transparently in systems like ScaleBase and dbShards providing a middle layer that fragments and distributes databases over multiple nodes. Based on the application scenarios, most of these new SQL systems are used to learn SQL [60].

5.1.2. Indexing

Indexing in big data is a challenge for several reasons [95], primarily because the volume, variety, and velocity (3V's of Gartner's definition) of big data are very large; a generated index should be small (a fraction of the original data) and fast in comparison with the data because searches must consider billions or even trillions of data values in seconds. As big data models entertain multi-variable queries, indexing is required to be in place for all in-operation variables to provide faster access for individual variable search

results that are further combined for a collective response as a whole. Models should have the ability to produce smaller indexes when there is a possibility for granularity reduction. The indexes should be able to be portioned easily into sections whenever the opportunities for parallel processing arise. As the rate of data generated under the big data era is very large, to process data in place, an index should be built at the same rate. The indexing strategies in Big Data can be classified into artificial intelligence (AI) approach and non-intelligence (NAI) approach.

In AI, the indexes are formed based on a relationship that is established by observing similar patterns and traits among data items or objects. The two popular approaches for the AI indexing approach are:

a. Latent semantic indexing (AI) which uses singular value decomposition for finding patterns between unstructured databases. It consumes more memory space.
b. Hidden Markov model is based on the Markov model. It uses states connected by transitions that depend on the present state and independent of previous states. States depend on the current query and store the results and it also helps to predict future queries. It demands high computational performance. In NAI, the indexes are based on frequently queried data sets. It uses indexes used in the relational model like B-tree, R-Tree, X-Tree, LSM, bloom filter. Also, there are indexes like generalized search trees, generalized inverted index which is implemented similar to B-tree and supports arbitrary indexes.

5.1.3. Query processing

Even for newer platforms based on MapReduce and its variants, the interest in big data models which are well known for highly parallel platforms, the network-transmission cost, storage cost, I/O cost of movement of data across nodes is significant when query optimization is concerned. The popular frame- work, such as MapReduce hence lags in query processing efficiency [95]. In Map Reduce, a query also has to be translated to a set of MapReduce jobs sentence by sentence. This is time-consuming and dif- ficult. Hence, high-level languages proposed for Map-reduce such as PIG and HIVE resemble SQL in many ways. Google's BigQuery runs in the cloud to overcome the optimization Issues. Here, the data is not indexed instead stored in columnar format and b-tree architecture is used. In this tree, the data is distributed among nodes. It provides a query language similar to SQL. Cassandra employs materialized views for complex query analysis. Expensive table joins and data aggregation is excluded.

5.1.4. Concurrency control

Versioning the database is an important feature to control concurrency in big data models [95]. New changes are assigned a number, known as the version or revision number. It allows us to find the latest version of a database table and eases audit trailing. MongoDB, CouchDB, HBase, Google's BigTable are few examples which use versioning. With this powerful technique of versioning for such voluminous data, comes the potential problems like all the data that makes the particular version needs to be stored again in the database. This could lead to interdependencies between objects so to track the differences is a task.

6. HTAP

HTAP systems are capable of reducing deployment, maintenance, and storage costs by eliminating the ETL cycle and, most significantly, allowing real-time analysis over production data. Terms such as online transactional analytical processing (OLTAP) or Online transactional or analytical processing (OLxP) are also synonymous with them [96]. The Oracle RDBMS in-memory option (DBIM) [97] is an industry-first distributed dual-format architecture which enables highly optimized columnar-format processing of a database object in main memory to break performance barriers in analytical query workloads while retaining transactional compatibility with the corresponding OLTP optimized row-major format which remains in storage and accesses through the database buffer cache.

SAP HANA scale-out extension [98], a modern distributed database architecture was designed to separate transaction processing from query processing and use a high-performance distributed shared log as the persistence backbone. It provides full guarantees of ACID (Atomicity, Consistency, Isolation, and Durability) and is based on a logical timestamp framework to give MVCC-based snapshot isolation without requiring synchronous replica refreshes. Rather, it uses asynchronous update propagation to ensure consistency with the validity of timestamps. MemSQL[99] is a distributed memory-optimized SQL database which is a shared- nothing architecture that excels in analytical and transactional mixed-time processing. This exploits in-memory data storage with MVCC and memory-optimized lock-free data structures to permit highly simultaneous reading and writing of data, empowering ongoing analytics over an operational database. It ingests as well as keeps the data in row format in the in-memory. However, once data are written to disk, it is translated to columnar format for faster review.

Appuswamy *et al*. [100] designed a new architecture for the hybrid transaction and analytical processing named heterogeneous-HTAP (H2TAP). OLAP queries are data-parallel tasks. HTAP uses the kernel-based execution model for executing OLAP queries on the GPU. However, for OLTP queries which are task-parallel tasks, it uses message passing-based parallelism. Kemper *et al*. [34] proposed a hybrid system, called HyPer, which can handle both ongoing and analytical queries simultaneously by maintaining consistent snapshots of the transactional data. HyPer maintains the ACID properties of OLTP transactions as well as executes multiple OLAP queries by taking the current and consistent snapshots. The system makes use of the operating system functionality of using fork(). All modifications to OLTP are made to a different area called Delta which consists of replicated database pages. OLAP operations are given the view of virtual memory. Whenever a transactional query is performed using the fork()-operation a virtual memory snapshot is created. This will be used for executing the OLAP session. Frequently, OLAP is converged with the Delta by forking another process for the state-of-the-art OLAP session.

The project Peloton [101] intends to create a self-governing multi-threaded, in-memory HTAP system. It provides versatile data organization of data, which changes the data design at run-time based on-demand type [102]. Wildfire [103] is an IBM Research project which produces an HTAP engine where both analytical and ongoing requests go through the same columnar data format., i.e. Parquet [104] (non-proprietary storage format), open to any reader for all data. The Spark ecosystem is also used by Wildfire to allow distributed analytics on a large scale. This allows analysis of the latest committed data immediately by using a single data format for both data ingestion and analysis.

Sadoghi *et al*. [105] proposed L-Store (direct information store) that joins ongoing transaction and analytical workload handling inside a single unified engine by creating a contention-free and lazy staging of data from write-optimized to read-optimized data. Pinot [106] is an open-source relational database that is distributed in nature proposed by LinkedIn. Though it is designed mainly to support OLAP, for applications that require data freshness, it is directly able to ingest the streaming data from Kafka. Like a traditional database, it consists of records in tables that are further divided into segments. The data in segments is stored in columnar format. Hence, data size can be minimized using bit packing, etc.

7. PERFORMANCE FACTORS

Various key performance factors affect the performance of the database. Learning these variables helps identify performance opportunities and avoid problems [107]. The performance of the database is heavily dependent on disk I/O and memory utilization. The need to know the baseline performance of the hardware on which the DBMS is deployed is needed to set performance expectations accurately. Hardware component performance such as CPUs, hard drives, disk controllers, RAM, and network interfaces will have a significant impact on how quickly your database performs. System resources are, therefore, one of the key factors affecting performance. The overall system performance may be influenced by DBMS upgrades. Formulation of the query language, database configuration parameters, table design, data distribution, etc., permit the optimizer of the database query to create the most productive access plans.

The workload equals all the requests from the DBMS, and over time it varies. The overall workload is a blend at some random time of user queries, programs, batch tasks, transactions, and machine commands. For example, during the preparation of month-end queries, there is an increase in workload whereas the other days' workload is minimum. Workload strongly influences database performance. The selection of data structures that adapt to changes in workloads and pinnacle request times plans help to prepare for the most productive utilization of the system resources and enables the maximum possible workload to be handled.

The transaction throughput generally measures the speed of a database system, expressed as a number of transactions per second. Disk I/O is slower relative to other system resources such as CPU which reduces throughput. Hence, techniques are identified to keep more data in cache or memory. The contention is another condition where at least two workload components endeavor to utilize the system in a clashing way for instance, multiple queries endeavoring to refresh a similar bit of information simultaneously or various workloads going after system resources. Along these lines, throughput diminishes as the contention increments. Latency otherwise called response time or access time is a proportion of to what extent it takes the database to react to a single request. Two such latencies in the database are (i) write latency is the number of milliseconds required to fulfill a single write request. (ii) read latency is the number of milliseconds required to fulfill a read request [108]. In parallel databases, every lock operation in a processor introduces a lot of latency. Overload Detection is also desired to restrict the response time of a server to be below a certain threshold to retain customers [109]. Nevertheless, if many user transactions are executed concurrently, a database system can be overwhelmed. As a consequence, computing resources such as CPU cycles and memory space can be exhausted. Also, because of data contention, most transactions can be blocked or prematurely ended and restarted.

In terms of physical space and query processing time, cost models are useful in analyzing the cost [110]. Cost models provide the inputs (data distribution statistics and operator execution algorithms in a query plan) needed to estimate the cost of query plans and thereby facilitate the selection of optimal plans [111, 112]. An important concept underlying the execution of a query plan is that it takes fewer resources and/or minimum time to process fewer data [113]. What series of operations require fewer data and time for request execution is not always clear. Thus, the cost model provides some real insight into a database system's actual physical design and performance.

8. CONCLUSION

The study focused on the importance of relational model merits to the big data. Potential determinants (e.g. data storage layouts and data retrieval techniques) related to the relational model were also highlighted. The paper mainly contributed to the literature on the storage layouts, indexing, query processing, and concurrency control techniques in the relational model. The study revealed there are a few key areas that still need to be worked upon in the relational models. The identification of gaps in indexing has led to an understanding that it is not straightforward to index the data being distributed and accessed on a large scale to allow efficient point lookups. Faster point access to these objects is still an open issue. Efficient indexing algorithms have been explored, but as the RUM conjecture states, any two optimizations lead to the third being reduced. Con-currency in many-core systems is still a matter of research. In the case of MVCC, the overhead grows with each concurrent update.

The paper also highlighted the on-going research on hybrid transactional and analytical processing (HTAP). While there are many systems classified as HTAP solutions out there, none of them support true HTAP. Current frameworks provide a reasonable platform to support transactional as well as analytical requests when sent separately to the system. Nonetheless, none of the existing solutions support effective transactional and analytical request processing within the same transaction. However, today most HTAP systems use multiple components to provide all the desired capabilities. Different groups of people usually maintain these different components. It is, therefore, a challenging task to keep these components compatible and to give end-users the illusion of a single system. Last, the study also proffered the acceptance and implementation of a few significant methodologies of the relational model in big data. The present system demands for data to be both "offensive" (data being exploratory) and "defensive" (control of data).

REFERENCES

[1] M. Procopio, et al., "Load-n-go: Fast approximate join visualizations that improve over time," in *Proceedings of the Workshop on Data Systems for Interactive Analysis (DSIA)*, Phoenix, AZ, USA, 2017, pp. 1-2.

[2] P. Atzeni, et al., "Data modeling in the nosql world," *Computer Standards & Interfaces*, vol. 67, p. 103149, pp. 1-36, 2020.

[3] V. Raman, et al., "Db2 with blu acceleration: So much more than just a column store," *Proceedings of the VLDB Endowment*, vol. 6, no. 11, 2013, pp. 1080-1091.

[4] M. F. Lamb, et al., "The vertica analytic database: C-store 7 years later," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, 2012, pp. 1790-1801.

[5] M. Stonebraker and A. Weisberg, "The voltDB main memory DBMS," *IEEE Data Engineering Bulletin*, vol. 36, no. 2, pp. 21-27, 2013.

[6] C. Diaconu, et al., "Hekaton: Sql server's memory-optimized oltp engine," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 2013, pp. 1243-1254.

[7] G. Moore, "Moore's law," *Electronics Magazine*, vol. 38, no. 8, p. 114, 1965.

[8] R. Sumbaly, et al., "Serving large-scale batch computed data with project voldemort," in *Proceedings of the 10th USENIX conference on File and Storage Technologies*, 2012, pp. 1-18.

[9] Cassandra, "Apache cassandra," 2014. [Online]. Available: http://planetcassandra.org/what-is-apache-cassandra.

[10] S. Dong, et al., "Optimizing space amplification in rocksdb," in *CIDR*, vol. 3, p. 3, 2017.

[11] M. A. Qader, et al., "A comparative study of secondary indexing techniques in lsm- based nosql databases," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 551-566.

[12] Z. H. Jin, et al., "Cirrodata: Yet another SQL-on-hadoop data analytics engine with high performance," *Journal of Computer Science and Technology*, vol. 35, no. 1, pp. 194-208, 2020.

[13] F. O¨zcan, et al., "Hybrid transactional/analytical processing: A survey," in *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, pp. 1771-1775.

[14] M. Armbrust, et al., "Spark sql: Relational data processing in spark," in *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, 2015, pp. 1383-1394.

[15] M. Pezzini, et al., "Hybrid transaction/analytical processing will foster opportunities for dramatic business innovation," *Gartner*, 2014.

[16] M. Mohamed, et al., "An improved algorithm for database concurrency control," *International Journal of Information Technology*, vol. 11, no. 1, pp. 21-30, 2019.

[17] Y. Tian, et al., "Dinodb: Efficient large- scale raw data analytics," in *Proceedings of the First International Workshop on Bringing the Value of Big Data to Users (Data4U 2014)*, 2014, pp. 1-6.
[18] M. Kiran, et al., "Lambda architecture for cost-effective batch and speed big data processing," in *2015 IEEE International Conference on Big Data (Big Data)*, 2015, pp. 2785-2792.
[19] E. F. Codd, "A relational model of data for large shared data banks," *Communications of the ACM*, vol. 13, no. 6, pp. 377-387, 1970.
[20] T. Wolfson, et al., "Break it down: A question understanding benchmark," *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 183-198, 2020.
[21] G. Strawn and C. Strawn, "Relational databases: Codd, stonebraker, and Ellison," *IT Professional*, vol. 18, no. 2, pp. 63-65, 2016.
[22] X. Wei, et al., "Online adaptive approximate stream processing with customized error control," *IEEE Access*, vol. 7, pp. 25 123-25 137, 2019.
[23] S. Agarwal, et al., "Blinkdb: queries with bounded errors and bounded response times on very large data," in *Proceedings of the 8th ACM European Conference on Computer Systems*, 2013, pp. 29-42.
[24] R. J. Lipton, et al., "Efficient sampling strategies for relational database operations," *Theoretical Computer Science*, vol. 116, no. 1, pp. 195-226,1993.
[25] C. W. Olofson, "The analytic-transactional data platform: Enabling the real-time enterprise," *CBS Interactive*, 2014.
[26] T. H. Nelson, "Complex information processing: a file structure for the complex, the changing and the indeterminate," in *Proceedings of the 1965 20th national conference*, 1965, pp. 84-100.
[27] R. C. Goldstein and A. J. Strnad, "The macaims data management system," in *Proceedings of the 1970 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control*, pp. 201-229, 1970.
[28] K. Sterjo, "Row-store/column-store/hybrid-store," pp. 1-10, 2017.
[29] L. Sun, et al., "Skipping-oriented partitioning for columnar layouts," *Proceedings of the VLDB Endowment*, vol. 10, no. 4, 2016, pp. 421-432.
[30] D. Tsitsigkos, et al., "A two-level spatial in- memory index," *arXiv preprint arXiv: 2005.08600*, 2020.
[31] H. Lang, et al., "Tree-encoded bitmaps," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 937-967.
[32] M. Olma, et al., "Adaptive partitioning and indexing for in situ query processing," *The VLDB Journal*, vol. 29, no. 1, pp. 569-591, 2020.
[33] L. Sun, "Skipping-oriented data design for large-scale analytics," Ph.D. dissertation, UC Berkeley, 2017.
[34] T. Li, et al., "Mainlining databases: Supporting fast transactional workloads on universal columnar data file formats," *arXiv preprint arXiv: 2004.14471*, 2020.
[35] F. Yang, et al., "Druid: A real-time analytical data store," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, 2014., pp. 157-168
[36] J. Patel and V. Singh, "Query morphing: A proximity-based data exploration for query reformulation," in *Computational Intelligence: Theories, Applications and Future Directions*, vol. 1, pp. 247-259, 2019.
[37] M. Grund, J. Kru¨ger, H. Plattner, A. Zeier, P. Cudre-Mauroux, and S. Madden, "Hyrise: a main memory hybrid storage engine," *Proceedings of the VLDB Endowment*, vol. 4, no. 2, pp. 105-116, 2010.
[38] M. Shao, et al., "Clotho: Decoupling memory page layout from storage organization," in *Proceedings of the Thirtieth international conference on Very large databases*, vol. 30, 2004, pp. 696-707.
[39] R. R. D. J. DeWitt and Q. Su, "A case for fractured mirrors," in *Proceedings 2002 VLDB Conference: 28th International Conference on Very Large Databases (VLDB)*, 2002, p. 430.
[40] S. I. Alagiannis and A. Ailamaki, "H2o: a hands-free adaptive store," in *Proceedings of the 2014 ACM SIGMOD International conference on Management of data*, 2014, pp. 1103-1114.
[41] L. Rodr´ıguez and X. Li, "A dynamic vertical partitioning approach for distributed database system," in *2011 IEEE International Conference on Systems, Man, and Cybernetics*, 2011, pp. 1853-1858.
[42] E. Bertino, et al., "Indexing techniques for advanced database systems," *Springer Science & Business Media*, vol. 8, 2012.
[43] M. A. Awad, et al., "Engineering a high- performance gpu b-tree," in *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*, 2019, pp. 145-157.
[44] C. Luo and M. J. Carey, "Lsm-based storage techniques: a survey," *The VLDB Journal*, vol. 29, no. 1, pp. 393-418, 2020.
[45] B. Ding, et al., "Bitvector-aware query optimization for decision support queries," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 2011-2026.
[46] H. Ibrahim, et al., "Analyses of indexing techniques on uncertain data with high dimensionality," *IEEE Access*, vol. 8, pp. 74 101-74 117, 2020.
[47] M. Zukowski and P. Boncz, "Vectorwise: Beyond column stores," *IEEE Data Engineering Bulletin*, pp. 1-6, 2012.
[48] C. O. Truica, et al., "Building an inverted index at the dbms layer for fast full text search," *Journal of Control Engineering and Applied Informatics*, vol. 19, no. 1, pp. 94-101, 2017.
[49] J. M. Medina, et al., "Evaluation of indexing strategies for possibilistic queries based on indexing techniques available in traditional rdbms," *International Journal of Intelligent Systems*, vol. 31, no. 12, pp. 1135-1165, 2016.
[50] R. Singh, "Inductive learning-based sparql query optimization," in *Data Science and Intelligent Applications*, pp. 121-135, 2020.
[51] G. Graefe, "Query evaluation techniques for large databases," *ACM Computing Surveys (CSUR)*, vol. 25, no. 2, pp. 73-169, 1993.

[52] P. Day, et al., "Dynamic adjustment of system resource allocation during query execution in a database management system," uS Patent App. 10/671,043, 2005.
[53] S. Krishnan, et al., "Stale view cleaning: Getting fresh answers from stale materialized views," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, 2015, pp. 1370-1381.
[54] Z. Scully and A. Chlipala, "A program optimization for automatic database result caching," in *ACM SIGPLAN Notices*, vol. 52, no. 1, pp. 271-284, 2017.
[55] X. Qin and X. Zhou, "Db facade: A web cache with improved data freshness," in *2009 Second International Symposium on Electronic Commerce and Security*, vol. 2, 2009, pp.483-487.
[56] S. M. Cheung and A. Solar-Lezama, "Sloth: Being lazy is a virtue (when issuing database queries)," *ACM Transactions on Database Systems (ToDS)*, vol. 41, no. 2, pp. 1-42, 2016.
[57] S. D. Thakare, et al., "Mining query plans for finding candidate queries and sub-queries for materialized views in bi systems without cube generation," *Computing and Informatics*, vol. 38, no. 2, pp. 473-496, 2019.
[58] Gong and W. Zhao, "Clustering-based dynamic materialized view selection algorithm," in *2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery*, vol. 5, 2008, pp. 391-395.
[59] Y. Huang, et al., "Opportunities for optimism in contended main- memory multicore transactions," *Proceedings of the VLDB Endowment*, vol. 13, no. 5, pp. 629-642, 2020.
[60] Y. N. Silva, et al., "Sql: From traditional databases to big data," in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, 2016, pp. 413-418.
[61] X. Yu, et al., "Staring into the abyss: An evaluation of concurrency control with one thousand cores," *Proceedings of the VLDB Endowment*, vol. 8, no. 3, 2014, pp. 209-220.
[62] K. Ren, et al., "Design principles for scaling multi-core oltp under high contention," in *Proceedings of the 2016 International Conference on Management of Data*, 2016, pp. 1583-1598.
[63] D. A. Yao, et al., "Exploiting single-threaded model in multi-core in-memory systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 10, pp. 2635-2650, 2016.
[64] M. Sadoghi and S. Blanas, "Transaction processing on modern hardware," *Synthesis Lectures on Data Management*, vol. 14, no. 2, pp. 1-138, 2019.
[65] K. Ren, et al., "Design principles for scaling multi-core oltp under high contention," in *Proceedings of the 2016 International Conference on Management of Data*, 2016, pp. 1583-1598.
[66] M. R. Shamis, et al., "Fast general distributed transactions with opacity using global time," *arXiv preprint arXiv: 2006.14346*, 2020.
[67] H. T. Kung and J. T. Robinson, "On optimistic methods for concurrency control," *ACM Transactions on Database Systems (TODS)*, vol. 6, no. 2, pp. 213-226, 1981.
[68] J. Guo, et al., "Adaptive optimistic concurrency control for heterogeneous workloads," *Proceedings of the VLDB Endowment*, vol. 12, no. 5, 2019, pp. 584-596,.
[69] H. Berenson, et al., "A critique of ansi sql isolation levels," in *ACM SIGMOD Record*, vol. 24, no. 2, pp. 1-10, 1995.
[70] T. Lahiri, et al., "Oracle timesten: An in-memory database for enterprise applications," *IEEE Data Enginnering Bulletin*, vol. 36, no. 2, pp. 6-13, 2013.
[71] R. Rankins, et al., "Microsoft SQL Server 2012 Un- leashed," *Pearson Education*, 2013.
[72] M. J. Cahill, et al., "Serializable isolation for snapshot databases," *ACM Transactions on Database Systems (TODS)*, vol. 34, no. 4, pp. 729-738, 2009.
[73] M. A. Bornea, et al., "One-copy serializability with snapshot isolation under the hood," in *2011 IEEE 27th International Conference on Data Engineering*, 2011, pp. 625-636.
[74] J. Lindstro¨m, et al., "IBM soliddb: In-memory database optimized for extreme speed and availability," *IEEE Data Engineering Bulletin*, vol. 36, no. 2, pp. 14-20, 2013.
[75] M. Athanassoulis, et al., "Designing access methods: The rum conjecture," in *EDBT*, vol. 2016, pp. 461-466, 2016.
[76] H. Zhang, et al., "Order-preserving key compression for in-memory search trees," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 1601-1615.
[77] G. Graefe, "Sorting and indexing with partitioned b-trees," in *CIDR*, vol. 3, pp. 5-8, 2003.
[78] H. Jagadish, et al., "Incremental organization for data recording and warehousing," in *VLDB*, vol. 97, pp. 16-25, 1997.
[79] T. Pelkonen, et al., "Gorilla: A fast, scalable, in-memory time series database," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, 2015, pp. 1816-1827.
[80] P. Flajolet, et al., "Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm," in *Discrete Mathematics and Theoretical Computer Science*, pp. 137-156, 2007.
[81] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58-75, 2005.
[82] P. Francisco, et al., "The netezza data appliance architecture: A platform for high performance data warehousing and analytics," IBM Corp., 2011.
[83] L. Sidirourgos and M. Kersten, "Column imprints: a secondary index structure," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 2013, pp. 893-904.
[84] M. Athanassoulis and S. Idreos, "Design tradeoffs of data access methods," in *Proceedings of the 2016 International Conference on Management of Data*, 2016, pp. 2195-2200.
[85] R. Sherkat, et al., "Native store extension for sap hana," *Proceedings of the VLDB Endowment*, vol. 12, no. 12, pp. 2047-2058, 2019.
[86] J. J. Levandoski, et al., "Identifying hot and cold data in main-memory databases," in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, 2013, pp. 26-37.

[87] N. W. Paton and O. D´ıaz, "Active database systems," *ACM Computing Surveys (CSUR)*, vol. 31, no. 1, pp. 63-103, 1999.
[88] S. Dar, et al., "Semantic data caching and replacement," in *VLDB*, vol. 96, pp. 330-341, 1996.
[89] T. H. Chen, et al., "Cacheoptimizer: Helping developers configure caching frameworks for hibernate-based database-centric web applications," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 666-677.
[90] P. Zheng, "Artificial intelligence for understanding large and complex datacenters," Ph.D. dissertation, Duke University, 2020.
[91] M. Boissier, et al., "Improving tuple reconstruction for tiered column stores: a workload-aware ansatz based on table reordering," in *Proceedings of the Australasian Computer Science Week Multiconference*, 2017, p. 25.
[92] M. Kuszera, et al., "Toward rdb to nosql: transforming data with metamor- fose framework," in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, 2019, pp. 456-463.
[93] R Roman Č. and M. Kvet, "Comparison of query performance in relational a non-relation databases," *Transportation Research Procedia*, vol. 40, pp. 170-177, 2019.
[94] Zhang and J. Lu, "Holistic evaluation in multi-model databases benchmarking," *Distributed and Parallel Databases*, pp. 1-33, 2019.
[95] S. Sharma, et al., "A brief review on leading big data models," *Data Science Journal*, pp. 14-41, 2014.
[96] J. P. Coelho, et al., "Htapbench: Hybrid transactional and analytical processing benchmark," in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*, 2017, pp. 293-304.
[97] N. Mukherjee, et al., "Distributed architecture of oracle database in-memory," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, 2015, pp. 1630-1641.
[98] K. Goel, et al., "Towards scalable real-time analytics: an architecture for scale-out of olxp workloads," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, 2015, pp. 1716-1727.
[99] Z. Xie, et al., "Cool, a cohort online analytical processing system," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, 2020, pp. 577-588.
[100] R. Appuswamy, et al., "The case for heterogeneous htap," in *8th Biennial Conference on Innovative Data Systems Research*, 2017.
[101] G. A. Pavlo, et al., "Self-driving database management systems," in *CIDR*, vol. 4, p. 1, 2017.
[102] J. Arulraj, et al., "Bridging the archipelago between row-stores and column-stores for hybrid workloads," in *Proceedings of the 2016 International Conference on Management of Data*. ACM, 2016, pp. 583-598.
[103] R. Barber, et al., "Evolving databases for new-gen big data applications," in CIDR, 2017.
[104] Vohra, "Apache parquet," in *Practical Hadoop Ecosystem*, pp. 325-335, 2016.
[105] M. Sadoghi, et al., "L-store: A real-time oltp and olap system," *arXiv preprint arXiv: 1601.04084*, 2016.
[106] J. F. Im, et al., "Pinot: Realtime olap for 530 million users," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 583-594.
[107] Gpdb.docs.pivotal.io, "Pivotal GreenplumQR 6.3 Documentation - Pivotal Greenplum Docs," 2020.
[108] Niyizamwiyitira and L. Lundberg, "Performance evaluation of sql and nosql database management systems in a cluster," *International Journal of Database Management Systems*, vol. 9, no. 6, pp. 1-24, 2017.
[109] N. Bhatti, et al., "Integrating user-perceived quality into web server design," *Computer Networks*, vol. 33, pp. 1-16, 2000.
[110] S. B. Yao and A. R. Hevner, "A guide to performance evaluation of database systems," *NBS Special Publication*, 1984.
[111] Olken and D. Rotem, "Random sampling from databases: a survey," *Statistics and Computing*, vol. 5, no. 1, pp. 25-42, 1995.
[112] R. K. Kurella, "Systematic literature review: Cost estimation in relational databases," Ph.D. dissertation, University of Magdeburg, 2018.
[113] Y. Theodoridis, et al., "Efficient cost models for spatial queries using r-trees," *IEEE Transactions on knowledge and data engineering*, vol. 12, no. 1, pp. 19-32, 2000.

BIOGRAPHIES OF AUTHORS

Sucharitha Shetty is currently working as Assistant Professor in the Department of Computer Sci- ence and Engineering at MIT, Manipal Academy of Higher Education, Manipal. She is also pursuing her PhD at Manipal Academy of Higher Education, Manipal. Her research interests include database and parallelism. She has published few papers in International conferences.

B. Dinesh Rao acquired his MTech degree from the Indian Institute of Technology, Kanpur in the year 1993 and PhD degree from the Manipal Academy of Higher Education, Manipal in the year 2014. He is currently working as Professor in Manipal School of Information Sciences, Manipal Academy of Higher Education, Manipal. His research interests include Fractals and theoretical Computer Sci- ence. He has around 26 years of teaching experience and has published around 11 research papers in National/International Journals/Conferences.

Srikanth Prabhu received his M.Tech degree and PhD degree from the Indian Institute of Technology, Kharagpur in the year 2003 and 2013 respectively. He is presently working as Associate Professor in the Department of Computer Science and Engineering, Manipal Institute of Technol- ogy, Manipal Academy of Higher Education, Manipal. His research interests include Biomedical Image processing, Agricultural image processing, Face recognition, wireless visual sensor networks. He has around 15 years of teaching experience and has published around 35 research papers in Na- tional/International Journals/Conferences. He is also on the editorial board of the some journals.