I L L I N O I S

UNIVERISTY OF ILLINOIS AT URBANA-CHAMPAIGN

Department of Computer Science

Team 2: NullDb

CS 511

Survey Report

Nov 2021

Contents

Chapter 1

Problem Taxonomy

1.1 Problem Statement

The problem necessitates us to perform two broad tasks. First, we need to create a dashboard which monitors real time data and refreshes its queries on demand. Second, we need to support a set of functions which can efficiently display the data on the dashboard in an intuitive manner via graphical tools and widgets. The specific scenario is marketer, where the user should perform market analysis on a very large database of product reviews to make better business decisions.

In a rudimentary model, new updates are handled by naively re-querying the entire database, filtering the resultant tuples, applying appropriate aggregate functions and displaying the results in the dashboard. Since continuously re-querying large databases is prohibitively expensive, our survey aims to find literature on similar problems and solutions to optimize the pipeline for a reduced data-to-query lag.

1.2 Taxonomy

It is important to break down the problem into subtopics that deserve to be focused on individually simply because of their potential in contributing to the overall performance. First, we need to develop an intuitive and interactive data interface to display significant amounts of aggregate data. However, in order to minimize data-to-query lag, further optimizations are necessary. Indexing and query processing are critical components in query optimization. Each use case is different, so it is necessary to determine the optimal indexing and execution plan for our specific scenario. Finally, as memory technology develops, it is becoming increasingly possible to cache and store databases entirely in main memory. Although it is impossible to store large databases in main memory, bringing relevant data into cache will likely result in significant performance benefits.

Therefore, we can classify the taxonomy as follows:

- Indexing
- Query Processing
- Caching
- Data Interface

Splitting the taxonomy into separate sections lets us explore the different problem parts concurrently and makes the entire project pipeline faster. Each independent research area has a significant a volume of literature survey and techniques which address a variety of problems. It is important find relevant work in each sub-domain and determine which approaches are complementary/improve the overall performance of the project.

1.2.1 Indexing

A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional writes and storage space to maintain the index data structure. Indexes are used to quickly locate data without having to search every row in a database table every time a database table is accessed. Indexes can be created using one or more columns of a database table, providing the basis for both rapid random look-ups and efficient access of ordered records[16].

Dashboard updates heavily depend on frequent concurrent database accesses. A well-formed index can significantly reduce the number of excess I/O operations, thereby resulting in a quicker search and retrieval of data to users. In general, indexes improve performance in SELECT queries while slowing down INSERT, UPDATE, and DELETE operations.[7]

Current literature explores multiple different indexing techniques, each with its own benefits and trade-offs. The dashboard updates depend heavily on frequent reads into the database; therefore, we optimize specifically for database accesses. The additional costs in INSERT and UPDATE queries are of minimal importance.

1.2.2 Query Processing

Query processing is the process which transforms a high-level query into an equivalent, correct, and efficient execution plan expressed in a low-level language. It typically involves interpreting the query, searching through the space storing data, and retrieving corresponding results. Query processing is responsible for transforming the query into a set of efficient queries by reducing redundant calls to the database, minimizing unnecessary joins, and determining an optimal query path.[35] It is important that the correctness of queries is not compromised when we try to optimize query paths. Traditionally, the step includes dynamic programming heuristics or a brute force searches.

Since the data is relatively large, it is important to optimize queries to eliminate redundant calls to the database. In large-scale database systems, it is extremely prohibitive to continuously execute complex queries without regards for performance. Our workload consists of relatively simple aggregate queries without complex joins. It continuously computes a pre-defined set of queries. Therefore, most of our research is oriented around delta queries and incremental view maintenance instead of lower-level details such as the parser, optimizer, and execution engine.

1.2.3 Caching

Caching helps us make better use of resources by taking advantage of locality of reference principle and also use the vast in-memory volume available in modern systems. Caches are used in every layer of computing - operating systems, web browsers, web applications, database etc. There are three different kinds of caches - write back cache, write through cache and write around cache. A database cache supplements the primary database by removing unnecessary pressure, typically in the form of frequently accessed read data.

Dashboard updates heavily depend on frequent concurrent database accesses. Since memory is orders of magnitude faster than disk, caches can help significantly reduce read-access queries. However, due to the nature and scale of the data, it is difficult to determine an effective caching solution while maintaining data integrity. We aim to build our model in a way to also support distributed caching in the future for big-data models[38] for scalability purposes. Caching enables us access data faster, which in turn increases query efficiency.

1.2.4 Data Interface

Data interfaces provide visibility in the data by aggregating and extracting values from the data warehouse. It provides a comprehensive overview of snapshots in time, allowing users to identify key metrics and trends relevant to the information in question. Data interfaces typically consist of a collection of graphical widgets. Widgets can either be refreshed automatically or manually. User interactivity facilitates better user experience [34]; therefore it is important to research not only effective ways to display data but also methods of interaction. A data interface should effectively represent the underlying data while minimizing the number of external database queries. Since new data is constantly incoming, the representation needs to be user-friendly and legible to take advantage of the aforementioned optimizations.

1.3 Existing Work

The individual entities in the problem taxonomy have been studied extensively. We initially used survey papers[16][35][38][4] to review the existing work and respective target areas within these fields. In accordance to our problem scenario, we mainly surveyed papers in the big-data domain. We aim to combine these entities to address our use case while ensuring the individual optimizations complement each other.

There is limited work detailing the combination of multiple areas. Current research aims to hyper-optimize each individual section, tailoring it for a specific use case. We aim to propose an end-to-end optimized pipeline for analyzing large-scale aggregate data. We take an extra effort to optimize our pipeline for marketers, the primary case of interest in our work.

Chapter 2

Technique Taxonomy

2.1 Challenges

2.1.1 Problem Challenges

The key challenges posed by our problem statement revolve around having a method for efficiently updating of dashboards without re-querying the entire database. We also need to alert the widgets about the update after each dashboard refresh and display the data. The data is streamed in real time so such updates need to minimally be faster than the input rate to prevent queuing. Otherwise the system may suffer from cascading failures due to rate overflow and lack realtime-ness.

2.1.2 Literature Survey

Our first key challenge is to identify the existing state of the art techniques and work done in the above mentioned fields. This involves benchmarking databases and selecting one which best suits our use-case. We used YCSB[11] and tested different databases on a variety of workloads. This paper gives performance benchmarks for various databases on simulated workloads. It also gives an interface to extend for newer databases and gives us the option to test them on custom workloads. This makes it a standard when doing database benchmarking. Since, our use case is read-heavy, we primarily focused on database query performance.

We also studied survey papers for indexing[16], query processing[35], caching[38], and data visualization[4] to understand existing literature and techniques. The papers cover important literature in the field and help further group the topics into sub-fields. The papers provide a general idea of the existing work and the future research direction in these fields.

2.1.3 Finding Novel Ideas

We did not want to solely replicate existing state of the art ideas. We also wanted contribute a novel idea of our own. This is one of the most challenging aspects of the project since developing and testing novel ideas require a lot of thought and experiments to confirm our intuition. We broadly identified two fields, namely query processing and caching to implement new ideas. We used the DbToaster[22] and FlashStore[12] paper as our primary reference. The papers match our use-case and provide significant performance improvement on the base model.

2.1.4 Integrating Sub-systems

It is important to integrate the various subsystems without sacrificing performance. The benefits of indexing, caching and fast query processing should all be complementary to the overall goal of the project. We need to ensure the access patterns of our in-memory caching is in line with the database accesses and that we index on the appropriate columns. Our queries then need to take advantage of our indexes cache. Without complimentary subsystems, the benefits of each sub-domain may not be fully realized. Hence, is is an important engineering challenge.

2.1.5 Technical Challenges

We also faced minor technical challenges to ensure a common environment for all our team members. With each one of us using a different operating systems(eg: Debian Linux, Mac and Windows), it was important for us to address this problem early. We used Docker[13] to this purpose. Since none of us have had experience with advanced database techniques, there was a significant learning curve.

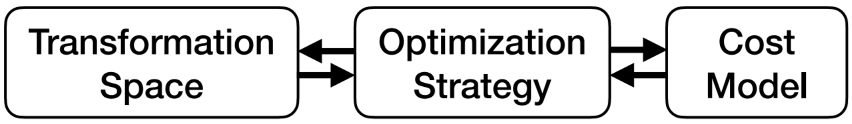
2.2 Technique Taxonomy

The following section contains the techniques we used for addressing the challenges posed.

2.2.1 Indexing

An interactive dashboard is measured by the performance of indexing and query processing techniques under the hood. For indexing, we explored in-house techniques offered by MySQL[27]. MySQL indexing scheme for both primary and secondary indexes work quite well for our use-case and we did not feel the need to develop something to replace this. We also evaluated other databases but for read heavy queries MySQL offered the best performance.

We evaluated other indexing techniques like Learned Indexing[23] which uses machine learning to reduce the search space of possibilities and find the appropriate index. Learning techniques help prune through the search space and assign each data point in this space with a cost using the cost model. The optimization strategy is then responsible for choosing the best data point that can represent the best index. In a way, it uses heuristics without needing any domain knowledge. The domain knowledge in this case is presented by the training set. The following image shows the interaction between the various components in the learning model.



We also explored Adaptive Data Series Indexing[44], which uses a strategic style of indexing to reduce the data-to-query gap. Instead of building the complete index over the complete data up-front and querying only later, we interactively and adaptively build the parts of the index tree necessary for the query. Although the approach beats the state-of-the-art indexing techniques, it is unsuitable for our target scenario. We query across the entire database, not a specific subsection, and thus adaptive indexing provide little to no tangible benefits. We decided to stick with MySQL indexing techniques and focus on other areas for performance.

Inverted indexing technique[8] and other techniques presented in the paper by Lehman et al.[24] largely covered indexing structures for main memory databases. These group of ideas are borrowed from the field of information retrieval and improve performance when we look to perform operations which retrieve specific information from a text document. However, these techniques are relatively complex and have pre-existing hardware requirements. When compared to MySQL in-house indexing techniques, the performance benefits are not significant enough to justify investment.

2.2.2 Delta Queries

Many existing works build upon the concept of incremental view maintenance. In incremental view maintenance, materialized views compute and apply incremental changes to stay up to date instead of recomputing the contents from the base relation. We focused heavily on delta queries[31], a technique to compute the incremental change and in turn ensure efficient dashboard updates. Griffin et al.[18] explains the base idea behind delta queries.

We used DbToaster[22] and the follow up paper[3] as the primary reference for the part of the problem. The papers propose high performance query executors which parse original queries into a query that hits a minimal set of tuples in the database. These are then used to update in-memory views which help supply the result of the original query. In terms of target applications, it aims to help high volume data streams that have frequent updates and need fast and low latency read queries. Performance is also improved by doing computation in main memory. In our case, each dashboard query is transformed into a smaller query which only queries against a smaller subset of pre-computed results. The idea reduces the search space of the query, thereby resulting in a quicker response time.

The ideas of change data capture and delta queries for the purposes of incremental view maintenance have been adopted by other existing works such as CoNoSQLDBs[20][21], object-oriented databases[5] and in data warehouse environment[6]. This gives us the confidence to apply delta queries to solve one area of our problem. In the rest of the report, we explore additional areas where we can extract faster query results.

We also looked at incremental deferred view maintenance[10]. The paper introduces the concept of auxiliary tables and how they are maintained in memory to support deferred refreshes of data interface. The key takeaway is how the auxiliary tables support delta query processing in-memory. Since we aim to support dashboard interactivity, we allow users to update database in multiple different manners. For multiple insertions, we look to He et al.[19].

A similar paper that we explored is in-situ query processing ideas presented by Olma et al.[29]. The core idea here is about query processing using a combination of data partitioning and indexing techniques. The in-situ query engine proposed in this paper builds indexes on the fly for different partitions of data. The idea is a little overkill for our specific use case, especially due to the lack of sharding. However, the paper facilitated further exploration of in-situ techniques, eventually leading to caching techniques.

2.2.3 In memory caching

We store auxiliary tables which contain data for the aggregate widgets on the dashboard. It is thus important to explore techniques which enable us to cache the data for further access performance. Zhou et al.[43] argue for lazy persistent updates on the views and caching the data in-memory. It uses a way to combine writes from several transactions, which are pushed onto the main database only when a read query is made on the one of the tuples updated in the write queries. While it works well in practice for read-write queries, it is not as relevant in our scenario as multiple reads occur immediately after a write.

Since we are storing a minimal set of key-value pairs in our auxiliary tables, we can migrate it to caches inside memory. We explored the FlashStore[12][32], BloomStore - an optimization on FlashStore[25] Memc3[15] paper in this regard, all of which focus on caching models. Our eventual implementation is derived from the FlashStore[12] paper. FlashStore exploits the methods of building an in-memory caching model using a custom hash table. The caching model proposed is persistent and is built to handle a high throughput. The hashing scheme proposed is also built to handle concurrency using a key-wise locking scheme.

In terms of the higher level model, it is a write-back cache. Data is written onto the cache and transferred to the database only when cache line is evicted. The model works best for read heavy workloads. Another option is a write-around cache, which provides good write performance but has a slower read performance as every new read is a cache miss. It is completely the opposite of our target scenario. The last option is write-through cache, which ensures consistency and is tolerant to failures. However, each operation being performed twice thus slowing down performance. Therefore, we designed our cache with many of the techniques from FlashStore in mind.

For the purpose of scaling our model for larger workloads, we also explored distributed caching models[39]. Such models help mitigate the issue of limited caching memory. Such models are useful when the size of the data exceeds the hardware limits on a single phycial machine. In such cases, the cache itself is distributed across machines with the data mapped onto it. This idea forms the core of data sharding. We also explored modern databases such as Oracle[33] to compare how these techniques are implemented. The paper explains an end-to-end implementation of a database which uses indexing, sharding, query processing and execution. This was important to read as it helped us validate some of our intuitions.

2.2.4 Data Interface

The direct representation of relations on screen is significantly easier to navigate and presents a lower learning curve when compared to other systems. Graphical representations allow users to quickly navigate and compare different sources of data, proving especially helpful when there are multiple relationships between objects/scenarios.[34][41] Therefore it is necessary to develop interactive visual widgets rather than text-based systems in our user interface.

With regards to display, the challenges are relatively design-oriented. Data visualization and aggregation is an extremely well-researched topic, and for the purposes of our representations, current techniques are sufficient. We surveyed many papers regarding real-time data visualization techniques[36][37] to understand the key features in modern dashboards. Relevant features are then aggregated to succinctly display the data in an informative and aesthetically pleasing manner.

Real-time dashboards can update either automatically or manually. In automatic updates, the dashboard either refreshes the queries at preset intervals or only when new information is available. There is a heavy load on the backend server/database, which can result in scalability issues. In manual updates, users need to interact with the visualization (for example clicking the refresh button or modifying the query parameters) to update the corresponding visualizations. The technique is significantly more scalable, but suffers from realtime-ness.[37]

Dynamic display introduces the potential for three cognitive issues of display interpretation: an inability to recognize updates in the display, an inability to locate updated elements, and an inability to determine the difference between old/updated elements. The issue is significantly more pronounced during display interruptions (ie page refresh). The effects can be minimized by minimizing visual interruptions, incorporating animations between states, and incorporating appropriate visual emphasis for significant new elements.[40]

In our scenario, the aggregate queries are largely of top-k style. Interpretation of the data is more intuitive with bar graphs. However, we want to be able to refresh only those parts of the interface updated in the database. We referred to the concepts of Novel Interface for Group Collaboration[14], determining the React UI framework best implements the techniques of the paper. The key idea is to have a multimodal layout[9] where individual modals can be refreshed without reloading the entire page.

In some applications, due to the large nature of the data, it can be difficult to effectively represent the data without overloading the end user. Data reduction methods such as filtering, sampling, and aggregation can address perceptual scalability. Panning, zooming, and brushing/linking can address interactive scalability. We reduce the data through binned aggregation and provide toggles to modify the granularity of results.[17]

2.3 Justification

We separated our techniques in line with the problem taxonomy: indexing, query processing, caching and data interface. This helps us focus on the individual problems independently, thereby simplifying the design of the entire system. Decoupling the system also helps in debugging as we can now identify the exact cause of a fault or slow performance by separating the systems and testing each component individually or performing an ablation study to locate faults.

2.4 Comparison with existing techniques

2.4.1 Challenges

Plenty of existing techniques offer significant performance benefits in their respective fields. However, it is important to select the appropriate techniques for our specific use case and build upon it with novel ideas to adapt to our problem setting. Only then will integrating the individuals sub-systems converge to a singular application that best solves our problem. Reading papers which present the entire database architecture like by Oracle[33] helps us understand how multiple components interact with each other to maximize performance. Our technique combines the best of each individual entity in our taxonomy and achieves performance that significantly outperforms the baseline vanilla model.

2.4.2 Future Work

While our model performs quite well, it is open to further improvement. Firstly, we can improve on the current in-house indexing scheme and build a new indexing scheme by basing it on pre-existing hashing techniques such as cuckoo hashing[30]. Cuckoo hashing is a newer hashing scheme that significantly improves performance by reducing collisions in a cache.

We can also try to scale out this model to handle concurrency on a larger scale. We would need to handle multiple threads simultaneously accessing data, thus require additional locking techniques. We can use in house techniques like the use of compare and set hardware locks or mutexes to implement our own locking protocol. Few techniques that are already available in the literature include escrow locking[28] and the database locking protocols proposed by Molesky et al..[26]. Escrow locking is also used by many online retail websites and is worth exploring if we want to explore concurrency control mechanisms.

Further, we can look at ways to solve this problem for nested aggregate queries, also called non-monotonic queries. The mini-batch execution model of G-OLA[42] and approximate query-answering[1] work well for these kind of queries. The approximation can be delivered in terms of error estimates[2] computed over a sample picked by bootstrap technique. It is especially suitable for human-driven interactive data-analysis where the human does not know the accuracy a priori.

References

[1] Swarup Acharya et al. “The Aqua Approximate Query Answering System”. In: *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’99. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1999, pp. 574–576. ISBN: 1581130848. DOI: 10.1145/304182.304581. URL: https://doi.org/10.1145/304182.304581.

[2] Sameer Agarwal et al. “Knowing When You’re Wrong: Building Fast and Reliable Approximate Query Processing Systems”. In: *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’14. Snowbird, Utah, USA: Association for Computing Machinery, 2014, pp. 481–492. ISBN: 9781450323765. DOI: 10.1145/2588555.2593667. URL: https://doi.org/10.1145/2588555.2593667.

[3] Yanif Ahmad and Christoph Koch. “DBToaster: A SQL Compiler for High-Performance Delta Processing in Main-Memory Databases”. In: *Proc. VLDB Endow.* 2.2 (Aug. 2009), pp. 1566–1569. ISSN: 2150-8097. DOI: 10.14778/1687553.1687592. URL: https://doi.org/10.14778/1687553.1687592.

[4] Wolfgang Aigner et al. “Survey of Visualization Techniques”. In: London: Springer London, 2011, pp. 147–254.

[5] Reda Alhajj and Faruk Polat. “Incremental View Maintenance in Object-Oriented Databases”. In: *SIGMIS Database* 29.3 (June 1998), pp. 52–64. ISSN: 0095-0033. DOI: 10.1145/313310.313338. URL: https://doi.org/10.1145/313310.313338.

[6] Abdulaziz Almazyad and Mohammad Khubeb Siddiqui. “Incremental View Maintenance: An Algorithmic Approach”. In: *International Journal of Electrical & Computer Sciences IJECS-IJENS* Vol: 10 (Jan. 2010), p. 16.

[7] Arnab Bhattacharya. *Fundamentals of Database Indexing and Searching*. 1st. Chapman & Hall/CRC, 2014. ISBN: 1466582545.

[8] B. Barla Cambazoglu et al. “A Term-Based Inverted Index Partitioning Model for Efficient Distributed Query Processing”. In: *ACM Trans. Web* 7.3 (Sept. 2013). ISSN: 1559-1131. DOI: 10.1145/2516633.2516637. URL: https://doi.org/10.1145/2516633.2516637.

[9] Philip Cohen et al. “QuickSet: A Multimodal Interface for Distributed Interactive Simulation”. In: (Aug. 2003).

[10] Latha S. Colby et al. “Algorithms for Deferred View Maintenance”. In: *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data.* SIGMOD ’96. Montreal, Quebec, Canada: Association for Computing Machinery, 1996, pp. 469–480. ISBN: 0897917944. DOI: 10.1145/233269.233364. URL: https://doi.org/10.1145/233269.233364.

[11] Brian F. Cooper et al. “Benchmarking Cloud Serving Systems with YCSB”. In: *Proceedings of the 1st ACM Symposium on Cloud Computing.* SoCC ’10. Indianapolis, Indiana, USA: Association for Computing Machinery, 2010, pp. 143–154. ISBN: 9781450300360. DOI: 10.1145/1807128.1807152. URL: https://doi.org/10.1145/1807128.1807152.

[12] Biplob Debnath, Sudipta Sengupta, and Jin Li. “FlashStore: High Throughput Persistent Key-Value Store”. In: *Proc. VLDB Endow.* 3.1–2 (Sept. 2010), pp. 1414–1425. ISSN: 2150-8097. DOI: 10.14778/1920841.1921015. URL: https://doi.org/10.14778/1920841.1921015.

[13] *Docker.* URL: https://docs.docker.com/.

[14] Bogdan Dorohonceanu, Boi Sletterink, and Ivan Marsic. “A Novel User Interface for Group Collaboration”. In: (Oct. 1999).

[15] Bin Fan, David G Andersen, and Michael Kaminsky. “MemC3: Compact and Concurrent MemCache with Dumber Caching and Smarter Hashing”. In: *10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13).* 2013, pp. 371–384.

[16] Abdullah Gani et al. “A Survey on Indexing Techniques for Big Data: Taxonomy and Performance Evaluation”. In: *Knowl. Inf. Syst.* 46.2 (Feb. 2016), pp. 241–284. ISSN: 0219-1377. DOI: 10.1007/s10115-015-0830-y. URL: https://doi.org/10.1007/s10115-015-0830-y.

[17] Parke Godfrey, Jarek Gryz, and Piotr Lasek. “Interactive Visualization of Large Data Sets”. In: *IEEE Transactions on Knowledge and Data Engineering* 28.8 (2016), pp. 2142–2157. DOI: 10.1109/TKDE.2016.2557324.

[18] Timothy Griffin and Leonid Libkin. “Incremental Maintenance of Views with Duplicates”. In: *SIGMOD Rec.* 24.2 (May 1995), pp. 328–339. ISSN: 0163-5808. DOI: 10.1145/568271.223849. URL: https://doi.org/10.1145/568271.223849.

[19] Hao He et al. “Asymmetric Batch Incremental View Maintenance”. In: *Proceedings of the 21st International Conference on Data Engineering.* ICDE ’05. USA: IEEE Computer Society, 2005, pp. 106–117. ISBN: 0769522858. DOI: 10.1109/ICDE.2005.22. URL: https://doi.org/10.1109/ICDE.2005.22.

[20] Yong Hu and Stefan Dessloch. “Extracting Deltas from Column Oriented NoSQL Databases for Different Incremental Applications and Diverse Data Targets”. In: *Data Knowl. Eng.* 93.C (Sept. 2014), pp. 42–59. ISSN: 0169-023X. DOI: 10.1016/j.datak.2014.07.002. URL: https://doi.org/10.1016/j.datak.2014.07.002.

[21] Yong Hu and Weiping Qu. “Efficiently Extracting Change Data from Column Oriented NoSQL Databases”. In: 21 (Jan. 2013), pp. 587–598. DOI: 10.1007/978-3-642-35473-1_58.

[22] Christoph Koch et al. “DBToaster: Higher-order Delta Processing for Dynamic, Frequently Fresh Views”. In: *The VLDB Journal* 23.2 (2014), pp. 253–278.

[23] Tim Kraska et al. “The Case for Learned Index Structures”. In: *Proceedings of the 2018 International Conference on Management of Data*. SIGMOD ’18. Houston, TX, USA: Association for Computing Machinery, 2018, pp. 489–504. ISBN: 9781450347037. DOI: 10.1145/3183713.3196909. URL: https://doi.org/10.1145/3183713.3196909.

[24] Tobin J Lehman and Michael J Carey. *A Study of Index Structures for a Main Memory Database Management System*. Tech. rep. University of Wisconsin-Madison Department of Computer Sciences, 1985.

[25] Guanlin Lu, Young Jin Nam, and David H. C. Du. “BloomStore: Bloom-Filter based memory-efficient key-value store for indexing of data deduplication on flash”. In: *2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*. 2012, pp. 1–11. DOI: 10.1109/MSST.2012.6232390.

[26] Lory D Molesky and Krithi Ramamritham. “Database Locking Protocols for Large-Scale Cache-Coherent Shared Memory Multiprocessors: Design, Implementation and Performance”. In: *Jun* 6 (1995), pp. 1–24.

[27] *MySQL*. URL: https://dev.mysql.com/.

[28] Patrick E O’Neil. “The Escrow Transactional Method”. In: *ACM Transactions on Database Systems (TODS)* 11.4 (1986), pp. 405–430.

[29] Matthaios Olma et al. “Adaptive partitioning and indexing for in situ query processing”. In: *The VLDB Journal* 29.1 (2020), pp. 569–591.

[30] Rasmus Pagh and Flemming Friche Rodler. “Cuckoo hashing”. In: *Journal of Algorithms* 51.2 (2004), pp. 122–144.

[31] Themistoklis Palpanas et al. “Incremental Maintenance for Non-Distributive Aggregate Functions”. In: *Proceedings of the 28th International Conference on Very Large Data Bases*. VLDB ’02. Hong Kong, China: VLDB Endowment, 2002, pp. 802–813.

[32] Anastasios Papagiannis et al. “An Efficient Memory-Mapped Key-Value Store for Flash Storage”. In: *Proceedings of the ACM Symposium on Cloud Computing*. SoCC ’18. Carlsbad, CA, USA: Association for Computing Machinery, 2018, pp. 490–502. ISBN: 9781450360111. DOI: 10.1145/3267809.3267824. URL: https://doi.org/10.1145/3267809.3267824.

[33] Sukhada Pendse et al. “Oracle Database In-Memory on Active Data Guard: Real-time Analytics on a Standby Database”. In: *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE. 2020, pp. 1570–1578.

[34] Shneiderman. “Direct Manipulation: A Step Beyond Programming Languages”. In: *Computer* 16.8 (1983), pp. 57–69. DOI: 10.1109/MC.1983.1654471.

[35] Hari Singh and Seema Bawa. “A Survey of Traditional and MapReduce Based Spatial Query Processing Approaches”. In: *SIGMOD Rec.* 46.2 (Sept. 2017), pp. 18–29. ISSN: 0163-5808. DOI: 10.1145/3137586.3137590. URL: https://doi.org/10.1145/3137586.3137590.

[36] Jennifer G Stadler et al. “Improving the Efficiency and Ease of Healthcare Analysis Through Use of Data Visualization Dashboards”. In: *Big data* 4.2 (2016), pp. 129–135.

[37] Samuel Stehle and Rob Kitchin. “Real-time and archival data visualisation techniques in city dashboards”. In: *International Journal of Geographical Information Science* 34 (June 2019), pp. 1–23. DOI: 10.1080/13658816.2019.1594823.

[38] Shakil Tamboli and Smita Shukla Patel. “A survey on innovative approach for improvement in efficiency of caching technique for big data application”. In: *2015 International Conference on Pervasive Computing (ICPC)*. 2015, pp. 1–6. DOI: 10.1109/PERVASIVE.2015.7087016.

[39] Oliver Theel and Markus Pizka. “Distributed Caching and Replication”. In: *Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences. 1999. HICSS-32. Abstracts and CD-ROM of Full Papers*. IEEE. 1999, pp. 1–2.

[40] Peyman Toreini and Stefan Morana. “Designing Attention-Aware Business Intelligence and Analytics Dashboards”. In: May 2017.

[41] Allison Woodruff et al. “DataSplash: A Direct Manipulation Environment for Programming Semantic Zoom Visualizations of Tabular Data”. In: *J. Vis. Lang. Comput.* 12 (Oct. 2001), pp. 551–571. DOI: 10.1006/jvlc.2001.0219.

[42] Kai Zeng et al. “G-OLA: Generalized On-Line Aggregation for Interactive Analysis on Big Data”. In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’15. Melbourne, Victoria, Australia: Association for Computing Machinery, 2015, pp. 913–918. ISBN: 9781450327589. DOI: 10.1145/2723372.2735381. URL: https://doi.org/10.1145/2723372.2735381.

[43] Jingren Zhou, Per-Ake Larson, and Hicham G Elmongui. “Lazy Maintenance of Materialized Views”. In: *Proceedings of the 33rd international conference on Very large data bases*. 2007, pp. 231–242.

[44] Kostas Zoumpatianos, Stratos Idreos, and Themis Palpanas. “Indexing for Interactive Exploration of Big Data Series”. In: *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’14. Snowbird, Utah, USA: Association for Computing Machinery, 2014, pp. 1555–1566. ISBN: 9781450323765. DOI: 10.1145/2588555.2610498. URL: https://doi.org/10.1145/2588555.2610498.