Hybrid Transactional and Analytical Processing Databases - State of Research and Production Usage

Daniel Hieber
Dept. of Computer Science
Aalen University
Aalen, Germany
Email: daniel.hieber@studmail.htw-aalen.de

Gregor Grambow
Dept. of Computer Science
Aalen University
Aalen, Germany
Email: gregor.grambow@hs-aalen.de

***Abstract*—The combination of Online Transactional Processing and Online Analytical Processing into one system is an emerging area in database research called Hybrid Transactional and Analytical Processing databases (HTAP, OLxP). Both Gartner and Forrester Research see disruptive potential in this technology as it provides important advantages. These include the elimination of redundant data sets for analytical and live data as well as the reduction of the total cost of ownership of analytics systems. The development of HTAP databases resulted in various advances in the database sector like the creation of new index and data structures or improvements of existing concurrency control implementations. However, there is a great variety regarding many architectural aspects in different HTAP systems. Examples include implementations of concurrency control, query handling, or scaling paradigms ranging from scaled-up single server systems using Multi Version Concurrency Control to scaled-out cluster based systems using last writer wins approaches. This contribution provides a general overview of contemporary HTAP implementations. On the one hand, different fundamental technical aspects are presented and compared in detail. On the other, it goes beyond a standard literature review by also presenting an overview of production ready HTAP systems, including both free and commercial systems.**

Keywords*—*Hybrid Transactional Analytical Processing; HTAP; Database; Literature Study; OLxP.

I. Introduction

The need to analyse data in realtime and not to rely on copies of old databases combined with the growing wish of companies to gather all data in one database lead to the rise of Hybrid Transactional Analytical Processing (HTAP) Databases. In our initial systematic literature review from 2020 on this topic, we provided a comprehensive summary focused on the research of these systems [1]. While HTAP as a term was coined by Gartner [2] in 2014 and even before that there had already been active research in the area these databases are still heavily evolving and only starting to get a foothold in production systems. Therefore, in this work, we extend our previous literature review with the current state of HTAP, highlighting new research conducted, but also introducing an overview of currently available HTAP systems for use in production use cases, including both free and commercial systems.

Solving the problems of keeping data in two separated databases and at the same time reducing the total cost of ownership by introducing one unified system instead, HTAP efficiently combines Online Transactional Processing and Online Analytical Processing capabilities in one system. Therefore, both Gartner [3] and Forrester Research [4] see disruptive potential in HTAP. A trend already projected to the industry, e.g., with commercial solutions provided by two of the world leaders SAPs HANA database [5] and Tableaus HyPer [6] integration.

In this work the basics of the different fundamental architectures for HTAP database systems like HyPer [7] and SAP HANA [8] are explained and different approaches regarding the concrete implementations as well as optimization approaches are introduced in form of a refined version of our earlier systematic literature review. Further we provide insight into the production implementation of HTAP databases like SAPs HANA [5] and Tableaus HyPer [6] systems. The aim of this paper is to reflect the current state of research in an ordered way, as well as to highlight important decisions leading to todays implementations and their production use.

This remainder of this paper is organized as follows: Section 2 provides background on database processing paradigms covered in this paper. Section 3 describes the underlying literature review process in detail. Section 4 discusses the findings and provides a overview of the current development and research state of HTAP.

The section is further separated into subsection ordering findings by the area of HTAP it deals with:

- IV-A Fundamental Architecture (containing scaling paradigms, data/table structures and ways of saving/partitioning data)
- IV-B Concurrency
- IV-C Garbage Collection
- IV-D Query Handling (containing query languages, general optimization approaches, query processing in differently scaling systems)
- IV-E Indices
- IV-F Big Data on HTAP
- IV-G Recovery, Error Handling and Logging
- IV-H Benchmarking of HTAP Systems
- IV-I Stream Processing with HTAP Systems
- IV-J HTAP as a Service
- IV-K Future trends of HTAP development
- IV-L Open Source and Free Versions

Section 5 then provides insights on the current sate of HTAP databases regarding their production usage.

Finally, Section 6 summarizes the provided work, supplying all required information in a short form.

II. Background

This section provides some background information regarding the database processing paradigms covered in this paper.

A. Online Transaction Processing

Online Transaction Processing (OLTP) describes a category of data processing that is focused on transaction-oriented tasks. The workload is heavily write oriented, consisting of insert, update and delete operations. The size of data involved is usually relatively small, while the amount of transactions can be massive.

Features like normalization and ACID are required by OLTP to function efficiently. Besides fast processing and highest availability, data consistency is also one of the most important features of OLTP databases.

B. Online Analytical Processing

Online Analytical Processing (OLAP) is focused on complex queries for dataset analysis. The workload is read heavy and can include enormous datasets. In order to efficiently analyse such big amounts of data, intelligent indexing and fast read times are necessary. OLAP workloads are resource heavy and require high performance systems.

C. Hybrid Transactional Analytical Processing

Hybrid Transactional Analytical Processing (HTAP) combines both OLTP and OLAP in one database. Therefore, writing and analyzing data is efficiently handled in the same database, removing the need to run two separate systems and thereby reducing implementation efforts, maintenance and cost. However, the resource intensive workload of OLAP queries and the required high availability of OLTP compete with each other and require new solutions to work on the same system.

III. Literature Review Methodology

While the term HTAP was first used 8 years ago in 2014, many researchers still did not adopt it and use other phrases like OLxP, HOAP or OLAP and OLTP hybrid databases. To ensure a comprehensive and high-quality literature base for the review, several searches were carried out with different search therms.

In the study, Kitchenham's systematic review procedure [9] was employed. The following steps were pursued:

1) Determining the topic of the research
2) Extraction of the studies from literature considering exclusion and inclusion criteria
3) Evaluation of the quality of the studies
4) Analysis of the data
5) Report of the results

As a topic the current state of HTAP databases was selected, summarizing the research conducted on the topic. To determine the best suited source and search query for the data search of the literature review multiple data sources (including Google Scholar, Semantic Scholar and IEEE) were tested with a multitude search queries.

The reviewing process (Figure 1) was conducted via Google Scholar as this search engine provided the highest quantity and quality of research for the topic. Further the Google Scholar searches included most of the results the other data sources contained. Searches with other search engines and data sources where either lacking a sufficient quantity or quality to conduct a meaningful literature review.

To counter the aforementioned issues regarding the insufficient usage of the therm Hybrid Transaction Analytical Processing databases in the research the literature acquisition was split into three queries using different search structure and therms. While very similar the returned research of each search query was mostly disjunct.

While this approach increased the number of relevant papers found it does not provide all-encompassing literature findings. This is mainly due to skipped keywords in the research papers themselves. E.g., Umbra [10] only describes itself as a further development of HyPer [7] but never mentions HTAP itself in the paper, while its extension [11] actively mentions HTAP. Therefore, the extension was found, while the main paper is not included in this study.

In order to prevent the absence of relevant systems not found by the systematic literature search itself, the sections "Open Source and Free Versions" as well as "Production Ready HTAP Databases" also contain HTAP systems not found by the systematic literature search if they provided relevant free/open source solutions or are used in production systems.

The search was carried out using (1) "htap" "data warehouse" OR "OLTP" "OLAP" (returning 183 entries), (2) HTAP OR OLAP OLTP hybrid database (returning 200 entries) and (3) hybrid transactional analytical processing (returning 200 entries).

In this first search only publications from 2010 to August 2020 were considered. Queries 2 and 3 returned more papers, but were reduced to the 200 most recommended papers, since quality and relevance were continuously decreasing. To provide an up-to-date overview of the current research in this paper slightly refined versions of the search queries were executed again for the time span from 2020 to October 2021 (cf. Figure 1). To provide a comprehensible and reliable literature review only publicly accessible papers or papers available with general institutional access were taken into account. The conducted papers were further reduced to papers using the German or English language. These exclusion criteria left 147 (1), 178 (2) and 179 (3) papers from the first search as well as 70 (1), 7 (2) and 37 (3) papers from the second search to refine further. In this step, all papers already included in the first search were also removed from the findings of the second search.

Following a title and abstract based elimination was conducted. This pruned papers lacking a combination of required key words or only mentioning HTAP as a side note. After this elimination step, 55 (1), 44 (2) and 56 (3) papers (first search)

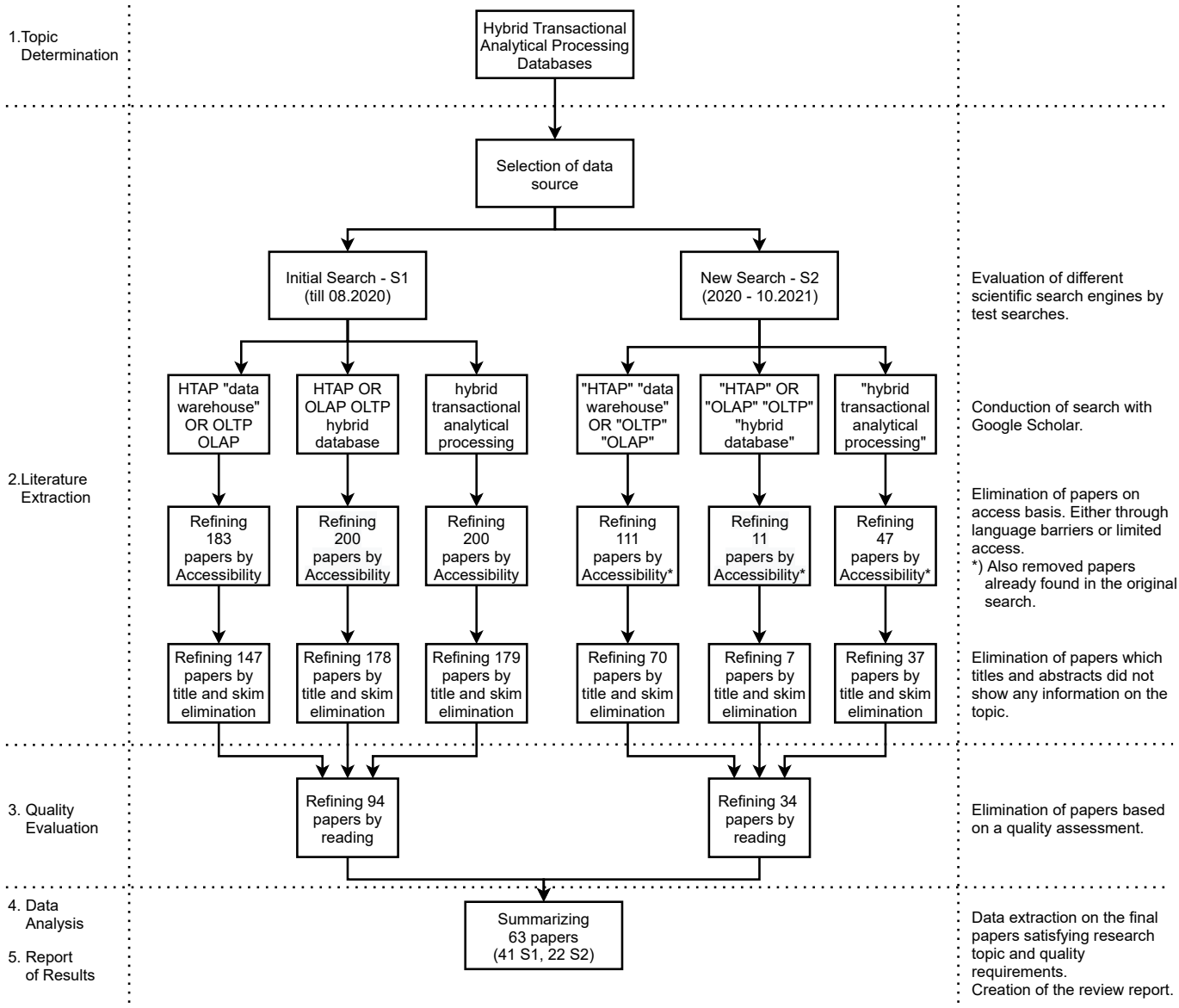

Figure 1. Literature Review Process.

as well as 30 (1), 4 (2) and 20 (3) papers (second search) were left for further analysis. Removing any duplications of papers found by two or more queries this left a total of 128 papers combined for a final review.

Of these 128 papers, 16 were found to be of insufficient quality (lacking evaluations or proper research procedures), and 30 did contrary to their title and abstract not focus on the topic of HTAP databases or on fundamental technologies for those. The 84 papers, which were found scientifically significant and fulfilling the quality requirements were finally reduced to 63, deducting papers providing only outdated non-fundamental information.

IV. Findings and Discussion

The methods to create HTAP databases, their functionality and their optimizations take many different approaches. The contents of the papers were organized into the following sections according to the kind of information provided.

A. Fundamental Architecture

HTAP databases build up a new database sector and there are many databases, which were newly developed for this workload, e.g., [7][12][13]. However, some existing databases also have been upgraded to handle HTAP workloads like SAP HANA [8], initially an OLAP database, and PostgreSQL [14] (as well as multiple systems building on it, cf. [15]), initially an OLTP database, proving that existing databases can be extended to handle HTAP.

Comparing the reviewed database architectures, two main storage paradigmas can be clearly identified with the reviewed solutions: (1) heavily main memory focused databases, keeping all of their (hot) data in memory like HANA [5], HyPer

[7], BatchDB [13] and Hyrise [16], as well as (2) cloud/shared disk data stores, keeping some data in memory but relying on a persistent out of memory data store accessible by all instances, e.g., Wildfire [17], TiDB [18] and Janus [19].

Further, a Non-Uniform Memory Access (NUMA) architecture is a base requirement for most main memory HTAP databases like SAP HANA [5], AIM [20], BatchDB [13], Hyrise [16] and HyPer [7] enabling multiple cores to access each others memory.

1) Scaling out and up: Another big difference in HTAP databases is their scaling approach. Systems like HyPer [7] (commercialized by Tableau), Poseidon [21] or Hyrise [16] are deployed on single servers utilizing NUMA to scale-up onto multiple cores, thus creating multiple nodes. This approach can reduce processing time as no data transfer between different servers is required and all data can be accessed in memory. As a downside however, large systems require a strong server with a large main memory. Both HyPer and Hyrise also provide scale-out approaches, normally keeping their OLTP processing on the main server, e.g., ScyPer [22].

The main memory database Polynesia [23] also follows a single server scale up approach. This happens by separating responsibility of workloads to different OLTP and OLAP "islands". By adding more islands the analytical and transactional throughput can be increased.

Like Hyrise and HyPer - SAP HANA [24] keeps the OLTP workload on one machine, utilizing NUMA to use as many cores as required and available, but implements scaling the OLTP workload out to other servers as a base feature. Using HANA Asynchronous Parallel Table Replication (ATR) the database distributes its data amongst multiple replicas enabling a more efficient OLAP approach.

BatchDB [13] also handles the OLTP workload on the main server. The OLAP workload can be either executed on a different node of the same machine, or an entirely different server.

Contrarily, Wildfire [25][26] (while initially commercialized as IBM DB2 Event Store not all Wildfire decisions seem to apply to Db2 Event Store anymore) utilizes a fully distributed approach. Heavily relying on Apache Spark and Apache Zookeeper, all requests pass Sparks API and get distributed across multiple Spark executors. These executors delegate the transactional and analytical requests to the Wildfire engine daemons. All daemons use their main memory as well as SSDs and are connected to one shared data storage, e.g., a cloud data store. With this approach more throughput can be achieved, but ACID on the other hand is no longer possible. One of the more recent researches on the Wildfire system, Wildfire-Serializable (WiSer) [27] also offers high availability besides HTAP. It is furthermore optimized for IoT workloads.

Like Wildfire, SnappyData [28] also uses Spark as a core component to scale out the system to a database cluster. Therefore, the system enables more information to be kept in memory without the need for one expensive server.

Janus [19] also uses a distributed setup but implements the query distribution on its own with execution servers. These delegate the query to a corresponding row partitioned server for OLTP workloads or a column partitioned server for OLAP workloads.

TiDB uses a refined multi Raft-Group-approach. The database is separated into multiple regions, each having their own Raft group. OLTP workloads are send to the group leader, which replicates them asynchronously to other OLTP followers and synchronously to an OLAP column store. Multiple Raft groups can share the same OLAP optimized column store. OLAP queries can then be executed using both, the OLTP leader and followers, as well as the OLAP database [18]. The primary scaling is achieved by splitting into more regions with their own Raft groups. Each group, however, only has one leader ingesting OLTP requests. OLAP requests can be handled by the whole Raft group as well as a read optimized column store. OLTP scaling is therefore only possible by splitting groups and creating more leaders.

Another unique approach is taken by AnyDB [29]. Instead of committing to one scheme they use servers with stateless nodes called AnyComponents (AC). These components can then take any required role (worker handling queries, query optimizer creating these queries). By adding more ACs to a server or by adding more servers they provide great scalability. The exchange between the ACs is handled by event streams buffered in queues.

VEGITO uses a shard-approach, distributing sets of a primary and backup OLTP stores with an OLAP store over multiple shared-nothing machines. The primary, backup and analytical store for the same key range however do not have to be situated on the same machine.

The PostgreSQL based Greenplum builds a cluster consisting of multiple PostgreSQL databases called segments [15]. One segment takes the role of the coordinator while the other segments build the actual database. Scaling is possible by simply adding more segments to the cluster.

2) Data/Table Structure: When dealing with OLTP and OLAP workloads, finding the right table format can be difficult. HTAP databases therefore employ different table and data structures. Wildfire [25] exclusively uses column oriented tables since they are the most efficient solution for OLAP workload. Db2 Event Store further heavily utilizes Apache Parquet and its encryption implementation [26].

SAP HANA [5] implements a row-store query engine and a column-storage engine to combine the advantages of both technologies. Thus, it is possible to save data in row or column tables. The column layout is the default, more optimized, option.

HyPer [30] and Hyrise [16] both use columnar stores with self implemented data models. Hyrise further presented a hybrid column layout in an older version [31], combining simple one-attribute-columns with rows. This is planned to be implemented again in the new version, but has low priority and is work in progress.

Opposed to this, PostgreSQL [14] continues to use its row data storage for OLTP, but has a column store extension for OLAP workloads, merging the delta from the row store

continuously in the column store. This is handled similarly by VEGITO [32] and TiDB [18]. However, column and row stores are situated on different servers for maximized query performance and safety (required by TiDB and possible by VEGITO).

While Greenplum also uses the default PostgreSQL row storage for fresh data, older data is moved to a column-oriented store and even older data can be stored in external storage systems like Hadoop [15].

Polynesia uses a row and column approach, where transactions are saved in a N-ary storage model while analytical data is saved in a three stacked memory approach [23]. The column data is stored in so called vault groups (consisting of four vaults each). Each column is then spread evenly across the vaults of a group. Following a decomposition storage model each vault further redundantly contains a dictionary to reduce lookup cost.

SnappyData [28] follows a hybrid approach, where the fresh data is stored in an in-memory row-store and is moved in an on-disk column-store after aging.

The Cloud data store Janus [19] is fully hybrid, utilizing row partitions for OLTP and column partition for OLAP. Via a redo-log inspired batching approach and a graph-based dependency management, the delta from the row replicas can be merged into the column replicas.

The Casper prototype [33] uses a tailored column layout to support mixed read/write workloads more efficiently. With this approach, runtime column adaptations are possible.

Flexible Storage Model (FSM) [34] presented a tile based architecture to allow a transition from OLTP optimized tables to OLAP optimized tables depending on the hotness of data. The data is saved in a row oriented manner at the beginning and, depending on the hotness, is tile-wise transitioned to an OLAP column oriented tile structure.

3) Saving and Partitioning Data: For scale-up focused databases, removing data from main memory to larger, more cost efficient stores (e.g., hard drives), or efficiently compressing its size, is crucial. HyPer uses horizontal partitioning and saves its hot data uncompressed on the main memory. The cold data can also be kept in memory. Instead of evicting data to a disk, the data is compressed into self implemented Data Blocks [30] and kept in main memory. However, it is possible to evict them to secondary storage solutions if preferred (e.g., non-volatile random-access memory) and use them as persistent backups. The compression technique is chosen based on the data actually saved in the Data Block.

Utilizing Small Materialized Aggregates (SMAs) including meta data like min and max values, irrelevant compressed data can easily be skipped in searches. If data cannot be skipped on SMA basis, Positional SMAs (PSMAs), another lightweight indexing structure developed by the HyPer team, can be used. These help to determine the range of positions in the Data Block were the relevant values are located.

Hyrise [31] solves this problem using horizontal partitioning and by saving data in 2 kinds of columns: Memory-Resident Columns for hot data in memory, allowing fast access, and uncompressed row-oriented Secondary Storage Column Groups for cold data on hard drives. As the cold data is saved uncompressed, the cost of accessing it is reduced in comparison to classical compressed approaches.

Furthermore, the data is organized in so-called chunks [16] similar to Data Blocks. Chunks can be mutable as long as they are not full. As soon as they reach their capacity they transition to an immutable append-only container. They also have indexes and filters on a per chunk basis like Data Blocks, allowing faster search and access operations.

In Poseidon, data is saved in non-volatile memory (NVM) as well as main memory [21]. New transactions are kept solely in main memory until the commit is conducted, then the final data is saved on NVM. With this approach the low latency of main memory can be used for all steps of an transaction, while at the same time the final data object is saved in a persistent way. Another project exploring NVM in the form of directly-attached-NVMe-arrays (DANA) was conducted by Haas et al. [35], optimizing LeanStore for DANA usage achieving promising results. However, updates of traditional disk-based systems seem to be no feasible option due to high CPU load.

Smart Larger Than Memory [36] stores cold data in files on the hard drive decoupled from the database. Modifications to the data are no longer possible. The entries can only be deleted. This happens via removing the reference entry in memory without accessing the cold data and thereby saving time. Updating cold data is possible, but the update is a hidden delete of the cold data index and an insert of new hot data. To fully take advantage of SmartLTM the read operations always check the main memory entries first. If the data cannot be found, cuckoo filters or SMAs are used to locate the data in the files on the hard drive.

In VEGITO data is stored in paged blocks called epochs [32]. The epoch counter is continuously increased (around every 15ms). Witch each new epoch the data from the old epoch either gets copied to a new page, if a change occurred in the block, or is simply referenced if no change occurred. This saves copy time as well as memory.

TiDB first saves deltas in memory. When the amount of small deltas increases they are merged to bigger deltas and moved to on-disk storage [18]. Like in F1-Lightning, data partitions can further be adjusted on the fly, splitting to large partitions or merging smaller ones for optimized performance.

Like most systems Db2 Event Store utilizes an approach were the most recent data is moved to the fastest storage and is moved down the chain (in total three zones) as it ages [26]. In the first step fresh data is saved on NVMe SSDs, then it is moved to a ”preShared Zone” in shared memory until it is finally moved to the ”shared Zone” by tracking its meta data in Zookeeper allowing the transfer of the data’s leadership between nodes.

The Relational-Memory Approach allows native access to a row and column format [37]. Utilizing Field-Programmable Gate Arrays, Programmable Logic In-the-Middle relational operations are implemented in reprogrammable hardware. Queries get then provided with the data by the Relational

Memory Engine in the optimal layout instead of accessing the memory itself.

Finally, partitioning workloads in an intelligent manner without extra statistical data structures is possible, too. As presented by Boissier and Kurzynski [38], physical horizontal data partitioning as well as the adapted aggressive data skipping approach can skip up to 90% of data on OLAP queries.

B. Concurrency

Handling multiple versions of data is a crucial part of all HTAP databases. Current OLTP and OLAP operations require a solution to parallelize data access.

The most common approach is Multi Version Concurrency Control (MVCC). It is utilized in combination with a delta by PostgreSQL [14], SAP HANA [5], F1-Lightning [39], TiDB [18], Poseidon [21] and in new versions of HyPer [40]. To improve scan times in such MVCC systems, a first generic effort is made by vWeaver, implementing a per record frugal skip list to reduce lookups by smartly adding ”shortcut”-pointers between different versions [41].

Hyrise [42] is also using MVCC, but is following a look free commit approach, replacing the delta.

SnappyData and BatchDB [13] also use MVCC oriented approaches. SnappyData [12] relies on GemFire to handle concurrent access and snapshots, while BatchDB [13] uses MVCC on its OLTP replica, while updating the isolated OLAP replica batch-wise.

Although HyPer is now using MVCC with delta, it initially used the fork systemcall to create multiple isolated in-memory snapshots [43]. Utilizing a copy on write approach to reduce memory consumption OLAP queries could be executed on snapshots while the OLTP operations updated the main memory entries.

In addition to the MVCC on its main OLTP replica, SAP HANA [24] further uses ATR with a replication log system to synchronize its multiple server architecture. This synchronizes data with sub-second visibility delay between the replicas.

Instead of using classical MVCC, Polynesia uses a snapshot chain for each column with versions containing whole columns rather than deltas. Following a lazy approach, versions are marked as dirty if changes to the data occur instead of creating a new snapshot. If a dirty column is then accessed by an OLAP query the new snapshot with the latest data is created. These snapshots can further be shared between multiple OLAP queries if required. The row and column store are further synchronized by the update shipping/application unit following two multi component architectures. With this approach simplified transactions are added to queues, intelligently merged to a single update buffer, and then applied to the column store.

Wildfire [25] chooses speed over concurrency as already mentioned. Therefore, a simple last writer wins approach is used by the Wildfire engine.

While most systems nowadays use some implementation of MVCC, research on more efficient snapshotting techniques is still being carried out, e.g., [44]. Inspired by earlier HyPer implementations, another research project on snapshotting, AnKer [45], uses a customized Linux kernel with an updated fork system call. This updated fork, called vm_snapshot, enables high frequency snapshotting. Through vm_snapshot the researchers are able to snapshot only the used columns. This significantly outperforms the default fork used initially by HyPers implementation, providing a possible alternative to MVCC systems.

In addition to PostgreSQL’s locking system, Greenplum further introduces a Global Deadlock Detector (GDD) to resolve deadlocks originating from the distributed server setup [15].

Wait free HTAP (WHTAP) [46] utilizes snapshotting for concurrency as well. In this dual snapshot engine approach data for OLAP and OLTP are stored in different replicas, using a five state process and two deltas. In this process, the deltas form the OLAP and OLTP replicas are switched and the old OLTP delta is merged into the OLAP replica, which takes effect without slowing the analytical queries down.

In VEGITO concurrency can be handled by the epochs utilized to save data [32]. While new updates are always applied to the primary row store of a shard the data is then asynchronously applied to the replica and the analytical column store by a non-volatile write-ahead log batched in epochs. The analytical stores save the latest epoch currently available on all stores and analytical queries are then executed on the epoch, rather than the current epoch used by the primary row store.

AnyDB once again takes a unique approach to concurrency control. By handling the communication between the ACs via event-streams and queues concurrency control is achieved by routing the steams intelligently, merging read requests events on AC nodes with the write requests, only providing the query response after the write event is received [29].

C. Garbage Collection

MVCC implementations require performant garbage collection to prevent large amounts of versions to slow down the transactions on the database. SAP HANA [5] uses timestamps and visibility bits to track versions of their data. Data gets created/edited with a timestamp. When all active transactions can see this version the timestamp is replaced with a bit indicating the visibility. If the row is no longer visible to any snapshot, it can be deleted with the next delta merge.

HyPers garbage collector Steam [40] follows a similar approach. The main difference is that the garbage collector is called with every new transaction instead of being a background task like with SAP HANA. This approach called eager pruning removes all versions not required by any transaction. This happens by checking every time the chain is extended whether all versions included in the version chain are used by a transaction. With eager pruning the version chain can only be as long as the amount of different queries. A similar approach is conducted by Poseidon [21] and Polynesia. In Poseidon on each transaction, data, which is invalid (e.g., due to an aborted transaction) or no longer visible on any version, is pruned from the database. Polynesia deletes all unused snapshots after each

finished analytical query, which are no longer used by any running query.

On VEGITO systems data is saved in epochs rather than with timestamp [32]. As soon as an epoch is outdated (no fresh data is contained in an epoch as all entries are either updated or deleted) the epoch can be removed as a whole from the system.

Due to its heavily distributed architecture with many data sets saved in main memory and SSDs on the different servers, Wildfire follows a different solution and implements a lazy garbage collection approach [25]. When performing lazy garbage collection, data is only deleted if there is no possibility that a query could require it. In Db2 Event Store data is pruned after being moved to the next zone, preventing data duplication [26]. It is also possible to delete the persistent data in the final zone by defining a time to live in the configuration.

D. Query Handling

The ways to access the concurrent data differ significantly from database implementation to implementation.

1) General Query Optimization: Besides the general handling of the queries, Sirin et al. [47] show the importance of isolating the OLTP and OLAP workloads on shared hardware systems in order to achieve optimal performance.

Tested on the Umbra SSD-based HTAP database a worst-case optimal join processing option is introduced providing a general improvement option for HTAP systems [11]. Implementing a hybrid query optimizer a single query plan can be build out of binary and worst-case optimal joins, greatly increasing OLTP performance while having no effect on OLAP workloads.

2) Query Handling in Scale-up Systems: The systems HyPer [48] and Hyrise [16], primarily engineered for scale-up solutions working on one dataset, implemented the query operators as C++ code in their database. The missing variables are inserted via just in time compilation. After the insertion, the code is compiled to LLVM assembler code, allowing fast query execution. As mentioned before, the two databases also have prototype scale-out options, but focus on the scale-up approach.

The LLVM approach is also utilized by Poseidon. The primary commands are already ahead of time (AOT) compiled and the query execution starts instantly. At the same time the query is compiled to LLVM code. If the LLVM code is compiled before the query finishes the query execution is moved from the AOT code to the more optimized LLVM code. The compiled LLVM queries are further saved in the non-volatile memory for faster processing of repetitive queries [21][49].

In Polynesia query operators get arranged in a tree structure [23]. These are then further separated into sub-tasks and if possible executed in parallel, speeding up execution time.

Another approach is to dynamically schedule memory and computing resources actively [50]. Utilizing their algorithm the Resource and Data Exchange engine uses a state based approach to assign the CPU cores to the OLAP or OLTP workload as required, always trying to maximize productivity and the database throughput.

3) Query Handling in Scaled-out Systems: Scale-out systems are separated in two major groups: On the one hand, systems keeping the OLTP workload on one server, scaling only the OLAP workload to other servers, as e.g., SAP HANA [24] or BatchDB [13]. On the other hand, systems distributing OLTP and OLAP workloads over multiple servers, e.g., Wildfire [17][25] and SnappyData [12][28].

BatchDB [13] and HANA [24] both handle their OLTP workload on a single server, scaling-up via NUMA as described earlier. For OLAP workloads they are able to scale out onto multiple servers working on replicas of the main data.

Wildfire [17] and SnappyData [28] contrarily scale out via Apache Spark, allowing OLTP and OLAP transactions to be executed on a cluster of nodes dealing with big data and streaming workloads. Wildfire [17] executes OLTP queries on the fresh data on Wildfire daemons. OLAP workloads can be executed via Spark Executor as requests to the daemons or directly accessing the shared data of the Wildfire database cluster. With this approach, old data can be consumed from the shared file system without slowing down OLTP throughput while the latest data can still be received if required. Db2 Event Store utilizes the Db2 BLU MPP cluster query engine instead of Apache Spark for the sake of better low latency query handling [26].

Greenplum handles all queries in a cluster via its central coordinator [15]. The queries are then processed and handled by workers. Via ”Motion”-nodes data can be transferred between separated machines/segments.

In AnyDB queries can be executed on multiple servers by simply routing the query events to the according ACs [29]. To speed up processing even further data can be ”beamed” to ACs in advance, before the query is available. By determining the AC responsible for the query execution and knowing, which data will be required by the query the data is send to the AC before the query optimization is done, providing a great speed up by parallelizing the two work steps.

4) Query Language: While the databases offer many new functionalities to access and modify data, SQL is still commonly supported. The database systems SAP HANA [8], Wildfire [25], Db2 Event Store [26], Hyrise [16], HyPer [7], SnappyData [12], TiDB [18], AnyDB [29] and AIM [20] all enable basic SQL queries to interact with the database. However, many of them further provide new optimized ways to interact with the data.

Wildfire [25], TiDB [18] and SnappyData [12] provide data access via an extended version of the SparkSQL API. SnappyData also further extends the Spark Streaming API.

SAP HANA [8] provides more specific access through SQL Script and Multidimensional Expressions (MDX). The database is also is natively optimized for the ABAP language and runtime. This allows to bypass the SQL connectivity stack by directly accessing special internal data representations via Fast Data Access (FDA). The Native For All Entries (NFAE)

technique further modifies the ABAP runtime to allow even more performance improvements.

Hyrise [16] provides a command-line interface, which allows SQL queries but also provides additional visualization and management functions. Furthermore, the wire protocol of PostgreSQL allows access through common PostgreSQL drivers and clients.

HyPer [48] uses HyPerScript as its query language. HyPerScript is a SQL-based query language and therefore allows base SQL statements as well. The features consist of passing whole tables as query parameters and providing the possibility to use query results in a later part of the query, removing the need to query the same value multiple times.

E. Indices

To allow efficient data access and querying on multiple servers and/or different versions of data, the right index structure is of special importance in HTAP databases.

Wildfire's multi-version multi-zone index Umzi [51] employs a LSM-like structure with multiple runs. It divides index runs in multiple zones and implements efficient evolve operations to handle zone switches of data. Further Umzi uses a multi-tier storage using SSDs and memory caching with self-updating functionality for fast execution while persisting the indexes on Wildfires shared data. Db2 Event Store's index is based on Umzi [26]. It is only applied to the preShared and shared zone, as the initial zone does not support synchronization.

A general approach to utilizes LSM-Trees for HTAP workloads is further presented by Saxena et al. [52]. Their prototype LASER utilizes a Real-time LSM-Tree allowing to store data in different formats during its lifecycle (similar to Db2 Event Store) providing significant speed ups to traditional methods.

HyPer developed the Adaptive Radix Tree (ART) [48] based on the radix tree. ART uses four different node types that can handle 4, 16, 48 and 256 entries. The maximum height for the tree is k for k-byte trees. To further reduce the tree height and required space, the tree is build lazily, saving single leaf branches higher in the tree. Additionally, path compression is used to remove common paths and to insert them as a prefix of the inner node thereby removing cache inefficient one-way node chains.

SAP HANA [5] and Hyrise [16] both use B-Trees. Hyrise further supports the ART index from HyPer [48] and a group-key-index, implemented by the Hyrise project.

TiDB uses its own implementation, the DeltaTree [18]. Its creators further built a B+-Tree on top of the delta tree, to speed up updates on key ranges and look ups on single key values, as well as merging of deltas.

A B+-Tree is further used as the index of Poseidon [21]. The leafs of the B+-Tree are saved in non-volatile memory, providing persistency and recovery options, while the inner structure of the tree is saved in memory. With this approach only one non main memory access has to be conducted, while still providing enough persistency for efficient recovery.

Once again utilizing its epoch system VEGITO [32] can use a buffered tree based index. As changes to the OLAP index tree only must be applied after an epoch finishes all changes during an epoch are applied in parallel to buffers and the three weights are updated. In the update step at the end of an epoch the tree first gets optimized (also a split is possible here) and the the buffered changes are applied. This approach evades slowdowns by removing the need for locking with buffers.

BatchDB utilizes a simplified version of the look-free Bw-Tree [13]. The version relies on atomic multi-word compare-and-swap updates.

In 2019, a predictive indexing approach [53] was introduced to cope with the dynamic demands of a HTAP database. Predictive indexing increases the throughput by up to 5%. In this approach, a machine learning system calculates the optimal index structure for the data according to the workload. A similar approach can be seen in the Multi-armed bandit solution from 2021 [54]. With this approach even greater improvements up to 59% speedup are possible.

The Multi-Version Partitioned B-Tree (MV-MBT) [55] is another recent research in the indexing sector for HTAP databases from 2019. This extension of partitioned B-Tree creates a version aware index, able to maintain multiple partitions within a single tree structure, sorted in alphanumeric order.

Likewise proposed in 2019, the Parallel Binary Tree (P-Tree) [56] is an extension of a balanced binary tree relying on copy-on write mechanisms to create tree copies on updates. With this approach, the indices become the version history without requiring other data structures.

F. Big Data on HTAP Databases

Wildfire/Db2 Event Store was created with big data IoT workloads as its primary use case [25][26]. Through the distributed design of its big data platform, Wildfire is able to concurrently handle high-volumes of transactions as well as execute analytics on latest data. At the same time, the system is able to scale onto many machines because of its close integration of Apache Spark. The usage of an open data format further enables compatibility with the big data ecosystem. Nowadays, the commercial version IBM Db2 Event Store is capable of handling more than 250 billion events per day [57][26]. SnappyData [28] is an analogically capable big data platform with an architecture similar to Wildfire.

The SAP HANA database can be used as part of the SAP HANA data platform to handle big data workloads [58]. Using a combination of different SAP products, namely SAP Synbase ESP and SAP Synbase IQ, as well as smart data access frameworks as Hadoop, Teradata or Apache Spark, the SAP HANA data platform is a fully functional big data system with SAP HANA in its core.

HyPerInsight [59] provides big data capabilities in the area of data exploration on the HyPer database. The goal is to minimize the required user expertise with the dataset while simultaneously supporting the user with the formulation of queries. The support for lambda functions in SQL queries

allows user defined code to be executed within the queries. In combination with the HTAP HyPer system as the database, data mining on real-time data is possible.

G. Recovery, Error Handling and Logging

As many HTAP databases rely on volatile main memory as primary storage and the other systems utilize distributed data sets, recovery in case of failure is of special importance. Data loss has to be prevented and downtime must be minimized.

SAP HANA instances log data persistently on the local drive for recovery on failure or restart purposes [8]. The logging approach is inspired by SAP MaxDB.

As already explained, HANA works with ATR in its distributed architecture [24]. Following the store-and-forward approach, the data is replicated to multiple servers. An algorithm then compares the record version IDs of the incoming data and stored data, requesting the resend of lost log entries if deviations occur.

Recovery for the latest version of Hyrise is still work in progress [16], but recovery for older versions of Hyrise was explained [42]. The database dumps the main partition of the table as a binary dump on the disk and records the delta to a log via group commits to hide the latency. At checkpoints, the delta partitions are also saved as a binary dump on the drive. If recovery is required, the main dump and delta dump from a checkpoint are restored and an eventually existing delta log is replayed on the table, restoring the old state.

BatchDB logs successful transactions on its OLTP replica in batches via command logging on durable storage [13]. In case of a failure, the database can recover from these logs. The OLAP replica itself has no durable logging and has to recover from the main OLTP replica on failure.

SnappyData [12] uses Apache Sparks logging and recovery mechanisms, logging transformations used to build Sparks Resilient Distributed Datasets (RDDs). Saving RDDs to storage is also possible. In SnappyData the combination with GemFire however allows Spark to save the RDDs in GemFires storage instead of the persistent storage of the server. Small recoveries can be handled directly by GemFires eager replication, leaving batched and streaming recovery to Spark, in combination with the GemFire storage. Further, a peer-to-peer (p2p) approach is used in SnappyData clusters. Any in-memory data can be synchronously replicated from the cluster. Additional to the replication via the p2p approach, data is always replicated to at least one other node in the cluster.

VEGITO [32] offers a quite refined recovery system, handling four failure cases. If the primary row store fails the leader role is transferred to its backup row store and a new backup row store is created. In case of a backup store failure it is simply recreated from the primary row store. If the OLAP column store fails it can be recreated from the row stores. Finally, if both row stores fail it is possible to recreate the primary row store on the same machine as the column store, afterwards a backup row store is initialized.

TiDB Raft groups recover in case of a leader failure by electing a new leader from the groups followers. OLTP workloads are then simply redirected to the new leader and continue there [18].

AnyDB once again introduces a new approach. As all communication inside AnyDB is handled by event streams these can simply be rerouted to a functional AC in case of a failure of an AC/server [29].

IBM's Db2 Event Store provides a catalog, which contains meta data required for initial cache population [26]. In order to prevent possible data loss the catalog data is saved in a shared persistent storage and provided to the system via a logical node. If the catalog node fails, the data can be reloaded from this storage by another server of the cluster, which then resumes the work of the catalog node. All data is further saved on fast local storage of one node, as well as to the local file system of at least two other nodes, allowing for easy recovery.

H. Benchmarking

The combination of OLTP and OLAP workloads on one database also created the need for new benchmarks covering this sector. In 2011, CH-benCHmark [60] was introduced. The CH-benCHmark is based on the TPC-C and TPC-H benchmarks. It executes a transactional and analytical workload in parallel on a shared set of tables on the same database. The benchmark can also be used for single workload databases.

In 2017, HTAPBench [61] was published. This benchmark is able to compare OLTP, OLAP and hybrid workloads on the database. Its main difference to CH-benCHmark lies in its Client Balancer, controlling the coexisting OLAP and OLTP workloads.

Hyrise [16] implements a special benchmark runner to easily execute benchmarks.

Another benchmark specially designed for document oriented NoSQL platforms is introduced by Tian et al. [62].

I. Stream Processing

Streaming as a special case of OLTP is an emerging use case for HTAP database systems. In 2016 scientists from ETH Zürich in cooperation with Huawei presented AIM [20], which is a high performance event-processing and real time analytics HTAP database. The three-tiered multi node system processes events at one tier, stores the data at a central tier and finally analyses the processed data in real time on the third tier. AIM however, is optimized for a special streaming use case from the telecommunications industry.

In early 2019, the research team around HyPer compared modified versions of HyPer with AIM and Apache Flink [63] in order to determine the current state of streaming capabilities of main memory database systems (MMDB). While MMDBs are still inferior to dedicated streaming frameworks like Flink, the HyPer team was confident, that HTAP databases could catch up with some adjustments, even implementing some of those on HyPer. The main areas requiring improvement are network optimization, parallel transaction processing, skew handling and a strong distributed architecture.

SnappyData aims to solve OLTP, OLAP and streaming all in one product [28] with their tight integration of Apache

Spark and GemFire. In an evaluation, SnappyData was able to outperform both Spark on TPC-H queries as well as MemSQL on all kinds of throughput. The focus with SnappyDatas stream processing lies on complex analytical queries on streams, which are not possible with default stream processor solutions [12].

SAPs approach for big data, SAP Big Data [58], supports streaming as well. Since it uses other SAP products to achieve it and is not a part of the base SAP HANA database infrastructure, but rather is built on top of it, it is not further discussed in this paper.

J. HTAP as a Service

In the last year HTAP became more and more popular with cloud solutions. Following this trend some of the global players enabled "HTAP-as-a-service" (HaaS) for their existing could solutions. Instead of providing new databases they introduce HTAP-like functionality with tightly coupled OLTP and OLAP multi database setups. Still they provide some interesting new research and provide a valid solution for production systems.

1) F1-Lightning: F1-Lightning consists of distributed multi actor system with data aggregation components, in memory and on disk data storage, as well as a metadata database, containing all required information to function. Instead of being a fully functional HTAP database itself, it can be selected as an addition to the Google Cloud Platform databases F1-DB and Spanner. Either for some tables or whole databases the analytical queries are moved to the F1-Lightning cluster, while the transactional queries stay on the OLTP databases. With this approach the currently used database does not have to be exchanged and F1-Lightning can be added and removed on the fly. While currently only F1-DB and Spanner are supported as the base OLTP database the architecture is highly adjustable and further databases could be supported in the future. For the databases currently being extended F1-Query is utilized as a query language.

Using a MVCC approach with snapshot isolation the queries are conducted either in the OLAP improved Lightning tables or on the OLAP F1-DB/Spanner tables, if the data was not yet copied. Using a time-to-live (TTL) based garbage collection approach and compaction techniques older data is deleted and small deltas are combined to larger, more memory efficient blocks. However, the used garbage collection approach also deletes still valid data, if it was not changed in a certain time span.

Due to information stored in the metadata database and the modular architecture, partitions of data can be split and merged on the fly as required, without disturbing the analytical queries. If certain deltas getting to large for in-memory usage they are moved and transformed from the row based in memory store to on-disk storage in a read optimized column structure.

2) Azure Synapse Link: Azure Synapse Link is currently only available as a service for Azure Cosmos DB [64]. Unlike F1-Lightning it can only be added to the whole database. Moreover, there is a high delay between the latest OLTP data and the data used for OLAP queries of 2-5 minutes. In its core functionality Azure Synapse Link synchronizes the data from the row based Azure Cosmos DB to a column based analytical database and provides a bridge to Azure Synapse Analytics. Like F1-Lightning a TTL-based garbage collection is utilized.

K. Future

HTAP databases are a new sector, which has evolved over the past 10 years. On an annual basis, companies and researchers contribute new ideas to lift their database above the competition. While we stated in our last work, that there are currently three trends it seems like a fourth has emerged.

1. CPU GPU collaboration: With new hardware supporting heterogeneous parallelism, Heterogeneous HTAP (HHTAP/H2TAP) becomes a possibility. In this approach, CPUs and GPGPUs can access shared memory and divide the workload between both. Complex OLAP queries are solved on the GPGPUs, leaving the OLTP workload for the CPUs. The Caldera [65] prototype proved the feasibility for HHTAP. Early 2020, the data store GridTables [66] was published and affirmed the concept again. However, in their summary, the authors of GridTables pointed out that there are still many research issues left to be solved. Further, early in 2020, a paper was published about GPU accelerated data management [67] explaining how to fully exploit hardware isolation between CPUs and GPUs and presenting a SemiLazy access method to reduce the required data transfer. EEVEE (inspired by and named after the highly adaptable Pokémon Evoli/Eevee) provides an interesting approach for scheduling such HHTAP workloads [68]. While the work was never formally published it can provide some valuable insights for upcoming developments in this area.

2. Streaming workloads: as described in detail in the previous section, stream processing is a possible use case for HTAP databases. Because of the optimization for high OLTP throughput and the ability to analyse these data streams in the same system, HTAP databases are an emerging alternative to current stream processing solutions. While still inferior to dedicated stream processors, the research on such solutions saw an increase in interest over the last years, e.g., by the HyPer team [63] and dedicated streaming HTAP databases like SnappyData [28] and AIM [20].

3. Optimization: while the bigger part of the last decade was spent on researching for new systems [7][8][42], the last quarter focused on their optimization. Few new database systems were proposed and research started optimizing existing systems even further [40][45][56]. Also research focusing on the general improvement of HTAP systems becomes more common, e.g., [47][41][52].

4. Over the last year the interest in NVMe storage has greatly increased. Different research is investigating how to efficiently utilize this kind of non volatile memory [21][35] while production systems already started utilizing it [26]. While using NVMe instead of ordinary SSD may not seem that interesting at first this approach offers many new opportunities (mainly connected to way better speed than ordinary SSDs

while much cheaper than main memory) and challenges arise (e.g., problems with upgrading traditional disk-based systems to NVMe).

Further, solutions utilizing machine learning in combination with HTAP are slowly emerging. These allow databases to adapt on their own according to current workload and requirements. However, there is still not enough research to speak of an own trend and it can rather be viewed as another kind of optimization research. An example for such research is the presented predictive indexing [53] or the Multi-armed bandit approach [54].

Another new development is the integration of HTAP capabilities into polystore databases. While polystore and HTAP databases evolved mostly disjunct over the last years first approaches to combine both paradigms were conducted with Polypheny-DB [69] and [70].

Not being a trend in the HTAP database development, because simply not being HTAP databases, more and more solutions emerge providing HTAP capabilities to pre-existing database systems, efficiently bridging the gap between OLTP and OLAP systems instead of creating new combined database systems. Examples can be found by the HTAP-as-a-Service solutions [39][64], as well as systems like IBM Db2 Analytics Accelerator [71].

L. Open Source and Free Versions

Some of the database systems summarized in this paper provide open source and/or free solutions.

SnappyData [76] is available with a getting started guide covering the basic usage. The source code can be found on GitLab. The project is licensed with the Apache License, Version 2.0. A comment in the GitHub Readme however declares the project as legacy. An official statement to the state of SnappyData could not be found.

MemSQL [77] (now SingleStore), is available, well documented and can be used for smaller projects up to 4 nodes for free. Many extensions are available at the official GitHub account.

Hyrise [75] is available under the MIT license. However, as it is a research database, breaking changes may occur more frequently.

A initial version of VEGITO is available under the Apache License Version 2.0 on GitHub [78]. With the same license a production ready version of TiDB is also available on GitHub [72]. It is provided with a complete documentation, however, 32 GB+ RAM are advised for a minimal production setup and enabling the HTAP capabilities requires further setup and understanding of some of the underlying software solutions.

Poseidon is available via GitHub under the GPL 3.0 license [79]. However, while the publications to this database are from 2021 the last commit is from February 2020. The git repository therefore seems outdated.

While PostgreSQL itself is available open source and can be modified to handle HTAP workloads [80] there are some other open source database systems building on top of PostgreSQL. These are most of the time more refined and easier to use with HTAP workloads. Greenplum (Apache 2.0 License) [73] requires linux servers with 16 GB RAM and all utilized servers of a Greenplum cluster require the same hard- and software configuration.

The free MemSQL version and a basic SnappyData setup can already be used on low end systems, naming 8GB main memory as their minimal requirement to operate efficiently.

To try a HTAP database without a setup process, HyPer and Umbra can be used. Both research version are provided via a simple web tool for exploration and testing [81] [82]. This version, however, is running on a low end system and cannot be used in production.

V. Production Ready HTAP Databases

While the previous part of this work is solely based on the systematic literature review this section also highlights databases, which were not found during the review process, if they are a valid production ready solution. The focus lays on technologies already mentioned in the literature review part of this paper, which are production ready, as well as the big players. Besides HTAP databases we further include HaaS solutions, enabling HTAP on existing OLTP cloud systems. Table I provides a compact overview (only links to free versions are provided).

VI. Conclusion

In this paper, we have shown that HTAP databases are nowadays serious alternatives to traditional database solutions. The existence of a market for commercial products like SAP HANA, IBM Db2 Event Store and Tableau further reinforces our findings. Moreover, we have highlighted differences of existing approaches regarding key properties like the fundamental architecture, concurrency, or big data capabilities. Furthermore, we have highlighted a set of currently available production ready HTAP implementations ranging from open source systems to commercial products. Thus, this study can aid both researchers and practitioners in the process of selecting a matching HTAP solution. Finally, by providing a comprehensive overview of current approaches, this study helps to identify trends and point out directions for future research. As there is a great amount of active research in this area, this article builds upon our previous literature study on this topic and enhances it with new alternatives as well as production ready approaches. The following paragraphs provide a brief summary of our findings.

Open source and free HTAP products place HTAP databases on the same level as traditional database systems, allowing the integration in other products and exploring this new technology without financial risks.

The combination of OLTP and OLAP queries on one database efficiently reduces the total cost of ownership and allows a narrower tech stack for companies. The possibility to analyse data in real time further validates HTAP databases as a productive solution with a great added value compared to conventional databases.

TABLE I
PRODUCTIVE HTAP DATABASE SYSTEMS

Database	Availability	Pricing	Core Facts
F1-Lightning	- Google Cloud Platform	- Spanner subscription + - F1-DB subscription +	highly scalable; can be added and removed on the fly; extends existing OLTP databases; (if already using F1-DB or Spanner) no initial database migration required; only available for Spanner and F1-DB; no "real" HTAP
Azure Synapse Link	- Azure	- requires multiple Azure subscriptions	can be added on the fly; extends existing OLTP databases; no initial database migration required; only available for Azure Cosmos DB; no "real" HTAP; high delay between OLTP and OLAP store (2-5 minutes)
IBM Db2 Event Store	- local - own Linux cluster - IBM Cloud Pak	- free developer version - free test, not specified - not specified	enormous scalability; IoT optimized; free developer version; can be self hosted (in contrast to Azure/Google Cloud); deeply integrated with other IBM solutions; very active ongoing research
SAP HANA	- self hosted - cloud	- free test, not specified	highly integrated with other SAP products; very active ongoing research; distributed or single server scaling
IBM Db2 Analytics Accelerator	- on IBM z/OS systems	- not specified	HTAP database extension for Db2 for z/OS systems; tightly coupled with other IBM products; requires a Db2 for z/OS database on a mainframe; very active ongoing research; one of the most researched systems
Amazon Aurora	- AWS	- multiple AWS subscriptions	cloud only; low visibility in HTAP research; easy scalability
Tableau	- hosted - self hosted	- stating at 12$/month	based on HyPer; one of the most researched systems; part of tableau technology stack; not available as simple database
TiDB	- GitHub [72]	- open source - enterprise available	highly scalable; moderate minimal hardware requirements; high fault resistance; setup requires technical knowledge
Greenplum	- GitHub [73]	- open source	PostgreSQL based; highly scalable; moderate minimal hardware requirements; integrated machine learning and analytical capabilities
Swarm64	- self hosted - cloud	- free test version - starting 528$/month	PostgreSQL based; can be used on common clouds; not present in research
Citus	- GitHub [74]	- open source	PostgreSQL based; great scalability; available on Azure; some research papers; multi-node PostgreSQL cluster
Hyrise	- GitHub [75]	- open source	very active research; active development; research system not suitable for production

Many different implementations, providing different advantages, are available and can be used as required by the costumer. Solutions using main memory as a primary/sole storage as well as solutions relying on shared data storages exist and are both valid options. Powerful single server database systems allow a slim tech stack while still being faster than most traditional OLTP and OLAP optimized databases. Distributed multi server clusters allow more fail-safe and easier to scale solutions, while at the same time requiring less performant machines. SnappyData and MemSQL, for example, can already be executed on machines with 8GB of memory, scaling up from there.

Over the last years, new indices, filters, data structures and replication techniques were developed, optimizing performant HTAP systems even further. The future seems to be heading in three main directions: HHTAP - utilizing new heterogeneous hardware to include the GPUs in HTAP databases and allow even more efficient architectures.

Streaming - HTAP databases optimized for streaming are making a combination with external stream processors unnecessary, further reducing the total cost of ownership and reducing the size of the required tech stack.

Optimization - while the bigger part of the 2010's was spent on developing the base technologies and databases themselves, the last quarter was primarily spent on optimization, still leaving much room for improvement.

Machine learning for self adapting databases also could be an emerging sector in the future, but currently there is not enough research in this direction to call it a trend.

There seems to be a growing interest in HTAP solutions yielding high numbers of novel approaches and providing innovative solutions regrading many technical aspects of databases. Therefore, our future work will focus on further exploring the state of research. Due to the amount of active research it will be difficult to capture the whole area in a comprehensive literature review. Thus, we aim at providing in-depth overviews of different aspects of this database area.

REFERENCES

[1] D. Hieber and G. Grambow, "Hybrid transactional and analytical processing databases: A systematic literature review," *DATA ANALYTICS 2020, The Ninth International Conference on Data Analytics*, pp. 90–98, 2020.

[2] R. Nigel, F. Donald, P. Massimo, and E. Roxane, "Hybrid transaction/analytical processing will foster opportunities for dramatic business innovation," 2014, last visited: 12.06.2022. [Online]. Available: https://www.gartner.com/en/documents/2657815

[3] U. Joseph *et al.*, "Predicts 2016: In-memory computing-enabled hybrid transaction/analytical processing supports dramatic digital business innovation," 2015, last visited: 12.06.2022. [Online]. Available: https://www.gartner.com/en/documents/3179439

[4] Y. Noel and G. Mike, "Emerging technology: Translytical databases deliver analytics at the speed of transactions," 2015, last visited: 12.06.2022. [Online]. Available: https://www.forrester.com/report/Emerging-Technology-Translytical-Databases-Deliver-Analytics-At-The-Speed-Of-Transactions/RES116487

[5] N. May, A. Böhm, and W. Lehner, "Sap hana – the evolution of an in-memory dbms from pure olap processing towards mixed workloads," in *Datenbanksysteme für Business, Technologie und Web (BTW 2017)*, B. Mitschang, D. Nicklas, F. Leymann, H. Schöning, M. Herschel, J. Teubner, T. Härder, O. Kopp, and M. Wieland, Eds. Gesellschaft für Informatik, Bonn, 2017, pp. 545–546.

[6] A. Kemper, V. Leis, and T. Neumann, *Die Evolution des Hauptspeicher-Datenbanksystems HyPer: Von Transaktionen und Analytik zu Big Data sowie von der Forschung zum Technologietransfer*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 149–154. ISBN 978-3-662-54712-0

[7] A. Kemper and T. Neumann, "Hyper: A hybrid oltp olap main memory database system based on virtual memory snapshots," in *2011 IEEE 27th International Conference on Data Engineering*, 2011, pp. 195–206.

[8] F. Färber *et al.*, "The sap hana database - an architecture overview," *Bulletin of the Technical Committee on Data Engineering / IEEE Computer Society*, vol. 35, no. 1, pp. 28–33, 2012.

[9] B. Kitchenham, "Procedures for performing systematic reviews," *Keele, UK, Keele Univ.*, vol. 33, 08 2004.

[10] T. Neumann and M. J. Freitag, "Umbra: A disk-based system with in-memory performance," in *10th Conference on Innovative Data Systems Research, CIDR 2020, Amsterdam, The Netherlands, January 12-15, 2020, Online Proceedings*. www.cidrdb.org, 2020. [Online]. Available: http://cidrdb.org/cidr2020/papers/p29-neumann-cidr20.pdf

[11] M. J. Freitag, M. Bandle, T. Schmidt, A. Kemper, and T. Neumann, "Adopting worst-case optimal joins in relational database systems," *Proc. VLDB Endow.*, vol. 13, no. 11, pp. 1891–1904, 2020. [Online]. Available: http://www.vldb.org/pvldb/vol13/p1891-freitag.pdf

[12] B. Mozafari, "Snappydata," in *Encyclopedia of Big Data Technologies*, S. Sakr and A. Y. Zomaya, Eds. Springer, 2019.

[13] D. Makreshanski, J. Giceva, C. Barthels, and G. Alonso, "Batchdb: Efficient isolated execution of hybrid oltp+olap workloads for interactive applications," *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017.

[14] M. Nakamura *et al.*, "Extending postgresql to handle olxp workloads," in *Fifth International Conference on the Innovative Computing Technology (INTECH 2015)*, 2015, pp. 40–44.

[15] Z. Lyu, H. H. Zhang, G. Xiong, G. Guo, H. Wang, J. Chen, A. Praveen, Y. Yang, X. Gao, A. Wang, W. Lin, A. Agrawal, J. Yang, H. Wu, X. Li, F. Guo, J. Wu, J. Zhang, and V. Raghavan, *Greenplum: A Hybrid Database for Transactional and Analytical Workloads*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 2530–2542. ISBN 9781450383431

[16] M. Dreseler *et al.*, "Hyrise re-engineered: An extensible database system for research in relational in-memory data management," in *EDBT*, 2019.

[17] R. Barber *et al.*, "Evolving databases for new-gen big data applications," in *CIDR*, 2017.

[18] D. Huang, Q. Liu, Q. Cui, Z. Fang, X. Ma, F. Xu, L. Shen, L. Tang, Y. Zhou, M. Huang, W. Wei, C. Liu, J. Zhang, J. Li, X. Wu, L. Song, R. Sun, S. Yu, L. Zhao, N. Cameron, L. Pei, and X. Tang, "Tidb: A raft-based htap database," *Proc. VLDB Endow.*, vol. 13, no. 12, pp. 3072–3084, Aug. 2020.

[19] V. Arora, F. Nawab, D. Agrawal, and A. E. Abbadi, "Janus: A hybrid scalable multi-representation cloud datastore," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 4, pp. 689–702, 2018.

[20] L. Braun *et al.*, "Analytics in motion: High performance event-processing and real-time analytics in the same database," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '15. New York, NY, USA: Association for Computing Machinery, 2015. doi: 10.1145/2723372.2742783. ISBN 9781450327589 p. 251–264. [Online]. Available: https://doi.org/10.1145/2723372.2742783

[21] M. Jibril, A. Baumstark, P. Götze, and K.-U. Sattler, "Jit happens: Transactional graph processing in persistent memory meets just-in-time compilation," in *24th International Conference on Extending Database Technology (EDBT)*, 03 2021. doi: 10.5441/002/edbt.2021.05 pp. 37–48.

[22] T. Mühlbauer, W. Rödiger, A. Reiser, A. Kemper, and T. Neumann, "Scyper: elastic olap throughput on transactional data," in *DanaC '13*, 2013.

[23] A. Boroumand, S. Ghose, G. F. Oliveira, and O. Mutlu, "Polynesia: Enabling effective hybrid transactional/analytical databases with specialized hardware/software co-design," *CoRR*, vol. abs/2103.00798, 2021. [Online]. Available: https://arxiv.org/abs/2103.00798

[24] J. Lee *et al.*, "Parallel replication across formats in sap hana for scaling out mixed oltp/olap workloads," *Proc. VLDB Endow.*, vol. 10, no. 12, p. 1598–1609, Aug. 2017.

[25] R. Barber, V. Raman, R. Sidle, Y. Tian, and P. Tözün, *Wildfire: HTAP for Big Data*. Germany: Springer, 2019.

[26] C. Garcia-Arellano, H. Roumani, R. Sidle, J. Tiefenbach, K. Rakopoulos, I. Sayyid, A. Storm, R. Barber, F. Ozcan, D. Zilio, A. Cheung, G. Gershinsky, H. Pirahesh, D. Kalmuk, Y. Tian, M. Spilchen, L. Pham, D. Pepper, and G. Lushi, "Db2 event store: A purpose-built iot database engine," *Proc. VLDB Endow.*, vol. 13, no. 12, pp. 3299–3312, aug 2020.

[27] R. Barber *et al.*, "Wiser: A highly available HTAP DBMS for iot applications," in *2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, December 9-12, 2019*. IEEE, 2019. doi: 10.1109/BigData47090.2019.9006519 pp. 268–277.

[28] R. Jags *et al.*, "Snappydata: Streaming, transactions, and interactive analytics in a unified engine," ser. SIGMOD '16, 2016. ISBN 0-89791-88-6/97/05

[29] T. Bang, N. May, I. Petrov, and C. Binnig, "Anydb: An architecture-less dbms for any workload," in *11th Conference on Innovative Data Systems Research, CIDR 2021, Virtual Event, January 11-15, 2021, Online Proceedings*. www.cidrdb.org, 2021. [Online]. Available: http://cidrdb.org/cidr2021/papers/cidr2021_paper10.pdf

[30] H. Lang *et al.*, "Data blocks: Hybrid oltp and olap on compressed storage using both vectorization and compilation," in *SIGMOD '16*, 2016.

[31] M. Boissier, R. Schlosser, and M. Uflacker, "Hybrid data layouts for tiered htap databases with pareto-optimal data placements," in *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, 2018, pp. 209–220.

[32] S. Shen, R. Chen, H. Chen, and B. Zang, "Retrofitting high availability mechanism to tame hybrid transaction/analytical processing," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. USENIX Association, Jul. 2021. ISBN 978-1-939133-22-9 pp. 219–238.

[33] M. Athanassoulis, K. S. Bøgh, and S. Idreos, "Optimal column layout for hybrid workloads," *Proc. VLDB Endow.*, vol. 12, no. 13, pp. 2393–2407, Sep. 2019.

[34] J. Arulraj, A. Pavlo, and P. Menon, "Bridging the archipelago between row-stores and column-stores for hybrid workloads," in *Proceedings of the 2016 International Conference on Management of Data*, ser. SIGMOD '16. New York, NY, USA: Association for Computing Machinery, 2016. doi: 10.1145/2882903.2915231. ISBN 9781450335317 p. 583–598.

[35] G. Haas, M. Haubenschild, and V. Leis, "Exploiting directly-attached nvme arrays in dbms," in *CIDR*, 2020.

[36] P. R. P. Amora, E. M. Teixeira, F. D. B. S. Praciano, and J. C. Machado, "Smartltm: Smart larger-than-memory storage for hybrid database systems," in *SBBD*, 2018.

[37] S. Roozkhosh, D. Hoornaert, J. H. Mun, T. I. Papon, U. Drepper, R. Mancuso, and M. Athanassoulis, "Relational memory: Native in-memory accesses on rows and columns," *CoRR*, vol. abs/2109.14349, 2021. [Online]. Available: https://arxiv.org/abs/2109.14349

[38] M. Boissier and D. Kurzynski, "Workload-driven horizontal partitioning and pruning for large htap systems," in *2018 IEEE 34th International Conference on Data Engineering Workshops (ICDEW)*, 2018, pp. 116–121.

[39] J. Yang, I. Rae, J. Xu, J. Shute, Z. Yuan, K. Lau, Q. Zeng, X. Zhao, J. Ma, Z. Chen, Y. Gao, Q. Dong, J. Zhou, J. Wood, G. Graefe, J. Naughton, and J. Cieslewicz, "F1 lightning: Htap as a service," *Proc. VLDB Endow.*, vol. 13, no. 12, pp. 3313–3325, aug 2020.

[40] J. Böttcher, V. Leis, T. Neumann, and A. Kemper, "Scalable garbage collection for in-memory mvcc systems," *Proc. VLDB Endow.*, vol. 13, no. 2, p. 128–141, Oct. 2019.

[41] J. Kim, K. Kim, H. Cho, J. Yu, S. Kang, and H. Jung, *Rethink the Scan in MVCC Databases*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 938–950. ISBN 9781450383431

[42] D. Schwalb, M. Faust, J. Wust, M. Grund, and H. Plattner, "Efficient transaction processing for hyrise in mixed workload environments," in *IMDM@VLDB*, 2014.

[43] F. Funke, A. Kemper, T. Mühlbauer, T. Neumann, and V. Leis, "Hyper beyond software: Exploiting modern hardware for main-memory database systems," *Datenbank-Spektrum*, vol. 14, no. 3, pp. 173–181, 2014.

[44] L. Li *et al.*, "A comparative study of consistent snapshot algorithms for main-memory database systems," *ArXiv*, vol. abs/1810.04915, 2018.

[45] A. Sharma, F. M. Schuhknecht, and J. Dittrich, "Accelerating analytical processing in mvcc using fine-granular high-frequency virtual snapshotting," in *Proceedings of the 2018 International Conference on Management of Data*, ser. SIGMOD '18. New York, NY, USA: Association for Computing Machinery, 2018. doi: 10.1145/3183713.3196904. ISBN 9781450347037 p. 245–258.

[46] L. Li, G. Wu, G. Wang, and Y. Yuan, "Accelerating hybrid transactional/analytical processing using consistent dual-snapshot," in *Database Systems for Advanced Applications*. Cham: Springer International Publishing, 2019. ISBN 978-3-030-18576-3 pp. 52–69.

[47] U. Sirin, S. Dwarkadas, and A. Ailamaki, "Performance characterization of HTAP workloads," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, Apr. 2021. doi: 10.1109/icde51399.2021.00162

[48] K. Alfons *et al.*, "Transaction processing in the hybrid oltp&olap main-memory database system hyper," *IEEE Computer Society Data Engineering Bulletin*, vol. Special Issue on "Main Memory Databases", 2013.

[49] A. Baumstark, M. A. Jibril, and K.-U. Sattler, "Adaptive query compilation in graph databases," in *2021 IEEE 37th International Conference on Data Engineering Workshops (ICDEW)*. IEEE, apr 2021. doi: 10.1109/icdew53142.2021.00027

[50] A. Raza, P. Chrysogelos, A. G. Anadiotis, and A. Ailamaki, "Adaptive HTAP through elastic resource scheduling," in *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 2020. doi: 10.1145/3318464.3389783 pp. 2043–2054.

[51] C. Luo *et al.*, "Umzi: Unified multi-zone indexing for large-scale htap," in *EDBT*, 2019.

[52] H. Saxena, L. Golab, S. Idreos, and I. F. Ilyas, "Real-time lsm-trees for HTAP workloads," *CoRR*, vol. abs/2101.06801, 2021. [Online]. Available: https://arxiv.org/abs/2101.06801

[53] J. Arulraj, R. Xian, L. Ma, and A. Pavlo, "Predictive indexing." arXiv, 2019. doi: 10.48550/ARXIV.1901.07064

[54] R. M. Perera, B. Oetomo, B. I. P. Rubinstein, and R. Borovica-Gajic, "No dba? no regret! multi-armed bandits for index tuning of analytical and HTAP workloads with provable guarantees," *CoRR*, vol. abs/2108.10130, 2021. [Online]. Available: https://arxiv.org/abs/2108.10130

[55] C. Riegger, T. Vinçon, R. Gottstein, and I. Petrov, "Mv-pbt: Multi-version index for large datasets and htap workloads," *ArXiv*, vol. abs/1910.08023, 2020.

[56] Y. Sun, G. Blelloch, W. S. Lim, and A. Pavlo, "On supporting efficient snapshot isolation for hybrid workloads with multi-versioned indexes," *Proc. VLDB Endow.*, vol. 13, pp. 211–225, 2019.

[57] "Ibm db2 event store," last visited: 12.10.2020. [Online]. Available: https://www.ibm.com/de-de/products/db2-event-store

[58] N. May *et al.*, "Sap hana - from relational olap database to big data infrastructure," in *EDBT*, 2015.

[59] N. Hubig *et al.*, "Hyperinsight: Data exploration deep inside hyper," *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017.

[60] R. L. Cole *et al.*, "The mixed workload ch-benchmark," in *DBTest '11*, 2011.

[61] F. Coelho, J. a. Paulo, R. Vilaça, J. Pereira, and R. Oliveira, "Htapbench: Hybrid transactional and analytical processing benchmark," in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*, ser. ICPE '17. New York, NY, USA: Association for Computing Machinery, 2017. doi: 10.1145/3030207.3030228. ISBN 9781450344043 p. 293–304.

[62] Y. Tian, M. Carey, and I. Maxon, "Benchmarking hoap for scalable document data management: A first step," in *2020 IEEE International Conference on Big Data (Big Data)*, 2020. doi: 10.1109/BigData50022.2020.9377937 pp. 2833–2842.

[63] A. Kipf *et al.*, "Scalable analytics on fast data," *ACM Trans. Database Syst.*, vol. 44, no. 1, Jan. 2019. [Online]. Available: https://doi.org/10.1145/3283811

[64] B. Shiyal, "Chapter 9 - synapse link," in *Beginning Azure synapse analytics transition from data warehouse to data lakehouse*. S.l: Apress, 2021. ISBN 978-1-4842-7060-8

[65] A. Raja, K. Manos, P. Danica, and A. Anastasia, "The case for heterogeneous htap," *8th Binnial conference on Innovative Data Systems Reseach (CIDR '17)*, 2017.

[66] M. Pinnecke, G. Campero Durand, D. Broneske, R. Zoun, and G. Saake, "Gridtables: A one-size-fits-most h2tap data store: Vision and concept," *Datenbank-Spektrum*, 01 2020.

[67] A. Raza, P. Chrysogelos, P. Sioulas, V. Indjic, A. C. Anadiotis, and A. Ailamaki, "Gpu-accelerated data management under the test of time," in *CIDR*. Zenodo, Jan. 2020. doi: 10.5281/zenodo.3827490

[68] K. Agrawal, A. Balasubramanian, S. Kamat, and G. P. M. Krishnan, "Scheduling for htap systems on cpu-gpu clusters," 2020. [Online]. Available: https://arjunbala.github.io/wisc-cs839-ngdb20-paper177.pdf

[69] M. Vogt, N. Hansen, J. Schönholz, D. Lengweiler, I. Geissmann, S. Philipp, A. Stiemer, and H. Schuldt, "Polypheny-db: Towards bridging the gap between polystores and htap systems," in *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*. Springer International Publishing, 2021. ISBN 978-3-030-71055-2 pp. 25–36.

[70] P. Kranas, B. Kolev, O. Levchenko, E. Pacitti, P. Valduriez, R. Jiménez-Peris, and M. Patiño-Martinez, "Parallel query processing in a polystore," *Distributed and Parallel Databases*, vol. 39, no. 4, pp. 939–977, Feb. 2021.

[71] D. Butterstein, D. Martin, K. Stolze, F. Beier, J. Zhong, and L. Wang, "Replication at the speed of change: A fast, scalable replication solution for near real-time htap processing," *Proc. VLDB Endow.*, vol. 13, no. 12, pp. 3245–3257, aug 2020.

[72] "Tidb - github repository," last visited: 30.12.2021. [Online]. Available: https://github.com/pingcap/tidb

[73] "Greenplum - github repository," last visited: 31.12.2021. [Online]. Available: https://github.com/greenplum-db/gpdb

[74] "Citus - github repository," last visited: 02.01.2022. [Online]. Available: https://github.com/citusdata/citus

[75] "Hyrise github," last visited: 12.10.2020. [Online]. Available: https://github.com/hyrise/hyrise

[76] SnappyData. Snappydata 1.2.0 - getting started in 5 minutes or less. Last visited: 12.10.2020. [Online]. Available: https://snappydatainc.github.io/snappydata/quickstart/

[77] MemSQL. Memsql documentation. Last visited: 12.10.2020. [Online]. Available: https://docs.memsql.com/v7.1/introduction/documentation-overview/

[78] "Vegito - github repository," last visited: 30.12.2021. [Online]. Available: https://github.com/SJTU-IPADS/vegito

[79] "Poseidon - github repository," last visited: 31.12.2021. [Online]. Available: https://github.com/dbis-ilm/poseidon_core

[80] "Postgresql - github repository," last visited: 31.12.2021. [Online]. Available: https://github.com/postgres/postgres

[81] "Hyper online interface," last visited: 12.10.2020. [Online]. Available: http://hyper-db.de/interface.html

[82] "Umbra online interface," last visited: 01.01.2022. [Online]. Available: https://umbra-db.com/interface/