2019 IEEE 35th International Conference on Data Engineering (ICDE)

Data Management at Huawei: Recent Accomplishments and Future Challenges

Jianjun Chen[*][†], Yu Chen[†], Zhibiao Chen[‡], Ahmad Ghazal[†], Guoliang Li[‡], Sihao Li[‡], Weijie Ou[‡]
Yang Sun[†], Mingyi Zhang[†], Minqi Zhou[‡]

[†]*Huawei America Research*, [‡]*Huawei Technologies Co., Ltd.*

***Abstract*—Huawei is a leading global provider of information and communication technologies (ICT) infrastructure and smart devices. With integrated solutions across four key domains: telecommunication networks, IT, smart devices, and cloud services, Huawei is committed to bringing digital transformation to every person, home and organization for a fully connected and intelligent world. Founded in 1987, Huawei currently has more than 180,000 employees, and operates in more than 170 countries and regions with revenue over 100 billion USD in 2018.**

Data management plays a key role in all of the four key domains above. We have developed innovative products and solutions to support rapid business growth driven by customer requirements. While many data management problems are common, each domain also has its own special requirements and challenges. In this paper, we will go through recent advancements in Huawei data management technologies including a petabyte scale enterprise analytics platform (FusionInsight MPPDB) and a highly available in-memory database for telecommunication networks (GMDB). In addition, we discuss data management challenges that we are facing in the areas of autonomous databases and device-edge-cloud collaboration data platforms.

***Index Terms*—Distributed Transaction Processing, Multi-Model, Auto-Tuning, HTAP, MPPDB, Autonomous DB, Distributed Data Collaboration**

I. Introduction

Founded in 1987, Huawei started as a provider of telecommunication equipments and software solutions. Today, Huawei is a leading global provider of information and communication technologies (ICT) infrastructure and smart devices. With integrated solutions across four key domains: telecom networks, IT, smart devices, and cloud services, Huawei is committed to bringing digital transformation to every person, home and organization for a fully connected, intelligent world. In addition, Huawei is also branching out to new business opportunities in emerging industries such as IoT and autonomous vehicles etc.

Data management is a key to Huawei's business areas and critical in Huawei's vision to provide the full stack of technology for its customers to meet their requirements. In this paper, we provide an overview of some recent advancements in Huawei data management technologies. In addition, we discuss challenges in the data management areas which we are currently facing.

[*] Dr. Jianjun Chen is the corresponding author, jianjun.chen1@huawei.com

Huawei developed the Fusion Insight MPP database system (FI-MPPDB) [10] that adopts a shared nothing massively parallel processing architecture. It supports petabyte scale data warehouses and runs on hundreds of machines. It was originally adapted from Postgres-XC [3] and supports ANSI SQL 2008 standard. Common features found in enterprise grade MPPDB engines have been added through the years, including hybrid row-column storage, data compression, vectorized execution etc. FI-MPPDB provides high availability through smart replication scheme and can access heterogeneous data sources including HDFS.

FI-MPPDB reduces the overall cost of ownership for customers compared to purchasing DBMS products from other commercial vendors. We use a full stack approach which allows us to dictate product directions and to provide a quick solution that meets our customer requirements. The first commercial version of FI-MPPDB was launched in 2015 and has been adopted by many customers over the globe, including some of the world's largest financial institutes in China. The success of FI-MPPDB leads to its recognition by the Gartner group with our technology included in its magic quadrant since 2016 [16]. In 2017, we announced our first version of FI-MPPDB on Huawei public cloud, a.k.a. LibrA. More detail about FI-MPPDB and LibrA can be found in [10].

In this paper, we provide quick updates of three most recent features that we have partially completed in FI-MPPDB.

- **Hybrid Transactional Analytical Processing (HTAP):** FI-MPPDB customers may want to run an OLTP workload occasionally without maintaining OLTP databases. Our FI-MPPDB is originally designed to support complex OLAP queries without consideration to OLTP transactions. Due to this requirement, we have been working on extending our MPPDB to handle mixed OLAP and OLTP workloads. One major area that we have been working in is to enhance distributed transaction processing in MPPDB to support large number of short transactions in OLTP workloads. The detail about our new distributed transaction processing algorithm (a.k.a. GTM-lite) is discussed in Section II-A.
- **Multi-model:** Data analytics have become more versatile in recent years. As an example, some of our FI-MPPDB customers in the IoT area require real-time analytics over time-series data and perform graph data analysis.

These are common requirements for handling financial risk analysis, network management, and customer churns etc. It is desirable to have an integrated system to handle all the analytics requirements. Hence, we extend our FI-MPPDB to support multiple data models. This is discussed in Section II-B.

- **Learning-based Query Optimizer:** Our recent accomplishments also include auto tuning for query processing. Auto tuning captures actual execution statistics and reuses them for the subsequent same or similar query plan generation. Section II-C describes this innovation in our FI-MPPDB.

Besides general IT database products, we also support data management in Huawei telecommunication products (CT). In this paper, we briefly describe how we handle data management of user billing data and session data. A requirement of the combination of low latency (*e.g.,* micro-seconds), high availability (*e.g.,* greater than 99.99% availability) and low resource consumption imposes tough challenges. By making trade-offs in durability and data consistency, we developed a product named GMDB. It is a distributed in-memory database system that supports both key-value and SQL APIs for our CT solutions. As a result, a single server using GMDB can support billing of millions of subscriber accounts. In recent years, network function virtualization (NFV) moves to cloud based infrastructure to accelerate the business innovation of customers. GMDB supports online schema changes to guarantee no system downtime during schema upgrade. The detail is discussed in Section III-B.

Moving forward, we are planning to make our data management more intelligent, *e.g.,* using machine learning algorithms to make the data management autonomous. Section IV-A presents our initial work and vision in the area of autonomous database management.

Over the years, IoT devices have become very popular in our day-to-day lives and many industry fields, such as the smart home and highway traffic monitoring system. One particular interesting case is autonomous vehicle, where hundreds GB and even TB data can be generated on a daily basis. The world is becoming more increasingly connected, and the requirement of accessing data anywhere and anytime becomes significant. A general trend of distributed computing paradigm is moving computation toward edge devices, where most data is generated, to reduce latency and network bandwidth consumption. Our vision and initial work in building a distributed data management platform that supports seamless devices, edge and cloud collaboration is described in Section IV-B.

The rest of this paper is organized as follows. Section II describes our recent advancements in FI-MPPDB, including a new distributed transaction processing algorithm, the multi-model, and the auto-tuning feature. Section III presents an overview of GMDB and a description of its online schema update. We discuss future challenges and research problems in the areas of building autonomous databases and distributed data collaboration platforms in Section IV. We conclude our paper in Section V.

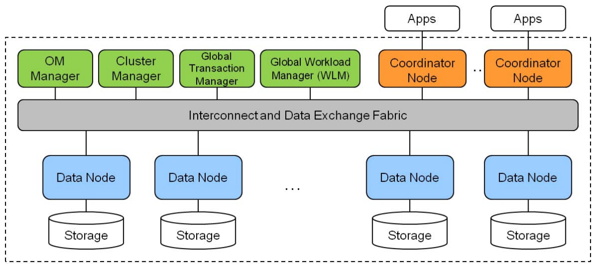


Fig. 1. FusionInsight MPPDB system architecture

II. Recent Accomplishments in FI-MPPDB

FI-MPPDB is Huawei's flagship data management product in the IT area. It has been commercially used for several years and contains many innovative features. In this section, we provide a brief overview of FI-MPPDB first, then present a set of key features that are developed recently.

Figure 1 illustrates a high-level architecture of the FI-MPPDB system. FI-MPPDB scales linearly to hundreds of physical machines and handles a wide spectrum of interactive analytics. In the system, data are partitioned and stored in data nodes, and the data nodes maintain the local ACID properties. Cross-partition consistency is maintained using the two phase commit protocol and global transaction management. FI-MPPDB supports both row and columnar storage formats. Our vectorized execution engine is equipped with latest SIMD instructions for fine-grained parallelism. Query planning and execution are optimized for large scale parallel processing across hundreds of servers. They exchange data on-demand from each other and execute the query in parallel [10].

A. HTAP/GTM-lite

1) Background and Motivation: HTAP is a system that combines the power of transactional system and analytical system for real-time decision support. Gartner group [16] first introduced this concept in 2014 report. The key advantages of such systems are eliminating the analytic latency and data movement across OLAP and OLTP database management systems. This enables the business to make quick decisions on recent data. In addition, a single HTAP system also reduces the cost and operational overhead compared to maintain two separate OLAP and OLTP databases. Many Huawei's businesses can benefit from using a HTAP system, including real-time operational reporting, fault and fraud detection and campus security monitoring etc.

A few existing systems offer such capability. SAP HANA [28] offers a common database approach for OLTP and OLAP using in-memory column database. Oracle Exadata follows suit, adding in-memory column option to enhance OLAP capability. Peloton [8] designs an innovative database engine that enables support of varied workloads on the same database without sacrificing the performance benefits of specialized systems. Its execution adapts to workload and continuously evolves its physical data storage format by analyzing the query pattern. Companies with strong engineering capability can also build loosely-coupled system based on lambda [2] architecture. To meet the need of HTAP, Huawei started enhancing transactional capability on top of FI-MPPDB. In this section, we discuss the effort of eliminating bottleneck in interaction to the global transaction manager.

In distributed systems such as PG-XC[1], applications interact with a sharded OLTP system by sending queries or statements to coordinator nodes (CN). CN interprets those statements and determines which database shard or data node (DN) contains relevant data. Those statements are then sent to specific DNs for query processing, including actual access to stored data. A global transaction manager (GTM) generates ascending global transaction ID (XID) for transactions and dispatches snapshots consisting of a list of current active transactions. Based on the global XID and snapshot, all transaction in the distributed system are ordered and global read consistency is guaranteed. Two-phase commit (2PC) is used to support atomic write operation across nodes.

The synchronization cost of using GTM in a sharded OLTP system is high if each transaction needs to communicate with GTM multiple times during execution. Many of those communications and interactions, such as acquiring GXID and enqueue/dequeue itself from GTMs active transaction list, are serialized and can limit overall throughput. As a result, GTM often becomes the system performance bottleneck when many database shards exist in the system.

In many practical application systems, database is designed with application sharding in mind and the majority of transactions in such systems are single-sharded. We want to take advantage of this property in designing distributed transaction management facility.

2) GTM-Lite Design Principles: Assuming majority transactions are single-sharded in typical OLTP workload, we propose GTM-lite distributed transaction management to improve throughput and scalability of sharded OLTP by avoiding centralized concurrency management overhead for single-shard transactions. The goal is to speed up majority workloads (i.e. single-shard queries) and to scale out sharded OLTP system without GTM as a bottleneck.

The main idea of GTM-lite is as follows: since the target is to optimize the performance of single-shard transactions and maintain consistent access of multi-shard transactions, we avoid sending single-shard transactions to GTM and GTM is only responsible for multi-shard transactions consistency:

- Single-shard transaction: CN sends transaction to DN, then DN uses local XID and local snapshot to execute and commit transaction locally.
- Multi-shard transaction: CN gets global XID (GXID) and global snapshot from GTM, and when the transaction gets to DN, it obtains local XID and local snapshot as well, and finally DN uses merged snapshot to execute, and 2PC commit.

The advantage of this design is: 1) GTM is only responsible for generating XID and snapshot for multi-shard transactions, hence its workload will be largely reduced. 2) The majority local transactions do not need many-round communication with GTM, which significantly increases the throughput. Given that there are 10% or less multi-shard transactions in common OLTP workloads, the use of more complicated logic to guarantee consistency-read is justified.

Although the above protocol looks simple and can be easily implemented, there are a few complications that need to be handled, e.g., the visibility checking for multi-shard transactions given both local and global snapshots. Because the timings of getting global and local snapshot are different, their view of visibility may contain conflicts. We identify a couple of anomalies shown below:

Anomaly1: When global snapshot tells one transaction is committed, but local snapshot tells it is active (prepared but not committed).

In our system, transactions are marked committed in GTM first and then on all nodes. Therefore, there is a time window when a reader gets global snapshot after the writer transaction commits on GTM, but the reader gets a local snapshot before the commit confirmation reaches one data node. To ensure reader gets all data committed by the writer on every data node, the reader has to wait for the writer to complete its commit process on data nodes. We call this wait-for-commit process UPGRADE since it is upgrading writer from active to committed state.

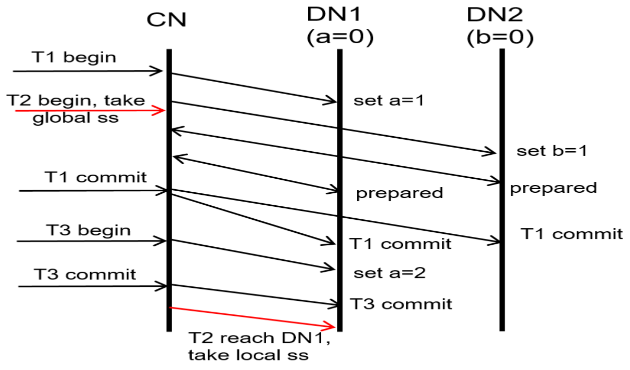


Fig. 2. Anomaly2 scenario

Anomaly2: When global snapshot says a writer is active (taken earlier), but local snapshot says it is committed (taken later).

The anomaly is illustrated in Figure 2:

- T1, T3 are two transactions from the same session, executing sequentially. T1 does multi-shard write on DN1 (set a=1) and DN2 (set b=1). Then T3 does single-shard write on DN1 (set a=2).
- T2 is a multi-shard reader, getting an old global snapshot (when T1 has not committed), and a new local snapshot on DN1 (when both T1 and T3 already committed).

Specifically:

- In T2's global snapshot $\{T1\}$: T1 is still active
- In T2's local snapshot $\{\}$: both T1, T3 committed locally

- When T2 reads tuples in the following table: T1 deletes tuple1 and inserts tuple2; T3 updates tuple2 and inserts tuple3. Based on its snapshots, T2 will think tuple1 and tuple3 are both visible, so T2 will read T3 update but not T1s, which causes anomaly.

	Xmin	Xmax	Data(a)	Visibility
tuple1	-	T1	0	yes
tuple2	T1	T3	1	no
tuple3	T3	-	2	yes

To solve this anomaly, the data dependency between two writers T1 and T3 is the key. T2 cannot view T1's update on data (i.e. a), therefore it should NOT see subsequent update on data (i.e. a) done by T3. Instead, reader should ignore any local commits that is dependent on uncommitted global writes. We call this reverse-local-commit process DOWNGRADE because it downgrades local commits to active state in readers snapshot.

Note that UPGRADE is done by readers pause and wait for local commit finish, with a slim window between PREPARE and COMMIT confirmation. However, DOWNGRADE does not require physical reverse of local commits. Reader just adjusts its snapshot to update its visibility.

Algorithm 1 summerizes how we merge given global and local snapshots. The downGradeTX and upgradeTX are two procedures to resolve the conflicts. After merge is done, the reader transaction will use the result as visibility criteria to access data tuples.

Algorithm 1 MergeSnapshot

```
Input: global snapshot, local snapshot, local commit order on each DN (LCO), xidMap
Output: mergeSnapshot to be used for visibility checking
1: For all active transactions txn in global snapshot
2:     If txn is found in xidMap, insert mapped local xid into mergedActiveTxMap
3: For all active transactions txn in local snapshot
4:     Insert txn xid into mergedActiveTxMap
5: Do downgradeTX to traverse LCO and find local committed txns that is invisible in global snapshot, make them appear "active" in mergedActiveTxMap;
6: Do upgradeTX for locally active, but globally visible txns to make them appear "committed" in mergedActiveTxMap;
7: adjust mergedXmin and mergedXmax in mergeSnapshot;
8: for all active transactions txn in mergedActiveTxMap
9:     insert txn into active transaction list in mergeSnapshot;
10: return mergeSnapshot;
```

Performance: The goal of GTM-lite is to improve the scalability of OLTP system, so we deployed the database on various cluster sizes from 1 node, 2 nodes, 4 nodes up to 8 nodes. We modified the TPC-C benchmark to issue 100% single-shard (SS) or 90% single-shard transactions (MS). As shown in Figure 3, GTM-Lite achieved higher throughput and scaled out much better than baseline. It performed better in 100% single-shard workload (SS) because there is no centralized coordination and each shard is responsible for its partition of the workload.

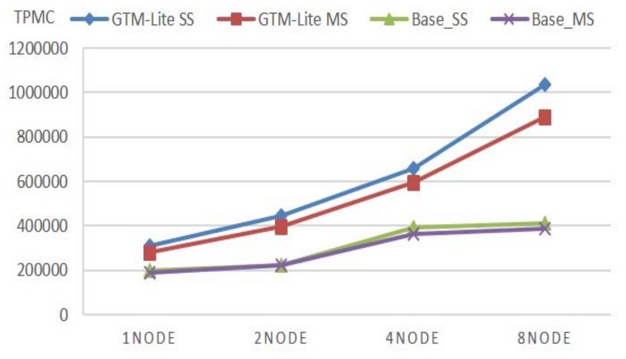


Fig. 3. GTM-Lite scalability

3) GTM-Lite Related Work: NoSQL databases typically do not require strong data consistency and may allow stale data to be read. Traditional RDMS such as Oracle and MS SQL Server use a commit timestamp mechanism as a basis for achieving a consistent view of the data. For example, in Oracle distributed systems, the process coordinating all nodes will collect SCNs from all participants, choose the highest one, and use the same SCN on all nodes. To provide data consistency, PostgreSQL makes use of snapshots as concurrency control mechanism. For Postgres-XC, Postgres-XL, and MPPDB, this is extended cluster-wide via a Global Transaction Manager (GTM), making transaction identifiers and snapshots global.

VoltDB [4] or H-Store [32] divide database data into partitions. Queries accessing data in multiple partitions are processed as multi-partition transactions. The commands from a multi-partition transaction need to complete atomically so a partition stalls between multi-partition transaction prepared and committed state.

Spanner [12] enables globally-consistent read and externally consistent writes across database, which may be distributed to multiple data centers. In Spanner, a Truetime approach is used to maintain a global synchronous timestamp and expose clock uncertainty between wall times of individual servers. Spanner uses a commit-wait mechanism before data become visible to overcome any uncertainty in time between the nodes. Concurrent read-only transactions get a consistent, but possibly slightly stale view of the data.

B. Multi-model Database System

1) Motivation and Scope: Recent big data advancements provided access to large volume of data with different varieties like relational tables generated by transnational database systems, time series data acquired from sensors, spatial coordinates generated by GPS systems and images from satellite and other sources like cameras. The varieties of these data mandate a need for a system that is able to store, process and query these different data sources in a unified manner. One solution is to use multiple systems to serve each data type (relational engine, graph engine etc). In addition to being costly, having separate data analytics product for each model has a couple drawbacks. First, it requires customers to learn multiple systems. Second, a single application often need to query multiple data sources in different native models (e.g. data-lake like environment). The multi-system solution is not expected to perform since data need to be moved around for analytics applied on different data sets. In our work, we extended FI-MPPDB to be a multi-model database system (MMDB) [38] to handle data sources with various formats.

Autonomous vehicle is a good example that can be used to illustrate the need of an MMDB. A lot of data with different formats are generated and used, which includes sensor data from vehicles and radars, images from cameras and data from knowledge bases. Autonomous vehicles need analysis of all these data in real time and one practical way to accomplish that is for all these data sources to be stored and processed by a single data management engine.

Our MMDB is expected to provide the capabilities below using the autonomous vehicles problem as an example.

Spatial-Temporal Synthesized Processing: An MMDB needs the capability to synthesize all kind of information in terms of spatial-temporal properties across various data types. Normally, such data with spatial-temporal information is generated from sensors, e.g. temperature and humidity sensors, with high frequency. Furthermore, the system should be able to support high ingestion rate for time-series data, and computation-intensive spatial-temporal algorithms.

Vision Data Processing Based on Metadata: High resolution cameras, lidar (light detection and ranging), and radar produce a lot of data related to autonomous cars and other systems. Sophisticated AI based algorithms have been developed to reorganize objects in vision or point cloud data . A multi-model system needs to store these objects and process queries on them. The storage of these objects requires special indexing and proper metadata for analysis and performance.

Domain Specific Knowledge Processing in Graph: Graph data are another common source of data for autonomous vehicles. Correlated rules for driving (rules and their relationships) can be represented as a graph. GPS and traffic data are spatio-temporal information that can be represented by graphs as well. Multi-model systems should be able to efficiently store and query such graphs.

Integrated Query Processing across Models: Autonomous vehicles may need to analyze data across different sources such as weather information in relational format and GPS data in graph format. An MMDB is expected to efficiently and seamlessly process these diverse types of data.

2) Our Multi-Model Database Architecture: Previous section describes the business and user requirements related to multi-model systems. In this section, we present the architecture and high level technical description of our solution. Figure 4 shows the architecture of our MMDB which consists

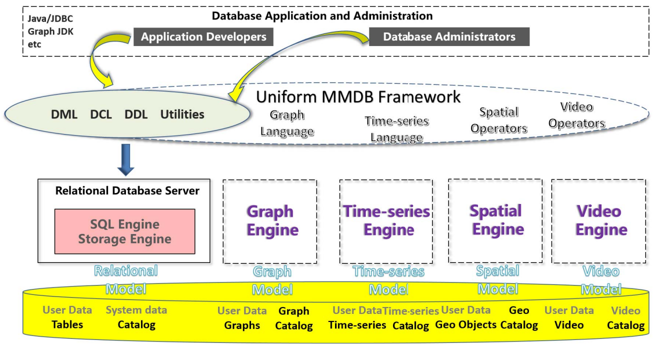


Fig. 4. Architecture of our multi-model database

of three subcomponents: a unified storage engine, multiple runtime execution engines, and a uniformed framework.

Unified Storage Engine: We employ a relational model for storing data in different formats given that RDBMSs are mature and widely used. Also, a lot of the incoming data are already in relational form and fit easily in our system. Other types of data need to be converted or represented as relational. For example, graphs are represented through tables for vertexes and edges. Metadata of graph, vision, spatial time-series object are also stored in relational tables.

Integrated Multiple Runtime Engines: To enhance the performance of query processing, we provide multiple execution engines, with the relational engine as the main one. We support the graph engine, time-series engine, spatial engine now, and plan to add the vision engine soon. All these engines are highly integrated, leveraging on a set of light-weighted hooks into the main execution engine. We also change the planner to produce a single plan that covers all the different executions and their flow into and from each other.

Uniformed Framework: Our MMDB works as a single database system with uniformed interface to accept queries and other common databases operations like DDL, DML, DCL and system utilities. We use the traditional relational database system connections, such as ODBC, JDBC and etc. We have our SQL extension to handle processing of other data sources. We integrate two languages in our SQL extensions: the Gremlin language which is used in graph traversal and a continuous query language used in streaming processing. In the future we plan to add a language for processing vision data.

3) Discusstions: A few commercial databases start to provide such capabilities. Microsoft Azure CosmosDB [25] provides support to key-value, graph and document data. Oracle 18c [27] supports relational, spatial and graph data. Our MMDB is capable of storing and processing various data models including relational, graph and key-value. In addition, our MMDB is able to synthesize all the information across multiple runtime execution engines in terms of spatial-temporal correlation, and across interaction in terms of materialized knowledge, based on an extended relational data model.

Our MMDB provides a new model for data definition which extends traditional relational model. On top of the relational model, we support mappings from entities and attributes to value sets in tables, and the access path from tables to tables through graph. Our model is more like the entity-relational model [11]

With our current development, our MMDB is able to support query as follows:

Example 1: Query in Unified Language

```
with cars (carid) as (select * from
        gtimeseries(select time, carid, juncid
                from high_speed_view
                where now() - time ≤ (30 minutes))),
    suspects (cid) as (select * from
        ggraph(g.V().has(cid,11111).inE(call).has(time,
gt(2018\6\1)).count().gt(3)),
select cid, phone, photo, carid
from suspects s, cars c
where s.cid =
    (select cid from car2cid as cc where cc.carid=c.carid)
```

In this query, we integrate a graph query written in Gremlin (line 6) and a time-series (line 2) into a relational query. The graph query and the time-series query are encapsulated using a table expression in SQL.

C. Learning-Based Query Optimizer

Our MPPDB optimizer is based on Postgresql optimizer with fundamental enhancements to support MPP and complex OLAP workloads. The MPP extension builds MPP plans and applies cost based optimizations that account for the cost of data exchange. In addition, the optimizer is generalized to support planning and cost based optimizations for vector executions and multiple file systems including Apache ORC file format. Query rewrite is another major ongoing enhancement to our optimizer, including establishing a query rewrite engine and adding additional rewrites which are critical to complex OLAP queries.

The enhancements mentioned above are common in other commercial database and big data platforms. We believe that we are closing the gap with those established products. In addition, to make our optimizer competitive, we are working on cutting edge technology based on *learning*. Our initial vision of a learning optimizer is in the area of cardinality estimation (statistics), which is one of the core components of cost based optimization. Traditional and classical methods of statistics used for cost based optimizations are complex, require large capital investment, and evolve through time. Different sources of data with their different formats pose additional challenges to collecting statistics with reasonable accuracy.

Based on the above challenges in the optimizer statistics area, we are developing a learning component in the DBMS

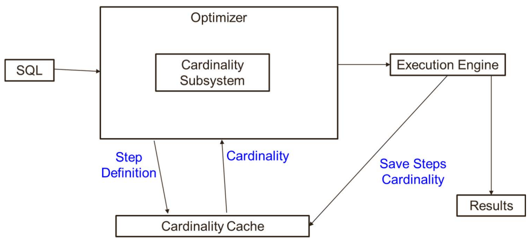


Fig. 5. Statistics learning architecture

that selectively captures actual execution statistics. The captured data in turn can be used by the optimizer for accurate statistics estimates for subsequent similar queries. Some earlier work and proof of concept in this area were presented in [10]. Recently, we moved from the early proof of concept to building an actual learning system.

Figure 5 is a high level illustration of our statistics learning approach. The new architecture has two sub-components: capturing execution plans (producer) and re-using them by the optimizer (consumer). Based on user settings/directives, the producer selectively captures execution plan into a plan store. The plan store schema is modeled after the plan execution steps. Steps that affect cardinality (scans, joins, aggregation steps, set operations and limit operator steps) are captured with the estimated and actual row counts. To make the store more efficient and useful, the executor captures only those steps that have a big differential between actual and estimated row counts.

The optimizer gets statistics information from the plan store and uses it instead of its own estimates. This is done through an API call to the plan store which is modeled as a cache. The key of the cache is an encoding of the step definition described below. The use of steps statistics is done opportunistically by the optimizer. If no relevant information can be found at the plan store, the optimizer proceeds with its own estimates.

We developed an innovative step definition form used by both the producer and consumer. The step definition is done as a prefix expression representing the logical operator and its operand(s). Only the logical operator (join instead of hash join or scan instead of index scan) is needed since our goal is store and re-use cardinality information which is independent of the actual physical operator used in the execution. The step definition for an execution operator captures the whole query tree underneath the operator. For future executions of the same or similiar queries, the optimizer can find the cardinality through matching the full definition of steps. This is more accurate than just relying on predicates or group columns alone. At the same time it has restricted scope of same or similar queries. However, we believe that reporting workloads (canned queries) are the most common in real life OLAP workloads.

We illustrate our logical step definition through an example. Assume that the SQL query ”select * from OLAP.t1, OLAP.t2 where OLAP.t1.a1=OLAP.t2.a2 and OLAP.t1.b1 $>$ 10” is

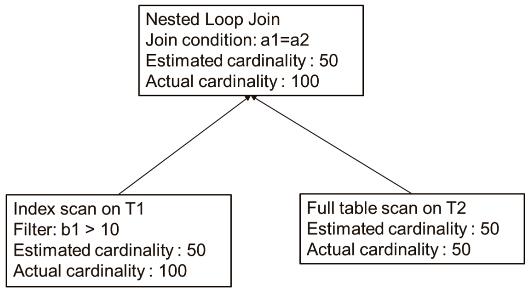


Fig. 6. Two-way join execution plan

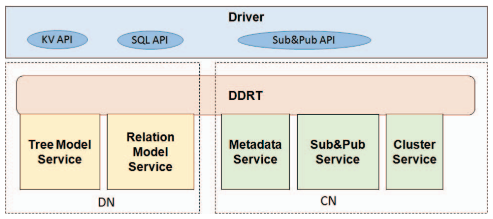


Fig. 7. GMDB system architecture

executed with a plan shown in figure 6 which is simply a nested loop join between an index scan on t1 and a table scan of t2. The producer in our model captures the scan on t1 and the join steps since their estimate is off the actual cardinality. These steps form two rows in the step store. A partial representation of these two rows is shown in Table I with the step text, estimated and actual cardinalities. Note that the join step entry specifies the full definition of the children to make sure it is used only for joins between t1 and t2 with the same scan predicate on t1. We apply some order on predicates and on join children to make the saved information applicable regardless of join or predicate order.

Step text could be huge for complex queries and we avoid the potential overhead of saving and retrieving of such complex text by using the MD5 hash value (32 bytes) of the step text. A hash collision results in possibly an incorrect cardinality. The odds of that scenario is very low and it is a lot less than the odds of the optimizer getting the wrong estimate.

Auto-Tuning Related Work: Early known work on database auto tuning include Database Tuning Advisor [5] from Microsoft SQL Server and LEO [18] from IBM DB2 system. IBM DB2 LEO [18] is a learning based optimizer that attempts to improve cardinality estimation related to predicate selectivities and column correlations. Our work differs with LEO in two major aspects. First, we cover all cardinality impacting operators (scans, joins, aggregations and set operations) and LEO does not. The other major difference between ours and LEO is that our model is based on logical definition of operators that describe it and its source(s). IBM LEO use history of predicate selectivities and column correlation in isolation of how they are used in queries. This provides more usage but less accurate as well. However, as we mentioned before, our focus is on reporting workloads (canned queries) and exact match works well for these cases.

Using machine learning in query optimization, performance modeling, and system tuning has increasingly drawn attention to the database community [7], [36]. Recent works [19], [23], [24] use machine learning methods to estimate cardinality for various database operators. The approach using machine learning technique is promising as it allows the system to learn the data distribution and then estimate the selectivity or cardinality. In contrast, the traditional statistics based selectivity estimation has its limitation on the information a system can store and use. However, we need to carefully assess the benefits of adopting machine learning based approaches in commercial database query optimizer as they may require significant system architecture changes. For example, [23], [24] use neural networks to model selectivity functions which require significant training to approximate the functions. In addition, the training set takes considerable system resources to generate.

III. Recent Accomplishments in GMDB

In this section, we provide an overview of GMDB, which is a Huawei's data management product in the CT area. The product has been commercially used for a few years. Due to the limit of space, we only present a brief system overview and the feature of online schema evolution that is developed recently.

A. GMDB Overview

The evolution of traditional telecommunication network equipment software from embedded architecture to cloud native architecture is the trend of telecommunication equipment software development. This trend requires telecom software to become decoupled from hardware and also separate business logic from data to achieve flexible deployment, highly available online services and elastic expansion.

GMDB is a distributed in-memory database that provides low-latency, high-throughput, elastic expansion and high-availability. It is made to meet the needs of cloud computing evolution of telecom software, while ensuring carrier-class experience and high availability requirements.

In designing GMDB, we made conscious decisions on choosing low latency, high availability and cost effectiveness over reliability and functionalities. First, since limited cases of data loss can be compensated through application logic, GMDB only asynchronously flush data to disk periodically. Second, the storage engine of GMDB achieves great performance by adopting light-weight fiber threads with a lock-free protocol to avoid the overhead of concurrency control. Each fiber is also allocated to a dedicated physical CPU core to maximize the resource utilization [17]. Third, GMDB only supports transactions on single objects and covers a subset of the ANSI SQL (only those needed for the use cases).

Figure 7 shows GMDB architecture. Driver is GMDBs client interface that provides the KV (key value) interface of the tree (object) model, the SQL interface of the relational

TABLE I
LOGICAL CANONICAL FORM

Step Description	**Estimate**	**Actual**
SCAN(OLAP.T1, PREDICATE(OLAP.T1.B1 $>$ 10))	50	100
JOIN(SCAN(OLAP.T1, PREDICATE(OLAP.T1.B1 $>$ 10)), SCAN(OLAP.T2), PREDICATE(OLAP.T1.A1=OLAP.T2.A2))	50	100

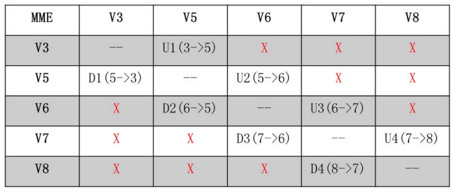
model, and the pub/sub interface. Distribute database Runtime (DDRT) is the unified distributed database operation framework that provides serviceability to support dynamic deployment and management of various services. The coordinator node (CN) manages global unified metadata management, cluster management and other services. Data nodes (DN) support multiple data model storage services such as the tree object model and the relational model.

B. *Online Schema Evolution*

This section focuses on online schema evolution in GMDB which is a key feature to achieve high availability for managing user session data. Session data management has a direct impact on user's phone and networking services and hence is very sensitive to the system availability. GMDB provides the schema evolution technology to provide the In Service Software Upgrading(ISSU) for session data management. Many research papers have been published in schema evolution in the last twenty years. More recently, Google presented an online asynchronous schema change algorithm in F1 [29]. Several database vendors such as Microsoft [13] also started to integrate partial schema evolution in their commercial products. [18] introduced end-to-end support of co-existing schema versions within one database, similar to what we implemented on the GMDB.

Typical volume of a single user session data is about 5-10KB and is represented as a tree-modeled object in a JSON format and stored in our KV store. In the following we give a quick overview of the GMDB object model. Each object has a record schema like a RDBMS table. However, unlike relational data model, related data of multiple tables with a key/foreign key relationship can be organized and stored together in a tree format. A record can contain multiple fields. Each field can be either a primary data type, or a record type with an array of records. A primary key is defined to uniquely identify a root record.

In a telecom network, various applications can co-exist with different schema versions. One simple but inefficient solution is to store data in separate copies with different schema versions, which wastes storage resources and also causes more complexity in operation and maintenance. Figure 8 shows the upgrading/downgrading matrix for the Mobility Management Entity(MME), which is a typical network function in LTE. For example, the upgrading of MME from V3 to V5 to support a new feature requires more fields to be added in the session data. In cased of a failed schema upgrade, schema downgrade can happen during rollback.



MME	V3	V5	V6	V7	V8
V3	--	U1(3->5)	X	X	X
V5	D1(5->3)	--	U2(5->6)	X	X
V6	X	D2(6->5)	--	U3(6->7)	X
V7	X	X	D3(7->6)	--	U4(7->8)
V8	X	X	X	D4(8->7)	--

Fig. 8. An example of multiple schema conversions in MME versions

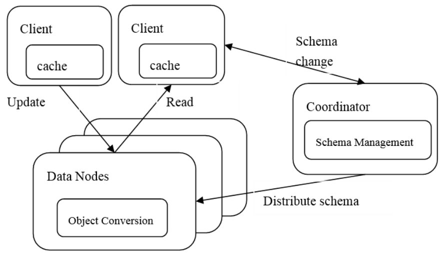


Fig. 9. Schema change and data flow in GMDB

Figure 9 shows the schema change and data access flow in GMDB. In summary, GMDB schema change is handled by the Coordinate Nodes (CNs). For example, when a client submits a new schema S to the CNs, CNs validate S and dispatch it to Data Nodes (DNs). A client sends a query or DML statement directly to DNs without involvement of CNs. Each client has a local data cache in its own schema version to reduce latency. While DNs only store one copy of data, different GMDB clients may be running applications with different schema versions. GMDB allows objects stored in the DNs to be read by a client with a different schema version. This is done by dynamically converting objects from the DN schema version to the requesting client's schema version before returning data from the DNs to the client. If the object is read with a newer schema version, we call it upgrade schema evolution. If the object is read with an older schema, we call this downgrade schema evolution.

Figure 10 shows a simple schema evolution example where two clients have copies of the same data with different schemas. Client X creates an object D {id = 'Jane'} with schema S {'id': string}, sends it to DN and keeps a copy in its local cache. The object D is stored in the DN with schema S and client X subscribes to future changes of D with schema S. Next client Y registers a new version S' {'name': string, 'age': number} of schema S. Client Y reads and subscribes to object D with S'. The DataNode transforms D to D' following schema change S to S', and sends D' to client Y.

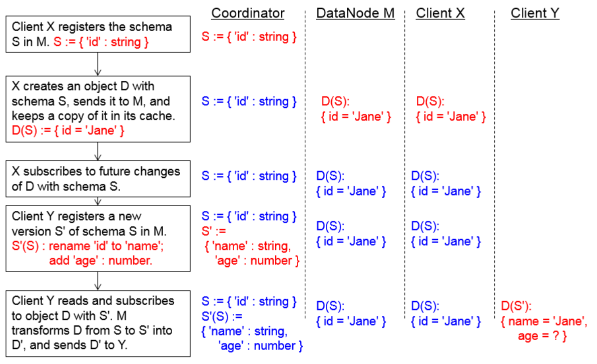

Coordinator
DataNode M
Client X
Client Y
Client X registers the schema S in M. S := { 'id' : string }
S := { 'id' : string }
X creates an object D with schema S, sends it to M, and keeps a copy of it in its cache. D(S) := { id = 'Jane' }
S := { 'id' : string }
D(S): { id = 'Jane' }
D(S): { id = 'Jane' }
X subscribes to future changes of D with schema S.
S := { 'id' : string }
D(S): { id = 'Jane' }
D(S): { id = 'Jane' }
Client Y registers a new version S' of schema S in M. S'(S) : rename 'id' to 'name'; add 'age' : number.
S := { 'id' : string } S' := { 'name' : string, 'age' : number }
D(S): { id = 'Jane' }
D(S): { id = 'Jane' }
Client Y reads and subscribes to object D with S'. M transforms D from S to S' into D', and sends D' to Y.
S := { 'id' : string } S'(S) := { 'name' : string, 'age' : number }
D(S): { id = 'Jane' }
D(S): { id = 'Jane' }
D(S'): { name = 'Jane', age = ? }

Fig. 10. A schema evolution example

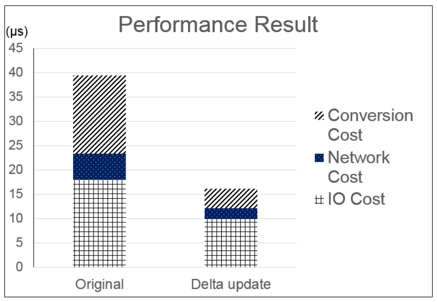


Fig. 11. GMDB online schema evolution experiment results

There are some limitations in the allowed schema changes for the online schema evolution. Deleting and re-ordering fields are two major cases that are not allowed. One factor that makes GMDB online schema evolution simpler is that GMDB only supports transactions on single object level.

Another thing worth mentioning in our system is that data updates and schema evolution happen on delta objects instead of whole objects. Similar is true when syncing data between clients and DNs. Such an approach achieves better performance and consumes less network bandwidth. Figure 11 shows performance results with real MME data in virtualized Linux clients and servers (3.0 GHz CPUs) connected through a 10Gbps network.

IV. Future Challenges

In the previous sections, we overviewed our recent work in the area of Huawei's data management. In addition to that, we also actively work on new challenges and solve new problems. In this section, we will discuss the challenges we are facing, potential research problems, and our vision for Huawei's future data management.

A. Autonomous Databases

1) Background and Related Work: Enterprises require capabilities of managing and achieving *service level agreements* (SLAs), which are typically signed between the service provider and service consumers to guarantee the business success. Specifically, in database systems, SLAs can specify the requirements of a system's performance, such as averaged transaction response time, system throughput and the system's availability (*e.g.,* 99.99%). A challenge is that the workloads are dynamic, and as a result, it is virtually impossible for database administrators (DBAs) to manually adjust the database configurations to meet the targeted SLA. To achieve the SLA, we are adding self-managing capabilities to our FI-MPPDB, which include *self-configuring, self-optimizing, self-protecting, and self-healing* properties [20].

Self-configuring means that autonomous database management systems (ADBMSs) are able to automatically configure its components to adapt to dynamically changing environments. This property allows the addition and removal of system components or resources without system service disruptions. Self-optimizing means ADBMSs automatically monitor and tune the resources to ensure optimal functioning with respect to the given SLA. This capability results in the initiation of a change to the system proactively in an attempt to improve performance or quality of service. Self-protecting means ADBMSs are able to proactively identify and protect themselves from arbitrary activities. This property enables ADBMSs to recognize and circumvent data, privacy and security threats, and facilitates the system to protect itself from physical harm. Self-healing means ADBMSs are able to recognize and diagnose deviations from normal conditions and take action to normalize them. This property enables an ADBMS to proactively circumvent issues which could cause service disruptions [20].

A great deal of effort has been put forth by researchers and engineers in both academia and industry to build autonomous database systems [7], [14], [33], [39]. IBM DB2 provides users *self-tuning memory manager* to dynamically allocate memory to adapt to a wide variety of workloads [33], [35]. IBM DB2 also provides users a performance wizard tool to automatically select initial values for configuration parameters [22]. Microsoft's SQL Server uses a resource monitoring tool to provide users self-predicting features [26]. Oracle employs an internal monitoring subsystem to detect system bottlenecks caused by the system's misconfiguration [15], [21]. The DBSherlock tool helps DBAs diagnose problems by comparing slow and normal regions based on the system's time-series performance data [37]. BerkeleyDB uses influence diagrams to model probabilistic dependencies among configuration parameters to infer expected outcomes of a particular system configuration [34]. Despite all the efforts to provide certain self-managing properties in a database system, a fully autonomous database system has not yet been realized. In our work, we aim to build a fully autonomous MPP database system.

2) Our Architecture: Figure 12 shows the architecture of our MPP autonomous database, which consists of five major components: *information store, change manager, anomaly manager, workload manager and In-DB machine learning*. Our autonomous database system is capable of continuously monitoring the database system and collecting information on system performance and workloads, such as query response

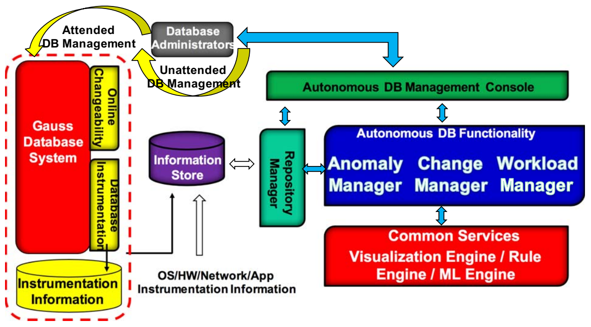


Fig. 12. Architecture of Huawei autonomous database

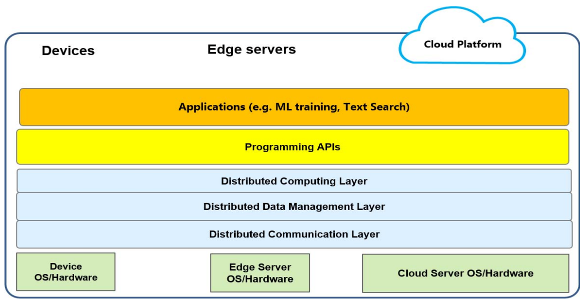

Devices
Edge servers
Cloud Platform
Applications (e.g. ML training, Text Search)
Programming APIs
Distributed Computing Layer
Distributed Data Management Layer
Distributed Communication Layer
Device OS/Hardware
Edge Server OS/Hardware
Cloud Server OS/Hardware

Fig. 13. Distributed collaboration platform across devices, edge and cloud

time and resource consumption, and stores the information in *information store*. Using the information, our system analyzes the current state of the database system and then determines if the controls, such as the automatic configuration, optimization and protection, need to be initiated. The *change manager* dynamically adapts to any change in system hardware and software. The *anomaly manager* detects and manages the anomalies, such as datanode failures, slow disk or insufficient memory. The *workload manager* monitors and controls query execution in the database system to ensure efficient use of system resources and achieve targeted SLA. The *In-DB machine learning* component provides functionalities of analyzing the stored information using machine-learning techniques.

B. Distributed Data Collaboration Platform Across Devices, Edge and Cloud

Huawei is one of the top three world smartphone makers in 2018. Besides smartphones and pads, Huawei also produces many kinds of consumer electronic devices, including smart speakers and smart watches. In addition, Huawei develops many Apps and tools that runs on Huawei's phones, such as Huawei Photos to store pictures online and Huawei Connect to share files across devices. Beside consumer electronics, Huawei also provides industrial IoT solutions, such as industrial warehouse management, highway traffic monitoring and autonomous vehicle solutions. Note that the different Huawei's devices have a broad spectrum of capabilities: on one end, smart sensors have very limited storage and power supply; on the other end, autonomous vehicle can generate hundreds of GB's or even TB's of data on a daily basis.

Cloud Computing (CC) has been playing a key role in smart consumer and IoT applications with three reasons. First, with almost infinite computing power and storage space, devices can offload many of their computing tasks to cloud platform through the Internet and also utilize it as their main data store. Second, cloud platforms can provide high availability and high elasticity with lower cost. However, as devices become ubiquitous in our lives, CC starts showing many limitations, including high latency, large data bandwidth consumption and security and privacy concern over storing user data in cloud. Another fairly obvious limitation is that applications using CC cannot work properly when the Internet is not available. In recent years, we see many new distributed system paradigms were proposed to address above limitations. Examples are Fog Computing [9], Edge Computing [6] and Dew Computing [31] etc. These solutions are mainly based on pushing computations close to the location of the data where they are generated on the different devices.

The next subsections discuss our solution in this area of distributed computing across devices, edge and cloud. We first show the architecture of our solution and then present our solutions to two key problems in this area: Mobile Backend as a Service (MBaaS) and data management for autonomous vehicles.

1) Architecture of Huawei's Distributed Computing Across Devices, Edge and Cloud: As shown in Figure 13, we envision a distributed collaboration framework across devices, edge and cloud having a software stack with at least three layers from top to bottom: distributed computing, distributed data management and distributed communication. The communication layer provides routing and communication services that can hide networking details to the layers above. On top of it, the distributed data layer provides a data service that hides data location and other important data properties such as number of copies. The distributed computing layer provides a distributed execution environment that allows running user programs anywhere. In our design, user defined functions can be dynamically downloaded from either cloud or neighboring edge servers or devices and executed locally. Functions can also access data through APIs provided by the underlying data layer. Such a distributed collaboration environment virtualizes communication, data and computing services that can provide great convenience to application developers by hiding many complex system details under the framework. In order to fully utilize the capability of the system, developers use similar programming APIs to write code that can be compiled and executed across devices, edge and cloud. Applications, such as ML training and text search, can utilize such a framework to provide distributed services without much difficulty.

Given data are now stored on many different edges/devices and upcoming 5G network enables much faster network access, we predicate that data must be accessible from anywhere, at anytime and on any user devices. In addition, we believe applications will become increasingly intelligent through understanding human behavior and intent. For example, we can imagine that urgent messages can be redirected to a TV if a user is watching TV while his/her phone is not nearby.

Our vision of such a distributed data collaboration platform across devices, edge and cloud should have following properties:

- **Ubiquitous:** Uniform API for transparent data access
- **Consistent:** Provide data consistency guarantees with configurable policies for various scenarios
- **Real-time:** Low latency data access and query-based event subscriptions (e.g. object location changes)
- **Secure:** Supports strong data privacy with declarative data sync and placement policy using fine granularity authorization rules
- **Smart:** User-oriented & AI-powered (e.g. query rewrite based on user profile)
- **Heterogeneous:** Able to support devices with a wide spectrum of capabilities

In the following sections, we briefly go through our early-stage works.

2) Direct Device to Device Data Sync on MBaaS: MBaaS provides mobile developers an integrated programming environment on the client side, automatically handles setting up cloud side servers and provides consistent data across devices and the cloud. This way the developers can focus on implementing client-side business logic without worrying about setting up servers and implementing server side programs. Google Firebase [1] and Apple CloudKit [30] are two well known MBaaS products currently on the market.

Current MBaaS solutions require devices sync data through the cloud, where devices send local updates to the cloud and refresh their local copies from the cloud. We envision to build a distributed data framework that allows seamless data collaboration across devices, edge and cloud. One important use case is to allow nearby devices to sync data through wireless ad hoc network without going through the cloud. There are multiple advantages with this approach. First, direct communication between devices based on Bluetooth is at least 10X faster than communications through the Internet. Second, it provides better privacy. For example, users may not want to upload home monitoring videos to cloud due to privacy concerns. Third, this approach works well in environments in remote areas with no or poor Internet connections.

We are developing an MBaaS product that can support direct device to device communication. We adopt a peer to peer architecture (P2P) for supporting device to device data sync in an ad hoc wireless network that allows devices to be added and removed dynamically. Our data sync mechanism guarantees no data loss and no redundant data. In addition, our system adopts a P2P sync algorithm to solve the time drift problem across devices. It currently supports eventual consistency and we may support stronger consistency models in the future.

There are many interesting research problems in developing a device to device real-time data sync platform. Below we just list a few due to the space limitation.

- **Data Sync Architecture:** Even though we have chosen P2P as our data sync paradigm to avoid a single point failure, a leader-based approach is also useful in a relatively stable network setting. For example, a wireless Internet router may be chosen as a leader in a home environment.
- **Resource Sharing:** A distributed data collaboration framework supports transparent data sharing across devices, allowing devices with limited storage to share resources with other more powerful devices. Smart watches are a good example of devices with limited storage and can benefit from other peer devices like smart phones or tablets.
- **Understand Human Being:** In order to provide a truly amazing user experience, applications need to understand a user's behaviors and preferences. For example, if a person is watching a movie on his phone with a near by smart TV then the platform can automatically project the movie onto the smart TV. Another use case is that a device is automatically selected out of multiple nearby home devices to respond to a user's voice command based on its capabilities and current status. This requires devices understand each other through syncing the states in real time.

3) Data Management for Autonomous Vehicles: With the advancement of AI and big data technologies, autonomous vehicles (AVs) are becoming closer and closer to reality. AVs produce huge amount of data but do not have the computing power to process this data. Therefore, AVs send their massive data to the cloud platform for processing. Below is a high level summary of these challenges and some thoughts on addressing them.

- **Massive Amount of Data:** Each data-collecting vehicle can generate about 800GB data daily including HD video and data collected from the vehicle's sensors, such as laser radar and inertial measurement unit (IMU) etc. How to manage such huge data sets and make them easy to query imposes a big challenge. Two ideas have been considered. First, perform data pre-aggregation for time series data at devices and edges. Second, separate cold data from hot data and utilize Huawei cloud storage to store them separately to reduce the storage cost.
- **High Dimensional Data Management:** AI algorithms are used to extract many properties from the raw data, such as determining if it is raining or if a car is passing by. We expect hundreds and even thousands of such dimensions/features to be generated dynamically. Indexes are created between the dimensions and the original raw data so that queries can be answered within sub-seconds latency. How to support flexible schema evolution and efficient high dimensional index (re)building in such an environment are two key challenges.
- **Secure and Trustworthy:** User's personal data such as personal driving preferences are synced to cloud for providing better driving experiences. We need to make sure such data are securely transferred and stored. In addition, data collected during car accidents is important in helping authorities and relevant parties to decide liability. Such

data need to be stored in a secure manner that guarantees accuracy and prevents from tampering.

V. Conclusion

As a leading global provider of ICT infrastructure and smart devices, Huawei faces a wide spectrum of data management challenges. In this paper, we provide an overview of FI-MPPDB and GMDB, which provides data management solutions to IT and CT respectively. We also present several recent features in above products in order for users to get a deeper understanding of our product. Finally, we present two big challenging areas that we are actively working on: autonomous database and distributed data collaboration platform across devices, edge and cloud. We think both areas are full of challenging and important data management research problems. We expect to develop complete solutions to those problems and invite collaborations from industries and academia.

VI. Acknowledgment

We thank Dr. Kuorong Chiang for reviewing this paper and providing helpful comments. We also thank all global members in Huawei database group for their contributions in the data management systems described in this paper.

References

[1] https://firebase.google.com/
[2] http://lambda-architecture.net/
[3] https://wiki.postgresql.org/wiki/Postgres-XC
[4] https://www.voltdb.com/blog/2016/02/18/appsneedacid/
[5] S. Agrawal, S. Chaudhuri, L. Kollar, A. Marathe, V. Narasayya, and M. Syamala, "Database Tuning Advisor for Microsoft SQL Server 2005". In Proceedings of the 30th VLDB Conference, Toronto, Canada, pp. 1110-1121, 2004.
[6] A. Ahmed and E. Ahmed, "A survey on mobile edge computing", Intelligent Systems and Control (ISCO), 10th International Conference on IEEE, pp. 18, 2016.
[7] D. V. Aken, A. Pavlo, G. J. Gordon, and B. Zhang, "Automatic database management system tuning through large-scale machine learning", ACM SIGMOD, pp. 1009-1024, 2017.
[8] J. Arulraj, A. Pavlo, and P. Menon, "Bridging the Archipelago between Row-Stores and Column-Stores for Hybrid Workloads", ACM SIGMOD, 2016.
[9] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things", Proceedings of the first edition of the MCC workshop on Mobile cloud computing, ACM, pp. 1316, 2012.
[10] L. Cai, J. Chen, Y. Chen, K. Chiang, M. Dimitrijevic, Y. Ding, Y. Dong, A. Ghazal, J. Hebert, K. Jagtiani, S. Lin, Y. Liu, D. Ni, C. Pei, J. Sun, Y. Wang, L. Zhang, M. Zhang, and C. Zhu, "Fusion insight librA: huawei's enterprise cloud data analytics platform", Proceedings of the VLDB Endowment 11(12), pp.1822–1834, August 2018.
[11] Peter Chen, "The Entity-Relational Model-Toward a Unified view of data.", ACM Transactions on database systems, 1976.
[12] James C. Corbett, and Jeffrey Dean, "Michael Epstein, Andrew Fikes, etc. Spanner: Google's Globally-Distributed Database", Proceedings of OSDI, 2012.
[13] Carlo Curino, Hyun Jin Moon, Alin Deutsch, and Carlo Zaniolo, "Automating the database schema evolution process", VLDB J. 22(1), pp. 73-98, 2013.
[14] B. K. Debnath, D. J. Lilja, and M. F. Mokbel, "Sard: A statistical approach for ranking database tuning parameters", ICDEW, IEEE, pp. 11-18, 2008.
[15] K. Dias, M. Ramacher, U. Shaft, V. Venkataramani, and G. Wood, "Automatic performance diagnosis and tuning in oracle", CIDR, pp. 84-94, 2005.
[16] Gartner Group https://www.gartner.com/doc/2657815/hybrid-transactionanalytical-processing-foster-opportunities
[17] V. Gasiunas, D. Dominguez-Sal, R. Acker, A. Avitzur, I. Bronshtein, R. Chen, E. Ginot, N. Martinez-Bazan, M. Müller, A. Nozdrin, W. Ou, N. Pachter, D. Sivov and E. Levy, "Fiber-based Architecture for NFV Cloud Databases", PVLDB Endowment 10(12), pp.1682–1693, August 2017.
[18] Kai Herrmann, Hannes Voigt, Andreas Behrend, Jonas Rausch, and Wolfgang Lehner, "Living in Parallel Realities: Co-Existing Schema Versions with a Bidirectional Database Evolution Language", SIGMOD Conference, pp. 1101-1116, 2017.
[19] O. Ivanov and S. Bartunov, "Adaptive Query Optimization in PostgreSQL", PGCon Conference, Ottawa, Canada, 2017.
[20] J. O. Kephart and D. Chess, "The Vision of Autonomic Computing", Computer, Volume 36, Issue 1, pp. 41-50, January 2003.
[21] S. Kumar, "Oracle database 10g: The self-managing database", 2003.
[22] E. Kwan, S. Lightstone, A. Storm, and L. Wu, "Automatic configuration for ibm db2 universal database", Proc. of IBM Perf Technical Report, 2002.
[23] H. Liu, M. Xu, Z. Yu, V. Corvinell, and C. Zuzarte, "Cardinality Estimation using Neural Networks", Proceeds of the 25th Annual International Conference on Computer Science and Software Engineering, pp. 53-59, 2015.
[24] Hongjun Lu and Rudy Setiono, "Effective Query Size Estimation Using Neural Networks", Applied Intelligence, pp. 173-183, May 2002.
[25] Microsft, "Azure Cosmos DB.", http://azure.microsoft.com/en-us/services/cosmos-db, 2018.
[26] D. Narayanan, E. Thereska, and A. Ailamaki, "Continuous resource monitoring for self-predicting dbms", IEEE, pp. 239-248, 2005.
[27] Oracle, "Oracle Database 18c Spatial and Graph", https://docs.oracle.com/en/database/oracle/oracle-database/18/spatial-and-graph.html, 2018.
[28] H. Plattner, "A Common Database Approach for OLTP and OLAP Using an In-Memory Column Database", SIGMOD, 2009.
[29] Ian Rae, Eric Rollins, Jeff Shute, Sukhdeep Sodhi, and Radek Vingralek, "Online, Asynchronous Schema Change", F1. PVLDB 6(11), pp. 1045-1056, 2013.
[30] A. Shraer, A. Aybes, B. Davis, C. Chrysafis, D. Browning, E. Krugler, E. Stone, H. Chandler, J. Farkas, J. Quinn, J. Ruben, M. Ford, M. McMahon, N. Williams, N. Favre-Felix, N. Sharma, O. Herrnstadt, P. Seligman, R. Pisolkar, S. Dugas, S. Gray, S. Lu, S. Harkema, V. Kravtsov, V. Hong, W. Yih, and Y. Tian, "CloudKit: Structured Storage for Mobile Applications", PVLDB 11(5), pp. 540-552, 2018.
[31] K. Skala, D. Davidovic, E. Afgan, I. Sovic, and Z. Sojat, "Scalable distributed computing hierarchy: Cloud, fog and dew computing", Open Journal of Cloud Computing (OJCC), vol. 2, no. 1, pp. 1624, 2015.
[32] M. Stonebraker and A. Weisberg, "The VoltDB main memory DBMS", IEEE Data Eng. Bull. 36(2), pp. 21-27, 2013.
[33] A. J. Storm, C. Garcia-Arellano, S. S. Lightstone, Y. Diao, and M. Surendra, "Adaptive self-tuning memory in db2", VLDB, pp. 1081-1092, 2006.
[34] D. G. Sullivan, M. I. Seltzer, and A. Pfeffer, "Using probabilistic reasoning to automate software tuning", volume32, ACM, 2004.
[35] W. Tian, P. Martin, and W. Powley, "Techniques for automatically sizing multiple buffer pools in db2", Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research, IBM Press, pp. 294-302, 2003.
[36] W. Wang, M. Zhang, G. Chen, H. V. Jagadish, B. C. Ooi, and K-L. Tan, "Database Meets Deep Learning: Challenges and Opportunities", SIGMOD Record, Vol. 45, No2, 2016.
[37] D. Y. Yoon, N. Niu, and B. Mozafari, "Dbsherlock: A performance diagnostic tool for transactional databases", ACM SIGMOD, pp. 1599-1614, 2016.
[38] H. Zhen, J. Lu, D. Gawick, and H. Helskyaho, "Multi-Model Database Management Systems - a Look Forward", VLDB Workshop, 2018.
[39] Y. Zhu, J. Liu, M. Guo, Y. Bao, W. Ma, Z. Liu, K. Song, and Y. Yang, "Bestconfig: tapping the performance potential of systems via automatic configuration tuning", SoCC, ACM, pp. 338–350, 2017.