Cloud-Native Database Systems at Alibaba: Opportunities and Challenges

Feifei Li
Alibaba Group
lifeifei@alibaba-inc.com

ABSTRACT

Cloud-native databases become increasingly important for the era of cloud computing, due to the needs for elasticity and on-demand usage by various applications. These challenges from cloud applications present new opportunities for cloud-native databases that cannot be fully addressed by traditional on-premise enterprise database systems. A cloud-native database leverages software-hardware co-design to explore accelerations offered by new hardware such as RDMA, NVM, kernel bypassing protocols such as DPDK. Meanwhile, new design architectures, such as shared storage, enable a cloud-native database to decouple computation from storage and provide excellent elasticity. For highly concurrent workloads that require horizontal scalability, a cloud-native database can leverage a shared-nothing layer to provide distributed query and transaction processing. Applications also require cloud-native databases to offer high availability through distributed consensus protocols.

At Alibaba, we have explored a suite of technologies to design cloud-native database systems. Our storage engine, X-Engine and PolarFS, improves both write and read throughputs by using a LSM-tree design and self-adapted separation of hot and cold data records. Based on these efforts, we have designed and implemented POLARDB and its distributed version POLARDB-X, which has successfully supported the extreme transaction workloads during the 2018 Global Shopping Festival on November 11, 2018, and achieved commercial success on Alibaba Cloud. We have also designed an OLAP system called AnalyticDB (ADB in short) for enabling real-time interactive data analytics for big data. We have explored a self-driving database platform to achieve autoscaling and intelligent database management. We will report key technologies and lessons learned to highlight the technical challenges and opportunities for cloud-native database systems at Alibaba.

PVLDB Reference Format:
Feifei Li. Cloud-Native Database Systems at Alibaba: Opportunities and Challenges. *PVLDB*, 12(12): 2263 - 2272, 2019.
DOI: https://doi.org/10.14778/3352063.3352141

Proceedings of the VLDB Endowment, Vol. 12, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3352063.3352141

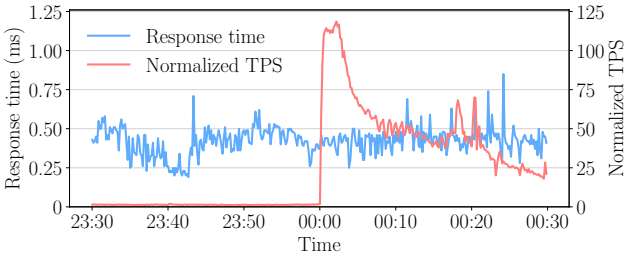

Figure 1: More than 100× increase in TPS along with stable response time observed during Alibaba's Singles' Day Shopping Festival in 2018.

1. INTRODUCTION

With more and more applications and systems moving to the cloud, cloud-native database systems start to gain wide support and popularity. Cloud database services provided by cloud service vendors, such as AWS, Microsoft Azure, Alibaba Cloud, and Google Cloud have contributed to the development of cloud-native databases. As a result, in recent years the market share of cloud databases has been growing rapidly. More and more enterprises and organizations have migrated their businesses from on-premise data centers to the cloud. The cloud platforms provide high elasticity, stringent service-level agreement (SLA) to ensure reliability, and easy manageability with reduced operational cost. The cloud databases play a key role in supporting cloud-based businesses. It becomes the central hub that connects underlying resources (IaaS) to various applications (SaaS), making it a key system for the cloud.

At the Alibaba group, database systems need to support a rich and complex business ecosystem that spans over entertainment and digital media, e-commerce and e-payment, and various new retail and o2o (offline to online) business operations. During the 2018 Singles' Day Global Shopping Festival (Nov. 11, 2018), Alibaba's databases process up to 491,000 sales transactions per second, which translates to more than 70 million transactions per second. The traditional on-premise deployment of databases are not able to catch up with the complexity of such business operations, due to the needs for elasticity, scalability, and manageability. For example, as shown in Figure 1, the TPS is suddenly increased in the first second of Singles' Day Shopping Festival, which is about 122 times higher than that of the previous second. When we simply deploy a MySQL or PostgreSQL on a cloud instance store with a local SSD and a high I/O VM, the resulting database instance has limited capacity that is

not suitable for providing scalable database services. It cannot survive underlying disk drive failures; and the database instance has to manage data replication for reliability. In addition, the instance uses a general-purpose file system, such as ext4 or XFS. When using low I/O latency hardware like RDMA or PCIe SSD, the message-passing cost between kernel space and user space may quickly saturate the throughput. In contrast, databases with a cloud-native design (such as decouple of computation and storage, autoscaling) are more appealing, which are able to provision more compute and storage capacity, and provide faster recovery and lower cost [30, 7].

There are also other essential capacities that are of critical importance for cloud-native databases: *multi-model* that supports heterogeneous data sources and diverse query interfaces; *autonomy and intelligence* that automatically manages and tunes database instances to reduce the cost of manual operations; *software-hardware co-design* that leverages the advantages of high-performance hardware; and *high availability* that meets stringent SLA needs (e.g., RPO=0 with very small RTO). With these designs in mind, cloud-native databases have gained rapid growth for cloud-based deployment.

In this paper, we report the recent progress in building cloud-native enterprise databases on Alibaba Cloud [1], which also support the entire business operations within the Alibaba group from its various business units (from entertainment to e-commerce to logistics). To cover a wide variety of application needs, we have provided a broad spectrum of database systems and tools as shown in Figure 3. In particular, we have developed POLARDB, a shared-storage OLTP database that provisions 100TB of storage capacity and 1 million QPS per processing node. To further scale out the capacity, we have developed POLARDB-X, a distributed OLTP database that integrates shared-nothing and shared-storage designs. We have also developed AnalyticDB, an OLAP database as a next-generation data warehouse for high-concurrency, low-latency, and real-time analytical queries at PB scale. To manage numerous database instances hosted on our cloud, we have built SDDP, an autonomous database operation platform that automatically manages instances and tunes performance with minimal DBA involvement. There are nearly half a million database instances running on Alibaba Cloud (from both our cloud customers and various business units within Alibaba group). Both POLARDB and AnalyticDB have gained rapid growth in usage from a broad range of business sectors, including e-commerce, finance, media and entertainment, education, new retail, and others. These cloud database systems and technologies have successfully served both the complex business ecosystem within the Alibaba group and many external enterprise customers.

2. ARCHITECTURES FOR DATABASE SYSTEMS AT ALIBABA

Depending on what is being shared, there are three popular architectures that we have explored at Alibaba for building our database systems, as illustrated in Figure 2. The first category is *single instance*, which is the most common architecture used by mainstream databases. In this model, all processes in a database share processor cores, main memory space and local disks (i.e., on a single machine). Such an architecture facilitates and eases intra-system communication and coordination.

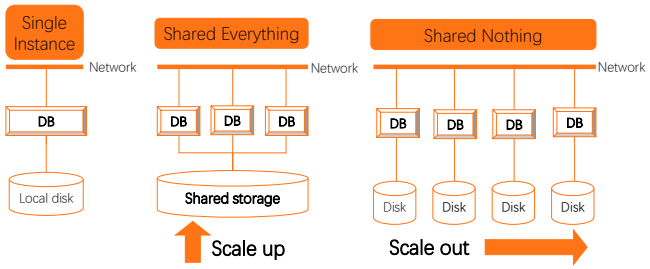

Figure 2: Different database system architectures.

With the rapid growth in both amounts of data and peak workloads of those encountered by giant Internet enterprises, such as that in Google, Amazon, Microsoft and Alibaba, it has been observed that the single-instance architecture has inherent limitations. The capacity of a single machine fails to meet ever-increasing business demands. Therefore, the *shared storage* architecture was proposed, represented by AWS Aurora [30] and Alibaba POLARDB [5]. In this model, the underlying storage layer (usually consists of multiple nodes) is decoupled and each data record in storage can be accessed by any upper database kernels running on any node. By exploiting a fast network such as RDMA, a database can interact with the shared distributed storage layer the same way as with a single (shared) local disk. On top of this shared storage, we can easily launch multiple compute nodes to create replicas of a single database, having the identical view on the same data. Therefore, requests can be distributed to different (read-only) nodes for parallel processing. However, to avoid write conflicts and to avoid the complexity of dealing with distributed transaction processing and distributed commits, there is usually a single node that processes all write requests (e.g., `INSERT`, `UPDATE`, `DELETE`) to a database. This architecture enables dynamic adjustment of query capacity on demand by changing the number of read-only nodes. It is also feasible to enable writes to multiple nodes (i.e., multi-master) to expand write capacity, but usually requires complex concurrency control mechanisms and consensus protocols [6, 12, 16, 17, 23, 34].

The shared-storage architecture also has its own limitations. First, low-latency data transmission cannot be always guaranteed between compute and storage nodes. For those messages transmitted cross switches, data centers or even regions, the transmission time will be significantly amplified, especially when local RDMA network is used. Second, the number of read-only nodes supported for a single database is limited. When the number of nodes reaches a certain scale, massive requests will be blocked, making accesses to remote storage prohibitively expensive and unaffordable. Therefore, a practical limit is to have roughly up to a dozen of read-only nodes. To address this issue, we need a *shared nothing* architecture. In this model, a logical database is divided into multiple shards, each of which is assigned to a node. These nodes can be placed and replicated in different data centers and regions. A representative implementation of this architecture is Google Spanner [7], which uses GPS and atomic clocks to achieve replica consistency and transactional consistency across regions. On Alibaba Cloud, we build POLARDB-X that extends POLARDB and explores the benefits of building a shared-nothing system on top of

multiple databases each with a shared distributed storage.

Note that this hybrid of shared-nothing and share-storage architectures brings some particular benefits. We can apply sharding at the top level, but assign many nodes to a shard (instead of one node per shard). Beneath this shard, a shared storage can be accessed by these nodes. The benefit of this hybrid architecture is that it mitigates the drawbacks of having too many small shards. In particular, it helps to ease the procedure of shard re-balancing, and reduces the probability of cross-shard transactions (and reduce the amount of expensive distributed commits). Meanwhile, it enables excellent horizontal scalability. This hybrid architecture that takes advantage of both shared nothing and shared storage is a promising direction explored by our database design in POLARDB-X.

3. OTHER KEY FEATURES OF ALIBABA DATABASE SYSTEMS

In addition to exploring different system architectures, there are other key features, driven by Alibaba's complex business applications, that have been taken into consideration during the design of Alibaba's database systems.

3.1 Multi-Model Analysis

An important application scenario at Alibaba is to support multi-model analysis, which consists of two aspects: southbound and northbound multi-model access. The southbound multi-model access indicates that the underlying storage supports different data formats and data sources. The stored data can be either structured or non-structured, e.g., graph, vector and document storage. The database then provides a unified query interface, such as a SQL or SQL-like interface, to query and access various types of data sources and formats, forming a data lake service. The northbound multi-model access indicates that a single data model and format (e.g., key-value model in most cases) is used to store all structured, semi-structured and unstructured data in a single database. On top of this single storage model, the database then supports multiple query interfaces, such as SQL, SPARQL and GQL depending on the application needs. Microsoft CosmosDB [9] is a representative system of this kind.

In addition to addressing our internal business operation needs, being able to support multi-model analysis is also an essential requirement for cloud database services. Many cloud applications require to collect large-volumes of data from heterogeneous sources, and conduct federated analysis to link different sources and reveal business insights (i.e., south-bound multi-model access). On the other hand, a cloud database (e.g., a large KV store such as HBase) is often a central data repository accessed by multiple applications with various application needs. They may prefer to use different query interfaces due to usability and efficiency, where northbound multi-model analysis is needed.

3.2 Autonomy and Intelligence

Given the large number of database instances to be managed and the complex workloads that are faced by the database systems at Alibaba, making the database operation platform more autonomous and intelligent is an essential requirement. With more than hundreds of thousands of live database instances running on our platform, it is infeasible to reply on conventional DBA-based manual operation, tuning, and maintenance in a per-instance manner. There exists many opportunities for supporting autonomous operations [8, 18, 19, 21, 24, 26, 29] from the perspective of both database kernels and the underlying operation platform. With that in mind, we are committed to building a self-driving database platform (SDDP) with capabilities of self-detection, self-decision, self-recovery and self-optimization. Consider self-optimization as an example, various modules in a database kernel (e.g., indexing, query optimizer, and buffer pool management) are to be enhanced by adopting machine learning techniques, so that these modules can adaptively optimize for dynamic workloads. However, making them both effective and efficient inside a database kernel is a challenging task, due to the high cost of training and inference of machine learning models. On the other hand, self-detection, self-decision and self-recovery target at improving the efficiency and effectiveness of database operation platform. There are several key challenges such as how to automatically inspect instance status and detect abnormal behaviors; and how to make a correct decision to repair the errors within a short reaction time, once it is detected.

3.3 Software-Hardware Co-Design

Another key subject of innovation for Alibaba's database systems is to explore and take advantage of the fast development and innovation in the hardware space. As with any other systems, our goal is to design and implement our database systems that are able to use limited system hardware resources in a safe and efficient manner. This objective means that the systems must pay attention to the constant change and improvement in hardware properties so that they can leverage the advantages of innovative new hardware features. As a performance-critical system, a database system needs to fully utilize available resources to execute queries and transactions robustly and efficiently. As new hardware properties are constantly improving, it is unwise to simply follow existing database designs and expect that they will maximize performance on new hardwares. For example, the performance of a sophisticated databases like MySQL running directly on RDMA-enabled distributed storage is significantly worse than those on local PCIe SSDs with the same CPU and memory configurations, which requires a careful re-design [5]. Hence, the opportunities brought by new hardware are of important considerations in designing Alibaba's database systems. For example, we have extensively explored and integrated new hardware technologies such as RDMA, NVM, GPU/FPGA, and NVMe SSD.

3.4 High Availability

High availability is one of the fundamental requirements for database systems at Alibaba to ensure zero-downtime for our business operations and our cloud customers, as most enterprise customers are intolerant to the interruption of their businesses. One canonical solution for high availability is replication, which can be done at the granularity of database instance, table or table shard. The widely used primary-backup and three-way replications are competent in most scenarios. For banking and finance sectors that needs higher availability, four or more replicas might be enforced, which are usually placed at different data centers (available zones) and regions in order to survive large-area failures (such as network failures and data center outages).

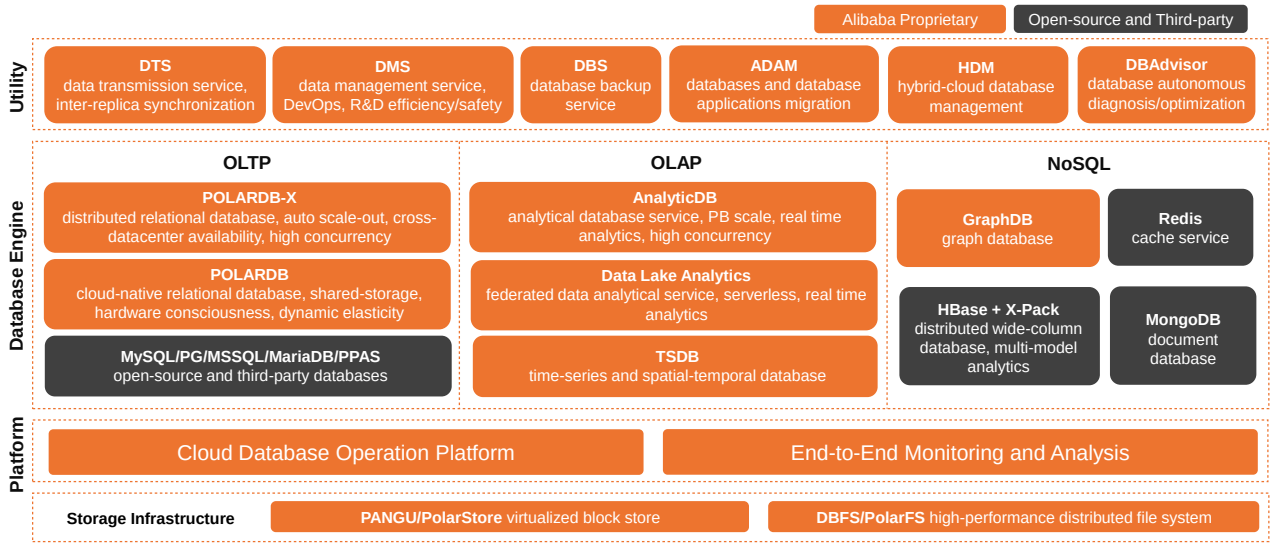

Figure 3: Database systems and services at Alibaba and Alibaba Cloud.

In the adoption of replications, the data consistency between replicas must be carefully handled. The CAP theorem concludes that only two out of three properties can be satisfied among consistency, availability and partition tolerance. At Alibaba, we design and implement our database systems with 'C' (consistency) and 'P' (partition tolerance) in mind, and ensure high availability with a customized parallel paxos protocol called X-Paxos, which ensures that we can still deliver an extremely high level of availability that is up to 99.999%. X-Paxos implements and optimizes sophisticated replication techniques and consensus protocols, and ensures data consistency and availability via logs.

4. ALIBABA CLOUD-NATIVE DATABASES

In this section, we share our recent progress in building cloud-native database systems at Alibaba. A complete overview of database systems and products at Alibaba and on Alibaba cloud is summarized in Figure 3. We focus on the discussion of POLARDB (a shared-storage OLTP database) and its distributed version POLARDB-X (a sharded shared-nothing OLTP database built on top of POLARDB), AnalyticDB (a real-time interactive OLAP database), and SDDP (an autonomous database operation platform).

4.1 POLARDB: cloud-native OLTP database

POLARDB is a relational database system built based on AliSQL (a fork of MySQL/InnoDB) [2], and is available as a database service on Alibaba Cloud. POLARDB follows a cloud-native architecture that provides high elasticity, high volume and high concurrency. In addition, POLARDB is fully compatible with MySQL and PostgreSQL, which helps customers to conduct transparent and smooth business application migrations.

4.1.1 System Design

POLARDB follows the shared-storage architecture, as shown in Figure 4. It consists of three layers: a PolarProxy acting as a unified access portal, a multi-node database cluster, and a distributed shared file system PolarFS. *PolarProxy* is a distributed stateless proxy cluster with self-adaptive capacity.

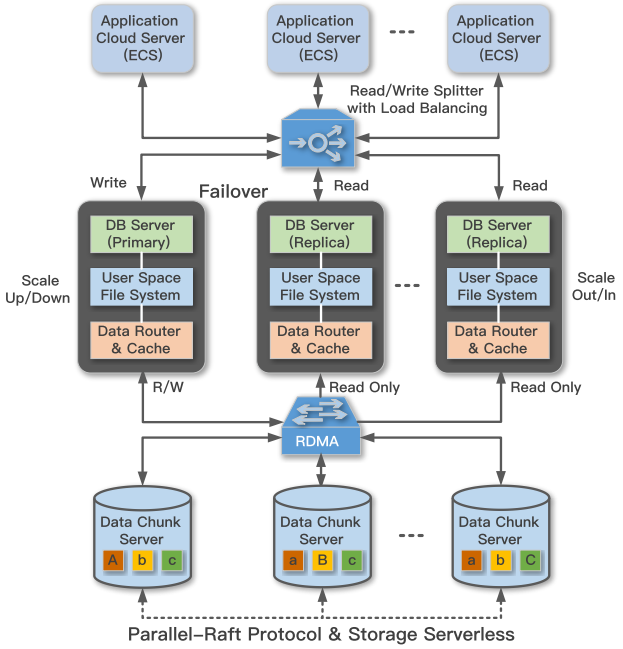

Figure 4: Architecture of POLARDB.

It integrates the resources of multiple computation nodes and provides a unified portal for applications to access. Its dynamic scale-out capability enables agile increase/decrease of nodes. The database nodes in POLARDB are divided into two types, i.e., a *primary* node and many *read-only* (RO) nodes. The primary node can handle both read and write queries, while RO nodes only process read queries. Both primary and RO nodes share redo log files and data files, which are managed by *PolarFS* (Section 4.1.2), a distributed file system with ultra-low latency, high throughput and high availability.

Such an architecture has several distinctive advantages. First, the compute and storage resources are decoupled. Compute and storage nodes can use different types of server hardware and can be customized separately. For example,

the compute nodes need no longer to consider the ratio of memory size to disk capacity, which is highly dependent on the application scenario and hard to predict. Second, it breaks the limitations in single-node databases (e.g., MySQL, PostgreSQL). Disks on storage nodes form a single storage pool, which reduces the risk of fragmentation, usage imbalance, and space wastage. The capacity and throughput of a storage cluster can scale out transparently. POLARDB is able to provision 100TB of storage capacity and achieve 1 millions QPS per node. Third, since data are all stored on the storage cluster, there is no local persistent state on compute nodes, making it easier and faster to perform database migration. Data reliability can also be improved because of the data replication and other high availability features provided by PolarFS.

Apart from POLARDB, other cloud database services can also benefit from this architecture. First, databases can build on a more secure and easily scalable environment based on virtualization techniques, such as Xen [3], KVM [13] or Docker [20]. Second, some key features of databases, such as multiple read-only instances and checkpoints, could be easily achieved since back-end storage clusters provide fast I/O, data sharing, and snapshot.

4.1.2 *PolarFS and PolarStore*

Data storage technology continues to change at a rapid pace, and current cloud platforms have trouble taking full advantage of the emerging hardware standards such as RDMA and NVMe SSD. For instance, some widely used open-source distributed file systems, such as HDFS [4] and Ceph [31], are found to have much higher latency than local disks. When the latest PCIe SSDs are used, the performance gap could even reach orders of magnitude. The performance of relational databases like MySQL running directly on these distributed storage is significantly worse than that on local PCIe SSDs with the same CPU and memory configurations.

To this end, we build PolarFS [5] as the shared storage layer for POLARDB. It is a distributed file system built on top of PolarStore (a shared distributed storage based on RDMA network), offering ultra-low latency, high throughput and high availability via following mechanisms. First, PolarFS takes full advantage of emerging hardware such as RDMA and NVMe SSD, and implements a lightweight network stack and I/O stack in user space to avoid trapping into kernel and dealing with kernel locks. Second, PolarFS provides a POSIX-like file system API, which is intended to be compiled into the database process and replace the file system interfaces provided by operating system, so that the whole I/O path can be kept in user space. Third, the I/O model of the PolarFS's data plane is also designed to eliminate locks and avoid context switches on the critical data path. All unnecessary memory copies are eliminated, while RDMA is heavily utilized to transfer data between main memory and RDMA NIC/NVMe disks. With all these features, the end-to-end latency of PolarFS has been reduced drastically, being quite close to that of local file system on SSD.

As node failures in a large POLARDB cluster are common, a consensus protocol is needed to ensure that all committed modifications will not get lost in corner cases. Replicas should always reach agreement and become bitwise identical. In PolarFS, we first used Raft [23], a variant of Paxos family [17, 16], which is easier to implement and widely used

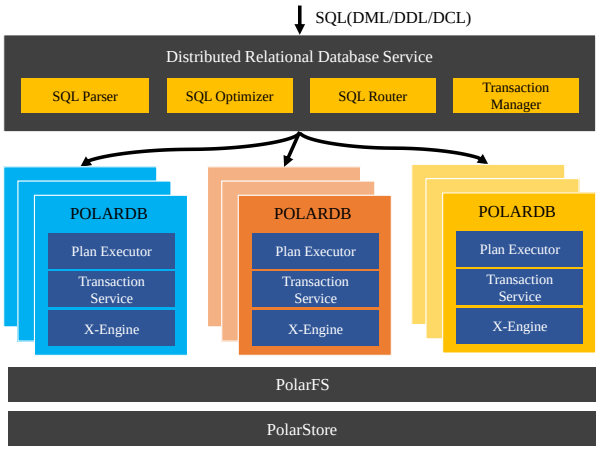

Figure 5: Architecture of POLARDB-X.

by many distributed systems. However, when Raft was applied, we found that it seriously impedes the I/O scalability of PolarFS where low-latency NVMe SSD and RDMA are used (whose latency are on the order of tens of microseconds). Therefore, we developed *ParallelRaft*, an enhanced consensus protocol based on Raft, which allows out-of-order log acknowledging, committing and applying, while letting PolarFS comply with traditional I/O semantics. With this protocol, parallel I/O concurrency has been significantly improved.

In summary, PolarFS supports POLARDB with following features: (1) PolarFS can synchronize the modification of file metadata (e.g. file truncation or expansion, file creation or deletion) from primary nodes to RO nodes, so that all changes are visible for RO nodes. (2) PolarFS ensures that concurrent modifications to file metadata are serialized, so that the file system itself is consistent across all database nodes. (3) In case of a network partition, two or more nodes might act as primary nodes writing shared files concurrently. PolarFS can ensure that only the real primary node is served successfully, preventing data corruption. More technical details can be found in [5].

4.2 POLARDB-X: distributed OLTP database

POLARDB scales well for up to tens of nodes (due to the limitation of the underlying RDMA network), but this is not sufficient for support highly concurrent workloads over massive amounts of data and transactions, such as that found on the Single's Day Shopping Festival. Hence, we have extended POLARDB and built POLARDB-X, a distributed shared-nothing OLTP database to enable horizontal scale-out, which combines shared-storage and shared-nothing architectures. The benefit of this design, as compared to a standard shared-nothing architecture using a single-node instance on each shard, is that each shard can now afford to store and process much more data and transactions due to the scale-up capability introduced by the shared-storage architecture. As a result, for the same amount of data and/or transaction processing needs, the hybrid architecture needs much less number of shards compared to a standard shared-nothing system; this in turns reduces the chances of dealing with complex and expensive distributed transaction processing and distributed commits. As a result, it supports highly concurrent transactions over massive data, and ensures cross-AZ and cross-region transaction consistency

through the parallel paxos protocol X-Paxos.

4.2.1 System Design

Figure 5 shows the architecture of POLARDB-X, in which relational data is partitioned into multiple POLARDB instances, and managed by a distributed relational database service (DRDS). DRDS takes in SQL queries or transactions, parses and optimizes their plans, and finally routes them to corresponding POLARDB instances for execution. As discussed previously, each POLARDB instance consists of one primary node and multiple read-only nodes. Each read node serves as a replica of the primary node, sharing the same storage residing on the PolarFS, which in turn sits on PolarStore, Alibaba's block storage system. Inside a POLARDB node, there is a plan executor for query plans pushed from the DRDS, a transaction service for transaction processing, and an X-Engine, Alibaba's LSM-tree based OLTP storage engine.

4.2.2 X-Engine

We find that, when handling such transactions at Alibaba and our big enterprise customers, three key challenges have to be addressed: (1) *The tsunami problem* - there are drastic increase in transactions with the kickoff of major sales and promotional events (e.g., there was a 122-time spike on Alibaba Singles' Day Global Shopping Festival), which puts tremendous pressure to the underlying database. (2) *The flood discharge problem* - large amount of hot records can easily overwhelm system buffers, which blocks subsequent transactions if buffers cannot be fast flushed. (3) *The fast moving current problem* - due to large numbers of promotion events that last over short time periods, quick shifts of record "temperatures" (i.e., hot, warm, cold) occurs frequently, which drastically lowers cache hit ratio.

We build X-Engine [10] to tackle above challenges faced by Alibaba's e-commerce platform, because a significant part of transaction processing performance boils down to how efficiently data can be made durable and retrieved from the storage. X-Engine processes most requests in the main memory by exploiting the thread-level parallelism (TLP) in multi-core processors, decouples writes from transactions to make them asynchronous, and decomposes a long write path into multiple stages in a pipeline in order to increase the overall throughput. To address the flood discharge problem, X-Engine exploits a tiered storage approach to move records among different tiers, taking advantage of a refined LSM-tree structure [22, 25] and optimized compaction algorithms. We also apply FPGA offloading on compactions. Finally, to address the fast-moving current problem, we introduce a multi-version metadata index which is updated in a copy-on-write fashion to accelerate point lookups in the tiered storage regardless of data temperatures.

Figure 6 shows the architecture of X-Engine. X-Engine partitions each table into multiple sub-tables, and maintains an LSM-tree, the associated metasnapshots and indexes for each sub-table. X-Engine contains one redo log per database instance. Each LSM-tree consists of a *hot data tier* residing in main memory and a *warm/cold data tier* residing in NVM/SSD/HDD (that are further partitioned into different levels), where the term hot, warm, and cold refers to data temperatures, representing the ideal access frequencies of data that should be placed in the corresponding tier. The hot data tier contains an *active memtable* and

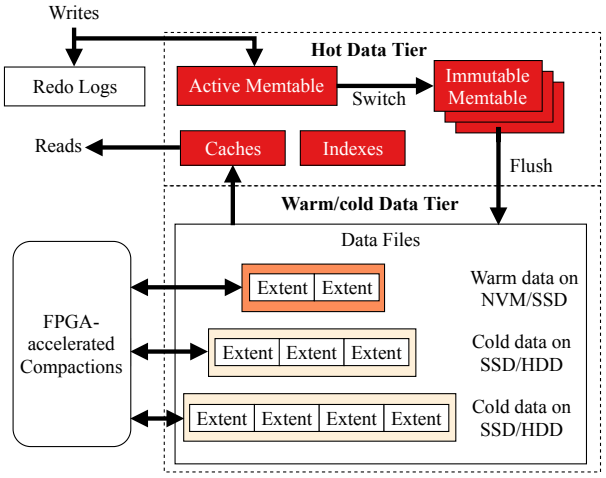

Figure 6: Architecture of X-Engine.

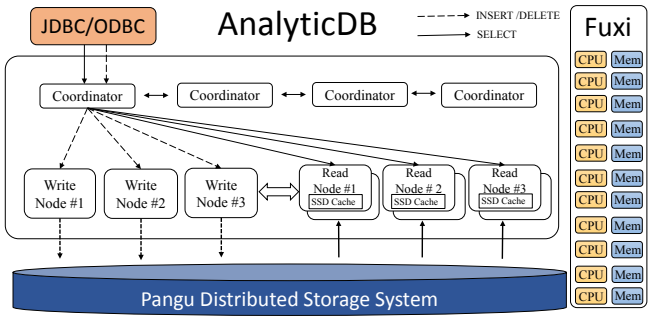

Figure 7: Architecture of AnalyticDB

multiple *immutable memtables*, which are skiplists storing recently inserted records, and *caches* to buffer hot records. The warm/cold data tier organizes data in a tree-like structure, with each level of the tree storing a sorted sequence of *extents*. An extent packages blocks of records as well as their associated filters and indexes.

X-Engine exploits redo logs, metasnapshots, and indexes to support Multi-version Concurrency Control (MVCC) for transaction processing. Each *metasnapshot* has a *metadata index* that tracks all memtables, and extents in all levels of the tree in the snapshot. One or multiple neighboring levels of the tree forms a tier to be stored on NVM, SSD, and HDD, respectively. Each sub-table in X-Engine has its own hot, warm and cold data tiers (i.e., LSM-trees), storing records in a row-oriented format. We design a multi-version memtables to store records with different versions to support MVCC. On the disks, the metadata indexes track all the versions of records stored in extents. More technical details can be found in [10].

4.3 AnalyticDB: realtime OLAP datawarehouse

AnalyticDB is a real-time OLAP database system designed for high-concurrency, low-latency, and real-time analytical queries at PB scale. It has been running on from as little as 3 nodes to up to 2000+ physical machines and is provided as a database service on Alibaba Cloud. It serves enterprise customers from a wide range of business sectors, including e-commerce, fintech, logistics, public transit, meteorological analysis, entertainment, etc., as well as internal business operations within Alibaba Group.

Recent works [28, 14, 15, 32, 11] have summarized the main challenges of designing an OLAP system as achieving low query latency, data freshness, flexibility, low cost, high scalability, and availability. Compared to these works, large-scale analytical workloads from our application scenarios elevate AnalyticDB to an even larger scale: 10PB+ data, hundred thousands of tables and trillions of rows, which presents significant challenges to the design and implementation of AnalyticDB: 1) Today's users face more complicated analytics scenarios than ever before, but still have high expectation for low query latency. Though queries from different applications are diverse and complex, they often do not tolerate queries that spend a long time. 2) Emerging complex analysis tends to combine different types of queries and data. More than half of our users' data has a complex data type, such as text, JSON string, or vector. A practical database should be able to efficiently support queries on heterogeneous data with complex types. 3) While processing real-time queries with low latency, the system also needs to handle tens of millions of online write requests per second. Traditional designs that read and write data in the same process path are no longer well-suited for this case. Careful designs to balance among query latency, write throughput and data visibility should be taken into consideration.

To address these challenges, we build AnalyticDB with several novel designs. First, AnalyticDB embeds an efficient and effective index engine. In this engine, indexes are built *on all columns in each table* for significant performance gain on ad-hoc complex queries. We further propose a runtime filter-ratio-based index path selection mechanism to avoid performance slow-down from index abuse. Since it is prohibitively expensive to update large indexes in the critical path, indexes are asynchronously built during off-peak periods. We also maintain a lightweight sorted-index to minimize the impact of asynchronous index building on queries involving incremental data (i.e., data written after the current round of index building has started).

Second, we design the underlying storage layout to support hybrid row-column storage for structured data and other data with complex types. In particular, we utilize fast sequential disk IOs, so that its overhead is acceptable under either OLAP-style or point-lookup workloads. We further incorporate complex data types in the storage (including indexes) to provide the capability of searching resources (i.e., JSON, vector, text) together with structured data.

Third, in order to support both high-throughput writes and low-latency queries, our system follows an architecture that decouples reads and writes, i.e., they are served by write nodes and read nodes respectively. These two types of nodes are isolated from each other and hence can scale independently. In particular, write nodes persist write requests to Pangu (a reliable distributed storage on Alibaba Cloud). To ensure data freshness when serving queries, a version verification mechanism is applied on read nodes, so that previous writes processed on write nodes are visible.

Forth, to further improve query latency and concurrency, we enhance the optimizer and execution engine in AnalyticDB to fully utilize the advantages of our storage and indexes. Specifically, we propose a storage-aware SQL optimization mechanism that generates optimal execution plans according to the storage characteristics, and an efficient real-time sampling technique for cardinality estimation in cost based optimizer. Besides, we design a high-performance vectorized execution engine for the hybrid storage that improves the efficiency of computationally intensive analytical queries.

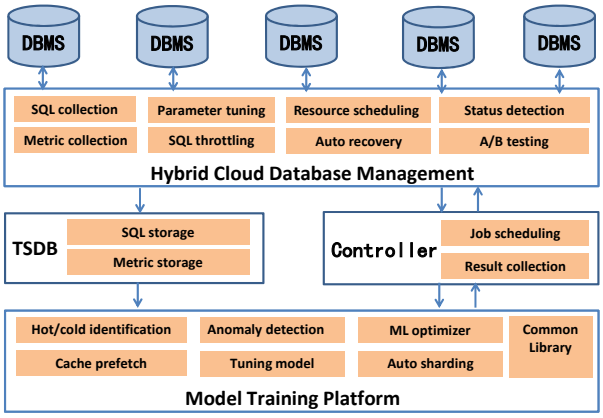

Figure 8: Architecture of SDDP.

Figure 7 shows the system architecture. There are mainly three types of nodes in AnalyticDB, i.e., coordinator, write node and read node. The *coordinator* collects requests (both writes and queries) from client connections, and dispatches them to corresponding write and read nodes. The *write nodes* are responsible for processing writes (such as `INSERT`, `DELETE`, `UPDATE`), and flush SQL statements into Pangu for persistence. *Read nodes* are responsible for handling queries (such as `SELECT`). In this manner, Write and read nodes are decoupled from each other. Fuxi (a resource manager and job scheduler on Alibaba Cloud) utilizes available resources in all these nodes to provide computation workers for asynchronous task execution. In addition, AnalyticDB provides a general-purpose and pipeline-mode execution engine that runs on computation workers. Data flows through the system in units of column blocks from the storage to the client. All data processes are in memory and are pipelined between different stages across the network. This pipeline workflow enables AnalyticDB to serve users' complex queries with high throughput and low latency. More technical details can be found in [33].

4.4 SDDP: Self-Driving Database Platform

To manage numerous database instances on Alibaba Cloud, we have built an autonomous database management platform, called SDDP (Self-Driving Database Platform). This platform collects real-time statistics from all running database instances, and uses machine learning and statistical methods to tune instances and provision resources.

4.4.1 System Design

Figure 8 shows the architecture of SDDP. The Hybrid Cloud Database Management (HDM) layer collects SQLs and metrics from database instances and stores them in a time-series database (TSDB). Meanwhile, HDM detects database status and synchronizes Controller with these information. Database status is normally changed by DDL operations. For example, we can use `ALTER TABLE t1 HOTCOLD = 'SMART'` to set a table to `SMART` mode and separate hot/cold data automatically (Section 4.4.3). HDM also assigns Controller to drive machine learning tasks, such as parameter tuning (Section 4.4.2), resource scheduling and anomaly detection. Controller schedules these tasks to Model Training

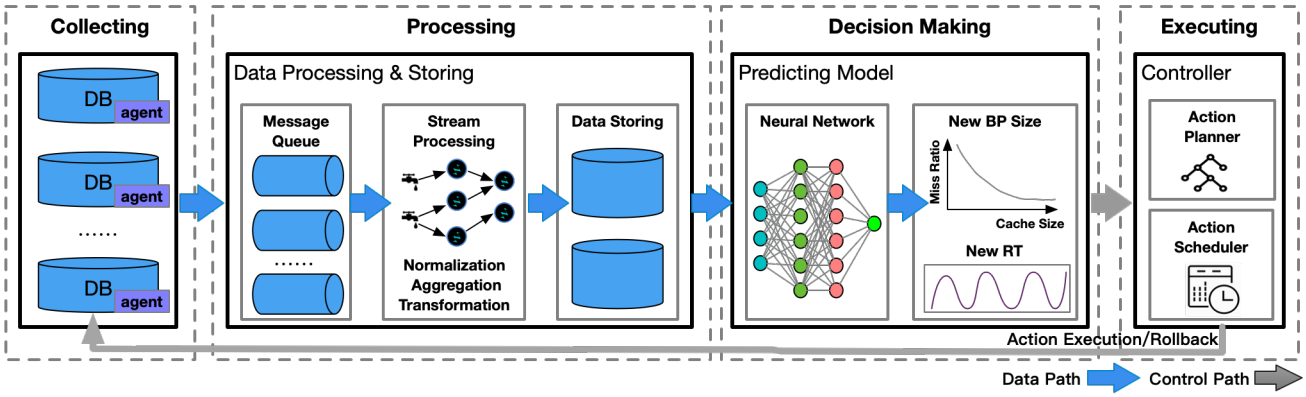

Figure 9: Workflow and overall architecture of iBTune.

Platform (MTP). MTP retrieves data, including SQL and metrics, from TSDB and uses different modules to complete the corresponding jobs. The result will be transferred back to HDM by Controller and applied to database instances.

4.4.2 *Buffer Size Tuning*

Buffer pool is a critical resource for an OLTP database, serving as a data caching space to guarantee desirable system performance. Empirical studies on Alibaba's OLTP database clusters with more than 10,000 instances show that the buffer pool consumes on average 98% of the memory space on each instance. Existing buffer pool configurations are almost unanimously based on database administrators (DBAs)' experiences and often take a fixed number of recommended values. This manual process is neither efficient nor effective, and even not feasible for large cloud clusters, especially when the workload may dynamically change on individual database instances.

To this end, we build iBTune [27], *individualized buffer tuning*, to automatically reduce buffer size for any individual database instance while still maintaining the quality of service for its response time, without relying on DBA to set forth a pre-defined level. We utilize the relationship between miss ratios and buffer pool sizes to optimize the memory allocation. Our models leverage the information from similar instances. Meanwhile, we design a novel pairwise deep neural network that uses the features from measurements on pairs of instances to predict the upper bounds of the response times. Till now, iBTune has been deployed on SDDP and applied to more than 10,000 database instances. We have successfully reduced the memory consumption by more than 17% ($\geq$ 27TB) while still satisfying the required quality of service for our diverse business applications.

Figure 9 presents an overview of iBTune's architecture and workflow. There are four key components: data collection, data processing, decision making, and execution. The iBTune workflow forms a closed cycle, since data is first collected from DBMS kernel, processed and used for training, and then resulting models are applied to the DBMS again. In *data collection*, we use customized agents to collect various database metrics and logs from DBMS. More than 1,000 metrics are collected. The agent sits outside DBMS to avoid unnecessary performance overhead to the DBMS kernel. All metrics and logs are collected in one second granularity and fed into a message queue. In *data processing*, a stream processing system reads data from the message queue and performs certain data manipulation/standardization operations such as normalization, aggregation and log transformation. After that, the processed metrics and logs are stored in a distributed data store for analysis and model training. In *decision making*, we propose a novel method to predict RT and compute the new BP (buffer pool) size. If the predicted RT meets the requirement, the computed new BP size is sent to the *execution* component, which contains an planner and a scheduler. To process a large number of database instances, *action planner* aims to make a globally efficient and non-conflicting execution plan for tens of thousands of actions. It includes priority settings among different action categories, action merging for the same instance, action conflict detection and resolution, canary executing strategy and so on. It finally outputs several action sequences to *action scheduler*. More technical details can be found in [27].

4.4.3 *Other Autonomous Scenarios*

Besides buffer size tuning, we have explored other autonomous designs as well, e.g., slow-SQL optimization, space reduction, hot/cold separation, ML optimizer, failure detection and recovery. Taking hot/cold separation as an example, the levels in X-Engine (Section 4.2.2) are differentiated by the temperature of data (extent). An extent's temperature is calculated by its access frequency in a recent window. When a compaction is performed, X-Engine selects the coldest extents with the number specified by a threshold, say 500 extents, and pushes these extents to the deeper level to do compaction. By doing so, X-Engine keeps the warm extents in upper levels and cold extents in deeper levels. But this method cannot handle dynamic workloads well. For example, when the current access frequency of an extent is low, this algorithm will treat the extent as cold data but it might become hot in near future. To this end, we have investigated machine learning based algorithms to identify the proper temperature of an extent. The intuition is that, in addition to extent, we also use row level (record) as a granularity to infer temperature (warm/cold). If a record has never been accessed in a recent window, it is identified as being cold. Otherwise, it is considered warm. As a result, temperature identification is translated into a binary classification problem and can be solved using a classification model, such as using random forest or a neural network based approach.

5. APPLICATIONS AND OPERATIONS

There have been nearly half a million database instances running on Alibaba Cloud, supporting both internal business operations within Alibaba group and external customers

business applications. By leveraging our cloud-native database systems, we have successfully served a large number of complex application scenarios.

POLARDB. POLARDB has obtained a rapid growth of population on Alibaba Cloud. It serves many leading companies in different business sectors, such as fintech, gaming and entertainment, education and multimedia. Many applications choose to migrate to POLARDB due to the limited transaction processing rate supported by their self-deployed databases. For example, an application experiences 5× latency increases and frequent transaction failures in MySQL instances during peak hours. POLARDB helps to keep all transaction latency in 1 second and improve peak throughput by 3×. In addition, one POLARDB instance is able to sustain the same query performance against a 5-node replicated MySQL cluster, and reduces the work needed by experienced DBAs. In most cases, it reduces the total cost of ownership (TCO) on databases by 50-80%.

POLARDB-X. POLARDB-X has been applied to serve many performance-critical and cost-sensitive businesses at Alibaba. For example, on the start of the Singles' Day Global Shopping Festival in 2018, we handled a 122× increase of transactions, processing up to 491,000 sales transactions per second which translate to more than 70 million database transactions per second. To this end, more than 2,000 nodes of POLARDB-X have been deployed online. The fast-rising cost of administrating OLTP databases and maintaining underlying servers has been a major challenge for Alibaba as the GMV (Gross Merchandise Volume) grows rapidly. To reduce such cost, we have replaced MySQL with POLARDB-X for many of Alibaba's businesses, which leveraged downgraded hardware (e.g., with less CPU cores and storage volumes) while sustaining the same level of QPS/TPS. In many cases, we have managed to reduce the total cost of ownership on databases by up to 50%.

AnalyticDB. AnalyticDB has been running on more than 2,000 nodes on Alibaba Cloud. It serves applications from a wide range of business sectors, such as e-commerce, fintech, logistics, public transit, and entertainment. Based on AnalyticDB, we have extended an end-to-end solution that covers the entire analysis pipeline from data acquisition to data visualization. Our customers are able to build their online analytic services seamlessly while reducing total cost compared against other solutions. AnalyticDB helps applications to better utilize their data: instant multi-dimensional analysis and business exploration for tera-scale data can be completed in milliseconds; users can define and launch their analytic tasks by invoking built-in functions and modules; the end-to-end latency for visualizing newly acquired data is reduced to less than one minute in many cases; and no manual operations from customers are required to maintain the service.

SDDP. SDDP has been used to manage over tens of thousands of database instances at Alibaba. We have successfully saved large amounts of cluster resources from the use of many autonomous and intelligent modules: SQL optimization module has detected and optimized 27.4 million inefficient SQL requests; space optimization module has freed 2.26PB storage space (e.g., by de-fragmentation and removal of useless indexes/tables); iBTune module has saved 27TB memory space while sustaining the service quality; and global workload scheduling module has increased disk utilization ratio from 46% to 63%.

6. CONCLUSION

Cloud-native database system is an increasingly important subject of research and development. It introduces numerous new technical challenges and opens many new opportunities to the database community and industry. In this paper, we have shared experiences and lessons learned at Alibaba in advancing cloud-native database techniques and building cloud-native database systems that have supported the complex and rich business operations both within and external to Alibaba group based on the Alibaba cloud. Given the rapid growth of moving to the cloud, cloud-native database systems will be even more critical in the road ahead and open new and exciting research directions and engineering challenges to support the next-generation cloud applications.

7. ACKNOWLEDGMENT

This work and the systems and techniques presented in the paper are the collective efforts and contributions from the entire Alibaba Cloud database team (officially known as the database products business unit under the Alibaba cloud intelligence group), as well as the database and storage lab under the DAMO academy (a research institute at Alibaba).

8. REFERENCES

[1] Alibaba Group. Alibaba Cloud. `https://www.alibabacloud.com`.

[2] Alibaba Group. AliSQL. `https://github.com/alibaba/AliSQL`.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *ACM SIGOPS operating systems review*, volume 37, pages 164–177. ACM, 2003.

[4] D. Borthakur et al. Hdfs architecture guide. *Hadoop Apache Project*, 53, 2008.

[5] W. Cao, Z. Liu, P. Wang, S. Chen, C. Zhu, S. Zheng, Y. Wang, and G. Ma. PolarFS: an ultra-low latency and failure resilient distributed file system for shared storage cloud database. *PVLDB*, 11(12):1849–1862, 2018.

[6] T. D. Chandra, R. Griesemer, and J. Redstone. Paxos made live: an engineering perspective. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 398–407. ACM, 2007.

[7] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, et al. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems (TOCS)*, 31(3):8, 2013.

[8] S. Das, F. Li, V. R. Narasayya, and A. C. König. Automated demand-driven resource scaling in relational database-as-a-service. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 1923–1934, New York, NY, USA, 2016. ACM.

[9] J. R. Guay Paz. *Introduction to Azure Cosmos DB*, pages 1–23. Apress, Berkeley, CA, 2018.

[10] G. Huang, X. Cheng, J. Wang, Y. Wang, D. He, T. Zhang, F. Li, S. Wang, W. Cao, and Q. Li.

X-Engine: An optimized storage engine for large-scale e-commerce transaction processing. In *SIGMOD*. ACM, 2019.

[11] J.-F. Im, K. Gopalakrishna, S. Subramaniam, M. Shrivastava, A. Tumbde, X. Jiang, J. Dai, S. Lee, N. Pawar, J. Li, et al. Pinot: Realtime olap for 530 million users. In *SIGMOD*, pages 583–594. ACM, 2018.

[12] J. Kirsch and Y. Amir. Paxos for system builders: An overview. In *Proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware*, page 3. ACM, 2008.

[13] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. kvm: the linux virtual machine monitor. In *Proceedings of the Linux symposium*, volume 1, pages 225–230, 2007.

[14] M. Kornacker, A. Behm, V. Bittorf, T. Bobrovytsky, C. Ching, A. Choi, J. Erickson, M. Grund, D. Hecht, M. Jacobs, et al. Impala: A modern, open-source sql engine for hadoop. In *CIDR*, volume 1, page 9, 2015.

[15] A. Lamb, M. Fuller, R. Varadarajan, N. Tran, B. Vandiver, L. Doshi, and C. Bear. The vertica analytic database: C-store 7 years later. *PVLDB*, 5(12):1790–1801, 2012.

[16] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2):133–169, 1998.

[17] L. Lamport et al. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[18] Z. L. Li, M. C.-J. Liang, W. He, L. Zhu, W. Dai, J. Jiang, and G. Sun. Metis: Robustly tuning tail latencies of cloud systems. In *ATC (USENIX Annual Technical Conference)*, July 2018.

[19] L. Ma, D. Van Aken, A. Hefny, G. Mezerhane, A. Pavlo, and G. J. Gordon. Query-based workload forecasting for self-driving database management systems. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 631–645, New York, NY, USA, 2018. ACM.

[20] D. Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.

[21] D. Narayanan, E. Thereska, and A. Ailamaki. Continuous resource monitoring for self-predicting dbms. In *13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 239–248, Sept 2005.

[22] P. O'Neil, E. Cheng, D. Gawlick, and E. O'Neil. The log-structured merge-tree (lsm-tree). *Acta Informatica*, 33(4):351–385, 1996.

[23] D. Ongaro and J. K. Ousterhout. In search of an understandable consensus algorithm. In *USENIX Annual Technical Conference*, pages 305–319, 2014.

[24] A. Pavlo, G. Angulo, J. Arulraj, H. Lin, J. Lin, L. Ma, P. Menon, T. Mowry, M. Perron, I. Quah, S. Santurkar, A. Tomasic, S. Toor, D. V. Aken, Z. Wang, Y. Wu, R. Xian, and T. Zhang. Self-driving database management systems. In *Proceedings of the 2017 Conference on Innovative Data Systems Research*, CIDR '17, 2017.

[25] R. Sears and R. Ramakrishnan. bLSM: A general purpose log structured merge tree. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 217–228, New York, NY, USA, 2012. ACM.

[26] R. Taft, N. El-Sayed, M. Serafini, Y. Lu, A. Aboulnaga, M. Stonebraker, R. Mayerhofer, and F. Andrade. P-Store: An elastic database system with predictive provisioning. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 205–219, New York, NY, USA, 2018. ACM.

[27] J. Tan, T. Zhang, F. Li, J. Chen, Q. Zheng, P. Zhang, H. Qiao, Y. Shi, W. Cao, and R. Zhang. iBTune: Individualized buffer tuning for largescale cloud databases. *PVLDB*, 12(12):1221–1234, 2019.

[28] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: a warehousing solution over a map-reduce framework. *PVLDB*, 2(2):1626–1629, 2009.

[29] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 1009–1024, New York, NY, USA, 2017. ACM.

[30] A. Verbitski, A. Gupta, D. Saha, M. Brahmadesam, K. Gupta, R. Mittal, S. Krishnamurthy, S. Maurice, T. Kharatishvili, and X. Bao. Amazon Aurora: Design considerations for high throughput cloud-native relational databases. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1041–1052. ACM, 2017.

[31] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 307–320. USENIX Association, 2006.

[32] F. Yang, E. Tschetter, X. Léauté, N. Ray, G. Merlino, and D. Ganguli. Druid: A real-time analytical data store. In *SIGMOD*, pages 157–168. ACM, 2014.

[33] C. Zhan, M. Su, C. Wei, X. Peng, L. Lin, S. Wang, Z. Chen, F. Li, Y. Pang, F. Zheng, and C. Chai. AnalyticDB: Realtime olap database system at alibaba cloud. *PVLDB*, 12(12), 2019.

[34] J. Zheng, Q. Lin, J. Xu, C. Wei, C. Zeng, P. Yang, and Y. Zhang. PaxosStore: high-availability storage made practical in wechat. *PVLDB*, 10(12):1730–1741, 2017.