AIDA-DB: A Data Management Architecture for the Edge and Cloud Continuum

Nuno Faria, Daniel Costa, José Pereira, Ricardo Vilaça, Luís Ferreira, Fábio Coelho
HASLab - High-Assurance Software Lab, INESC TEC & U. Minho, Portugal.
Email: {nuno.f.faria,daniel.v.costa,jose.o.pereira,ricardo.p.vilaca,luis.m.ferreira,fabio.a.coelho}@inesctec.pt

***Abstract*—There is an increasing demand for stateful edge computing for both complex Virtual Network Functions (VNFs) and application services in emerging 5G networks. Managing a mutable persistent state in the edge does however bring new architectural, performance, and dependability challenges. Not only it has to be integrated with existing cloud-based systems, but also cope with both operational and analytical workloads and be compatible with a variety of SQL and NoSQL database management systems.**

We address these challenges with AIDA-DB, a polyglot data management architecture for the edge and cloud continuum. It leverages recent development in distributed transaction processing for a reliable mutable state in operational workloads, with a flexible synchronization mechanism for efficient data collection in cloud-based analytical workloads.

***Index Terms*—Stateful edge; data management; polyglot processing; hybrid transactional analytical processing.**

I. Introduction

The advent of 5G networks and growing adoption of *Internet of Things* (IoT) devices lead to more opportunities for data collection and processing with hybrid edge-cloud systems. In this architecture, edge devices – placed near where the data is being collected/accessed – execute some of the processing while offloading other more complex work to the cloud, which is scalable on demand. However, edge-cloud architectures present several challenges when it comes to data management. Most of the difficulties are due to their inherently large-scale, distributed, and heterogeneous deployments, namely: *(i)* Replicating stateful (edge or otherwise) components for scalability requires synchronization, which can be expensive and makes fault tolerance more complex; *(ii)* large-scale data replication between the edge and the cloud raises issues in terms of network latency and storage capacity; and *(iii)* the heterogeneity of edge devices, namely in the context of IoT, encounters a very diverse set of data models, which requires data processing frameworks to handle each one on a case-by-case approach.

For example, in the network of edge-cloud services, such as firewalls and load balancers, there has been a shift to virtualizing functions to cut operation and management costs and increase elasticity [1]. However, Virtual Network Functions (VNFs) that store data [2] often rely on two techniques: use the operating system level memory sharing techniques for scalability [3] or delegate data management to a standalone centralized database [4]. The former helps to reduce the latency but fault-tolerance is harder to achieve and it limits all VNFs replicas to the same virtual machine. The latter improves scalability and fault-tolerance but increases latency. Although some solutions rely on eventual consistency to improve both scalability and latency [5], it is not enough for functions that require strong consistency or present non-deterministic behavior. However, this can be circumvented with expensive distributed locking [5].

On the other hand, the system must answer analytical queries on data collected in the edge. For example, it must be capable of efficiently answering ad-hoc queries for exploratory data analysis. One of the main challenges of this workload is that fetched data are unpredictable. Therefore, the system must be able to adapt to different workloads, by achieving an optimal tradeoff between the network and the computing power of the edge. In an environment with a substantial number of IoT devices, sending all collected data to the cloud consumes network and CPU resources on the edge, which is often limited due to cost and energy factors. Additionally, it introduces delays to the data that is actually needed at the moment. Finally, the heterogeneity of the devices and the collected data makes it difficult for the cloud to have a consistent and holistic view of data, increasing the complexity of analytical workloads.

The edge computing paradigm aims at leveraging the computational and storage capabilities of edge devices while resorting to cloud computing services for more demanding processing tasks that cannot be done at the edge. Edge devices generate large volumes of data that may need to be transferred to the cloud and that come from several types of data sources. Therefore, we propose the AIDA-DB unified data management architecture for an edge and cloud continuum, summarized in Figure 1, that is able to tackle both analytical and transactional workloads, both relying on a polyglot middleware.

In detail, the polyglot middleware processes data from different sources, which have different formats and encodings. This component is capable of efficiently performing queries over an integrated view of the data, by exploiting the underlying edge's query engines. The synchronization middleware is in charge of efficiently transferring data from the edge to the cloud. It does so by considering the data that is currently required by the cloud and the data that is already cached. Then, it synchronizes the missing data based on a balance between the network delay and the impact on the edge resources. Finally, the transactional middleware guarantees consistent

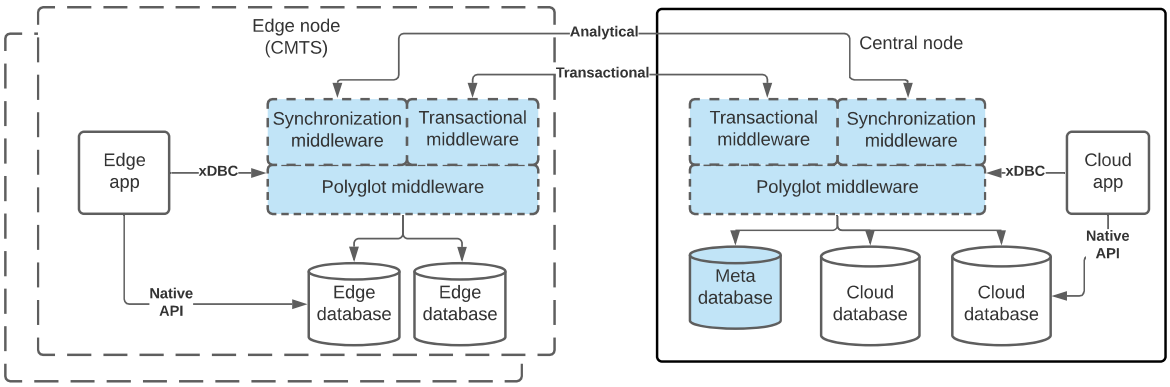

Fig. 1. Overview of the AIDA-DB data management architecture for edge and cloud continuum.

reads and isolated and atomic updates across the edge and cloud continuum.

The AIDA-DB proposal is thus a multi-faceted data management architecture for edge-cloud systems that: *(i)* exploits the edge in order to reduce processing and unnecessary data transfer to the cloud, and *(ii)* exploits the cloud in order to increase the scalability on the edge. In a nutshell, AIDA-DB combines the following contributions:

- A SQL data layer in the cloud that automatically translates and optimizes queries to be executed on the edge, taking advantage of distributed processing and reducing unnecessary data transfer (Section II-A).
- An edge-cloud synchronization algorithm that optimizes the response time by balancing the edge processing and the network transfer costs (Section II-B).
- An efficient transactional architecture that improves the scalability of shared-state edge-cloud components while reducing latency (Section III-A).
- A data management design that provides transactional guarantees to the edge without requiring it to be modified, maintaining compatibility with existing clients (Section III-B).

II. Analytical workloads

Analytical workloads are at the core of edge-cloud systems. For example, processing data collected from IoT devices [6] or managing and orchestrating a *Network Function Virtualization* (NFV) [1] infrastructure by continuously collecting metrics [7]. Thus, an efficient data management layer is a key factor in building these type of systems. However, both the heterogeneity of edge devices and the large amounts of data collected are challenges that hinder the efficiency of this layer [8]. To address this, AIDA-DB architecture includes two concepts that *(i)* ease querying data in diverse data sources and *(ii)* optimize the data synchronization between edge and cloud.

A. Querying diverse sources

In order to access data from diverse data sources, the architecture includes a layer that maps underlying stores into a common interface, based on CloudMdsQL [9]. To offer the maximum compatibility with existing analytical tools and developer know-how, we propose an implementation with a relational schema and SQL as the common interface, as implemented by [10], using PostgreSQL as the common query engine and *Foreign Data Wrappers* (FDWs) to manage the mapping.

While data is queried using a common language, this layer is still able to exploit the query engines of the underlying data stores on a case-by-case basis. This means, for example, pushing down filters, projections, aggregations, and/or even joins, to reduce the data transfer through the network and use indexes when available. Of course, these optimizations are dependent on the data store's capabilities and must be implemented manually for each one. Luckily, there is already much work pertaining to this area that can be reused [11].

B. Efficient edge-cloud synchronization

Usually, edge-cloud systems are restricted by the edge computing power, memory, and the network. In the cloud, however, these limitations are almost non-existing. Therefore, analytical queries are preferably executed in the cloud, where computing power and memory are abundant. Nonetheless, the query's execution time is usually dependent on the locality of the data and the processing power. Therefore, it is essential to keep a copy of the data in the cloud to improve query performance.

Consequently, to efficiently support edge-cloud systems, where the rate of data generated at any given time keeps increasing, we employ a data synchronization technique. Synchronization is justified because it takes advantage of cached data in the cloud to avoid re-transfers. Hence, the reduction of data transferred from the edge to the cloud reduces the execution time.

To achieve an efficient data synchronization, we employ an algorithm that executes when the current workload demands data that is missing from the cloud. With analytical workloads, where data requirements are often unpredictable, this avoids unnecessary network and computing costs. The algorithm executing is comprised of two steps: First, it generates query predicates, meant to be executed in the edge, that encompass data that is currently needed but exclude data that is already cached in the cloud. And second, it simplifies the generated

predicates to diminish their impact on the edge if their execution outweighs the reductions in network costs.

Experimental results of this algorithm show that it can adapt to simple and complex workloads alike, choosing to send the entire predicate in the former and simplifying it in the latter. Furthermore, it is shown to be a better solution than the common incremental replication – where all new data is sent – by significantly reducing the impact on the network, and than data differential algorithms such as [12], [13] – which rely on computational heavy tasks – by reducing the impact on the edge.

III. Transactional workloads

Depending on the use case, guaranteeing consistent reads and/or isolated and atomic updates is necessary to ensure the correct delivery of some edge-cloud service. Common examples are online transactional processing (OLTP) applications that must serve clients all over the globe. Here, deploying replicas on the edge is common to ensure low latency, but strongly consistent systems often operate under expensive Serializability guarantees and rely on geo-distributed transactions/synchronization protocols that hinder their performance [14], [15]. Another example is the network functions implemented through virtual means, *i.e., Virtual Network Functions* (VNFs). To meet demands, VNFs can be easily scaled, which leads to stateful functions such as *Network Address Translation* (NAT), load balancing, firewalls, among others, to require transactional guarantees to manage their shared state, as data sharding is not always possible [16]. State management performance here is especially important, as network functions must display considerably low delays [2]. On the other hand, providing transactional guarantees to the edge is also a challenge if it is already used to serve existing applications. Current solutions often disrupt the native clients' execution and increase storage overhead – by performing changes to the engine and schema – and/or offer coarse concurrency control, limiting parallelism [17].

To efficiently support transactional workloads, AIDA-DB *(i)* offers transactional isolation for multi-writer systems while ensuring low overhead on computing and storage, and *(ii)* provides the ability to perform transactions over diverse data sources without changing their engine or schema.

A. Efficient multi-site transactional execution

To efficiently support transactional guarantees in multi-site deployments, we propose a system design that avoids some of the common performance limiters found in alternative solutions. Briefly, it builds on the following concepts:

- Full-replication – although full-replication can make distributed concurrency control more complex (something we address shortly), it allows indiscriminate access to all data stored. This not only ensures low-latency reads to every object stored but also enables easier scale-up and scale-down of stateful services, by not requiring data to be moved around different sites, as imposed by some solutions [5];
- Multi-version schema and optimistic execution – we propose storing multiple versions of each data object in order to allow non-blocking reads, similarly to *Snapshot Isolation* [18]. Additionally, we propose a transactional execution that optimizes for the common scenario – where *commit* has a higher probability than *abort* – known as optimistic concurrency control [19]: Transactions execute as if they are guaranteed to succeed; Thus, we avoid the locking overhead on every write inherent from techniques such as *two-phase locking* [20], at the expense that conflicts are not preemptively detected;
- Logically centralized certification and recovery – instead of relying on synchronization techniques such as geo-distributed *two-phase commit*, consensus, or locking, our design considers a centralized certification deployed on the cloud which requires, at most, one wide-area network (WAN) round trip. Additionally, this component stores the most recent copy of all data in the system with high-availability guarantees, removing that responsibility from the edge, greatly easing fault-tolerance, which is a common concern in, for example, stateful VNFs [2]. Even though this component is logically centralized in the cloud, it can still be physically distributed for fault tolerance and to hide failover. However, a key difference from common solutions is that the deployment is done with low latencies among replicas (*e.g.,* in the same data center), considerably reducing response times. To avoid becoming a bottleneck, the certification maximizes parallelism and uses efficient relational algorithms for conflict detection;
- Heterogeneous edge – we can exploit the centralized certification to allow each edge to store data using different data models and provide different interfaces. Our only requirement is for the certification package sent to the cloud certifier to be consistent across all nodes. This leads to a higher flexibility of our design, which is especially useful if we consider our edge to be IoT devices with different architectures and protocols.
- Asynchronous replication – although Serializable guarantees are useful for some strict use cases, we can exploit asynchronous replication and local access patterns in order to greatly improve performance for the plenitude of other cases. We thus propose the usage of *Parallel Snapshot Isolation* (PSI) [21] as our main isolation criteria, in which data is eventually replicated to all sites after commit. Furthermore, to reduce conflict probability, this criteria does not force different sites to advance in the same order. Instead, transactions local to some site are immediately applied after committing in the certifier, meaning remote transactions have no impact on response time. If we assume that clients follow local data access patterns, which is common in many OLTP applications and alike, collision probability among edge nodes is greatly reduced. Even though data is eventually replicated, strong consistency is still ensured by certifying against all previously committed data. Although greatly

reducing response times, eventual replication can be challenging in some contexts, namely networking, even if identified to be often sufficient [22]. For instance, while updating the load value of some server in a load balancer can easily be asynchronous – as it can lead to, at most, inefficient utilization of the available resources – the same thing is not true for a NAT function, where we do not want different sites to see different values. For those cases, we can default to regular Snapshot Isolation;

- Low-footprint time management – finally, we integrate efficient time management in our solution. Instead of using PSI as implemented by Walter [21], where each object is tagged with one timestamp for every site in the system, we only consider the usage of two timestamps independently of the number of sites used, as specified by *Totally-Ordered Prefix Parallel Snapshot Isolation* (TOPSI) [23]. Briefly, it considers a local timestamp, which is used to compute snapshots for the clients, and a global timestamp, used for transaction certification and replication. This not only leads to a smaller storage overhead as the scale of the system increases but also reduces the time management complexity when we add or remove sites.

These concepts provide the foundation of a distributed-first transactional system that prioritizes flexibility, scalability, and low response times while ensuring strong consistency.

B. Transactional isolation without disruption

Although the design in the previous section enables strong transactional guarantees with low overhead, it requires storing metadata (*i.e.,* modifying the schema) and running custom middleware code in each edge. This can be unfeasible for analytical-first systems which require some transactional guarantees only for reliability, as it can lead to the disruption of the normal service.

We thus propose another transactional layer that requires no modification to the edge, by leveraging a cloud cache and query-engine layer [24]. Briefly, write-sets of temporary transactions are kept in the cloud; After commit, the data gets transferred to a persistent table in the cache, still in the cloud, tagged with the respective commit timestamp. Eventually, every object with a timestamp $\leq \delta$ is moved to the edge, without the attached metadata and replacing older versions. When reading, a transaction joins both the data cached in the cloud and the data in the edge and filters objects that it cannot read based on its starting timestamp. If a version only exists in the edge, it is automatically readable by any current transaction (which has a start timestamp $\geq \delta$), given its timestamp can be inferred to be $\leq \delta$.

With this proposal, the native edge clients can execute unaware of the transactional layer. Furthermore, this layer can reuse both the centralized certifier of Section III-A, acting as its "edge", and the polyglot layer of Section II-A to handle data translation. There is still the challenge of ensuring consistency of concurrent native and transactional clients, but assuming they update disjoint data, this solution is enough.

IV. Related work

Network Functions (NFs) explore virtualized network components that substitute dedicated hardware devices, anywhere in the edge-cloud continuum. This shift allows for remote deployment and orchestration of these services, as well as to provide them in an elastic manner. NFs form processing chains (*e.g.,* layer1: load-balancer; layer2: 2 firewall nodes, layer3: 3 IDS instances), which need to share state among themselves (*e.g.,* maintaining standby nodes synchronized with an active node for failover). NFs present two alternatives for state management, namely stateful and stateless NFs. In stateful NFs, state management is done by the NF itself. This reduces read/write latency, as the data is closer to the execution, at the expense that it reduces scalability and elasticity, as data must be split/replicated among NFs [3], [25]. As an alternative, stateless NFs offload data management responsibilities to an external data store [4], [26]. Externalizing data management allows NFs to seamlessly failover and scale, since data synchronization is handled by the data store. However, external data stores increase access latency and can install a bottleneck even when using an in-memory database management system such as Redis [2]. In our architecture, we provide a hybrid solution that considers the best of both alternatives: cache the data in each NF (edge) to provide low latency access, and offload the complex data management functions – such as transactional certification and replication – to the cloud, to ease the edge scalability and reduce its load.

Efficient data synchronization is fundamental to ensure low-overhead data transfer between the edge-cloud continuum. One the of most simple algorithms is to tag each object with an ever-increasing timestamp and send all new data since the last synchronization [27], which is computationally cheap but requires more storage. More complex algorithms [12], [13] use differential algorithms in order to avoid storage overheads, but are more computationally complex, thus being less viable to the edge. Independently of how new data is detected, these algorithms transfer data unaware of the workload, which can cause unnecessary delays. In contrast, our architecture relies on an algorithm that aims to send only the new data that is needed by the current workload, reducing superfluous transfers.

To provide transactional isolation to heterogeneous sources, Polypheny-DB [17] uses *two-phase locking* with coarse granularity, which limits parallelism. In contrast, a proposal of the CloudMdsQL project [9] aims at *Snapshot Isolation* with fine-grained concurrency control but relies on the implementation, from scratch, of custom wrappers and possibly changes to the core of each datastore. Our architecture instead achieves low granularity with no disruption to the underlying storage, by leveraging a cache that stores newly committed data and SQL code to reconstruct the snapshot.

There are several research projects that explore the qualitative differences between the different layers formed by edge nodes. These differences manifest themselves as varying distances to the user, wide geographical dispersion, and

varying levels of resource capability. These allow to improve both performance and privacy control, but also impose new difficulties for development, and for ensuring secure operation. DITAS [28] explores privacy enforcement in the edge, by allowing data owners to specify how and where their data can be processed (*i.e.,* node filtering). They also employ Service Level Agreements (SLAs) defined by the user, which trigger computation and data replication/movement between nodes when those agreements are not being met. PrEstoCloud [29] focuses on big data streams in edge space. CHARIOT [30] and SOFIE [31] focus on security aspects of IoT systems that run in the Edge. In contrast, our work did not make explicit divisions between different edge nodes nor specified how security/privacy mechanisms should be implemented in them. Instead, our architecture can in practice be extended with the concepts where cited.

Comparatively to the edge, the cloud space is much more developed and mature. Techniques can be leveraged from these systems to empower data processing and storage at the edge level. Within the cloud environment, geo-replicated data store solutions share similarities with edge systems, in that nodes can be at a significant distance from each other, as these systems try to minimize the impact of intra-replica communication on the performance of the databases. Techniques such as caching [32] [33], optimizing shard allocation [34] [35] [36], enabling control over data locality [37] [14] or employing layering of data [38] can all be adapted into our architecture to further improve its scalability and performance.

V. Discussion

Our proposal of the AIDA-DB architecture is aimed at enabling emerging complex VNFs and edge-based application services in emerging 5G networks to manage data across a cloud and edge continuum. This approach has been applied to a distributed revenue assurance system, that employs both analytical processes for triggering events as well as workflows to deal with them that span the cloud and edge reliably and efficiently. The key advantages of AIDA-DB which allow this are:

a) Global transactional reliability: Managing distributed data across distributed system boundaries and multiple SQL and NoSQL database systems is hard and prone to inefficiencies and errors. By providing a seamless transactional layer, AIDA-DB ensures that business-critical workflows execute across these boundaries and makes edge-based application components and services first-class citizens in complex distributed applications.

b) Hybrid transactional-analytical processing: The edge adds an axis of complexity to traditional extract, transform, and load (ETL) procedures, as data needs to be copied across the network to a centralized data warehouse. In contrast, a key driver for the adoption of the edge is to make use of fresher data in applications and services. Therefore, in AIDA-DB we strive for hybrid transactional analytical processing systems that take advantage of cloud elasticity and automatic synchronization to run interactive analytics, such as needed for data exploration, on live or freshly updated data.

c) Polyglot processing with a standard API: Although it has become clear that current applications and services need a variety of data management paradigms and tools, this pushes additional complexity into applications and calls for polystores to bridge between them [39], [40]. AIDA-DB innovates by catering for polyglot workloads, across different systems, for both transactional and analytical workloads while exposing all functionality in a standard xDBC interface that accepts polyglot queries.

d) Separation of concerns: Abstract management of the cloud and edge divide and support for the SQL query language makes it possible to separate application development concerns – what functionality is needed – from deployment and optimization – how it is provided. This extends to the cloud and edge computing environment the traditional strength of database management systems of providing separate and declarative interfaces for both developers and database administrators (DBAs), making applications reusable and deployable in different contexts. This is particularly suited for platforms as a service (PaaS) as it allows the platform provider to offer managed services.

Acknowledgments

Special thanks to the anonymous reviewers for their helpful feedback. Partially funded by project AIDA – Adaptive, Intelligent and Distributed Assurance Platform (POCI-01-0247-FEDER-045907) co-financed by the European Regional Development Fund (ERDF) through the Operational Program for Competitiveness and Internationalisation (COMPETE 2020) and by the Portuguese Foundation for Science and Technology (FCT) under CMU Portugal.

References

[1] M. Chiosi *et al.*, "Network functions virtualisation—introductory white paper network," 2012.

[2] S. G. Kulkarni, K. Ramakrishnan, and T. Wood, "Managing state for failure resiliency in network function virtualization," in *2020 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN*. IEEE, 2020, pp. 1–6.

[3] S. Rajagopalan, D. Williams, and H. Jamjoom, "Pico replication: A high availability framework for middleboxes," in *Proceedings of the 4th annual Symposium on Cloud Computing*, 2013, pp. 1–15.

[4] M. Kablan, A. Alsudais, E. Keller, and F. Le, "Stateless network functions: Breaking the tight coupling of state and processing," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 97–112.

[5] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield, "Split/merge: System support for elastic execution in virtual middleboxes," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, 2013, pp. 227–240.

[6] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[7] F. Z. Yousaf, V. Sciancalepore, M. Liebsch, and X. Costa-Perez, "MANOaaS: A multi-tenant NFV MANO for 5G network slices," *IEEE Communications Magazine*, vol. 57, no. 5, pp. 103–109, 2019.

[8] A. Botta, W. De Donato, V. Persico, and A. Pescapé, "Integration of cloud computing and internet of things: a survey," *Future generation computer systems*, vol. 56, pp. 684–700, 2016.

[9] B. Kolev, P. Valduriez, C. Bondiombouy, R. Jiménez-Peris, R. Pau, and J. Pereira, "CloudMdsQL: querying heterogeneous cloud data stores with a common language," *Distributed and parallel databases*, vol. 34, no. 4, pp. 463–503, 2016.

[10] A. N. Alonso, J. Abreu, D. Nunes, A. Vieira, L. Santos, T. Soares, and J. Pereira, "Building a polyglot data access layer for a low-code application development platform," in *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer, 2020, pp. 95–103.

[11] T. P. G. D. Group, "PostgreSQL Wiki - foreign data wrappers," 2021. [Online]. Available: https://wiki.postgresql.org/wiki/Foreign_data_wrappers

[12] Y. Minsky, A. Trachtenberg, and R. Zippel, "Set reconciliation with nearly optimal communication complexity," *IEEE Transactions on Information Theory*, vol. 49, no. 9, pp. 2213–2218, 2003.

[13] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," in *Advances in Cryptology - EUROCRYPT 2004*, C. Cachin and J. L. Camenisch, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 523–540.

[14] 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)*, vol. 31, no. 3, pp. 1–22, 2013.

[15] R. Taft, I. Sharif, A. Matei, N. VanBenschoten, J. Lewis, T. Grieger, K. Niemi, A. Woods, A. Birzin, R. Poss *et al.*, "CockroachDB: The resilient geo-distributed SQL database," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 1493–1509.

[16] V. Gasiunas, D. Dominguez-Sal, R. Acker, A. Avitzur, I. Bronshtein, R. Chen, E. Ginot, N. Martinez-Bazan, M. Müller, A. Nozdrin *et al.*, "Fiber-based architecture for NFV cloud databases," *Proceedings of the VLDB Endowment*, vol. 10, no. 12, pp. 1682–1693, 2017.

[17] M. Vogt, A. Stiemer, and H. Schuldt, "Polypheny-DB: towards a distributed and self-adaptive polystore," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 3364–3373.

[18] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil, "A critique of ANSI SQL isolation levels," *ACM SIGMOD Record*, vol. 24, no. 2, pp. 1–10, 1995.

[19] H.-T. Kung and J. T. Robinson, "On optimistic methods for concurrency control," *ACM Transactions on Database Systems (TODS)*, vol. 6, no. 2, pp. 213–226, 1981.

[20] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency control and recovery in database systems*. Addison-wesley Reading, 1987, vol. 370.

[21] Y. Sovran, R. Power, M. K. Aguilera, and J. Li, "Transactional storage for geo-replicated systems," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, 2011, pp. 385–400.

[22] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "ONOS: towards an open, distributed SDN OS," in *Proceedings of the third workshop on Hot topics in software defined networking*, 2014, pp. 1–6.

[23] N. Faria and J. Pereira, "Totally-ordered prefix parallel snapshot isolation," in *Proceedings of the 8th Workshop on Principles and Practice of Consistency for Distributed Data*, ser. PaPoC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3447865.3457966

[24] N. Faria, J. Pereira, A. N. Alonso, and R. Vilaça, "Towards generic fine-grained transaction isolation in polystores," in *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*. Springer International Publishing, 2022.

[25] S. G. Kulkarni, G. Liu, K. Ramakrishnan, M. Arumaithurai, T. Wood, and X. Fu, "Reinforce: Achieving efficient failure resiliency for network function virtualization based services," in *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*, 2018, pp. 41–53.

[26] J. Khalid and A. Akella, "Correctness and performance for stateful chained network functions," in *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, 2019, pp. 501–516.

[27] P. Bellavista and A. Corradi, *The Handbook of Mobile Middleware*. USA: Auerbach Publications, 2006.

[28] C. E. research results, "DITAS: Data-intensive applications improvement by moving data and computation in mixed cloud/fog environments," 2019. [Online]. Available: https://cordis.europa.eu/project/id/731945/results

[29] PrEstoCloud, "PrEstoCloud: Proactive cloud resources management at the edge for efficient real-time big data processing," 2018. [Online]. Available: https://prestocloud-project.eu

[30] CHARIOT, "CHARIOT: Privacy, security and safety of IoT," 2018. [Online]. Available: https://www.chariotproject.eu

[31] SOFIE, "SOFIE: Secure open federation for internet everywhere," 2018. [Online]. Available: https://www.sofie-iot.eu

[32] N. Bronson, Z. Amsden, G. Cabrera, P. Chakka, P. Dimov, H. Ding, J. Ferris, A. Giardullo, S. Kulkarni, H. Li *et al.*, "TAO: Facebook's distributed data store for the social graph," in *2013 {USENIX} Annual Technical Conference ({USENIX}{ATC} 13)*, 2013, pp. 49–60.

[33] K. Ngo, H. Lu, and W. Lloyd, "K2: Reading quickly from storage across many datacenters," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2021, pp. 199–211.

[34] C. Curino, E. P. C. Jones, Y. Zhang, and S. R. Madden, "Schism: a workload-driven approach to database replication and partitioning," 2010.

[35] M. S. Ardekani and D. B. Terry, "A self-configurable geo-replicated cloud storage system," in *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, 2014, pp. 367–381.

[36] M. Annamalai, K. Ravichandran, H. Srinivas, I. Zinkovsky, L. Pan, T. Savor, D. Nagle, and M. Stumm, "Sharding the shards: managing datastore locality at scale with akkio," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 445–460.

[37] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, pp. 1–26, 2008.

[38] N. Tran, M. K. Aguilera, and M. Balakrishnan, "Online migration for geo-distributed storage systems." in *USENIX Annual Technical Conference*, 2011.

[39] M. Stonebraker and U. Cetintemel, ""one size fits all": An idea whose time has come and gone," in *Proceedings of the 21st International Conference on Data Engineering*, ser. ICDE '05. USA: IEEE Computer Society, 2005, p. 2–11. [Online]. Available: https://doi.org/10.1109/ICDE.2005.1

[40] M. Stonebraker, "The case for polystores," ACM SIGMOD Blog, 2015. [Online]. Available: https://wp.sigmod.org/?p=1629