Polypheny-DB: Towards a Distributed and Self-Adaptive Polystore

Marco Vogt Alexander Stiemer Heiko Schuldt
Department of Mathematics and Computer Science
University of Basel, Switzerland
{*firstname.lastname*}*@unibas.ch*

***Abstract*—Cloud providers are more and more confronted with very diverse and heterogeneous requirements their customers impose on the management of data. First, these requirements stem from service-level agreements that specify a desired degree of availability and a guaranteed latency. As a consequence, Cloud providers *replicate* data across data centers or availability zones and/or *partition* data and place it close to the location of their customers. Second, the workload at each Cloud data center or availability zone is diverse and may significantly change over time – e. g., an OLTP workload during regular business hours and OLAP analyzes over night. For this, *polystore* and *multistore databases* have recently been introduced as they are intrinsically able to cope with such mixed and varying workloads. While the problem of heterogeneous requirements on data management in the Cloud is either addressed at global level by replicating and partitioning data across data centers or at local level by providing polystore systems in a Cloud data center, there is no integrated solution that leverages the benefits of both approaches. In this paper, we present the Polypheny-DB vision of a distributed polystore system that seamlessly combines replication and partitioning with local polystores and that is able to dynamically adapt all parts of the system when the workload changes. We present the basic building blocks for both parts of the system and we discuss open challenges towards the implementation of the Polypheny-DB vision.**

***Keywords*-Polystore database; data replication; data partitioning.**

I. Introduction

With an increasing number of companies and organizations moving their entire IT infrastructure to the Cloud, also the challenges Cloud providers have to cope with become more and more diverse and demanding. Especially in the Big Data era, this comes with various requirements on the data stored in the Cloud and the characteristics when accessing this data. From a Cloud users' perspective, these characteristics are governed by service-level agreements (SLAs). Usually, SLAs are motivated by legal constraints and performance expectations and specify the desired level of availability, consistency, and the accepted latency. In terms of availability, data is replicated across different data centers or across availability zones. In terms of consistency in the presence of replication, usually the level according to which consistency is relaxed is specified in SLAs. This is necessary since, according to the CAP theorem [1], one out of the three properties *C*onsistency, *A*vailability, and tolerance to network *P*artitions cannot be enforced in case of errors [2]; in most cases, a relaxed level of consistency (e. g., eventual consistency) is chosen. In terms of latency, Cloud providers aim at *partitioning* [3] the data and at moving data partitions close to their users. Again, this is a property that cannot be considered in isolation since there is a trade-off between latency and consistency, as governed by PACELC, an extension to CAP ("if there is a partition (P), how does the system trade off availability and consistency (A and C); else (E), when the system is running normally in the absence of partitions, how does the system trade off latency (L) and consistency (C)" [4]). Hence, a joint consideration of all elements of the SLAs of a Cloud user require to combine data replication and partitioning (potentially with different replication degrees for different partitions, as not all data has the same importance or is not equally often accessed), and to dynamically adapt either the partitions or the replicas (number and placement) if the access characteristics change.

While data replication and partitioning are addressed at global level, across data centers and availability zones, the additional challenges individual nodes in the Cloud have to face stem from heterogeneous and changing workloads. Examples can be found in applications that combine OLTP workloads during normal business hours and OLAP-style analyzes during times when business is closed, or in applications which combine graph data with unstructured data or data that does not adhere to any schema. According to [5], such mixed workloads cannot be properly addressed with a single system. Despite of the fact that the last years have seen the emergence of several special-purpose databases, each of these systems alone is not able to address heterogeneous, mixed workloads. Therefore, *multistore* and *polystore databases* have become popular as they combine different specialized database systems and are thus well suited to cope with heterogeneous workloads. This also allows to offer different query languages and interfaces and thus to tailor data management and access to the need of applications and the characteristics of the data to be accessed.

In order to properly address the requirements of Cloud users, SLA-based replication and partitioning need to be applied at global level, while the heterogeneity of the access workload and the different characteristics of data need to be addressed at local level, by applying polystores or multistores at the individual nodes of a Cloud data center or availability zone, respectively. However, even though several

of these tools and techniques are already used individually by Cloud providers, to the best of our knowledge, such combination leveraging the advantages at global Cloud level and at local data center node level are not yet exploited.

In this paper, we introduce the Polypheny-DB vision of a distributed and adaptive data management in the Cloud. Polypheny-DB seamlessly combines cost-based data replication and partitioning at global level, across data centers, and polystore databases at local level, i. e., at the individual Cloud nodes. A particular feature of Polypheny-DB is the ability to dynamically adopt to changing workloads and user requirements. At global level, this means that the size of partitions and their placement is continuously assessed and changed, if necessary. The same holds for the replication degree and the location of replicas. At local level, this includes the selection of database systems available locally, underneath the polystore deployed at a Cloud node, the storage medium used, or the placement of data across the underlying stores of the polystore. In addition, a special feature of Polypheny-DB is that it also combines vector-space style similarity queries, especially in multimedia collections with Boolean queries, for instance on the structured metadata of such multimedia collections. We identify the necessary building blocks required for the Polypheny-DB vision, out of which we have already provided a large share as part of our previous work and we discuss the open challenges towards the Polypheny-DB vision.

The contribution of this paper is twofold: first, we introduce the Polypheny-DB vision of a system for distributed and adaptive data management in the Cloud that combines cost-based data replication and partitioning in the large with flexibility by applying polystores in the small and that is able to automatically adapt, at both levels, to changing workloads. Second, we identify the building blocks needed for Polypheny-DB and we discuss the open challenges stemming from the Polypheny-DB vision.

The remainder of this paper is structured as follows: Section II presents a sample application that shows the need to combine flexible data management at both global and local level in the Cloud. Section III introduces the Polypheny-DB vision and the challenges to be faced at global level (Section III-A), at local level (Section III-B), and the cross-level challenges stemming from dynamically changing workloads (Section III-C). Section IV discusses related work and Section V concludes and summarizes open challenges towards the full implementation of the Polypheny-DB vision.

II. Motivating Scenario

In this section, we present a scenario that shows the need for a data management solution that combines SLA-based replication and partitioning with polystores and that extends the discussion provided in [6].

Consider, as an example, “Gavel”, a global online auction house. Originally founded as an auction house for the European market, Gavel is now present in all major markets worldwide and has deployed its servers, thanks to the Cloud, at several locations around the globe. Besides specific application servers, Gavel also runs database servers at each of these Cloud locations.

Assume the following business model: Gavel takes 1 % of the price of each item sold. They apply sophisticated analyzes of their users’ behavior and other statistics (e. g., final auction prices, bidding histories of users, or histories of visited auctions) to advertise their auctions. The analytical database queries they run for this purpose always need the latest data, which makes an offline approach infeasible.

As a consequence, the overall workload of Gavel is very diverse and contains both transactional and analytical queries. Gavel wants to use a product recommendation system which requires parts of the data to be represented as a graph. They also plan to deploy an application for business analytics which works on multidimensional OLAP cubes. Gavel therefore requires a DBMS being capable of storing and retrieving the data based on different data models at the same time and to be queried using different query languages.

Besides the different data models, Gavel also requires the database system to support advanced features such as location-based queries to support functions like searching for auctions within a certain distance to the address of the customer. Another more advanced database feature is the support for handling temporal data. Because of legal reasons, Gavel has to store, for instance, all versions of auction descriptions in case they are edited.

Another feature Gavel would like to implemented in their online platform is similarity search for images. This should allow customers to upload a picture and find auctions which offer a (visually) similar product. Gavel therefore requires a database system being able to deal with this type of data and providing support for similarity search in image collections.

Not all data stored in the Gavel database is of equal importance. While customer data, auctions, and bids are of high importance and have to be stored in a redundant and consistent way, other data like the access logs of user requests can be partitioned and do not need to be stored multiple times on different Cloud nodes.

It is a characteristic feature of auctions that they are most interesting just before they end. This results in soon ending auctions being significantly more frequently queried than other auctions. This should be leveraged for storing such frequently queried auctions in main memory and old auctions on slow archive storage and eventually lead to both an increase of the performance and a reduction of the costs.

Due to shipping costs, language barriers, and different time zones, most customers only bid on auctions in their local Gavel shop. This results in a subset of the data nearly always being updated and deleted at the same location while other parts of the data are used at all locations equally. There is also data which is nearly never updated but heavily read

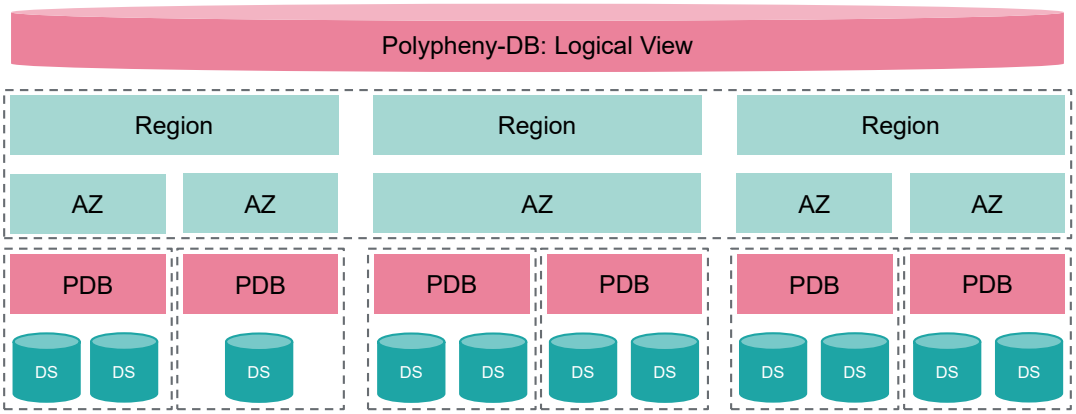

Figure 1. Logical view of Polypheny-DB. Data is physically distributed over more regions. Every region contains one or multiple availability zones (AZ). Every availability zone contains one Polypheny-DB instance (PDB) which has one or multiple underlying data stores on which the data is actually stored.

(e. g., categories). This circumstance should be used to make Gavel more robust against network partitioning.

At every data center location, Gavel operates multiple database servers which are placed in different availability zones. While the availability zones of a data center are interconnected quite well, there are huge differences in the available bandwidths and the latencies between data centers. A distributed DBMS has to take this into account by being able to dynamically react to changes.

However, besides the financial and technical aspects (performance, latency) there are also legal aspects which have to be considered. Because of the General Data Protection Regulation (GDPR) (or other similar regulations), for instance, Gavel is not allowed to store sensitive data in other countries outside of the EU. Furthermore, there are legal regulations regarding data safety which, for instance, require some portions of the data to be replicated.

Gavel's research on user behavior has shown that there is a direct correlation between the performance of the web application and the number of biddings. Therefore, Gavel has an interest in an autonomous cost-based optimization and adaption of the database system at runtime to achieve both: high performance under high workload and low resource costs in times of low workload.

Such heterogeneous and diverse requirements are neither addressed with a single off-the-shelf database system nor with standard techniques already available from Cloud providers and thus necessitate a holistic solution addressing global data replication and partitioning with several local databases combined in a flexible way.

III. Polypheny-DB

The key idea behind Polypheny-DB is to combine the flexibility of self-adaptive and data model agnostic polystores (see Figure 1) with the power of adaptive data management protocols. With this, we will bridge the gap between optimization "in the small" and "in the large" to get higher throughput, lower latency, and a larger set of features at minimal costs – compared to both optimizations alone.

A. Cost-based Data Partitioning and Replication

At the global level, Polypheny-DB will be organized as a cluster of sites of equal importance. The Polypheny-DB cluster has no dedicated master and follows a read-anywhere and update-anywhere [7] approach.

In contrast to NoSQL systems favoring availability over data consistency [8], Polypheny-DB will, like NewSQL systems [9], provide ACID guarantees [10], [11]. This comes with challenges like the proper synchronization and propagation of updates while still keeping the throughput as high as possible. This coordination can be done using coordination protocols like the two-phase commit (2PC) [12] or three-phase commit (3PC) [13] protocol, or by using consensus protocols like Paxos [14] or Raft [15] [16]. Polypheny-DB will use the 2PC protocol for coordination since it requires less messages compared to 3PC or Paxos knowing the drawback of the blocking behavior for failure cases.

The following subsections discuss Polypheny-DB's cost-based data partitioning and replication pipeline.

1) Data Partitioning: In the need of dealing with the "volume" of Big Data [17], data partitioning (a.k.a. data fragmentation or data sharding) is able to distribute the data across several sites. Additionally, if partitioned in a proper way, data partitioning balances the load on the sites and increases the overall throughput by reducing the number of distributed transactions.

The actual partitioning of the data can be done horizontally (e. g., by distributing records), vertically (e. g., by distributing attributes), or in a hybrid way, i. e., a combination of horizontal and vertical partitioning.

In addition, different approaches for building the data partitions exist: partitions can be build, for example, by using value ranges of attributes, by using graph partitioning algorithms by first organizing transactions or the workload as

a graph (e. g., Schism [18]), or by building an affinity matrix encoding the "connection" of transactions to attributes or tuples (e. g., AutoStore [19]). Recently, also machine learning approaches have been proposed (e. g., GridFormation [20]).

Based on our previous work on Cumulus [21], Polypheny-DB will be able to partition data explicitly based on user-defined flags on attributes resulting in horizontal partitions based on value ranges, or implicitly by collecting access patterns of the workload occurring on the Polypheny-DB sites resulting in hybrid partitions aiming on the minimization of the overall query response latency.

2) Data Replication: The second step is the identification of partitions which are subject to replication. This can have several reasons: First, if the workload analysis identified a partition which is mostly read from, load balancing can be exploited by replicating this partition. Second, requirements may exist demanding a certain availability for a specific data partition. Such demands are typically raised by high availability applications or for data safety reasons (keeping a certain number of copies).

However, replicating a partition comes at the cost of additional overhead for update transactions since the partitions have to be kept consistent. To ensure this data correctness, it is important that the set of sites participating in a read transactions intersect with the set of sites participating in a write transaction. This intersection property [22] is required for having the possibility to get the latest data. However, note that this also depends on the applied concurrency control model. These sets of sites are called *quorums* and various quorum protocols exist: Read-One-Write-All (ROWA) [23], Weighted Voting [24] and its instance Majority Quorum [25], Tree-based [26], and Grid-based Quorum protocols [27].

Polypheny-DB will be able to choose from a variety of replication protocols to select the best option satisfying the requirements. Partitions will be replicated autonomously if it can exploit the benefits of load balancing or they will be replicated based on user or application preference (e. g., minimum amount of copies). Further and based on our previous work on QuAD [28], Polypheny-DB will use novel adaptive replication protocols supporting changing requirements and dynamic environments.

3) Data Allocation: The third and final step of the pipeline in the large is the allocation of data to the Polypheny-DB sites. The allocation step focuses primarily on load balancing, storage capacities, network distances, and further policies. In general, the allocation step again tries to minimize the overall query response latency without violating constrains introduced by the system, its environment, SLAs, or policies.

In a first version, Polypheny-DB will use a static allocation technique; yet, because of the dynamic environment, further versions of Polypheny-DB will use a dynamic allocation strategy.

4) Cost-based Data Management: All three techniques of the data management pipeline in the large presented above are driven by an underlying cost model. This model attempts to reflect the benefits and drawbacks of each technique: For example, the cost model for data replication considers the increasing coordination overhead for each added replica versus the increase of the availability of a data partition.

For Polypheny-DB, we will extend the BEOWULF cost model introduced in [29] which combines two replication protocols with a data partitioning protocol to a comprehensive cost model integrating all three pipeline steps, namely data partitioning, data replication, and data allocation.

Further, the cost model contains parameters reflecting the Service Level Objectives (SLOs) derived from SLAs, like the desired availability, maximum monetary costs, or minimum throughput. To do so, a translation from SLAs to SLOs is required like the approach introduced in [30].

B. Polystore

This section focuses on the actual storage and retrieval of data on the individual instances of Polypheny-DB. In the following, we use the taxonomy introduced in [31].

The idea of *polyglot persistence* is to choose the right tool (query language) for a concrete use case: when storing data used by different types of applications, it is beneficial to also use different languages for retrieving the data. This idea has its roots in the concept of *polyglot programming*, which takes advantage of the fact that different languages are suitable for solving different problems. According to [31], a polyglot system uses a set of *homogeneous* data stores and exposes multiple query interfaces and languages.

Multistore systems are systems which manage data across *heterogeneous* data stores. All data is accessed through one single query interface which supports one query language.

The idea of a polystore is to combine the advantages of polyglot persistence and multistore database systems.

As depicted in Figure 2, in Polypheny-DB, clients submit queries using different query languages. In addition, Polypheny-DB stores data in a set of heterogeneous data stores.

In the following we will have a closer look into the most important parts of the local data management within the Polypheny-DB vision.

1) Query Interfaces: Building a database system with a maximal degree of flexibility already starts at the level of the supported query languages. Polypheny-DB will support a wide range of different query languages based on different data models and through different query interfaces. A key feature of Polypheny-DB is that all data is available through all query interfaces and using all supported query languages. Polypheny-DB will feature at least one SQL dialect and a simple Create, Read, Update, Delete (CRUD) [32] query language, but also other query languages such as a graph-based query language like openCypher [33] are planned. For this, incoming queries are translated into an algebraic tree

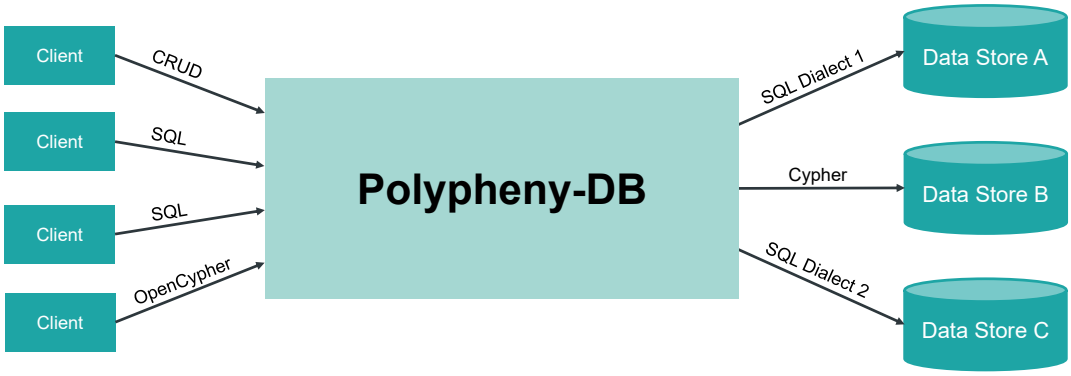

Figure 2. Query interfaces used to communicate with Polypheny-DB from outside and interfaces within the system.

representation, for instance following [34] which introduces a mathematical model unifying sets (SQL), graphs (NoSQL), and matrices (NewSQL) based on associative arrays.

2) Data Storage: Based on the assumption that some data stores are better suited for a specific scenario than others, we build Polypheny-DB in a polystore manner by combining multiple heterogeneous data stores with different characteristics. We even go a step further and create a pool of very specialized data store connectors which can be loaded and unloaded at runtime.

In our previous work [6], we have introduced a multistore which connects to different relational, SQL-based data stores. With Polypheny-DB we will extend the set of supported data stores by adding document stores and graph stores. The idea is to support as many specialized database systems as possible. As depicted in Figure 3, the data stores can either be located on the same physical node as the Polypheny-DB instance itself or on another node within the same availability zone. This allows to easily scale the capacity of the logical Polypheny-DB instance by adding additional physical nodes.

Depending on the type of data store, it can be deployed using different types of storage media, as the latter have a strong influence on the performance of a data store. Because there are significant differences w.r.t. the costs of a slow spinning disk and fast main memory, cost models will address this trade-off by storing the frequently used (hot) data items in stores running on fast (but expensive) storage and the only rarely queried (cold) data items on data stores running on slow but rather cheap storage. This is especially important when Polypheny-DB is deployed in a Cloud with a *pay-as-you-go* model. Furthermore, it also depends on the constraints by the user if the system is allowed to store data only on volatile memory.

As depicted in Figure 3, there is one special store located within Polypheny-DB. This store deals with one of the major drawbacks of this kind of polystore (see [6]): short-running and simple transactional queries. While the additional overhead introduced by the polystore is not significant for long running queries, it can be for short-running transactional queries. Hence, this is addressed by having a data store which stores frequently used data and which is integrated into Polypheny-DB.

Because the underlying data stores require the query to be expressed using different query languages and are based on different internal data models, Polypheny-DB has to translate from its algebraic query representation into the native query languages of the underlying data stores, including translations between different SQL dialects [6] and mapping between different data models.

The available pool of specialized data stores will allow Polypheny-DB to deal with a wide range of possible workloads. Furthermore, the possibility to add and remove data stores at runtime will allow to adapt Polypheny-DB to changes in the workload. Besides a manual configuration, we also plan in a second step to enable Polypheny-DB to automatically adapt itself by adding and removing data stores.

3) Adaptive Data Placement: A key element of Polypheny-DB is the dynamic placement of data items across the underlying data stores based on the workload. This can either be done by placing all data on all stores (e. g., Icarus [6]) or only on one store (e. g., MuSQLE [35]).

A benefit of storing all data on all stores is the possibility to decide for every query which store has the best characteristics for executing it. While this is a good approach for scenarios with no or only a small amount of data manipulation queries, it has massive drawbacks for scenarios which include frequent changes of the data due to the required overhead for keeping all data stores consistent. Also, the redundant data storage results in massive waste of storage space, especially in big data scenarios. In such scenarios, we further have the problem that the smallest data store defines the maximum capacity of the database system.

Storing data only on one store solves these problems. However, this comes with several other drawbacks: while in a fully-replicated environment the system can always choose a data store which supports all required features and is expected to be well suited for this type of query, this is

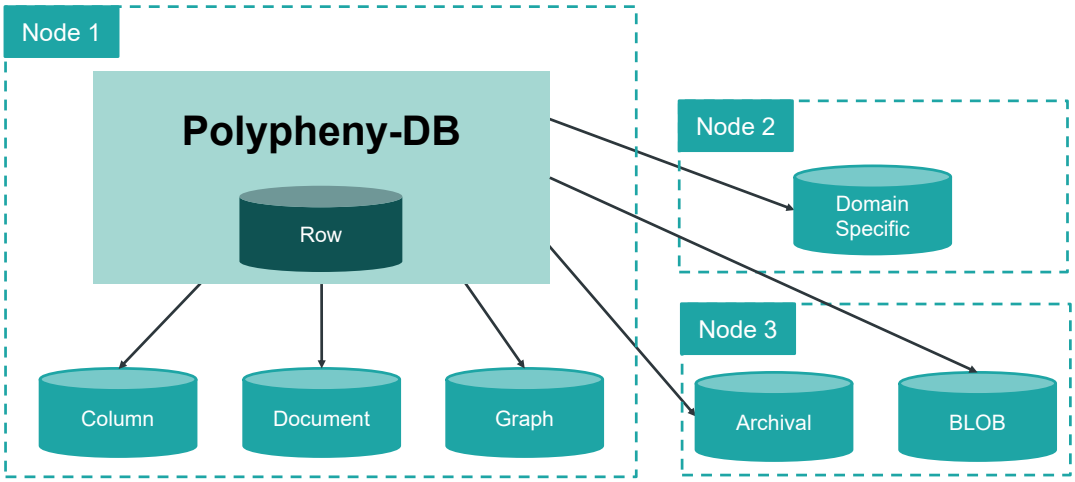

Figure 3. Possible deployment of a Polypheny-DB instance. While there is one data store which is tiddly integrated into Polypheny-DB, the other underlying data stores are either located on the same physical node as Polypheny-DB or on another physical node in the same availability zone.

not possible anymore when storing data just on one store.

Hence, a combination of partitioning and replication of data is also required at the local polystore level of Polypheny-DB. In general, we can distinguish between partitioning the database (distributing whole entities) and partitioning the table (distributing parts of an entity). For the latter, we can furthermore distinguish between vertical partitioning (distributing the attributes of an entity) and horizontal partitioning (splitting the set of items of an entity). Polypheny-DB will adaptively combine both approaches depending on the specified requirements and the observed and predicted workload.

This allows those data items of an entity which are frequently requested to be stored in main memory while the items which are only rarely requested in transactional workloads are stored on a data store optimized for OLAP queries running on cheaper storage. Furthermore, the system can statically replicate relatively small entities which are often used in combination with other data (e. g., the category table in the Gavel scenario).

4) Temporal Data Management: Updates can be done in-place or by storing several versions of the same data item. For the latter, the locations where these new versions are stored are relevant. The decision whether to update in-place or to store several versions strongly depends on the expected queries, storage layout, and the frequency of updates.

In previous work, we have introduced ARCTIC [36], an index structure tailored to archive queries which allows to search for different versions of data items in a partially replicated environment. In Polypheny-DB we plan to integrate temporal data management as integral part of the data placement.

This enables Polypheny-DB to decide on the level of data items whether the current version should be stored on all stores which hold versions of this data item or only on a subset of these stores. The latter results in stores which only hold some outdated versions of the data. These stores can for instance be updated periodically or in times of low workload.

In Polypheny-DB, clients will have the option to provide an "acceptable" freshness for each database transaction [37]. This allows to use "outdated" data stores for instance to execute queries where data freshness is not that important (or where an acceptable freshness level is specified explicitly).

5) Multimedia Data: A special focus of Polypheny-DB is on supporting retrieval tasks within large multimedia collections. Due to the enormous data size of multimedia collections and due to the complexity of their associated metadata, this introduces new and interesting problems. In our previous work, we have introduced ADAM$_{pro}$ [38], a combined database and information retrieval system that is tailored to big multimedia collections and used together with our multimedia retrieval engine vitrivr [39]. With Polypheny-DB we want to go a step further by building a holistic system which jointly supports business typical applications and multimedia retrieval. This allows to handle scenarios like the similarity search for auction photos introduced in Section II.

6) Indexes: While indexes allow to speed-up read queries, they introduce an overhead for data manipulation queries [40]. Furthermore, they consume additional memory and storage space. Therefore, while an index potentially improves the overall performance of the database system, it also increases the costs.

In general, Polypheny-DB distinguishes between two kinds of indexes: *in-store indexes* and *polystore indexes*.

In-Store Indexes: are indexes which are created on one of the underlying data stores. This allows the system to create indexes which are beneficial for the usual workload of a specific data store. The drawback of in-store indexes is

that they are not available on other data stores.

Polystore Indexes: are indexes being created and maintained on the level of Polypheny-DB. They are accessed by the query engine to simplify the query before its execution.

An interesting body of research concerns the strategy which controls the index generation in a polystore environment. To decide weather or not it is beneficial to create a polystore index or an in-store index on the subset of the underlying data stores (if they support indexes) requires a sophisticated control logic. The reason for this is the separation between Polypheny-DB and its underlying data stores. The polystore index can only be used to simplify the queries sent to the underlying data stores. But this is only feasible if the set of results from the index look-up is not too large since otherwise we end up with extremely large queries. Therefore, the combination of a polystore index and an in-store index on some stores might be beneficial.

Index support in Polypheny-DB will be implemented in a modular manner which allows to support different types of indexes at the same time. Hence, special index types (e. g., similarity indexes for the multimedia data) can be added.

7) Query Planning: When Polypheny-DB receives an incoming query, the planner analyzes the query and estimates its execution time. It does so by analyzing the operators used and by comparing it with the execution time of other queries which are similar in structure, entities, functions, and operators [6]. This very rough estimation is used to limit the time for the following more detailed analyzes and query plan generation. The idea is to invest more time for (potentially) long-running queries than for short-running queries.

First, the planner generates a set of viable query plans. The process of creating new query plans stops when the previously determined time is over and there is at least one viable query plan. For creating a query plan, Polypheny-DB will try different methods including splitting the query into sub-queries and executing them on different data stores, replacing query functions not available on a data store with an equivalent statement, and copying data to another store.

In a second step, the query planer assigns costs to every query plan. It therefore considers the current workload on the involved data stores, the availability of indexes, and data freshness. Furthermore, it considers the estimated runtime of the query based on the execution time of previous queries which had the same structure (as described in [6]).

Finally, the planner selects the query plan with the lowest costs and sends it to the execution engine which executes it.

C. Self-Adaptiveness

A key feature of Polypheny-DB will be its ability to adjust itself according to the current and predicted workload. Polypheny-DB achieves this by continuously adjusting its own configuration by, for instance, adding and removing underlying data stores and by creating and dropping indexes.

Furthermore, an automated resource management will make sure that Polypheny-DB is always well-prepared for processing the current workload. Especially if deployed in the Cloud, this feature allows Polypheny-DB to autonomously add and remove resources (physical storage, compute nodes, Cloud DBs and analytic services) at runtime.

To define the space in which the system will be able to adapt itself, Polypheny-DB allows the user to define requirements the system has to fulfill. These requirements can include aspects like the required level of data persistence, support for special query functions, and the required level of data freshness. Furthermore, it will be possible to specify the required level of data consistency which influences which data stores can be used by the system.

With these requirements the system has a lower bound which it is not allowed to break and therefore defines, together with a cost model, the space in which Polypheny-DB can adapt itself to the current and predicted workload. Additionally, it will be possible to provide additional requirements or override existing ones on a per transaction-level (e. g., overriding the required level of data freshness).

Such self-adaptiveness requires a good estimation of the current and future workload to proactively address the necessary changes. Polypheny-DB will rely on previous work on workload prediction in Cumulus [21] and BEOWULF [29].

Other events Polypheny-DB has to react on are, for instance, changes in the network latency or bandwidth which might require a reorganization of the data allocation. This is also the case for schema changes initiated by the user.

D. Discussion

The previous subsections have introduced the two levels of data management in Polypheny-DB (data center-wide replication and partitioning, and inter-center polystores) and the additional capability to automatically adapt both layers to changing workloads. However, these aspects discussed before do not cover all the major challenges to be addressed by a system like Polypheny-DB, especially regarding the interactions between both levels and topics that are relevant at both levels. Therefore, in what follows, we extend our discussion to these additional challenges.

1) Inter-level Dependencies in Polypheny-DB: An intrinsic feature of Polypheny-DB is the consideration of two levels of abstraction for the optimization of data management and data access. Hence, many aspects need to be considered at both levels. In addition to concurrency control, this also includes the replication and partitioning of data. The global level determines the sites at which replicas or partitions need to be stored. Then, within each site, replication and partitioning in the small is under control of each local polystore. Hence, two replica sites (at global Cloud level) can independently decide to have completely different patterns for partitioning or replicating their data across data stores, based on the different, site-specific workloads or on different

SLA-based requirements of the local users. Therefore, a major challenge in Polypheny-DB is to give both levels the necessary autonomy for these decisions while avoiding that the combination of the chosen allocation and replication scheme is adverse (i. e., that the decision at one levels negatively impacts a decision made at the other level). This is particularly important in the presence of Polypheny-DB's self-adaptiveness, since the decision to migrate, replicate, or relocated large volumes of data needs to be coordinated across both layers.

2) Concurrency Control: The concurrency control model (CCM) applied is a requirement for data consistency. CCMs reach from strong session one-copy serializability (SS1SR) [41] over one-copy serializability (1SR) [42], distributed snapshot isolation (DSI) [43], causal consistency (CC) [44] to eventual consistency (EC) [45]. Note that the stronger data consistency is, the more synchronization overhead it requires and the more it reduces the overall latency compared to weaker data consistency levels.

Based on our previous work on C^3 [46], Polypheny-DB will include a meta-CCM allowing the system to dynamically switch between various consistency models (e. g., 1SR, Session Consistency, EC) based on a dedicated cost model.

At the polystore level, the multiple versions of data items that are kept locally will be leveraged by applying multiversion concurrency control [47].

Following the ideas of multi-level transaction management [48], concurrency control in Polypheny-DB is applied at global level and within the local polystores separately, albeit not independently.

3) Data Freshness: If applications do not request the most recent version of a data item, the inconsistencies caused by weaker CCMs can be leveraged by allowing the result having older versions implicitly created by weaker CCM. Further, the temporal data management feature of Polypheny-DB can be used to allow clients to query outdated data. However, this necessitates that applications are able to express their constraints on data freshness [37] or the allowed level of "outdatedness" which then also has to be propagated down to the local stores of Polypheny-DB.

4) Data Migration: The dynamic creation of partitions or replicas requires (large) volumes of data to be moved between sites within the Cloud and between stores. For the actual data migration, two approaches exist: in the offline approach, the system halts for user interaction, migrates the data, and starts again. While this has consequences on the overall performance, it avoids problems regarding data consistency which may occur when data that is subject to migration is updated in parallel. The online approach can either be done in background while the system is operating normally, or it can leverage user queries to migrate the data, i. e., to materialize the query results on another site / store. Polypheny-DB will focus be on the online migration, both at global level and within the local polystores.

IV. Related Work

The last years have seen a vast proliferation of different types of poly- and multistore database systems [31]. In contrast to Polypheny-DB and to our best knowledge, none of the existing systems combines data management techniques at global (Cloud) level with techniques combining different data stores at the local level. In this section we compare Polypheny-DB with these systems and argue why none of these systems existing systems provides a holistic solution.

In *BigDAWG* [49], heterogeneous data stores are organized into "islands" (e. g., relational or array islands). Every island has an associated query language and data model. Inter-island queries are resolved by migrating data between the islands. In contrast to Polypheny-DB which provides a query language independent access to the whole data and takes care of required data migrations, BigDAWG requires this to be specified by the user.

The *CloudMdsQL* multistore system [50] is an SQL-like engine and language which together are able to integrate data from relational, unstructured, and big data data stores. Subqueries can take advantage of the processing frameworks of the connected data stores. Similar to Polypheny-DB, CloudMdsQL's query engine follows the wrapper approach preserving the autonomy of the underlying data stores. However, Polypheny-DB allows its clients to chose from a variety of query interfaces making application development more comfortable, since application developers can continue using their respective data model.

The *AWESOME* [51] polystore solves the problem of the initial assignment of data to the appropriate data store by using a decision table that maps the data model of the data source to a specific data store. Further, [51] introduces the data declaration language *ADIL* together with inference rules mapping ADIL onto data stores. Compared to Polypheny-DB and the other polystores introduced before, AWESOME has its main focus on data investigation and analytics.

Hybrid.poly [52] is a main memory polystore using several data models (e. g., a relational, an array, and a JSON model) that are stored in memory. To query Hybrid.poly, an extension of SQL is used to allow complex analytical queries on the combination of non-relational and relational data. In contrast, Polypheny-DB will offer different query interfaces and Polypheny-DB is also not limited to holding the data exclusively in main memory.

QUEPA, introduced in [53], allows the augmentation and exploration of the data stored in a polystore. It enriches the query results with further information and links to connected data being stored in other data sources which have not been queried in the original query. QUEPA can be easily built on top of Polypheny-DB by implementing a DBMS connector. This would allow the exploration and augmentation of the data stored in Polypheny-DB.

V. Conclusion and Outlook

In this paper, we have presented the Polypheny-DB vision of a globally distributed, adaptive polystore in the Cloud. The main feature of Polypheny-DB is the combination of i.) global data partitioning and replication in the Cloud, based on user requirements and the resource optimization of Cloud providers and ii.) local polystores at individual Cloud data centers that leverage the strengths of different data models, data stores, and storage media. In addition, as most decisions at both levels are based on the actual workload, both global and local level data management is subject to dynamic adaptations. We expect this combination to holistically address a very broad range of current and forthcoming challenging on managing large and heterogeneous collections of data.

While both approaches (global data replication and partitioning, and local polystores) already exist, the originality of Polypheny-DB is the seamless combination of both layers and advanced features such as the support for multimedia retrieval, temporal data management, and different levels of data freshness.

Although the paper's focus is on the presentation of the Polypheny-DB vision, most building blocks, both at global and local level, already exist. In our future work, we aim at extending and combining them, with particular attention to the interactions and dependencies between both levels.

Acknowledgment

The work has been partly funded by the Swiss National Science Foundation (SNSF) in the context of the project Polypheny-DB (contract no. 200021_172763).

References

[1] E. A. Brewer, "Towards Robust Distributed Systems," in *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing - PODC'00*, 2000.

[2] S. Gilbert and N. Lynch, "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services," *ACM SIGACT News*, 2002.

[3] D. Nashat and A. A. Amer, "A Comprehensive Taxonomy of Fragmentation and Allocation Techniques in Distributed Database Design," *ACM Computing Surveys*, 2018.

[4] D. Abadi, "Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story," *Computer*, 2012.

[5] M. Stonebraker and U. Cetintemel, "One Size Fits All: An Idea Whose Time Has Come and Gone," in *21st International Conference on Data Engineering (ICDE'05)*, 2005.

[6] M. Vogt, A. Stiemer, and H. Schuldt, "Icarus: Towards a Multistore Database System," in *Proceedings of the 2017 IEEE International Conference on Big Data (Big Data)*, 2017.

[7] J. Gray, P. Helland, P. O'Neil, and D. Shasha, "The dangers of replication and a solution," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1996.

[8] K. Grolinger, W. Higashino, A. Tiwari, and M. A. Capretz, "Data management in cloud environments: NoSQL and NewSQL data stores," *Journal of Cloud Computing: Advances, Systems and Applications*, 2013.

[9] A. Pavlo and M. Aslett, "What's Really New with NewSQL?" *ACM SIGMOD Record*, 2016.

[10] J. Gray, "The Transaction Concept: Virtues and Limitations," *Proceedings of the 7th International Conference on Very Large Data Bases*, 1981.

[11] T. Haerder and A. Reuter, "Principles of transaction-oriented database recovery," *ACM Computing Surveys*, 1983.

[12] J. N. Gray, "Notes on Data Base Operating Systems," in *Operating Systems, An Advanced Course*, 1978.

[13] D. Skeen and M. Stonebraker, "A Formal Model of Crash Recovery in a Distributed System," *IEEE Transactions on Software Engineering*, 1983.

[14] L. Lamport, "The Part-Time Parliament," *ACM Transactions on Computer Systems*, 1998.

[15] D. Ongaro and J. K. Ousterhout, "In Search of an Understandable Consensus Algorithm," in *2014 USENIX Annual Technical Conference*, 2014.

[16] J. Gray and L. Lamport, "Consensus on transaction commit," *ACM Transactions on Database Systems*, 2006.

[17] D. Laney, "3-D Data Management: Controlling Data Volume, Velocity, and Variety," *META Group Research Note*, 2001.

[18] C. Curino, E. Jones, Y. Zhang, and S. Madden, "Schism: a Workload-Driven Approach to Database Replication and Partitioning," *Proceedings of the VLDB Endowment*, 2010.

[19] A. Jindal and J. Dittrich, "Relax and Let the Database do the Partitioning Online," in *Lecture Notes in Business Information Processing*, 2012.

[20] G. C. Durand, M. Pinnecke, R. Piriyev, M. Mohsen, D. Broneske, G. Saake, M. S. Sekeran, F. Rodriguez, and L. Balami, "GridFormation: Towards Self-Driven Online Data Partitioning using Reinforcement Learning," in *Proceedings of the 1st International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, 2018.

[21] I. Fetai, D. Murezzan, and H. Schuldt, "Workload-driven adaptive data partitioning and distribution The Cumulus approach," in *IEEE International Conference on Big Data*, 2015.

[22] R. Jiménez-Peris, M. Patiño-Martínez, G. Alonso, and B. Kemme, "Are Quorums an Alternative for Data Replication?" *ACM Transactions on Database Systems*, 2003.

[23] J. B. Rothnie, P. A. Bernstein, S. Fox, N. Goodman, M. Hammer, T. A. Landers, C. Reeve, D. W. Shipman, and E. Wong, "Introduction to a system for distributed databases (SDD-1)," *ACM Transactions on Database Systems*, 1980.

[24] D. K. Gifford, "Weighted voting for replicated data," in *Proceedings of the 7th Symposium on Operating Systems Principles*, 1979.

[25] R. H. Thomas, "A Majority consensus approach to concurrency control for multiple copy databases," *ACM Transactions on Database Systems*, 1979.

[26] D. Agrawal and A. El Abbadi, "The Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data," *ACM Transactions on Database Systems*, 1992.

[27] V. Kumar and A. Agarwal, "Generalized Grid Quorum Consensus for Replica Control Protocol," in *2011 International Conference on Computational Intelligence and Communication Networks*, 2011.

[28] I. Fetai, A. Stiemer, and H. Schuldt, "QuAD: A Quorum Protocol for Adaptive Data Management in the Cloud," in *IEEE International Conference on Big Data*, 2017.

[29] A. Stiemer, I. Fetai, and H. Schuldt, "Analyzing the performance of data replication and data partitioning in the cloud: The BEOWULF approach," in *IEEE International Conference on Big Data*, 2016.

[30] I. Fetai, F.-M. Brinkmann, and H. Schuldt, "PolarDBMS: Towards a cost-effective and policy-based data management in the cloud," in *IEEE 30th International Conference on Data Engineering Workshops*, 2014.

[31] R. Tan, R. Chirkova, V. Gadepally, and T. G. Mattson, "Enabling Query Processing Across Heterogeneous Data Models: A Survey," in *IEEE International Conference on Big Data*, 2017.

[32] J. Martin, *Managing the Data Base Environment*. Pearson Education, 1983.

[33] J. Marton, G. Szárnyas, and D. Varró, "Formalising openCypher Graph Queries in Relational Algebra," in *Advances in Databases and Information Systems*, 2017.

[34] H. Jananthan, Z. Zhou, V. Gadepally, D. Hutchison, S. Kim, and J. Kepner, "Polystore Mathematics of Relational Algebra," in *IEEE International Conference on Big Data*, 2017.

[35] V. Giannakouris, N. Papailiou, D. Tsoumakos, and N. Koziris, "MuSQLE: Distributed SQL Query Execution Over Multiple Engine Environments," in *2016 IEEE International Conference on Big Data*, 2016.

[36] F.-M. Brinkmann and H. Schuldt, "Towards Archiving-as-a-Service: A Distributed Index for the Cost-effective Access to Replicated Multi-Version Data," in *Proceedings of the 19th International Database Engineering & Applications Symposium (IDEAS '15)*, 2014.

[37] U. Röhm, K. Böhm, H.-J. Schek, and H. Schuldt, "FAS a Freshness-Sensitive Coordination Middleware for a Cluster of OLAP Components," in *VLDB '02: Proceedings of the 28th International Conference on Very Large Databases*, 2002.

[38] I. Giangreco and H. Schuldt, "ADAMpro : Database Support for Big Multimedia Retrieval," *Datenbank-Spektrum*, 2016.

[39] L. Rossetto, I. Giangreco, C. Tanase, and H. Schuldt, "Vitrivr: A flexible retrieval stack supporting multiple query modes for searching in multimedia collections," in *Proceedings of the 2016 ACM on Multimedia Conference*, 2016.

[40] H. Garcia-Molina, J. D. Ullman, and J. Widom, *Database Systems: Pearson New International Edition*, 2nd ed. Pearson Education Limited, 2013.

[41] V. Zuikevičit and F. Pedone, "Correctness Criteria for Database Replication: Theoretical and Practical Aspects," in *On the Move to Meaningful Internet Systems*, 2008.

[42] P. Bernstein and N. Goodman, "Concurrency Control in Distributed Database Systems," *ACM Computing Surveys*, 1981.

[43] S. Elnikety, W. Zwaenepoel, and F. Pedone, "Database Replication Using Generalized Snapshot Isolation," in *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*, 2005.

[44] M. Raynal, G. Thia-Kime, and M. Ahamad, "From serializable to causal transactions for collaborative applications," in *EUROMICRO 97. Proceedings of the 23rd EUROMICRO Conference: New Frontiers of Information Technology*, 1997.

[45] M. Shapiro and B. Kemme, "Eventual Consistency," in *Encyclopedia of Database Systems*, 2017.

[46] I. Fetai and H. Schuldt, "Cost-Based Data Consistency in a Data-as-a-Service Cloud Environment," in *2012 IEEE Fifth International Conference on Cloud Computing*, 2012.

[47] P. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Longman Publishing Co., Inc., 1987.

[48] C. Beeri, H. Schek, and G. Weikum, "Multi-Level Transaction Management, Theoretical Art or Practical Need?" in *Proceedings of the International Conference on Extending Database Technology (EDBT'88)*, 1988.

[49] J. Duggan, S. Zdonik, A. Elmore, M. Stonebraker, M. Balazinska, B. Howe, J. Kepner, S. Madden, D. Maier, and T. Mattson, "The BigDAWG Polystore System," *ACM SIGMOD Record*, 2015.

[50] C. Bondiombouy, B. Kolev, O. Levchenko, and P. Valduriez, "Multistore Big Data Integration with CloudMdsQL," in *Transactions on Large-Scale Data- and Knowledge-Centered Systems XXVIII*, 2016.

[51] S. Dasgupta, K. Coakley, and A. Gupta, "Analytics-driven data ingestion and derivation in the AWESOME polystore," in *IEEE International Conference on Big Data*, 2016.

[52] M. Podkorytov, D. Soderman, and M. Gubanov, "Hybrid.poly: An Interactive Large-Scale In-memory Analytical Polystore," in *IEEE International Conference on Data Mining Workshops (ICDMW)*, 2017.

[53] A. Maccioni, E. Basili, and R. Torlone, "QUEPA: QUerying and Exploring a Polystore by Augmentation," in *Proceedings of the 2016 International Conference on Management of Data - SIGMOD '16*, 2016.