Check Out the Big Brain on BRAD: Simplifying Cloud Data Processing with Learned Automated Data Meshes

Tim Kraska*
MIT CSAIL
Amazon Web Services
kraska@mit.edu
timkrask@amazon.com

Tianyu Li*
MIT CSAIL
litianyu@mit.edu

Samuel Madden*
MIT CSAIL
madden@csail.mit.edu

Markos Markakis*
MIT CSAIL
markakis@mit.edu

Amadou Ngom*
MIT CSAIL
ngom@mit.edu

Ziniu Wu*
MIT CSAIL
ziniuw@mit.edu

Geoffrey X. Yu*
MIT CSAIL
geoffxy@mit.edu

ABSTRACT

The last decade of database research has led to the prevalence of specialized systems for different workloads. Consequently, organizations often rely on a combination of specialized systems, organized in a *Data Mesh*. Data meshes present significant challenges for system administrators, including picking the right system for each workload, moving data between systems, maintaining consistency, and correctly configuring each system. Many non-expert end users (e.g., data analysts or app developers) either cannot solve their business problems, or suffer from sub-optimal performance or cost due to this complexity. We envision BRAD, a cloud system that automatically integrates and manages data and systems into an instance-optimized data mesh, allowing users to efficiently store and query data under a unified data model (i.e., relational tables) without knowledge of underlying system details. With machine learning, BRAD automatically deduces the strengths and weaknesses of each engine through a combination of offline training and online probing. Then, BRAD uses these insights to route queries to the most suitable (combination of) system(s) for efficient execution. Furthermore, BRAD automates configuration tuning, resource scaling, and data migration across component systems, and makes recommendations for more impactful decisions, such as adding or removing systems. As such, BRAD exemplifies a new class of systems that utilize machine learning and the cloud to make complex data processing more accessible to end users, raising numerous new problems in database systems, machine learning, and the cloud.

PVLDB Reference Format:
Tim Kraska, Tianyu Li, Samuel Madden, Markos Markakis, Amadou Ngom, Ziniu Wu, and Geoffrey X. Yu. Check Out the Big Brain on BRAD: Simplifying Cloud Data Processing with Learned Automated Data Meshes. PVLDB, 16(11): 3293 - 3301, 2023.
doi:10.14778/3611479.3611526

*All authors contributed equally to this paper.

Proceedings of the VLDB Endowment, Vol. 16, No. 11 ISSN 2150-8097.
doi:10.14778/3611479.3611526

1 INTRODUCTION

The last decade has seen an explosion of specialized database engines for both transactional and analytical workloads following the "one size does not fit all" mantra [71]. Today, Amazon Web Services (AWS) alone lists nearly 30 different services under its "Analytics" (e.g., Redshift, EMR, Athena) and "Database" (e.g., Aurora, DynamoDB, DocumentDB) categories. This is because no single system can provide adequate performance for all of an organization's data needs. For example, an S&P 500 corporation we are familiar with uses, among other services, a dozen Amazon Aurora databases for their website and ERP systems, MemoryDB for caching, S3 managed by AWS Lake Formation and queried by AWS EMR for their logs, over ten different Redshift clusters for dashboards and data science, and DocumentDB for content serving. Such *Data Mesh* architectures [20] are now common in organizations of all sizes.

Building and maintaining such a data mesh is challenging. Experts must pick the right combination of engines based on deep understanding of the strengths and weaknesses of each engine, devise custom solutions to move data between engines, track data locations and formats, and actively evolve the mesh over time. This leads to highly complex systems that require large teams of skilled engineers to operate. Meanwhile, data mesh users (e.g., data scientists or app developers) often lack the expert knowledge to quickly identify which exact service(s) to use for their purposes, which leads to poor user experience and sub-optimal use of the data mesh. Furthermore, modern data infrastructure is often deployed on the public cloud [27] with fine-grained auto-scaling capabilities [37]. Cloud data mesh users must additionally optimize for cost-efficiency, besides performance. The ensuing complexity is quickly growing beyond human capabilities. Previous efforts have focused on automating individual systems (e.g., auto-scaling data warehouses) [9, 42, 56, 62] or optimizing for a single metric (e.g., knob tuning for performance) [39, 40, 76, 78, 87], whereas we call for a more holistic approach that navigates the complex trade-offs that arise when when choosing which systems to use and which data to place on them to minimize costs and / or maximize performance.

In this paper, we argue that the way forward is to build highly autonomous, learning-powered *Self-Organizing Data Meshes*, and present our vision for the first such system called BRAD. BRAD uses automation techniques, instead of human experts, to assemble,

Table 1: Runtime of two queries on Aurora and Redshift.

Query	Aurora	Redshift	Performance Gap	Joint
19b	**0.18 s**	3.1 s	17×	—
19e	9.6 s	**2.6 s**	3.7×	**1.8s**

optimize, and evolve data meshes in the cloud. Users largely interact with BRAD through a unified interface under the illusion of a single system with one copy of the data and one (SQL-based) API. Underneath the hood, BRAD uses ML models to extract insights about the strengths and weaknesses of available engines, discover workload patterns, smartly create and evolve the data mesh infrastructure and optimally distribute the workload among the available engines. If needed, users can bypass the one-size-fits-all interface and directly intervene in some underlying systems while leaving others to BRAD. With BRAD, developers can enjoy increased productivity from a strong abstraction and simple interface, which hides away the management of various specialized systems; organizations can enjoy performance improvements and cost savings as our models uncover insights and adapt the data pipeline at a speed and frequency infeasible for human experts; and database internals developers can enjoy greater impact, as BRAD lowers the cost of innovation adoption by automating workload migration.

BRAD's vision presents several novel technical challenges. BRAD needs a query planner that can cleverly divide work between engines, supported by an accurate, learned model for query performance on different engines. Then, BRAD must leverage sophisticated strategies to navigate the complex trade-off space of data mesh design. To make BRAD practical, we must also develop novel learning techniques to adapt to unseen workloads and deployments and solve challenges around data synchronization and consistency. In this paper, we present the architecture of BRAD, outline our plan to address these challenges, and present promising initial results.

2 MOTIVATION AND BACKGROUND

We will first present examples of counter-intuitive optimizations on a simple data mesh: an OLTP engine and an OLAP engine.

2.1 Motivating Scenarios

OLAP systems are not always better at analytics. Conventional wisdom suggests that for the best performance within this simple data mesh one should execute transactional queries on the OLTP system (e.g., Aurora), analytical queries on the OLAP system (e.g., Redshift), and periodically synchronize between the two. This is not true; as a counter-example, we run query 19b from the Join-Order Benchmark [46] (an analytical query) on Aurora and Redshift on the IMDB dataset, along with a modified version denoted 19e. Both queries have the same join template, but 19e omits some highly selective filter predicates (on `title`, `cast_info`, and `name`). As shown in Table 1, we observe that Aurora is 17× faster than Redshift on 19b, but 3.7× slower on 19e. This is because the selective filters in query 19b allow Aurora to leverage its indexes for the join. Redshift, lacking indexes, must resort to table scans for both queries. Had one chosen to process 19b on Redshift, based on the query type, they would see an order-of-magnitude slowdown.

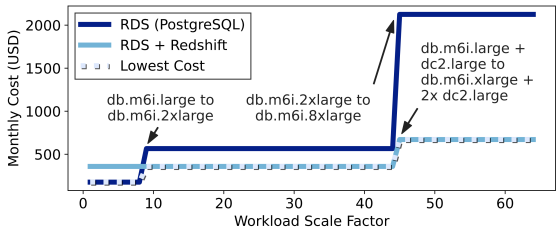


Figure 1: The cost of two setups across workload scales. We label the changes made to maintain latency targets.

The Best Execution Plan may be Federated. The best execution plan for a given query on a data mesh may need to combine the strengths of different engines. To illustrate, we manually split query 19e from Table 1 into two sub-queries: sub-query 1 can be optimally executed using index scans and joins, which only Aurora supports, so we route it to Aurora. We export the results as a CSV file, which we then import into Redshift. Sub-query 2 lacks filter predicates, so it is more efficient to execute on Redshift, a column store. Per Table 1 this joint execution plan is indeed faster than either Aurora or Redshift alone. The reported runtime includes 0.8 seconds to transfer the intermediate results, which could be further optimized.

Knowing When to Scale is Non-Trivial. Operating cloud data infrastructure also invites cost optimization, as modern systems support fine-grained resource scaling. Ideally, one would only pay for the resources they need at any given time, but doing so is not easy. To illustrate this, we ran a simple e-commerce workload (consisting of sales transactions and periodic analytical reporting queries), representing a typical company's data needs as it grows, against two setups: a single instance deployment of Amazon RDS PostgreSQL and a deployment of RDS PostgreSQL *and* Redshift, along with an ETL pipeline that periodically copies the latest writes from RDS into Redshift. The former setup is simpler to maintain and more economical at small workload scales, but the latter setup may perform much better at larger scales. Figure 1 shows our results. The RDS-only deployment starts as the most economical setup, but at a large enough scale (scale factor 8) a combined RDS and Redshift setup becomes cheaper. Importantly, in real cloud deployments, such inflection points tend to be *dynamic*, subject to changing workloads, pricing models and offerings, etc. Human developers are unlikely to be able to always follow the optimal cost line.

2.2 The Case for a New Approach

As shown, automated solutions that manage a data mesh and decide how to execute user queries are needed. We argue that we should cast the challenge of hybrid workload processing as automated system composition. BRAD encompasses three novel directions:

- BRAD is a *backward-compatible, incrementally-deployable* solution on top of existing data meshes. Advanced users can directly access underlying systems where necessary, or use BRAD's programmable policy interface to restrict its interaction with the data mesh. This minimizes impact on legacy workloads and controls the pace of transition to autonomous operation.

Table 2: Differences between BRAD and related work.

System	Incremental Adoption	Specialized Feat. Support	Multiple Data Model Support	Autonomous Operation
AlloyDB (HTAP System) [30]	No	No	No	No
DeltaLake (Lakehouse) [11]	Some	Yes	Some	No
BigDAWG (Polystore) [25]	No	Some	Yes	No
Oracle Autonomous Database [61]	No	No	No	Yes
BRAD	**Yes**	**Yes**	**Some**	**Yes**

- BRAD is a *cloud-native* system optimizing not just for performance, but for a complex cost-performance trade-off curve, with many more degrees of freedom than conventional ML-optimized data systems: it can dynamically change the provisioning of systems, spin up new systems, shift workload and data between systems, and take advantage of serverless [5, 7] offerings.
- BRAD is a *learned* system that leverages machine learning techniques to automatically extract insights about the strengths and weaknesses of available engines, discover workload patterns, smartly create and evolve the data mesh infrastructure and distribute the workload among the available engines optimally.

Below, we compare BRAD to prior work (summarized in Table 2).

HTAP systems. Recent advances in HTAP systems have shown it possible to build engines that match specialized engines across scenarios [1, 26, 35, 41, 45, 70]. Compared to BRAD, HTAP systems may be more efficient for specific workloads, as they may optimize execution holistically, across engine boundaries. However, HTAP systems are *replacements* for existing systems; because of the complexity of migration, we believe modern enterprises are unlikely to adopt HTAP systems for existing and legacy applications. BRAD instead helps organizations manage their *existing* infrastructure, rather than requiring them to move to a new system. In addition, if an HTAP system excels at certain workloads, BRAD can incorporate it into the data mesh and take advantage of its performance.

Data Lakehouses. Data lakehouses process, curate, and store data directly on a cloud data lake [12]. For the user, BRAD looks like a data lakehouse: a unified data storage layer that supports various workloads and automatically transforms and prepares data. However, the emphasis and implementation of the two approaches are quite different. BRAD focuses on automatic management of multiple structured data engines, while lakehouses typically rely on open direct-access data formats and support unstructured and semi-structured data as a first-class concern. If such concerns are relevant, lakehouses can simply be incorporated into a data mesh as a component system, and therefore leverage BRAD. Our techniques may also benefit lakehouses that support multiple structured data frontends (e.g., SparkSQL and Presto on Delta Lake).

Polystores and Federated Databases. Prior work on polystores [2, 3, 24, 65, 79, 80, 88] and federated databases [13, 15, 16, 28, 36, 38, 66, 69, 86] resembles BRAD in serving diverse data processing needs with a collection of specialized systems under a unified query interface. These systems focus on tackling heterogeneous data models and query languages, leaving it up to developers to deploy, maintain, and curate the underlying systems. Instead, BRAD *automates* the management of the underlying systems and *jointly optimizes* query execution with system composition and data placement. In short, BRAD is the next-generation polystore, emphasizing autonomy more than heterogeneous data models and federation.

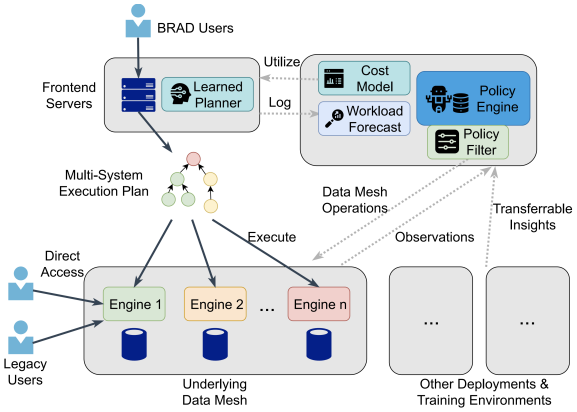


Figure 2: BRAD intelligently serves a unified data API with specialized engines, leveraging the modern cloud to provide an auto-scaling, management-free DBMS

Self-Driving and Instance-Optimized Systems. There has been a wealth of work in *instance-optimization techniques* [22, 23, 42–44, 52–54, 57, 84, 85], which allow systems to automatically specialize themselves to a specific workload's characteristics; and *self-driving systems* [49, 62–64, 78], which automate the running of complex systems that previously relied on regular human intervention. BRAD builds on top of this important work, but must (i) tackle additional complexity in deciding how systems compose with one another, including dynamic data placement and cross-system query optimization and planning and (ii) optimize for a more complex, dynamic objective (e.g., cost-savings and resilience to future business shifts in addition to raw performance) due to its cloud setting.

3 BRAD OVERVIEW

We now introduce our proposed architecture for BRAD. As shown in Figure 2, each instance of BRAD sits atop an ensemble of backend data processing engines; each engine excels at a different workload or provides a different cost-performance trade-off. We assume that each component engine supports the relational model and SQL to some degree. BRAD users define schema, issue queries, and execute transactions against a unified SQL interface. BRAD maintains frontend servers that act as entry points for translating user queries into an execution plan using a *learned query planner*. In the simplest case, the planner routes the query to one engine based on predicted performance, data availability, and dynamic information such as system load. In other cases, BRAD may split a query into subcomponents on different engines and combine the results.

To power the learned planner, BRAD must derive insights about each engine, including supported features, pricing, and performance

on different query types. Such information, along with statistics collected by the engines, drives BRAD's *cost model*. We envision that performance insights are *transferable* across deployments and workloads on the same engine. It is therefore possible to obtain reasonable cost models through experiments in offline training deployments instead of exploring in production. By collecting large volumes of workload information and performance metrics in the cloud setting, BRAD can avoid relying on human-supplied information (e.g., that AWS Aurora is optimized for transactions) and instead *discover* them from real workloads and environments.

Beyond query execution, BRAD maintains and evolves the data mesh to match workload changes (e.g., business growth or demand spikes). BRAD first uses historical data for *workload forecasting*; the forecast is used by an intelligent *policy engine* to trigger necessary actions (e.g. increasing resources for an engine). Perhaps the most important policy decision regards data placement across engines. For example, if a user frequently runs analytics on transactionally hot tables, BRAD may need to replicate the tables in an analytical engine and trigger frequent batch export jobs to keep them in sync. The problems of query planning and mesh optimization constitute a joint optimization problem. For example, BRAD may decide to under-provision Redshift in a mesh, and instead route burst workloads to a serverless engine such as Athena. Alternatively, BRAD may over-provision an OLTP system such as Aurora to handle some analytical workloads (e.g., to take advantage of indexes).

Lastly, BRAD must be practical: organizations already operate data meshes and want to avoid disruptions. BRAD is designed for compatibility and gradual adoption: one may deploy BRAD on the existing data mesh and it can immediately start serving users with the single-interface experience after some initial bootstrapping. Meanwhile, legacy workloads can still interact with the underlying engines directly, bypassing BRAD. To aid gradual adoption, BRAD lets users apply *policy filters*. For example, a user may enforce that some table is always loaded into Redshift, or that Redshift is always provisioned with a minimum amount of resources. Policy filters can also be used to run BRAD in advisory mode, by intercepting migration decisions and asking users for permission. This addresses the corner cases where BRAD is faced with underlying systems with weaker semantics (e.g., DynamoDB) or non-SQL interfaces (e.g., Redis). For example, BRAD cannot unilaterally migrate from a relational DBMS to a lightweight key-value store, as users may anticipate a future feature that needs strong transactional support.

4 RESEARCH DIRECTIONS

4.1 Learned Query Planner

Central to BRAD is a learned query planner that maps queries to execution plans, considering factors like data availability, the strengths of each engine, and each engine's load.

4.1.1 Execution Time Cost Model. Arguably, the core component of a learned planner is a cost model that predicts the query execution time for each of the underlying engines, which the planner can use to route a query to the engine with the lowest predicted execution time. Developing this model poses new research challenges. For example, it is necessary to predict a query's performance on engines and hardware that may not yet have been tested for the user's current workload and dataset. Therefore, we must transcend existing approaches [47, 50, 52, 53, 55, 72], which use previously executed workloads on a specific engine as training data.

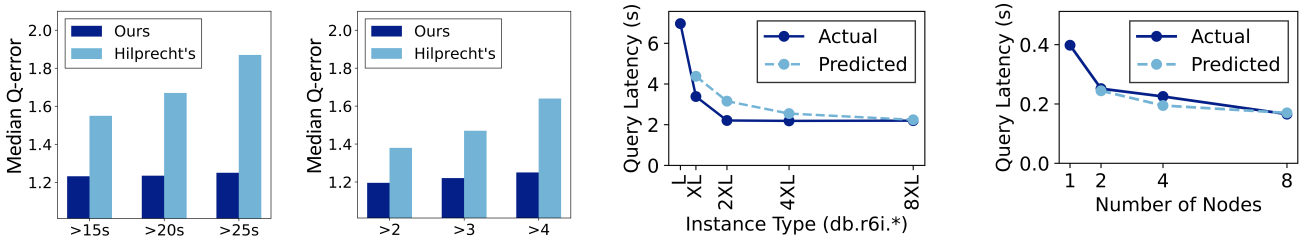
Recently, a dataset-agnostic cost model [33] was proposed to predict the runtime for unseen workloads. However, directly applying this model in BRAD is sub-optimal for three reasons: (i) the model requires the query execution plan as input, which may not be available, as some engines may not support or contain certain tables to execute functionality that produces a query execution plan (i.e., `EXPLAIN`); (ii) the model allows dataset-specific information leakage into the model, influencing the performance on unseen workloads, and (iii) the model is tailored to a single-node PostgreSQL engine with fixed hardware, which may not be generalizable.

To tackle problem (i), we designed a transferrable cost model that takes a SQL query as input and outputs its estimated runtime in PostgreSQL. For problem (ii), we provide our cost model with the true cardinalities during the training phase to prevent the model from learning dataset-specific knowledge (e.g., cardinality). Our cost model only needs base-table and pair-wise join cardinalities, which are relatively easy to obtain either from the underlying engines directly or by using a learned cardinality estimator [58, 83].

These two ideas already provide better generalizability than Hilprecht's model [34]. As a preliminary experiment, we use the same datasets and analytical query workloads as in [34]. We train our cost model on 19 datasets and test it on 2,000 analytical queries on the unseen IMDB dataset. For this experiment, we provide our model with the true cardinalities for training and testing queries. In practice, during testing time, our cost model would not have access to the true cardinality. Therefore, we propose integrating a lightweight cardinality corrector from our recent research [59] that takes DBMS estimates as input and adaptively evolves when observing more queries. Comprehensive experiments [59] have been conducted to show the accuracy and practicality of this cardinality corrector. Our experiments in Figure 3 show the robustness of our cost model against Hilprecht's [33]. Figure 3 shows that our cost model, when trained on queries from 19 datasets with less than 15 s runtime (3(a)) or up to two join predicates (3(b)), can generalize to unseen IMDB queries with longer runtimes or more join predicates, respectively (Q-error is defined as max{predicted/true, true/predicted} - better estimates have a Q-error closer to 1).

For problem (iii), we use lightweight parameterized functions to predict query performance on different hardware. Figures 3(c) and 3(d) show our ability to accurately predict an example query's runtime on instances with unseen types (3(c)) or node counts (3(d)), given its runtime on current hardware. We are currently integrating all these components to derive an accurate and robust cost model.

4.1.2 Cross-Engine Translation. Different component engines of BRAD may support different SQL dialects, data types, or specialized operators and therefore be mutually incompatible. To fully utilize underlying engines, BRAD must be able to potentially rewrite queries for different engines. Writing manual rules for translation between systems is challenging and error-prone. Recent work in automatic code understanding provides an alternative solution [18, 75]. Specifically, large language models (LLMs) trained on the documentation of each engine can translate special features and



(a) Query Runtime (b) Number of Joins (c) Instance Type (d) Number of Nodes

Figure 3: Cost model performance overview: (a), (b) Our cost model generalizes to unseen queries with longer runtimes or more join predicates. (c), (d) We accurately predict query runtime under different instance types and/or number of nodes.

possibly generate UDFs in the process. We illustrate the feasibility of this approach by using GPT-4 to convert the Athena query **`SELECT CONTAINS ('10.0.0.0/8', IPADDRESS address)`** to Postgres- and Redshift-compatible SQL, where it needs to use a Python UDF for the missing IP address processing feature. For Postgres SQL, the LLM correctly used the `inet` type and the » operator. For Redshift, the LLM correctly generated a Python UDF using `ipaddress` (part of the standard library) to check if an IP address is within a subnet. These early results show that it is possible to represent query semantics as a dialect-independent embedding. However, relying solely on LLMs is not yet practical due to the performance overhead and hallucination risk. Developing robust, interpretable translation schemes with LLM-like techniques instead of static rules is a promising research direction to address this challenge.

4.1.3 Multi-System Query Plans. As shown earlier, BRAD can sometimes achieve better performance by combining results across engines. Similar ideas have been explored for federated query execution [21, 29, 31, 32, 38, 48, 67, 69, 74], but these systems (i) largely focus on data integration and easy querying rather than performance, and (ii) often rely on manual rules or heuristics. We envision tackling multi-system query planning as an extension of the single-node query optimization problem [68]. Each operator will have variants backed by different engines, along with data movement and results merging operators. This framing will also allow BRAD to incorporate standalone components, such as a serverless or hardware-accelerated hash join operator, opening up exciting new opportunities. The optimizer then must efficiently explore the plan space, using system statistics and the cost model to pick the best plan. Recent developments have yielded learned query optimizers that match or outperform traditional methods; we plan to leverage such optimizers in this new environment. Lastly, a resilient execution layer must execute the plan and hide away distributed complexity and possible anomalies due to transient failures [8, 17].

4.2 Joint Query-Mesh Optimization

Beyond executing queries, BRAD optimizes the data mesh structure. Unlike self-driving databases [62–64], BRAD faces dynamic trade-offs among query performance, cost, service level agreements, anticipated growth, etc. The mesh architecture impacts query planning and vice-versa, yielding a joint optimization problem. BRAD needs several learning-based methods to tackle this.

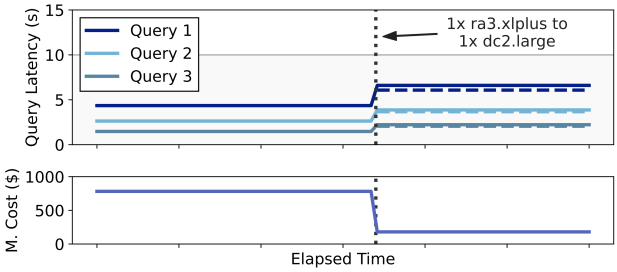


Figure 4: BRAD changes Redshift instance types to reduce cost (bottom) as it correctly predicts that query latency will remain below a user-specified ceiling (top, shaded).

4.2.1 Configuration and Placement Searching. BRAD optimizes the mesh in two major ways: (i) individual engine configuration, and (ii) data placement. At a high-level, BRAD picks a deployment type (e.g., off vs. serverless vs. provisioned); provisioning (e.g., VM type); and various knobs (e.g., number of nodes, cache size) for each system and then decides how to place, replicate, and partition data across engines to best serve a workload. The resulting search space is exponential in the number of tables and configurations, which is prohibitively large for practical deployment.

We envision that BRAD will first adopt a heuristics-based approach, using simple observations (e.g., tables referred to in a single query should be placed together, transactions should be routed to Aurora) and optimization strategies (e.g., [78]) to narrow down the search space. Then, for each candidate mesh configuration and data placement, BRAD can take advantage of its cost model to predict the performance of a forecasted workload [51, 73], which is then combined with the dollar operating cost of the configuration and other metrics to determine how desirable the candidate is.

The holy grail of mesh optimization, however, is to explicitly *learn* to optimally operate a data mesh. BRAD can either choose to learn to score each mesh configuration, or directly learn a policy to evolve it [81, 82]. The challenge is that users of BRAD are unlikely to tolerate the amount of trial and error required for a learning-based method to arrive at a stable and performant configuration. BRAD must therefore devise new techniques to quickly adapt learned models to new, unseen operating conditions. One possible avenue

of approach is to leverage BRAD's cloud-native setting and learn from both the large corpus of passive observations from client deployments and carefully curated shadow deployments that are able to experiment on what-if scenarios (see Section 4.2.2).

Figure 4 shows an example of BRAD optimizing a data mesh; we plot query latency (top) and monthly Redshift cost (bottom) over time. In this example, a user deploys BRAD on a mesh with Redshift running on one `ra3.xlplus` node and tells BRAD that their queries should finish within 10 seconds (shaded region in the figure). When BRAD's mesh optimizer runs, it predicts each query's latency across Redshift provisionings using a learned regression model; the model uses the query's measured latency on the current provisioning and the ratios between the hardware resources (vCPUs and amount of memory) across the two compared provisionings. BRAD correctly predicts (the dashed lines on the graph) that all three queries will run under 10 seconds on one `dc2.large` node—Redshift's most economical instance type. BRAD applies this change and reduces the mesh's monthly Redshift cost by 4× (bottom graph).

4.2.2 System Exploration and Transfer Learning. BRAD must rely on automated, learning-based methods to explore each engine's strengths and weaknesses due to the sheer number of engines BRAD must support. However, learning requires exploring unseen configurations and execution plans, which may hurt performance in an online environment. We aim to leverage BRAD's cloud-native deployment to mitigate this. Cloud providers have access to traces of many client deployments and therefore large amounts of training data. More importantly, they can transparently capture workload traces and spin up "what-if" shadow deployments or experiments instead of exploring on live deployments. This approach has security and privacy implications, but we see these concerns as orthogonal to our system. The critical challenge is whether our model can efficiently transfer insights to unknown databases and deployments, which we have briefly addressed in Section 4.1.1. We envision that in its complete form, BRAD is able to automatically incorporate a new engine into the mesh by first obtaining a rough performance model of it through offline deployments running standard benchmarks (e.g., TPC-C and TPC-H), and then fine-tuning using shadow deployments and real client performance data.

4.3 Data Synchronization and Consistency

A key challenge in a data mesh is to correctly synchronize data across component systems and maintain consistency where it matters, without incurring overhead elsewhere. BRAD's automated placement and migration decisions must address this challenge.

4.3.1 Session-Based Freshness Guarantees. Consistency is a natural concern in BRAD, as it encompasses multiple engines that cannot always be synchronized performantly. Since BRAD is externally a unified system, stale reads and distributed anomalies would violate its abstraction. To avoid them, we propose *session-based freshness guarantees* (similar to Daudjee et al. [19]), where clients issue queries within explicitly defined sessions. Within a session, a query runs against a consistent snapshot of the database and future queries will run against the same, or a later, snapshot. For example, a session S issuing a large data lake query Q_1 may use a snapshot on cloud storage, but if S then issues a transactional update Q_2, it is promoted to the latest snapshot. A future analytical query Q_3 will wait for the analytic engine to receive Q_2's changes. Users can still avoid interleaving analytics/transactions within a session to minimize latency. These guarantees can be achieved through epoch-based logical snapshots [77] and *tuple multi-versioning* [14]. The challenge is to do so without modifying the underlying engines or introducing excessive runtime overhead.

4.3.2 Auto-ETL. In addition to providing consistency guarantees across table replicas, BRAD needs to support more complex data dependency relationships between tables—typically handled by extract, transform, and load (ETL) jobs today. For example, ETL jobs may be used to transform the tables in a transactional DBMS before loading them into the data warehouse (e.g., to de-normalize the tables, re-arrange the tables in a star schema [60], or to compute aggregate statistics). Currently, users often rely on handcrafted transformation logic and ETL frameworks such as AWS Glue [10] or EMR [6]. This setup is both tedious for users and restrictive for BRAD, as users typically hard-code the source and destination systems of such transformations in black box logic—preventing BRAD from freely placing tables. Instead, we envision that BRAD will support a higher-level declarative API for specifying table dependencies (e.g., table B is obtained by running the given SQL statements on table A), which allows BRAD to (i) change the locations of the inputs and outputs to a transformation (e.g., to migrate a table off of an engine), and (ii) select the system(s) on which to execute a transform (e.g., using spare capacity on Redshift or new features such as zero-ETL [4] instead of AWS Glue or EMR).

5 CONCLUSION

BRAD shows a new way to assemble and operate data meshes in the cloud, relying on recent advances in automation techniques instead of human experts. For the vast majority of end users, BRAD significantly simplifies the operation of state-of-the-art data meshes and allows easier derivation of timely insights from vast amounts of data. For database researchers, BRAD lowers the barrier of adoption by providing room for automated and user-transparent migration to new engines where appropriate. This paper outlines our plan to build BRAD and presents preliminary results to show the promise of our approach. If successful, we expect BRAD to unlock the true potential of last decade's research into specialized data systems and have a significant impact on the efficiency of modern enterprises.

ACKNOWLEDGMENTS

This research was supported by Amazon, Google, and Intel as part of the MIT Data Systems and AI Lab (DSAIL) at MIT and NSF IIS 1900933. Geoffrey X. Yu was partially supported by an NSERC PGS D. This research was also sponsored by the United States Air Force Research Laboratory and the Department of the Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Department of the Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

REFERENCES

[1] Michael Abebe, Horatiu Lazu, and Khuzaima Daudjee. 2022. Proteus: Autonomous Adaptive Storage for Mixed Workloads. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22)* (Philadelphia, PA, USA). Association for Computing Machinery, New York, NY, USA, 700–714. https://doi.org/10.1145/3514221.3517834

[2] Divy Agrawal, Sanjay Chawla, Bertty Contreras-Rojas, Ahmed Elmagarmid, Yasser Idris, Zoi Kaoudi, Sebastian Kruse, Ji Lucas, Essam Mansour, Mourad Ouzzani, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, Saravanan Thirumuruganathan, and Anis Troudi. 2018. RHEEM: Enabling Cross-Platform Data Processing: May the Big Data Be with You! *Proceedings of the VLDB Endowment* 11, 11 (July 2018), 1414–1427. https://doi.org/10.14778/3236187.3236195

[3] Rana Alotaibi, Damian Bursztyn, Alin Deutsch, Ioana Manolescu, and Stamatis Zampetakis. 2019. Towards Scalable Hybrid Stores: Constraint-Based Rewriting to the Rescue. In *Proceedings of the 2019 International Conference on Management of Data (SIGMOD '19)*. 1660–1677.

[4] Amazon Web Services. 2022. *AWS announces Amazon Aurora zero-ETL integration with Amazon Redshift* . https://aws.amazon.com/about-aws/whats-new/2022/11/amazon-aurora-zero-etl-integration-redshift/.

[5] Amazon Web Services. 2023. *Amazon Athena.* https://aws.amazon.com/athena/.

[6] Amazon Web Services. 2023. *Amazon EMR.* https://aws.amazon.com/emr/.

[7] Amazon Web Services. 2023. *Amazon Redshift Serverless.* https://aws.amazon.com/redshift/redshift-serverless/.

[8] Amazon Web Services. 2023. *AWS Step Functions.* https://aws.amazon.com/step-functions/.

[9] Amazon Web Services. 2023. *Redshift Concurrency Scaling.* https://docs.aws.amazon.com/redshift/latest/dg/concurrency-scaling.html.

[10] Amazon Web Services. 2023. *What is AWS Glue?* https://docs.aws.amazon.com/glue/latest/dg/what-is-glue.html.

[11] Michael Armbrust, Tathagata Das, Liwen Sun, Burak Yavuz, Shixiong Zhu, Mukul Murthy, Joseph Torres, Herman van Hovell, Adrian Ionescu, Alicja Łuszczak, Michał undefinedwitakowski, Michał Szafrański, Xiao Li, Takuya Ueshin, Mostafa Mokhtar, Peter Boncz, Ali Ghodsi, Sameer Paranjpye, Pieter Senster, Reynold Xin, and Matei Zaharia. 2020. Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3411–3424. https://doi.org/10.14778/3415478.3415560

[12] Michael Armbrust, Ali Ghodsi, Reynold Xin, and Matei Zaharia. 2021. Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics. In *Proceedings of the 11th Annual Conference on Innovative Data Systems Research (CIDR '21)*.

[13] Graham Bent, Patrick Dantressangle, David Vyvyan, Abbe Mowshowitz, and Valia Mitsou. 2008. A Dynamic Distributed Federated Database. In *Proc. 2nd Ann. Conf. International Technology Alliance (ACITA '08')*.

[14] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. 1987. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley.

[15] Yuri Breitbart, Hector Garcia-Molina, and Abraham Silberschatz. 1992. Overview of Multidatabase Transaction Management. *VLDB Journal* 1 (10 1992), 181–239. https://doi.org/10.1145/1925805.1925811

[16] Yuri Breitbart and Avi Silberschatz. 1988. Multidatabase Update Issues. In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '88)*. Association for Computing Machinery, New York, NY, USA, 135–142. https://doi.org/10.1145/50202.50217

[17] Sebastian Burckhardt, Chris Gillum, David Justo, Konstantinos Kallas, Connor McMahon, and Christopher S Meiklejohn. 2021. Durable Functions: Semantics for Stateful Serverless. *Proc. ACM Program. Lang.* 5, OOPSLA (2021), 1–27.

[18] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374 [cs.LG]

[19] Khuzaima Daudjee and Kenneth Salem. 2006. Lazy Database Replication with Snapshot Isolation. *Proceedings of the VLDB Endowment (VLDB '06).*

[20] Z. Dehghani. 2022. *Data Mesh.* O'Reilly Media. https://books.google.com/books?id=jmZjEAAAQBAJ

[21] Amol Deshpande and Joseph M Hellerstein. 2002. Decoupled Query Optimization for Federated Database Systems. In *Proceedings 18th International Conference on Data Engineering (ICDE '02)*. IEEE, 716–727.

[22] Jialin Ding, Umar Farooq Minhas, Badrish Chandramouli, Chi Wang, Yinan Li, Ying Li, Donald Kossmann, Johannes Gehrke, and Tim Kraska. 2021. Instance-Optimized Data Layouts for Cloud Analytics Workloads. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 418–431. https://doi.org/10.1145/3448016.3457270

[23] Jialin Ding, Vikram Nathan, Mohammad Alizadeh, and Tim Kraska. 2020. Tsunami: A Learned Multi-Dimensional Index for Correlated Data and Skewed Workloads. *Proceedings of the VLDB Endowment* 14, 2 (November 2020), 74–86. https://doi.org/10.14778/3425879.3425880

[24] Jennie Duggan, Aaron J. Elmore, Michael Stonebraker, Magda Balazinska, Bill Howe, Jeremy Kepner, Sam Madden, David Maier, Tim Mattson, and Stan Zdonik. 2015. The BigDAWG Polystore System. *SIGMOD Rec.* 44, 2 (August 2015), 11–16. https://doi.org/10.1145/2814710.2814713

[25] Aaron J. Elmore, Jennie Duggan, Mike Stonebraker, Magdalena Balazinska, Ugur Çetintemel, Vijay Gadepally, Jeffrey Heer, Bill Howe, Jeremy Kepner, Tim Kraska, Samuel Madden, David Maier, Timothy G. Mattson, Stavros Papadopoulos, Jeff Parkhurst, Nesime Tatbul, Manasi Vartak, and Stan Zdonik. 2015. A Demonstration of the BigDAWG Polystore System. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1908–1911. http://www.vldb.org/pvldb/vol8/p1908-Elmore.pdf

[26] Franz Färber, Norman May, Wolfgang Lehner, Philipp Große, Ingo Müller, Hannes Rauhe, and Jonathan Dees. 2012. The SAP HANA Database – An Architecture Overview. *IEEE Data Eng. Bull.* 35 (03 2012), 28–33.

[27] Gartner. 2022. *DBMS Market Transformation 2021: The Big Picture.* https://blogs.gartner.com/merv-adrian/2022/04/16/dbms-market-transformation-2021-the-big-picture/.

[28] Dimitrios Georgakopoulos, Marek Rusinkiewicz, and Amit P. Sheth. 1991. On Serializability of Multidatabase Transactions Through Forced Local Conflicts. In *Proceedings of the Seventh International Conference on Data Engineering (ICDE '91)*. IEEE Computer Society, USA, 314–323.

[29] Victor Giannakouris and Immanuel Trummer. 2022. Building Learned Federated Query Optimizers. In *CEUR workshop proceedings*, Vol. 3186.

[30] Google, Inc. 2023. *AlloyDB.* https://cloud.google.com/alloydb.

[31] Laura Haas, Donald Kossmann, Edward Wimmers, and Jun Yang. 1997. Optimizing Queries Across Diverse Data Sources. In *Proceedings of the VLDB Endowment (VLDB '97)*.

[32] Joachim Hammer, Hector Garcia-Molina, Kelly Ireland, Yannis Papakonstantinou, Jeffrey Ullman, and Jennifer Widom. 1995. Information Translation, Mediation, and Mosaic-Based Browsing in the TSIMMIS System. In *Proceedings of the International Conference on Management of Data (SIGMOD '95)*.

[33] Benjamin Hilprecht and Carsten Binnig. 2022. Zero-Shot Cost Models for Out-of-the-box Learned Cost Prediction. *arXiv preprint arXiv:2201.00561* (2022).

[34] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2019. Deepdb: Learn from data, not from queries! *arXiv preprint arXiv:1909.00607* (2019).

[35] Dongxu Huang, Qi Liu, Qiu Cui, Zhuhe Fang, Xiaoyu Ma, Fei Xu, Li Shen, Liu Tang, Yuxing Zhou, Menglong Huang, Wan Wei, Cong Liu, Jian Zhang, Jianjun Li, Xuelian Wu, Lingyu Song, Ruoxi Sun, Shuaipeng Yu, Lei Zhao, Nicholas Cameron, Liquan Pei, and Xin Tang. 2020. TiDB: A Raft-Based HTAP Database. *Proceedings of the VLDB Endowment* 13, 12 (August 2020), 3072–3084. https://doi.org/10.14778/3415478.3415535

[36] S.-Y. Hwang, E.-P. Lim, H.-R. Yang, S. Musukula, K. Mediratta, M. Ganesh, D. Clements, J. Stenoien, and J. Srivastava. 1994. The MYRIAD Federated Database Prototype. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data* (Minneapolis, Minnesota, USA) *(SIGMOD '94)*. Association for Computing Machinery, New York, NY, USA, 518. https://doi.org/10.1145/191839.191986

[37] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Jayant Yadwadkar, Joseph Gonzalez, Raluca A. Popa, Ion Stoica, and David A. Patterson. 2019. Cloud Programming Simplified: A Berkeley View on Serverless Computing. *ArXiv* abs/1902.03383 (2019).

[38] Vanja Josifovski, Peter Schwarz, Laura Haas, and Eileen Lin. 2002. Garlic: A New Flavor of Federated Query Processing for DB2. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data (SIGMOD '02)*. 524–532.

[39] Konstantinos Kanellis, Ramnatthan Alagappan, and Shivaram Venkataraman. 2020. Too Many Knobs to Tune? Towards Faster Database Tuning by Pre-selecting Important Knobs. In *12th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '20)*.

[40] Konstantinos Kanellis, Cong Ding, Brian Kroth, Andreas Müller, Carlo Curino, and Shivaram Venkataraman. 2022. LlamaTune: Sample-Efficient DBMS Configuration Tuning. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2953–2965.

[41] Alfons Kemper and Thomas Neumann. 2011. HyPer: A Hybrid OLTP & OLAP Main Memory Database System Based on Virtual Memory Snapshots. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering (ICDE '11)*. IEEE Computer Society, USA, 195–206. https://doi.org/10.1109/ICDE.2011.5767867

[42] Tim Kraska, Mohammad Alizadeh, Alex Beutel, Ed H. Chi, Ani Kristo, Guillaume Leclerc, Samuel Madden, Hongzi Mao, and Vikram Nathan. 2019. SageDB: A Learned Database System. In *9th Biennial Conference on Innovative Data Systems Research, (CIDR '19), Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*. www.cidrdb.org. http://cidrdb.org/cidr2019/papers/p117-kraska-cidr19.pdf

[43] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2017. The Case for Learned Index Structures. *CoRR* abs/1712.01208 (2017). arXiv:1712.01208 http://arxiv.org/abs/1712.01208

[44] Sanjay Krishnan, Zongheng Yang, Ken Goldberg, Joseph Hellerstein, and Ion Stoica. 2018. Learning to Optimize Join Queries with Deep Reinforcement Learning. *arXiv preprint arXiv:1808.03196* (2018).

[45] Tirthankar Lahiri, Shasank Chavan, Maria Colgan, Dinesh Das, Amit Ganesh, Mike Gleeson, Sanket Hase, Allison Holloway, Jesse Kamp, Teck-Hua Lee, Juan Loaiza, Neil Macnaughton, Vineet Marwah, Niloy Mukherjee, Atrayee Mullick, Sujatha Muthulingam, Vivekanandhan Raja, Marty Roth, Ekrem Soylemez, and Mohamed Zait. 2015. Oracle Database In-Memory: A Dual Format In-Memory Database. In *2015 IEEE 31st International Conference on Data Engineering (ICDE '15)*. 1253–1258. https://doi.org/10.1109/ICDE.2015.7113373

[46] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good are Query Optimizers, Really? *Proceedings of the VLDB Endowment* 9, 3 (2015), 204–215.

[47] Jiexing Li, Arnd Christian König, Vivek Narasayya, and Surajit Chaudhuri. 2012. Robust Estimation of Resource Consumption for SQL Queries Using Statistical Techniques. *Proceedings of the VLDB Endowment* 5, 11 (2012).

[48] Ee-Peng Lim and Jaideep Srivastava. 1993. Query Optimization and Processing in Federated Database Systems. In *Proceedings of the Second International Conference on Information and Knowledge Management (CIKM '93)*. 720–722.

[49] Wan Shen Lim, Matthew Butrovich, William Zhang, Andrew Crotty, Lin Ma, Peijing Xu, Johannes Gehrke, and Andrew Pavlo. 2023. Database Gyms. In *Conference on Innovative Data Systems Research (CIDR '23)*.

[50] Lin Ma, Bailu Ding, Sudipto Das, and Adith Swaminathan. 2020. Active Learning for ML Enhanced Database Systems. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20)*. 175–191.

[51] Lin Ma, Dana Van Aken, Ahmed Hefny, Gustavo Mezerhane, Andrew Pavlo, and Geoffrey J Gordon. 2018. Query-Based Workload Forecasting for Self-Driving Database Management Systems. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18)*. 631–645.

[52] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2022. Bao: Making Learned Query Optimization Practical. In *Proceedings of the International Conference on Management of Data (SIGMOD '22)*.

[53] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2019. Neo: A Learned Query Optimizer. *Proceedings of the VLDB Endowment* 12, 11 (2019).

[54] Ryan Marcus and Olga Papaemmanouil. 2018. Deep Reinforcement Learning for Join Order Enumeration. In *Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management (aiDM '18)*.

[55] Ryan Marcus and Olga Papaemmanouil. 2019. Plan-Structured Deep Neural Network Models for Query Performance Prediction. *Proceedings of the VLDB Endowment* 12, 11 (2019).

[56] Microsoft Corporation. 2023. *Serverless Compute Tier for Azure SQL Database*. https://learn.microsoft.com/en-us/azure/azure-sql/database/serverless-tier-overview?view=azuresql&tabs=general-purpose.

[57] Vikram Nathan, Jialin Ding, Mohammad Alizadeh, and Tim Kraska. 2020. Learning Multi-Dimensional Indexes. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 985–1000. https://doi.org/10.1145/3318464.3380579

[58] Parimarjan Negi, Ryan Marcus, Andreas Kipf, Hongzi Mao, Nesime Tatbul, Tim Kraska, and Mohammad Alizadeh. 2021. Flow-Loss: Learning Cardinality Estimates That Matter. *Proceedings of the VLDB Endowment* 14, 11 (2021).

[59] Parimarjan Negi, Ziniu Wu, Andreas Kipf, Nesime Tatbul, Ryan Marcus, Sam Madden, Tim Kraska, and Mohammad Alizadeh. 2023. Robust Query Driven Cardinality Estimation under Changing Workloads. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1520–1533.

[60] Patrick O'Neil, Betty O'Neil, and Xuedong Chen. 2006. *Star Schema Benchmark*. Technical Report. University of Massachusetts Boston. https://www.cs.umb.edu/~poneil/StarSchemaB.PDF.

[61] Oracle. 2023. *Oracle Autonomous Database*. https://www.oracle.com/autonomous-database/.

[62] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd Mowry, Matthew Perron, Ian Quah, Siddharth Santurkar, Anthony Tomasic, Skye Toor, Dana Van Aken, Ziqi Wang, Yingjun Wu, Ran Xian, and Tieying Zhang. 2017. Self-Driving Database Management Systems. In *Conference on Innovative Data Systems Research (CIDR '17)*. https://db.cs.cmu.edu/papers/2017/p42-pavlo-cidr17.pdf

[63] Andrew Pavlo, Matthew Butrovich, Ananya Joshi, Lin Ma, Prashanth Menon, Dana Van Aken, Lisa Lee, and Ruslan Salakhutdinov. 2019. External vs. Internal: An Essay on Machine Learning Agents for Autonomous Database Management Systems. *IEEE Data Engineering Bulletin* (June 2019), 32–46. https://db.cs.cmu.edu/papers/2019/pavlo-icde-bulletin2019.pdf

[64] Andrew Pavlo, Matthew Butrovich, Lin Ma, Wan Shen Lim, Prashanth Menon, Dana Van Aken, and William Zhang. 2021. Make Your Database System Dream of Electric Sheep: Towards Self-Driving Operation. *Proceedings of the VLDB Endowment* 14, 12 (2021), 3211–3221. https://db.cs.cmu.edu/papers/2021/p3211-pavlo.pdf

[65] Maksim Podkorytov and Michael Gubanov. 2019. Hybrid.Poly: A Consolidated Interactive Analytical Polystore System. In *2019 IEEE 35th International Conference on Data Engineering (ICDE '19)*. 1996–1999. https://doi.org/10.1109/ICDE.2019.00223

[66] Calton Pu. 1988. Superdatabases for Composition of Heterogeneous Databases. In *Proceedings of the Fourth International Conference on Data Engineering*. IEEE Computer Society, USA, 548–555.

[67] Mary Tork Roth, Laura M Haas, and Fatma Ozcan. 1999. *Cost Models Do Matter: Providing Cost Information for Diverse Data Sources in a Federated System*. IBM Thomas J. Watson Research Division.

[68] P. Griffiths Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. 1979. Access Path Selection in a Relational Database Management System. In *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data (SIGMOD '79)* (Boston, Massachusetts) *(SIGMOD '79)*. Association for Computing Machinery, New York, NY, USA, 23–34. https://doi.org/10.1145/582095.582099

[69] Amit P Sheth and James A Larson. 1990. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys (CSUR)* 22, 3 (1990), 183–236.

[70] Vishal Sikka, Franz Färber, Wolfgang Lehner, Sang Kyun Cha, Thomas Peh, and Christof Bornhövd. 2012. Efficient Transaction Processing in SAP HANA Database: The End of a Column Store Myth. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (Scottsdale, Arizona, USA) *(SIGMOD '12)*. Association for Computing Machinery, New York, NY, USA, 731–742. https://doi.org/10.1145/2213836.2213946

[71] Michael Stonebraker and Ugur Cetintemel. 2005. "One Size Fits All": An Idea Whose Time Has Come and Gone. In *Proceedings of the 21st International Conference on Data Engineering (ICDE '05)*. IEEE Computer Society, USA, 2–11. https://doi.org/10.1109/ICDE.2005.1

[72] Ji Sun and Guoliang Li. 2019. An End-to-End Learning-based Cost Estimator. *Proceedings of the VLDB Endowment* 13, 3 (2019).

[73] Rebecca Taft, Nosayba El-Sayed, Marco Serafini, Yu Lu, Ashraf Aboulnaga, Michael Stonebraker, Ricardo Mayerhofer, and Francisco Andrade. 2018. P-Store: An Elastic Database System with Predictive Provisioning. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18)* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 205–219. https://doi.org/10.1145/3183713.3190650

[74] Anthony Tomasic, Remy Amouroux, Philippe Bonnet, Olga Kapitskaia, Hubert Naacke, and Louiqa Raschid. 1997. The Distributed Information Search Component (Disco) and the World Wide Web. *ACM SIGMOD Record* 26, 2 (1997), 546–548.

[75] Immanuel Trummer. 2022. CodexDB: Generating Code for Processing SQL Queries using GPT-3 Codex. arXiv:2204.08941 [cs.DB]

[76] Immanuel Trummer. 2022. DB-BERT: A Database Tuning Tool That "Reads the Manual". In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 190–203. https://doi.org/10.1145/3514221.3517843

[77] Stephen Tu, Wenting Zheng, Eddie Kohler, Barbara Liskov, and Samuel Madden. 2013. Speedy Transactions in Multicore In-Memory Databases. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13)*. 18–32.

[78] Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. 2017. Automatic Database Management System Tuning Through Large-Scale Machine Learning. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 1009–1024. https://doi.org/10.1145/3035918.3064029

[79] Marco Vogt, Alexander Stiemer, and Heiko Schuldt. 2018. Polypheny-DB: Towards a Distributed and Self-Adaptive Polystore. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 3364–3373.

[80] Jingjing Wang, Tobin Baker, Magdalena Balazinska, Daniel Halperin, Brandon Haynes, Bill Howe, Dylan Hutchison, Shrainik Jain, Ryan Maas, Parmita Mehta, Dominik Moritz, Brandon Myers, Jennifer Ortiz, Dan Suciu, Andrew Whitaker, and Shengliang Xu. 2017. The Myria Big Data Management and Analytics System and Cloud Services. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR '17)*.

[81] Christopher J. C. H. Watkins and Peter Dayan. 1992. Q-learning. *Machine Learning* 8, 3 (1992), 279–292. https://doi.org/10.1007/BF00992698

[82] Ronald J. Williams. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Mach. Learn.* 8, 3–4 (May 1992), 229–256. https://doi.org/10.1007/BF00992696

[83] Ziniu Wu, Parimarjan Negi, Mohammad Alizadeh, Tim Kraska, and Samuel Madden. 2023. FactorJoin: A New Cardinality Estimation Framework for Join Queries. *Proc. ACM Manag. Data* 1, 1, Article 41 (May 2023), 27 pages. https://doi.org/10.1145/3588721

[84] Geoffrey X. Yu, Markos Markakis, Andreas Kipf, Per-Åke Larson, Umar Farooq Minhas, and Tim Kraska. 2022. TreeLine: An Update-In-Place Key-Value Store for Modern Storage. *Proceedings of the VLDB Endowment* 16, 1 (2022), 99–112.

[85] Xiang Yu, Guoliang Li, Chengliang Chai, and Nan Tang. 2020. Reinforcement Learning with Tree-LSTM for Join Order Selection. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1297–1308.

[86] Jianqiu Zhang, Kaisong Huang, Tianzheng Wang, and King Lv. 2022. Skeena: Efficient and Consistent Cross-Engine Transactions. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 34–48. https://doi.org/10.1145/3514221.3526171

[87] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, Minwei Ran, and Zekang Li. 2019. An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 415–432. https://doi.org/10.1145/3299869.3300085

[88] Xiuwen Zheng, Subhasis Dasgupta, Arun Kumar, and Amarnath Gupta. 2022. AWESOME: Empowering Scalable Data Science on Social Media Data with an Optimized Tri-Store Data System. arXiv:2112.00833 [cs.DB]