Design Considerations Towards AI-Driven Co-Processor Accelerated Database Management

Vision Paper

Anh Trang Le
Gabriel Campero Durand
Otto-von-Guericke-Universität
Magdeburg, Germany
firstname.lastname@ovgu.de

Bala Gurumurthy
David Broneske
Otto-von-Guericke-Universität
Magdeburg, Germany
firstname.lastname@ovgu.de

Christoph Steup
Gunter Saake
Otto-von-Guericke-Universität
Magdeburg, Germany
firstname.lastname@ovgu.de

ABSTRACT

Adopting AI techniques for query optimization is an ongoing research interest in the database community. Currently, the search space for the best plan increases drastically, with the growing heterogeneity of the target hardware, the novel tuning choices offered, and co-processing. Hence, the need for AI techniques to identify such a best plan in a reasonable time-frame is imminent. Though AI-based solutions for improving query processing exist, there is still a need for principled designs able to incorporate the different innovations, leverage synergy effects, and keep with production-readiness expectations when using AI. In this paper, we propose a series of seven ideal characteristics we envision for such designs. We then propose an early system design, based on the Mariposa system, that supports these characteristics. Altogether we expect that this short paper could be a modest contribution towards AI-driven heterogeneous processing, emphasizing the practical aspects of a supportive and principled overall design.

Keywords

AI for DBMS, Self-driving DBMS, Heterogeneous query processing, Hardware-accelerated query processing

1. INTRODUCTION

In the recent decade, computer systems composed of multiple heterogeneous processors have quickly become the norm, rather than the exception [25]. Along with this rapid growth in hardware platforms, we also witness an increasing adoption of hybrid processor database systems that circumvent the 'power wall' [3] and show great potentials for speeding up query processing [22]. However, without tailored optimization strategies, these systems cannot achieve the best performance gains.

In fact, studies show that some GPU-accelerated systems with better operator-level implementations than their CPU counterparts still have poor performance for overall query processing in analytical benchmarks [27]. This is partly explained by the fact that optimizing query processing for such systems encounters numerous intrinsic challenges caused by the diversity of tuning techniques per device , the uncertainty (given such techniques) in accurately modelling real-world performance impact factors, the influence of workloads, as well as the need to support scalability to more devices and data [6]. All these aspects create a huge optimization space, which is hard to evaluate [15], turning the task of establishing a uniform research prototype for their detailed study into a tough nut to crack.

In this paper, we propose an early vision for a system architecture with the goal of exposing and easing query optimization. In traditional systems, optimization decisions are commonly addressed with hard-corded rules and heuristics. Such systems, can result in weaknesses for generalizing to unknown workloads or devices, as well as difficulties for maintenance. Throughout the last decade, there has been a shift towards employing AI techniques in handling these tasks more efficiently, alleviating the mentioned drawbacks of traditional methods. In the database community, there is a strong research trend that studies how AI can benefit database optimizations [18, 13, 19, 28, 4]. Though the prospect is bright, there are several obstacles coming from AI itself, especially in the deployment of AI solutions [12, 17]. Hence, adopting them in the co-processing domain forces us to solve a dual challenge: enhancing the performance of hybrid processor databases, as well as maintaining the AI, and specially machine learning (ML) models in production.

In order to confront this situation, we propose an early overall design for heterogeneous hardware database systems, which can facilitate the inclusion of learning from the ground-up, for addressing the different challenges in these systems. To achieve this we establish seven characteristics we deem as essential for the system: C1) task modularization, C2) collaborative agents as building blocks, C3) exchangeability of optimizers, C4) the separation of representation and policies, C5) concepts for database administrators (DBAs) to manage AI components, C6) ease of adaptation to training scenarios, and C7) learning from demonstrations. Overall, we propose that all these characteristics would contribute to improve the solutions for heterogeneous database query processing, while simultaneously addressing the needs for AI production readiness.

In more detail, our core contributions in this paper are:

32^{nd} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 25.05.2021 - 28.05.2021, Grimma, Germany.

- We present to the community a first proposal of seven ideal features that we deem as central to building and maintaining a practical AI-based database system.
- We introduce to the community an early high-level principled design for an AI-driven DBMS for heterogeneous processors, based on the Mariposa system introduced originally by Stonebraker et al. [24].

The remainder of this paper is structured as follows: Sec. 2, points out various challenges in incorporating AI components to support co-processor database management. Sec. 3 justifies our design decisions and describes the architecture of our proposed system. This section covers the system features, as well as the main workflow. Sec. 4, formalizes research questions that we aim to address with our proposed system. Sec. 5, provides context to our design, by considering related work. Finally, Sec. 6, wraps-up this paper with a summary and points for future work.

2. CHALLENGES OF HETEROGENEOUS DATA MANAGEMENT

In developing an AI-driven DBMS for heterogeneous processors, the literature suggests several challenges.

Storage Engine Design: From the perspective of a storage engine, the trade off between consistency, availability and partition-tolerance is a foremost concern [20]. Addressing availability, some challenges are: mechanisms to efficiently use scale-out processing to an increasing number of heterogeneous processors, while keeping in mind aspects such as different data transfer rates [1]. Up front data distribution strategies, such as hardware islands [22] or layered designs [27] for hot-cold data are commonly adopted, but exploration of alternatives is limited in the domain (e.g. [14]). Furthermore, co-processor-friendly data structures and storage optimization adapted to the diversity of application scenarios, devices and data characteristics (e.g. increasing relevance of textual and semi-structured data), remains important for availability. Addressing consistency, strong mechanisms for supporting isolation level guarantees are necessary for system maturity.

Summary: Altogether, relevant directions for storage technologies to enhance their co-processing efficiency, while keeping with consistency constraints, include: support for workload-tuned data distribution, transfer-aware processing, the ease for incorporating alternative processor-specific data optimizations (e.g. layouts, or compression), and finally the ability to seamlessly share data structures across processors.

Query Engine Design: The processing of single queries over heterogeneous hardware offers a large number of optimization choices, as compared to the case of homogeneous hardware (e.g. just-in-time code generation to fuse pipelined operators into unified kernels, more diversity of operator variants, or opportunities for resource sharing of concurrent kernels). Furthermore, there are many variants for a single operator present depending on the underlying device [3, 5]. This number of choices only increases when considering distributing single plans across multiple devices, or optimizing groups of queries at-a-time (which is relevant for high parallelism devices). Other than that, many relevant performance factors (e.g. device saturation, query expression complexity) or implementation details (e.g. cache consistency) could result in difficulties to model accurately for cost estimations.

Summary: Altogether, for efficient query processing over varied hardware, it is important to consider methods that are able to deal with large-scale optimization, and to work with uncertain models for performance factors.

AI Adoption: It might be conceptually simple to alter a certain computer system task (e.g. magic number selection, to build a hash function), replacing hard-coded rules with an AI-friendly interface that provides experience for a model, to eventually master the task. However, in practice it is far from trivial to build and maintain such models at the highest production-readiness levels [12]. In essence, unlike traditional software components, AI models can be harder to test and can fail in unexpected ways, specially for deep learning. Models can often be black boxes, or require copious training to be efficiently used. Machine learning models comprise of an entirely particular lifecycle going from data management tasks (including data collection and feature engineering), model learning (and tuning), validation and deployment, with challenges and cross-cutting concerns all through this lifecycle [17]. Some commonly discussed challenges are: insufficient data, concept drift, feedback loops and adversarial attacks.

Summary: Altogether the incorporation of AI components might drastically affect the guarantees that a system can provide. To overcome this issue, it is fundamental that the systems is made safe to critical AI errors with fall-back mechanisms, and finally that an easy-to-use interface is provided for administrators to engage with AI metrics and model lifecyle management.

3. DEVELOPING AN AI-ENABLED CO-PROCESSOR ACCELERATED DATABASE PROTOTYPE

In order to overcome the challenges of scalability as well as the need for adaptability and support for the AI/ML lifecycle, a principled design for building a system is required. To date, similar designs have already been considered by researchers in other areas, already offering design principles. In this section, we consider briefly some of such design concepts from a general, to a more specific case, which we then use to propose the series of ideal characteristics, that become the basis of our system design.

From a general application perspective, university courses[1] and textbooks already study good practices for building systems that incorporate machine learning [8]. Furthermore, there are several papers that highlight intrinsic challenges which require designs to adapt to them [23]. They usually refer to difficulties such as modularization, or to specific problems of an ML approach (e.g. [7]).

Moving to a more specific application, the authors of the AutoSys framework [16] suggest 4 principles organized around the goals of making systems learnable, and making the learning manageable: exposing system behavioral features for learning through well-defined interfaces (P1), careful monitoring of model behavior (P2), modularization of the learning to scope complexity (P3), and resource management for system exploration and maintenance (P4). In further work authors report experience in applying their framework, with practical tips for the machine learning scheduling.

[1]For example, SE4AI offered by Christian Kästner at CMU: `https://github.com/ckaestne/seaibib`

Similar design considerations have a long history in the community that studies self-driving data management, albeit not often coupled with AI (e.g. Babu et al. have argued for experiment-driven adaptive tuning by having replicated test databases [2]); most recently Kossmann and Schlosser also highlight the importance of modular designs and the plug-and-play nature of optimizations in designing such adaptive systems [10]. Furthermore, authors identify co-related tasks as a core challenge to efficient modularization, proposing and testing a linear programming framework that enables them to deal with such complication.

In recent years, research in AI-based databases has proposed designs tailored to the needs in the area. Due to space limitations, we discuss in the following a few key ideas from a limited set of them: Pavlo et al. [18] build a system designed with a principled distinction between workload modeling (i.e., representation learning) and system control (i.e., policies). In further work, they continue their approach while distinguishing between externally and internally coupled intelligent mechanisms [19], illustrated by their work in OtterTune and NoisePage, respectively. Within the research scope of SageDB, Kraska et al. [11] present and evaluate a comprehensive vision for how common database components can be replaced with AI. Among their core ideas authors develop the concept of instance optimality, which posits that a learned model for a database needs only to be provably optimal to the intended workload and system configuration. Finally, the authors of XuanYuan [13] present a broad high-level design that focuses on identifying what are the learnable components of current databases, considering task modularization, and categorizing tasks according to the functionality they offer to the overall system (e.g. self-healing, self-assembling).

Based on this preceding work, and on the challenges we identify in the domain of heterogeneous co-processing, we propose the following seven characteristics we envision that an AI-based database should reasonably offer for this domain. We should note that the proposed characteristics might not be exhaustive, but intend to serve as a starting point.

C1- Task modularization: The growing device heterogeneity increasingly expands the space of possible combinations of optimization choices that database systems have to consider. Already query optimizers tend to employ staging, wherewith optimizations are configured into stages and at each one there are specific sets of rules and mechanisms that can be adopted. Concerning ML components, employing a single, monolithic model to learn such a complex space and address the optimizing at a single shot can drastically escalate the learning cost and complexity. Task modularization is a good alternative, since the optimization problems to be tackled can be decomposed and solved separately, resulting easier to learn.

C2- Collaborative agents as building blocks: The best models for a given task on a selected device are only required to be instance optimal (i.e, their strategies do not need to generalize to other devices). Hence, as much as possible, it might be beneficial for designs to strive towards supporting device-specific simple instance-optimal models by decomposing a task (e.g. a single query optimization) into parts that can be solved in a collaboration among agents (e.g. subquery selection and optimization per device). To this end, clean abstractions for the task and communication protocols are required. This characteristic seeks to address the storage engine challenge for high device adaptability.

C3- Exchangeability of optimizers: Similar to current databases that already employ alternative optimizers in tasks like join order optimization, for a research-oriented prototype it becomes essential to support the use of alternative optimizers in a plug-and-play manner. To the point, extending optimizers (with new features), and integrating new optimizers should also be supported as design goals that would facilitate the evolution of the overall system.

C4- The separation of representation and policies: Following related work [18], we propose that a smart separation between representation learning (i.e., how a model decides to represent an entity) and policies (i.e., the decisions made by a model, for a task, given a representation), will be of value. This separation would facilitate representation re-use (transfer learning) across tasks that work on similar entities (e.g. a query) and the analysis of alternative multi-modal solutions for a task which could provide benefits on different scenarios (e.g. a query can be represented as multi-sets of traditionally encoded predicates, joins, tables; but it can also be represented as a graph of such features). As different policies can benefit from stable learned representations, this characteristic seeks to help in the aforementioned query engine challenges for large-scale optimization.

C5- Concepts for DBAs to manage AI components: In consideration of the many steps requiring human management in the ML lifecycle, we envision that the role of the DBA could be extended to incorporate a degree of actions to manage this lifecycle. To support this, novel services exposing ML management with clearly-defined interfaces will be needed in the database context.

C6- Ease of adaptation to training scenarios: Different user scenarios will create different alternatives for training the ML models. It might be that some scenarios accept live training in the background for a given tasks, while other scenarios might require collecting experience data for offline learning at a later stage. Some scenarios might allow for ample large-scale training, while training on other scenarios might be severely resource-constrained. In either case, the design for the system components in charge of scheduling model training should be able to cater to such variations. The ability of models to schedule self-training should also be supported.

C7- Learning from demonstrations: The final feature that we believe is essential for successful adoption of AI models to solve computer system tasks has to do with robustness. In order for the model to be able to replace a current strategy, an efficient and reasonable solution would be starting with the model by being pre-trained on experience collected from the current strategy. Hence mechanisms for creating and using demonstrations for training are important.

After listing these ideal system characteristics, we can now present a tentative design that can be adopted to fulfill them. We take this design as a starting point for future work.

At its core, we base our proposed design on Mariposa - a market-based distributed DBMS [24], which we will discuss further in Sec. 5. In general, Mariposa operates the query processing in a decentralized manner that allows for local autonomy regarding query execution in each site contained within a network, instead of centralized management.

Mariposa's working mechanism primarily bases itself on an economic paradigm, focusing on two separate markets, for data and query distribution, respectively. In our research, we argue for timely extensions to the original model in two key ways: First, by considering an architecture with heterogeneous processing capabilities. Second, by investigating how AI-based solutions can augment the proposed markets. In this regard, we come up with a design that takes as a hypothesis that the modularization of the optimization mechanisms presented in Mariposa (C1, C2) serve to scope the complexity of the learning tasks, enabling better scenarios for incorporating technological innovations as well as the productization readiness. Figure 1 envisions a general architecture of our AI-driven DBMS for heterogeneous processors.

At a high level, four components are involved: (1) **Global Optimizer**: This component maps SQL queries to actual query plans. It is also responsible for global query optimization including: the generation of global query plans, partial splitting of the plans (to distribute among devices), decision support for selection of plans returned from the device processor class optimizer, and (optionally) requests for data re-distribution. (2) **Storage Manager**: This component provides a centralized collection on statistics about devices and a tracking mechanism of data distribution schemes. It enables user-facing configurations of the overall storage, including schema management, index selection and coarse-grained partitioning. It also is intended to provide the DBA with an interface to the AI components, including learning from demonstration. Hence this component realizes C5 and C7. (3) **Device Processor Class Optimizer**: This is the key component for decentralized modularized data management. As we could propose a component per processor, or per compute node/device (i.e. irrespective of the co-processor variety included), we find that a component per type of processor serves as a workable middle-ground. This component is in charge of local data fragmentation, local query optimization (and pricing), algorithm selection and the actual execution of queries. It is also responsible for autonomous data sharing with other devices. (4) **AI Support System**: This element encompasses the functionality required to support the ML lifecycle. It includes model management, model training, among others. It is intended to facilitate C6 and C7.

Our architecture enables the distribution of query and sub-query plans for cost estimation on the device processor class optimizers; besides the distribution of data driven by the device-specific component, in addition to (partial input from) the global optimizer.

The lifecycle for query processing in our system, can be understood as follows: First, a group of queries enters the system at a given timestep, and at the global optimizer, they are ranked by their importance to overall performance goals. Second, they are globally partitioned and subsets of their plans are shipped to the devices, for pricing. In third place, the different device processor class optimizers provide a set of optimizations and prices for the queries requested. To do this, they featurize the query plans (C4), and suggest different combinations of sub-queries to execute with different costs (for this they consider local data statistics and learned models for algorithm selection). The prices are then returned, in a fourth step, to the global optimizer, so this optimizer can select among the bids until all queries are served. Once choices are made, finally queries can be executed on the devices. This scheme describes a *query market*, similar to the proposed by Stonebraker et al. [24]. By framing the problem in economic terms, this approach helps decentralized coordination and favors local strategies for optimization (C2). Some optimization choices include: variant selection, operator merging into unified kernels, different sub-query splitting and pipelining strategies in a single or cross-device setting, parallelism tuning with morsel-driven execution, locality awareness, intermediate results reuse and operator sharing across queries.

The lifecycle of data distribution, which occurs in the background of our proposed system can be understood as follows: On system start, given the lack of information for distributing the data, some assumptions can be made by the storage manager to achieve fragmentation and distribute the data for load balancing. In general, data can be grouped into fragments that are commonly co-accessed and that provide a given utility. While the system is online there are two ways in which data can be redistributed: First, when an optimal plan for a query cannot be found, global requests for data reorganization can be made by the global optimizer (with some pre-designed mechanism). Following these requests, the device optimizers can organize autonomously how to serve the global hints. Second, device optimizers themselves are responsible for tracking the utility derived from a given data fragment (i.e., depending on the queries that can be served by such fragment). This enables devices to have metrics to be able to assess how much utility can be derived from fragments that are not locally available. Hence, by using information from the storage manager, device optimizers can participate in a *data market*. In this market, devices buy copies of fragments, and delete local copies of fragments, while keeping with some constraints (e.g., for co-location or availability). The market formulation is expected to facilitate adaptivity and work distribution (C2). Some learning tasks to be tackled with this system include: local algorithm selection, local query optimization, global plan selection,, local fragment partitioning, data sharing, global management for data redistribution, query classification/prioritization.

Altogether our design is intended to realize the ideal characteristics we set as goals (C1-C7). To achieve this our design seeks to build on characteristics (C1,C2) of the original Mariposa design, to facilitate the use of alternative optimizers or models for a task (C3), to provide chances for reusing representations across components (e.g. of queries with respect to the device optimizers), which that can then adopt different policies (C4), and creating an opportunity for components to have flexible AI training including the DBA and leveraging demonstration data (C5-C7).

4. OPEN QUESTIONS

Based on our proposed design where we describe a certain modularization of tasks, while following ideal design characteristics that we have proposed (C1-C7), in this section we turn to open questions that we envision our design to be able to help address. These questions relate to query engine (Q1-Q2), storage engine (Q3) or machine learning (Q4-Q5) challenges.

Q1: What building blocks for intelligent and collaborative query processing are necessary to achieve improvements on heterogeneous processors, considering single-query optimization –focusing on algorithm selection, parallelism tuning, splitting, merging and pipelining of operators; compared to

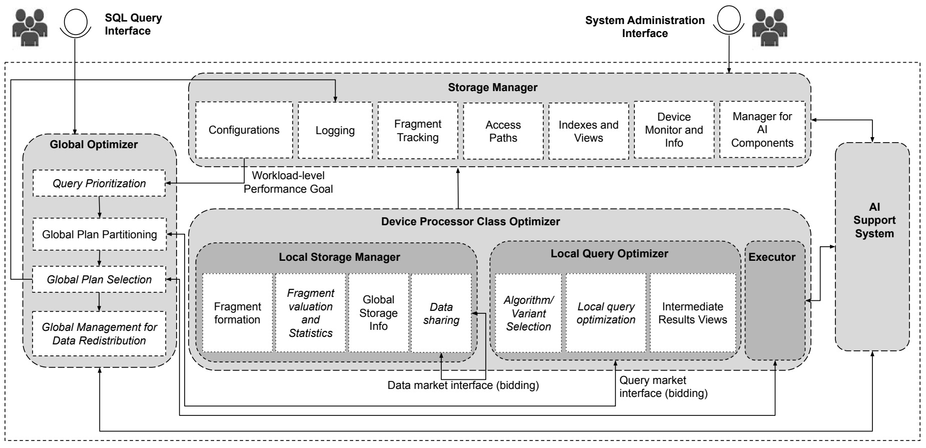

Figure 1: General Architecture of our Proposed System

strong baselines?
Q2: What strategic designs for intelligent and collaborative query processing lead to performance gains on heterogeneous processors, considering multi-query optimization (MQO) – with a focus on intermediate results reuse and operator sharing? To what degree do intelligent methods compete with non AI-based alternatives?
Q3: What precise contributions are brought from different applications of AI, to the efficiency of data sharing across co-processors; contrasted with competitive baselines?
Q4: How do AI-based approaches perform in robustness tests, compared to heuristic baselines with respect to changing assumptions such as novel processors or unseen workloads/kinds of queries? What level of improvements does curricula management bring regarding robustness and sample-efficiency?
Q5: What techniques from learning management (such as learning from demonstrations, or transfer optimization) or from database implementation contribute the most to an efficient integration of the AI components into the lifecycle of data management? Do these techniques contribute to trade-off management between approaches? To what extent do these techniques improve the overall readiness of our solution over baseline choices?

5. RELATED WORK

5.1 AI-based database systems

Incorporating AI components to traditional systems, for improving the overall system performance, is a significant topic that is currently catching great attention from researchers, in both theoretical and applied aspects. On one hand, multiple studies investigate the strategies for an overall co-design of systems and AI, fitting for general applications [16] and more specific ones in databases [19]. On the other hand, many research zooms into the particular problems that can benefit from using suitable AI techniques. Those, in databases, range from knob tuning with the aid of deep reinforcement learning [26], cost and cardinality estimation assisted by deep neural networks [9], and many more [28, 13]. In a bigger scope, other recent literature also studies complete database management system assisted by machine intelligence, instead of focusing on only one or some certain tasks within it. Some highlighted work, which can be considered to be in a relative early stage, include Peloton [18], SageDB [11] and GaussDB[2].

5.2 Market-based distributed database systems

In economics, a *market* is defined as any structure that enables trading activities among its participants, for any types of goods, services or information, following a pricing mechanism that aims for optimal distribution and allocation of resources. Interestingly, this concept has been reformulated to efficiently solve the problems of query optimization in many distributed data management systems. To help with some of our ideal design characteristics (C2), we consider this concepts to be relevant.

The system that we base our design on is Mariposa [24], which adopts market concepts to achieve autonomous data sharing and query processing. In the wide-area network, Mariposa allows each single site takes a full control over its own resources, the data objects to buy or sell and the queries to bid on to execute. And a bidding protocol is defined to regulate the transactions among all sites within the two markets: 1) *Query Market*: each query Q enters the system with a budget $B(t)$ indicating the price that the user wants to pay for running Q within time t. Also, Q is administered by a *broker*, which sends out to bidder sites the requests for bids to execute subqueries $Q_1, ..., Q_n$ and then decides on the winning sites. 2) *Data Market*: each table included in the FROM clause of a query can be split into a set of *fragments*. A site needs to buy fragments referenced in the subquery that it wants to bid on, and can sell its must-evicted fragments at any time by conducting an auction, following the system pricing mechanism. The trading process runs continuously. Each site makes decisions on storing, buying and selling fragments or the replicas of fragments made by the site itself, aiming at maximizing its profit per unit time.

[2]https://e.huawei.com/en/solutions/cloud-computing/big-data/gaussdb-distributed-database

Another framework that is based on an economic paradigm to acquire self-adaptive query allocation in large-scale distributed systems is SQLB [21], in which the authors highlight the importance of maintaining constantly the interests of the participators throughout the on-going market. The system targets at preserving participants' satisfaction on query allocation/execution and guaranteeing query load balancing within the system, which then helps in minimizing the response time and maximizing system throughput.

NashDB [14] is a more recent framework that shows efficiency in autonomously handling data fragmentation, replicas generation, allocation and cluster sizing to attain the *Nash equilibrium*, i.e. supply-demand balance in markets.

6. CONCLUSION

The search space of traditional query optimizers is very large. Such search space is further increased many folds with the introduction of co-processors for query execution. AI techniques are promising solutions in traversing such a large search space, identifying the best plan in an effective and time-efficient way. As a consequence, there is a growing body of research devoted to AI-based solutions [28]. However, turning these solutions into a production-ready contribution remains a challenge since AI, and machine learning in specific, contain many intrinsic challenges that require overall system considerations. In sum, systems builders are placed in the difficult position of having to simultaneously tackle a set of heterogeneous co-processing challenges, next to a set of AI adoption challenges.

As a motivation for this work, we considered that a principled system design could contribute to addressing the aforementioned 2 sets of challenges, while at the same time helping in the integration of different technological innovations. In order to contribute towards this goal, in this paper we summarized a list of preceding work that helped us to identify 7 design characteristics (C1-C7), addressing needs for scoping complexity and difficulty of learning (C1, C7), high adaptability/instance optimality (C2-C3), scale (C4), and machine learning issues in general (C5-C6). Based on this we proposed an early overall system design, based on the Mariposa system and on economic concepts for a data and query market. We propose this design to fulfill the design characteristics, while offering an AI-based heterogeneous co-processing database. To conclude, we listed open questions that we would like to review by using our proposed design.

7. ACKNOWLEDGMENTS

This work was partially funded by the DFG (grant no.: SA 465/51-1 and PI 447/9). The authors would like to thank Marcus Pinnecke, Andrey Kahritonov, Rajatha Rao and Yash Shah for collaborations related to this work.

8. REFERENCES

[1] I. Arefyeva, D. Broneske, G. Campero, M. Pinnecke, and G. Saake. Memory management strategies in CPU/GPU database systems: A survey. In *BDAS*.

[2] S. Babu, N. Borisov, S. Duan, H. Herodotou, and V. Thummala. Automated experiment-driven management of (database) systems. In *HotOS*, 2009.

[3] D. Broneske, S. Breß, M. Heimel, and G. Saake. In *EDBT*.

[4] G. C. Durand, M. Pinnecke, R. Piriyev, M. Mohsen, et al. GridFormation: Towards self-driven online data partitioning using reinforcement learning. In *aiDM@SIGMOD*.

[5] B. Gurumurthy, D. Broneske, M. Pinnecke, G. Campero, and G. Saake. Simd vectorized hashing for grouped aggregation. In *ADBIS*.

[6] B. Gurumurthy, T. Drewes, D. Broneske, G. Saake, and T. Pionteck. Adaptive data processing in heterogeneous hardware systems. In *GvDB*.

[7] A. Haj-Ali, N. K. Ahmed, T. Willke, J. Gonzalez, et al. A view on deep reinforcement learning in system optimization. *arXiv preprint arXiv:1908.01275*, 2019.

[8] G. Hulten. *Building Intelligent Systems*. Springer, 2019.

[9] A. Kipf, D. Vorona, J. Müller, T. Kipf, et al. Estimating cardinalities with deep sketches. In *SIGMOD*. ACM, 2019.

[10] J. Kossmann and R. Schlosser. Self-driving database systems: A conceptual approach. *DAPD*.

[11] T. Kraska, M. Alizadeh, A. Beutel, H. Chi, et al. SageDB: A learned database system. In *CIDR*, 2019.

[12] A. Lavin, C. M. Gilligan-Lee, A. Visnjic, S. Ganju, et al. Technology readiness levels for machine learning systems. *arXiv*, 2021.

[13] G. Li, X. Zhou, and S. Li. Xuanyuan: An AI-native database. *IEEE Data Eng. Bull.*, 42(2), 2019.

[14] R. Marcus, O. Papaemmanouil, S. Semenova, and S. Garber. NashDB: An end-to-end economic method for elastic database fragmentation, replication, and provisioning. In *SIGMOD*. Association for Computing Machinery, 2018.

[15] A. Meister, S. Breß, and G. Saake. Toward GPU-accelerated database optimization. *Datenbank-Spektrum*, 15(2).

[16] C.-J. Mike Liang, H. Xue, M. Yang, and L. Zhou. The case for learning-and-system co-design. *ACM SIGOPS Operating Systems Review*, 53(1).

[17] A. Paleyes, R.-G. Urma, and N. Lawrence. Challenges in deploying machine learning: A survey of case studies. *arXiv*, abs/2011.09926, 2020.

[18] A. Pavlo, G. Angulo, J. Arulraj, H. Lin, et al. Self-driving database management systems. In *CIDR*, volume 4, 2017.

[19] A. Pavlo, M. Butrovich, A. Joshi, L. Ma, et al. External vs. internal: An essay on machine learning agents for autonomous database management systems. *IEEE Data Eng. Bull.*, 42, 2019.

[20] M. Pinnecke, D. Broneske, G. C. Durand, and G. Saake. Are databases fit for hybrid workloads on GPUs? A storage engine's perspective. In *ICDE*.

[21] J.-A. Quiane-Ruiz, P. Lamarre, and P. Valduriez. SQLB: A Query Allocation Framework for Autonomous Consumers and Providers.

[22] A. Raza, P. Chrysogelos, P. Sioulas, V. Indjic, et al. GPU-accelerated data management under the test of time. In *CIDR*, 2020.

[23] I. Stoica, D. Song, R. A. Popa, D. Patterson, et al. A berkeley view of systems challenges for ai. *arXiv preprint arXiv:1712.05855*, 2017.

[24] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, et al. Mariposa: A wide-area distributed database system. *VLDB Journal*, 5(1), 1996.

[25] M. Zahran. Heterogeneous computing: Hardware and software perspectives. ACM, 2016.

[26] J. Zhang, Y. Liu, K. Zhou, G. Li, et al. An end-to-end automatic cloud database tuning system using deep reinforcement learning. Association for Computing Machinery, 2019.

[27] Y. Zhang, Y. Zhang, J. Lu, S. Wang, et al. One size does not fit all: Accelerating OLAP workloads with GPUs. *DAPD*, 38, 2020.

[28] X. Zhou, C. Chai, G. Li, and J. Sun. Database meets artificial intelligence: A survey. *TKDE*, 2020.