Disaggregation: A New Architecture for Cloud Databases

Xiangyao Yu
University of Wisconsin-Madison
yxy@cs.wisc.edu

ABSTRACT

Disaggregation—the separation of database components into independently managed and scalable services—has emerged as a foundational architecture for cloud-native databases. It enables key benefits such as elasticity, resource pooling, and cost efficiency. This paper offers a perspective on the disaggregation trend, tracing its evolution, and presents a set of research efforts that redesign and optimize distributed databases in this new architecture. Finally, the paper outlines future directions and open challenges, highlighting disaggregation as a rich and still largely unexplored area for database research.

PVLDB Reference Format:
Xiangyao Yu. Disaggregation: A New Architecture for Cloud Databases. PVLDB, 18(12): 5527 - 5530, 2025.
doi:10.14778/3750601.3760520

1 INTRODUCTION

Databases are transitioning from on-premises deployments to the cloud. Modern cloud databases adopt a **disaggregation architecture** where different system components, such as computation and storage layers, are managed as *physically separated* services. Disaggregation enables independent scaling and billing of resources, as well as resource pooling, which significantly improves cost efficiency and elasticity of cloud databases.

Disaggregation represents a fundamental architectural shift that departs from traditional assumptions in database systems. It extends distributed databases from a single tightly coupled cluster to multiple loosely coupled clusters, each responsible for a subset of database functions. This shift opens a vast new design space: rethinking classic database protocols, redistributing traditional database functions across disaggregated components, introducing new disaggregated components to enable novel features, and beyond. Optimizations for the disaggregation architecture have been explored in both research and production systems in recent years, but many challenges and research opportunities remain, especially as cloud platforms and cloud databases continue to evolve.

This paper aims to offer a perspective on how disaggregation is reshaping the database landscape today and potential directions for the future. The paper begins by briefly describing the key characteristics of the disaggregation architecture and its evolution, from storage disaggregation to more general disaggregation (Section 2). It then highlights several research projects from our lab that introduce new techniques to optimize for the architecture (Section 3). Finally, the paper discusses several future directions from the author's perspective (Section 4), followed by a conclusion (Section 5).

Proceedings of the VLDB Endowment, Vol. 18, No. 12 ISSN 2150-8097.
doi:10.14778/3750601.3760520

2 THE EVOLUTION OF DISAGGREGATION ARCHITECTURE

A key advantage of the cloud over on-premises systems is *on-demand scalability*—the capability for users to dynamically allocate and release resources and pay only for what they use. Classic database architectures, such as shared-nothing, struggle to fully exploit this feature. As a result, cloud-native databases have begun to adopt a new disaggregation architecture.

2.1 Storage disaggregation

Early cloud-native databases, such as Snowflake [9, 22] and Aurora [20, 21], adopt a storage-disaggregation architecture, where *compute* and *storage* clusters are physically separated. The two clusters can scale independently and often use different cluster sizes and machine types.

The disaggregation of storage and compute is driven by the fundamental mismatches between these two services: (1) Compute is significantly more expensive than storage in modern cloud environments. (2) Compute demands fluctuate more drastically while storage demands change slowly. (3) Compute can often be stateless and thus easier to scale in contrast to the inherently stateful storage service. By decoupling these two services, the expensive compute layer can quickly scale up/down and out/in to accommodate workload changes, while the cheaper storage service can stay relatively stable with less frequent reconfigurations.

Storage disaggregation resembles the traditional on-premises shared-disk architecture in that both physically separate the compute and storage components. However, cloud storage services offer richer capabilities, such as built-in high availability, multi-region durability, built-in horizontal scalability, and advanced APIs. These capabilities enable new use cases beyond what traditional shared-disk systems could support. Moreover, the principle of disaggregation can be generalized beyond compute and storage, as discussed in the next subsection.

2.2 Generalized Disaggregation

Besides enabling independent scalability, disaggregation can also improve the modularity of complex systems and facilitate sharing and pooling of resources, leading to higher efficiency. Driven by these salient features, modern cloud databases are being disaggregated into even more components, beyond just compute and storage. The list below shows several examples but is by no means exhaustive.

Further Disaggregated Storage.: Socrates [3] adopts a design similar to Aurora but further disaggregates the storage layer into (1) a logging service, (2) a page cache, and (3) a durable page store.

These services have different storage footprint, performance, and cost tradeoffs that can be better optimized when they are physically separated. For example, the logging service has a small data footprint but stringent write-latency requirements, and therefore can be deployed over more advanced storage technologies.

Computation Pushdown.: Both Redshift Spectrum [6] and S3 Select [1] introduce a serverless layer close to the storage service to process a subset of query operators, such as filtering and aggregation. Remote Compaction in RocksDB [11] pushes compaction in LSM tree to a dedicated host. The pushdown layer can execute these operators in a serverless manner with massive parallelism and high cost-efficiency. It can significantly reduce the network traffic sent to the compute layer, which improves the overall performance.

Intermediate Data Caching.: Snowflake introduces a *Distributed Ephemeral Storage* layer for spilling intermediate results [22]; the insight is that intermediate query results do not require strong durability but prefer lower access latency. Instead of using S3, a storage service specifically designed for intermediate results can make better performance and cost tradeoff.

Metadata Layer in Lakehouse.: The lakehouse architecture [5] introduces a metadata layer that sits between the compute and storage layers. The metadata layer can support various database functions, including transaction management [4], data quality enforcement, and data governance features, etc. Similar to the computation pushdown layer, this middle layer can handle operations that are closely tied to storage but cannot be easily pushed into the storage service itself.

Memory Disaggregation.: PolarDB [8] goes beyond storage disaggregation and further disaggregates a shared pool of remote memory. This design allows memory to be provisioned, scaled, and shared independently from compute, improving overall resource utilization and reducing memory over-provisioning. High-speed network technology such as RDMA or CXL can mitigate the increased memory access latency.

2.3 Design Tradeoff in Disaggregation Architecture

Disaggregation enables rich design flexibility and optimizations that were not possible in traditional monolithic database architectures. Each component in a disaggregated system can itself be a complex distributed system, offering rich functionalities. Given the large number of functions in a typical database, there exists an enormous design space for partitioning these functions into physically separated system components. Today, we are still in the early stages of exploring this design space. As cloud platforms continue to evolve, this space will likely expand further, opening up new opportunities for research and system development.

One important tradeoff in disaggregated databases is the degree of disaggregation vs. performance. Since disaggregated components are physically separated, communication between components can incur significant overhead. In general, the more aggressively a system is disaggregated, the higher performance overhead it needs to pay. In fact, with sufficient optimizations, traditional shared-nothing architecture can incur less network traffic and offer better performance than disaggregation architectures [19]. As a result, disaggregation should be applied judiciously and we should avoid disaggregating components when the resulting communication overhead cannot be justified. At the same time, this tradeoff motivates new research opportunities that can reduce communication costs between disaggregated components.

3 EXPLORING THE DESIGN SPACE OF DISAGGREGATION

In this section, I present several projects from my lab that explore the design space of disaggregated databases. These efforts revisit fundamental protocols (Section 3.1), re-architect core database functions (Section 3.2), and demonstrate how disaggregation can enable new system capabilities (Section 3.3).

3.1 Rethinking Core Protocols

Many foundational protocols in database systems were designed with the assumption of a traditional architecture, such as shared-nothing. As a result, they need to be revisited in the context of disaggregation. One such example is the two-phase commit (2PC) protocol.

Cornus. 2PC is a protocol for a distributed transaction to reach a final decision (i.e., commit or abort) across participating servers. Each participant logs its own vote (i.e., VOTE-YES or VOTE-NO) locally, and a coordinator logs the final decision, which determines the transaction's outcome. If any participant votes no, the transaction must abort. Even if all participants vote yes, the outcome may still be an abort under certain failure scenarios—for example, if the coordinator times out while waiting for votes.

A well-known limitation of 2PC is the *blocking problem*, which occurs when the coordinator fails before broadcasting the final decision. In a shared-nothing architecture, where compute and storage are tightly coupled, the coordinator's log becomes inaccessible after a failure. As a result, the system cannot determine the transaction's outcome or even whether a decision was made. This uncertainty forces all participants of the pending transaction to hold their locks, potentially blocking other transactions indefinitely until the failed coordinator recovers and replays its log.

Fundamentally, the blocking problem exists because a failed node's log cannot be accessed by other servers in the system. While this is true in shared-nothing systems, it is no longer the case in a storage disaggregation architecture, where storage is provided as a separate, highly available service. Even if a compute server fails, its log is stored in the storage service and remains accessible to other active servers in the system.

Cornus [13] is a 2PC protocol specifically optimized for storage disaggregation, leveraging the insight above. While the full protocol is described in detail in the original paper, its core idea is simple: Cornus allows active nodes to vote NO on behalf of failed nodes by directly writing to the failed node's log in the disaggregated storage service. It uses a compare-and-swap-like API to ensure that only one decision can be recorded, preserving correctness. Another benefit of Cornus is that the critical execution path of 2PC is reduced from two logging events to one logging event. This optimization is viable because the ground truth of a distributed transaction is no longer determined by the coordinator's log but the *collective* *votes from all participants' log files*; this allows the removal of the coordinator's log which reduces latency.

Marlin. The disaggregation in modern cloud databases mostly focus on the data planes. For the control plane, many disaggregated systems still rely on external, centralized coordination services, such as ZooKeeper [15] and etcd [2]. These coordination services lack elasticity, thereby introducing operational complexity and performance bottleneck.

Marlin [14] is a cloud-native coordination mechanism (i.e., the control plane) that is specifically designed for the storage disaggregation architecture. Marlin eliminates the need for external coordination services by consolidating coordination functionality into the existing cloud-native database it manages. Specifically, it stores coordination state (e.g., cluster membership, data mapping) in the disaggregated storage layer and performs coordination logic using compute-layer transaction managers. This design enables scalable, cost-efficient coordination for disaggregated databases.

To ensure correctness and efficiency, Marlin adopts a new 2PC protocol that further extends Cornus. In Marlin's commit protocol, a participant does not have to be a compute node; instead, it can be either a compute node or a log instance in the disaggregated storage. This allows different compute nodes to initiate a reconfiguration, and enforce a global decision across multiple log files.

3.2 Disaggregating The Query Engine

One limitation of disaggregation architecture is the network overhead between different components. In our earlier study [19], we found that disaggregation can impose a 10× throughput degradation compared to a highly-optimized shared-nothing database due to excessive network traffic. Therefore, an important goal in disaggregated database design is to develop optimizations to mitigate the network bottleneck.

Computation pushdown is a well-established technique to reduce data traffic to the compute engine, especially when the pushdown operators are selective. The idea has been extensively explored in the context of database machines [12, 23], Smart Disk/SSD [10], processing-in-memory (PIM) [16], and cloud databases [6]. In fact, the pushdown idea fits even better in the storage disaggregation context compared to computational storage devices (e.g., Smart Disk/SSD, PIM). This is because cloud storage, as a service, has richer support for resource management, security, and data consistency, compared to hardware devices [7, 27].

PushdownDB. We have developed PushdownDB [28], a database engine that uses S3 Select [1], a serverless layer in front of S3, to pushdown basic operators (e.g., filter, aggregation) that are natively supported in S3 Select, and more advanced operators (e.g., group-by, top-K, probe in hash join) that we built by leveraging existing filter and aggregation support. PushdownDB can reduce query runtime by 6.7× and cost by 30%, validating the potential of the idea.

FlexPushdownDB (FPDB). One issue of pushdown is its inherent tension with data caching in the compute layer—another technique to reduce network traffic; most systems implement only one of these two ideas. In FlexPushdownDB (FPDB) [24], we aim to combine these two techniques in a single design. The key observation is that many common operators—such as filtering, aggregation, hash probe, etc.—can execute on both cached data locally and use pushdown to process remote data simultaneously; the results of the two execution paths can then be merged. FPDB employs a *fine-grained hybrid execution mode* to combine the benefits of caching and pushdown, and outperforms both techniques alone by 2.2×.

Adaptive Pushdown. By default, a pushdown request does not consider the pushdown layer's computational capacity, which can sometimes be scarce (e.g., due to multi-tenancy), causing pushdown to underperform. With adaptive pushdown [25], the pushdown layer can choose to *push back* the task if it has no resource to execute it, and the compute layer can directly read the remote data to execute the task locally. This work also identifies two more operators that are amenable to pushdown, *selection bitmap* and *distributed data shuffle*, which are common in distributed columnar databases. These techniques lead to 1.7–3× further speedup.

3.3 Enabling New Capabilities

Modern applications increasingly demand *real-time analytics* so that the most up-to-date insights can be extracted from the data. Hybrid Transactional/Analytical Processing (HTAP) addresses this need by integrating TP and AP into a single engine. However, existing HTAP solutions require a compulsory migration—users need to migrate from existing TP and AP databases to a new HTAP engine, incurring extra migration cost and complexity.

Hermes. We aim to achieve *off-the-shelf* real-time analytics on top of existing TP and AP engines, so that users can enjoy real-time analytics without migrating away from their existing databases running in the cloud. The key idea is to introduce a new *Hermes layer* [17] that sits between compute and storage, that intercepts the logging events in the TP engine(s) and the read requests in the AP engine(s). Hermes will replay the recent transactional logs from TP engines and merge the updates into the analytical reads from the AP engines, such that each analytical query can see the latest updates. In the background, Hermes will merge updates in batches into the stable analytical storage.

Hermes also supports *True HTAP transactions* [18], which are transactions that contain long-running analytical queries within. We refer to this capability as *Transactional Analytics*. Hermes allows the analytical query within a transaction to run in the AP side of the system, thereby reducing the overall execution time. Hermes can support different isolation levels for these cross-engine transactions, such as read committed, snapshot isolation, and serializability.

4 FUTURE WORK

We are still in the early stage in exploring the design space of disaggregation architecture and tremendous opportunities exist ahead for the community to explore. Below are few directions that I find promising. This list is by no means exhaustive, but rather a set of initial thoughts intended to spark deeper discussion and inspire new ideas.

Disaggregate More Database Functions. While the disaggregation of many database functions have been studied as discussed in earlier sections, many other database functions (e.g., indexing, concurrency control, query optimization, statistics management, materialized view, etc.) remain underexplored. Moreover, many of these functions can be consolidated into a unified disaggregated component; for example, the pushdown layer, lakehouse metadata layer, and Hermes layer are all middle layers between compute and storage. The design space opens up rich research opportunities.

Multi-Cloud Database. Disaggregation architecture today largely focuses on a single cloud. When databases expand to a multi-cloud environment (e.g., multiple public clouds or hybrid public/private clouds), the cross-cloud communication overhead can be significant. This calls for the design of *multi-disaggregated* systems, where components are disaggregated within each cloud but the communication between clouds must be a first-class design consideration.

Embrace New Hardware.: Disaggregation naturally facilitates the adoption of new hardware, such as GPU, FPDB, RDMA, CXL, since different components can use different hardware for the best performance cost tradeoff. This flexibility opens up even further research and development opportunities. For example, in our own research, we use GPU to replace the execution engine of DuckDB and achieve significant speedup by leveraging the massive parallelism of GPU hardware [26].

5 CONCLUSION

Disaggregation is emerging as the new architectural trend for cloud-native databases, offering new opportunities and challenges for performance, cost efficiency, elasticity, and modularity. We are still in the early stages of exploring this paradigm, and the design space remains vast and largely unexplored. Now is a great time for the community to rethink traditional assumptions and build a new system foundation for disaggregated cloud databases.

ACKNOWLEDGMENTS

The work presented in this paper was mainly done by my outstanding PhD students Zhihan Guo, Elena Milkai, Yifei Yang, and Wenjie Hu. I would like to thank Michael Stonebraker for his invaluable mentoring, guidance, and collaboration over the years. I was fortunate to work with many amazing collaborators, most notably Philip Bernstein, Mahesh Balakrishnan, Jignesh Patel, Marco Serafini, and Ashraf Aboulnaga. I want to thank AnHai Doan, Joseph Hellerstein, Andrew Pavlo, Samuel Madden, and Daniel Abadi for their constant encouragement and support during my early research career. This research was funded in part by the NSF CAREER Award IIS-2144588 and NSF Award STTR-2135007, a Sloan fellowship, a Microsoft Research PhD Fellowship, the Wisconsin Alumni Research Foundation (WARF), gifts from Google and Snowflake, and donations from UW–Madison alumni.

REFERENCES

[1] [n.d.]. S3 Select and Glacier Select – Retrieving subsets of objects. https://aws.amazon.com/blogs/aws/s3-glacier-select. Accessed: 2025-08-07.
[2] 2024. etcd. https://etcd.io. Accessed: 2025-08-07.
[3] Panagiotis Antonopoulos, Alex Budovski, Cristian Diaconu, Alejandro Hernandez Saenz, Jack Hu, Hanuma Kodavalla, Donald Kossmann, Sandeep Lingam, Umar Farooq Minhas, Naveen Prakash, et al. 2019. Socrates: The new sql server in the cloud. In *SIGMOD*. 1743–1756.
[4] Michael Armbrust, Tathagata Das, Liwen Sun, Burak Yavuz, Shixiong Zhu, Mukul Murthy, Joseph Torres, Herman van Hovell, Adrian Ionescu, Alicja Łuszczak, et al. 2020. Delta lake: high-performance ACID table storage over cloud object stores. *VLDB* 13, 12 (2020), 3411–3424.
[5] Michael Armbrust, Ali Ghodsi, Reynold Xin, Matei Zaharia, et al. 2021. Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics. In *CIDR*, Vol. 8. 28.
[6] Nikos Armenatzoglou, Sanuj Basu, Naga Bhanoori, Mengchu Cai, Naresh Chainani, Kiran Chinta, Venkatraman Govindaraju, Todd J Green, Monish Gupta, Sebastian Hillig, et al. 2022. Amazon Redshift re-invented. In *SIGMOD*. 2205–2217.
[7] Antonio Barbalace and Jaeyoung Do. 2021. Computational storage: Where are we today?. In *CIDR*.
[8] Wei Cao, Yingqiang Zhang, Xinjun Yang, Feifei Li, Sheng Wang, Qingda Hu, Xuntao Cheng, Zongzhi Chen, Zhenjun Liu, Jing Fang, et al. 2021. Polardb serverless: A cloud native database for disaggregated data centers. In *SIGMOD*. 2477–2489.
[9] Benoit Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, Allison W. Lee, Ashish Motivala, Abdul Q. Munir, Steven Pelley, Peter Povinec, Greg Rahn, Spyridon Triantafyllis, and Philipp Unterbrunner. 2016. The Snowflake Elastic Data Warehouse. In *SIGMOD*. 215–226.
[10] Jaeyoung Do, Yang-Suk Kee, Jignesh M. Patel, Chanik Park, Kwanghyun Park, and David J. DeWitt. 2013. Query Processing on Smart SSDs: Opportunities and Challenges. In *SIGMOD*. 1221–1230.
[11] Siying Dong, Shiva Shankar P, Satadru Pan, Anand Ananthabhotla, Dhanabal Ekambaram, Abhinav Sharma, Shobhit Dayal, Nishant Vinaybhai Parikh, Yanqin Jin, Albert Kim, et al. 2023. Disaggregating rocksdb: A production experience. In *SIGMOD*, Vol. 1. ACM New York, NY, USA, 1–24.
[12] Phil Francisco. 2011. The Netezza Data Appliance Architecture.
[13] Zhihan Guo, Xinyu Zeng, Kan Wu, Wuh-Chwen Hwang, Ziwei Ren, Xiangyao Yu, Mahesh Balakrishnan, and Philip A Bernstein. 2022. Cornus: Atomic commit for a cloud DBMS with storage disaggregation. *VLDB* 16, 2 (2022), 379–392.
[14] Wenjie Hu, Guanzhou Hu, Mahesh Balakrishnan, and Xiangyao Yu. 2026. Marlin: Efficient Coordination for Autoscaling Cloud DBMS. In *SIGMOD*.
[15] Patrick Hunt, Mahadev Konar, Flavio P Junqueira, and Benjamin Reed. 2010. {ZooKeeper}: Wait-free coordination for internet-scale systems. In *USENIX ATC*.
[16] Tiago R. Kepe, Eduardo C. de Almeida, and Marco A. Z. Alves. 2019. Database Processing-in-Memory: An Experimental Study. *VLDB* 13, 3 (2019), 334–347.
[17] Elena Milkai, Xiangyao Yu, and Jignesh Patel. 2025. Hermes: Off-the-Shelf Real-Time Transactional Analytics. *VLDB* (2025).
[18] Fatma Özcan, Yuanyuan Tian, and Pinar Tözün. 2017. Hybrid transactional/analytical processing: A survey. In *SIGMOD*. 1771–1775.
[19] Junjay Tan, Thanaa Ghanem, Matthew Perron, Xiangyao Yu, Michael Stonebraker, David DeWitt, Marco Serafini, Ashraf Aboulnaga, and Tim Kraska. 2019. Choosing A Cloud DBMS: Architectures and Tradeoffs. *VLDB* 12, 12 (2019), 2170–2182.
[20] Alexandre Verbitski, Anurag Gupta, Debanjan Saha, Murali Brahmadesam, Kamal Gupta, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvili, and Xiaofeng Bao. 2017. Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases. In *SIGMOD*. 1041–1052.
[21] Alexandre Verbitski, Anurag Gupta, Debanjan Saha, James Corey, Kamal Gupta, Murali Brahmadesam, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvilli, et al. 2018. Amazon Aurora: On Avoiding Distributed Consensus for I/Os, Commits, and Membership Changes. In *SIGMOD*. 789–796.
[22] Midhul Vuppalapati, Justin Miron, Rachit Agarwal, Dan Truong, Ashish Motivala, and Thierry Cruanes. 2020. Building an Elastic Query Engine on Disaggregated Storage. In *NSDI*. 449–462.
[23] Ronald Weiss. 2012. A Technical Overview of the Oracle Exadata Database Machine and Exadata Storage Server. *Oracle White Paper.* (2012).
[24] Yifei Yang, Matt Youill, Matthew Woicik, Yizhou Liu, Xiangyao Yu, Marco Serafini, Ashraf Aboulnaga, and Michael Stonebraker. 2021. FlexPushdownDB: Hybrid Pushdown and Caching in a Cloud DBMS. *VLDB* 14, 11 (2021), 2101–2113.
[25] Yifei Yang, Xiangyao Yu, Marco Serafini, Ashraf Aboulnaga, and Michael Stonebraker. 2024. FlexpushdownDB: rethinking computation pushdown for cloud OLAP DBMSs. *VLDB Journal* 33, 5 (2024), 1643–1670.
[26] Bobbi Yogatama, Yifei Yang, Kevin Kristensen, Devesh Sarda, Abigale Kim, Adrian Cockcroft, Yu Teng, Joshua Patterson, Gregory Kimball, Wes McKinney, Weiwei Gong, and Xiangyao Yu. 2025. Rethinking Analytical Processing in the GPU Era. *arXiv preprint arXiv:2508.04701* (2025).
[27] Xiangyao Yu. 2022. Computation Pushdown across Layers in the Storage Hierarchy. https://www.sigarch.org/computation-pushdown-across-layers-in-the-storage-hierarchy. Accessed: 2025-08-07.
[28] Xiangyao Yu, Matt Youill, Matthew Woicik, Abdurrahman Ghanem, Marco Serafini, Ashraf Aboulnaga, and Michael Stonebraker. 2020. PushdownDB: Accelerating a DBMS using S3 Computation. In *ICDE*. 1802–1805.