Optimizing Analytical Query Processing on Disaggregated Hardware

Andreas Geyer

Supervised by: Wolfgang Lehner
Dresden University of Technology (TU Dresden), Dresden, 01069, Germany

Abstract
In a world of ever-growing amounts of data, hardware-scalability and energy-efficiency become more important with every year. Traditional scale-up and scale-out *database management systems (DBMS)* struggle to scale well with their growing analytical workloads. Due to this, the emerging technology of disaggregated hardware becomes more and more popular. However, there is no free ride and specific challenges arise. In my PhD topic, I want (i) to look into these challenges for analytical query workloads on disaggregated hardware and (ii) to provide appropriate solutions. First initial results concerning data movement are promising and show the potential of adapted solutions.

Keywords
RDMA, Disaggregated Memory, Disaggregated Hardware

1. Introduction

With the ongoing shift to a data-driven world in almost all application domains, the management and analytics of data gain importance. However, the demand for computing power as well as memory capacities is also growing to enable efficient data analytics over an ever-increasing amount of data. To satisfy these ever-growing hardware demands in a scalable and flexible way, the emerging technology of hardware disaggregation is considered the "next big thing" [1]. Hardware disaggregation is an approach that decomposes general-purpose monolithic servers into separated, network-attached resource pools, each of which can be built, managed, and scaled independently. This hardware disaggregation offers various valuable possibilities such as (i) fast, fine-grained scalability depending on individual workloads, (ii) energy proportionality, or (iii) resource sharing capabilities. However, there is no free ride and specific challenges arise. In my thesis, I want to focus on analytical query processing on disaggregated memory systems and solving the specific challenges in that scope.

The foundation of our work is an appropriate system architecture, similar to the one from [2], with dedicated units for computation and memory. These are named *Compute Units (CUs)* and *Memory Units (MUs)*. CUs and MUs are explicitly decomposed and connected via a network. With modern network technologies like *Compute Express Link (CXL)* or *Remote Direct Memory Access (RDMA)* over *InfiniBand (IB)*, there are already high-throughput, low-latency interconnects available. Nevertheless and as we focus on analytical query processing (OLAP) as a prime representative for data-intensive workloads, the efficient data exchange between MUs and CUs is a major challenge.

To minimize the amount of data to be transferred between CUs and MUs, solution approaches such as function-to-data (operator push-down) like in [3] or near-memory computing schemes [4] are heavily applied. However, these approaches lead to the fact that they cannot scale the computations, due to limited resources, as we would be able to achieve it by the general decomposition of CUs and MUs. Thus, they only provide limited applicability. To overcome that shortcoming, we focus on solutions regarding the data-to-function schema by making the data transfer explicit as first-class citizen in such an architecture. With this explicit treatment, we are able to synchronize the assignments of computations to CUs and the necessary data exchange in a scalable and flexible way. This synchronisation includes several aspects such as (i) different computations on the same data across different queries can be grouped at one CU, so that the necessary data must be transferred only to this CU, (ii) data transfer can be done in an asynchronous way to interleave it with the computation to hide latency, or (iii) preparing data during transfer for subsequent processing at the CU by e.g., adapting the data layout.

While still being in an early stage of my PhD thesis (1st year), we argue that this research direction has high potential. To show that, the remainder of this paper is organized as follows: In Section 2, we discuss a selection of already existing solutions. Then, we give a more detailed view of our current approach including some preliminary results in Section 3. Finally, we conclude with a summary and outlook in Section 4.

Published in the Workshop Proceedings of the EDBT/ICDT 2023 Joint Conference (March 28-March 31, 2023, Ioannina, Greece)
Andreas.Geyer@tu-dresden.de (A. Geyer)

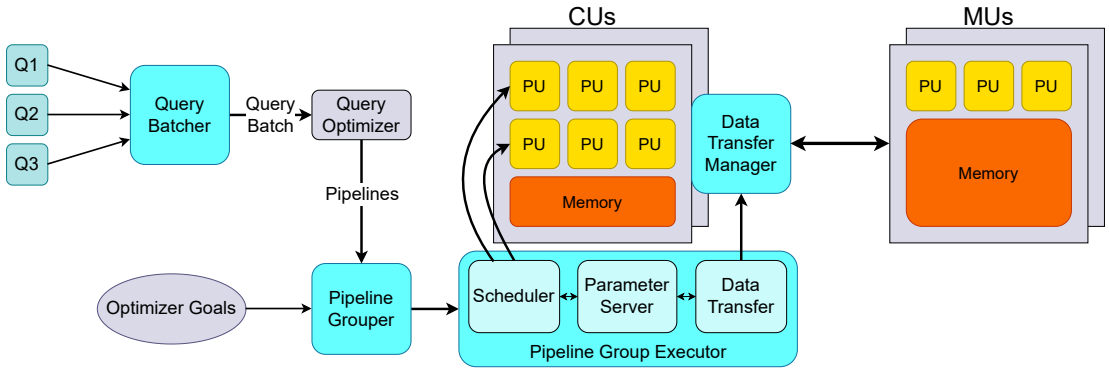


Figure 1: Envisioned design for a data transfer centric disaggregated DBMS.

2. Related Work

Disaggregated hardware revolutionizes the design and architecture of modern database systems and thus database researchers have just started to investigate the potential implications of such a novel hardware model. For example, [1, 3, 5, 6] discuss the general impact and among other things infer a new architecture as well as database primitives. We fully agree that disaggregation leads to an alteration of traditional query handling.

The authors of [2] introduce an approach for *distributed shared-memory databases (DSM-DB)*. Their system architecture is similar to ours as we will describe in Section 3.1, but it focuses on OLTP workloads, while we focus on OLAP. However, it will be interesting to compare this similar strategy to our own in the future.

There are already system prototypes like LegoOS [7], PolarDB [8], Teleport [6], Farview [3], and more emerge. LegoOS tackles the operating system side for steering and controlling the actual hardware components, which is an extremely interesting feature for elasticity, but orthogonal to our proposed processing model. PolarDB – very similar to our architectural blueprint – plans with separate compute nodes but attributes the remainder of the resources to individual pools. Contrarily, we argue that dedicated units with individual compute resources as in our system architecture yield benefits, for example, the preservation of the opportunity for operator push-down. In Teleport, the authors observe that the high network latency of 'remote' accesses is impacting data-intensive systems and thus opt for compute or operator push-down. Approaches with *network-attached memory (NAM)* [9, 10] are promising but lack the possibility of operator push-down, similar to Teleport. Farview's on-demand provisioning of compute nodes paired with the FPGA-controlled storage serves as a general inspiration for our work. However, Farview considers the execution of individual pipelines, which is contrary to our processing model, which is based on shared data access similar to the principle of *scan sharing* [11].

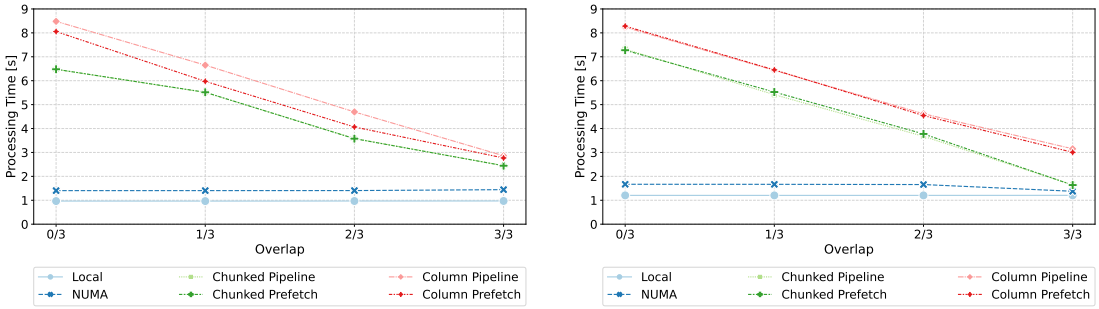
On the one hand, recent work also has just shown the viability of CXL-attached main memory [12]. Our prototype implementation is currently based on one-sided RDMA verbs, but our memory access layer is already prepared to also work with memory via CXL as soon as we have access to corresponding hardware. On the other hand, DFI [13] is a framework to efficiently exploit high-speed networks, such as IB. They show that adding an abstraction layer on top of RDMA verbs does not impose a significant performance degradation. However, their experiments are tailored towards tuple-based data processing, whereas we focus on column- or batch-oriented data transfer.

3. Our Contributions

The focus of my thesis lies on pipeline execution, as state-of-the-art execution model for analytical query processing (OLAP), introduced in [14], on disaggregated hardware. The challenges coming with it arise from the main OLAP properties. First, ① it is necessary to access a lot of data for these workloads, which traditionally results in scans of whole columns or even tables. Second, ② a lot of queries are executed simultaneously and most probably access the same data multiple times. Therefore, data transfer is a potential bottleneck and thus, an intelligent and optimized data transfer is crucial. The idea of making the data transfer explicit allows us to tune the pipeline execution in a way that the latency through network communication is nearly negligible. Additionally, we argue that this allows us to utilize the given flexibility that disaggregated hardware offers when it comes to resource management.

3.1. General System Architecture

As there is a wide variety of possibilities to structure a system based on disaggregated hardware, we start with



(a) 4 pipelines executed sequentially with 4 threads for each pipeline

(b) 4 pipelines executed fully parallel with 1 thread for each pipeline

Figure 2: Two different pipeline group execution strategies with different amounts of data-overlap

our anticipated system design as introduced in our CIDR 2023 publication [15]. Figure 1 depicts a related sketch. As already introduced, we separate our system into CUs and MUs based on their respective task. While MUs feature high memory capacities (DRAM, NVRAM, ...) with a limited amount of compute resources, CUs provide a high amount of compute resources with a limited main memory capacity. Thus, CUs are merely responsible for executing queries, managing the lifecycle of intermediates and feeding results to the clients. Apart from operator pushdown, base data has to be fetched from MUs.

This architecture works with one-to-one connections for CU and MU, as long as the MU holds all relevant data. However, the idea is to have an N-to-M connection between CUs and MUs. The connections are realised through a high-throughput, low-latency interconnect like IB or CXL, for instance.

As this architecture is already realizable with commodity systems, we argue that it allows transferring knowledge from the well-known system architectures to the new one based on disaggregated hardware. In the absence of real disaggregated hardware, we emulate both CU and MU with standard monolithic server systems directly connected through IB. As soon as disaggregated hardware is available for us, we will apply our proposed architecture and components to it.

3.2. Communication Layer

Even with fast interconnects, the network is prone to be the bottleneck of the whole architecture, especially in data heavy OLAP scenarios. Thus, we started our research by developing a flexible communication layer based on RDMA over IB, which is well prepared to incorporate CXL as soon as we get access to the respective hardware. Following the communication scheme of RDMA there are reserved buffers on each system. We implemented this by a separation into *Receive Buffer (RB)* and *Send Buffer (SB)*. As the name suggests, the SB is responsible for the sending process (one-sided IB verb RDMA_WRITE) and writes to the RB of the remote system. Since we make the buffers exclusive for each CU-to-MU connection, we can prevent conflicts of concurrent write processes to the same buffer.

Through extensive evaluation of different configurations of our initial communication layer, we found a multi-buffer approach as best fitting. This approach implies that there can be multiple SBs and RBs on each system. Thereby, we can hide the latency introduced by the consumption of the content from the RB and write continuously to other free buffers. With this approach, we found that already a configuration with one SB and two remote RBs has big benefits in comparison to a single-buffer approach. Using multiple threads on each node also improves the performance further.

This communication layer is continuously developed to further allow a multitude of other interconnect technologies additional to RDMA over IB.

3.3. Pipeline Group Concept

Based on the implementation of the communication layer, the pipeline-based processing model is re-evaluated on the given system architecture. As base tables are assumed to be in-memory on the MU, when answering the query, the data needs to be transferred from the MU to the CU. The naïve implementation based on the state-of-the-art pipeline-execution model shows that the processing time for a pipeline is mainly dominated by the data transfer. Following these results and tackling properties ① and ②, the main target is to reduce the amount of transferred data and interleave the communication with the computation. Thus, we propose an approach of grouping pipelines with similar or the same data-needs into pipeline-groups. This grouping allows us to prevent redundant data transfer.

Figure 2 depicts two experiments to highlight the benefits of our pipeline-group approach. Both graphs show

the processing time for the execution of 4 simple pipelines depending on the amount of overlap within their data-need. For these experiments *Local* means all data is located in-memory attached to the working CPU, *NUMA* that there is a NUMA-hop between the memory and the CPU, *Chunked* that the data is transferred over the network in smaller pieces and *Column* that whole data columns are transferred at once. In Figure 2a, the more traditional approach of executing each pipeline with full resources one after the other is displayed. Orthogonal to this, Figure 2b shows the fully parallel execution of the same four pipelines. It is visible, that a higher overlap in the data-need reduces the processing time of the four pipelines tremendously in both cases. However, with full parallelism, it is possible to nearly match the performance of the NUMA-curve, while the traditional approach does not perform that well. With these experiments, we argue that pipeline groups offer the potential to nearly eliminate the latency introduced by network communication. This work has been submitted to CIDR 2023 and was accepted for publication [15].

3.4. Resource Adaption

One of the key aspects of the pipeline group is also that it allows utilisation of the given flexibility of disaggregated hardware by scaling the resources individually with the workload. Several of the blue components of Figure 1 offer dimensions to exploit this flexibility. The shown *Pipeline Group Executor* is capable of managing the resource allocation in the best fitting way. Thus, it can for example distribute the workload across multiple CUs to prevent network-bottlenecks or move the workload to a CU closer to the data-holding MU. With different scheduling strategies, it is possible to determine how many resources are needed. Hence, for example, scenarios, where there is a budget involved can greatly benefit from our approach as resources (CPU cores, RAM, etc.), are allocated just when they are needed and released after the work is done. Additionally, when we integrate some form of operator push-down, we can react nearly immediately on the side of the MU to the increased workload. More CPU power can be allocated to work on this operator, without impact on the other connections of the MU. However, even though we already have a proof of concept for our pipeline group approach, most of these components are still up for development.

4. Conclusion and Future Work

We outlined the advantages and challenges of DBMS on disaggregated hardware, gave a brief overview of the existing solutions and touched on why we think they are not sufficient to solve the outlined challenges completely. Additionally, our described approach of pipeline group execution showed some results of our previous work, to prove that the concept is capable of solving the challenges of DBMS on disaggregated hardware as well as utilizing its flexibility and other advantages.

To research the topic of a DBMS processing model that makes the best use of the opportunities and finds solutions to the introduced challenges of disaggregated hardware will be one of the key topics of my PhD thesis. Thus, we will continue to develop our introduced pipeline group approach.

References

[1] Q. Zhang, et al., Rethinking data management systems for disaggregated data centers, in: CIDR, 2020.

[2] R. Wang, et al., The case for distributed shared-memory databases with rdma-enabled memory disaggregation, 2022.

[3] D. Korolija, et al., Farview: Disaggregated memory with operator off-loading for database engines, in: CIDR, 2022.

[4] G. Singh, et al., Near-memory computing: Past, present, and future, CoRR abs/1908.02640 (2019).

[5] Q. Zhang, et al., Understanding the effect of data center resource disaggregation on production dbmss, Proc. VLDB Endow. 13 (2020).

[6] Q. Zhang, et al., Optimizing data-intensive systems in disaggregated data centers with teleport, in: SIGMOD, 2022, p. 1345–1359.

[7] Y. Shan, et al., LegoOS: A disseminated, distributed OS for hardware resource disaggregation, in: USENIX ATC, 2019.

[8] W. Cao, et al., Polardb serverless: A cloud native database for disaggregated data centers, in: SIGMOD, 2021, pp. 2477–2489.

[9] C. Binnig, et al., The end of slow networks: It's time for a redesign, CoRR abs/1504.01048 (2015).

[10] E. Zamanian, et al., The end of a myth: Distributed transactions can scale, Proc. VLDB Endow. 10 (2017) 685–696.

[11] L. Qiao, et al., Main-memory scan sharing for multi-core cpus, PVLDB 1 (2008) 610–621.

[12] M. Ahn, et al., Enabling CXL memory expansion for in-memory database management systems, in: DaMoN, 2022, pp. 8:1–8:5.

[13] L. Thostrup, et al., DFI: the data flow interface for high-speed networks, in: SIGMOD, 2021, pp. 1825–1837.

[14] V. Leis, et al., Morsel-driven parallelism: a numa-aware query evaluation framework for the many-core age, in: SIGMOD, 2014, pp. 743–754.

[15] A. Geyer, et al., Pipeline group optimization on disaggregated systems, in: CIDR, 2023.