Using CXL to expand memory of IMDBMSs

Robert Kuntze
robert.kuntze@mailbox.tu-dresden.de
TU Dresden
Germany

ABSTRACT

General Summary

The increasing memory demand of modern applications has made overtaken the amount of memory monolithic systems can support efficiently, so distributed systems are on a rise. With novel remote memory solutions such as CXL it is possible to expand a systems memory by simply connecting more remote devices.

1/4 page

CCS CONCEPTS

• **Information systems** → *Database performance evaluation*; **Physical data models**; **Data exchange**.

KEYWORDS

Distributed System, Distributed Joins, CXL, Compute Express Link

ACM Reference Format:
Robert Kuntze. 2025. Using CXL to expand memory of IMDBMSs
Pro- und Hauptseminar Datenbanken

1 INTRODUCTION

Modern Applications are demanding more and more memory, be it Deep Learning [2] or In-Memory Database Management Systems [1], the amount of memory required is always increasing. Monolithic systems struggle at this, because their memory scalability is inherently limited and quite rigid. This results in memory expansion becoming very costly and inefficient at the required levels.
Distributed Systems allow for multiple nodes, each with their own CPU and memory, to work in parallel. This means more memory is available for the entire system, and more nodes, both for compute and memory, can be added later as needed. This approach shifts the bottleneck from the available memory, to the data transfer between nodes [10]. Any traditional network communication solution will result in high latency and low bandwidth, with increased complexity to handle proper memory coherency. A more novel approach is remote memory access with Remote Direct Memory Access (RDMA) or Compute Express Link (CXL). This allows any node to access the memory of a remote node, without any involvement of the host CPU.
RDMA uses interconnects like InfiniBand to achieve a high bandwidth and specialized Network Interface Cards (NICs) to bypass the network stack.

CXL is a newer open standard that has incorporated most of the already existing solutions, such as CCIX, Gen-Z and OpenCAPI. CXL enables high bandwidth, low latency and cache coherent memory access to headless nodes [7]. These headless nodes can be inserted into a system at any time, potentially making it easier and cheaper to increase available memory, even after deployment.
In this paper we evaluate the performance of possible CXL connections using different types of distributed joins comparing them to their RDMA counterpart.

2 RELATED WORK

There has been a lot of development in using remote memory expansion for distributed systems, showing the possibilities of remote memory technology, but also showing that its not yet close to intra-rack memory [4, 8]. There is already some research on CXL, both with real hardware and with emulation, that is showing promising first results [1, 2, 5].
[9] further shows that the emulation of CXL might perform worse than real hardware, so any performance gains seen with emulated CXL could be further improved with real hardware.
While CXL struggles with physical distances, [10] proposes a hybrid system that uses the advantages of both CXL and RDMA to improve against pure RDMA connections.

3 BACKGROUND

3.1 Remote memory access solutions

In this section we discuss the different solutions for the remote memory access in distributed systems.

3.1.1 Message Passing Interface (MPI). is a standardized communication protocol first developed in the 1990s. It was not originally developed with shared or distributed memory in mind, but most modern implementations (OpenMPI, MPICH, Intel-MPI) support it, and it has become the de facto standard. It supports both point to point and broadcasted communications.

3.1.2 Remote Direct Memory Access (RDMA). is a protocol for accessing remote memory without network or OS overhead, by using specialized network interface cards (NICs). After registering an area in memory with the NIC it can be accessed by remote nodes asynchronously without host CPU involvement. RDMA requires specialized hardware, for the connection itself like InfiniBand, RDMA over Converged Ethernet or iWarp. The biggest problem for RDMA is the missing cache coherent protocol, requiring an implementation of software-level coherency, increasing complexity immensely.

3.1.3 Compute Express Link (CXL). is an open standard, that specifies different device types, including NICs, accelarators and memory expansion devices, and their interconnect with a processor. CXL is

composed of three main protocols: CXL.io for initializing a connection over PCIe, CXL.cache for cache-coherent memory access of a device, and CXL.mem for accessing remote memory with load/store instructions over PCIe.
CXL 2.0 and 3.0 introduce switches, allowing for multiple devices to be controlled by a single host and expand, and expand the cache-coherency to allow for P2P communication between hosts. CXL 3.0 is based on PCIe 6.0, supporting a bandwidth of 121 GB/s. [6]
The CXL memory expansion devices function as headless NUMA nodes, which the Linux kernel will show as device entries. After creating a CXL region and configuring the device as a Direct Access (DAX) device, it can be accessed with an exposed path. Utilizing MMap allows for direct memory access on the remote node, even by multiple hosts at the same time (when using CXL 3.0+) [6].

3.2 Distributed Joins

For a distributed DBMS to efficiently use its compute power, it is important to partition the data, so multiple workers can work on the same join. There are different approaches to this, either pre-partiioning the data, or partitioning on the fly [3].
It is important that data is properly co-partitioned and co-located to decrease the need for slow data-transfer before a join. Hash-partitioning works, by using the join key as the partitioning key and range-partitioning works by using the same ranges as the join. This will ensure that all join partners are located in the same partitions. Co-location is harder to achieve, as any real world database will have too many tables to co-locate all frequently joined tables, without leaving out at least some pairs.
In the following we briefly discuss the different join types for data that is not yet co-partitioned, used in [6].

3.2.1 Repartition Hash Join (R-HJ). A R-HJ will repartition both sides on the join key to ensure the co-partitioning. After this the partitions can be processed independently by different nodes.
To co-locate the data, each partition is assigned a worker host and sent to the shared memory. With CXL shared memory, all data only needs to be written once to the proper CXL region and can then be read by multiple hosts.

3.2.2 Broadcast Hash Join (B-HJ). In a B-HJ the build side is broadcasted and each worker can build an independent hash table and probe it.
With CXL the data only needs to be written once to shared memory, instead of multiple times with a network based approach [6]. If the data is pre-partitioned across hosts, multiple hosts can start broadcasting at the same time to their assigned CXL area, without any synchronization overhead.

3.2.3 Shared Hash Table Hash Join (SHT-HJ). A SHT-HJ utilizes the shared memory, by allocating a global hash table in shared memory. After allocation a number of workers can work on building the table in parallel, by assigning each their own bucket-range. Working directly in shared memory means more granular and frequent access, increasing the affect latency has on performance. After the global table has been built, multiple workers can probe it, again each having their own bucket range [6]

4 CXL IMPLEMENTATION FOR INTER PROCESS COMMUNICATION

Technical Explanation of the IPC Implementation with CXL. This could also be combined with the CXL Background. No good title idea

- Three Modes of Communication tested
- polling - reader is busy waiting for the write pointer to be changed by the writer, so it knows when it can read more
- conditional - reader checks a condition in shared memory during its main loop, writer sets the condition after writing
- hybrid MPI/CXL - after writing to CXL Memory, the writer sends the CXL pointer over MPI, so the reader knows when and where to read
- spinlocks for synchronization
- differentiation between CXL-D (direct access) and CXL-C (copy to local)
- initial connection with MPI
- Data Access Table (DAT) hold offsets and sizes of data in CXL
- greater detail how join types are affected by the implementation

5 EVALUATION

Overview of the Results obtained in the original paper. Ideal point for some comparison with similar papers. Can I use graphics from the original paper? Focus on methods or results?

- papers communication only used PCIe 4.0, so lower bandwidth than max possible
- still higher than network bandwidth - room to grow still
- polling achieves the best latency, MPI is worse than all CXL modes
- CXLs higher bandwidth means better Performance at higher transfer amounts
- MPI does not suffer with latency as much as CXL (MPI seems to only slow down in two tests. is that weird?)
- CXL-C suffers less from higher latency, as it reduces duplicate transfers
- CXL-C beats CXL-D in R-HJ and BC-HJ
- CXL-D is better at SHT-HJ because of direct tuple access (because CXL-C has more overhead per operation from copying the data and its not used again?)
- in low latency BC-HJ and SHT-HJ outperform R-HJ
- TPC-H benchmark shows CXL can improve performance in Queries bottlenecked by data transfer - interesting what performance in other queries is like?

6 CONCLUSION

- CXL seems promising, improving both latency and bandwidth in distributed Applications
- cache-coherence is major advantage over RDMA
- CXL shared memory reduces network traffic volume
- CXL allows for great parallelism in distributed systems

1/2 page

REFERENCES

[1] Minseon Ahn, Andrew Chang, Donghun Lee, Jongmin Gim, Jungmin Kim, Jaemin Jung, Oliver Rebholz, Vincent Pham, Krishna Malladi, and Yang Seok Ki. 2022. Enabling CXL Memory Expansion for In-Memory Database Management Systems. In *Proceedings of the 18th International Workshop on Data Management on New Hardware* (Philadelphia, PA, USA) *(DaMoN '22)*. Association for Computing Machinery, New York, NY, USA, Article 8, 5 pages. https://doi.org/10.1145/3533737.3535090

[2] Moiz Arif, Kevin Assogba, M. Mustafa Rafique, and Sudharshan Vazhkudai. 2023. Exploiting CXL-based Memory for Distributed Deep Learning. In *Proceedings of the 51st International Conference on Parallel Processing* (Bordeaux, France) *(ICPP '22)*. Association for Computing Machinery, New York, NY, USA, Article 19, 11 pages. https://doi.org/10.1145/3545008.3545054

[3] Maximilian Bandle, Jana Giceva, and Thomas Neumann. 2021. To Partition, or Not to Partition, That is the Join Question in a Real System. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 168–180. https://doi.org/10.1145/3448016.3452831

[4] Claude Barthels, Simon Loesing, Gustavo Alonso, and Donald Kossmann. 2015. Rack-Scale In-Memory Join Processing using RDMA. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (Melbourne, Victoria, Australia) *(SIGMOD '15)*. Association for Computing Machinery, New York, NY, USA, 1463–1475. https://doi.org/10.1145/2723372.2750547

[5] Claude Barthels, Ingo Müller, Timo Schneider, Gustavo Alonso, and Torsten Hoefler. 2017. Distributed join algorithms on thousands of cores. *Proc. VLDB Endow.* 10, 5 (Jan. 2017), 517–528. https://doi.org/10.14778/3055540.3055545

[6] Alexander Baumstark, Marcus Paradies, Kai-Uwe Sattler, Steffen Kläbe, and Stephan Baumann. 2024. So Far and yet so Near - Accelerating Distributed Joins with CXL. In *Proceedings of the 20th International Workshop on Data Management on New Hardware* (Santiago, AA, Chile) *(DaMoN '24)*. Association for Computing Machinery, New York, NY, USA, Article 7, 9 pages. https://doi.org/10.1145/3662010.3663449

[7] CXL Consortium [n. d.]. https://computeexpresslink.org/cxl-specification-landing-page/ accessed 10.05.2025.

[8] Philipp Fent, Alexander van Renen, Andreas Kipf, Viktor Leis, Thomas Neumann, and Alfons Kemper. 2020. Low-Latency Communication for Fast DBMS Using RDMA and Shared Memory. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 1477–1488. https://doi.org/10.1109/ICDE48307.2020.00131

[9] Yan Sun, Yifan Yuan, Zeduo Yu, Reese Kuper, Chihun Song, Jinghan Huang, Houxiang Ji, Siddharth Agarwal, Jiaqi Lou, Ipoom Jeong, Ren Wang, Jung Ho Ahn, Tianyin Xu, and Nam Sung Kim. 2023. Demystifying CXL Memory with Genuine CXL-Ready Systems and Devices. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture* (Toronto, ON, Canada) *(MICRO '23)*. Association for Computing Machinery, New York, NY, USA, 105–121. https://doi.org/10.1145/3613424.3614256

[10] Zhonghua Wang, Yixing Guo, Kai Lu, Jiguang Wan, Daohui Wang, Ting Yao, and Huatao Wu. 2024. Rcmp: Reconstructing RDMA-Based Memory Disaggregation via CXL. *ACM Trans. Archit. Code Optim.* 21, 1, Article 15 (Jan. 2024), 26 pages. https://doi.org/10.1145/3634916