Pipeline Group Optimization on Disaggregated Systems

Andreas Geyer, Alexander Krause, Dirk Habich, Wolfgang Lehner

CIDR – January 10, 2023
State-of-the-Art Execution Model in DBMS

**SQL Queries**
- are transformed into pipeline-based query execution plans

**Pipeline Properties**
- each pipeline consists of multiple pipeline-friendly operators with a pipeline-breaking (sub-)operator at the end
- input data of a pipeline is partitioned into chunks, so that the chunks can be processed in parallel
- One pipeline after the other
Hardware Shifts to Disaggregation

**Traditional Scale-Up**
- Hard-wired setup
- Predictable latencies
- Elasticity
  - Very minimal on hardware level
  - Based on VM-level

**Disaggregated Hardware**
- Software composable system
- Altering hardware live
- Latency depending on physical distance
Pipelines on Disaggregated Hardware

State of the Art Approach

- Operator push-down
- Existing systems like Farview [1]
- Limited applicability due to limited compute power of Smart-NIC

Pipelines on Disaggregated Hardware

Our Approach

- Shipping data to compute
- Multiple queries may lead to redundant data transfer
- Limited Operator Push-Down possible

➢ Idea: similar to group commits [2] → grouped data access

---

[2] Hagmann; Reimplementing the Cedar File System Using Logging and Group Commit; 1987
Pipeline Groups
Building Pipeline Groups

- Batch and translate incoming queries
- Analyze resulting pipelines
- Group according to largest data overlap
- Schedule pipeline groups → transfer needed data once

### Building Pipeline Groups

1. **Pipeline Dependency Graph**
   - `P_4` → `P_3` → `P_2` → `P_1`

2. **Analyze**
   - Data Need

3. **Pipeline Groups**
   - `PG 1`
   - `PG 2`
   - `...`
   - `PG n`

4. **Pipeline Groups**
   - Compute Pool
   - Storage Pool
   - Memory Pool
Pipeline Execution on Disaggregated Hardware

- Query Batcher
- Query Optimizer
- Pipeline Grouper
- Pipeline Group Executor
- Compute Pool
  - PU
  - Memory
- Memory Pool
  - PU
  - Memory

Steps:
1. Query Batcher
2. Query Optimizer
3. Pipeline Grouper
4. Pipeline Group Executor
5. Compute Pool
6. Data Transfer Manager
7. Memory Pool
Proof of Concept
Experimental Setup

**RDMA simulated disaggregation**

- 2 monolithic servers connected via InfiniBand
- Mellanox ConnectX-4 (up to 12.5 GB/s)
- CN: 384GB Memory; 4 Intel Xeon Gold 6130
- MN: 384GB Memory; 4 Intel Xeon Gold 5130
**RDMA Benchmarks**

**Throughput Benchmark**
- Sending data from MN to CN without using it
  - Best possible performance for our RDMA implementation

**Consume Benchmark**
- Sending data from MN to CN with operator on CN
- More realistic than throughput
  - Close to throughput performance

**Take Away Message**
- Our RDMA implementation comes close to the theoretical hardware performance of up to 12.5 GB/s
  - Validation for evaluating pipeline group approach on this network implementation

Code available on GitHub: https://github.com/alexKrauseTUD/memoRDMA
Experimental Setup

RDMA simulated disaggregation
- 2 monolithic servers connected via InfiniBand
- Mellanox ConnectX-4 (up to 12.5 GB/s)
- CN: 384GB Memory; 4 Intel Xeon Gold 6130
- MN: 384GB Memory; 4 Intel Xeon Gold 5130

Data:
- Different columns, one column 1.5GB size
- Integer values between 0 and 100

Selectivity:
- Values for n: 1, 25, 50, 75, 100

Query Template
\[
\sum_{\text{col}2 \times \text{col}3} \sigma_{\text{col}1 < n} \\
\text{SELECT SUM(col2} \times \text{col3)} \\
\text{FROM data} \\
\text{WHERE col1 < n}
\]

Pipeline Groups
- Different queries of the same template
- Varying overlap of required columns
Pipeline Group Execution Benchmark

Heatmap
- Find best performing chunk and buffer sizes
- Showing time [s] for processing of pipeline
- Transfer asynchronous + interleaved with compute
  ➢ Both values with significant impact

Data Overlap
- 512KiB Buffer and 4MiB Chunk size
- 4 pipelines executed fully parallel
- Overlap → how many of the needed 3 columns are shared between all 4 pipelines

Sharing opportunities allow for efficient latency hiding.

Code available on GitHub: https://github.com/alexKrauseTUD/dataProvider
Future Work

1. Evaluate batching strategies
2. Test grouping strategies
3. Implement work and data placement and scheduling
4. Integrate additional technologies (CXL)
Pipeline Group Optimization on Disaggregated Systems

Andreas Geyer, Alexander Krause, Dirk Habich, Wolfgang Lehner

Firstname.Lastname@tu-dresden.de