Dynamic Sharing of Data and Work Across Queries

Stavros Harizopoulos,
Vladislav Shkapenyuk, and Anastassia Ailamaki

Thread-based execution in DBMS

- Queries are handled by a pool of threads
- Threads execute independently
- No means to exploit common operations
Query-centric design of DB engines

- Queries are evaluated independently
- Newer hardware allows higher concurrency
- More opportunities to share across queries

Need a new query engine philosophy

Exploit common
- data
- instructions
- work
across operators

Match SW-HW parallelism

… but **without** rewriting code
**QPipe: operator-centric engine**

- Conventional: “one-query, many-operators”
- **QPipe:** “one operator, many-queries”
- Relational operators become \( \mu \) Engines
- Queries break up in tasks and queue up

![Diagram showing conventional and QPipe execution models](image)

**Reusing data & work in QPipe**

- Detect overlap at run time
- Shared pages and intermediate results are simultaneously pipelined to parent nodes
- Up to 2x speedup with TPC-H queries

![Diagram showing overlap in red operator and simultaneous pipelining](image)
Talk outline

- Introduction
- QPipe: design and implementation
- Simultaneous Pipelining
- Experimental results
- Conclusions
A μEngine in detail

- tuple batching \(\Rightarrow\) I-cache
- query grouping \(\Rightarrow\) I&D-cache

Related work

- StagedDB
  - Break entire DBMS into stages CIDR 03
  - Reuse instructions in the cache VLDB 04

- Stream engines
  - NiagaraCQ Chen00 (SIGMOD)
  - TelegraphCQ Madden02 (SIGMOD)
    Chandrasekaran02 (VLDB)

- Parallel DBs DeWitt90 (TKDE)
  […]
Talk outline

- Introduction
- QPipe: design and implementation
- Simultaneous Pipelining
- Experimental results
- Conclusions

Sharing data & work across queries

**Query 1**: “Find average age of students enrolled in both class A and class B”

**Query 2**: work sharing opportunity

**Query 3**: min
Sharing opportunities at run time

- Q1 executes operator R
- Q2 arrives with R in its plan

**without SP**

- Q1 reads R
- Q2 writes R

**with SP**

- Q2 reads R
- Q1 writes R

result production for R in Q1

sharing potential

result production for R in Q2

Mechanisms for sharing

- Multi-query optimization **not used in practice**
- Materialized views **requires workload knowledge**
- Buffer pool management **opportunistic**
- Shared scans **limited use**
  - RedBrick, Teradata, SQL Server

QPipe complements above approaches
Talk outline

• Introduction
• QPipe: design and implementation
• Simultaneous Pipelining
• Experimental results
• Conclusions

The QPipe prototype

• Implemented on top of BerkeleyDB
• 7,000 lines of C++ code
• Shared-memory buffers
• Native OS threads
• Also tested on MySQL, Predator, Shore
Experimental setup

- **Platform**
  - 2GHz Pentium 4, 2GB RAM, 4 SCSI disks
- **Benchmarks**
  - TPC-H (4GB)
- **Systems compared**
  - Baseline: BerkeleyDB-based QPipe
  - QPipe with Simultaneous Pipelining
  - DBMS X: commercial

Sharing order-sensitive scans

- Two clients send query at different intervals
- QPipe performs 2 separate joins
Sharing order-sensitive scans

- Two clients send query at different intervals
- QPipe performs 2 separate joins

TPC-H workload

- Clients use pool of 8 TPC-H queries
- QPipe reuses large scans, runs up to 2x faster
- ..while maintaining low response times
Conclusions

• DB engines evaluate queries independently
• Limited existing mechanisms for sharing

• QPipe requires few code changes
• SP is simple yet powerful technique
• Allows dynamic sharing of data and work

• Other benefits (not evaluated in the paper)
  • Efficiently execute MQO plans
  • I-cache, D-cache performance

StagedDB: new DB architecture

[CIDR03, VLDB04, SIGMOD05]

Availability, scalability, easy maintenance, high performance at a low implementation cost
Thank you

www.cs.cmu.edu/~StagedDB

TPC-H workload

- Clients use pool of 8 TPC-H queries
- QPipe reuses large scans, runs up to 2x faster
- ..while maintaining low response times
Simultaneous Pipelining in QPipe

**without SP**

1. **read**
2. **write**
3. **COMPLETE**
4. **pipeline**

**with SP**

1. **attach**
2. **copy**
3. **COMPLETE**