Carnegie Mellon University

Lecture #14

ADVANCED DATABASE SYSTEMS

Networking

@Andy_Pavlo // 15-721 // Spring 2018

COURSE ANNOUNCEMENTS

Mid-Term: Wednesday March 7th @ 3:00pm

Project #2: Monday March 12th @ 11:59pm

Project #3 Proposal: Monday March 19th

TODAY'S AGENDA

Database Access APIs

Database Network Protocols

Kernel Bypass Methods

Project #3 Topics

DATABASE ACCESS

All of the demos in the class have been through a terminal client.

→ SQL queries are written by hand.

→ Results are printed to the terminal.

Real programs access a database through an API:

→ Direct Access (DBMS-specific)

→ Open Database Connectivity (ODBC)

→ Java Database Connectivity (JDBC)

OPEN DATABASE CONNECTIVITY

Standard API for accessing a DBMS. Design to be independent of the DBMS and OS.

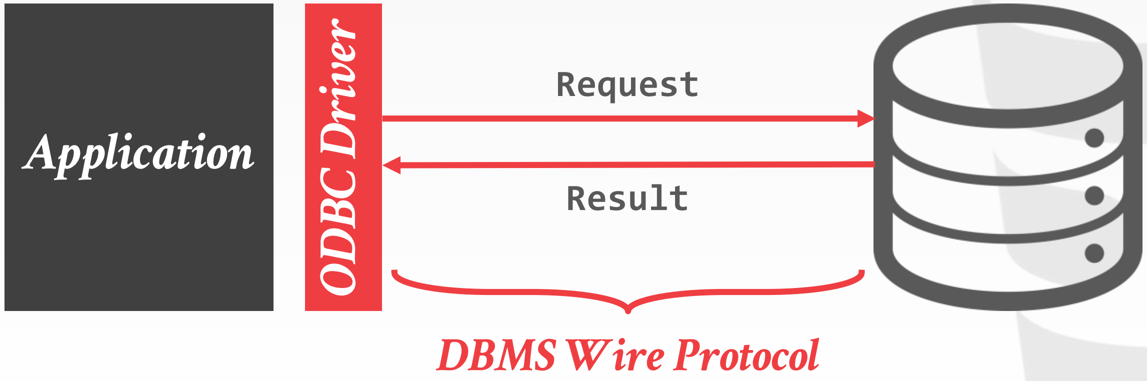
Originally developed in the early 1990s by Microsoft and Simba Technologies.

Every major relational DBMS now has an ODBC implementation.

OPEN DATABASE CONNECTIVITY

ODBC is based on the device driver model.

The **driver** encapsulates the logic needed to convert a standard set of commands into the DBMS-specific calls.



JAVA DATABASE CONNECTIVITY

Developed by Sun Microsystems in 1997 to provide a standard API for connecting a Java program with a DBMS.

JDBC can be considered a version of ODBC for the programming language Java instead of C.

JAVA DATABASE CONNECTIVITY

Approach #1: JDBC-ODBC Bridge
→ Convert JDBC method calls into ODBC function calls.

Approach #2: Native-API Driver
→ Convert JDBC method calls into native calls of the target DBMS API.

Approach #3: Network-Protocol Driver
→ Driver connects to a middleware that converts JDBC calls into a vendor-specific DBMS protocol.

Approach #4: Database-Protocol Driver
→ Pure Java implementation that converts JDBC calls directly into a vendor-specific DBMS protocol.

DATABASE NETWORKING PROTOCOLS

All major DBMSs implement their own proprietary wire protocol over TCP/IP.

A typical client/server interaction:

→ Client connects to DBMS and begins authentication process. There may be an SSL handshake.

→ Client then sends a query.

→ DBMS executes the query, then serializes the results and sends it back to the client.

EXISTING PROTOCOLS

Most newer systems implement one of the open-source DBMS wire protocols. This allows them to reuse the client drivers without having to develop and support them.

Just because on DBMS "speaks" another DBMS's wire protocol does not mean that it is compatible.
→ Need to also support catalogs, SQL dialect, and other functionality.

EXISTING PROTOCOLS

MySQL™

MEMSQL

Clustrix

actorDB

TiDB

PostgreSQL

amazon REDSHIFT

Greenplum

VERTICA

HyPer

Cockroach LABS

Peloton

HIVE

APACHE Spark™

PROTOCOL DESIGN SPACE

Row vs. Column Layout

Compression

Data Serialization

String Handling

DON'T HOLD MY DATA HOSTAGE: A CASE FOR CLIENT PROTOCOL REDESIGN
VLDB 2017

ROW VS. COLUMN LAYOUT

ODBC/JDBC are inherently row-oriented APIs.

→ The DBMS packages tuples into messages one tuple at a time.
→ The client has to deserialize data one tuple at a time.

But modern data analysis software operates on matrices and columns.

One potential solution is to send data in vectors.

COMPRESSION

Approach #1: Naïve Compression

Approach #2: Columnar-Specific Encoding

More heavyweight compression is better when the network is slow.

Better compression ratios for larger message chunk sizes.

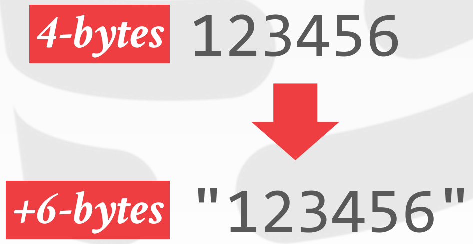
DATA SERIALIZATION

Approach #1: Binary Encoding

→ Have to handle endian conversion on client.
→ The closer the serialized format is to the DBMS's binary format, then the lower the overhead to serialize.
→ DBMS can implement its own format or rely on existing libraries (ProtoBuf, Thrift).

Approach #2: Text Encoding

→ Convert all binary values into strings.
→ Do not have to worry about endianness.



STRING HANDLING

Approach #1: Null Termination

→ Store a null byte (`'\0'`) to denote the end of a string.

→ Client has to scan the entire string to find end.

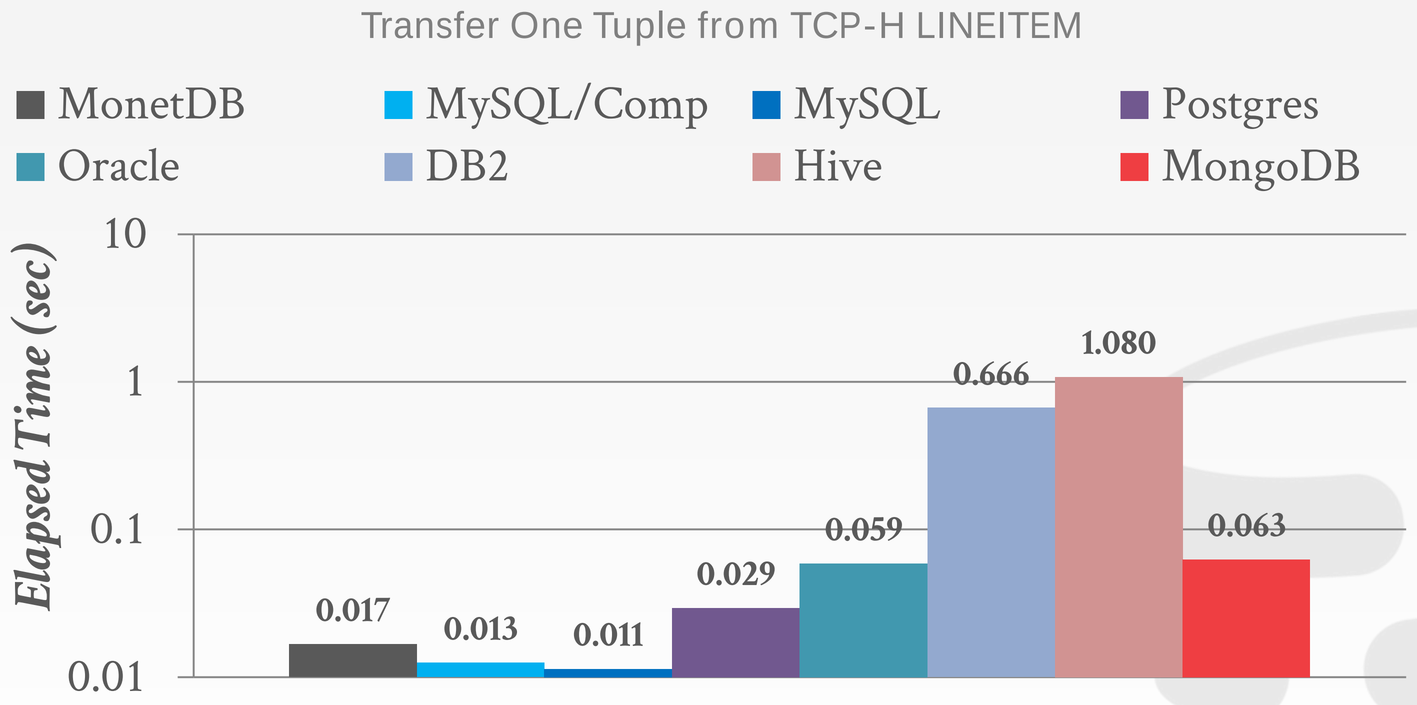
Approach #2: Length-Prefixes

→ Add the length of the string at the beginning of the bytes.

Approach #3: Fixed Width

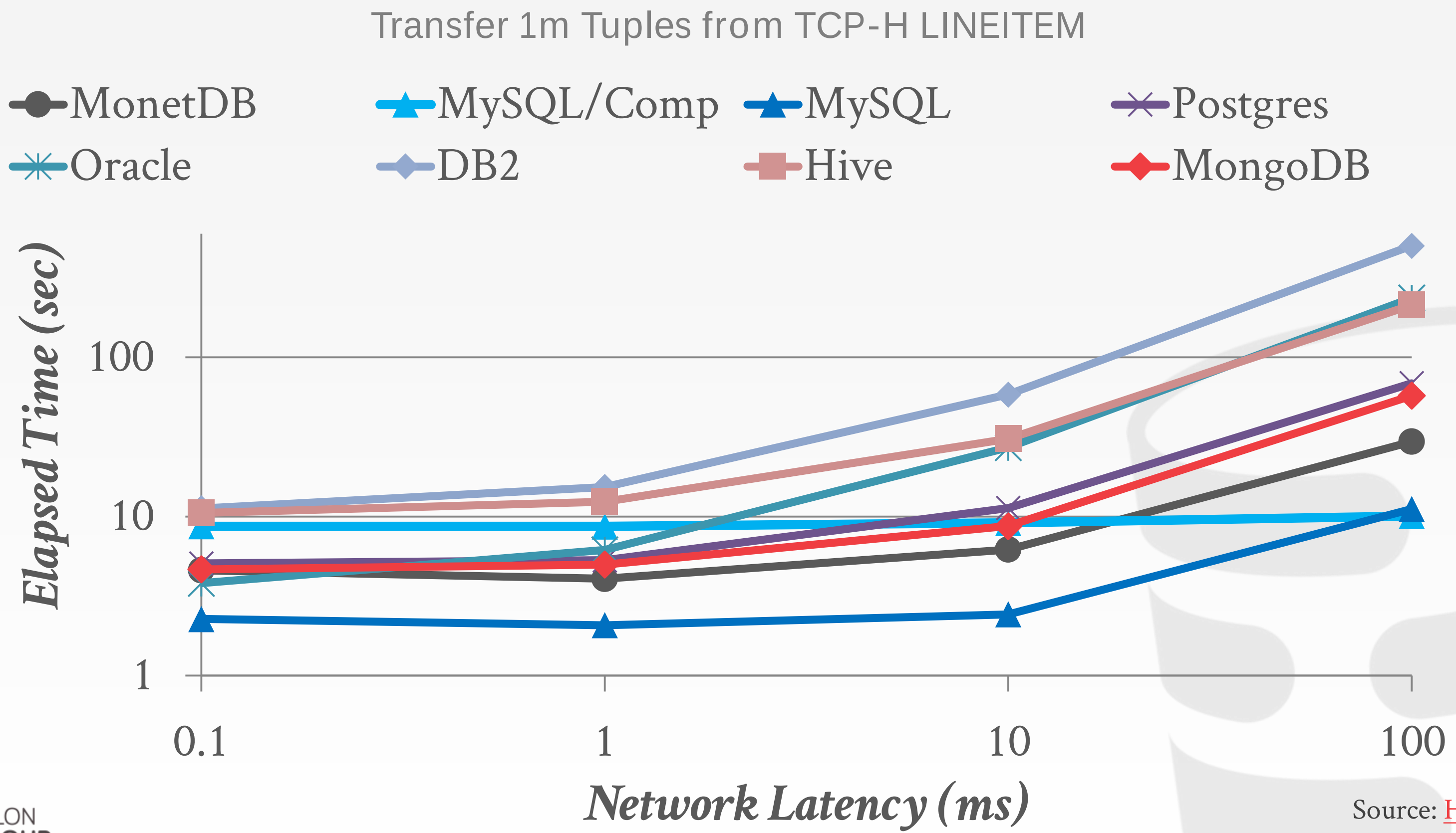
→ Pad every string to be the max size of that attribute.

NETWORK PROTOCOL PERFORMANCE



Source: Hannes Mühleisen

NETWORK PROTOCOL PERFORMANCE



Source: Hannes Mühleisen

OBSERVATION

The DBMS's network protocol implementation is not the only source of slowdown.

The OS's TCP/IP stack is slow…
→ Expensive context switches / interrupts
→ Data copying
→ Lots of latches in the kernel

KERNEL BYPASS METHODS

Allows the system to get data directly from the NIC into the DBMS address space.

→ No unnecessary data copying.

→ No OS TCP/IP stack.

Approach #1: Data Plane Development Kit

Approach #2: Remote Direct Memory Access

DATA PLANE DEVELOPMENT KIT

Set of libraries that allows programs to access NIC directly. Treat the NIC as a bare metal device.

Requires the DBMS code to do more to manage memory and buffers.
→ No data copying.
→ No system calls.

Example: ScyllaDB

REMOTE DIRECT MEMORY ACCESS

Read and write memory directly on a remote host without going through OS.

→ The client needs to know the correct address of the data that it wants to access.

→ The server is unaware that memory is being accessed remotely (i.e., no callbacks).

Example: Microsoft FaRM

PARTING THOUGHTS

A DBMS's networking protocol is an often overlooked bottleneck for performance.

Kernel bypass methods greatly improve performance but require more bookkeeping.
→ Probably more useful for internal DBMS communication.

PROJECT #3

Group project to implement some substantial component or feature in a DBMS.

Projects should incorporate topics discussed in this course as well as from your own interests.

Each group must pick a project that is unique from their classmates.

PROJECT #3

Project deliverables:
- → Proposal
- → Project Update
- → Code Review
- → Final Presentation
- → Code Drop

PROJECT #3 – PROPOSAL

Five minute presentation to the class that discusses the high-level topic.

Each proposal must discuss:
→ What files you will need to modify.
→ How you will test whether your implementation is correct.
→ What workloads you will use for your project.

PROJECT #3 – STATUS UPDATE

Five minute presentation to update the class about the current status of your project.

Each presentation should include:
→ Current development status.
→ Whether your plan has changed and why.
→ Anything that surprised you during coding.

PROJECT #3 – CODE REVIEW

Each group will be paired with another group and provide feedback on their code.

There will be two separate code review rounds.

Grading will be based on participation.

PROJECT #3 – FINAL PRESENTATION

<u>10</u> minute presentation on the final status of your project during the scheduled final exam.

You'll want to include any performance measurements or benchmarking numbers for your implementation.

Demos are always hot too…

PROJECT #3 – CODE DROP

A project is **not** considered complete until:

→ The code can merge into the master branch without any conflicts.

→ All comments from code review are addressed.

→ The project includes test cases that correctly verify that implementation is correct.

→ The group provides documentation in both the source code and in separate Markdown files.

We will select the merge order randomly.

PROJECT TOPICS

Query Optimizer
Schema Changes
Add/Drop Index
Index Storage (Cicada)
Sequences
Materialized Views
Pre-Compiled Queries

TileGroup Compaction
Multi-Threaded Queries
Database Compression
Temporary Tables
ENUM Type
Alternative Protocols

QUERY OPTIMIZER

Peloton has a sophisticated query optimizer based on the Cascades model.

Project: Expand features in Peloton's optimizer

→ Outer joins
→ Expression rewriting
→ Nested queries
→ Note: You have to send me your CV if you choose this project because companies want to hire you. Seriously.

SCHEMA CHANGES

We currently do not support any schema changes.

It would be nice if we did.

Project: Add support for **ALTER TABLE**.

→ Have to add support in SQL parser + planner.

→ Change Column Name

→ Add/Drop Column

→ Change Column Type

ADD/DROP INDEXES

Peloton does not build indexes in a transactionally consistent manner. It also does not refresh cached query plans when an index is added or removed.

Project: Correct index creation/deletion

→ Maintain a delta storage for capturing changes made to table while the index is being built.
→ Temporarily halt txns when the index is built and then apply missed changes.
→ Bonus: Support building indexes with multiple threads.

INDEX STORAGE

Recall that Cicada stores all index nodes as tuples in a table. This gives you concurrency for "free".

Project: Build an index that stores nodes directly inside of data tables.

→ We can get a basic B+tree from the Germans.
→ Need to add support for fixed-length binary attributes that are stored in directly in the tuple.
→ Will need to change the IndexFactory API to pass the database's storage manager.

SEQUENCES

Global counters that can be used as auto-increment keys for tables.

Project: Add support for Sequences

→ Store sequences in the catalog (follow Postgres v10).
→ Provide helper methods for efficient access.
→ Need to special case them from the DBMS's txn manager.
→ Add support for `nextval` native function.
→ Add support for `SERIAL` attribute type.
→ Will also want to integrate them with the new WAL component when it becomes available.

VIEWS

A materialized view is like a view that is updated whenever its underlying table is updated. Peloton already supports triggers (old engine).

Project: Implement support for incremental updates to materialized views in Peloton.
→ Will need to leverage Postgres' catalog infrastructure and then populate new data structures.

PRE-COMPILED QUERIES

Our LLVM engine compiles each query the first time that it is executed. Some queries can be pre-compiled when the table is created.

Project: Add support for pre-compilation
→ Basic **SELECT**, **UPDATE**, and **DELETE** queries.
→ Support all catalog access methods too.
→ Look into partial compilation and runtime stitching.

TILEGROUP COMPACTION

Peloton currently never frees memory. If you delete all of the tuples in the table, then it should free up the memory.

Project: Add support for TileGroup Compaction
→ Easy: Free a TileGroup if it is empty and there are others.
→ Harder: Combine two less than half full TileGroups into a single TileGroup.
→ Will need to add a background compaction thread.

MULTI-THREADED QUERIES

Peloton currently only uses a single worker thread per txn/query.

Project: Implement support for intra-query parallelism with multiple threads.

→ Will need to implement this to work with the new LLVM execution engine.
→ **Bonus:** Add support for NUMA-aware data placement. Will need to update internal catalog.

DATABASE COMPRESSION

Compression enables the DBMS to use less space to store data and potentially process less data per query.

Project: Implement different compression schemes for table storage.

→ Delta encoding, Dictionaries
→ Will need to implement new query operators that can operate directly on this data.
→ **Bonus:** Implement the ability to automatically determine what scheme to use per tile.

TEMPORARY TABLES

The client can create ephemeral tables that are blown away when it disconnects.

Project: Add support for temporary tables.
→ Maintain table for the duration of a session.
→ Should support all operations just like a real table.
→ Need to modify the catalog, binder, and planner to support them.

ENUM TYPE

The ENUM type allows the programmer to map names to values and restrict the domain.

Project: Add support for ENUM type
→ Need to update catalogs to store ENUM information.
→ Need to modify binder to enforce the ENUM constraint and then map entries to integers.
→ Need to support ENUM type in LLVM expression evaluation.

ALTERNATIVE PROTOCOLS

Add support for different ways to connect to Peloton and ingest/query data.

Examples: Kafka, Memcached

Project: Implement these APIs directly inside of Peloton and enable it to read/write directly to in-memory data.

→ Need to overhaul the client connection handling code.
→ GET/PUT can be implemented as a single-query txn with prepared statements.

TESTING

We plan to expand Peloton's SQL-based regression test suite to check that your project does not break high-level functionalities.

Every group is going to need to implement their own unit tests for their code.

COMPUTING RESOURCES

Use the same machines as your other projects.
→ Dual-socket Xeon E5-2620 (6 cores / 12 threads)
→ 128 GB DDR4

Let me know if you think you need special hardware.

OLTP-BENCH

We already have a full-featured benchmarking framework that you can use for your projects.

It includes 15 ready to execute workloads
→ OLTP: TPC-C, TATP, YCSB, Wikipedia
→ OLAP: CH-Benchmark, TPC-H

http://oltpbenchmark.com/

PROJECT #3 PROPOSALS

Each group will make a **5** minute presentation about their project topic proposal to the class on **Tuesday March 21st**.

I am able during Spring Break for additional discussion and clarification of the project idea.

NEXT CLASS

Mid-Term Exam!!!

After Spring Break:
Project #3 Proposals