ADMINISTRIVIA

Project #2:
→ Feedback Submission: **Saturday April 1**\textsuperscript{st}
→ Final Submission: **Monday May 1**\textsuperscript{st}
→ I sent out dbdb.io signup links on Monday!

Project #3
→ Status Update Presentation: **Monday April 3**\textsuperscript{rd}
→ Final Presentations: **Friday May 5th @ 5:30pm**
ARCHITECTURE OVERVIEW

SQL Query

Networking Layer

Planner

Compiler

Execution Engine

Storage Manager

- SQL Parser
- Binder
- Rewriter
- Optimizer / Cost Models

- Scheduling / Placement
- Operator Execution
- Indexes

- Storage Models
- Compression
TODAY'S AGENDA

Database Access APIs
Database Network Protocols
Kernel/User Bypass Methods
DATABASE ACCESS

All 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)
DATABASE ACCESS

All 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)
DATABASE

All 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)
DATABASE ACCESS

All 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. Designed 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.

JDBC supports different client-side configurations because there may not be a native Java driver for each DBMS.
JAVA DATABASE CONNECTIVITY

Approach #1: JDBC-ODBC Bridge _removed in 2014_  
→ 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 in a separate process that converts JDBC calls into a vendor-specific DBMS protocol.

Approach #4: Database-Protocol Driver _best approach_  
→ Pure Java implementation that converts JDBC calls directly into a vendor-specific DBMS protocol.
All major DBMSs implement their own proprietary client wire protocol over TCP/IP.

→ Use **Unix domain sockets** if running on same box as app.
→ Andy doesn't know of any DBMS using UDP for clients.

A typical client/server interaction:

→ Client connects to DBMS and begins authentication process. There may be an SSL/TLS 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 one DBMS "speaks" another DBMS's wire protocol does not mean that it is compatible. → Need to also support catalogs, SQL dialect, and other functionality.
PROTOCOL DESIGN SPACE

Row vs. Column Layout
Compression
Data Serialization
String Handling
ROW VS. COLUMN LAYOUT

ODBC/JDBC are row-oriented APIs.
→ Server packages tuples into messages one tuple at a time.
→ Client deserializes data one tuple at a time.

But switching to a column-oriented API is a bad too because client may access multiple columns for a tuple.

Solution: Vector-oriented API
**COMPRESSION**

**Approach #1: Naïve Compression**
- DBMS applies a general-purpose compression algo (lz4, gzip, zstd) on message chunks before transmitting.
- Few systems support this (Oracle, MySQL).

**Approach #2: Columnar-Specific Encoding**
- Analyze results and choose a specific compression encoding (dictionary, RLE, delta) per column.
- No system implements this.

Heavyweight compression is better when network is slow. DBMS achieves better compression ratios for larger message chunk sizes.
DATA SERIALIZATION

Approach #1: Binary Encoding

→ Client handles endian conversion.
→ 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 (ProtoBuffers, Thrift, FlatBuffers).
DATA SERIALIZATION

Approach #1: Binary Encoding

→ Client handles endian conversion.
→ 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 (ProtoBuffers, Thrift, FlatBuffers).
DATA SERIALIZATION

Approach #1: Binary Encoding
→ Client handles endian conversion.
→ 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 (ProtoBuffers, Thrift, FlatBuffers).
DATA SERIALIZATION

Approach #1: Binary Encoding

→ Client handles endian conversion.
→ 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 (ProtoBuffers, Thrift, FlatBuffers).

ProfaneDB
DATA SERIALIZATION

Approach #1: Binary Encoding
→ Client handles endian conversion.
→ 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 (ProtoBuffers, Thrift, FlatBuffers).

Approach #2: Text Encoding
→ Convert all binary values into strings (atoi).
→ Do not have to worry about endianness.
Approach #1: Binary Encoding
→ Client handles endian conversion.
→ 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 (ProtoBuffers, Thrift, FlatBuffers).

Approach #2: Text Encoding
→ Convert all binary values into strings (atoi).
→ 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 scans 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**

*Transfer One Tuple from TCP-H LINEITEM*

<table>
<thead>
<tr>
<th>Protocol</th>
<th>MySQL+GZIP</th>
<th>MySQL</th>
<th>MonetDB</th>
<th>Postgres</th>
<th>Oracle</th>
<th>MongoDB</th>
<th>DB2</th>
<th>Hive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elapsed Time (sec)</td>
<td>0.013</td>
<td>0.011</td>
<td>0.017</td>
<td>0.059</td>
<td>0.063</td>
<td>0.666</td>
<td>1.080</td>
<td></td>
</tr>
</tbody>
</table>

*Text Encoding*

*Lower is Better*

*All Other Protocols Use Binary Encoding*

Source: Hannes Mühleisen
NETWORK PROTOCOL PERFORMANCE

Transfer 1m Tuples from TCP-H LINEITEM

Source: Hannes Mühleisen

Lower is Better

Compression overhead is bad tradeoff when network is fast.
Network Protocol Performance

Transfer 1m Tuples from TCP-H LINEITEM

Elapsed Time (sec)

Lower is Better

Network Latency (ms)

MySQL+GZIP  MySQL  MonetDB  Postgres  Oracle  MongoDB  DB2  Hive

Lower is Better

Verbose protocol overhead is more pronounced on slower network.

Source: Hannes Mühleisen

CMU-DB 15-721 (Spring 2023)
Standardized column-oriented format (PAX) memory representation of tables.
→ Think of it like Parquet/ORC but for in-memory data.
→ Initial Java implementation from Apache Drill.

Allows systems to exchange data without having to (de)serialize into proprietary formats.

Arrow project includes components around format:
→ Wire Protocols (ADBC, Arrow Flight)
→ Execution Engine (DataFusion)
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
Approach #3: io_uring
DATA PLANE DEVELOPMENT KIT (DPDK)

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 network stack (layers 3+4), memory, and buffers.

→ TCP/IP in usercode (e.g., F-Stack).
→ No data copying.
→ No system calls.

Example: ScyllaDB's Seastar, Yellowbrick's ybRPC
DATA PLANE DEVELOPMENT KIT (DPDK)

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

Requires the DBMS code to do more to manage network stack (layers 3+),

→ TCP/IP in usercode (e.g., F-Stack).
→ No data copying.
→ No system calls.

Example: ScyllaDB's Seastar, Yellowbrick's ybRPC
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: Oracle RAC, Microsoft FaRM
IOURING

Linux system call interface for zero-copy asynchronous I/O operations.
→ Originally added in 2019 for accessing storage devices.
→ Expanded in 2022 to support network devices.
→ Windows has something similar called ICOP.

OS exposes two circular buffers (queues) to store submission and completion I/O requests.
→ DBMS submits requests for the kernel to perform read/write operations to DBMS-provided buffers.
→ When OS completes request, it puts the event on the competition queue and invokes callback.
Introduction

As an open source time series database company, we are constantly looking for ways to improve the performance of our systems. It is also important to us that our work is transparent and reproducible. In this blog post, we want to share our thoughts on a recent enhancement we made to our system and discuss our experience with it.

Our customers are using our product to process large amounts of time series data. As the size of their datasets grows, so does the importance of efficient data processing. We want to be able to support their needs, so we decided to dedicate our new release, QuestDB 3.2, to improving the way we handle time series data. We want to tell you about the latest improvement to our system, which is a new feature called IO_URING.

IO_URING

IO_URING is a Linux system call interface for zero-copy asynchronous I/O operations. It was originally added in 2019 for accessing storage devices. In 2022, it was expanded to support network devices.

Windows has something similar called ICOP.

OS exposes two circular buffers (queues) to store submission and completion I/O requests. DBMS submits requests for the kernel to perform read/write operations to DBMS-provided buffers. When OS completes request, it puts the event on the completion queue and invokes callback.

DBMS

DBMS is a software system that manages access to the data. It is responsible for storing, retrieving, and managing the data. In our case, we use ClickHouse as our DBMS.

ClickHouse

ClickHouse is a high-performance column-oriented database management system. It is designed to store, analyze, and process large amounts of data. It is written in C++, which makes it fast and efficient.

A Programmers' Abstraction

A Programmers' Abstraction: the kqueue

Consider this tale of I/O and perfection. Things like auring and kqueue, and take home that perfection may be familiar.

This is a twist on King's talk at Sorcery.

Classical approach

When you want to read from a file, you have to decide how many times as necessary to fill a buffer, and thus, what direction you follow.

Linux

Linux is an open-source operating system. It is used by a wide range of devices, from desktop computers to servers to mobile devices.

Windows

Windows is a family of several computer operating systems, primarily developed, marketed, and sold by Microsoft.

A journey to io_uring, AIO and modern storage devices

While main memory is considered to be slower than some systems designers it is not an option, especially when it comes to complex operations. When data is stored in external storage, it is often necessary to access the data. However, there are several kinds of storage technologies available, such as RAID, SSD, and hard drives. In all cases, the system call to read from them: `read`.

The system calls perform for available devices. In RAID, SATA, SSD, or other storage technologies, the read requests are fulfilled by retrieving the requested data from the storage device. Low latency, high bandwidth, and low power consumption are the main characteristics of modern storage devices.

Conclusion

In conclusion, we believe that IO_URING is a powerful tool for improving the performance of time series databases. We are excited to see how it will be used by our customers and look forward to continuing to improve our product.
IO_URING

Linux system call interface for zero-copy asynchronous I/O operations.
→ Originally added in 2019 for accessing storage devices.
→ Expanded in 2022 to support network devices.
→ Windows has something similar called ICOP.

OS exposes two circular buffers (queues) to store submission and completion I/O requests.
→ DBMS submits requests for the kernel to perform read/write operations to DBMS-provided buffers.
→ When OS completes request, it puts the event on the competition queue and invokes callback.
DATA EXPORT PERFORMANCE
Transfer 7GB of Tuples from TPC-C ORDER_LINE

Throughput (MB/sec)

Transport Method

Higher is Better

Postgres  Vectorized Postgres  Arrow Flight  RDMA

0  38  150  891  1057

1500  800  400  0

↑Higher is Better
USER BYPASS METHODS

Execute logic inside of the OS kernel when packets arrive instead of copying it into the DBMS via extended-Berkeley Packet Filters (eBPF).

→ eBPF programs are written in a DSL and then compiled into bytecode, verified, then JIT-ed at runtime.

→ Programming model is limited (no malloc, # of instrs.).

Only useful for parts of the DBMS that operate on I/Os that the system does not retain for long periods of time.
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.

User bypass is an interesting direction for ephemeral I/Os in DBMSs.
Query Optimizer