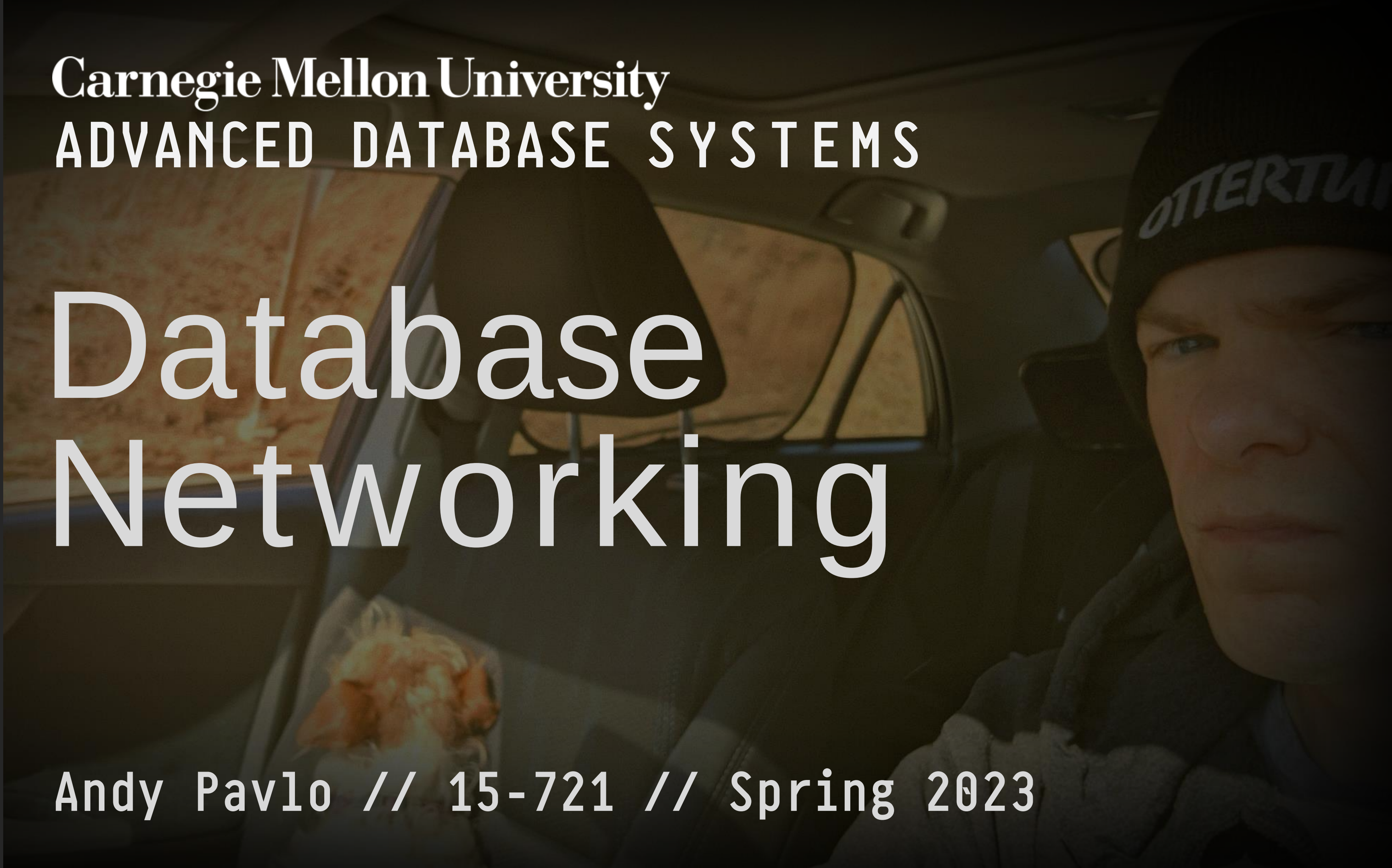
Lecture #15

Carnegie Mellon University

ADVANCED DATABASE SYSTEMS

Database Networking



Andy Pavlo // 15-721 // Spring 2023

ADMINISTRIVIA

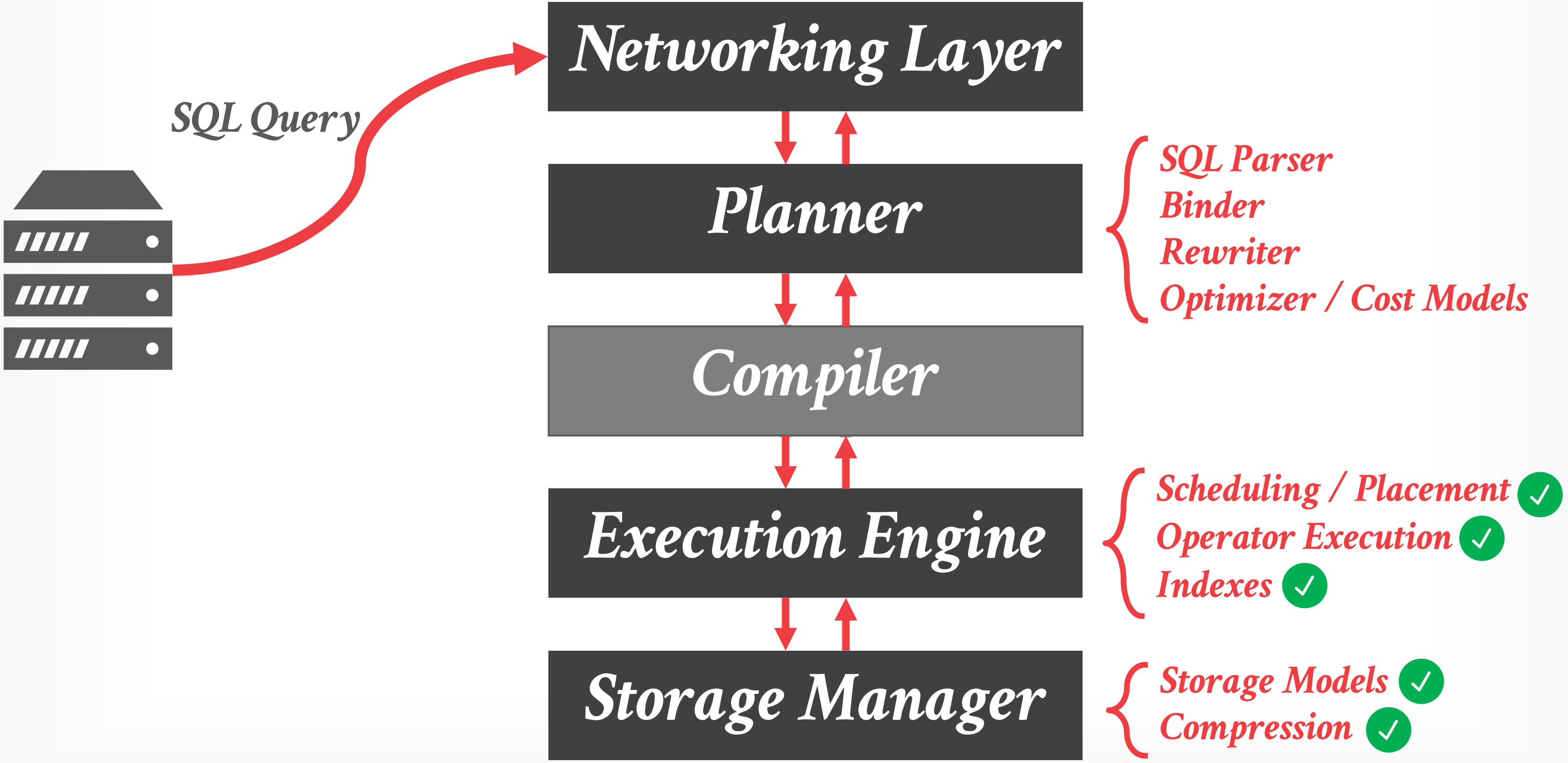
Project #2:

→ Feedback Submission: **Saturday April 1st**
→ Final Submission: **Monday May 1st**
→ I sent out dbdb.io signup links on Monday!

Project #3

→ Status Update Presentation: **Monday April 3rd**
→ Final Presentations: **Friday May 5th @ 5:30pm**

ARCHITECTURE OVERVIEW



TODAY'S AGENDA

Database Access APIs
Database Network Protocols
Kernel/User Bypass Methods

DATABASE ACCESS

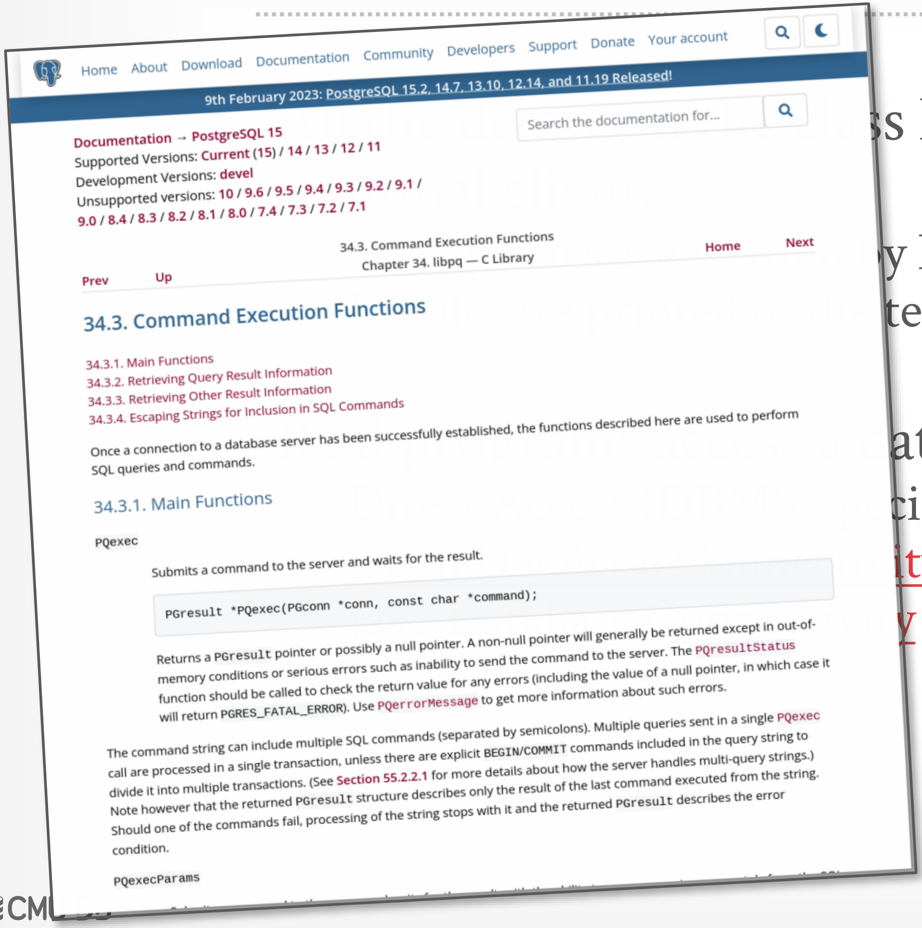
All the demos in the class have been through a terminal client.

→ 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


Home About Download Documentation Community Developers Support Donate Your account

9th February 2023: PostgreSQL 15.2, 14.7, 13.10, 12.14, and 11.19 Released!

Search the documentation for...

Documentation → PostgreSQL 15
Supported Versions: Current (15) / 14 / 13 / 12 / 11
Development Versions: devel
Unsupported versions: 10 / 9.6 / 9.5 / 9.4 / 9.3 / 9.2 / 9.1 / 9.0 / 8.4 / 8.3 / 8.2 / 8.1 / 8.0 / 7.4 / 7.3 / 7.2 / 7.1

34.3. Command Execution Functions
Prev Up Chapter 34. libpq — C Library Home Next

34.3. Command Execution Functions

34.3.1. Main Functions
34.3.2. Retrieving Query Result Information
34.3.3. Retrieving Other Result Information
34.3.4. Escaping Strings for Inclusion in SQL Commands

Once a connection to a database server has been successfully established, the functions described here are used to perform SQL queries and commands.

34.3.1. Main Functions

PQexec

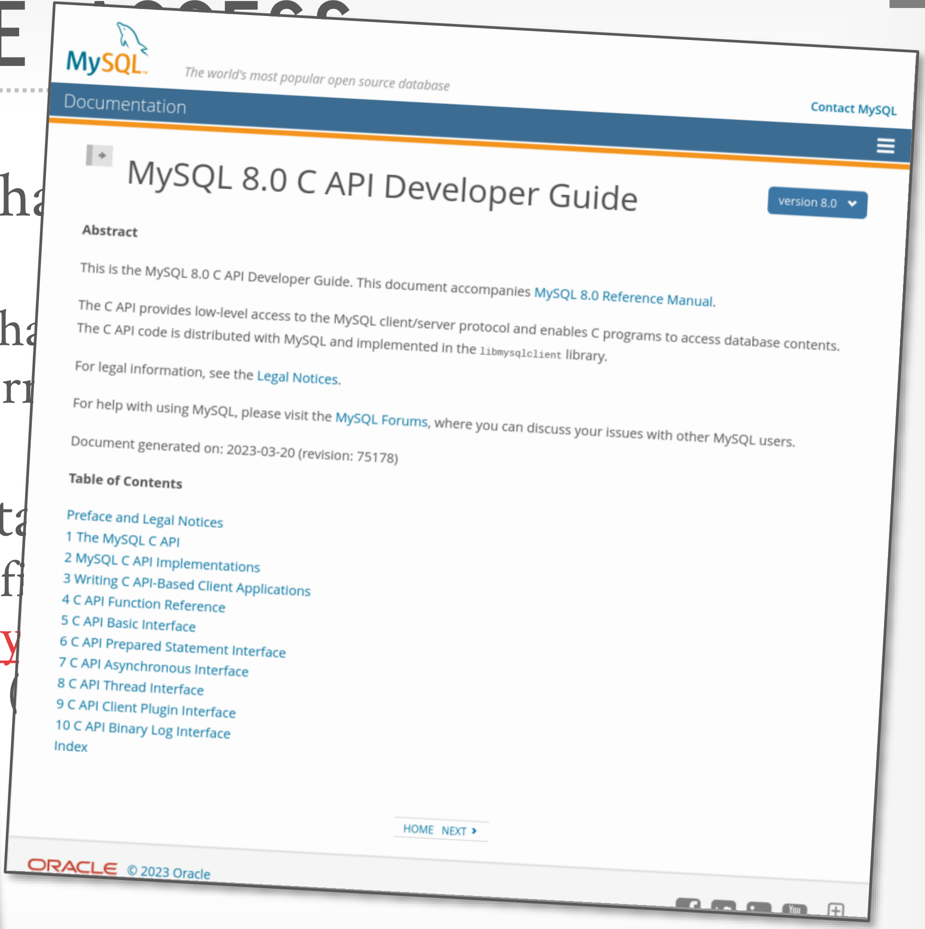
Submits a command to the server and waits for the result.

```
PGresult *PQexec(PGconn *conn, const char *command);
```

Returns a PGresult pointer or possibly a null pointer. A non-null pointer will generally be returned except in out-of-memory conditions or serious errors such as inability to send the command to the server. The PQresultStatus function should be called to check the return value for any errors (including the value of a null pointer, in which case it will return PGRES_FATAL_ERROR). Use PQerrorMessage to get more information about such errors.

The command string can include multiple SQL commands (separated by semicolons). Multiple queries sent in a single PQexec call are processed in a single transaction, unless there are explicit BEGIN/COMMIT commands included in the query string to divide it into multiple transactions. (See Section 55.2.2.1 for more details about how the server handles multi-query strings.) Note however that the returned PGresult structure describes only the result of the last command executed from the string. Should one of the commands fail, processing of the string stops with it and the returned PGresult describes the error condition.

PQexecParams


MySQL

The world's most popular open source database

Documentation Contact MySQL

MySQL 8.0 C API Developer Guide

version 8.0

Abstract

This is the MySQL 8.0 C API Developer Guide. This document accompanies MySQL 8.0 Reference Manual.

The C API provides low-level access to the MySQL client/server protocol and enables C programs to access database contents. The C API code is distributed with MySQL and implemented in the libmysqlclient library.

For legal information, see the Legal Notices.

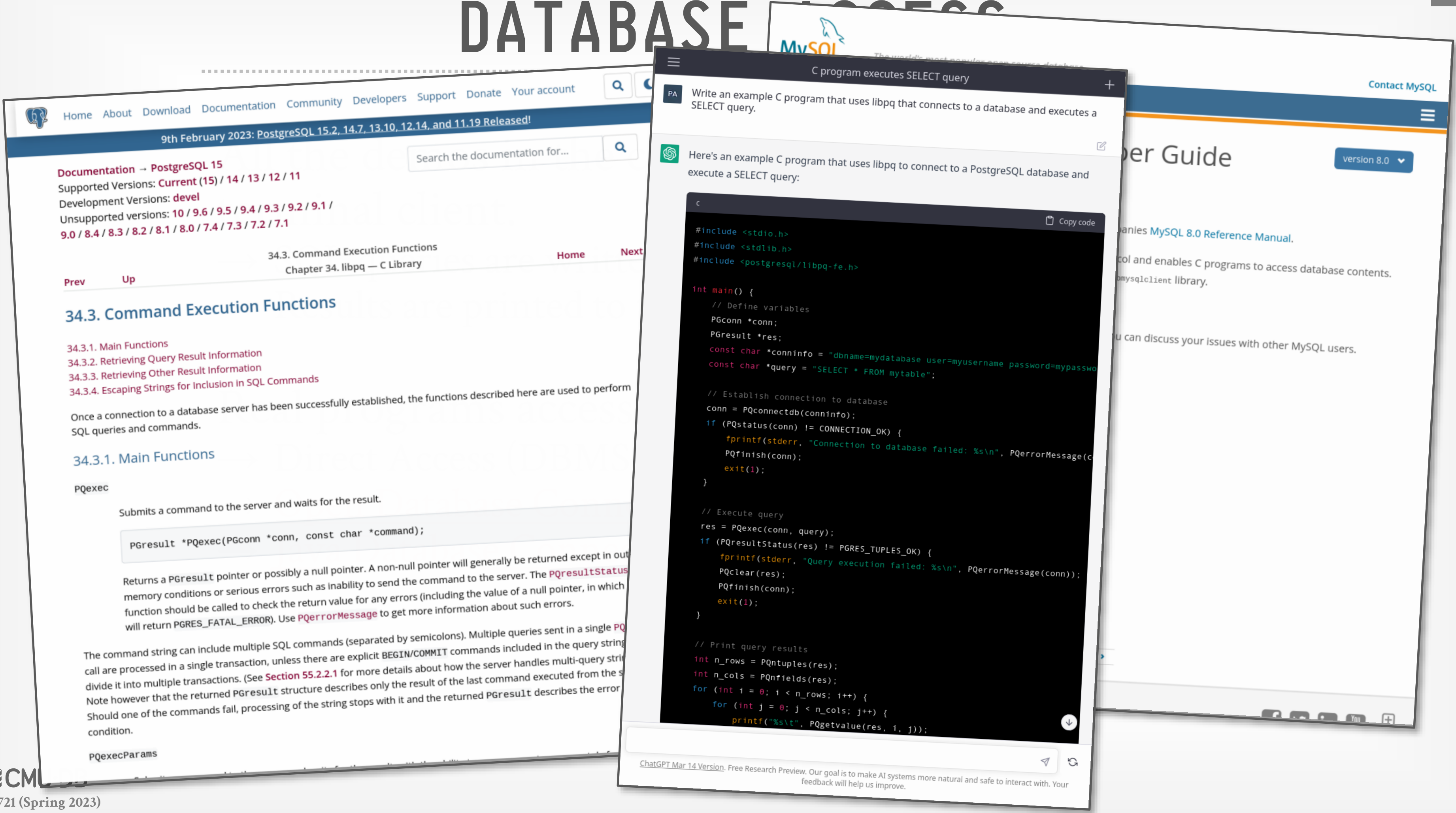
For help with using MySQL, please visit the MySQL Forums, where you can discuss your issues with other MySQL users.

Document generated on: 2023-03-20 (revision: 75178)

Table of Contents

HOME NEXT ›

DATABASE ACCESS



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.

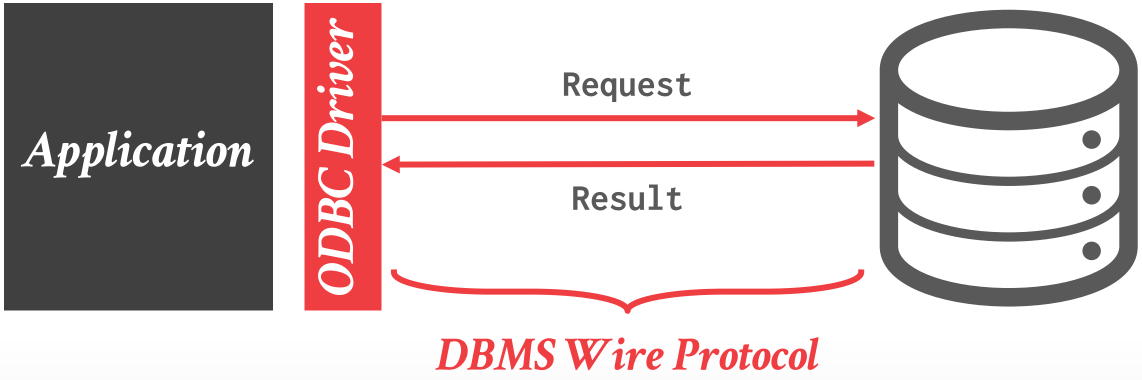
Originally developed in the early 1990s by Microsoft and Simba Technologies.

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.

DATABASE NETWORKING PROTOCOLS

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 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

- SingleStore
- Clustrix
- actorDB
- TiDB
- dolt
- CLEARDB
- STARDOG
- Amazon Aurora

PostgreSQL

- amazon REDSHIFT
- Greenplum
- HyPer
- VERTICA
- Cockroach LABS
- yugabyteDB
- UMBRA
- CrateDB
- Materialize
- Amazon Aurora
- Yellowbrick
- QuestDB

redis

- APACHE GEODE
- curiodb
- rosedb
- Dragonfly
- Kvrocks
- Tendis
- KeyDB
- SUMMIT|DB

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

```
String sql = "SELECT * FROM xxx";
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(sql);
while (rs.next()) {
  // Do something magical row by row!
  rs.getInt(1);
  rs.getString(2);
  rs.getDate(3);
}
stmt.close();
```

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

```
String sql = "SELECT * FROM xxx";
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(sql);
while (rs.nextCol()) {
  while (rs.nextRow()) {
    // Do something magical per column!
    rs.getValue();
  }
}
stmt.close();
```

Not Real JDBC Code!

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.

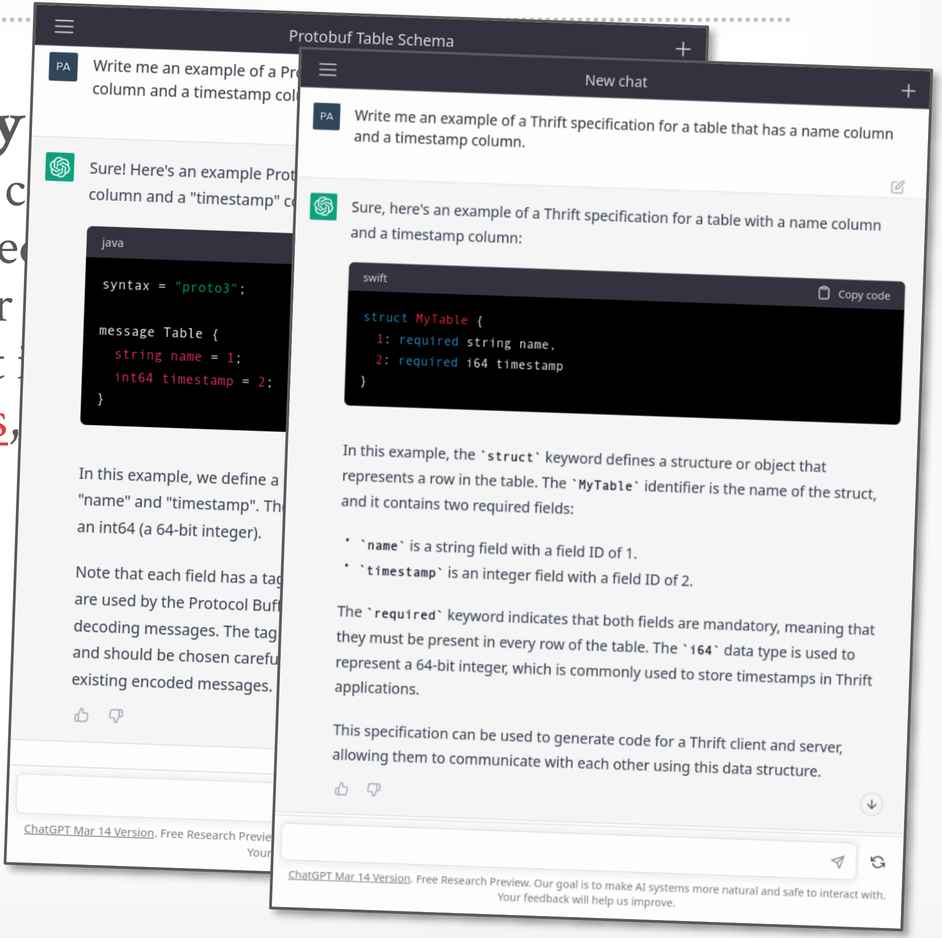
DATA SERIALIZATION

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
→ Client handles endian c
→ The closer the serialize
format, then the lower
→ DBMS can implement
libraries (ProtoBuffers,



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).

ProfaneDB

Approach #2: Text Encoding

→ Convert all binary values into strings (`atoi`).
→ Do not have to worry about endianness.

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

Approach #2: Text Encoding

→ Convert all binary values into strings (`atoi`).
→ Do not have to worry about endianness.

4-bytes 123456

+6-bytes "123456"

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

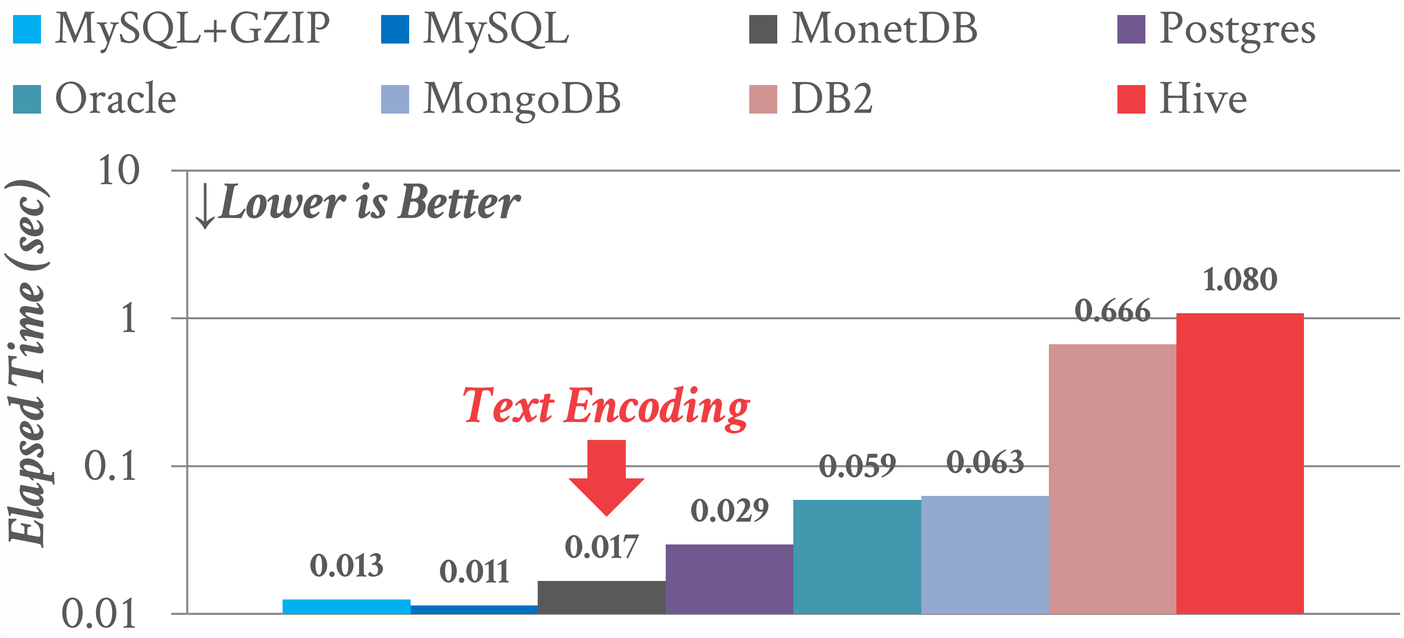
→ Add the length of the string at the beginning of the bytes.

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

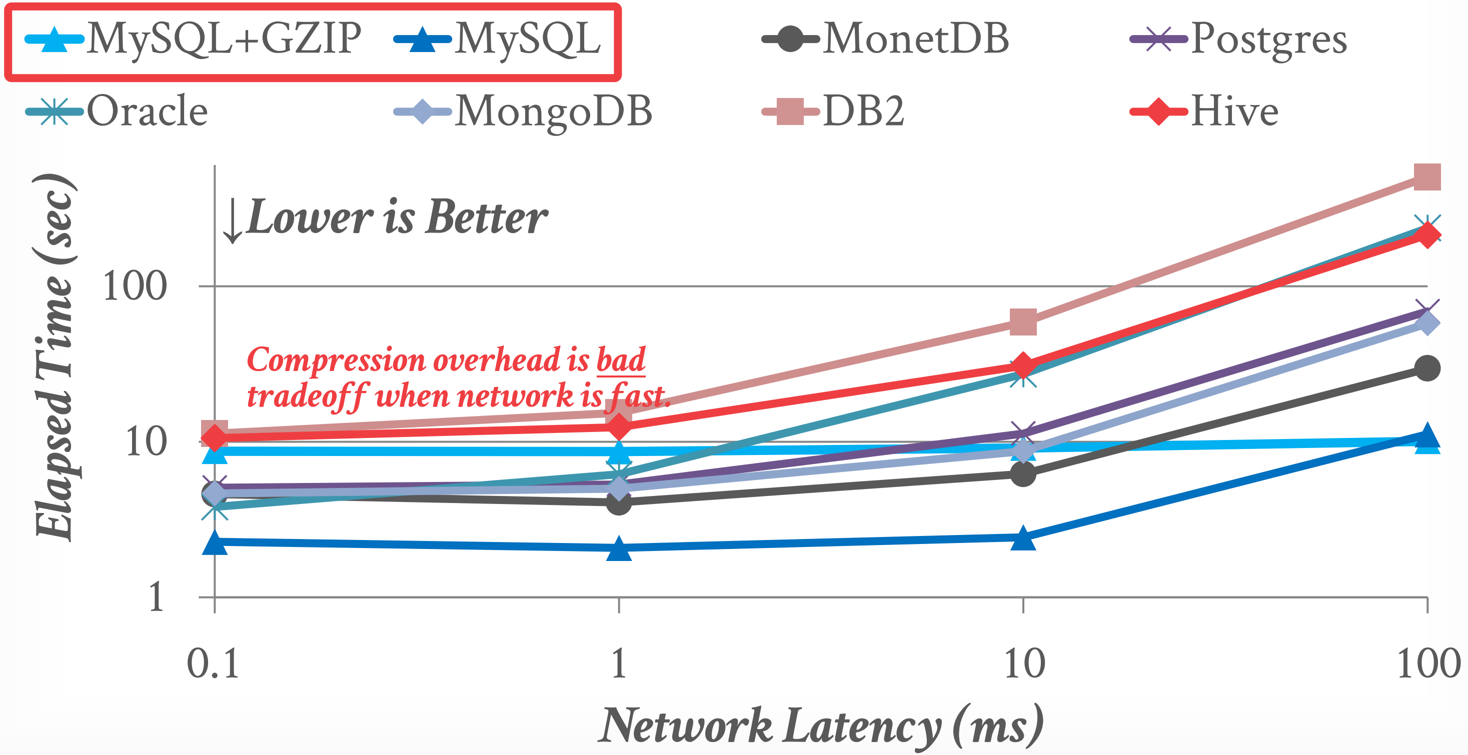


All Other Protocols Use Binary Encoding

Source: Hannes Mühleisen

NETWORK PROTOCOL PERFORMANCE

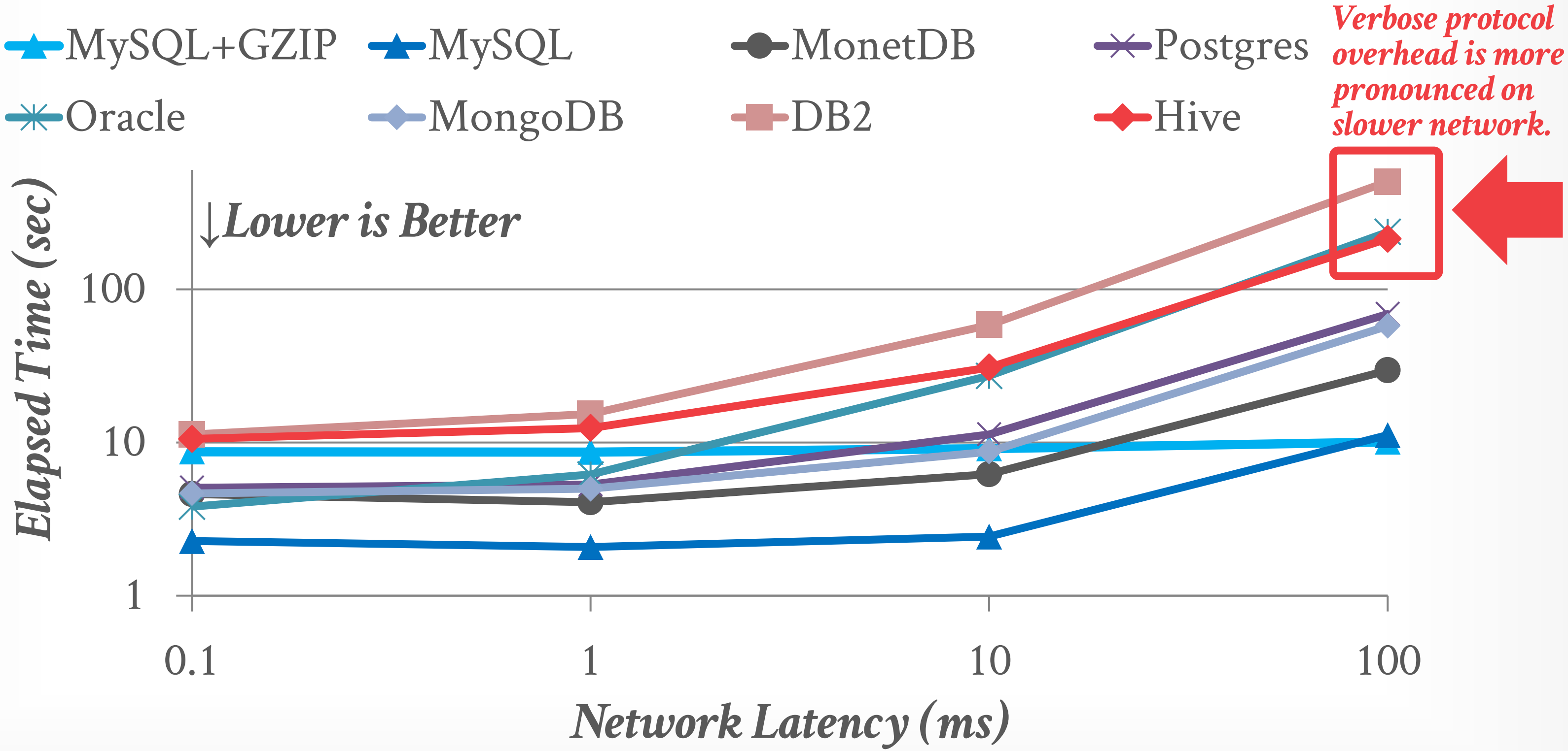
Transfer 1m Tuples from TCP-H LINEITEM



Source: Hannes Mühleisen

NETWORK PROTOCOL PERFORMANCE

Transfer 1m Tuples from TCP-H LINEITEM



Source: Hannes Mühleisen

APACHE ARROW

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)

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

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.

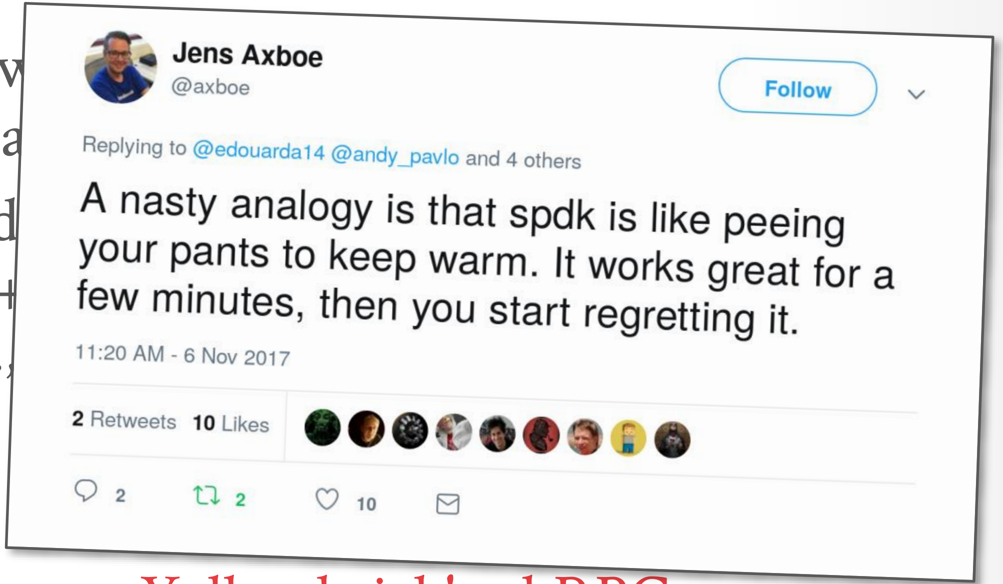
Requires the DBMS code to do more to manage network stack (layers 3+4), memory, and buffers.
→ TCP/IP in usercode (e.g., F-Stack).
→ No data copying.
→ No system calls.

Example: ScyllaDB's Seastar, Yellowbrick's ybRPC

DATA PLANE DEVELOPMENT KIT (DPDK)

Set of libraries that allow
directly. Treat the NIC a

Requires the DBMS cod
network stack (layers 3+
→ TCP/IP in usercode (e.g.,
→ No data copying.
→ No system calls.



Example: ScyllaDB's Seastar, Yellowbrick's ybRPC

DPDK
DATA PLANE DEVELOPMENT KIT

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

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.

...URING

QuestDB

Importing 300...

September 12, 2022 · 13 min read

Andrey Pechkurov
QuestDB Engineering

In this blog post, QuestDB's very own Andrei ... using the SQL COPY statement, and takes ... how the new improvement is made possible ... known OLAP and time-series databases in C...

Introduction

As an open source time series database con... fast and convenient manner is as important ... we decided to dedicate our new release, Que... discuss what parallel import means for our ... recent ClickHouse team's benchmark helped ...

TigerBeetle Docs Blog Slack GitHub Subscribe

A Programm... Abstraction... and kqueue...

By King Butcher and Phil Eaton...

Consider this tale of I/O and perfo... io_uring and kqueue, and take hom... may find familiar.

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

Classical approach...

When you want to read from a file... times as necessary to fill a buffer... direction, you call ...

ClickHouse Product Use Cases Company Learn Pricing Sign in Get Started

Engineering

A journey to io_uring, AIO and modern storage devices

Ruslan Savchenko
Mar 8, 2021

While main memory is considered to be rather cheap by some systems designers it is not always possi... everything in the main memory. When data is stored in external ... carefully how to access the data. There are several kind of storage ... e system call to read from them. We performed experiments to find ... stem calls perform for available devices. In total HDD, SATA SSD, ... ne were accessed via single-threaded and multi-threaded pread, ... interfaces. Full report is available in PDF format: link. We give one ... an example.

... Read

... block devices which means data transfer between a device and ... r than single bytes. Typically 512 bytes or 4 kilobytes blocks are ... been chosen by manufactures long time ago and may be not the ... es. By requesing larger amount of contigious data we can ... s find out how modern devices perform with larger blocks.

... block size for a random read. An application (or filesystem) can pick any block size and access data with respect to this block size. We vary block size from 4 kilobytes up to 32 megabytes. For each block size we make some random reads. Among these reads we calculate average, minimum and maximum latency as well as 99,0 and 99,9 percentiles. We use system call pread(2) in this experiment. We believe that lseek(2) followed

alexey-milovidov on Sep 16, 2022 • edited — Member

There was an experiment adding uring by @sauliusvl
Unfortunately, it proves to be unsustainable.

There is only a marginal improvement in performance. But the code becomes way more complicated. It became so complicated that even an experienced C++ engineer (the author of the code) cannot figure out why there are rare hangs of queries (found by our automated testing before the release).

👍 1

IO_URING

Linux system call interface for zero-copy asynchronous I/O operations.

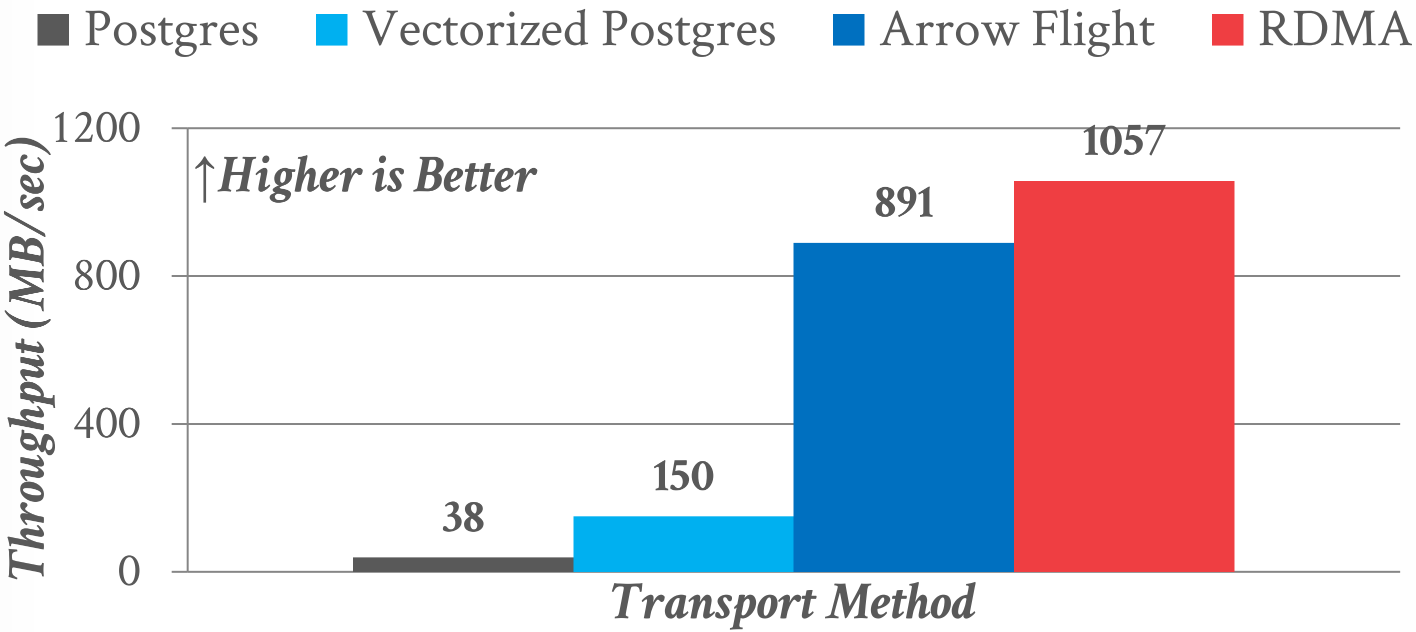
→ Originally added in 2019 for accessing storage devices.
→ Expanded in 2022 to support network devices.
→ Windows has something similar called ICOP.

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



MAINLINING DATABASES: SUPPORTING FAST TRANSACTIONAL WORKLOADS ON UNIVERSAL COLUMNAR DATA FILE FORMATS
VLDB 2020

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.

TASTES GREAT! LESS FILLING! HIGH PERFORMANCE AND ACCURATE TRAINING DATA COLLECTION FOR SELF-DRIVING DATABASE MANAGEMENT SYSTEMS
SIGMOD 2022

eBPF

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.

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

NEXT CLASS

Query Optimizer