Lecture 25: Networking + Course Retrospective

Recap

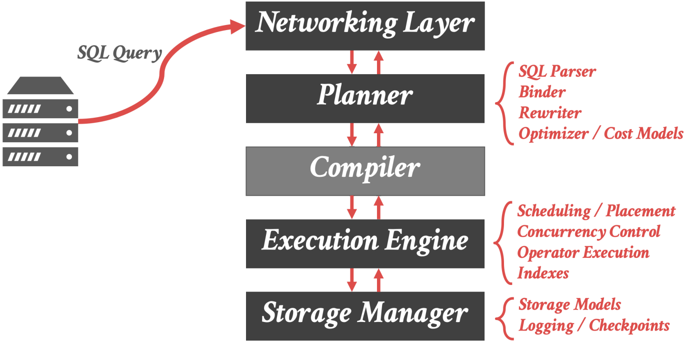
User-Defined Functions

- A **user-defined function** (UDF) is a function written by the application developer that extends the system's functionality beyond its built-in operations.
  - It takes in input arguments (scalars)
  - Perform some computation
  - Return a result (scalars, tables)
- **Examples:** PL/SQL, plPG/SQL

Froid: UDF In-lining

- Automatically convert UDFs into relational expressions that are inlined as sub-queries.
    - Does not require the app developer to change UDF code.
- Perform conversion during the rewrite phase to avoid having to change the cost-base optimizer.
    - Commercial DBMSs already have powerful transformation rules for executing sub-queries efficiently.
- Reference

Architecture Overview



Today’s Agenda

- Database Access APIs
- Database Network Protocols
- Database Replication Protocols
- Kernel Bypass Methods
- Course Retrospective

Database Access APIs

Database Access APIs

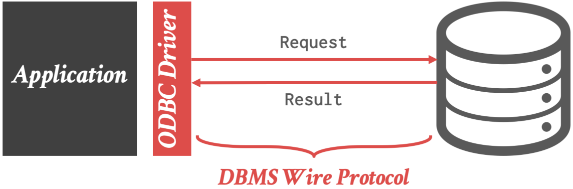
- With a terminal-based client (*e.g.*, `psql`):
  - SQL queries are written by hand.
  - Results are printed to the terminal.
- Real programs access a database through an API:
  - Direct Access (DBMS-specific)
  - Open Database Connectivity (ODBC)
  - Java Database Connectivity (JDBC)

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.
- The driver can emulate missing DBMS features (*e.g.*, cursors).



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 Network Protocols

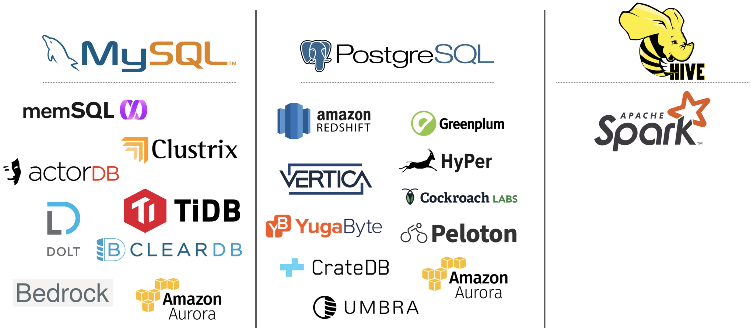
Database Network Protocols

- 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



Protocol Design Space

- Row vs. Column Layout
- Compression
- Data Serialization
- String Handling
- Reference

Row vs. Column Layout

- ODBC/JDBC are inherently row-oriented APIs.
  - Server packages tuples into messages one tuple at a time.
  - Client must 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.
  - Batch of rows organized in a column-oriented layout.

Compression

- **Approach 1: Naive 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

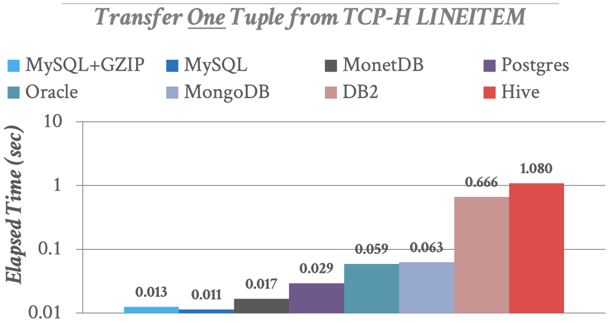
- **Approach 1: Binary Encoding**
  - Client handles endian conversion.
  - The closer the serialized format is to the DBMS's binary format, then the lower the overhead to serialize.
  - DBMS can implement its own format or rely on existing libraries (ProtoBuffers, Thrift, FlatBuffers).
- **Approach 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.

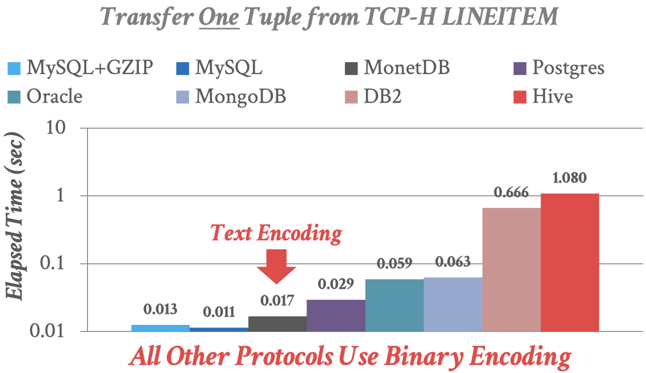
Network Protocol Performance

- Transfer One Tuple from TCP-H LINEITEM



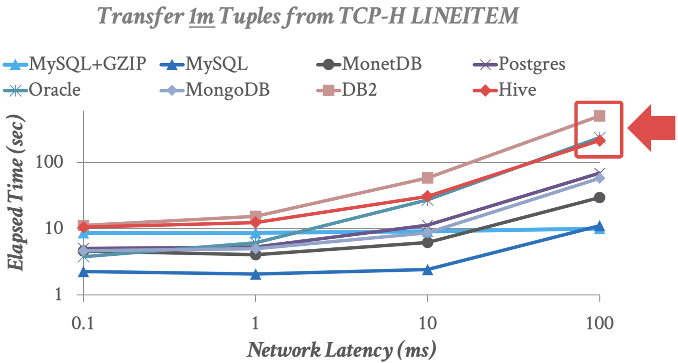
Network Protocol Performance

- Transfer One Tuple from TCP-H LINEITEM



Network Protocol Performance

- Transfer 1m Tuples from TCP-H LINEITEM



Database Replication Protocols

Replication Protocols

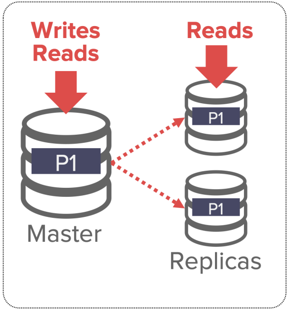
- DBMSs will propagate changes over the network to other nodes to increase availability.
  - Send either physical or logical log records.
  - Granularity of log record can differ from WAL.
- Design Decisions:
  - Replica Configuration
  - Propagation Scheme

Replica Configurations

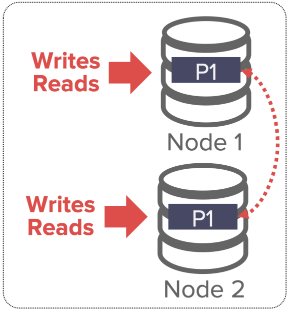
- **Approach 1: Master-Replica**
  - All updates go to a designated master for each object.
  - The master propagates updates to its replicas without an atomic commit protocol.
  - Read-only txns may be allowed to access replicas.
  - If the master goes down, then hold an election to select a new master.
- **Approach 2: Multi-Master**
  - Txns can update data objects at any replica.
  - Replicas must synchronize with each other using an atomic commit protocol.

Replica Configurations

Master-Replica



Multi-Master

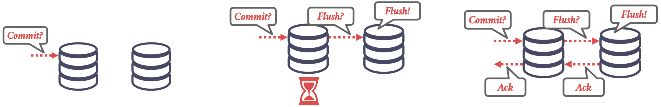


Propagation Scheme

- When a txn commits on a replicated database, the DBMS decides whether it must wait for that txn's changes to propagate to other nodes before it can send the acknowledgement to application.
- Propagation levels:
  - Synchronous (Strong Consistency)
  - Asynchronous (Eventual Consistency)

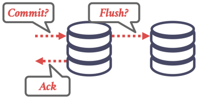
Propagation Scheme

- **Approach 1: Synchronous**
  - The master sends updates to replicas and then waits for them to acknowledge that they fully applied (i.e., logged) the changes.



Propagation Scheme

- **Approach 2: Asynchronous**
  - The master immediately returns the acknowledgement to the client without waiting for replicas to apply the changes.



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

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 (DPDK)

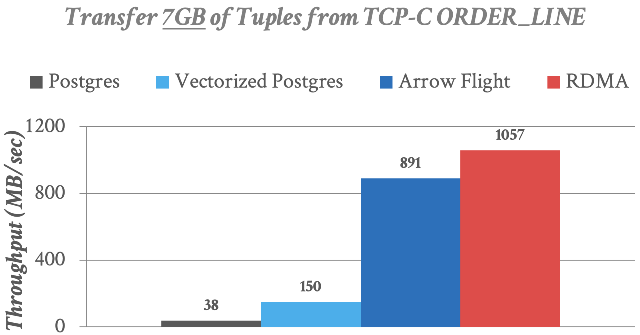
- Set of libraries that allows programs to access NIC directly. Treat the NIC as a bare metal device.
- Requires the DBMS code to do more to manage memory and buffers.
  - No data copying.
  - No system calls.
- Example: ScyllaDB

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: Oracle RAC, Microsoft FaRM

Data Export Performance

- Transfer 7GB of Tuples from TCP-C ORDER_LINE



Conclusion

Conclusion

- 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.

Retrospective

Lessons learned

- Let’s take a step back and think about what happened
- Systems programming is both hard **and** rewarding
- Become a better programmer through the study of database systems internals
- Going forth, you should have a good understanding how systems work

Big Ideas

- Database systems are awesome – but are not magic.
- Elegant abstractions are magic.
- Declarativity enables usability and performance.
- Building systems software is more than hacking
- There are recurring motifs in systems programming.
- CS has an intellectual history and you can contribute.

What Next?

- We have covered the entire stack of systems programming
  - Storage Management (Part 1)
  - Access Methods (Part 1)
  - Query Execution (Part 1)
  - Logging and Recovery Methods (Part 2)
  - Concurrency Control (Part 2)
  - Query Optimization (Part 2)
- Stay in touch
  - Tell me when this course helps you out with future courses (or jobs!)
  - Ask me cool DBMS questions

Parting Thoughts

- You have surmounted several challenges in this course.
- You make it all worthwhile.
- Please share your feedback via CIOS.
- Go forth and spread the gospel of data systems!

Next Class

- Project Presentations