Yellowbrick Database System
**Project:**
→ Final Presentations: *Thursday May 2^{nd} @ 9:00am*
→ See [Piazza@59](https://piazza.com) for more information.

**Final Exam:**
→ Given in class on *Wednesday April 24th*
→ Due on the same day as Final Presentation
LAST CLASS

DuckDB Embedded OLAP DBMS
HISTORICAL CONTEXT

Most DBMSs are designed for off-the-shelf hardware. But some vendors sell a complete solution ("appliance") where the DBMS is optimized for a specific hardware configuration. Companies also fab custom database accelerators too.

Yellowbrick started off as a high-end OLAP appliance...
HISTORICAL CONTEXT

Most DBMSs are designed for off-the-shelf hardware. But some vendors sell a complete solution ("appliance") where the DBMS is optimized for a specific hardware configuration. Companies also fabricate custom database accelerators.

IBM Netezza
- Performance and price/performance leader
- Speed and ease of deployment and administration

IBM Netezza stand-alone appliance
- Strategic requirement for stand-alone decision support system
- If primary data feeds are from distributed applications
- Deep analytics applications or in-database mining

IBM DB2 Analytics Accelerator for z/OS
- Transparent acceleration of existing reporting workload on DB2

Clustrix Appliance
Clustrix Appliance 3 Node Cluster (CLX 4110)
- 24 Intel Xeon CPU cores
- 144GB RAM
- 8GB NVRAM
- 1.35TB Intel SSD protected
- (2.7TB raw) data capacity
- Low-latency InfiniBand interconnect

Teradata IntelliFlex
100% Solid State Performance

Up to: 7.5x Performance for Computation Intensive Analytics

4.5x Performance for Data Warehouse Analytics

3.5x Data Capacity

2.0x Performance per kW

Oracle Database Appliance
**HISTORICAL CONTEXT**

Most DBMSs are designed for off-the-shelf hardware. But some vendors sell a complete solution ("appliance") where the DBMS is optimized for a specific hardware configuration.

Companies also fabricate custom database accelerators too.

Yellowbrick started off as a high-end OLAP appliance…

---

**Who and what is Yellowbrick?**

- Founded 2014, on the market since 2017
- MPP SQL relational data warehouse
  - Shared nothing
  - Deployed on your own hardware
  - FPGA-accelerated
- Hybrid row/column store
  - 10 TB/hr bulk load speed, 1M+ rows/s streaming
- Used by hedge funds, telcos, banks,…
  - Business-critical applications
- …not talking about hardware today…it’s about the software and journey to cloud
**HISTORICAL CONTEXT**

Most DBMSs are designed for off-the-shelf hardware. But some vendors sell a complete solution (“appliance”) where the DBMS is optimized for a specific hardware configuration. Companies also fabricate custom database accelerators.

Yellowbrick started off as a high-end OLAP appliance…

---

**Kalidah Accelerator**

The gap between CPU performance and storage throughput continues to grow with storage throughout, making every couple of years put CPU core counts and clock rates increasing. Accelerator processors. This has transformed the GPUs for graphics and has evolved into special processor variants widely used in large-scale data processing to move data away from the host, recognizing the environment in autonomous vehicles, or taking better photos or mutually increasing.

Kalidah warehousing, we have observed that substantial amounts of CPU time are spent just finding the data on which we want to operate — coming through a substantial layer of overhead. We have started to address our accelerator by offloading this effort from the host CPU to a new dedicated processor, designed to do that at higher rates than software alone can accomplish.

The Kalidah processor is a high-bandwidth, high-bandwidth data processing blade used in the table space for data validation, compression, filtering, compression, and organization. Each Kalidah core receives instructions and compression data model to the Kalidah protocol. Like other Teradatian data, the Kalidah data is a high-performance, scalable, and parallel for complex data management and query optimization.

- Moving and organizing data via DMA.
- Parallelizing data with multiple kernels on the host.
- Applying range filters and bloom filters to data.
- Reorganizing data to remove rows that do not meet filter criteria.

Kalidah can support these operations on all data types currently supported by the Yellowbrick platform. This can be aggressively parallelized and optimized to be on the same scale data in its memory in a database.

Kalidah offers a substantial increase in performance compared with running optimized software on the CPU, resulting in much higher node ratios in the following capabilities:

- **Stream throughput:** Each and every query can be decompressed, filtered, compressed, and reorganized on the CPU. This results in a query that is 10x faster than the CPU.
- **Bloom filtering:** Bloom filtering is one of the most important operations used to look for specific data in large datasets and to accelerate queries. It involves masking and matching the Bloom filter against multiple Bloom filters. Each Bloom filter is a binary vector, and it is used to check if the data type is in the table. This is equivalent to about 1x CPU core of performance for fixed-length data types and 5x CPU core of performance for variable-length data types.

Yellowbrick continually tunes and optimizes its database software to make more use of Kalidah.
OLAP DBMS written on C++ and derived from a hardfork of PostgreSQL v9.5.

→ Uses PostgreSQL's front-end (networking, parser, catalog) to handle incoming SQL requests.

Originally started as an on-prem appliance with FPGA acceleration. Switched to DBaaS in 2021.

Cloud-version uses Kubernetes for all components.
Shared-Disk / Disaggregated Storage
Push-based Vectorized Query Processing
Transpilation Query Codegen (C++)
Compute-side Caching
Separate Row + PAX Columnar Storage
Sort-Merge + Hash Joins
PostgreSQL Query Optimizer++
Insane Systems Engineering
YELLOWBRICK: ARCHITECTURE

Data Warehouse Instance:
→ Front-end service that manages connections, parsing, plan caching, row store, meta-data, and concurrency control.

Worker Nodes:
→ Responsible for query execution, managing compute hardware, and maintaining local cache.

Background / Maintenance Nodes:
→ Compilation, Bulk Loading
YELLOWBRICK: ARCHITECTURE

Source: Mark Cusack
YELLOWBRICK: ARCHITECTURE

Source: Mark Cusack
YELLOWBRICK: ARCHITECTURE

- **Row-Store**
  - PostgreSQL
  - Scheduler
  - Compiler Service
  - Bulk Loader Service

- **Worker Nodes**
  - Custom S3 Client
  - Custom NVMe Driver
  - Custom UDP Protocol

- **Object Store**

Source: Mark Cusack

15-721 (Spring 2024)
YELLOWBRICK: ARCHITECTURE

Based on a microservice architecture where all components run as Docker pods in Kubernetes.
→ Kubernetes handles system state management, scalability, and provisioning.
→ Hides all Kubernetes operations behind SQL (!!!).

Assigns one worker pod per worker node to guarantee exclusive access to hardware.
YELLOWBRICK: QUERY EXECUTION

Pushed-based vectorized query processing that supports both row- and columnar-oriented data with early materialization.

→ Introduces transpose operators to convert data back and forth between row and columnar formats.

Holistic query compilation via source-to-source transpilation.

Yellowbrick's architecture goal is for workers to always process data residing in the CPU's L3 cache and not memory.
YELLOWBRICK: QUERY COMPILATION

Split query plan into independent fragments and then transpilie each fragment into C++ source code. Dedicated compilation service uses LLVM to compile each fragment into machine code.

→ Use separate threads to compile fragments and then stitch them back together at runtime with dynamic linking.

Compiler service maintains a fragment cache to reduce compilation costs.

→ Tracks engine version and other dependencies.
YELLOWBRICK: QUERY OPTIMIZER

Heavily modified version of PostgreSQL's stratified optimizer to do support zone map filtering.

Yellowbrick's main addition is a cost-based join order selection using statistics collected from row-store compaction and **ANALYZE** passes over data. → Histograms, HyperLogLog, HeavyHitters

Supports sideways information passing of Bloom filters for hash joins.

Source: Mark Cusack
YELLOWBRICK: STORAGE

Yellowbrick only supports managed storage based on its proprietary file format.
→ Can specify sharding / local-sorting attribute per table.
→ ~100MB files with 2MB chunks
→ Supports bulk loading Parquet files with some limitations.

Maintains row-store data in front-end and columnar data in object store.
→ Background task to move row-store data to columnar files.
→ Also supports compaction of modified columnar files.
→ DBMS bulk loads to object store in columnar files, bypassing row-store and worker SSD caches.
YELLOWBRICK: SHARDING

The DBMS assigns data files to workers using **Rendezvous Hashing**. → Also used in **Druid** and **Ignite**.

For each file, generate a hash value for each worker by concatenating the worker's identifier to the hashed key. Pick the hash value with the highest weight.

- $\text{hash(file1 + worker1)} = 100$
- $\text{hash(file1 + worker2)} = 90$
- $\text{hash(file1 + worker3)} = 80$
The DBMS assigns data files to workers using **Rendezvous Hashing**. → Also used in **Druid** and **Ignite**.

For each file, generate a hash value for each worker by concatenating the worker's identifier to the hashed key. Pick the hash value with the highest weight.
The DBMS assigns data files to workers using **Rendezvous Hashing**. Also used in **Druid** and **Ignite**.

For each file, generate a hash value for each worker by concatenating the worker's identifier to the hashed key. Pick the hash value with the highest weight.
The DBMS assigns data files to workers using **Rendezvous Hashing**. → Also used in **Druid** and **Ignite**.

For each file, generate a hash value for each worker by concatenating the worker's identifier to the hashed key. Pick the hash value with the highest weight.
The DBMS assigns data files to workers using **Rendezvous Hashing**. Also used in **Druid** and **Ignite**.

For each file, generate a hash value for each worker by concatenating the worker's identifier to the hashed key. Pick the hash value with the highest weight.
The DBMS assigns data files to workers using **Rendezvous Hashing**. → Also used in **Druid** and **Ignite**.

For each file, generate a hash value for each worker by concatenating the worker's identifier to the hashed key. Pick the hash value with the highest weight.
The DBMS assigns data files to workers using **Rendezvous Hashing**. → Also used in **Druid** and **Ignite**.

For each file, generate a hash value for each worker by concatenating the worker's identifier to the hashed key. Pick the hash value with the highest weight.
Remember that the OS is our enemy.

What can a DBMS implement for itself if it wants to ensure that it never has to talk to the OS after starting up?
YELLOWBRICK: OS OPTIMIZATIONS

Memory Allocator
Thread Scheduler
Device Drivers
Network Protocols
YELLOWBRICK: MEMORY ALLOCATOR

Custom NUMA-aware, latch-free allocator that gets all the memory needed upfront at start-up

→ Using `mmap` with `mlock` with `huge pages`.

→ Allocations are grouped by query to avoid fragmentation.

→ Claims their allocator is 100x faster than libc `malloc`.

Each worker also has a buffer pool manager that uses MySQL-style approximate LRU-K to store cached data files.
MEMORY PAGES

OS maps physical pages to virtual memory pages. The CPU's MMU maintains a TLB that contains the physical address of a virtual memory page.

→ The TLB resides in the CPU caches.
→ It cannot obviously store every possible entry for a large memory machine.

When you allocate a block of memory, the allocator keeps that it aligned to page boundaries.
HUGE PAGES

Instead of always allocating memory in 4 KB pages, Linux supports creating larger pages (2MB to 1GB)
→ Each page must be a contiguous blocks of memory.
→ Greatly reduces the # of TLB entries

Recent research from Google suggests that huge pages improved their data center workload by 7%.
→ 6.5% improvement in Spanner's throughput

Huge Pages makes sense in an OLAP DBMS that is accessing large read-only data blocks at a time.

Source: Evan Jones
HUGE PAGES

Instead of always allocating memory in 4 KB pages, Linux supports creating larger pages (2MB to 1GB)
→ Each page must be a contiguous blocks of memory.
→ Greatly reduces the # of TLB entries

Recent research from Google suggests that huge pages improved their data center workload by 7%.
→ 6.5% improvement in Spanner's throughput

Huge Pages makes sense in an OLAP DBMS that is accessing large read-only data blocks at a time.

Source: Evan Jones
With **Transparent Huge Pages** (THP), the OS reorganizes and compacts pages in the background.

- Split larger pages into smaller pages.
- Combine smaller pages into larger pages.
- Can cause the DBMS process to stall on memory access.

Nearly every DBMS advises to disable THP:

- Oracle, SingleStore, NuoDB, MongoDB, Sybase, TiDB.
- Vertica says to enable THP only for newer Linux distros.

Source: Alexandr Nikitin
With **Transparent Huge Pages (THP)**, the OS reorganizes and compacts pages in the background.

→ Split larger pages into smaller ones.
→ Combine smaller pages into larger pages.
→ Can cause the DBMS process to stall on memory access.

Nearly every DBMS advises to **disable** THP:

→ Oracle, SingleStore, NuoDB, MongoDB, Sybase, TiDB.
→ Vertica says to **enable THP only for newer Linux distros**.

**Source:** Alexandr Nikitin
Single LRU linked list but with two entry points ("old" vs "young").

→ New pages are always inserted to the head of the old list.
→ If pages in the old list is accessed again, then insert into the head of the young list.
Single LRU linked list but with two entry points ("old" vs "young").
→ New pages are always inserted to the head of the old list.
→ If pages in the old list is accessed again, then insert into the head of the young list.
MYSQL APPROXIMATE LRU-K

Single LRU linked list but with two entry points ("old" vs "young").
→ New pages are always inserted to the head of the old list.
→ If pages in the old list is accessed again, then insert into the head of the young list.
Single LRU linked list but with two entry points ("old" vs "young").

→ New pages are always inserted to the head of the old list.

→ If pages in the old list is accessed again, then insert into the head of the young list.
MySQL Approximate LRU-K

Single LRU linked list but with two entry points ("old" vs "young").

→ New pages are always inserted to the head of the old list.
→ If pages in the old list is accessed again, then insert into the head of the young list.
YELLOWBRICK: SCHEDULER

Custom cooperative multi-tasking thread scheduler (coroutines) that synchronizes every 100ms with a centralized cluster scheduler.

Only one query executes at a time in a cluster. All cores on the same worker execute the same task at the same time.
→ The goal is to ensure that cores are processing recently arrived data in L3 instead of memory.
YELLOWBRICK: DEVICE DRIVERS

Custom NVMe / NIC drivers that run in user-space to avoid memory copy overheads.
→ Falls back to Linux drivers if necessary.

Custom reliable UDP network protocol with kernel-bypass (DPDK) for internal communication.
→ Each CPU has its own receive/transmit queues that it polls asynchronously.
→ Only sends data to a "partner" CPU at other workers.
BENCHMARK

TPC-DS (Scalefactor 1)

Source: Mark Cusack
PARTING THOUGHTS

Yellowbrick's systems engineering street skills are ridiculously impressive.
→ If building it today, you should probably use eBPF instead of DPDK.

But remember that all these optimizations will not matter if the DBMS chooses crappy query plans.
PARTING THOUGHTS

Yellowbrick's systems engineering street skills are ridiculously impressive.
→ If building it today, you should probably use eBPF instead of DPDK.

But remember that all these optimizations will **not** matter if the DBMS chooses crappy query plans.
Last lecture: Amazon Redshift