NautDB: Towards a Hybrid Runtime for Processing Compiled Queries

HExSA Lab

Samuel Grayson (sag150430@utdallas.edu)

University of Texas at Dallas

UT DALLAS Erik Jonsson School of Engineering and Computer Science

ILLINOIS INSTITUTE OF TECHNOLOGY

Abstract

Goal: to integrate specialization techniques from the OS community **hybrid runtimes** and DB community **compiled queries** for high-performance querying on **big data**.

- Certain abstractions improve generality, but get in the way of performance at **exascale computing**.
- In specific cases, give up flexibility and generality in exchange for performance.
- We prototype **NautDB**: specialized using a **hybrid runtime kernel** (based on the **Nautilus Aerokernel** [1]) and executing pre-compiled building-blocks.
- We demonstrate performance benefits in certain cases, while maintaining a simple interface for users.

Specialized Hybrid Runtimes [1]

- Kernel + Runtime run in ring 0 (unikernel-like) $\implies$ fewer context switches
- Partition physical resources between general purpose OS and hybrid runtime [2] $\implies$ can call general purpose OS where needed.
- Unikernel-inspired design gives programmer fine-grained control over ...
  - Not time-shared $\implies$ No context-switches or thread-migration
  - Avoid interrupts $\implies$ Faster and more predictable
  - Single address space $\implies$ Huge page-sizes (1Gb) $\implies$ No TLB misses
  - Memory allocation $\implies$ Specialize allocator for workload/application (see Fig. 1)

Compiled Query Processing [4, 3]

- DB engine is specialized for a particular query
- Can be pre-compiled or JIT-compiled
  - Smaller code $\implies$ less i-cache misses and branches
  - Specialized data structures
  - Combine code for multiple operators (pipeline) $\implies$ optimizations within operators and less func calls
  - Specialize code for hardware without blow up in code size
- The DB-programmer writes compiled code in the unikernel, giving them unfettered access HW.
  - Compose operators within a block $\implies$ avoid function-call overhead.
  - Control HW $\implies$ NUMA-aware code.

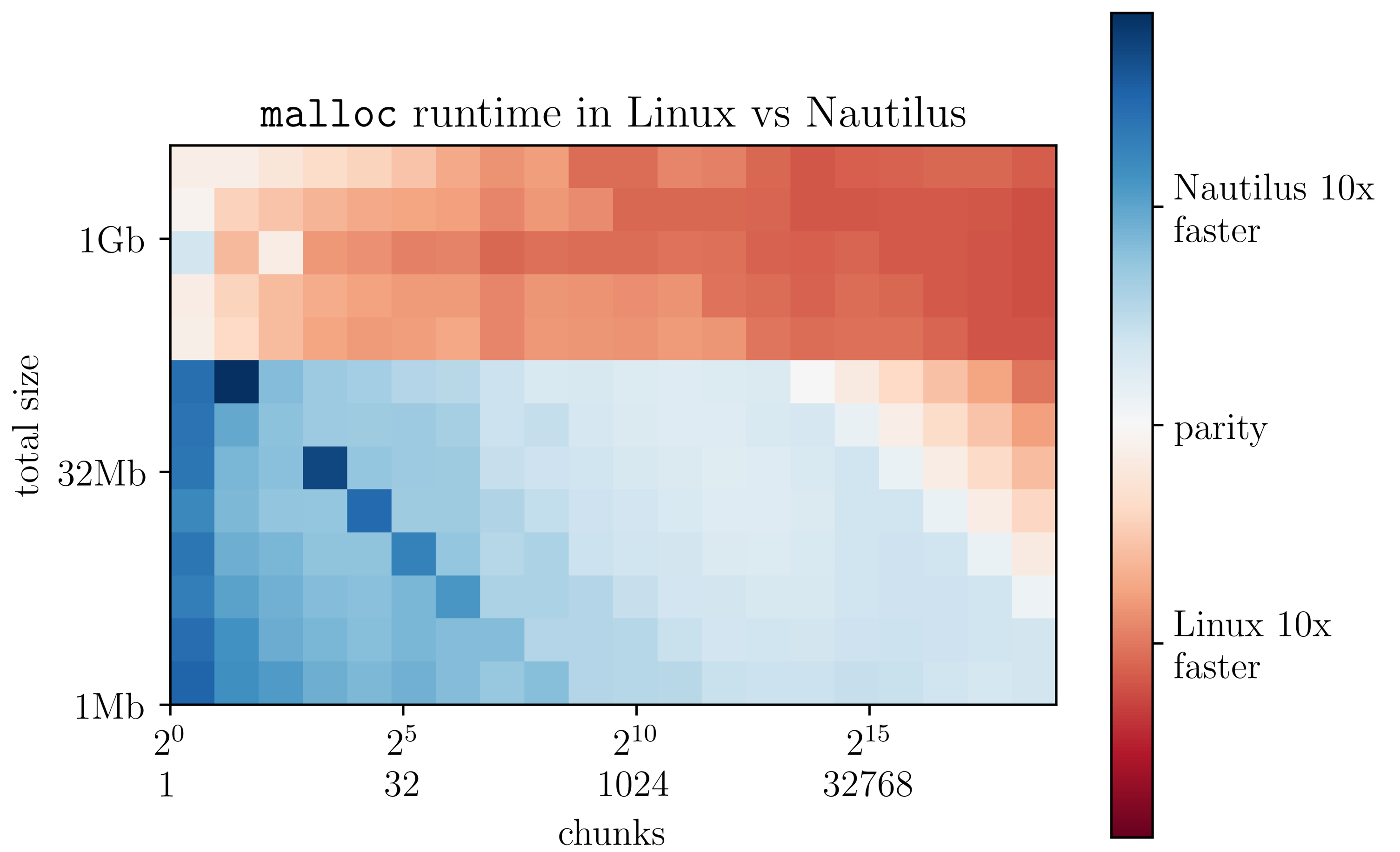

Figure 1: The custom allocator outperforms the highly optimized allocator of linux for smaller sizes by more than an order of magnitude. This demonstrates the potential of specialization—the behaviour of a component can be fine-tuned to workload characteristics.

Hypothesis

Executing compiled queries in a hybrid runtime can improve performance within parameter ranges that benefit from a particular specialization and leads to more predictable performance.

NautDB Prototype

- Hand-optimized, in-memory, columnar, chunk-oriented operator implementations
- Tables are stored in a column-oriented fashion, the data of a columns is split into 256 chunks - each is an array of data values of a fixed size
- We tested creating, sorting, selection (filter), creating, and freeing tables
- The database server is compiled into Nautilus and invoked on startup (single-application OS)

Configuration

- We vary the number of columns and the size of chunks
- We aggregated over 10 trials
- Hardware: 16-core x86_64 AMD EPYC 7281 with 4 NUMA nodes
- OS: linux kernel 4.17.6, Nautilus - git commit `2fb4e52816`
- Nautilus has default configuration with debugging removed and extra devices disabled. It uses 1GB pages.

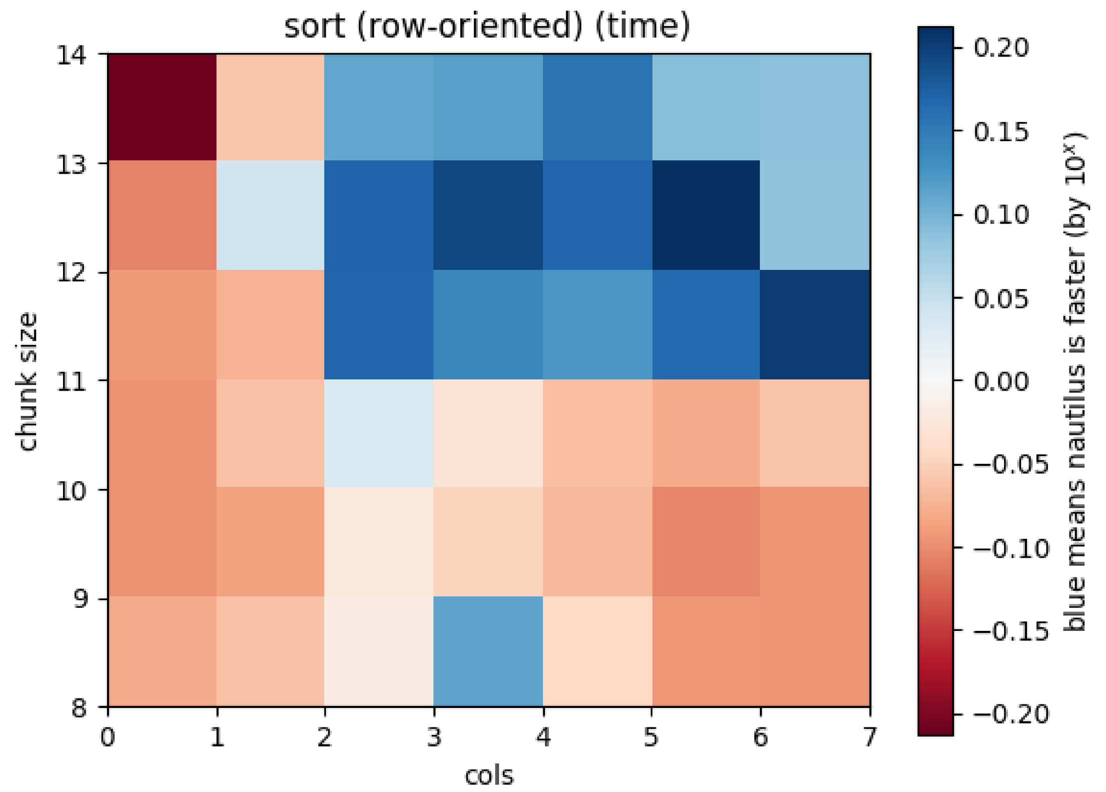

Figure 2: Row-oriented sort in Nautilus and Linux varying the # of columns and chunk-size

For larger number of columns and chunk sizes, Nautilus outperforms Linux. This is because Nautilus has larger page size and incurs less TLB misses (see Table 1).

Kernel	TLB misses	Instruction cache misses
Linux	135,000,000	3,030,000
Nautilus	1	480,000

Table 1: Row-oriented sorting for a 128 column table with 256 chunks with 8192 elements

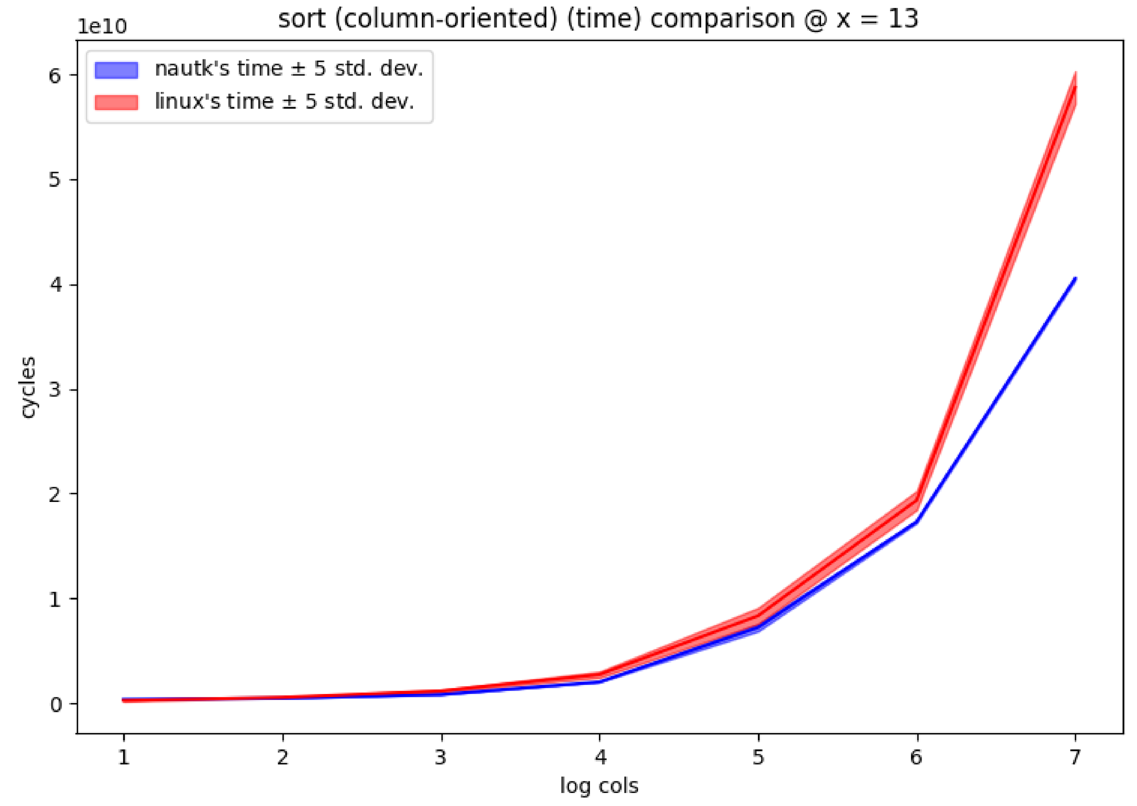

Figure 3: Column-oriented sort measuring runtime and uncertainty for a fixed chunk size (8192), varying the number of columns.

Nautilus performance is more predictable than Linux, even when its raw performance is worse. Nautilus does not have scheduling interrupts, so it avoids unpredictable detours which also leads to better cache performance (see Table 2).

Kernel	TLB misses	Instruction cache misses
Linux	2,500,000	8,200,000
Nautilus	1	1,100,000

Table 2: Column-oriented sorting for a 128 column table with 256 chunks with 8192 elements each

Discussion

- Specialization allows us to beat Linux with relatively little effort.
- We should be able to beat Linux everywhere
  - By switching to Linux's algorithm when the workload is suited to it
- Hybrid runtimes have more predictable performance than general purpose OS
  - will lead to better performance once we go parallel (e.g., bulk-synchronous model)

Next Steps

- Borrow algorithms from Linux where they outperform the existing Nautilus algorithms
- Make the development process simpler for the user
- Evaluate parallel implementations of operators
- Evaluate effect of noise due to other applications running (in a partitioned VM environment)

Our Vision: NautDB

Implement our vision of *NautDB* as a hybrid dataflow engine for compiled queries

- **VMs**: Since NautDB is an application-specific OS, it could be running in its own VM alongisde a general-purpose OS.
- **Frontend**: compiles queries into executable tasks for the dataflow backend and takes care of higher-level scheduling and runtime optimization tasks (e.g., task placement and global planning)
- **Threads**: will be pinned to CPU cores; they will run to completion without interruption.
- **Work distribution**: Cores are either workers that pull tasks from a local (lock-free) queue and execute them or management threads that pass tasks from clients to workers and hand results to clients, push statistics about task execution and resource utilization to clients, and garbage collect task memory.

References

[1] K. C. Hale and P. A. Dinda. A case for transforming parallel runtimes into operating system kernels. In *Proceedings of the 24th ACM Symposium on High-performance Parallel and Distributed Computing (HPDC 2015)*, June 2015.

[2] B. Kocoloski, J. Lange, H. Abbasi, D. E. Bernholdt, T. R. Jones, J. Dayal, N. Evans, M. Lang, J. Lofstead, K. Pedretti, and P. G. Bridges. System-level support for composition of applications. In *Proceedings of the 5th International Workshop on Runtime and Operating Systems for Supercomputers (ROSS 2015)*, June 2015.

[3] T. Neumann. Efficiently compiling efficient query plans for modern hardware. *PVLDB*, 4(9):539–550, 2011.

[4] A. Shaikhha, I. Klonatos, L. E. V. Parreaux, L. Brown, M. Dashti Rahmat Abadi, and C. Koch. How to architect a query compiler. In *SIGMOD*, number EPFL-CONF-218087, 2016.