NautDB: Towards a Hybrid Runtime for Processing Compiled Queries

Samuel Grayson[1], Kyle Hale[2], Boris Glavic[2]

1 University of Texas at Dallas, 2 Illinois Insitute of Technology

ILLINOIS INSTITUTE OF TECHNOLOGY

Abstract

- With the explosion of "big data", executing queries on vast amounts of data in parallel has become a bottleneck in many HPC systems and beyond.
- Complex multi-layered abstractions behind generic interfaces in the traditional OS and database software stack are getting in the way of exploiting the characteristics of modern multi-core systems to maximize performance.
- To mitigate this inefficiency, we built **NautDB**, a **hybrid runtime kernel** (based on the **Nautilus Aerokernel** [2]) for the parallel execution of **compiled queries**, thus, combining for the first time the concepts of hybrid runtimes and compiled query processing developed by the operating system and database communities.
- We implement a **testing prototype** to evaluate the potential performance benefits that can be achieved by our specialized hybrid runtime against a general-purpose runtime, for which Linux will serve as our exemplar.

The goal of this research is to integrate specialization techniques from the OS community (unikernels) and DB community (compiled queries) to achieve high-performance.

Motivation

Status quo

- Too many abstractions $\implies$ high-performance is difficult to achieve
- General purpose OS are not aware of the workload; app-programmer is.
- DBs already use specialized scheduling and memory management abstractions.

Solution: Give up flexibility and generality in exchange for performance.

Specialized Hybrid Runtimes

- Kernel + Dataflow Runtime run in ring 0 (unikernel-like).
- Advances in multi-kernel environments allow physical **TODO: (cite pisces)** resources to be split between a full-weight kernel and a hybrid runtime.
- Unikernel-inspired design allows the app-programmer fine-grained control over the execution environment.
  - Control over scheduling $\implies$ no context-switches.
  - Control over OS $\implies$ avoid interrupts $\implies$ faster and more predictable.
  - No virtual memory $\implies$ huge page-sizes (1Gb) $\implies$ no TLB misses.
  - Control over memory $\implies$ specialize memory-allocator for workload/application (see Fig. 1)

Compiled Query Processing

- Could be pre-compiled or JIT-compiled.
- The app-programmer writes compiled code in the unikernel, giving them

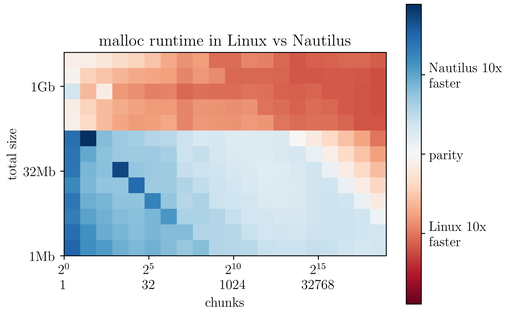


Figure 1:Performance of allocating memory in chunks comparing linux against a custom memory allocator in Nautilus. Using specialization the custom allocator outperforms the highly optimized memory allocator of linux for smaller sizes by more than an order of magnitude. This demonstrates the potential of specialization - the behaviour of a component can be fine-tuned to workload characteristics.

Our Vision: NautDB

Previous Work

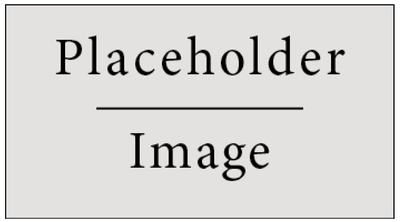
- Customized OS support for data-processing [1]: **TODO: I am not sure what specific improvements we have over this paper.**
- A Case for Transforming Parallel Runtimes Into Operating System Kernels [2]: the authors use a specialized runtime and show performance benefit for runtimes such as Legion.
- System-Level Support for Composition of Applications [3]: the authors introduce Pisces which allows multiple kernels to run on the same physical node, partitioning the resources. This makes it more convenient to run an single-application OS alongside a general-purpose OS for other applicatoins.
- Query compilation [5, 4] allows database engine code to be fine-tuned for individual queries

Testing prototype

- in-memory, columnar, chunk-oriented operators
- embedded in Nautilus or as application in Linux
- 16-core x86_64 AMD EPYC with 4 NUMA nodes
- **TODO: version of linux kernel**

Preliminary results I - Basic Array Operations

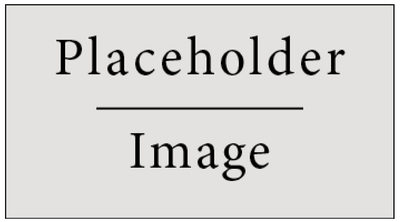
Basic array operations (within a chunk) perform better in the specialized runtime.



Box-and-whisker graph showing the performance (measured in cycles) and uncertainty

Figure 2:Nautilus can run array operations faster than Linux.

Preliminary results I - Deterministic Performance



TODO: Line plot with $\pm\sigma$ region shaded.

Operation	Linux	Nautilus
TLB miss		1
Page fault		0
Context switches		0
Interrupts		0

TODO: measure linux with perf.

Table 1:Nautilus can run array operations faster than Linux.

Preliminary results II - query processing primitives

Some operator implementations perform worse in the specialized runtime. We are researching why this is, and if there is a way to beat Linux's performance.
TODO: Sort vary chunk size and num columns, parameters favorable to Nautilus
TODO: Select vary chunk size and num columns and domain size, parameters favorable to Nautilus
TODO: Create vary chunk size and num columns, parameters

Discussion

TODO: Since we are specialized we can improve the performance for specific workloads. If we are not better, we could switch to different customizations (chosen by the application-programmer).

Future Work

- Implement performance counters (hardware-specific).
- Test more operators.
- Test parallel operations.

Conclusion

The research is promising, but more work needs to be done. Future work should seek to understand and improve the preformance details such as TLB misses and data cache misses in order to beat a general purpose OS.
Making the process simpler from the application-programmer's perspective is an active topic of research.

References

[1] J. Giceva, G. Zellweger, G. Alonso, and T. Rosco.
Customized OS support for data-processing.
In *Proceedings of the 12th International Workshop on Data Management on New Hardware (DaMoN 2016)*, June 2016.

[2] 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.

[3] 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.

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

[5] 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.