NautDB: Towards a Hybrid Runtime for Query Processing

Samuel Grayson[1], Kyle Hale[2], Boris Glavic[2]
1 University of Texas at Dallas, 2 Illinois Insitute 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. Complex towers of abstractions behind generic interfaces in the traditional software stack are getting in the way of better performance from multi-core systems.
To mitigate this inefficiency, we built hybrid runtime kernel (based on the Nautilus Aerokernel) for the paralel execution compiled queries. We evaluate the performance of our specialized runtime against a general-purpose runtime, for which Linux will serve as our examplar.

Motivation

The goal of this research is to integrate the improvements specialized systems (unikernels) with the improvements in specialized DB (compiled queries) to achieve high-performance.

- Status quo
  - Too many abstractions makes high-performance difficult.
  - OS is not aware of the workload; app-programmer is.
  - DB systems already have specialized scheduling abstraction, mem. management abstraction.
  - Give up the flexibility and generality in exchange for performance.
- Specialized Hybrid Runtime
  - 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 our specialized 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 (by app-programmer). See Fig 1
- Compiled Queries
  - Could be pre-compiled or JIT-compiled.
  - The app-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.

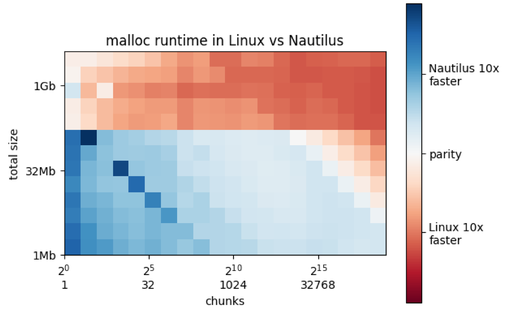

Figure 1:An alternative memory allocator can have a better performance for certain workloads.

Previous Work

- Customized OS support for data-processing: **TODO: I am not sure what specific improvements we have over this paper.**
- A Case for Transforming Parallel Runtimes Into Operating System Kernels: the authors use a specialized runtime and show performance benefit for runtimes such as Legion.
- **TODO: get a paper on Compiled 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

Preliminary results I

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

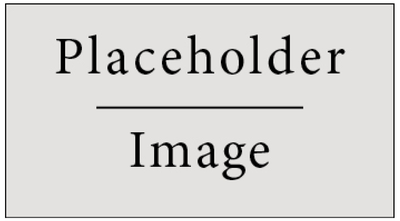

Figure 2:Nautilus can reverse an array faster.

Operation	Linux		Nautilus	
Allocate	1.0e9	1.0x	1.0e9	1.0x
Free	1.0e9	1.0x	1.0e9	1.0x
Copy (`memcpy`)	1.0e9	1.0x	1.0e9	1.0x
Copy (individually)	1.0e9	1.0x	1.0e9	1.0x
Reverse	1.0e9	1.0x	1.0e9	1.0x

Runtime measured in cycles according to `rdtsc`.

Table 1:Performance of Linux vs Nautilus on basic array operations averaged over 10 runs.

Preliminary results II

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.

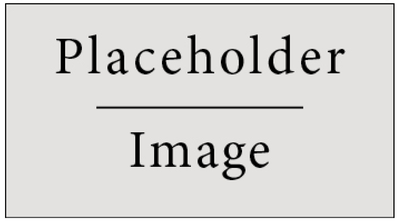

Figure 3:Sorting, for example, performs worse.

Discussion

TODO: discuss

Future Work

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

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.

References

TODO: References