NautDB: Towards a Hybrid Runtime for Processing Compiled Queries

Samuel Grayson
Department of Computer Science
University of Texas at Dallas
samuel.grayson@utdallas.edu

***Abstract*—General purpose operating and database system suffer from their generality which makes achieving optimal performance extremely hard, especially on modern hardware. The goal of this research is to integrate specialization techniques from the OS community (hybrid runtimes) and DB community (compiled queries) for high-performance query processing on modern hardware. We envision a system called *NautDB*, a hybrid dataflow runtime for executing compiled queries. As a first step towards our goal we use a simple prototype to evaluate the performance of compiled queries on Linux and run as a *Nautilus* hybrid runtime. Our results demonstrate that combining these specialization techniques has transformative potential for building the next generation (distributed) high-performance query processing systems and big data platforms.**

***Index Terms*—hybrid runtimes, light-weight kernels, compiled query processing, high-performance query processing**

I. Introduction

Both the operating system (OS) and database (DB) communities have strived to build general purpose systems which exhibit reasonable performance for a wide variety of applications and are user-friendly. However, the performance of general purpose operating and database systems suffers from being overly generic and from hiding implementation details behind multiple layers of abstraction [1]. Both communities have worked on achieving better performance on modern hardware without sacrificing ease of use. Specifically, OS researchers have introduced hybrid runtimes [2]–[5] as a means to give an application more immediate access to hardware and control over OS behavior while database researchers have studied query compilation to specialize a database execution engine to a particular query [6], [7].

The ultimate goal of this research is to integrate, for the first time, these specialization techniques from the OS community (hybrid runtimes) and DB community (compiled queries) for high-performance query processing. As a first step towards this goal, we build a testing prototype that uses hand-optimized query operator implementations and use this prototype to evaluate the potential performance benefit of running compiled queries as a hybrid runtime. Our preliminary results demonstrate that, even though this first version of our prototype is still quite naive, we can achieve better and more predictable performance through specialization.

This work was supported by NSF Award #1640864. Opinions, findings and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

II. Preliminary Evaluation

As a preliminary assessment of the potential of our idea, we have built an initial prototype consisting of manually optimized implementations of common database query operators. We then evaluate the performance of this prototype on a standard Linux distribution and embedded into the Nautilus aerokernel [2]. The purpose of this experiment is to evaluate how specialized implementations of OS functionality (e.g., memory management) and the immediate control over hardware might benefit evaluation of compiled query plans (like the ones already produced by modern query compilers used in main memory database systems [6]).

A. Testing Prototype

Our prototype supports several important database operators including selection, projection, and sort. Our implementation stores data in columnar chunks: the table is split into chunks of rows, which are stored column-at-a-time. Here we focus on the implementation of the sort operator, because this operator is used to implement other operators like aggregation and joins (merge-join). Our sort implementation uses counting sort to sort the data within a chunk and merged sort to sort across chunks. We have implemented a row-oriented and a column-oriented variant of this sort algorithm.

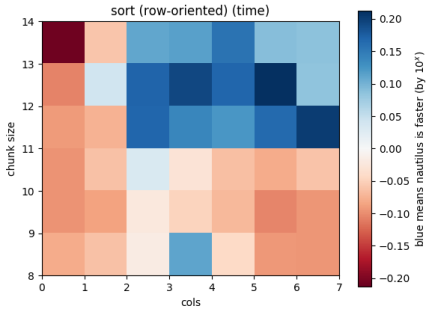
B. Experimental Setup

We ran our prototype on Linux (linux kernel 4.17.6) and as a hybrid runtime using Nautilus[1] [2], [3]. All experiments were run on a 16-core x86_64 AMD EPYC machine with 4 NUMA nodes. We sort a table consisting of 256 chunks varying the number of elements[2] per chunk and the number of columns in the table.

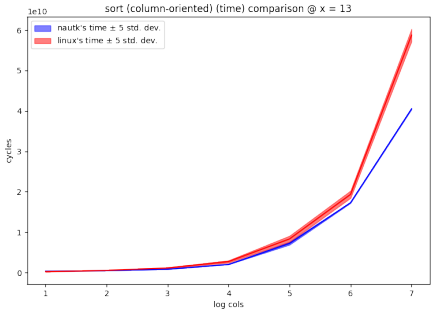
C. Experimental Results

The results of these experiments are shown in Figure 1. For larger number of columns and larger chunks, Nautilus (with relatively low developer-effort) outperforms Linux. This is because Nautilus has larger page size and incurs less TLB misses (see Table 1c). Furthermore, note (Figure 1b) that

[1]https://github.com/HExSA-Lab/nautilus - git commit `2fb4e52816`
[2]elements are stored as `uint32_t` values



(a) Row-oriented sorting for a table with 256 chunks and up to 128 columns



(b) Column-oriented sort measuring runtime and uncertainty for a fixed chunk size ($2^{13} = 8192$), varying the number of columns

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

(c) TLB and cache misses for row-oriented sorting of a 128 column table with 8192 elements per chunk

Fig. 1: Evaluating the performance of sort on Nautilus and Linux.

performance in Nautilus is much more predictable than in Linux. This effect is observed even in configurations where Linux outperforms Nautilus. This is because Nautilus does not have scheduling interrupts, so it avoids unpredictable detours which also leads to better cache performance.

D. Discussion

These preliminary result demonstrate that specialization allows us to tune both the OS and DB engine to a query workload and that compiled queries that consist of highly-specialized code benefit from a hybrid runtime environment. Furthermore, hybrid runtimes have much more predictable performance than general purpose OSes which will improve performance of bulk-synchronous parallel algorithms. Our prototype based on the *Nautilus* aerokernel does outperforms Linux in some parameter settings, despite our comparatively low developer-effort (Linux has been maturing for decades).

III. Our Vision for NautDB

As a long term goal, we envision to build *NautDB*, a hybrid dataflow runtime which executes tasks that are represented as compiled (machine) code. The frontend of NautDB will be a query compiler that translates high-level queries (SQL or another high-level and data-centric language) into compiled low-level tasks to be executed by the runtime. The present paper represents the first important step towards this goal.

IV. Conclusions and Future Work

This work demonstrated the potential of combining hybrid runtimes with compiled query processing. In future work, we plan to evaluate additional database operators, parallel execution of such operators, and evaluate how performance is affected by environmental noise from other applications running at the same time.

References

[1] J. Giceva, G. Zellweger, G. Alonso, and T. Rosco. Customized OS support for data-processing. In *Proceedings of the* 12^{th} *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* 24^{th} *ACM Symposium on High-performance Parallel and Distributed Computing (HPDC 2015)*, June 2015.

[3] K. C. Hale and P. A. Dinda. Enabling hybrid parallel runtimes through kernel and virtualization support. In *Proceedings of the* 12^{th} *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 2016)*, pages 161–175, Apr. 2016.

[4] K. C. Hale, C. Hetland, and P. A. Dinda. Automatic hybridization of runtime systems. In *Proceedings of the* 25^{th} *ACM International Symposium on High-Performance Parallel and Distributed Computing*, pages 137–140, June 2016.

[5] K. C. Hale, C. Hetland, and P. A. Dinda. Multiverse: Easy conversion of runtime systems into os kernels via automatic hybridization. In *Proceedings of the* 14^{th} *IEEE International Conference on Autonomic Computing (ICAC 2017)*, July 2017.

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

[7] 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.