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 scale (e.g., computing)
- In specific cases, give up flexibility and generality in exchange for performance.
- We prototype NautDB using a hybrid runtime kernel (based on the Nautilus Aerokernel) [2] 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 (kernel-like) => fewer context switches
- Partitions physical resources between general purpose OS and hybrid runtime [2] => can call general purpose OS where needed.
- Uninherited design gives programmer fine-grained control over ...
- Not time-shared => No context-switches or thread-switching
- Local interrupts => Faster and more predictable
- Single address space => Huge page-izes (2GB) => No TLB misses
- Memory allocation => 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 => less i-cache misses and branches
- Specialized data structures
- Combine code for multiple operators (pipeline) => optimizations within operators and less function calls
- Avoid interrupts

NautDB Prototype

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

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.

Configuration

- We vary the number of columns and the size of chunks.
- We aggregated over 20 trials.
- Hardware: 32-core X86, 64 AMD EPYC 7281 with 4 NUMA nodes
- OS kernel 4.17.6, Nautilus - git commit 201806250216.
- Nautilus has default configuration with debugging removed and extra devices disabled.

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

<table>
<thead>
<tr>
<th>Columns</th>
<th>Nautilus</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>64K</td>
<td>156.0K</td>
<td>189.0K</td>
</tr>
<tr>
<td>256K</td>
<td>480.0K</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Columns</th>
<th>Nautilus</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>64K</td>
<td>3.00K</td>
<td>0.92K</td>
</tr>
<tr>
<td>256K</td>
<td>11.00K</td>
<td></td>
</tr>
</tbody>
</table>

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 behavior of a component can be fine-tuned to workload characteristics.

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).

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

NautDB performance is more predictable than Linux, even when its raw performance is worse. Nautilus does not have scheduling interrupts, so it avoids unpredictable detours (e.g., context switches and global scheduling).

References


[2] NautDB Prototype: Since NautDB is an application-specific OS, it could be running in its own VM alongside a general-purpose OS.

[3] Nautilus: Make the development process simpler for the user

[4] Specialization allows us to beat Linux with relatively little effort.

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.

Next Steps

- Resolve algorithm from Linux where they outperform the existing Nautilus algorithm
- Make the development process simpler for the user

Our Vision: NautDB

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

- VMs: Since NautDB is an application-specific OS, it could be running its own VM alongside a general-purpose OS
- Frontend: compile queries into executable tasks for the database backend and takes care of higher-level scheduling and runtime-optimization tasks (e.g., task placement and global scheduling)
- Threading: will be pinned to CPU cores, they will run to completion without interruptions.

- Work distribution: Core are other workers that pull tasks from a local (federated) queue and execute them or management threads that pass tasks from clients to workers and handle results to clients, push statistics about task execution and resource utilization to clients, and garbage collect tasks from client queue.

- Global planning: will lead to better performance once we go parallel (e.g., bulk-synchronous model)
- 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.

- 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.

- 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.

- 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.

- 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.

- 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.