White Paper
Mar 31, 2024

DB^3: A Modern Database Kernel Technical Overview

1. Overview

DB^3 targets a modern database kernel. It started on March 2023, initially with one and later joined by two founding engineers. Each of them is a contributor or committer to significant OSS projects.

A modern database kernel is built upon resilience as its foundation. It should be owner developer friendly since contented developers lead to satisfied customers. Additionally, it must be born with performance and scalability, as we are capitalizing technology advancements over years. Lastly, it needs to maintain an open-minded approach to possibilities, acknowledging the dynamic nature of the world.

DB^3 is based on the relational$^+$ model, ensuring full ACID compliance by integrating cloud-level resilience. It accommodates complex batch, streaming, and PL/SQL queries and remains adaptable to resource limitations, operating seamlessly from a standalone machine to a cluster. Our demonstration of its capabilities includes benchmarks like TPCC, TPCDS, TPCH single machine, cluster, and its streaming variants.

Minimizing complexity is a promise for the DB^3 engineers. Starting from a clean state, through streamlined design with the advances in database, programming languages, open-sources and environments, DB^3 is standing on giants' shoulder while shedding numerous traditional redundancies. DB^3 achieves functionalities, quality and exceptional performance with significantly reduced codebase size compared to traditional implementations.

The majority of the code is written in C++23 and Rust. Differentiator components, such the storage engine, runtime and optimizer, are entirely new implementations from the ground up. Development progress is monitored through CI process. It comes with complete git logs, detailed commit messages and design documents to facilitate further developments.

1.1 Possibilities with DB^3

DB^3 is building on relational$^+$ model, which is an open foundation to embrace old new data processing scenarios present in future. It encourages more features, less products.

- Graph/Stream/Vector/… processors: SQL2023/Property Graph (PGQ) standard essentially says "just do it". Streaming is already in ANSI's formal meeting. Will Vector be next? These extensions keep pushing more features into relational$^+$ model's atlas, providing basis for new application like RAG.
- With the open mindsets and its extensible architecture, DB^3 hopefully enables new scenarios without compromise and with reuse. For example, supporting both OLTP and OLAP actually mutual beneficial: TP fallbacks to AP to handle complex queries, while TP sets a solid foundation for AP's data ingestion or meta-data management.
- DB^3 is built with a set of core libraries and multiple servers. With this structure, we may package it as a new database, or integrate into existing kernel like PostgreSQL extension, or as plug-in query accelerator following substrait protocols, or introduce DataFrame APIs for data scientists, and there are more.

DB3 is in active development. You can find the latest progress here: https://dbsup3.github.io/

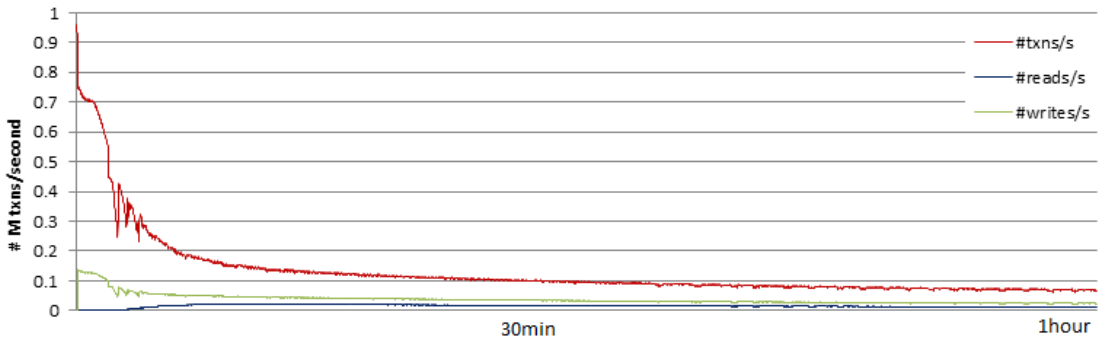
2. Scenarios

In this section, we assess DB3's capabilities across various scenarios using corresponding benchmarks. Except for distributed benchmarks, the machine we are using is the same desktop with an Intel i9-13900k processor (8 P-cores + 16 E-cores), 64G memory, and one P7000Z-4TB NVMe SSD drive (Official parameters read: 7.45G/860K IOPS, write: 6.75G/690K IOPS). The system is priced at $1500.

2.1 OLTP

We evaluate DB3's OLTP capacity by TPCC benchmark.

The TPCC five transactions and consistency checks are written in PL/SQL. The benchmark is configured with 8 warehouses. It runs using 8 hardware threads for 1 hour. The maximal data cached in memory is set to a fixed 4GB plus a fixed working set. The benchmark is running with isolation level default to SI. Logs are generated but not write out. At the end of the test, tpcc consistency conditions are verified for compliance. Note the running configuration is different from official one without key/think time, in order to exert more pressure on the server with arbitrary number of databases.



Configuration: 8C/4G fixed data cache/1-SSD, 8 Warehouses

So during the first 4 seconds, DB3 is running at full speed when data does not exceed the 4G limit. Later when data grow out of the limit, DB3 does cache replacement by reading from / writing to the SSD.

The experiment demonstrates DB3's transaction and PL/SQL JIT capacity. Even when compared to cutting-edge memory-only engines, which often evaluated via C/C++ APIs without a SQL layer, DB3 stands strong, achieving a transaction rate of 0.94M #txns/s.

Its performance declines smoothly when memory is insufficient. With only 1 SSD drive, it remains superior (0.06M #txns/s at 1-hour point) to traditional disk-based engines. There are some development work going on and we expect to push performance to 0.08M ~ 0.1M #txns/s at 1 hour point.

We also measured log replay speed, by enabling log write during TPCC run and replay these generated logs with DOP=8. Like TPCC run, it also limited by IO, the number below is the sustained speed before data spilling.

	GB/s	# MLogEntries/s
TPCC Log Replay	3.6	16.9

Configuration: 8C/1-SSD, logs generated by TPCC run

2.2 OLAP/Batch

We evaluate DB³'s OLTP capacity by TPCH and TPCDS benchmark.

In our testing of benchmarks at various scales, DB³ utilizes external Parquet files generated by DuckDB for storage, while DuckDB loads data into its native format. No index is created. TPCH tests are conducted with DOP=8. Note that due to limitations in its loading and operator support for spilling, DuckDB is unable to run at the 1000 scale (1T data) on this machine. Regarding the TPCDS benchmark, currently operates only with DOP=1, as efforts are underway to parallelize and optimize the necessary physical operators.

Following the practice of TPCH/TPCDS benchmark, we present both SUM and GEOMEAN of run time, where SUM indicates the overall performance and GEOMEAN measures the central tendency.

			DuckDB 0.92 (s)			DB³		
TPCH-30	100	1000 SUM	16	70	-	12	46	1110
		GEOMEAN	.51	1.9	-	.42	1.6	37.2
TPCDS-30 (DOP=1)		SUM	194			187		
		GEOMEAN	.72			.90		

Configuration: 8C/64G/1-SSD

The following distributed TPCH-1000 test is conducted on 4 aliyun ECS.R8I.2XLarge/PL2 instances (8vCPU/64G/IO-420M[i] /Network unknown). Each instance contains ¼ data in mounted drive. Notes: (1) To the best of our knowledge, GreenPlum is configured to with GUC adjustments[ii] and it does not support native parallel (DOP>1); (2) DB³ performance is limited by the IO maximal bandwidth at 420M/s with PL2 storage.

	GreenPlum 7.0 (s)	DB³
TPCH-1000 4xcluster (DOP=1) SUM	9449	3267
GEOMEAN	281	122
TPCH-1000 4xcluster (DOP=8) SUM	-	2316
GEOMEAN	-	86

Configuration: 4 * ECS.R8I.2XLarge/PL2 (420M/s)

Above experiments demonstrate DB³'s analytical processing capacity, supporting complex queries with details like NULL optimization, math overflow and type coerce all handled. It also demonstrates DB³ is flexible with resource constraints where it can run on single machine or cluster, can run with or without sufficient memory.

2.3 OLAP/Streaming

We evaluate DB³'s OLAP/Streaming capacity by a variant of TPCH benchmark for streaming data sources.

The TPCH-streaming benchmark is using the original TPCH queries. It emulates the largest two tables LINEITEM (180M rows) and ORDERS (45M rows) as streaming sources, while keeps others as original static tables. "Stream/<N>" means each streaming barrier consumes around 1/N data, where we

exercise an incremental materialized view maintenance (IVM). We can run 21 out of 22 queries, while the rest queries needs more development work.

	Batch(s)	Stream/1000
TPCH 30-Streaming SUM	11	55
GEOMEAN	.4	1.7

Configuration: 8C/64G/1-SSD, 21/22 queries

The following distributed TPCH-100 streaming TPCH is conducted on 4 aliyun ECS.C8I.2XLarge/PL1 instances (8vCPU/16G/IO-220M/Network unknown). Each instance contains ¼ data in mounted drive. DB^3 is running with DOP=1, as DB^3 is working on parallelize query with distributed deployment.

	Batch(s)	Stream/1000
TPCH 100-Streaming 4xcluster (DOP=1) SUM	108	451
GEOMEAN	5.9	12.4

Configuration: 4 * ECS.C8I.2XLarge/PL1, 15/22 queries

This benchmark demonstrates DB^3 can handle complex streaming queries in single machine or distributed, exhibits excellent low incremental maintenance efficiency. Recall in TPCC benchmark, we showed exceptional transactional throughput, so DB^3 itself can acts as a frontend to ingest events or state management.

3. Formulas

DB^3 follows the classic three major components design: optimizer, runtime and storage engine. For each one, we do a ground up new implementation with significant innovation and modernization. In this section, we will briefly discuss how each component works and some highlights.

3.1 Optimizer: DAG ↦ IRs

The query optimizer is responsible for translate query into optimized plan or code.

- After parser translates the PL/SQL into a parse tree, the optimizer applies a DAG oriented optimization. A DAG is considered superior to a tree for sub-plan sharing purpose.
- Following the Cascades framework, DB^3 implements a top-down rule based CBO. It further accelerates planning by early avoiding or pruning without compromising plan quality. The framework supports rule based transformations with logical/physic property enforcement and stats propagation. It supports both bounded and unbounded, singleton and coordinated query. Optimization rules include completed subquery decorrelation; cost based remote exchange marking, aggregation placement, CTE pooling/inlining, and numerous others.
- The optimized PL/SQL queries and control statements are either run in interpreter mode, or compiled into IRs and further lowering down layer by layer to instructions and runtime primitives for execution. This method comes with much better debug-ability than traditional LLVM based method without sacrificing performance.

- Optimizer extensively uses compile time code generation (CTCG) help to reduced 12K+ lines of boilerplate code for serde, clone, hash/equal etc. It is built on DB^3's managed memory infrastructure as easy as coding in GC supported language.

3.2 Runtime: (R + B + …) ×JIT⇌ Optimizer

Runtime is responsible for types and primitives, execution and optimizer feedback.

- A query can run in row, batch or mixed mode. Allowing mixed run benefits customers and internally best leverages each other's power. JIT is an accelerator for both modes.
- Realizing batch (bound) and streaming (unbound) query processing's duality, runtime implements a unified $V^{U|B}$ vector engine which shares native parallelization, spill support for both batch and streaming query. This largely reduces development efforts.
- Batch mode runtime is for performance: native parallel operators with light memory footprint; parallel scheduling effectively addresses skew and scales; cost based remote exchange placement maximally reduces data shuffling in cluster deployment. Besides that, it supports controls like pause/resume/cancel and other query resilience features.
- Runtime is flexible with resource constraints and is optionally stateless. It can complete OLTP queries in micro-seconds, and it can also coordinately run large OLAP queries in small machine or cluster. In distributed mode, plan is self-contained without catalog dependencies.
- A ground up rewrite gives runtime freedom to leverage CTCG, commercial friendly high quality OSS to accelerate development. For example, in internationalization, date/interval types and concurrent containers etc.

Here is a table summarizing row vs. batch mode:

	Row	**Batch (bounded + unbounded)**
Target	Mediocre	Analytic, Streaming/IVM
Functional completeness	Partial + fallback to batch	Partial + fallback to row
Resource constraints	Query is in memory but data cache can do swap	Minimal requirement on memory for both query and cache
JIT	Full stack	Partial
Scale up/out		Full
Query resilience		Yes
Error handling	Abort	Abort or partial restart
Pause/resume/cancel	Partial	Yes
Always-on profiling		Yes

3.3 Storage Engine: Local 凸 + Remotes 凹

The storage engine is responsible for transactions, data accesses and resiliency.

- DB^3 is designed to differentiate while seamlessly coordinate local and remotes: local handles transactions while remote storage is the final guarantee of capacity and resilience; Local processes read/write while remotes offload read-only; Local runs user centric logics while shifting maintenance to remote spot stances.

- Built on tiered low latency local + high throughput and highly reliable remote shared storage, it achieves performance and resilience while simplifies implementation. It designs to leverage shared storage and idle computational power in cloud-like environment.
- The local/remote protocol is TLA+ proved. It is ACID with maximal SI isolation level. It is supported by fully parallel SI maintenance, GC and write-behind-logging, recovery, memory and IO optimized indexing with memory size limit management. The coroutine based backbone hides I/O and synchronization latency without developer's awareness. This allows us to handle many connections easily.
- The catalog prioritizes reads over writes and diverges from the traditional schema design. The approach results in a much simpler implementation, including support for online DDLs.

4. Conclusion

DB^3 is designed to be resilient, friendly to its owner developers, born with performance and scalability, and maintains an open-minded architecture. The DB^3 team is looking for anything to take it to next.

---

[i] Each node has 600G storage, so according to the formula here (https://help.aliyun.com/zh/ecs/user-guide/essds), it has read throughput min{120+0.5*600, 750}=420M/s.

[ii] GreenPlum is configured with the following GUC modifications:

```
shared_buffers = 131072
max_statement_mem = 32GB
gp_vmem_protect_limit = 32GB
work_mem = 32GB
max_parallel_workers_per_gather = 0
```

Greenplum schema is without PK/FK. We have experimented to add PK/FK constraints which actually largely degrade its benchmark performance.