

您刚收到一个 Parquet 文件……接下来该怎么办？

探讨一个数据工程师或科学家的普遍困境：需要在没有服务器的情况下，快速查询一个大型 CSV 或 Parquet 文件。

点出现有工具的局限性：Pandas 在处理大型数据集时性能下降，而启动一个完整的数据仓库又显得“杀鸡用牛刀”。

引出问题：在轻量级内存工具和重型云数据仓库之间，存在一个明显的空白。



Pandas

DuckDB

Snowflake / BigQuery

在复杂的数据平台世界中寻求简洁

数据平台的选择往往令人不知所措。工程师们需要在成本、性能、易用性和供应商生态系统之间做出复杂的权衡。

这种复杂性常常导致过度设计，而许多分析任务实际上只需要一个更直接、更高效的工具。

What matters to you when choosing a data platform? i.e. Snowflake, Databricks, BigQuery, Redshift



discord-ian

You are way overthinking it. Here is the flow chart of how this decision is made.
Are you on Google? ➔ If yes BigQuery.
If you are on AWS or Azure, then ask are you a spark shop. ➔ If yes, then Databricks.
If no, do you have money and like a positive experience? ➔ If yes, then Snowflake. Otherwise, choose Redshift.

这张来自数据工程社区的截图，幽默地揭示了工程师们在选择大型数据平台时所面临的决策困境。我们需要一个更简单的答案。

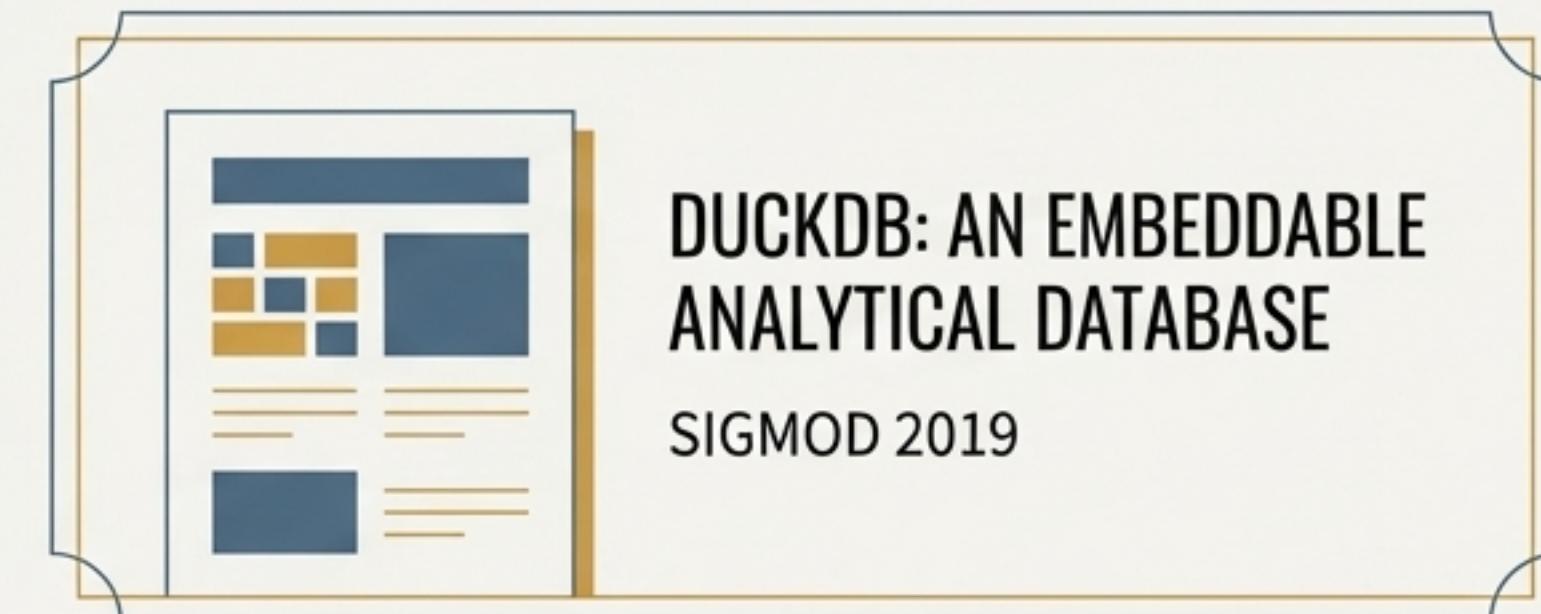
DuckDB：为分析而生的 SQLite



- 它是什么？一个高性能的**进程内 (in-process)** 联机分析处理 (OLAP) 数据库。
- **核心特性：**开源、列式存储、无需依赖、无服务器。
- **SQL 方言：**兼容 PostgreSQL，并增加了许多“生活质量”方面的改进。

历史渊源

- 源自荷兰 CWI（国家数学和计算机科学中心），由 Mark Raasveldt 和 Hannes Mühleisen 开发。
- 其诞生是为了解决 CWI 早期项目 MonetDBLite 的历史遗留问题，满足了数据科学家对一个轻量级、嵌入式 OLAP 数据库的需求。

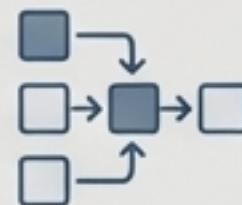


精心设计的架构：DuckDB 的核心设计原则

DuckDB 的卓越性能并非偶然，而是源于一系列深思熟虑的架构选择。这些原则共同构成了其高效分析引擎的基础。



Shared-
Everything:
共享一切架构



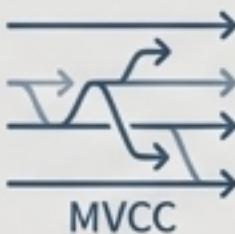
Push-based
Vectorized Query
Processing:
基于推送的向量
化查询处理



Morsel-driven
Parallelism:
基于数据块的并
行处理



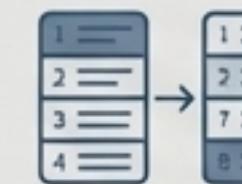
PAX Columnar
Storage:
PAX 列式存储



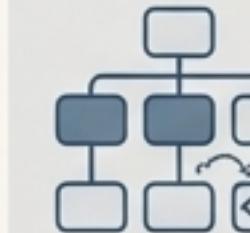
Multi-Version
Concurrency
Control (MVCC):
多版本并发控制



Precompiled
Primitives:
预编译原生函数



Sort-Merge + Hash Joins:
支持排序合并与哈希连接



Stratified Query
Optimizer:
分层查询优化器
(支持任意子查
询解嵌套)

引擎的革命：从“拉”到“推”的查询执行模型

问题

最初的拉取式 (pull-based) 向量化模型在实现复杂的并行处理（如同时执行多个流水线）时遇到了瓶颈。

解决方案

在 2021 年，DuckDB 转向了推送式 (push-based) 模型。在该模型中，每个算子自行决定是否并行执行，而不是依赖于一个中心化的执行器。

带来的优势

- **更精细的控制：**实现了算子暂停、向量缓存 (Vector Cache) 和扫描共享 (Scan Sharing) 等高级功能。
- **高效的 I/O：**支持背压 (Backpressure) 和异步 I/O，当缓冲区已满或等待远程数据时可暂停算子。
- **复杂的并行性：**显著提升了系统处理复杂并行操作的能力。

Merged

Switch to Push-Based Execution Model #2393

Mytherin commented on Oct 9, 2021

Files changed: 212

+6,097 -3,002

This PR implements #1583 and switches to a push-based execution model.

A summary of the changes:

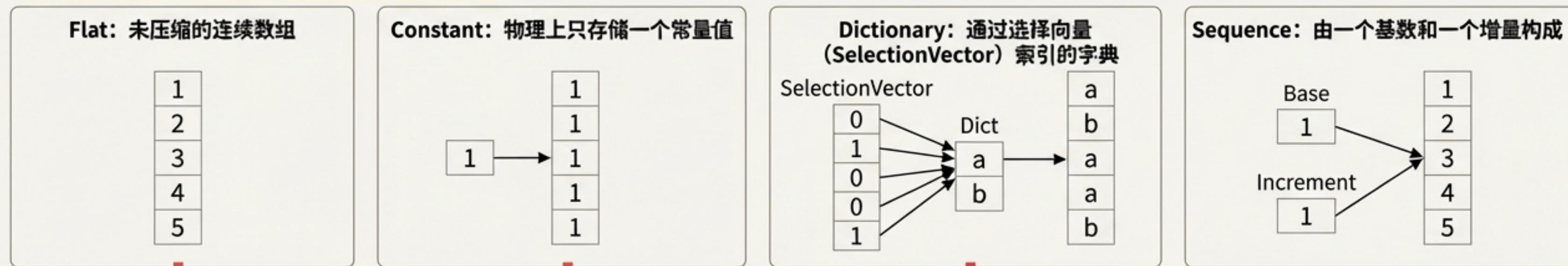
- All PhysicalOperators are reworked to use a push-based API,

Pull Request #2393: 这一历史性的合并记录了 DuckDB 向推送式执行模型的转变——这是其架构演进中的一个关键时刻。

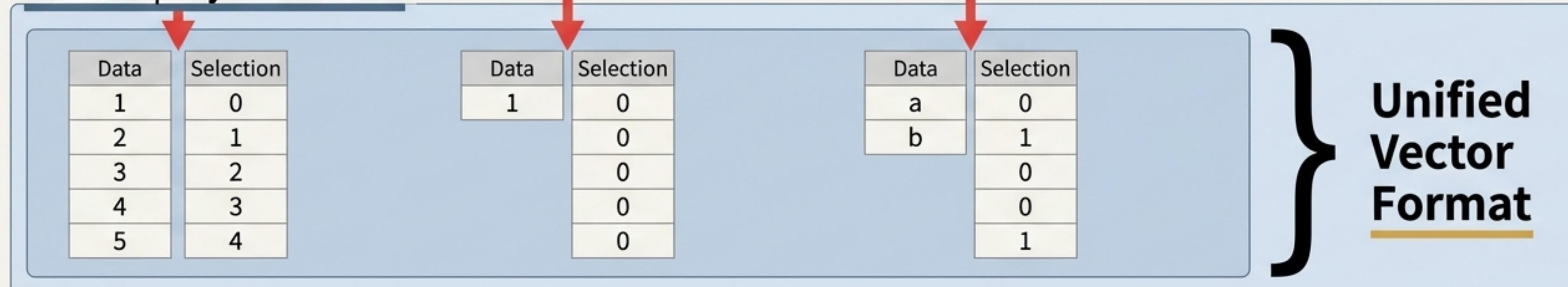
统一向量格式：直接在压缩数据上实现高效处理

DuckDB 采用统一的向量格式，使其能够**无需解压**就直接处理多种类型的向量数据，极大地减少了处理开销和为每种向量类型编写专门原生函数的需求。

逻辑向量类型 | Logical Vector Types



物理格式 | Physical Format



SQL 无处不在：直接查询文件与数据帧

DuckDB 的强大之处在于它能将 SQL 的分析能力直接应用在多种数据源上，无需繁琐的导入步骤。

```
import duckdb

# 直接对 Parquet 文件运行 SQL 查询
# 无需定义表结构，无需加载数据
results = duckdb.sql("""
    SELECT passenger_count, AVG(total_amount)
    FROM 'nyc_taxi_2019.parquet'
    GROUP BY passenger_count;
""").df()

print(results)
```

支持的数据源 | Supported Data Sources

- **文件**: Parquet, CSV, JSON 等。
- **数据帧**: Pandas, Polars, Apache Arrow。
- **远程文件**: 通过 `httpfs` 扩展直接查询 S3 或 HTTPS 上的文件。

这种将文件和内存对象视为“虚拟表”的能力，是 DuckDB 提升数据探索效率的关键。

强大的生态系统与灵活的扩展性

DuckDB 的设计理念是保持核心精简，并通过一个强大的扩展生态系统来增强功能。几乎所有功能，从文件格式支持到外部数据库连接器，都通过扩展实现。

与他人和睦相处

语言



数据工具



文件格式



扩展模型

Clean beautifully typeset table inspired duckdb_extensions()

extension_name	description	aliases
httpfs	Adds support for reading remote files in HTTP(S).	[http, https, s3]
postgres_scanner	Adds support for connecting to Postgres to Postgres.	[postgres]
spatial	Geospatial extension to spastatia.	
fts	Adds support for Full-Text Search in SynnriPL.	
json	Adds support for JSON in four JSON.	
parquet	Adds support for reading Parquet files in Parquet.	

httpfs: 直接查询云存储 (S3, GCS) 和 HTTP(S) 上的文件。

postgres_scanner: 无需 ETL，直接查询 PostgreSQL 数据库。

spatial: 提供地理空间数据分析能力。

fts: 全文搜索功能。

何时选择 DuckDB? 与 SQLite 的对比

DuckDB 被誉为“分析领域的 SQLite”，但它们的用途截然不同。理解它们的差异是做出正确技术选型的关键。

特性	SQLite	DuckDB
工作负载	OLTP (在线事务处理)	OLAP (在线分析处理)
存储模型	行式存储	列式存储
核心优势	快速的单行读写、高并发事务	快速的大规模聚合、扫描和连接
典型用例	移动应用后端、网站数据库、配置存储	数据探索、BI 报表、交互式分析、ETL
数据模型	优化用于事务一致性	优化用于分析查询性能

从本地到云端：MotherDuck 带来的混合执行能力

引出问题：DuckDB 在本地表现出色，但当数据量超出单机容量，或需要团队协作时，我们该怎么办？

解决方案：MotherDuck 并非要取代 DuckDB，而是对其进行优雅的扩展。它将 DuckDB 的简洁哲学带到了云端。



“ What if you could scale your local analysis to the cloud without changing your tool? ”

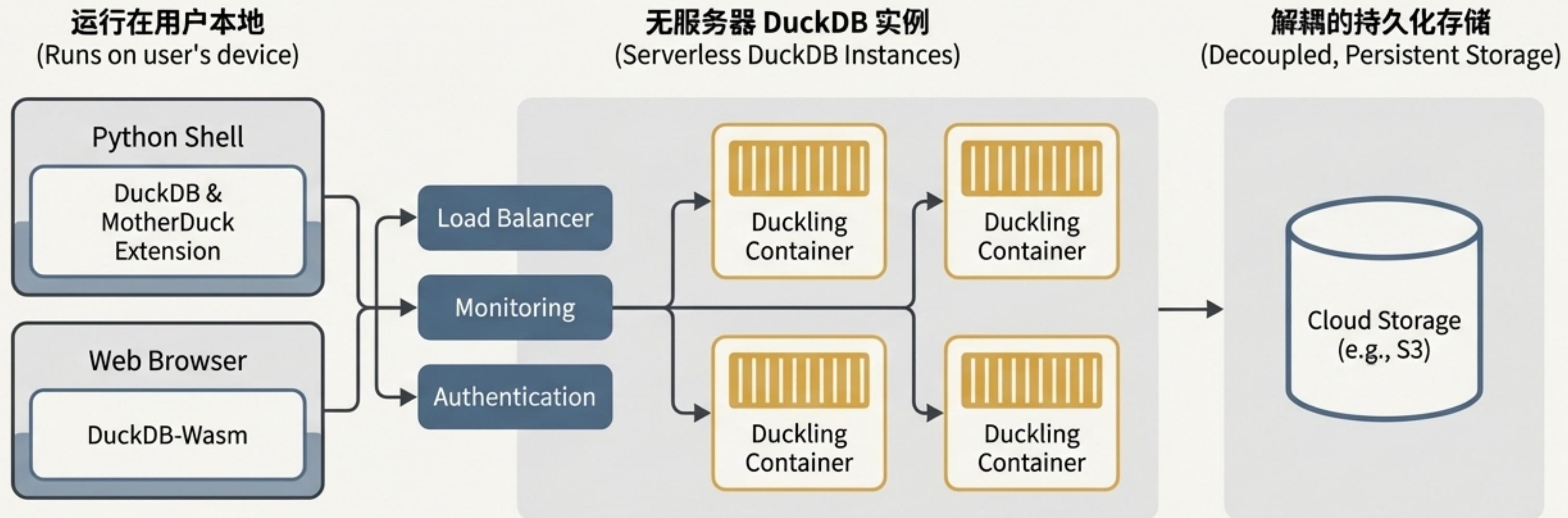
本地优先 • 您的笔记本电脑仍然是主要的计算客户端。

无缝扩展 • 当查询需要时，计算可以被无缝地“分派”到云端的无服务器实例上执行。

统一接口 • 对用户而言，无论是本地查询还是云端查询，都使用相同的 DuckDB API 和 SQL 语法。

MotherDuck 架构：本地与云端的智能协同

MotherDuck 的架构设计巧妙地结合了本地客户端的响应速度和云端计算的规模。



客户端始终在本地运行，确保了低延迟的交互体验，而计算和存储则可以根据需求弹性扩展。

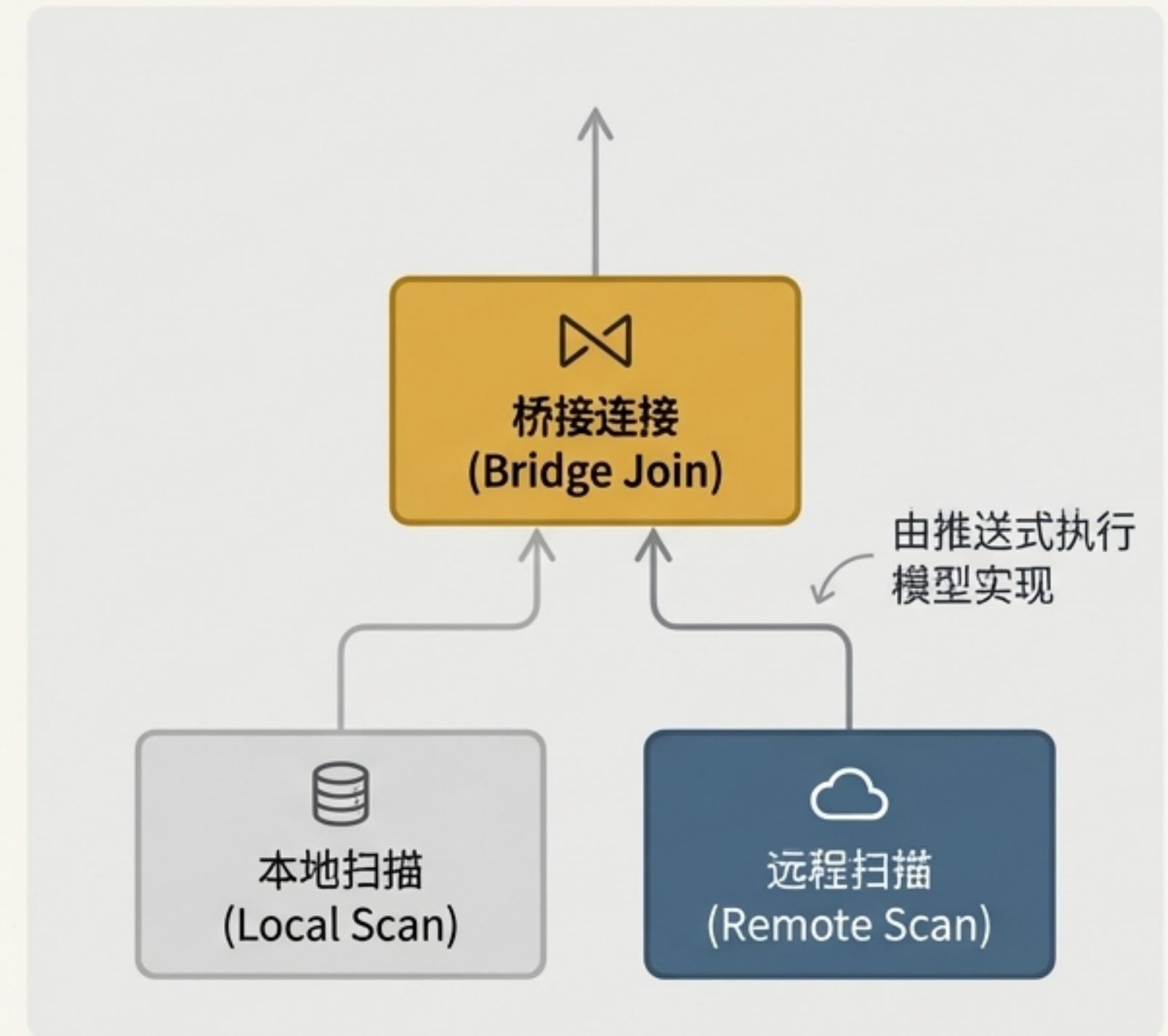
“桥接”算子：实现混合查询处理的魔法

工作原理

MotherDuck 的查询优化器能够感知数据的位置（本地或远程），并制定一个混合执行计划。

关键技术

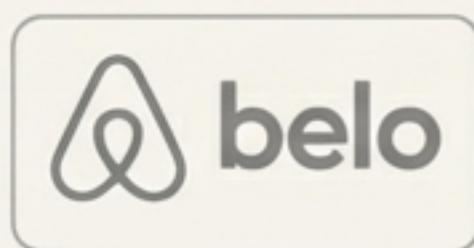
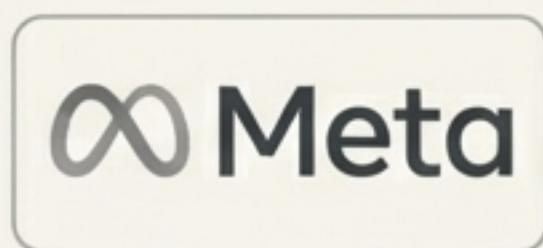
- **“桥接”算子 (Bridge Operators)**：在查询计划中引入新的算子，负责在本地和远程 DuckDB 实例之间高效地传递元组流。
- **利用推送式执行**：这个混合模型得以实现，正是得益于 DuckDB 的推送式执行引擎。其算子暂停 (operator pausing) 功能允许查询计划在等待远程数据时暂停本地部分，从而实现无缝的异步协作。
- **基于成本的优化**：本地实例会根据成本模型，智能地决定查询的哪个部分在本地执行，哪个部分推送到云端。



真实世界的验证：性能、成本与社区的认可

行业采纳

已被 Facebook (Meta), Google, Airbnb 等领先科技公司在生产环境中使用。



可量化的效率

Version	Taxi	On-Time	Lineitem	Notes	Date
DuckDB v0.2.8	15.3GB	1.73GB	0.85GB	Uncompressed	July 2021
DuckDB v0.3.3	6.9GB	0.23GB	0.32GB	Dictionary	April 2022
DuckDB v0.6.0	4.8GB	0.21GB	0.17GB	FSST + Chimp	October 2022
CSV	17.0GB	1.11GB	0.72GB		
Parquet (Snappy)	3.2GB	0.11GB	0.18GB		
Parquet (ZSTD)	2.6GB	0.08GB	0.15GB		

“大卫与歌利亚”的故事

“我们如何构建一个便宜 70% 的数据仓库
(从 Snowflake 到 DuckDB) ”

— 广为流传的 Reddit 帖子标题

这并非要取代 Snowflake，而是证明对于特定类型的分析负载（尤其是小于 1TB 的数据集），DuckDB 提供了数量级的成本效益。

随着版本的迭代，DuckDB 的原生存储格式在压缩效率上不断提升，其存储空间占用已优于多种 Parquet 压缩格式。

不断进化的旅程：DuckDB 的未来路线图

DuckDB 的开发由非营利性的 DuckDB 基金会管理，确保其永久开源。团队对未来有着清晰的规划。

计划中的特性（未来一年）

-  对 Rust 扩展的支持
-  改进对 Iceberg 和 Delta Lake 等湖仓一体格式的支持
-  支持异步 I/O
-  使用 `MATCH_RECOGNIZE` 用于模式匹配
-  并行 Python UDFs

未来工作 / 寻求资助

-  对 Go 语言扩展的支持
-  物化视图 (Materialized views)
-  时间序列与分区感知优化
-  支持 PL/SQL 存储过程
-  XML 读取支持

“Right place. Right time. Right problem.”

—— Andy Pavlo, Carnegie Mellon University

DuckDB 的成功在于它精准地解决了数据分析领域一个长期存在的问题。
它将数据库的强大功能与本地工具的便捷性完美结合。



速度 (Speed) :
向量化执行与列式存储。



简洁 (Simplicity) :
零依赖，进程内运行。



灵活 (Flexibility) :
强大的生态系统与扩展。



成本效益 (Cost-Effectiveness) :
开源，高效利用资源。