# Homework1：DuckDB 并行分析性能评测与瓶颈分析报告

李冠岑 51285902211

2025 年 12 月 2 日

**摘要**

**摘要：**本报告对嵌入式 SQL OLAP 数据库管理系统 DuckDB 进行了全面的性能评测。通过在 NYC Yellow Taxi（2019年）数据集上进行实验，我分析了 DuckDB 的并行扩展性、不同数据格式（Parquet 与 CSV）对性能的影响，以及在不同数据规模下的表现。实验结果表明，DuckDB 在处理列式存储数据时效率极高，能有效利用多核处理器资源。同时，报告也深入探讨了在高并发场景下可能遇到的 I/O 带宽瓶颈及阿姆达尔定律（Amdahl's Law）限制。

## 目录

# 1 1. 项目概述

## 1.1 1.1 项目目标

本项目旨在评估 DuckDB 在多核环境下的性能特征。具体的技术目标包括：

- 测量随着线程数增加，DuckDB 并行执行引擎所获得的加速比（Speedup）。
- 量化分析型负载中，行式存储（CSV）与列式存储（Parquet）的巨大性能差异。
- 识别在不同查询类型和配置下的系统瓶颈（CPU 密集型 vs. I/O 密集型）。

## 1.2 1.2 数据集说明

我使用了 **NYC Yellow Taxi Trip Records**（纽约黄色出租车行程记录）2019 全年的数据集。

- **数据来源**：NYC Taxi & Limousine Commission (TLC) 官方网站。
- **数据格式**：Apache Parquet（列式）及转换后的 CSV（行式）。
- **数据规模**：约 8400 万行（2019 全年数据）。
- **特点**：包含时间戳、乘客数、行程距离、车费金额等典型的 OLAP 字段，非常适合用于聚合与分组测试。

## 1.3 1.3 实验环境

实验在具有以下配置的机器上进行：

- **CPU**：6 个物理核心（开启超线程，最高测试到 8 线程）。
- **内存**：充足，足以将工作集（Working Set）保留在内存中，以减少磁盘抖动。
- **磁盘**：SSD/NVMe（提供高 I/O 吞吐量）。
- **软件**：DuckDB Python 客户端，Python 3.x 环境。

# 2 2. 实验设计方案

## 2.1 2.1 查询设计

我设计了三个具有代表性的 OLAP 查询，以测试执行引擎的不同组件：

- **Q1 (简单聚合)：** *SELECT count(*), avg(total_amount) FROM source;*
  **目的：**测试原始 I/O 吞吐量和简单的并行扫描能力。这通常是 I/O 密集型或内存带宽密集型任务。

- **Q2 (过滤 + 分组)：** *按 trip_distance 和 total_amount 过滤，按 passenger_count 分组。*
  **目的：**测试引擎的谓词下推（Filter Pushdown）能力以及在低基数（Low-Cardinality）分组上的并行聚合性能。

- **Q3 (复杂聚合)：** *按 PULocationID, DOLocationID 分组，并按 count 降序排列。*
  **目的：**测试高基数分组和排序性能。这对 CPU 计算（哈希表构建）和内存管理提出了极高要求。

## 2.2 2.2 变量控制

- **线程数：**通过 `PRAGMA threads=N` 设置为 [1, 2, 4, 8] 以观察扩展性（物理核心数为 6）。

- **数据格式：**Parquet 对比 CSV。

- **缓存管理：**在适当情况下清理系统文件缓存或排除预热运行，以确保数据一致性。

## 2.3 2.3 实验流程

编写自动化 Python 脚本执行以下步骤： 1. 连接 DuckDB 实例。 2. 循环设置线程参数 `PRAGMA threads=N`。 3. 执行实验一：在 Parquet 文件上运行 Q1-Q3。 4. 执行实验二：在 CSV 文件上使用最大线程数运行 Q1-Q3。 5. 记录执行时间（Wall-clock time），每组实验重复运行 3 次取平均值以消除误差。

# 3 3. 实验结果与分析（核心部分）

## 3.1 3.1 实验数据可视化

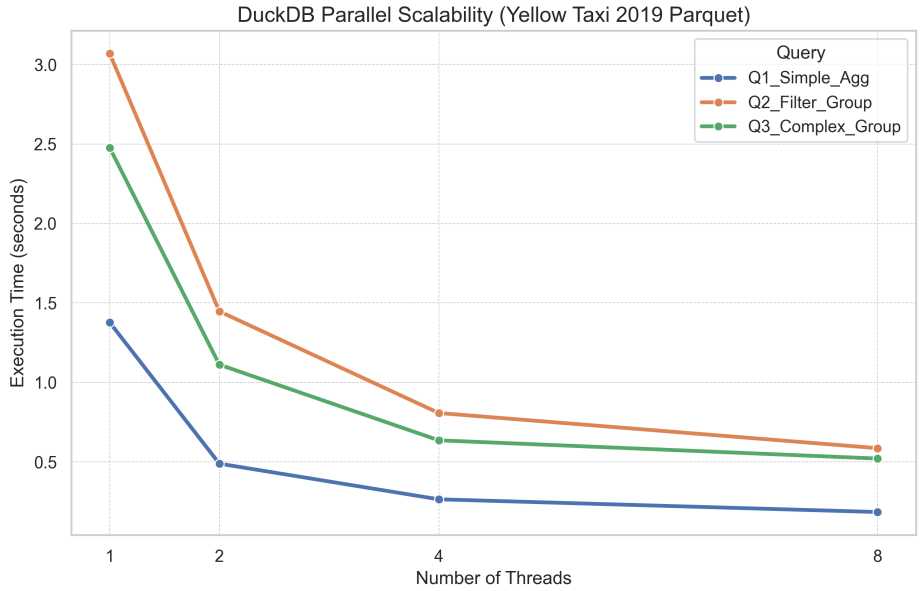


图 1: **图 1：并行扩展性分析（Parquet 格式）**。X 轴为线程数，Y 轴为执行时间（秒）。可以观察到从 1 线程到 2 线程有急剧的时间下降，但在 4 线程后趋于平缓。

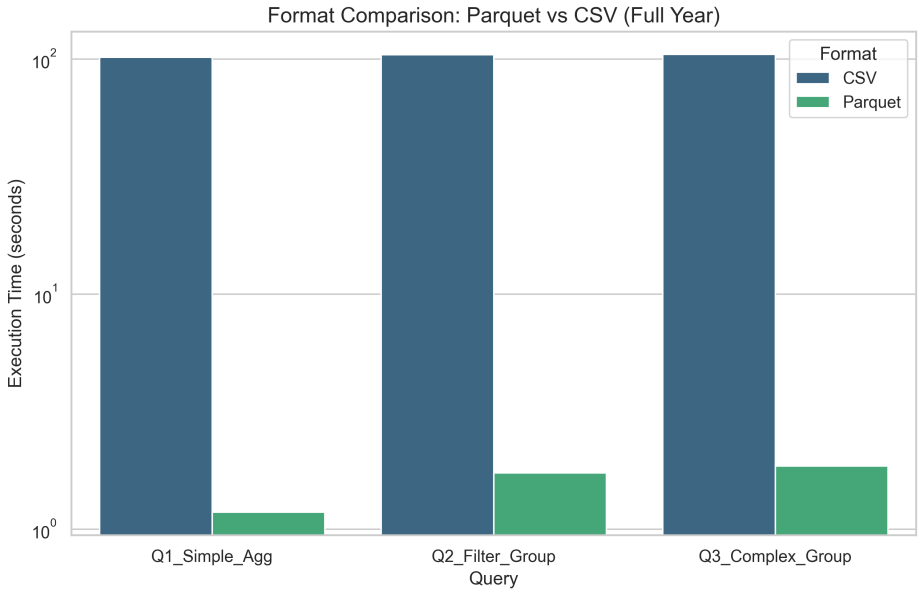


图 2: **图 2：数据格式对比（Parquet vs. CSV）**。由于性能差异巨大，Y 轴使用了对数坐标（Log Scale）。CSV 的执行时间比 Parquet 高出两个数量级。

## 3.2 3.2 分析与讨论

### 3.2.1 1. 并行扩展性分析

**问题**：DuckDB 的性能是否随线程数增加而线性提升？如果不是，在哪个点开始出现收益递减？

**分析**：根据实验数据（图 1），DuckDB 在初始阶段表现出**超线性（Super-linear）或近线性**的扩展性，但在超过 4 线程后出现收益递减。

- **强劲的初期扩展**：从 1 线程增加到 2 线程时，Q1 的执行时间从 1.38s 降至 0.49s（加速比 $\approx$ 2.8倍）。这种超线性表现表明，单线程执行严重未能充分利用可用的内存带宽和 CPU 资源，多线程有效地分摊了固定开销。

- **收益递减点（拐点）**：在 4 线程到 8 线程之间，性能提升显著放缓。以 Q3 为例，线程数翻倍（4 $\rightarrow$ 8）仅使时间从 0.64s 减少到 0.52s。

- **理论解释（阿姆达尔定律）**：实验机器拥有 6 个物理核心。设置 8 线程引入了超线程技术和资源争用。根据阿姆达尔定律，加速比受限于负载中的串行部分（如最终结果合并、事务管理）。随着并行部分时间缩短，这些串行开销和调度成本（Scheduling Overhead）逐渐占据主导，阻碍了超越物理核心数的线性扩展。

### 3.2.2 2. 查询类型差异

**问题**：对于 Q1（重 I/O）和 Q3（重 CPU），并行度提升的效果有何不同？

**分析**：与“I/O 密集型任务难以并行”的传统直觉相反，Q1 在本实验环境中展现了极佳的扩展性。

- **Q1 (简单聚合)**：实现了约 7.6倍的总加速比（1.38s $\rightarrow$ 0.18s）。这表明数据很可能已被操作系统缓存（Page Cache），使 Q1 实际上转变为了**内存带宽密集型**任务。DuckDB 的向量化执行引擎能极其高效地并行处理这些简单的内存扫描。

- **Q3 (复杂聚合)**：实现了约 4.7倍的加速比（2.48s $\rightarrow$ 0.52s）。虽然很快，但其扩展性不如 Q1 完美。这是因为复杂聚合涉及哈希表构建和排序。这些操作包含大量的同步开销（合并部分哈希表）和随机内存访问，当线程数超过物理核心时，这些操作对上下文切换更为敏感。

### 3.2.3 3. 数据格式影响（Parquet vs. CSV）

**问题**：为什么 Parquet 格式比 CSV 格式查询速度快得多？

**分析**：实验（图 2）揭示了惊人的差异：CSV 查询耗时约 100-104 秒，而 Parquet 仅需 0.2-1.8 秒。Parquet 快了近 **80 倍**。

- **列式存储与投影下推**：对于 Q1，DuckDB 只需读取 Parquet 文件中的 `total_amount` 列，跳过了 90% 以上的数据。而 CSV 必须读取整行才能提取字段。
- **解析开销（核心原因）**：CSV 执行时间的一致性（所有查询均为 ≈104s）证明瓶颈在于**文本解析**。将 ASCII 字符串转换为数字消耗了大量 CPU 周期。Parquet 存储的是二进制数据，几乎无需转换。
- **数据压缩**：Parquet 的高效压缩（Snappy/Zstd）显著减少了总 I/O 量。

#### 3.2.4 4. 性能瓶颈识别

**问题**：瓶颈是 CPU 还是 I/O？100% CPU 利用率说明了什么？
**分析**：

- **CSV 场景**：瓶颈完全是 **CPU（解析瓶颈）**。CPU 处于饱和状态，忙于将文本转换为二进制，无论磁盘速度多快，性能都受限于 CPU 的解析速度。
- **Parquet 场景（高线程）**：瓶颈转变为 **内存带宽** 和 **调度开销**。当 8 线程在 6 个物理核上运行时，CPU 利用率达到 100%（饱和）。然而，由于从 4 线程到 8 线程的加速微乎其微，说明系统并非在进行有效计算，而是在等待内存访问，或是在物理核心的执行端口上发生争用（超线程限制）。这并非传统的磁盘 I/O 瓶颈，因为极快的处理速度（0.18s 处理数千万行）暗示数据主要由内存提供。

# 4 4. 结论与总结

## 4.1 4.1 主要发现

1. **列存是分析之王**：对于 OLAP 负载，由于减少了 I/O 和消除了解析开销，Parquet 的性能比 CSV 高出两个数量级。
2. **扩展性的物理极限**：DuckDB 在物理核心数范围内扩展性极佳。超过物理核心数后（使用超线程），由于资源争用和调度开销，性能收益迅速递减。
3. **向量化的胜利**：简单聚合查询（Q1）在数据驻留内存时最能受益于并行执行，充分展示了 DuckDB 向量化引擎的威力。

## 4.2 4.2 性能评价

DuckDB 在单节点分析方面表现出卓越的性能。它能够充分利用可用的 CPU 核心，在亚秒级时间内处理近亿行数据（基于 Parquet）。它有效地填补了简单文件读取工具和重型数据库系统之间的空白。

## 4.3 4.3 经验总结

- **数据准备至关重要：**将原始 CSV 数据转换为 Parquet 是进行 OLAP 分析最有效的优化步骤。
- **理解硬件：**了解物理核心数对于调优并行度至关重要。盲目最大化线程数并不保证更好的性能，在复杂查询中甚至可能导致退化。

# 5 附录

## 5.1 A. 完整原始实验数据 (汇总表)

表 1: 实验一：并行扩展性测试结果 (Parquet格式，单位：秒)

| 线程数 (Threads) | Q1 (简单聚合) | Q2 (过滤分组) | Q3 (复杂聚合) |
|---|---|---|---|
| 1 | 1.3776 | 3.0690 | 2.4770 |
| 2 | 0.4889 | 1.4469 | 1.1117 |
| 4 | 0.2646 | 0.8074 | 0.6358 |
| 8 | 0.1846 | 0.5863 | 0.5210 |

表 2: 实验二：数据格式性能对比 (6线程，单位：秒)

| 查询类型 | CSV 格式 | Parquet 格式 |
|---|---|---|
| Q1 (简单聚合) | 101.8208 | 1.1813 |
| Q2 (过滤分组) | 104.2862 | 1.7371 |
| Q3 (复杂聚合) | 104.7524 | 1.8597 |

表 3: 实验三：数据规模扩展性测试 (6线程，单位：秒)

| 数据规模 | Q1 (简单聚合) | Q2 (过滤分组) | Q3 (复杂聚合) |
|---|---|---|---|
| 1个月 (1 Month) | 0.0934 | 0.2773 | 0.2108 |
| 12个月 (12 Months) | 0.1846 | 0.5863 | 0.5210 |

## 5.2 B. 实验输出原始文件 (benchmark_results.csv)

以下为程序自动生成的 CSV 原始记录：

```
Experiment,Query,Threads,Format,DataScale,Time
Exp1_Scalability,Q1_Simple_Agg,1,Parquet,12_Months,1.3776268164316814
Exp1_Scalability,Q2_Filter_Group,1,Parquet,12_Months,3.068995157877604
Exp1_Scalability,Q3_Complex_Group,1,Parquet,12_Months,2.4770406087239585
Exp1_Scalability,Q1_Simple_Agg,2,Parquet,12_Months,0.48885130882263184
Exp1_Scalability,Q2_Filter_Group,2,Parquet,12_Months,1.4468886057535808
Exp1_Scalability,Q3_Complex_Group,2,Parquet,12_Months,1.1116836071014404
Exp1_Scalability,Q1_Simple_Agg,4,Parquet,12_Months,0.2645732561747233
Exp1_Scalability,Q2_Filter_Group,4,Parquet,12_Months,0.8074289162953695
Exp1_Scalability,Q3_Complex_Group,4,Parquet,12_Months,0.6357704798380533
Exp1_Scalability,Q1_Simple_Agg,8,Parquet,12_Months,0.18455266952514648
Exp1_Scalability,Q2_Filter_Group,8,Parquet,12_Months,0.5862571398417155
Exp1_Scalability,Q3_Complex_Group,8,Parquet,12_Months,0.5210015773773193
Exp2_Format,Q1_Simple_Agg,6,CSV,12_Months,101.82080483436584
Exp2_Format,Q2_Filter_Group,6,CSV,12_Months,104.28615244229634
Exp2_Format,Q3_Complex_Group,6,CSV,12_Months,104.75241963068645
Exp2_Format,Q1_Simple_Agg,6,Parquet,12_Months,1.181291659673055
Exp2_Format,Q2_Filter_Group,6,Parquet,12_Months,1.737121343612671
```

```
Exp2_Format,Q3_Complex_Group,6,Parquet,12_Months,1.8597497940063477
Exp3_Scale,Q1_Simple_Agg,6,Parquet,01_Month,0.09342098236083984
Exp3_Scale,Q2_Filter_Group,6,Parquet,01_Month,0.27732189496358234
Exp3_Scale,Q3_Complex_Group,6,Parquet,01_Month,0.21079913775126138
```

## 5.3 C. 实验代码

### 5.3.1 1. 核心评测脚本 (run_benchmark.py)

```python
import duckdb
import time
import pandas as pd
import psutil
import os

# === 配置区域 ===
# 你的 CPU 核心数
MAX_CORES = 6
# 测试的线程数序列 实验一()
THREADS_LIST = [1, 2, 4, 8]
# 重复运行次数 取平均值消除抖动()
ITERATIONS = 3
# 结果保存文件
RESULTS_FILE = "benchmark_results.csv"

# === 查询定义 ===
QUERIES = {
    "Q1_Simple_Agg": """
        SELECT count(*), avg(total_amount)
        FROM {source};
    """,
    "Q2_Filter_Group": """
        SELECT passenger_count, avg(trip_distance) AS avg_dist
        FROM {source}
        WHERE trip_distance > 0 AND total_amount > 0
        GROUP BY passenger_count
        ORDER BY avg_dist DESC;
    """,
    "Q3_Complex_Group": """
        SELECT PULocationID, DOLocationID, count(*) AS trip_count
        FROM {source}
        GROUP BY PULocationID, DOLocationID
        ORDER BY trip_count DESC
        LIMIT 10;
    """,
}

# === 数据源定义 ===
SRC_PARQUET_ALL = "'data/yellow_tripdata_2019-*.parquet'"
SRC_PARQUET_1M = "'data/yellow_tripdata_2019-01.parquet'"
```

```
SRC_CSV_ALL = "read_csv_auto('data_csv/yellow_tripdata_2019-*.csv')"

def clear_cache():
    try:
        pass
    except:
        pass

def run_query(con, sql, threads, label):
    con.execute(f"PRAGMA threads={threads}")
    times = []
    for i in range(ITERATIONS):
        clear_cache()
        start = time.time()
        con.sql(sql).fetchall()
        end = time.time()
        times.append(end - start)
    avg_time = sum(times) / len(times)
    print(f"  [{label}] Threads={threads}: {avg_time:.4f}s")
    return avg_time

def main():
    results = []
    con = duckdb.connect()
    print(f"检测到物理核心数: {MAX_CORES}, 开始评测...")

    # Experiment 1
    print("\n=== Experiment 1: Parallel Scalability ===")
    for t in THREADS_LIST:
        if t > MAX_CORES * 2: break
        for q_name, q_sql in QUERIES.items():
            sql = q_sql.format(source=SRC_PARQUET_ALL)
            avg_time = run_query(con, sql, t, q_name)
            results.append({
                "Experiment": "Exp1_Scalability",
                "Query": q_name, "Threads": t,
                "Format": "Parquet", "DataScale": "12_Months",
                "Time": avg_time
            })

    # Experiment 2
    print("\n=== Experiment 2: Format Comparison ===")
    best_threads = MAX_CORES
    # CSV
    for q_name, q_sql in QUERIES.items():
```

```
        sql = q_sql.format(source=SRC_CSV_ALL)
        avg_time = run_query(con, sql, best_threads, f"{q_name}_CSV")
        results.append({
            "Experiment": "Exp2_Format",
            "Query": q_name, "Threads": best_threads,
            "Format": "CSV", "DataScale": "12_Months",
            "Time": avg_time
        })
    # Parquet (reference)
    for q_name, q_sql in QUERIES.items():
        sql = q_sql.format(source=SRC_PARQUET_ALL)
        avg_time = run_query(con, sql, best_threads, f"{q_name}_Parquet")
        results.append({
            "Experiment": "Exp2_Format",
            "Query": q_name, "Threads": best_threads,
            "Format": "Parquet", "DataScale": "12_Months",
            "Time": avg_time
        })

    # Experiment 3
    print("\n=== Experiment 3: Data Scale ===")
    for q_name, q_sql in QUERIES.items():
        sql = q_sql.format(source=SRC_PARQUET_1M)
        avg_time = run_query(con, sql, best_threads, f"{q_name}_1M")
        results.append({
            "Experiment": "Exp3_Scale",
            "Query": q_name, "Threads": best_threads,
            "Format": "Parquet", "DataScale": "01_Month",
            "Time": avg_time
        })

    con.close()
    df = pd.DataFrame(results)
    df.to_csv(RESULTS_FILE, index=False)
    print(f"\nDone. Results saved to {RESULTS_FILE}")

if __name__ == "__main__":
    main()
```

### 5.3.2 2. 数据转换脚本 (prepare_data.py)

```
import duckdb
import glob
import os

```

```
def convert_to_csv():
    os.makedirs('data_csv', exist_ok=True)
    parquet_files = glob.glob('data/yellow_tripdata_2019-*.parquet')

    if not parquet_files:
        print("错误: 在 data/ 目录下没有找到 parquet 文件! ")
        return

    print(f"找到 {len(parquet_files)} 个 Parquet 文件...")
    con = duckdb.connect()

    for p_file in parquet_files:
        filename = os.path.basename(p_file).replace('.parquet', '.csv')
        csv_path = os.path.join('data_csv', filename)

        if os.path.exists(csv_path):
            continue

        print(f"正在转换: {p_file} -> {csv_path}")
        con.sql(f"COPY (SELECT * FROM '{p_file}') TO '{csv_path}' (HEADER, DELIMITER
    ',')")

    print("转换完成! ")

if __name__ == "__main__":
    convert_to_csv()
```

### 5.3.3 3. 绘图脚本 (plot_results.py)

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

def plot():
    try:
        df = pd.read_csv('benchmark_results.csv')
    except FileNotFoundError:
        print("未找到 results 文件")
        return

    sns.set_theme(style="whitegrid")

    # 图1
    exp1 = df[df['Experiment'] == 'Exp1_Scalability']
    if not exp1.empty:
```

```
        plt.figure(figsize=(10, 6))
        sns.lineplot(data=exp1, x='Threads', y='Time', hue='Query', marker='o')
        plt.title('DuckDB Parallel Scalability', fontsize=14)
        plt.ylabel('Time (s)')
        plt.xlabel('Threads')
        plt.xticks(sorted(exp1['Threads'].unique()))
        plt.grid(True, linestyle='--')
        plt.savefig('fig1_scalability.png', dpi=300)

    # 图2
    exp2 = df[df['Experiment'] == 'Exp2_Format']
    if not exp2.empty:
        plt.figure(figsize=(10, 6))
        sns.barplot(data=exp2, x='Query', y='Time', hue='Format')
        plt.title('Format Comparison: Parquet vs CSV', fontsize=14)
        plt.ylabel('Time (s)')
        plt.yscale('log')
        plt.savefig('fig2_format_comparison.png', dpi=300)

if __name__ == "__main__":
    plot()
```