DBMS 性能评估：PostgreSQL 、openGauss 与文件操作的对比分析

SUSTech

CS213/Project 1: DBMS Performance Evaluation

火明贤

10/19/2025

摘要

本项目基于数据库原理课程（即 CS213）知识，进行简要的 DBMS 与文件操作对比分析和 PostgreSQL 与 openGauss 性能对比分析。使用课上的 movies 数据集，在 PostgreSQL ，Java 文件 I/O 和 openGauss 间进行了检索与更新测试。测试发现在简单操作与较小数据集上，由于数据库数据传输需读取硬盘与网络传输的时间(即 fetching time)，文件操作快于 DBMS，但也存在一些安全隐患。PostgreSQL 则在多方面优于 openGauss，但在并行运算时，由于 openGauss 有更强的自觉性使用多个 cpu（在允许时）以达到更快速度，openGauss 快于 PostgreSQL。

1.引言

- **项目背景**：在数据量爆炸式增长、应用无处不在的今天，DBMS 是确保数据被安全、高效、可靠地管理和利用的核心基础设施。没有 DBMS，企业和组织将无法有效处理其数据资产，也无法支持复杂的应用程序运行和关键的业务决策。作为初学者，自然地对 DBMS 之间的差别与 DBMS 和文件操作的差别应该有所探究和了解。
- **1.2 研究对象**：PostgreSQL、openGauss 和 Java 文件操作。
- **1.3 主要问题**：明确提出本项目要回答的核心问题：

  1. 相比于文件操作，DBMS 的独特优势是什么？

2. PostgreSQL 和 openGauss 哪个更好？

2.实验环境与设计

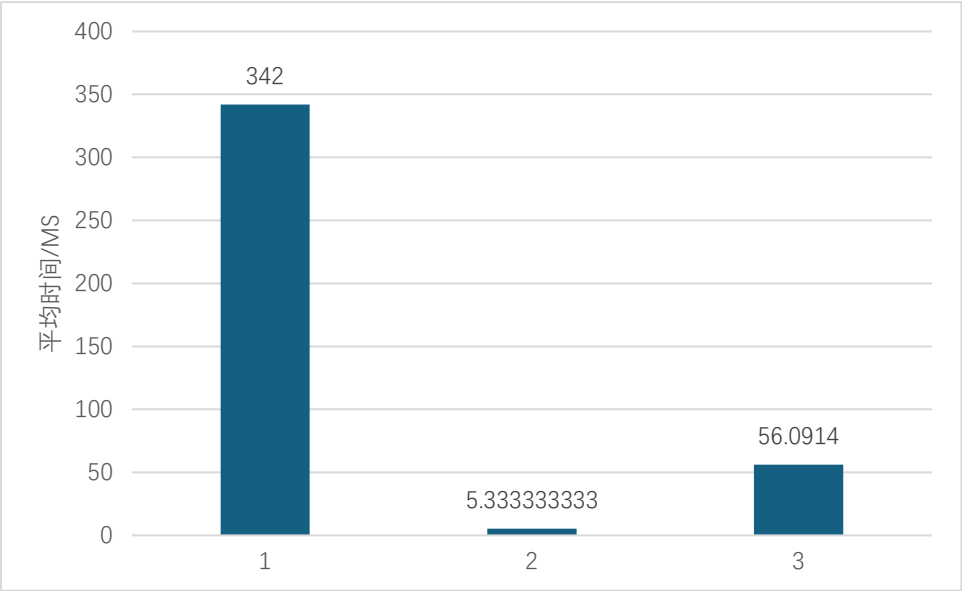
- **2.1 软件环境**：Windows、 PostgreSQL 、JDK 17、openGauss。
- **2.2 实验数据**：movies 数据 9.2k 行，随机生成数据 1000w 行
- **2.3 实验方法**：
  - **数据准备**：利用 DataGrip 将 movies 导出为 CSV 文件 。
  - **DBMS 操作**：使用 DataGrip 客户端执行 SQL，记录 execution time，fetching time，total time 。
  - **文件操作**：使用 Java 编写程序 ，通过 System.nanoTime() 计时，模拟 SQL 的检索 和更新 操作。

3.实验结果与分析

- **3.1 实验一：精确检索性能（select）**

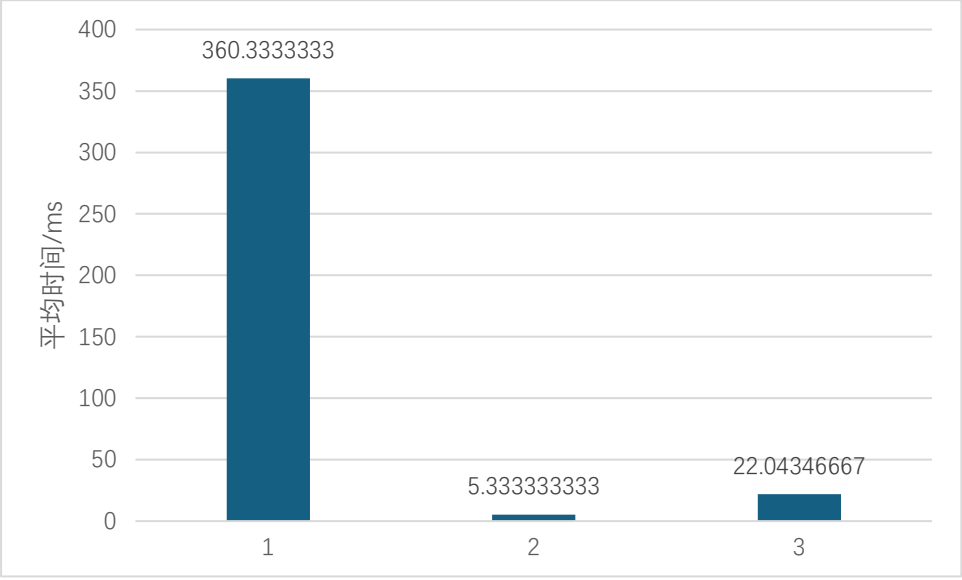
简述：本实验主要探究检索操作消耗的时间。在 DBMS 中多次用“select”操作探究检索 title 的时间损耗。在 java 中探究 I/O 检索 title 消耗的时间。

结果分析：忽略 DBMS 冷处理的情况下，分别测试三次对所有 title 列的检索所需时间，取平均时间绘图，得出如下表格



从左到右分别是 DBMS 总时间，DBMS 的 execution time，文件操作。

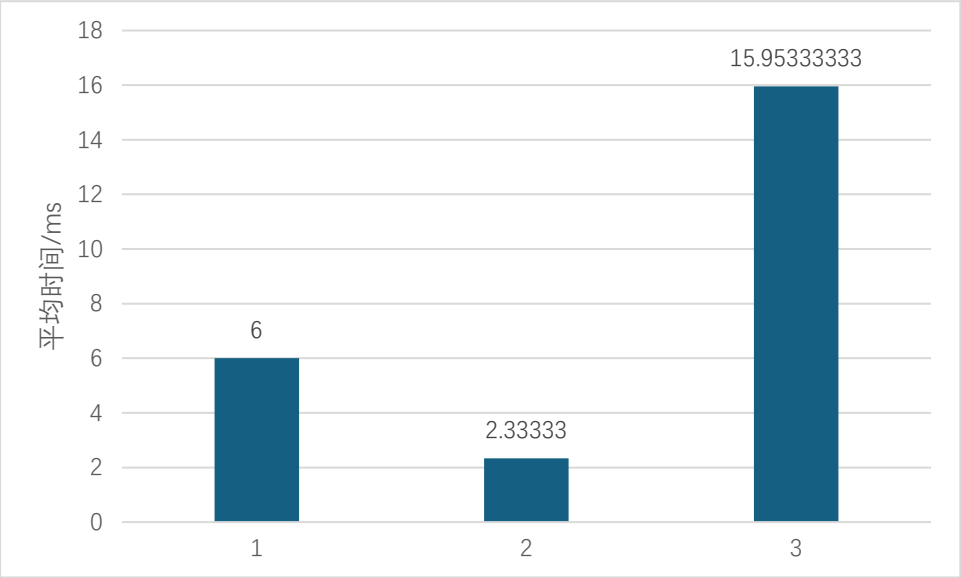
再检索所有含关键词“T”的 title 并记录，绘图



从左到右分别是 DBMS 总时间，DBMS 的 execution time，文件操作。可初步得知：总时间上 java 文件操作的检索十分迅速，但 DBMS 的 execution time 十分迅速，可知 fetching（即 DBMS 网络连接等时间）在此实验中对 DBMS 影响显著。

- **3.2 实验二：索引性能**

简述：本实验加入 DBMS 的检索功能，先创建 title 索引再测试检索含关键词“Dracula”的 title 所消耗的时间，并根据“Explain plan”得出检索逻辑变化，由于 fetching time 无法缓解，本次实验仅分析 execution time。

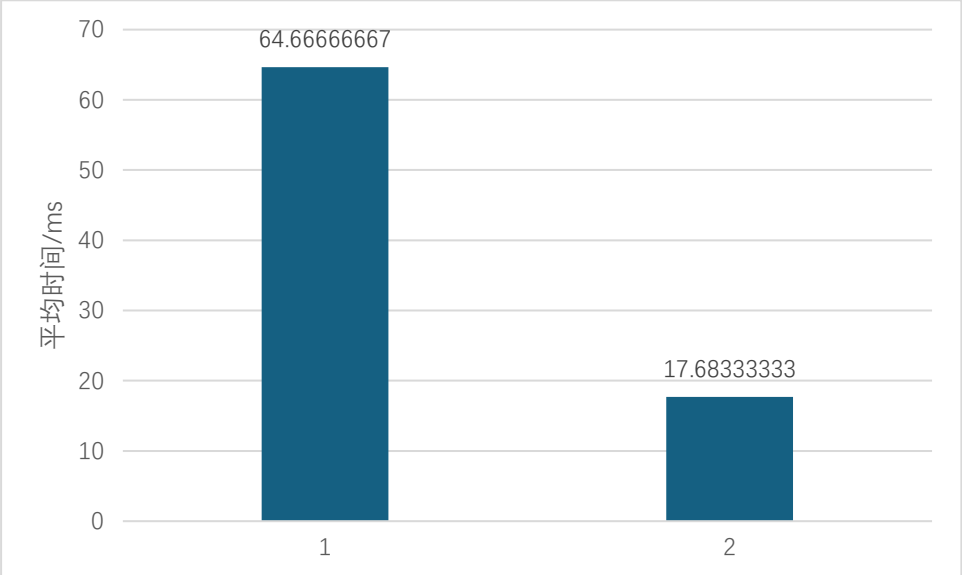


从左到右依次为，DBMS 无索引，DBMS 有索引，文件操作

可得：创建索引可显著提升检索时间，其内部逻辑是将 DBMS 的 Seq scan 变为 Bitmap Index Scan，来缩短时间，且这两个的 execution time 都少于 java 文件操作。

- **3.3 实验三：批量更新性能（update）**

简述：本实验用 DBMS 的“update”操作将 title 列中的所有“To”换为“TTOO”并记录时间。在 java 对文件进行相同的操作并记录时间。分别测量两次后取平均时间并绘图：

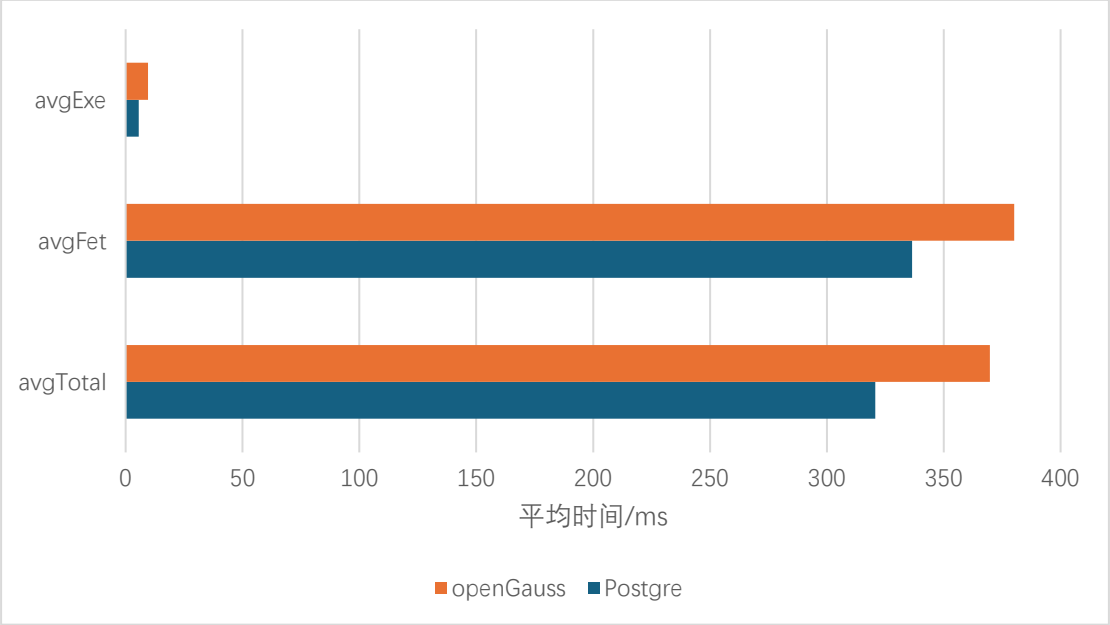


左边是 DBMS 时间，右边是 Java 文件操作时间。

从表中得知：Java 的文件操作在此数据集下显著快于 DBMS 的操作，这依旧可能是 fetching time 无法忽略的原因。同时在操作中发现 Java 操作对于文件的保护性有所欠缺，而 DBMS 则会弹出提醒，让我们警惕错误操作引发的数据破坏。

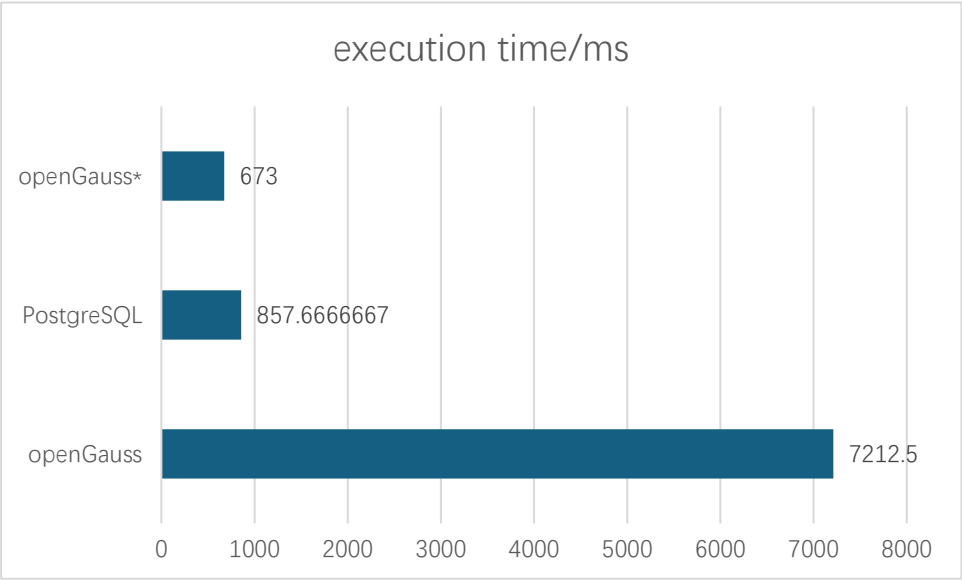
- **3.4 实验四：PostgreSQL 与 openGauss 性能测试对比**

简述 1：本实验分别在两个 DBMS 下多次运行课上学习的语句（具体见附录 1），记录时间，计算耗时平均值并绘图：



由实验数据可得：PostgreSQL 的运行速度快于 openGauss，且在 execution time 和 fetching time 都更优。

简述 2：用 create 语句随机生成 1000w 行的数据集，在此基础上测试两者面对超大量数据时的并行运算能力，并用“EXPLAIN ANLAYZE”检测能力。多次运行操作，记录使用 CPU 数与时间，计算平均耗时并绘图：



其中 openGauss*是允许多核处理后的结果。

可见默认状态下，PostgreSQL 快于 openGauss（限制单线程），但在允许多核处理后，openGauss 略快于 PostgreSQL。

4. 讨论：DBMS vs. 文件操作的优势

- **4.1 性能优势：极致的性能与优化**

① 缓存 ：DBMS 拥有智能的内存缓存池 (Buffer Pool)。第一次“**冷查询**”后，数据会留在内存中，后续的“**热查询**”会快非常多，具体体现在 execution time 大幅降低。但仍**无法解决** fetching time 的问题。

② 查询优化器 ：DBMS 会自动分析你的 SQL，选择**最高效**的执行计划（例如，是使用索引还是全表扫描对多列检索更划算）。

③ 索引 (Indexing)：（对应实验一）DBMS 使用 B-Tree 等高效数据结构。一旦**创建索引**，精确查询可以从 O(N)（全表扫描）降至 O(log N)（索引扫描），速度提升几个数量级。

- **4.2 安全性与一致性：ACID**

这是文件操作完全不具备的核心优势。DBMS 确保 UPDATE 操作是**原子性**的（要么全部成功，要么全部失败）。如果中途断电，数据库会自动回滚，数据不会损坏。而 Java 文件更新操作（读-写-替换），如果中途失败，可能会导致**数据丢失**或**文件损坏**。

这意味着 DBMS 系统默认很好地保护数据，而 Java 需手动改写逻辑，但效果可能**仍不及** DBMS。

- **4.3 开发与维护优势：**

① 声明式语言 (SQL)：对于 DBMS，只需要用“SELECT、UPDATE 、WHERE ...”告诉 SQL 想做什么，而不需要关心“具体怎么做”。Java 文件操作中需要考虑其**内部逻辑**并需要尽量简化逻辑，达到目的十分繁琐。

② 代码简洁性:(对应你的实验三) SQL 一行 UPDATE 语句，对比 Java 需要几十行代码（BufferedReader, BufferedWriter, File.move...）来实现，且要手动处理各种 IOException。

- **4.4 文件操作的优点：**

① 简单轻量：不需要安装和配置重量级的数据库服务。

② 可移植性:CSV/TXT 文件是通用格式,任何程序都可读取。

- **4.5 总结：**

① 面对日常操作的**大量数据**,DBMS 有着高效,可**维护性好，安全，简单**等特点，可以很好地覆盖大部分操作需要，且不用花费精力在内部逻辑上。

② 文件操作需要我们深刻理解数据操作**背后的逻辑**,面对大量数据时稍缓慢，但在**小数据集**中的操作快于 DBMS，且不用安装大数据库，不受“冷热查询”的影响，适用于只读或偶尔写入的场景。

5.PostgreSQL 与 openGauss 性能对比

- ## 5.1 PostgreSQL 的优劣

①**优点 ： 智能与成熟**。PostgreSQL 能**自动**分析查询，识别出这是一个昂贵的（heavy）查询，并**主动**启用了并行处理（Parallel Seq Scan)。用户什么都不用做，就得到了一个相当不错的性能（838 ms)。这展示了它作为一款极其成熟的数据库，其查询优化器在默认配置下是多么“智能”和“均衡”。

②**缺点 ： 性能上限可能稍低** 在这次特定的“短跑比赛”中，它最终的成绩（838 ms）被调优后的 openGauss（725 ms）击败了。这**暗示**在极限的、高度并行的分析查询（OLAP）场景下，它的性能可能不是最顶尖的。

- ## 5.2 openGauss 的优劣

①优点 ： 极高的**并行性能**上限 这是 openGauss 获胜的关键。在您手动“**唤醒**”它之后（SET query_dop)，它展现了惊人的爆发力，性能提升了 **3.75 倍**，并以 725 ms 的成绩反超了 PostgreSQL。它执行计划中的 dop: 12/12 也显示出它在“火力全开”、**压榨多核 CPU** 性能方面极具侵略性，这完全符合它面向**高性能分析**场景的设计目标。

②缺点 ： “开箱即用”体验差，需要**专业调优** 这是它最致命的弱点。您使用的 lite 版镜像，其**默认配置** query_dop = 1

（关闭并行）简直是“自废武功”。如果不是您深入排查，您会得出一个“openGauss 性能较差”的错误结论。这说明它不是一个“即插即用”的数据库，您必须**了解它的配置参数**，才能发挥它的真正实力。

5.3 谁更好？

①如果更看重通用、全能、极其可靠的数据库,或者应用场景是混合负载（OLTP + OLAP），比如网站后端、SaaS 应用、常规的业务系统，亦或者更看重**“开箱即用”的智能性**、成熟的社区生态，以及不想花大量时间去深入调优默认参数，那么PostgreSQL 更好。

②如果目标是构建一个高性能数据仓库（Data Warehouse）或纯分析平台(OLAP),核心需求是压榨多核 CPU(尤其是 ARM 架构，如华为鲲鹏）的全部性能，追求极致的查询速度，并且拥有专业的时间或团队来进行深入的数据库调优(Tuning),并且不介意它的配置与主流 PG 有所不同，那么openGauss 是一个不错的选择。

6.回答问题

- ### 6.1 相比文件操作，DBMS 的优势

DBMS 内部逻辑智能，不要求掌握操作背后的逻辑，代码简洁，安全性高，对新手更友好。

- 6.2 PostgreSQL 与 openGauss 哪个更好

  对于绝大多数开发者和通用场景，不追求极致效率的情况下，PostgreSQL 依然是更稳妥、更省心的选择。

附件一：DataGrip 代码（PostgreSQL 与 openGauss 测试代码相同）

```sql
select * from movies;

select count(*) from movies;

select m.title from movies m;

select movies.title from movies where title LIKE '%T%';

select m.title, m.country, m.year_released
    from movies m;

select m.title
    from movies m where year_released < 2010;

select m.title, m.country, m.year_released
    from movies m where year_released < 2010 and runtime > 100;

select count(*) from movies m where country = 'us';

select country, count(country), max(runtime) from movies where year_released > 2010 group by country;

select distinct
    case when m.country in('cn', 'hk', 'tw', 'jp', 'kr') then surname||' '||coalesce(first_name,'')
        else coalesce(first_name,'')||' '|| surname end as director
from movies m
   join credits c 1<->1..n: on c.movieid = m.movieid
   join people p 1..n<->1: on c.peopleid = p.peopleid
where c.credited as = 'D' and m.year released = 2015:
```

```sql
select m.title, m.country, m.year_released, m.runtime
from movies m
    join credits c on c.movieid = m.movieid and credited_as = 'D'
    join people p on p.peopleid = c.peopleid and p.gender = 'F'
where m.runtime = (select max(m.runtime)
                   from movies m
                   join credits c on c.movieid = m.movieid and c.credited_as = 'D'
                   join people p on c.peopleid = p.peopleid and p.gender = 'F');

SELECT title, REPLACE(title, 'To', 'TTOO') AS new_title FROM movies;

update movies
set title = replace(title, 'To', 'TTOO');

update movies
set title = replace(title, 'TTOO', 'To');

SELECT * FROM movies WHERE title = 'Dracula';

CREATE INDEX idx_title ON movies(title); --创建索引

-- 再次运行和步骤1完全相同的查询
SELECT * FROM movies WHERE title = 'Dracula';

DROP INDEX idx_title;
```

```sql
-- 1. 创建一个新表
CREATE TABLE parallel_test (
    id INT,
    group_id INT,
    payload TEXT  -- 我们将在这里放一些随机文本
);

-- 2. 插入 1000 万行数据 (这可能需要几分钟时间)
INSERT INTO parallel_test (id, group_id, payload)
SELECT
    i,
    (i % 1000),  -- 1000个不同的分组
    md5(random()::text) || md5(random()::text) -- 一些随机的 MD5 字符串
FROM generate_series(1, 10000000) AS i;

-- 3. 收集统计信息 (非常重要！这能帮助优化器做出正确决策)
ANALYZE parallel_test;
```

```sql
EXPLAIN ANALYZE
SELECT
    group_id,
    COUNT(*)
FROM
    parallel_test
WHERE
    payload LIKE '%abcde%'  -- 这是一个 CPU 密集型的字符串搜索
GROUP BY
    group_id
ORDER BY
    group_id;  --并未运用多个cpu

SHOW query_dop;  --return 1,即单线程

SET query_dop = 12; --达到分配的最大cpu
```