**PostgreSQL数据库并发访问测试**

**实验目的**

本实验旨在通过模拟多用户同时读取和修改数据库的并发场景，观察缓冲池管理在高并发环境下的性能表现

**实验环境**

操作系统：Ubuntu

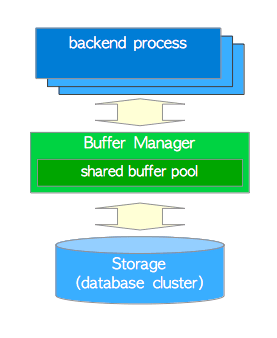
PostgreSQL：关系型数据库管理系统

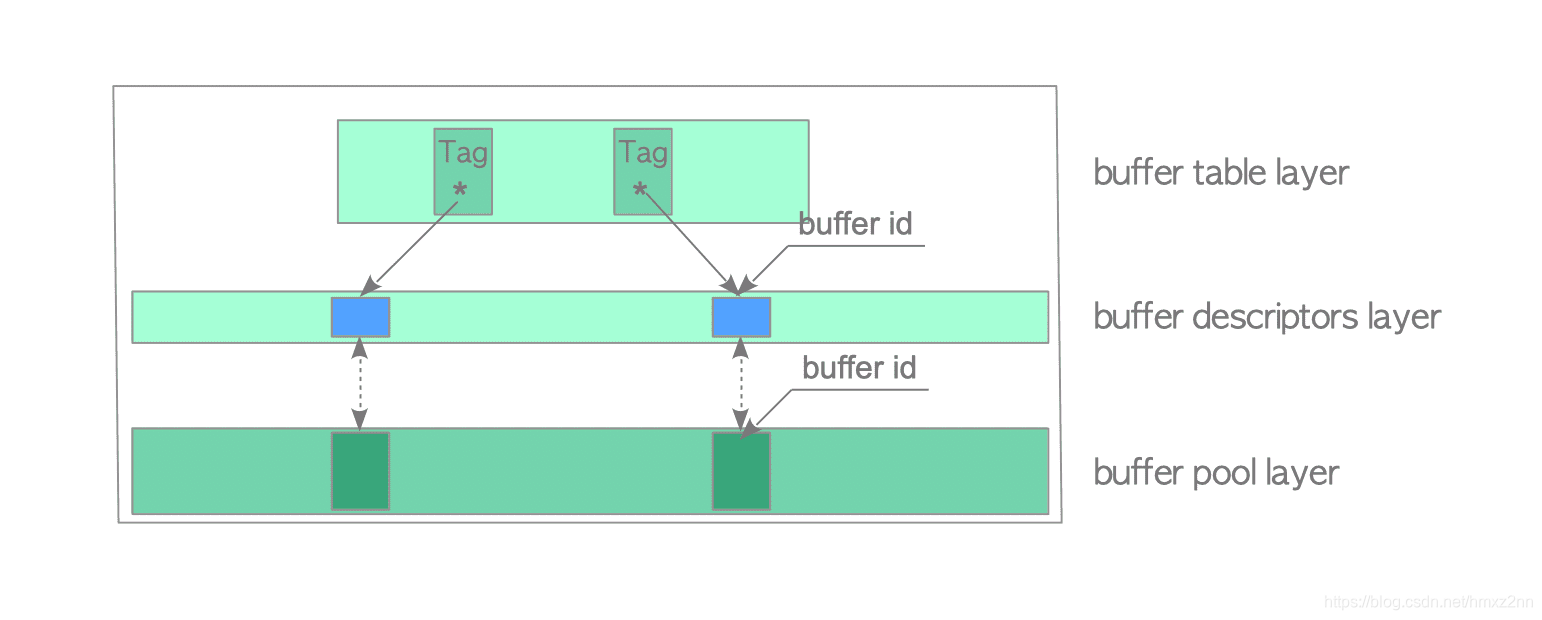
**实验要求**

对Docker和PostgreSQL有基本的了解

熟悉Linux命令行操作

具备基本的数据库操作知识





**实验内容和步骤**

**步骤 1: 在Ubuntu上安装PostgreSQL数据库**

1. 打开终端，并确保系统已更新： sudo apt update

2. 安装PostgreSQL： sudo apt install postgresql postgresql-contrib

3. 启动PostgreSQL： sudo su - postgres

psql

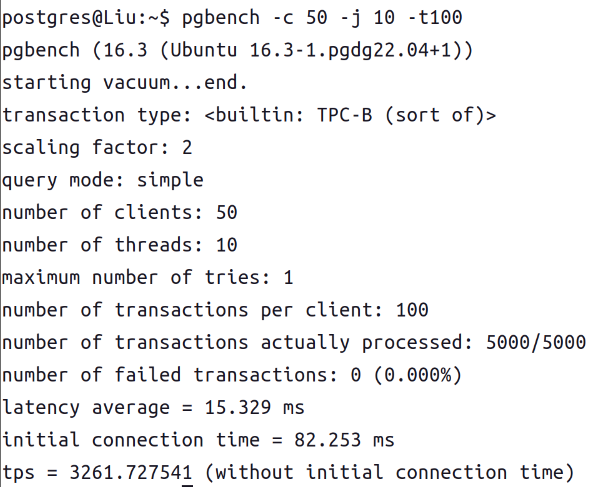
**步骤 2: 使用pgbench进行并发测试**

pgbench -i -s 2 -F 50

* ****-i 或 --initialize****  
  这个选项告诉 pgbench 进入初始化模式，它会在指定的数据库中创建表并插入测试数据。这是初始化数据库的必备选项。
* ****-s 或 --scale****  
  指定初始化数据的规模因子。默认值通常是1，意味着会创建大约100,000个账户。每增加1，数据量大约翻倍。例如，-s 10 会创建大约1000万个账户。
* ****-F 或 --fillfactor****  
  设置表填充因子。这会影响表中数据的实际存储方式，较低的填充因子可以预留空间以减少未来的页面分裂，但会增加索引大小。默认情况下，PostgreSQL 使用的填充因子是100。

pgbench -c 50 -j 10 -t100

* -c clients：指定并发客户端的数量。
* -j jobs：指定同时运行的事务线程数，这可以模拟更复杂的并发情况。
* -t transactions：每个客户端要执行的事务数,默认1。
* -T time：测试持续的时间（秒）。



* scaling factor: 2: 这个参数在初始化测试数据库时使用，决定了数据规模的大小。一个更大的scale factor会创建更多的数据表条目。在这个例子中，基础数据集被放大了2倍。
* query mode: simple: 测试采用的查询模式为简单模式。这意味着它将执行基本的、预定义的事务集合，没有复杂的查询逻辑。
* number of transactions actually processed: 5000/5000: 这显示了实际完成的事务数与计划的事务数。在这个测试中，计划并成功执行了5000个事务（50个客户端 \* 每个客户端100个事务）。
* number of failed transactions: 0 (0.000%): 表示没有事务执行失败。
* latency average = 15.329 ms: 平均延迟时间，也就是每个事务执行完毕所需时间的平均值，这里是15.329毫秒。
* initial connection time = 82.253 ms: 初始连接到数据库所花费的时间。
* tps = 3261.727541 (without initial connection time): 事务吞吐量（Transactions Per Second），表示每秒钟可以处理的事务数量。这个值是在不考虑初始连接时间的情况下计算得到的，为3261.73次/秒。

**步骤 3: 更改配置文件参数观察对并发访问的性能影响**

新开一个窗口

sudo vim /etc/postgresql/16/main/postgresql.conf

****shared\_buffers****: 这是最直接影响数据库缓冲区大小的参数，决定了有多少内存在服务器启动时被分配给共享缓冲区。增大这个值可以提高数据缓存的容量，减少磁盘I/O，但过大会占用大量内存，可能影响操作系统和其他进程。 修改方法：在postgresql.conf文件中调整此值，然后重启PostgreSQL服务。

****effective\_cache\_size****: 这是一个提示性参数，虽然不会直接分配内存，但它会影响PostgreSQL的查询规划器（planner），使其更好地估计可用缓存大小，从而可能影响索引扫描和顺序扫描的选择。 修改方法：同样在postgresql.conf中调整，并重启服务。

**Background Writer**是PostgreSQL后台进程之一，负责在不影响正常查询处理的情况下，将缓冲区中的脏块异步地写回到磁盘。Background Writer的行为可以通过调整与I/O相关的配置间接影响，例如：

* bgwriter\_delay：指定Background Writer休眠的毫秒数，默认是200ms。
* bgwriter\_lru\_maxpages 和 bgwriter\_lru\_multiplier：控制Background Writer每次唤醒时尝试写入的最大页面数和基于LRU策略的额外页面数。
* bgwriter\_flush\_after：指定在执行多少字节的写操作后强制执行fsync，以减少延迟。

Background Writer与数据库的并发访问和整体性能有着密切的联系，尽管它的直接职责是管理缓冲区与磁盘之间的数据同步。以下是几点说明其如何影响并发访问的：

1. ****减少检查点压力****：在没有频繁的checkpoint操作时，Background Writer异步地将脏页写回磁盘，有助于在检查点发生时减少必须写回的数据量。这减少了检查点引起的I/O高峰，从而降低了对并发查询处理的干扰。
2. ****提高写操作的响应速度****：当事务提交并修改数据时，这些修改首先被记录在共享缓冲区中。如果Buffer Manager立即执行磁盘写操作，可能会阻塞正在进行读或写操作的其他事务。Background Writer通过异步写磁盘，允许事务快速完成，提高了并发写操作的响应速度，减少了锁等待时间。
3. ****内存管理与LRU策略****：Background Writer还参与了基于LRU（最近最少使用）策略的缓冲区管理，帮助维持一个健康的缓冲区使用状态。通过选择合适的页面写回磁盘，它可以确保经常访问的数据保留在内存中，提高缓存命中率，这对于高并发读取场景尤为重要。
4. ****I/O平滑化****：由于Background Writer以较平稳的速率执行I/O操作，而不是集中爆发（如在checkpoint期间），这有助于减少I/O瓶颈，使得并发访问在总体上更加稳定和高效，避免了因突发的大规模I/O操作导致的系统响应延迟。

**更改shared\_buffers**



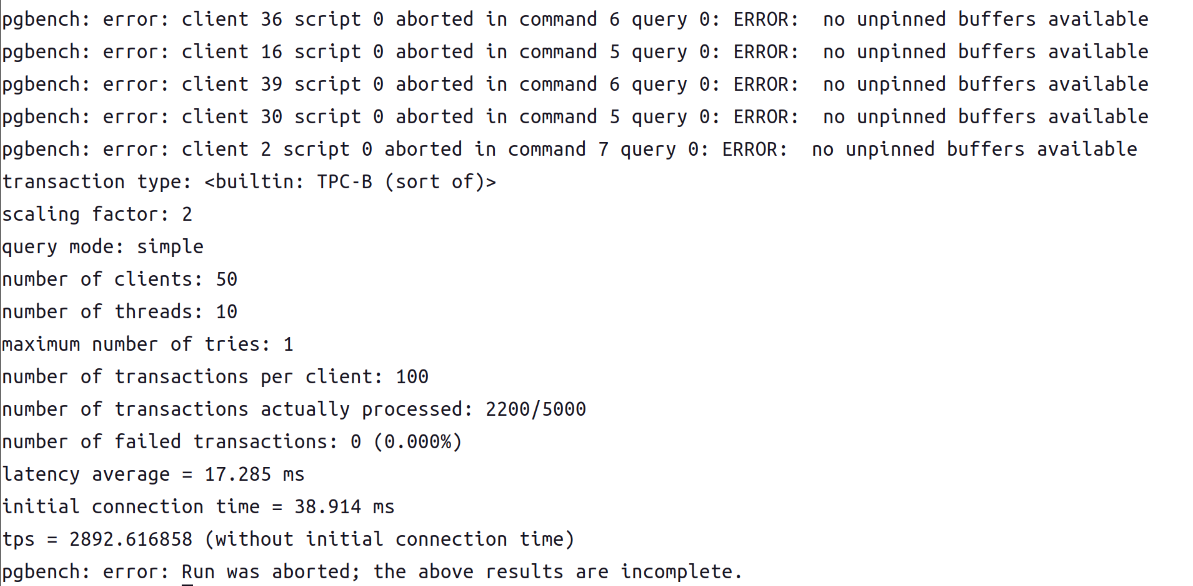
将其改为128kB试试

重启数据库

systemctl restart postgresql

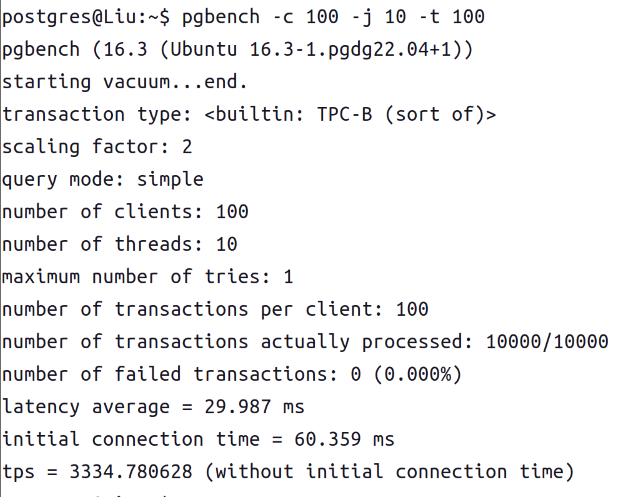
**注意，每次更改conf文件需要重启数据库**

继续之前的压力测试可以看出缓存不够



改为1MB，提高压力

pgbench -c 100 -j 10 -t 100

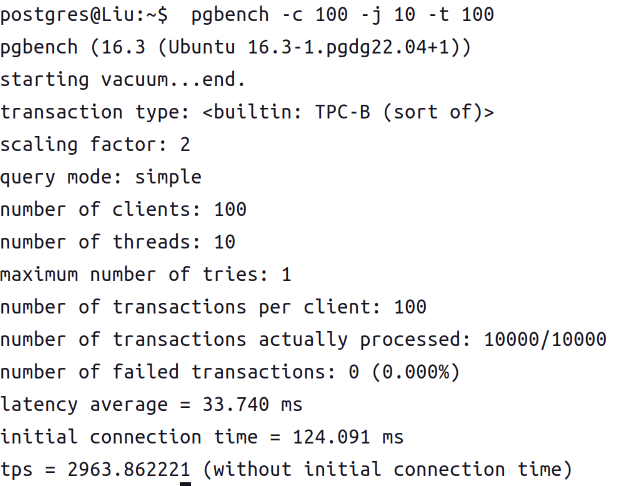


改回128MB，观察速度并没有多少提升，可知默认共享缓存还是很够用的

**更改**effective\_cache\_size****

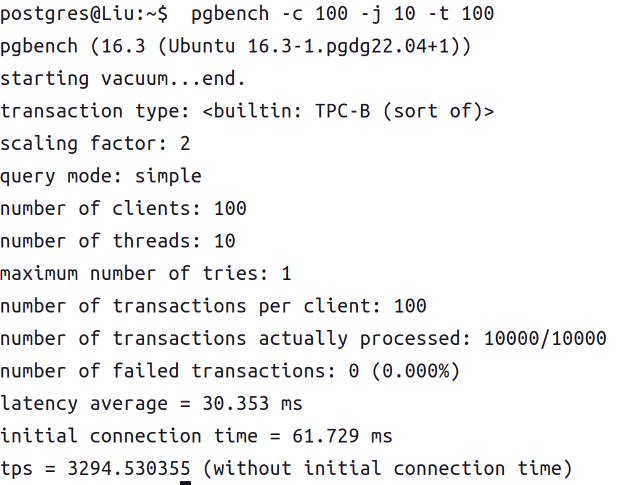


默认内存的一半，改为128MB后

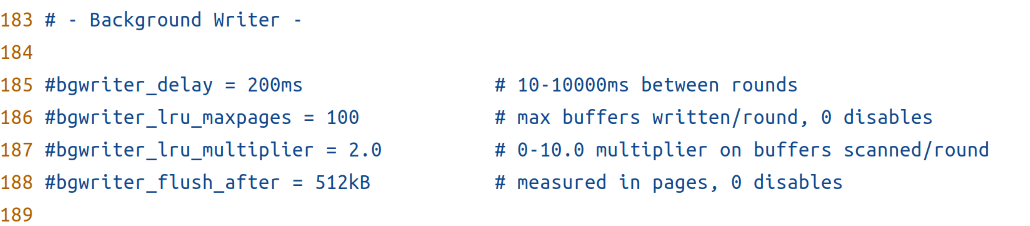


速度有所下降

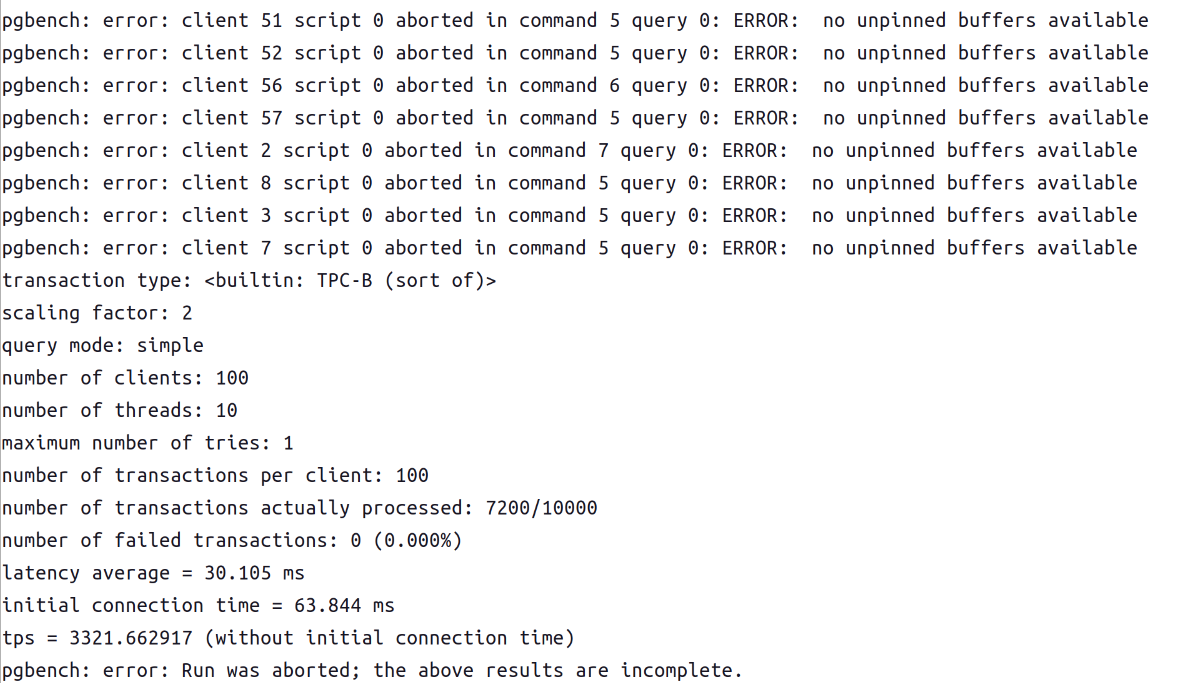
改为1GB后速度与默认值几乎无异



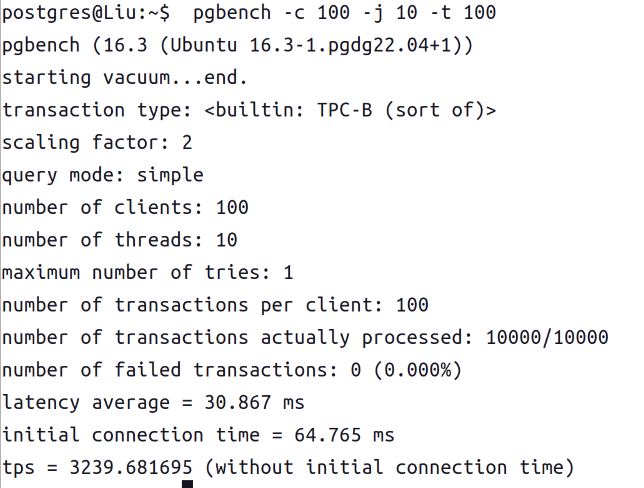
**更改Background Writer进程的参数**



将shared\_buffers改为512kB后缓冲区不够用



修改bgwriter\_lru\_maxpages为500，测试正常



**实验总结**

本次实验通过修改缓冲区管理器相关参数，调整PostgreSQL缓存管理参数可以显著提高数据库在高并发场景下的性能；