Practical 10: Concurrency scenario configuration and its impact on performance

GAHAN SARAIYA (18MCEC10), RUSHI TRIVEDI(18MCEC08)

18mcec10@nirmauni.ac.in, 18mcec08@nirmauni.ac.in

I. INTRODUCTION

Aim of this practical is to demonstrate any one concurrency scenario configuration and its impact on performance.

Table-level locks Available lock modes in PostgreSQL (used automatically; can also acquire explicitly)

Lock	Context
ACCESS SHARE	The SELECT command acquires a lock of this mode on referenced tables. In general, any query that only reads a table and does not modify it will acquire this lock mode.
ROW SHARE	The SELECT FOR UPDATE and SELECT FOR SHARE commands acquire a lock of this mode on the target table(s) (in addition to ACCESS SHARE locks on any other tables that are referenced but not selected FOR UPDATE/FOR SHARE).
ROW EXCLUSIVE	The commands UPDATE, DELETE, and INSERT acquire this lock mode on the target table (in addition to ACCESS SHARE locks on any other referenced tables). In general, this lock mode will be acquired by any command that modifies data in a table.
SHARE UPDATE EXCLUSIVE	Acquired by VACUUM (without FULL), ANALYZE, CREATE INDEX CONCURRENTLY, CREATE STATISTICS and ALTER TABLE VALIDATE and other ALTER TABLE variants (for full details see ALTER TABLE).
SHARE	Acquired by CREATE INDEX (without CONCURRENTLY).
SHARE ROW EXCLUSIVE	Acquired by CREATE COLLATION, CREATE TRIGGER, and many forms of ALTER TABLE (see ALTER TABLE).
EXCLUSIVE	Acquired by REFRESH MATERIALIZED VIEW CONCURRENTLY.

ACCESS EXCLUSIVE[1]	Acquired by the DROP TABLE, TRUNCATE, REINDEX, CLUSTER, VACUUM FULL, and REFRESH MATERIALIZED VIEW (without CONCURRENTLY) commands. Many forms of ALTER TABLE also acquire a lock at this level. This is also the default lock mode for LOCK TABLE statements that do not specify a mode explicitly.

Once acquired, a lock is normally held till end of transaction. But if a lock is acquired after establishing a savepoint, the lock is released immediately if the savepoint is rolled back to. This is consistent with the principle that ROLLBACK cancels all effects of the commands since the savepoint. The same holds for locks acquired within a PL/pgSQL exception block: an error escape from the block releases locks acquired within it.

II. Impact of Concurrency in PostgreSQL

If running two queries concurrently using two separate connections at the same time in PostgreSQL –

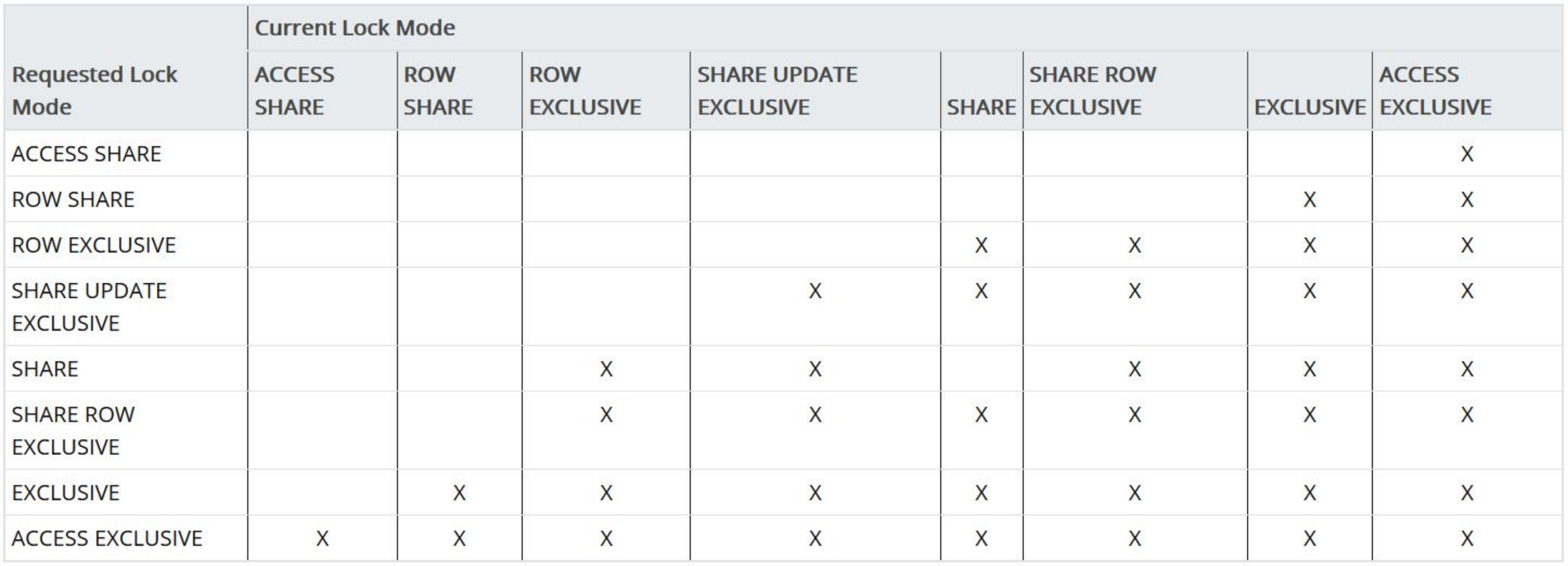
- ✓ If they were both using sequential scans of the same table then in PostgreSQL it'd be a huge performance win because of its support for synchronized sequential scans.
- ✓ If they shared the same indexes then they'd likely benefit from each others' reads in to cache.
- ✗ If they're independent and touch different data then they might compete for I/O bandwidth, in which case they might take the same amount of time as running sequentially. If the I/O subsystem benefits from concurrency (higher net throughput with more clients) then the total time might be less. If the I/O subsystem handles concurrency poorly then they might take longer than running them sequentially. Or they might not be I/O bound at all, in which case if there's a free CPU for each they could well execute as if the other wasn't running at all.

Running 1,000 non-trivial queries concurrently –

- PostgreSQL's own overheads in inter-process coordination, transaction and lock management, buffer management, etc. This can be quite a big cost, and PostgreSQL isn't really designed for high client counts
- Competition for working memory, cache, etc.
- Operating System scheduling overhead as it juggles 1000 competing processes all wanting time slices. Pretty minor these days, modern Operating Systems have fast schedulers.
- I/O thrashing. Most I/O systems have a peak performance client count. Sometimes it's 1, i.e. it's best with only one client, but it's often higher. Sometimes performance decreases again above the threshold. Sometimes it just reaches a plateau.

[1]Only an ACCESS EXCLUSIVE lock blocks a SELECT (without FOR UPDATE/SHARE) statement.

III. Summary



Requested Lock Mode	Current Lock Mode							
	ACCESS SHARE	ROW SHARE	ROW EXCLUSIVE	SHARE UPDATE EXCLUSIVE	SHARE	SHARE ROW EXCLUSIVE	EXCLUSIVE	ACCESS EXCLUSIVE
ACCESS SHARE								X
ROW SHARE							X	X
ROW EXCLUSIVE					X	X	X	X
SHARE UPDATE EXCLUSIVE				X	X	X	X	X
SHARE			X	X		X	X	X
SHARE ROW EXCLUSIVE			X	X	X	X	X	X
EXCLUSIVE		X	X	X	X	X	X	X
ACCESS EXCLUSIVE	X	X	X	X	X	X	X	X

PostgreSQL is an advanced RDBMS, with a solid foundation. The main service PostgreSQL implements to its users is a correct handling of concurrent operations. In the SQL model, that means handling both transactions and isolation levels.

A database server only has so many resources, and if you don't have enough connections active to use all of them, your throughput will generally improve by using more connections. Once all of the resources are in use, you won't push any more work through by having more connections competing for the resources. In fact, throughput starts to fall off due to the overhead from that contention. You can generally improve both latency and throughput by limiting the number of database connections with active transactions to match the available number of resources, and queuing any requests to start a new database transaction which come in while at the limit.

Contrary to many people's initial intuitive impulses, you will often see a transaction reach completion sooner if you queue it when it is ready but the system is busy enough to have saturated resources and start it later when resources become available.