Query Parallelism In PostgreSQL
Expectations and Opportunities

Dilip Kumar
(Principal Software Engineer)
Rafia Sabih
(Software Engineer)
Overview

- Infrastructure for parallelism
- Intra-query parallelism in v9.6
  - Parallel aware executor nodes
  - Performance on TPC-H
  - Limitations
- Parallelism enhancements in v10
  - More executor-nodes
  - Performance on TPC-H
- Take away
Overview of parallel query infrastructure

Groundwork for parallelism
- Dynamic background workers
- Dynamic shared memory
- Shared messaging capabilities
- Group locking
- Parallel context

Intra-query parallel support
- Parallel executor
- Parallel-aware nodes
  - seq scan,
  - joins, and
  - aggregates
Parallel Query Architecture

- **Gather** nodeGather.c

- **Tuple Queue Reader and DestReceiver** tqueue.c

- **Parallel Context** parallel.c

- **Parallel Executor Support** execParallel.c

- **Parallel-Aware Executor Nodes**
  - nodeSeqScan.c
  - nodeForeignScan.c
  - nodeCustom.c

- **State Synchronization**
  - dfmgr.c, guc.c, combocid.c, snapmgr.c, xact.c

- **Dynamic Background Workers**
  - bgworker.c

- **Error/Notice Forwarding**
  - pqmq.c

- **Group Locking**
  - lock.c

- **Shared Memory Message Queue**
  - shm_mq.c

- **Dynamic Shared Memory**
  - dsm.c, dsm_impl.c

- **Shared Memory Table of Contents**
  - shm_toc.c
Groundwork for parallelism

- Dynamic background worker (shm_mq)
  - Postmaster can launch the background worker processes at run time

- Dynamic shared memory
  - Allocate a chunk of memory that can be shared among co-operating processes
  - Shared memory table of contents
    - A simple scheme for carving DSM into numbered chunks

- Shared messaging capabilities
  - Shared memory message queue
    - For error/notice forwarding
    - Tuple queue reader and DestReceiver
Groundwork for parallelism

- **Parallel context**
  - Core toolkit for parallel operations
  - Launch a number of workers, establish “useful” state, run C code you specify and ensure timely shutdown.

- **State synchronization**
  - To ensure same GUC values, libraries, and transactions with same snapshot across workers

- **Group locking**
  - To solve the issue of undetected deadlock
  - Leader and its workers are treated as one entity for locking purposes
Intra-query parallel support

- Parallel executor support
  - Execute a plan by a set of worker
  - Pass instrumentation information to each worker

- Parallel aware executor nodes
  - Different behaviour when run in parallel or otherwise

- Gather
  - Collect results across all workers and merge them into a single result stream
Parallel access methods
- Seq scan is the only parallel access method
- No support for parallel index, index-only or bitmap-heap scan

Parallel joins
- NestedLoop and Hash joins are supported for parallel execution
- For hash-join, each worker prepares its own copy of hash-table
- Merge join cannot execute in parallel

Parallel aggregates
- Each worker performs partial aggregate and finalize aggregate is done by leader
Performance evaluation on TPC-H

- **Experimental setup**
  - IBM power7 box (popularly known as Hydra in community)

- **Parameter settings**
  - Max_parallel_degree = 4
  - Work_mem = 64 MB
  - Shared_buffers = 8 GB

- **Database setup**
  - Scale factor = 10
Performance evaluation on TPC-H

![Graph showing query execution time for TPC-H queries with and without parallel-query support. The graph compares the execution time for queries Q1 to Q22, with red bars indicating the time for queries without parallel-query support and blue bars indicating the time for queries with parallel-query support.]
Performance analysis: what’s missing…?

- Need parallel-index scan
  - Q6, Q14
- Need parallel bitmap-heap scan
  - Q4, Q15
- Need parallel merge-join
  - Q2, Q3, Q9, Q20
- Need parallel hash table build
  - Q3, Q5, Q7, Q8, Q21
- Need parallel subquery handling
  - Q2, Q22
Limitations

- Many forbidden operations
  - Suspended queries e.g. cursors, etc.
  - No serializable isolation level
  - Many parallel unsafe functions
    - Statements changing transaction state

- Restricted operations
  - Scan nodes with InitPlan or Subplan or Subquery scan nodes cannot leverage parallelism
  - No parallelism for temporary tables, CTEs, foreign table scans
  - Parallel restricted functions
What to expect in coming versions

- More parallel executor nodes
  - Access methods
    - Parallel index, index-only, bitmap-heap
  - Join methods
    - Merge join
    - Hash join with shared hash
  - Other
    - Gather-merge
    - Relaxation for nodes using uncorrelated sub-plan, init-plan
    - Improvements in parallel-append

- Parallel DDL/maintenance commands
  - Index-creation
  - Vacuum
Parallel index scan

Firstly, a worker will process the intermediate pages of B-tree and determine the starting leaf page where scan is to be started.

Next, all the workers start scanning the leaf pages block by block.

Finally, all the filtered tuples are gathered by the leader process.

This operator improves the performance significantly when the database is in-memory.

Similar mechanism is built for index-only scans.

Upcoming v10: at a glance
Upcoming v 10: at a glance

- Parallel bitmap heap scan
  - A bitmap scan fetches all the pointers from index in one go, sort them using in-memory “bitmap”, finally, visits the tuple in physical order
  - Bitmap will be created by a single worker
  - Next, all the workers will jointly scan the heap, page by page

- For further improvement, we can also build bitmap by multiple workers
Upcoming v 10: at a glance

- Parallel bitmap heap scan

Gather

Workers Planned: 2

-> Parallel Bitmap Heap Scan on foo

Recheck Cond: ((a < 100000) OR (b < 10000))

-> BitmapOr

-> Bitmap Index Scan on idx1
Index Cond: (a < 100000)

-> Bitmap Index Scan on idx2
Index Cond: (b < 10000)
Parallel Merge Join

If outer node is using parallelism then we consider parallel merge-join
- Outer node will be scanned in parallel by multiple workers
- Inner node will be processed completely by individual workers

There is still scope of improvements in this strategy
- Parallelise inner sort or materialize nodes
Parallel shared hash

Previously, each worker builds its own copy of hash table
This is particularly favourable to cases when hash table is small

Improved mechanism is to employ the workers for building hash-table in parallel
Once, hash-table is ready, parallel probing can be done
This facilitates the usage of parallel operators on either sides of joins
Upcoming v 10: at a glance

Parallel shared-hash

**Gather**

Workers Planned: 2
Workers Launched: 2

-> Hash Join

  Hash Cond (foo.b = bar.b)

-> **Parallel Seq Scan** on foo

-> **Parallel Shared Hash**

  -> **Parallel Seq Scan** on bar
Gather-merge

Previously, there was only one option to collect the result from parallel operators i.e. gather, it does not maintain interesting order.

Therefore, extra sort node is required on top for ordered output.

Now, if workers are providing sorted result specifically, output from parallel index, parallel merge join, etc. then gather-merge will maintain the sort-order in the final result.
Performance evaluation on TPC-H

- **Experimental setup**
  - RAM = 512 GB
  - Number of cores = 32

- **Parameter settings**
  - Work_mem = 64 MB
  - Shared_buffers = 8 GB
  - Effective_cache_size = 10 GB
  - Random_page_cost = seq_page_cost = 0.1
  - Max_parallel_workers_per_gather = 4

- **Database setup**
  - Scale factors = 20, 300
  - Additional indexes: l_shipmode, l_shipdate, o_orderdate, o_comment
Performance evaluation on TPC-H

Results on scale factor 20
Performance evaluation on TPC-H

Results on scale factor 20
Performance evaluation on TPC-H

Performance improvement with more parallel operators

Results on scale factor 300
How to further improve performance

Tuning parameters

- **Max_parallel_workers_per_gather**
  - Recommended value 1 to 4

- **Reduce following costs**
  - **Parallel_tuple_cost**: planner's estimate of the cost of transferring one tuple from a parallel worker process to another process
  - **Parallel_setup_cost**: planner's estimate for launching parallel workers and initializing dynamic shared memory
  - **Min_parallel_table_scan_size**: the minimum size of relations to be considered for parallel sequence scan
  - **Min_parallel_index_scan_size**: the minimum size of index to be considered for parallel scan
  - **Random_page_cost**: estimated cost of accessing a random page in disk

- **Increase following parameters**
  - **Work_mem**
  - **Effective_cache_size**
  - **Shared_buffers**
Nightmares for parallel operators

- If the actual number of workers at execution time is less than planned number of workers
  - Increase max_parallel_workers
- Shared_buffer_size is lesser than the size of inner side of join
- Random_page_cost is set too low but database is not in memory
  - Increase random_page_cost
  - Load data in memory
- Drastic under-estimation in selectivities
Conclusion

- Adding intra-query parallelism improves per query response time
  - Previously, overall throughput was the only focus
  - This makes it more suitable for OLAP environments

- Till version 9.6, parallel support for sequence scan, hash join, nestloop join, and aggregates is available
  - Out of 22 queries of TPC-H, performance improved for 15 queries
  - In which 3 queries are at least 4 times faster and 11 queries are 2 times faster

- More parallel executor nodes are planned for upcoming versions
  - More parallel access methods - index, index-only are already committed
  - Improved parallel join mechanisms
  - Gather with interesting order
  - Removed restrictions for nodes using SubPlans(already committed) or InitPlans
  - Around 10 of 22 TPC-H queries show significant improvement in performance
  - In which around 4 queries show more than 2x improvement
Output: Thank You

Gather

Workers Planned: 2
Workers Launched: 2

-> Parallel Index Scan on Common_phrases
   Index Cond: ( value = ‘Thank You’ )
   Filter: Language = ‘English’

Slide credits: