Leveraging Query Parallelism In PostgreSQL

Get the most from intra-query parallelism!

Dilip Kumar
(Principle Software Engineer)

Rafia Sabih
(Software Engineer)

Overview

- Further enhancements in parallel query
  - Parallel scans
  - Parallel joins
  - Other parallel operators
- How to get the most from query parallelism
  - Dos and don’ts of parallel operators
- Tuning parallel query
- Take away

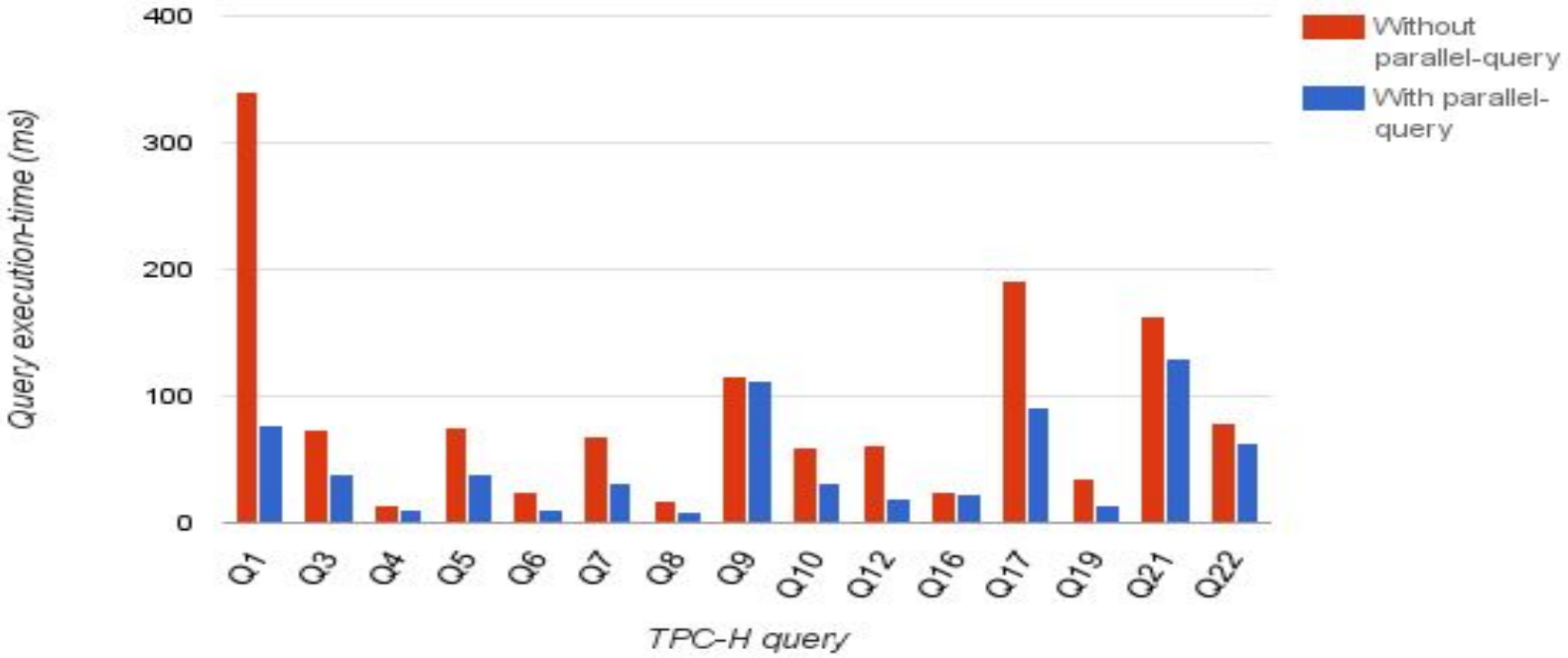
Intra-query parallelism in v 9.6

- 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 leader finalizes the aggregate

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



Performance analysis: what was 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

Further enhancements

- More parallel access methods
  - Parallel index and index-only scan
  - Parallel bitmap heap scan
- More parallel joins
  - Parallel merge join
  - Parallel hash join with shared hash
- Other parallel operators
  - Parallel append
  - Gather-merge
  - Pushing sub/init plans to workers

New Parallel access methods at a glance

- 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

New Parallel access methods 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

New Parallel joins at a glance

- 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

New Parallel joins at a glance

- 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

Other parallel enhancements at a glance

- Pushing sub/init plans to workers
  - An init plan is computed in the gather node and result is stored as param
  - Workers can directly access this value and use it wherever required
  - Uncorrelated sub-plan can be passed down to the workers
  - Each worker then executes the sub-plan individually
  - This way, parallelism is not inhibited for the nodes having uncorrelated subqueries

Other parallel enhancements at a glance

- Parallel append
  - Each scan below the append can run parallely with any plan of their choice
  - The child plans of parallel append need not to be a parallel plan
  - This helps in fair distribution of workers for the child nodes
  - This is particularly useful in partitioned relations

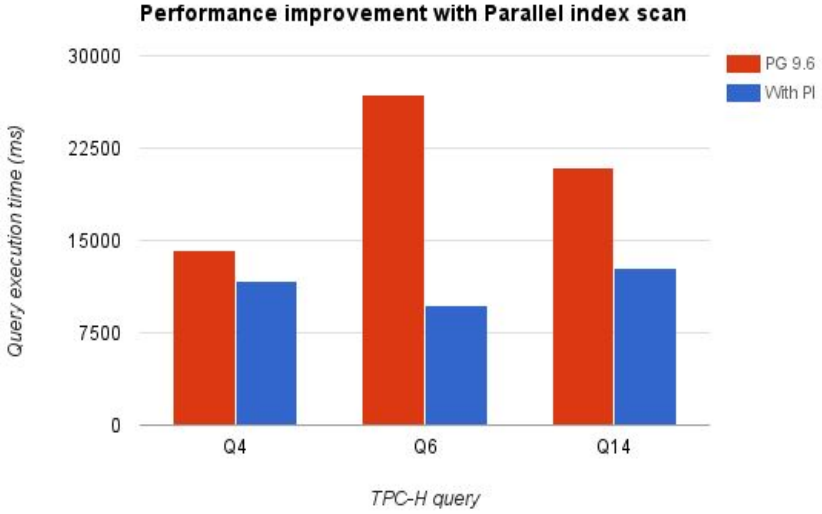
Other parallel enhancements at a glance

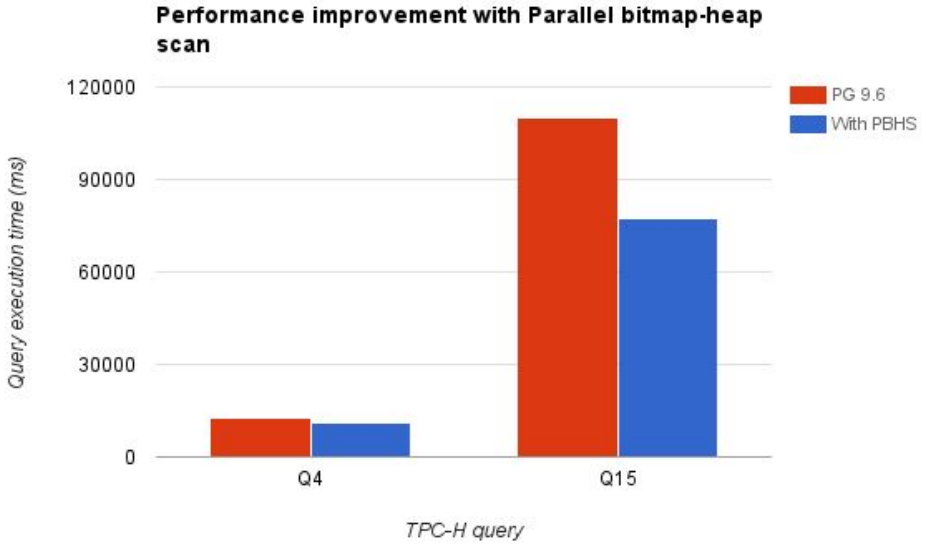
- Gather merge
  - To maintain the interesting order from parallel scans
  - It saves the cost of extra sort node that could be required on top for ordered output
  - For example outputs from parallel index, parallel merge join, etc. can use gather-merge to maintain the order in the final results

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

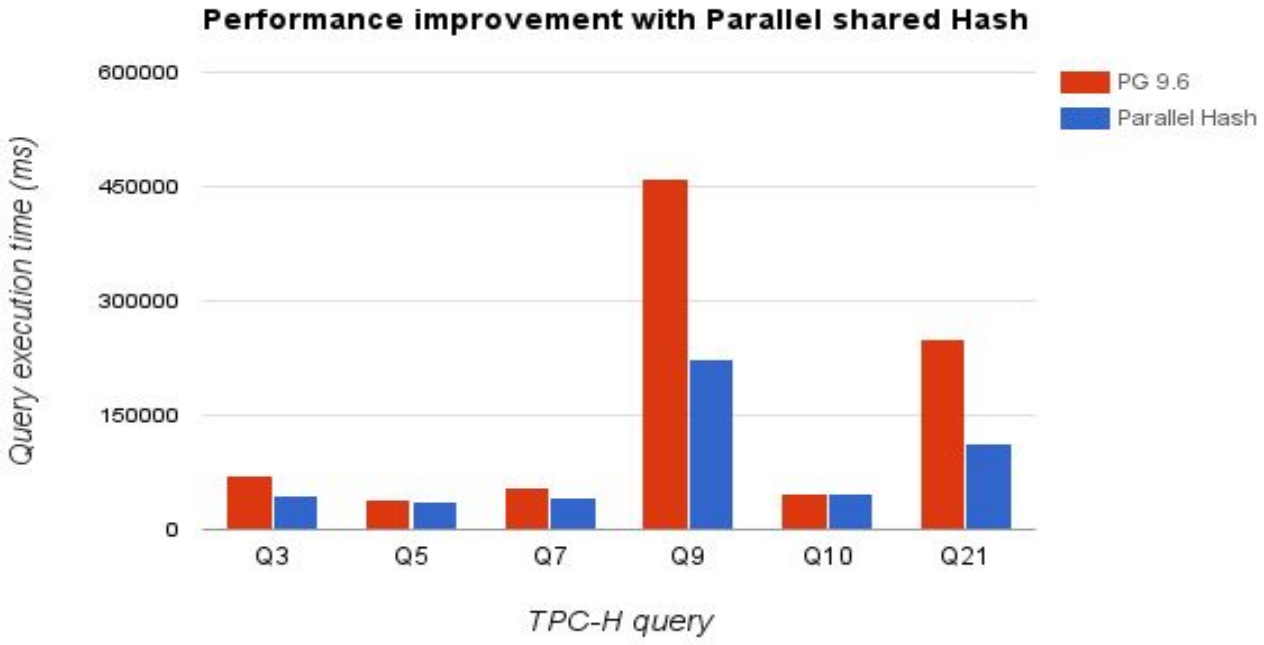
Performance evaluation on TPC-H





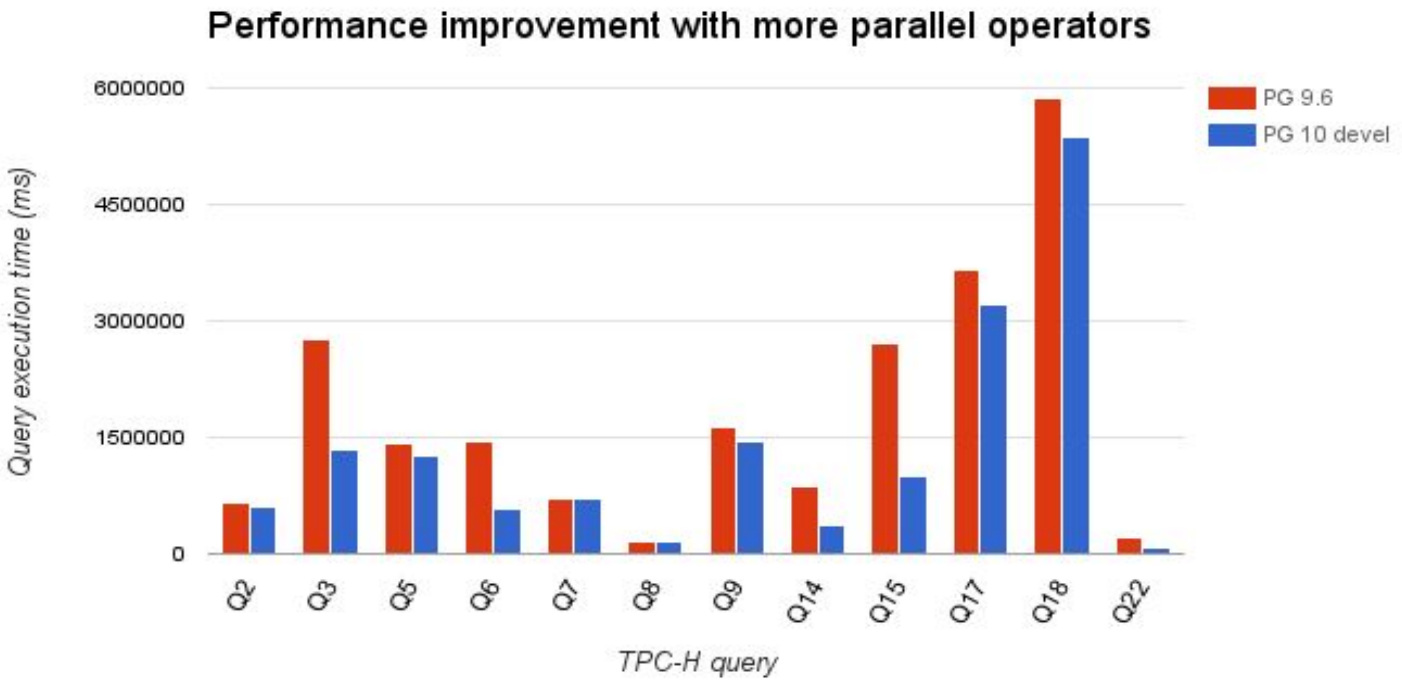
Results on scale factor 20

Performance evaluation on TPC-H



Results on scale factor 20

Performance evaluation on TPC-H



Results on scale factor 300

Tuning parallelism

- Max_parallel_workers_per_gather
  - Number of workers per node for a parallel operator
  - Recommended value 1 to 4
  - The ideal value of this parameter is determined by number of cores in the system and the work required at a node
    - E.g. If *the number of cores is 8* but the work required at node is enough for *2 workers* only then increasing this parameter will not help
    - Similarly, if the *number of cores is 2* and we increased this *parameter to 10*, then it's likely to cause *degradation in performance*

Tuning parallelism

- 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
- We can lower the values of these parameters to diagnose the performance of parallel operators

Tuning parallelism

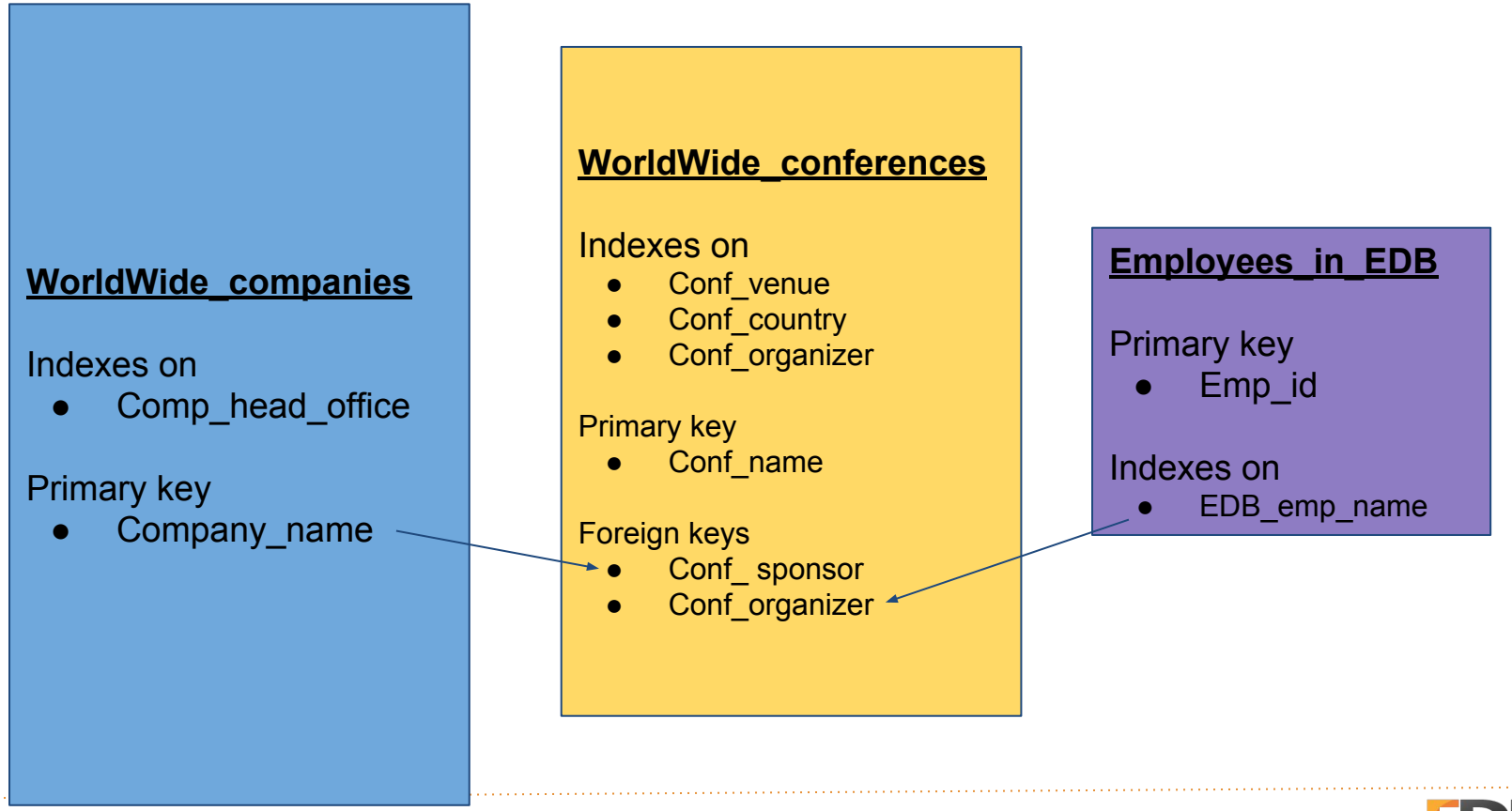
- Min_parallel_table_scan_size
  - Minimum size of relations to be considered for parallel sequence scan
  - The default value of this parameter is 8MB
  - If the database mostly has large tables then it is better to increase this parameter
  - For diagnostic purposes we can decrease it to lower values to analyse the query plans, etc.

- Min_parallel_index_scan_size
  - the minimum size of index to be considered for parallel scan
  - The default value is 512kB

Tuning parallelism

- Work_mem
  - Amount of memory given to per worker per node
- Effective_cache_size
  - If random_page_cost is low then this parameter should be enough to accommodate the secondary indexes
- Shared_buffers
- Random_page_cost
  - Estimated cost of accessing a random page in disk

Our example database



Favourable cases of parallelism: Table scans

- Parallel seq-scans
  - Small number of selected rows from a large table
    - E.g. Select conf_name, date_of_conf from WorldWide_conferences where conf_name like 'PG_conf India%'
  - Complicated qualification condition

```
Gather
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Seq Scan on WorldWide_conferences
         Filter: conf_name like 'PG_conf India%'
```

Favourable cases of parallelism: Table scans

- Parallel index-scans
  - Large number of rows filtered out after indexed qual
    - E.g. Select conf_name, conf_place from WorldWide_conferences where conf_venue = 'Hotel Park Plaza' and conf_name like 'PG_conf India%'
    - Where index in on conf_venue

```
Gather
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Index Scan on WorldWide_conferences
         Index Cond: ( conf_venue = 'Hotel Park Plaza' )
            Filter: conf_name like 'PG_conf India%'
```

Favourable cases of parallelism: Table scans

- Parallel bitmap-heap-scans
  - Mid-way selectivitiy of scan is required for choosing bitmap heap scan
  - For parallel bitmap heap scan, the number of selected rows should be small
    - E.g. Select conf_name, conf_year from WorldWide_conferences where conf_country ='USA' and conf_name like 'PG_conf USA %'
    - Where index is on conf_country

```
Gather
   Workers Planned: 2
   Workers Launched: 2
  -> Parallel Bitmap Heap Scan on WorldWide_conference
      -> Bitmap Index Scan on conf_country
          Index Cond: (conf_country = 'USA')
```

Favourable cases of parallelism: Joins

- Parallel NestedLoop
  - Join filter is filtering out most of the tuples coming from scans
  - Where outer-scan is large
  - E.g. Select conf_name from WorldWide_conferences, employees_in_EDB where conf_organizer = EDB_emp_name
  - Where index in on EDB_emp_name
  - Sample output

Conf_name
PG_Conf India 2017
PG_Conf India 2016
PG_Conf USA 2017

Favourable cases of parallelism: Joins

- Parallel NestedLoop
  - E.g. Select conf_name from WorldWide_conferences conf, employees_in_EDB where conf_organizer = EDB_emp_name
  - Where index in on EDB_emp_name

```
Gather
    Workers Planned: 2
    Workers Launched: 2
       -> Nested Loop
          -> Parallel Seq Scan on WorldWide_conferences
          -> Index Scan on emp_in_EDB using EDB_emp_name_idx
             Index Cond: (EDB_emp_name = conf_organiser)
```

Favourable cases of parallelism: Joins

- Parallel Merge join
  - Join filter is filtering out most of the tuples coming from scans
  - Sorted output is required from join
  - E.g. Select conf_name, comp_name, conf_country as venue from WorldWide_conferences conf, WorldWide_companies comp where conf_country = comp_head_office order by comp_head_office desc
  - Where index in on conf_country and comp_head_office
  - Sample output

Conf_name	comp_name	Venue
PG_Conf USA 2017	EDB	USA
PF_conf Asia 2016	Ashnik	Singapore

Favourable cases of parallelism: Joins

- Parallel Merge join
  - E.g. Select conf_name, comp_name, conf_country as venue from WorldWide_conferences conf, WorldWide_companies comp where conf_country = comp_head_office order by comp_head_office
  - Where index in on conf_country and comp_head_office

```
Sort
    Sort Key: comp_head_office
    Gather
        Workers Planned: 2
        Workers Launched: 2
        -> Merge Join
            Merge Cond (conf_country = comp_head_office)
                -> Parallel Index Scan on comp
                -> Index scan on conf
```

Favourable cases of parallelism: Joins

- Parallel hash join with shared hash
  - Query should be favourable for hash join
  - Join filter filters out a large number of rows
  - E.g. Select conf_name, comp_name, from WorldWide_conferences, WorldWide_companies where comp_name = conf_sponsor and conf_name like 'PG_Conf%'
  - Sample output:

Conf_name	comp_name
PG_Conf India 2017	EDB
PG_Conf India 2017	2nd Quadrant
PG_Conf USA 2017	EDB

Favourable cases of parallelism: Joins

- Parallel hash join with shared hash
  - E.g. Select conf_name, comp_name, from WorldWide_conferences conf, WorldWide_companies comp where comp_name = conf_sponsor and conf_name like 'PG_Conf%'

```
Gather
Workers Planned: 2
Workers Launched: 2
 -> Hash Join
      Hash Cond (comp_name = conf_sponsor)
   -> Parallel Seq Scan on comp
   -> Parallel Hash
        -> Parallel Seq Scan on conf
             Filter: (conf_name like 'PG_conf%')
```

Favourable cases of parallelism: Aggregates

- Number of groups is small, when scan output is large
- E.g. Select conf_country, count(*) as number_of_conf from WorldWide_conferences group by conf_country
  - Sample output:

Conf_country	number_of_conf
India	1500
USA	30000

```
Finalize Aggregate
    -> Gather
        -> Partial Aggregate
            -> Parallel Seq Scan on WorldWide_conferences
```

Favourable cases of parallelism: Gather-merge

- Gather-merge
  - Interesting order is to be maintained till higher nodes
    - E.g. Select conf_venue, conf_name from WorldWide_conferences where conf_venue = 'Hotel Park Plaza' and conf_name like 'PG_conf India%' order by conf_venue
    - Where index in on conf_venue

```
Gather-Merge
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Index Scan on WorldWide_conferences
         Index Cond: ( conf_venue = 'Hotel Park Plaza' )
            Filter: conf_name like 'PG_conf India%'
```

Favourable cases of parallelism: Init plan

- E.g. Select conf_name, conf_sponsor from WorldWide_conferences where conf_sponsor = (select comp_name from WorldWide_companies where num_emp > 300 limit 1)
  - Sample output:

Conf_name	conf_sponsor
PGConf	EDB

Favourable cases of parallelism: Init plan

```
InitPlan1 (return $2)
    Limit
        Gather
            Workers Planned: 2
            Workers Launched: 2
            Parallel Seq scan on WorldWide_companies
                Filter: (num_emp > 300)

Gather
    Workers Planned: 2
    Workers Launched: 2
    Params Evaluated: $2
    ->  Parallel Seq Scan on WorldWide_conferences
            Filter: conf_spnsor = $2
```

Conclusion

- More parallel operators, the better query performance
  - Previously, overall throughput was the only focus
  - Intra-query parallelism makes it suitable for OLAP environments as well
- Remarkable performance improvements with parallel operators on TPC-H
  - Till v9.6 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
  - Further in v10, around 10 of 22 TPC-H queries show significant improvement in performance, in which around 4 queries show more than 2x improvement
- Tuning parameters for parallelism
- Favourable cases for each of the parallel operators

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:

[1] https://www.pgcon.org/2016/schedule/events/913.en.html

[2] https://www.postgresql.eu/events/schedule/pgconfeu2016/session/1360-parallel-query-in-postgresql/

[3] http://pgconf.in/schedule/query-parallelism-in-postgresql-expectations-and-opportunities/