Parallelizing Query Optimization for Left-Deep and Bushy Tree Joins

Members:
Sumit Chaturvedi 160110098
Nilay Pande 160070013
Nitish Joshi 160070017
Adwait Godbole 160070021

Goal:
We plan to hack into PostgreSQL in order to implement parallelized algorithms for query optimization. Our broad goal is to enable PostgreSQL to exploit the ever-growing degree of parallelism forced by the breakdown of Moore's law for Query Optimization.

Bibliography:
Query Optimisation on Shared Nothing Architectures
http://www.vldb.org/pvldb/vol9/p660-trummer.pdf

`Link to our codebase: https://github.com/Vrroom/DB-Project/`

Highlights of the paper:

1. Parallel DP:
    a. Dynamic programming is an obvious choice as clearly there is optimal substructure.
    b. Even then traditional DP involves multiple dependencies (polyadic) often which not the immediately preceding levels. A good example of this is the matrix chain multiplication problem. Hence it is difficult to parallelize.
    c. This paper takes a much different approach to this problem. It partitions the search space such that there are no cross dependencies. It allocates a worker thread to each such component.
    d. By doing so it completely isolates each worker leading to a highly parallel algorithm. The tradeoff is that the algorithm will have redundant computation.
2. Master-Worker:
    a. The master node partitions the search space into components which are passed to the worker nodes. These partitions are made by constraints on the join order which are enforced by the worker when performing DP. Each constraint is encoded by a bit vector.
    b. The component problems are designed so that there are no cross-dependencies. This saves on worker-worker communication cost.
3. Constraints:

a. Left-deep $(x < y)$ semantics dictate that table x must be joined before y.
b. Bushy joins are based a general join tree as hence is more general than left-deep joins. Consider a path p from the leaf corresponding to z to the root. The semantics for $(x < y \mid z)$ dictate that as you traverse p the table corresponding to x is joined before the table corresponding to y.
c. The sequence of tables are paired up and a bit in the encoding is specified corresponding to the order corresponding to each pair or triplet.

Specific Implementations:

1. Parallelizing Left-Deep Tree Joins:
    a. Implement a master node algorithm which divides the query plan search space among the worker nodes so that there is no need of any data sharing between the worker nodes. This is implemented by assigning each worker node a unique set of constraints on the set of query plans it needs to optimize over. These constraints specify the ordering for some pairs of tables which the query plan must satisfy. The union of these constraints over all worker nodes is the entire universe of join orderings. Each worker node returns a best query plan for the given set of constraints. Finally, the master node selects the best query plan among all the query plans returned.
    b. Implement a worker node algorithm which returns the best query plan under the given set of constraints by using a dynamic programming algorithm. It first determines the set of admissible query plans of all possible cardinality under the given set of constraints using the `AdmJoinResults` method.
    c. It then incrementally builds the best query plan of higher and higher cardinality by memoizing the previous results till the cardinality equals the total number of tables in the original query. The space of subcases to consider is given by the `TrySplits` method. Finally, it returns its best query plan to the master node.
2. Parallelizing Bushy Tree Joins:
    a. Master node algorithm is similar to the one for Left-Deep Tree join. The only difference is in the constraint specification for worker nodes. Also the functions for `PartConstraints` and `AdmJoinResults` are similar.
    b. The main difference arises from the fact that we use a triplet wise encoding for the constraints. Hence we need to a modified `TrySplits` algorithm to take into account the structural difference between Left-Deep and Bushy cases.

Functionalities Provided:

1. Primary:
    a. SELECT * FROM Q_1, Q_2, Q_3, …, Q_n WHERE P_1, P_1, P_1, P_1
        i. where Q_i are the tables and P_i are the predicates on attributes in tables
2. Optional:

a. Right now, in our vanilla implementation, we will be only trying to optimize one type of scans and a particular implementation of join methods. If we have time, the we could try to incorporate the various join methods and scan methods into an improved cost function which will give a better plan.
b. As of now, we haven't promised to handle special kinds of joins such as Left Join since there planning is handled differently in Postgresql. We can try to overcome this.

Codebase Touched:

1. We plan to tinker with PostgreSQL. More specifically we will have to tweak the planner backend. The code is available in `src/backend/optimizer/`.
2. The file `src/backend/optimizer/path/allpaths.c` searches through all possible paths in the search space and chooses the optimal join sequence. It does a bunch of other things also and considers a variety of join scenarios that are present in the SQL language.
3. In the file mentioned above, there is a function called `search_join_tree`. This takes as input the complete information that has been obtained so far about the current query. This information is present in a struct called `PlannerInfo`. This struct contains the current Query, the number of base relations (i.e. tables) involved in the current query and a list of join relations.
4. A very crucial structure for our purpose is the `RelOptInfo` structure. This structure contains information about the base relations and the join relations. Join relation basically represents the intermediate result of joins on a subset of the entire set of base relations. For example, if {T1, T2, T3} is our set of tables, then we'll have a `RelOptInfo` Structure for each individual Ti. Additionally, as part of our join order optimization endeavour, we'll have to fill in a `RelOptInfo` structure for the set {T1,T2,T3} which represents the join result of these three tables. Such a `RelOptInfo` structure is called a join relation. The `RelOptInfo` structure also store the various plans for obtaining the set they represent.
5. Our job basically boils down to filling the `RelOptInfo` structure with the optimal plan in the `search_join_tree` function and making sure that we don't break any existing code.

Constraints:

1. For bushy tree joins we require number of relations to be 0 mod 3 and for left deep we require the number of relations to be 0 mod 2.
2. The maximum number of workers for left deep join is 2^((num_relations)/2) and for bushy is 2^((num_relations)/3).
3. In case where the actual number of workers exceeds above, the remaining workers would not be initiated.
4. We have used locks in some critical PostgreSql functions (palloc) for addressing concurrency issues. Even then, for large number of workers we have some concurrency

related bugs. If we further serialize the worker threads by adding locks, the issue is resolved at the cost of parallelism.

Results:

Query 1: Looks at only left deep joins with 2 workers.

```
backend> explain analyse select * from student, instructor;
     1: QUERY PLAN    (typeid = 25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "Nested Loop  (cost=0.00..4.23 rows=156 width=51) (actual
time=0.055..0.287 rows=156 loops=1)"    (typeid = 25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "  ->  Seq Scan on student  (cost=0.00..1.13 rows=13 width=25) (actual
time=0.026..0.033 rows=13 loops=1)"    (typeid = 25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "  ->  Materialize  (cost=0.00..1.18 rows=12 width=26) (actual
time=0.002..0.006 rows=12 loops=13)"    (typeid = 25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "        ->  Seq Scan on instructor  (cost=0.00..1.12 rows=12 width=26)
(actual time=0.009..0.016 rows=12 loops=1)"    (typeid = 25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "Planning Time: 1.848 ms"    (typeid = 25, len = -1, typmod = -1, byval =
f)
    ----
     1: QUERY PLAN = "Execution Time: 0.431 ms"    (typeid = 25, len = -1, typmod = -1, byval =
f)
    ----
```

Query 2: Looks at bushy joins with 2 workers.

```
backend> explain analyse select * from time_slot, instructor, student;
     1: QUERY PLAN    (typeid = 25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "Nested Loop  (cost=0.00..44.48 rows=3120 width=73) (actual
time=125.863..129.305 rows=3120 loops=1)"    (typeid = 25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "  ->  Nested Loop  (cost=0.00..4.23 rows=156 width=51) (actual
time=0.077..0.265 rows=156 loops=1)"    (typeid = 25, len = -1, typmod = -1, byval = f)
    ----
```

```
     1: QUERY PLAN = "          ->  Seq Scan on student  (cost=0.00..1.13 rows=13 width=25)
(actual time=0.025..0.032 rows=13 loops=1)"    (typeid = 25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "          ->  Materialize  (cost=0.00..1.18 rows=12 width=26) (actual
time=0.003..0.007 rows=12 loops=13)"    (typeid = 25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "                    ->  Seq Scan on instructor  (cost=0.00..1.12 rows=12
width=26) (actual time=0.022..0.030 rows=12 loops=1)"    (typeid = 25, len = -1, typmod = -1,
byval = f)
    ----
     1: QUERY PLAN = "  ->  Materialize  (cost=0.00..1.30 rows=20 width=22) (actual
time=0.806..0.810 rows=20 loops=156)"    (typeid = 25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "          ->  Seq Scan on time_slot  (cost=0.00..1.20 rows=20 width=22)
(actual time=125.758..125.769 rows=20 loops=1)"    (typeid = 25, len = -1, typmod = -1, byval =
f)
    ----
     1: QUERY PLAN = "Planning Time: 40.808 ms"    (typeid = 25, len = -1, typmod = -1, byval =
f)
    ----
     1: QUERY PLAN = "Execution Time: 129.849 ms"    (typeid = 25, len = -1, typmod = -1, byval
= f)
```

Query 3 : Bushy joins for large (9) number of relations.

```
backend> explain select * from instructor, student, takes, teaches, section, course, advisor,
time_slot, department;
     1: QUERY PLAN    (typeid = 25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "Nested Loop  (cost=0.00..158113016.19 rows=12648636000
width=231)"    (typeid = 25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "  ->  Nested Loop  (cost=0.00..4529.93 rows=360360 width=146)"
(typeid = 25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "          ->  Nested Loop  (cost=0.00..18.07 rows=1092 width=85)"
(typeid = 25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "                    ->  Nested Loop  (cost=0.00..3.26 rows=84 width=46)"
(typeid = 25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "                    ->  Seq Scan on instructor  (cost=0.00..1.12 rows=12
width=26)"    (typeid = 25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "                    ->  Materialize  (cost=0.00..1.10 rows=7 width=20)"
(typeid = 25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "                              ->  Seq Scan on department  (cost=0.00..1.07
rows=7 width=20)"    (typeid = 25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "                    ->  Materialize  (cost=0.00..1.19 rows=13 width=39)"
(typeid = 25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "                    ->  Seq Scan on course  (cost=0.00..1.13 rows=13
width=39)"    (typeid = 25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "          ->  Materialize  (cost=0.00..8.18 rows=330 width=61)"    (typeid =
25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "                    ->  Nested Loop  (cost=0.00..6.53 rows=330 width=61)"
(typeid = 25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "                    ->  Seq Scan on takes  (cost=0.00..1.22 rows=22
width=28)"    (typeid = 25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "                    ->  Materialize  (cost=0.00..1.22 rows=15 width=33)"
(typeid = 25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "                              ->  Seq Scan on section  (cost=0.00..1.15 rows=15
width=33)"    (typeid = 25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "  ->  Materialize  (cost=0.00..624.01 rows=35100 width=85)"    (typeid =
25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "          ->  Nested Loop  (cost=0.00..448.51 rows=35100 width=85)"
(typeid = 25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "                    ->  Nested Loop  (cost=0.00..4.75 rows=195 width=51)"
(typeid = 25, len = -1, typmod = -1, byval = f)
    ----
     1: QUERY PLAN = "                    ->  Seq Scan on teaches  (cost=0.00..1.15 rows=15
width=26)"    (typeid = 25, len = -1, typmod = -1, byval = f)
    ----
```

```
    1: QUERY PLAN = "                    ->  Materialize  (cost=0.00..1.19 rows=13 width=25)"
(typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "                              ->  Seq Scan on student  (cost=0.00..1.13 rows=13
width=25)"    (typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "                    ->  Materialize  (cost=0.00..5.46 rows=180 width=34)"
(typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "                    ->  Nested Loop  (cost=0.00..4.56 rows=180 width=34)"
(typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "                              ->  Seq Scan on time_slot  (cost=0.00..1.20
rows=20 width=22)"    (typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "                              ->  Materialize  (cost=0.00..1.14 rows=9 width=12)"
(typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "                              ->  Seq Scan on advisor  (cost=0.00..1.09 rows=9
width=12)"    (typeid = 25, len = -1, typmod = -1, byval = f)
   ----
```

Query 4 : Example of a complex query

```
backend> explain select * from instructor, student, takes, teaches, section, course, advisor,
time_slot, department where student.id = takes.id and advisor.i_id = instructor.id;
    1: QUERY PLAN    (typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "Hash Join  (cost=70.38..918506.07 rows=81081000 width=231)"
(typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "  Hash Cond: ((takes.id)::text = (student.id)::text)"    (typeid = 25, len =
-1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "  ->  Nested Loop  (cost=1.41..5236.35 rows=415800 width=141)"
(typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "        ->  Nested Loop  (cost=0.14..19.22 rows=440 width=50)"    (typeid
= 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "                    ->  Index Scan using takes_pkey on takes
(cost=0.14..12.47 rows=22 width=28)"    (typeid = 25, len = -1, typmod = -1, byval = f)
   ----
```

```
    1: QUERY PLAN = "                    ->  Materialize  (cost=0.00..1.30 rows=20 width=22)"
(typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "                    ->  Seq Scan on time_slot  (cost=0.00..1.20 rows=20
width=22)"    (typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "          ->  Materialize  (cost=1.27..21.99 rows=945 width=91)"    (typeid =
25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "                    ->  Nested Loop  (cost=1.27..17.27 rows=945 width=91)"
(typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "                    ->  Nested Loop  (cost=1.27..4.27 rows=63 width=58)"
(typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "                              ->  Hash Join  (cost=1.27..2.39 rows=9 width=38)"
(typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "                              Hash Cond: ((advisor.i_id)::text =
(instructor.id)::text)"    (typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "                              ->  Seq Scan on advisor  (cost=0.00..1.09 rows=9
width=12)"    (typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "                              ->  Hash  (cost=1.12..1.12 rows=12 width=26)"
(typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "                                        ->  Seq Scan on instructor  (cost=0.00..1.12
rows=12 width=26)"    (typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "                              ->  Materialize  (cost=0.00..1.10 rows=7 width=20)"
(typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "                              ->  Seq Scan on department  (cost=0.00..1.07
rows=7 width=20)"    (typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "                    ->  Materialize  (cost=0.00..1.22 rows=15 width=33)"
(typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "                              ->  Seq Scan on section  (cost=0.00..1.15 rows=15
width=33)"    (typeid = 25, len = -1, typmod = -1, byval = f)
   ----
```

```
    1: QUERY PLAN = "  ->  Hash  (cost=37.28..37.28 rows=2535 width=90)"    (typeid = 25, len
= -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "        ->  Nested Loop  (cost=0.00..37.28 rows=2535 width=90)"
(typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "              ->  Nested Loop  (cost=0.00..4.40 rows=169 width=64)"
(typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "              ->  Seq Scan on student  (cost=0.00..1.13 rows=13
width=25)"    (typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "              ->  Materialize  (cost=0.00..1.19 rows=13 width=39)"
(typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "                    ->  Seq Scan on course  (cost=0.00..1.13 rows=13
width=39)"    (typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "              ->  Materialize  (cost=0.00..1.22 rows=15 width=26)"
(typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "              ->  Seq Scan on teaches  (cost=0.00..1.15 rows=15
width=26)"    (typeid = 25, len = -1, typmod = -1, byval = f)
   ----
```

Query 5: left outer join

```
backend> explain select * from student left outer join takes on student.id = takes.id;
    1: QUERY PLAN    (typeid = 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "Hash Right Join  (cost=1.29..2.59 rows=22 width=53)"    (typeid = 25,
len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "  Hash Cond: ((takes.id)::text = (student.id)::text)"    (typeid = 25, len =
-1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "  ->  Seq Scan on takes  (cost=0.00..1.22 rows=22 width=28)"    (typeid
= 25, len = -1, typmod = -1, byval = f)
   ----
    1: QUERY PLAN = "  ->  Hash  (cost=1.13..1.13 rows=13 width=25)"    (typeid = 25, len = -1,
typmod = -1, byval = f)
   ----
```

```
     1: QUERY PLAN = "          ->  Seq Scan on student  (cost=0.00..1.13 rows=13 width=25)"
(typeid = 25, len = -1, typmod = -1, byval = f)
    ----
```