Vectorized Postgres (VOPS extension)

Konstantin Knizhnik
Postgres Professional

Why Postgres is slow on OLAP queries?

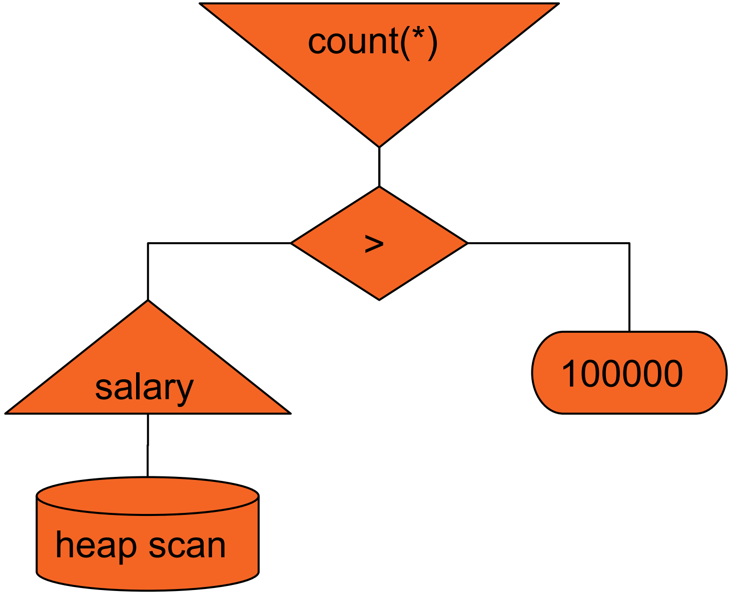
1. Unpacking tuple overhead (heap_deform_tuple)
2. Interpretation overhead (invocation of query plan node functions)
3. Abstraction penalty (user defined types and operations)
4. Pull model overhead (saving/restoring context on each access to the page)
5. MVCC overhead (~20 bytes per tuple space overhead + visibility check overhead)

Typical OLAP query profile

```
16.57%  postgres  postgres          [.] slot_deform_tuple
13.39%  postgres  postgres          [.] ExecEvalExpr
 8.64%  postgres  postgres          [.] advance_aggregates
 8.58%  postgres  postgres          [.] advance_transition_function
 5.83%  postgres  postgres          [.] float8_accum
 5.14%  postgres  postgres          [.] tuplehash_insert
 3.89%  postgres  postgres          [.] float8pl
 3.60%  postgres  postgres          [.] slot_getattr
 2.66%  postgres  postgres          [.] bpchareq
 2.56%  postgres  postgres          [.] heap_getnext
```

Query execution plan

select count(*) from where salary > 100000;



Traditional query execution

```
SELECT sum(quantity*price) FROM lineitems;
```

shipdate	quantity	price
21.02.2017	100	99
23.02.2017	200	60
24.02.2017	150	120

100 * 99 = 9900

+

200 * 60 = 12000

+

150 * 120 = 18000

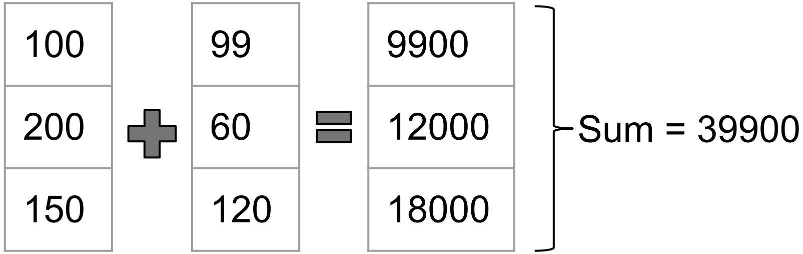
= 39900

Vectorized query execution

SELECT sum(quantity*price) FROM lineitems;

shipdate	quantity	price
21.02.2017, 23.02.2017, 24.02.2017	100, 200, 150	99, 60, 120
25.02.2017, 26.02.2017, 28.02.2017	300, 110, 80	100, 60, 230

Tile



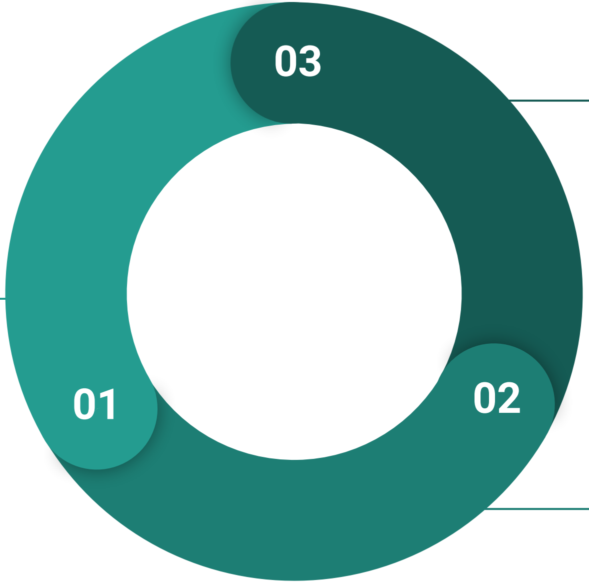
100
200
150

99
60
120

9900
12000
18000

Sum = 39900

VOPS integration in Postgres



Abstract data types

VOPS defines special types and operators for tiles, which should be used instead of scalar types

Planner hooks

Change query plan for operators which can not be redefined

FDW

Define Foreign Data Wrapper allowsing to use VOPS table in any query

User defined types,operators,aggregates

```
create type vops_float4 (
    input = vops_float4_input,
    output = vops_float4_output,
    alignment = double,
    internallength = 272 -- 16 + 64*4);
create operator - (leftarg=vops_float4, rightarg=vops_float4, procedure=vops_float4_sub);
create operator + (leftarg=vops_float4, rightarg=vops_float4, procedure=vops_float4_add, commutator= +);
create operator * (leftarg=vops_float4, rightarg=vops_float4, procedure=vops_float4_mul, commutator= *);
create operator / (leftarg=vops_float4, rightarg=vops_float4, procedure=vops_float4_div);
create operator = (leftarg=vops_float4, rightarg=vops_float4, procedure=vops_float4_eq, commutator= =);
create operator <> (leftarg=vops_float4, rightarg=vops_float4, procedure=vops_float4_ne, commutator= <>);
create operator > (leftarg=vops_float4, rightarg=vops_float4, procedure=vops_float4_gt, commutator= <);
create operator < (leftarg=vops_float4, rightarg=vops_float4, procedure=vops_float4_lt, commutator= >);
create operator >= (leftarg=vops_float4, rightarg=vops_float4, procedure=vops_float4_ge, commutator= <=);
create operator <= (leftarg=vops_float4, rightarg=vops_float4, procedure=vops_float4_le, commutator= >=);
create operator - (rightarg=vops_float4, procedure=vops_float4_neg);
create aggregate sum(vops_float4) (
    sfunc = vops_float4_sum_accumulate,
    stype = float8,
    combinefunc = float8pl,
    parallel = safe);
```

Creating VOPS projections

```
create table vops_lineitem_projection(
   l_shipdate vops_date not null,
   l_quantity vops_float4 not null,
   l_extendedprice vops_float4 not null,
   l_discount vops_float4 not null,
   l_tax vops_float4 not null,
   l_returnflag "char" not null,
   l_linestatus "char" not null
);
-- Load data from existed (normal) table
select populate(destination := 'vops_lineitem'::regclass,
                source := 'lineitem'::regclass);

-- Load data directly from CSV file
select import(destination := 'vops_lineitem'::regclass,
              csv_path := '/mnt/data/lineitem.csv', separator := '|');
```

VOPS special operators

```
-- Q6 using VOPS special operators
select sum(l_extendedprice*l_discount) as revenue
from vops_lineitem
where filter(betwixt(l_shipdate, '1996-01-01', '1997-01-01')
                & betwixt(l_discount, 0.08, 0.1)
                & (l_quantity < 24));

-- Q1 using VOPS group by
select reduce(map(l_returnflag||l_linestatus, 'sum,sum,sum,sum,avg,avg,avg',
    L_quantity, l_extendedprice,
    l_extendedprice*(1-l_discount),
    l_extendedprice*(1-l_discount)*(1+l_tax),
    L_quantity, l_extendedprice,
    l_discount)) from vops_lineitem where filter(l_shipdate <= '1998-12-01'::date);
```

Access through Postgres FDW

```
create foreign table lineitem_fdw  (
   l_shipdate date not null,
   l_quantity float4 not null,
   l_extendedprice float4 not null,
   l_discount float4 not null,
   l_tax      float4 not null,
   l_returnflag "char" not null,
   l_linestatus "char" not null
) server vops_server options (table_name 'vops_lineitem');

select
    l_returnflag,v l_linestatus, sum(l_quantity) as sum_qty,
    sum(l_extendedprice) as sum_base_price,
    sum(l_extendedprice*(1-l_discount)) as sum_disc_price,
    sum(l_extendedprice*(1-l_discount)*(1+l_tax)) as sum_charge,
    avg(l_quantity) as avg_qty, avg(l_extendedprice) as avg_price, avg(l_discount) as avg_disc
from  lineitem_fdw
where  l_shipdate <= '1998-12-01'
group by l_returnflag,l_linestatus
order by l_returnflag,l_linestatus;
```

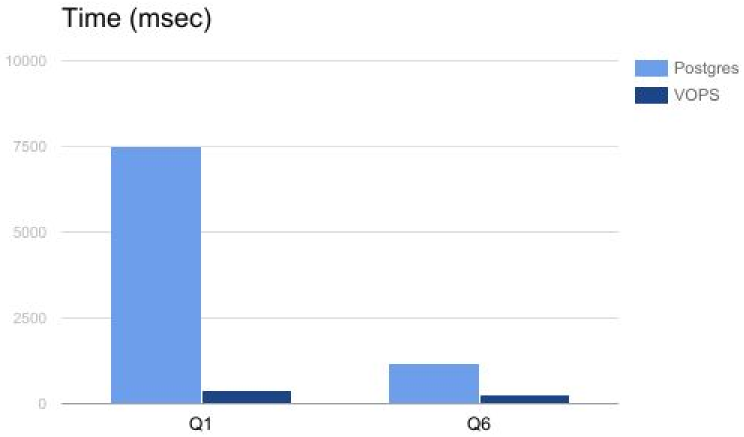
Standard SQL query vs. VOPS query

```
select
    sum(l_extendedprice*l_discount)
as revenue
from
    Lineitem
where
    l_shipdate between
'1996-01-01' and
'1997-01-01'
    and l_discount between 0.08 and
0.1
    and l_quantity < 24;
```

```
select
    sum(l_extendedprice*l_discount)
as revenue
from
    vops_lineitem_projection
where
    l_shipdate between
'1996-01-01'::date and
'1997-01-01'::date
    and l_discount between 0.08 and
0.1
    and l_quantity < 24;
```

Performance advantage

TPC-H scale 10 (8Gb)



	Postgres	VOPS
Q1	7503	396
Q6	1171	249

Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz, 16GB, SSD

Alternatives

- Citus Data: parallel distributed execution of query. Several times faster than Spark SQL. Columnar store and vectorized extensions.
- Greenplum: The World’s First Open-Source & Massively Parallel Data Platform.
- Vitesse Data: Deep Green 4.5 faster than Geeenplum
- JIT (Just-in-Time compilation)
  - +5 times on Q1 for ISPRAS implementation
  - +2 times on Q1 for Andres Freund implementation

Thank you!

Repository: https://github.com/postgrespro/vops

Contact: <Konstantin Knizhnik> k.knizhnik@postgrespro.ru