Vectorized Execution Introduction

HAWQ Vectorized Execution Design:

Vectorized Execution(VE) is an advanced technique in database query execution stage. HAWQ is supported via vexecutor plugin/extension, please refer to the design doc:

HAWQ Vectorized Execution design

Installation and Activation

1. Make sure HAWQ compilation and installation success.
2. Check if vexecutor.so is installed into HAWQ execution directory. If not, you can execute "make ;make install" in contrib/vexecutor directory to install it.
3. Add library name "vexecutor" to "shared_preload_libraries" in postgresql.conf which has to copies allocated in master and segment data directories respectively. Make sure both of them is assigned. (E.g. postgresql.conf: shared_preload_libraries = 'vexecutor')
4. Restart HAWQ cluster
5. Execute "create_udv.sql" with psql. then some vectorized data types and vectorized aggregate functions will be add to HAWQ.
6. How to remove Vectorized Execution:
    - Delete metadata by executing "clean_udv.sql" in psql.
    - Remove 'vexecutor' from shared_preload_libraries in postgresql.conf
    - Restart HAWQ cluster

Performance Comparation

Conclusion

Vectorized execution is faster than normal execution at least two times in both AO and Parquet table. Some queries can reach 8 times acceleration. You can find the performance test tables structure and queries as follow.

Table Structure

Create Append only and Parquet tables: test_ao, test_parquet

Table Name: test_ao	
Column	Type
a	integer
b	integer

Table Name: test_parquet	
Column	Type
a	integer
b	integer

Create Parquet table "lineitem":

Column	Type
l_orderkey	bigint
l_partkey	integer
l_suppkey	integer
l_linenumber	integer

l_quantity	double precision
l_extendedprice	double precision
l_discount	double precision
l_tax	double precision
l_returnflag	integer
l_linestatus	integer
l_shipdate	date
l_commitdate	date
l_receiptdate	date
l_shipinstruct	character(25)
l_shipmode	character(10)
l_comment	character varying(44)

Please check the performance results as below:

AO Talbe

SQL	Non-Vec (ms)	VE (ms)
select count(a),avg(a),sum(b) from test_ao;	4075	1806
select count(a),avg(a),sum(b) from test_ao group by a;	5713	3347
select count(a),avg(a),sum(b) from test_ao group by b;	5755	3077
select count(a) from test_ao;	2595	1225
select avg(a) from test_ao;	2511	1218
select sum(a) from test_ao;	2426	1208
select count(a) from test_ao group by a;	3442	2623
select avg(a) from test_ao group by a;	3565	2626
select sum(b) from test_ao group by a;	3845	3241

PARQUET Table

SQL	Non-Vec (ms)	VE (ms)
select count(a),avg(a),sum(b) from test_parquet;	8632	995
select count(a),avg(a),sum(b) from test_parquet group by a;	11748	4637
select count(a),avg(a),sum(b) from test_parquet group by b;	11628	4072
select count(a) from test_parquet;	4887	673
select avg(a) from test_parquet;	5666	620
select sum(a) from test_parquet;	4882	625
select count(a) from test_parquet group by a;	7571	3886
select avg(a) from test_parquet group by a;	8268	3861
select sum(b) from test_parquet group by a;	8456	4374

l_returnflag, 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, count() as count_order from lineitem where l_shipdate <= '1998-12-01'::date - 65 group by l_returnflag, l_linestatus order by l_returnflag, l_linestatus;	109060.121	32056.300
select sum(l_extendedprice * l_discount) as revenue from lineitem where l_shipdate >= '1997-01-01'::date and l_shipdate < '1997-01-01'::date + 365 and l_discount > (0.06 - 0.01) and l_discount < (0.06 + 0.01) and l_quantity < 24;	17121.384	9015.241

How to create new vectorized type in HAWQ?

For the original version of HAWQ vectorized execution, it only supports six type in [vtype.h/c] for vectorized execution. The README file represents lots of information relevant to custom your own type for vectorized execution. However, if you still struggle with the coding process, this document may help you pave the way to familiar with type customization. We will provide an example as follow. The date type is chosen as a vtype internal to build a new vectorized execution feature, and all detail and practice experiences are presented in this doc step by step. You can find the type and relevant expression function implementation in [vtype_ext.h/c].

I. Vectorizable type

Before a new vectorized type implementation initiated, an original type must be selected, which is named as internal type in this example. At present, the internal type length must smaller than Datum. The reason is that reference depended type serialization methods are not supported, which may require huge amounts of memory resource if support those in vectorized execution. Consequently, if you want to patch this feature, you should do some further workload, including refactor serialization function and manage type required memory manually. Since the date type satisfies the requirement, and frequently appear in database case, we extend it to the vectorized version named vdateadt.

II. Type defination in SQL script

For type definition, we normally execute four SQL querys as follow:

```
CREATE TYPE vdateadt;

CREATE FUNCTION vdateadtin(cstring) RETURNS vdateadt AS 'vexecutor.so' LANGUAGE C IMMUTABLE STRICT;

CREATE FUNCTION vdateadtout(vdateadt) RETURNS cstring AS 'vexecutor.so' LANGUAGE C IMMUTABLE STRICT;

CREATE TYPE vdateadt ( INPUT = vdateadtin, OUTPUT = vdateadtout, element = date , storage=external);
```

The function vdateadtin and vdateadtout is the input and output casting function for the vdateadt type separately. The declare script include the function formal parameter and return type, as well as specific library name which HAWQ can find the function. When the declaration query has been executed, the new type metadata is recorded in HAWQ pg_type catalog table. You can query the detail using SELECT * FROM pg_type WHERE typname='vdateadt';

Secondly, the new type related expression function need to declare. For date type, it holds date_mi function to process date minus date expression. As the affine function in vdateadt, we declare function vdataadt_mi to handle vdateadt - vdateadt calculation. However, for some queries (e.g. SELECT col1 from tab1 WHERE vdate - '1998-08-02' > 20), the vdateadt - date expression is required. Since that, we declare another function named vdateadt_mi_date. These two function declaration query is posted as follow:

```
CREATE FUNCTION vdateadt_mi(vdateadt, vdateadt) RETURNS vint4 AS 'vexecutor.so' LANGUAGE C IMMUTABLE STRICT;

CREATE OPERATOR - ( leftarg = vdateadt, rightarg = vdateadt, procedure = vdateadt_mi, commutator = - );

CREATE OPERATOR - ( leftarg = vdateadt, rightarg = date, procedure = vdateadt_mi_dateadt, commutator = - );

CREATE FUNCTION vdateadt_pli_int4(vdateadt, int4) RETURNS vdateadt AS 'vexecutor.so' LANGUAGE C IMMUTABLE STRICT;
```

Like the case we show above, all the expression declaration is wrote in create_type.sql.

III. Type implementation

As the declaration notice, all function should define and implement in library vexecutor. The header file of vtype_ext.h is included in vexecutor.h and vcheck .h. Thus, HAWQ query dispatcher and library can recognize the date type vectorizable.

For example, vdateadt_pli_int4 has been defined as follow.

```
PG_FUNCTION_INFO_V1(vdateadt_pli_int4);

Datum vdateadt_pli_int4(PG_FUNCTION_ARGS)

{

  int size = 0;

  int i = 0;

  vdateadt* arg1 = PG_GETARG_POINTER(0);

  int arg2 = PG_GETARG_INT32(1);

  vdateadt* res = buildvint4(BATCHSIZE, NULL);

  size = arg1->dim;

  while(i < size)

 {

     res->isnull[i] = arg1->isnull[i] ;

     if(!res->isnull[i])

        res->values[i] = Int32GetDatum((DatumGetDateADT(arg1->values[i])) + arg2);

     i++;

 }

  res->dim = arg1->dim;

  PG_RETURN_POINTER(res);

}
```

Depends on the SQL script declare in the previous step. HAWQ dispatch two parameters into when the function is invoked. The MARCO PG_FUNCTION_INFO_V1 notice the compiler how to compile and load the function into the system. isnull attribute is an array of bool to indicate every internal slot if is null. The dim attribute indicates the number of values which contain in the vtype.

All expression functions are defined in vtype_ext.c. For convenience, you can use MARCO function to implement, since most calculation functions logic are same but operator different.

IV. Install

After finish coding, compile and install the library first. HAWQ require restart to recognize new thrid-part library load. the library name should record into local_preload_libraries parameter postgresql.conf file both master and segment data directories.

Executing \i create_type.sql all vecotrized type will be created in HAWQ, and open vectorized_executor_enable GUC value to on trigger VE work.

V. Test

We are glad to accept new feature into HAWQ. If you complete a new type or has some enlightenment job, I appreciate it and you can create a PR in HAWQ project in GitHub. We will review and merge it if it is helpful and bug-free.

However, before publishing your code out, you should test your coding in our test framework. For the VE feature test, it is located in src/test/feature/vexecutor directory. Add your own test case and proof everything right.

All in all, this is a brief introduction for vtype extension.