Presto SQL Functions

Rongrong Zhong (rongrong@fb.com)

Facebook

BIG NEWS

- Simple SQL-invoked function is available to use
  - Supporting expression as SQL functions
  - Dynamic SQL Functions are not part of Presto codebase. They are stored externally and managed by a `FunctionNamespaceManager`. You need a MySQL database to use the out-of-box version.
  - Still WIP so please open issues if you see any problems. Feedback is a gift!
- We are working on remote function support (ETA 2020H2)
  - This could enable arbitrary functions that cannot run within worker JVM: unreliable Java functions, C++, Python, …

Agenda

- How to setup and use the SQL function feature
- Updates on remote function support
- Q & A

How to Setup SQL-Invoked Functions

- Configure `FunctionNamespaceManager`
  - https://prestodb.io/docs/current/admin/function-namespace-managers.html

To instantiate a MySQL-based function namespace manager that manages catalog `example`, administrator needs to first have a running MySQL server. Suppose the MySQL server can be reached at `localhost:1080`, add a file `etc/function-namespace/example.properties` with the following contents:

```
function-namespace-manager.name=mysql
database-url=localhost:1080
function-namespaces-table-name=example_function_namespaces
functions-table-name=example_sql_functions
```

When Presto first starts with the above MySQL function namespace manager configuration, two MySQL tables will be created if they do not exist.

- `example_function_namespaces` stores function namespaces of the catalog `example`.
- `example_sql_functions` stores SQL-invoked functions of the catalog `example`.

Multiple function namespace managers can be instantiated by placing multiple properties files under `etc/function-namespace`. They may be configured to use the same tables. If so, each manager will only create and interact with entries of the catalog to which it binds.

To create a new function namespace, insert into the `example_function_namespaces` table:

```
INSERT INTO example_function_namespaces (catalog_name, schema_name)
VALUES('example', 'test');
```

What is a Function Namespace (Manager)

- A function namespace is a special `catalog.schema` that contains functions
- A function namespace manager manages function catalogs (a connector for functions, no tables)
- We separated function management from connector API for flexibility (SQL functions can be used across connectors)

Writing SQL Function

Return type

Qualified function name

Parameter name

Parameter type

```
CREATE FUNCTION example.array.array_sum (input array<double>)
RETURNS double
COMMENT 'Calculate sum of all array elements'
RETURNS NULL ON NULL INPUT
DETERMINISTIC
RETURN REDUCE(input, double'0.0', (s, x) -> s + COALESCE(x, double'0.0'), s -> s)
;
```

Call convention
(or CALLED ON NULL INPUT)

Determinism
(or NOT DETERMINISTIC)

Function body (RETURN expression)

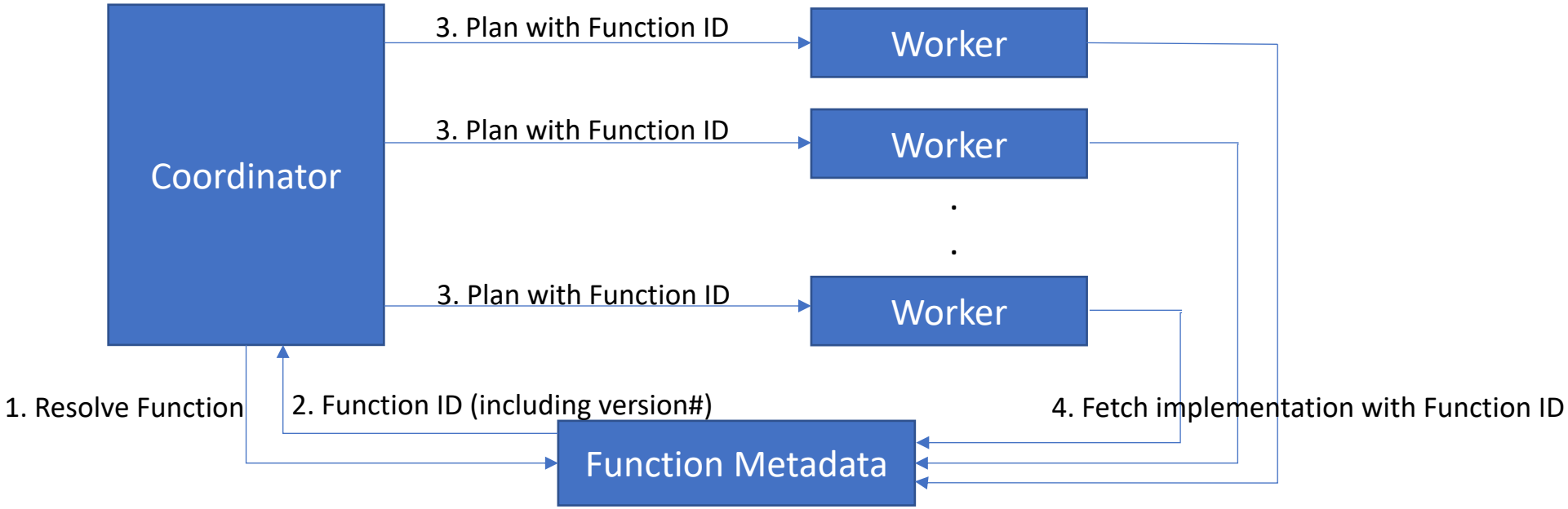
Using SQL Function

```
SELECT example.array.array_sum(array1)
FROM my_table
WHERE example.array.array_sum(array2) > 100
```

Use fully qualified function name

Modifications are versioned

- A query is guaranteed to use the same version through out execution



Coming up

- Built-in SQL functions
- Function access control

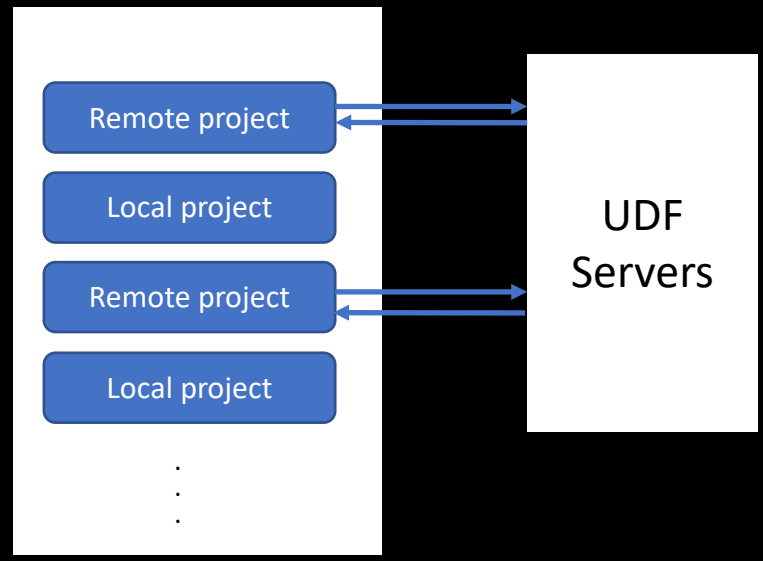
Remote Functions

- Augment function metadata to indicate whether a function is local or remote
- Function namespace managers that manage remote functions know where to run them
- Planner change to allow batch processing

Remote UDF Support (GitHub WIP Issue)

```
presto:di> explain (type distributed) select testing.test.local_foo(x), testing.test.remote_foo(testing.test.remote_foo(x+1) + 1), testing.test.remote_foo(x) from (values (1), (2), (3)) t(x);
                                                Query Plan
-----------------------------------------------------------------------------------------------------------------------
 Fragment 0 [SINGLE]
     Output layout: [local_foo, remote_foo, remote_foo_2]
     Output partitioning: SINGLE []
     Stage Execution Strategy: UNGROUPED_EXECUTION
     - Output[_col0, _col1, _col2] => [local_foo:bigint, remote_foo:bigint, remote_foo_2:bigint]
             Estimates: {rows: 3 (81B), cpu: 393.00, memory: 0.00, network: 0.00}
             _col0 := local_foo
             _col1 := remote_foo
             _col2 := remote_foo_2
         - Project[REMOTE] => [local_foo:bigint, remote_foo:bigint, remote_foo_2:bigint]
                 Estimates: {rows: 3 (81B), cpu: 393.00, memory: 0.00, network: 0.00}
                 remote_foo := testing.test.remote_foo(add_17)
             - Project[LOCAL] => [local_foo:bigint, add_17:bigint, remote_foo_2:bigint]
                     Estimates: {rows: 3 (81B), cpu: 312.00, memory: 0.00, network: 0.00}
                     add_17 := (testing_test_remote_foo) + (expr_16)
                 - Project[REMOTE] => [local_foo:bigint, testing_test_remote_foo:bigint, expr_16:bigint, remote_foo_2:bigint]
                         Estimates: {rows: 3 (108B), cpu: 231.00, memory: 0.00, network: 0.00}
                         testing_test_remote_foo := testing.test.remote_foo(cast_15)
                         remote_foo_2 := testing.test.remote_foo(cast_20)
                     - Project[LOCAL] => [local_foo:bigint, cast_15:bigint, expr_16:bigint, cast_20:bigint]
                             Estimates: {rows: 3 (108B), cpu: 123.00, memory: 0.00, network: 0.00}
                             local_foo := testing.test.local_foo(CAST(field AS bigint))
                             cast_15 := CAST((field) + (INTEGER 1) AS bigint)
                             expr_16 := BIGINT 1
                             cast_20 := CAST(field AS bigint)
                         - LocalExchange[ROUND_ROBIN] () => [field:integer]
                                 Estimates: {rows: 3 (15B), cpu: 15.00, memory: 0.00, network: 0.00}
                             - Values => [field:integer]
                                     Estimates: {rows: 3 (15B), cpu: 0.00, memory: 0.00, network: 0.00}
                                     (INTEGER 1)
                                     (INTEGER 2)
                                     (INTEGER 3)

(1 row)
```



Q & A