SkyhookDM projection-only pushdown and Arrow dataset integration into Skyhook objects

Xiongfen Song

(Mentored by Jeff LeFevre, Ivo Jimenez, Noah Watkins)
SkyhookDM – “Programmable storage”

• Utilizes and extends Ceph distributed object storage with customized C++ "object classes”

• Supports database operations such as SELECT, PROJECT, AGGREGATE to be offloaded (i.e., pushed down) directly into the object storage layer

• Supports row-based Flatbuffers and column-based Arrow
Goal Overview

• Phase 1: Skyhook projection-only pushdown for Arrow tables

• Phase 2: In storage Arrow ListArray datatype operations

• Phase 3: Setup Arrow-native SkyhookDM Architecture
Phase 1: Why we want to do this?

For example:

`SELECT a, b WHERE a < 2 && c > 1` => ["a", "b", "c"]

`SELECT a, b WHERE a > 1` => ["a", "b"]

Figure 1.1: Physical layout of column-oriented vs row-oriented databases.
Input Query
projection: muon
predicate: A=true, B=true

run-query.cc
Set query schema and query predicate for cls

launch aio_exec

exec_query_op
process query using cls

return raw_result

query.cc

if more processing
query.cc
use sky_qry_schema and sky_qry_preds to process the previous result

What we want to implement.

Modify the query schema as (muon, A, B) and query predicate as empty, pass it to cls.

use_cls returns all columns which are related to projection and predicate

Currently, only if luse_cls, it is possible to do more processing

Real predicate will be done in query.cc's more_processing

Use more processing, sky_qry_schema is muon, sky_qry_preds is A=true, B=true
Major commits:

- Add a new flag pushdown-cols-only (default false) to the run-query.cc. When the flag is set, the query execution will only push down columns projection to storage level and leave predicates processing to the more_processing step in query.cc.
- Reconstruct schema and preds in more processing, fixed print data
- Append columns directly when there is no predicate and row_nums is empty
- Extract indexing lookup from exec_query_op

See https://github.com/uccross/skyhookdm-ceph-cls/pull/55
Phase 2: List array and reducers operations

• A common data format in HEP data is Jagged/Awkward Array.
• Currently these jagged arrays are stored in skyhook as Arrow::List types.
• Each entry in the column (row) is a list of arbitrary and non-uniform length.

• We would like to implement reducer ops like: min, max, count...
• However, we found arrow compute APIs
Apache Arrow compute APIs

api_scalar: Add(), Subtract(), And, Or, XOR, IsIn()...

api_aggregate

api_vector: Filter(), Take()...

### Aggregations

<table>
<thead>
<tr>
<th>Function name</th>
<th>Arity</th>
<th>Input types</th>
<th>Output type</th>
<th>Options class</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>Unary</td>
<td>Any</td>
<td>Scalar Int64</td>
<td>CountOptions</td>
</tr>
<tr>
<td>mean</td>
<td>Unary</td>
<td>Numeric</td>
<td>Scalar Float64</td>
<td></td>
</tr>
<tr>
<td>min_max</td>
<td>Unary</td>
<td>Numeric</td>
<td>Scalar Struct (1)</td>
<td>MinMaxOptions</td>
</tr>
<tr>
<td>sum</td>
<td>Unary</td>
<td>Numeric</td>
<td>Scalar Numeric (2)</td>
<td></td>
</tr>
</tbody>
</table>
Phase 3: Move to Arrow!
Works done for phase 3

• Issue 1: arrow-cls setup for running unit tests

• Issue 2: InMemoryFragment interface to object store’s API
In Memory Fragment

Create Arrow table

Get TableBatchReader and Schema from arrow table

Generate a InMemoryFragment or dataset

Create a ScannerBuilder from fragment, schema

Filter and Projection operations can be done here

Get the Scanner, read the filtered data and convert back to table!
In Memory Fragment

```cpp
class ARROW_DS_EXPORT Scanner {

    /// \brief Scanner is a materialized scan operation with context and options
    /// bound. A scanner is the class that glues ScanTask, Fragment,
    /// and Dataset. In python pseudo code, it performs the following:
    ///
    ///    def Scan():
    ///        for fragment in self.dataset.GetFragments(this.options.filter):
    ///            for scan_task in fragment.Scan(this.options):
    ///                yield scan_task

    /// \brief Set the filter expression to return only rows matching the filter.
    ///
    /// The predicate will be passed down to Sources and corresponding
    /// Fragments to exploit predicate pushdown if possible using
    /// partition information or Fragment internal metadata, e.g. Parquet statistics.
    ///
    /// \param[in] filter expression to filter rows with.
    ///
    /// \return Failure if any referenced columns does not exist in the dataset's
    ///     Schema.

    Status Filter(std::shared_ptr<Expression> filter);
```
Thank you!