Implement Skyhook row index filter operation, Awkward list in-storage operations and Coffea processor/executor

Skyhook project is an extension of Ceph for the scalable storage of tables and for offloading common data management operations on them, such as selection, projection, and aggregation, as well as indexing and user-defined functions. It can process several data formats from disk. For example, row-based processing via Google Flatbuffers format and col-based processing via Apache Arrow formats.

Phase 1:

Assume we have an Arrow format table T extracted from NanoAOD files. Skyhook has partitioned the Arrow data by column, each column is stored in one or more objects. We have the following columns: bool type A, bool type B, bool type C, (Jagged array) list type muon_pt and list muon_phi. The Boolean type A, B, C are trigger columns, which indicate we should keep the row or not. They are primitive arrays (bool type) in Arrow. The trigger columns are used as a Boolean selection, which is an operation that can be offloaded to storage. It returns the result to our driver that then performs the appropriate logical operations (AND/OR/UNION/INTERSECTION of matching rows) of the trigger columns query results, then it broadcasts the matching row numbers back to the appropriate objects in Skyhook to retrieve the analysis columns data from the specified rows. Given the distributed query and the number of matching row indices could be very large, we should keep the broadcasted matching row numbers result as small as possible (possibly bitmask or compression if appropriate).

The jagged array/Arrow list type columns muon_pt and muon_phi are columns of interest in physics analysis. What we want to implement then is to select the rows according to the Boolean predicate (return true) from trigger columns. This is similar to the Arrow's take operation. Suppose we have a column or array storing row index numbers, the resulting table will be the specific rows extracted from the input (muon_pt and muon_phi). Also, there is no explicit row index (or ID) stored in the Arrow table, so this looks like virtual IDs in column-stores, see overview.

In Skyhook, we already have operations to select certain rows which satisfy predicates like greater than, less than, equal or other value predicates. But we do not have filters for row index only. This operation is to be implemented, we have defined it as SkyhookOpType_in, see this. Our implementation plan is as followed. When applying a filter/list operation, the applyPredicatesArrowCol() will check the specified columns and set the vector with all the row numbers that passed the filter. These are stored in result_rows, and then we can get the matching data for those row numbers.

Phase 2:

In the second phase, if time permits, we will determine a feasible subset of operations that can be applied in storage by Skyhook, and then implement them for each list type (int, float, bool, etc.). A common data format in HEP data is Jagged/Awkward Array. Currently these jagged arrays are stored in skyhook as Arrow::List types. We will be able to get input and feedback from analysts and common query workloads. An example op might be to count the number of

elements with value > X. Some common operations in physics might include mass of A is computed from X cross-join Y, or X self-join X, see this. We do not want to re-implement the awkward array operations, but use awkward as motivation for very simple standard list operations that we could implement on Arrow data in Ceph objects. It will be generally useful for processing Arrow data since lists types in Skyhook are supported in Arrow. The most useful ops to consider would be ones that perform data reduction, and since the op will be applied inside the storage layer, this will reduce the amount of data sent back over the network to the client.

Phase 3:

In phase 3, we plan to implement the cpp version of Coffea (Columnar Object Framework For Effective Analysis). There is one for Spark, we want to create a new processor interface for Skyhook, which will be very similar to the Spark processor/executor interface.

Timeline:

Week 1: Project set up. Get familiar with Skyhook environment and documentation. Read codes in cls_tabular_processing, utils and run-query.

Week 2 - 4: Implement the row index “In” operation in Skyhook.

Week 5-6: Determine a feasible subset of operations that can be applied in storage by Skyhook, and then implement them for each list type (int, float, bool, etc.).

Week 7-8: Implement the Skyhook’s Coffea processor/executor interface.