PROPOSAL TO ENABLE HDFql WITH BATCH-PROCESSING CAPABILITIES

1. INTRODUCTION

Currently, two main approaches (extremes) exist to batch-process HDF5 data: on one end, the usage of existing HDF5 (imperative) APIs, which are performant but somewhat complicated to operate with due to users being exposed to low-level HDF5 details and have to deal with the complexity of accessing multiple HDF5 files, eventually stored across various locations; in the other end, the usage of high-level frameworks such as Apache Drill, which abstract users from HDF5 details at the expense of performance and certain HDF5 functionalities – e.g. hyperslab selections.

The present document proposes a middle-ground solution between the two approaches by extending HDFql with batch-processing features that are easy to use. It consists in allowing HDFql's *SELECT* operation to read and (post-)process multiple datasets/attributes (across various groups and sub-groups) potentially held across multiple HDF5 files (across multiple directories, sub-directories and locations). The proposed extension will effectively lower the complexity of batch-processing HDF5 data through (the execution of) one single (HDFql) operation while guaranteeing excellent performance and availability of HDF5 functionalities.

2. EXAMPLES

As a general rule (and unless certain post-processing functions are employed), when multiple datasets/attributes are selected (i.e. read), the data is sequentially combined and linearized into a one dimensional array (and independently of what the datasets/attributes' number of dimensions and sizes may be). In other words, HDFql's *SELECT* operation reads the selected datasets/attributes and flattens their data as a one dimensional non-overlapping contiguous (big) array. Users may employ other HDFql operations – e.g. *SHOW DIMENSION* – to retrieve information about the datasets/attributes and help interpreting/consuming the one dimensional array in a correct way/appropriate manner. In addition (and unless certain post-processing functions are employed), in case multiple datasets/attributes are selected (i.e. read) and have different data types amongst them, an error is raised.

Some examples are presented next to illustrate how the extension of HDFql's *SELECT* operation to support reading and (post-)processing multiple datasets/attributes (eventually across multiple groups/sub-groups and/or HDF5 files and/or directories/sub-directories/locations) looks in practice and is foreseen to work.

1. **SELECT FROM dset1, dset2, dset3**
    - select (i.e. read) datasets *dset1*, *dset2* and *dset3* from the HDF5 file currently in use (i.e. open)
    - if either *dset1*, *dset2* or *dset3* is missing, an error is raised

2. **SELECT FROM file1.h5 dset1, dset2, file3.h5 dset3**
    - select (i.e. read) datasets *dset1* from file *file1.h5*, *dset2* from the HDF5 file currently in use (i.e. open) and *dset3* from file *file3.h5*
    - if either *dset1*, *dset2* or *dset3* is missing, an error is raised

3. **SELECT FROM dset1, CAST(dset2 AS INT), dset3**
    - select (i.e. read) datasets *dset1*, *dset2* and *dset3* from the HDF5 file currently in use (i.e. open)
    - convert *dset2* as an INT
    - if either *dset1*, *dset2* or *dset3* is missing, an error is raised

4. **SELECT FROM UPPER(dset1, CAST(dset2 AS VARCHAR), dset3)**
    - select (i.e. read) datasets *dset1*, *dset2* and *dset3* from the HDF5 file currently in use (i.e. open)
    - convert *dset2* as a VARCHAR
    - convert all datasets in upper case
    - if either *dset1*, *dset2* or *dset3* is missing, an error is raised

5. **SELECT FROM file1.h5 dset1, CAST(file2.h5 dset2 AS INT), file3.h5 dset3**
    - select (i.e. read) datasets *dset1* from file *file1.h5*, *dset2* from file *file2.h5* and *dset3* from file *file3.h5*
    - convert *dset2* as an INT
    - if either *dset1*, *dset2* or *dset3* is missing, an error is raised

6. **SELECT FROM COUNT(dset1, 10), COUNT(dset2, 15), COUNT(dset3, 20)**
    - select (i.e. read) datasets *dset1*, *dset2* and *dset3* from the HDF5 file currently in use (i.e. open)
    - count the occurrence of value 10 in *dset1*, value 15 in *dset2* and value 20 in *dset3*
    - return the count of occurrences as a one dimensional array of size 3
    - if either *dset1*, *dset2* or *dset3* is missing, an error is raised

7. **SELECT FROM SUM(COUNT(dset1, 10), COUNT(dset2, 15), COUNT(dset3, 20))**
   - select (i.e. read) datasets *dset1*, *dset2* and *dset3* from the HDF5 file currently in use (i.e. open)
   - count the occurrence of value 10 in *dset1*, value 15 in *dset2* and value 20 in *dset3*
   - return the sum of the count of occurrences as a scalar
   - if either *dset1*, *dset2* or *dset3* is missing, an error is raised

8. **SELECT FROM LIKE **/^dset**
   - select (i.e. read) all datasets found recursively (i.e. stored in groups/sub-groups) that have a name starting with *dset* from the HDF5 file currently in use (i.e. open)

9. **SELECT FROM CAST(LIKE **/^dset AS INT)**
   - select (i.e. read) all datasets found recursively (i.e. stored in groups/sub-groups) that have a name starting with *dset* from the HDF5 file currently in use (i.e. open)
   - convert all datasets found as an INT

10. **SELECT FROM /grp LIKE **/^dset**
    - select (i.e. read) all datasets found recursively (i.e. stored in groups/sub-groups) starting from root group *grp* that have a name starting with *dset* from the HDF5 file currently in use (i.e. open)

11. **SELECT FROM file.h5 /grp LIKE **/^dset**
    - select (i.e. read) all datasets found recursively (i.e. stored in groups/sub-groups) starting from root group *grp* that have a name starting with *dset* from file *file.h5*

12. **SELECT FROM file.h5 / LIKE **/^abc, LIKE def$**
    - select (i.e. read) all datasets found recursively (i.e. stored in groups/sub-groups) starting from root group */* that have a name starting with *abc* from file *file.h5* and all datasets that have a name ending with *def* from the HDF5 file currently in use (i.e. open)

13. **SELECT FROM LIKE **/^dset WHERE DATA TYPE == FLOAT AND attrib > 10**
    - select (i.e. read) all datasets found recursively (i.e. stored in groups/sub-groups) that have a name starting with *dset* from the HDF5 file currently in use (i.e. open)
    - only the datasets that are of data type float and have an attribute named *attrib* with a value greater than *10* are selected

14. **SELECT FROM file1.h5 / LIKE **/^abc, file2.h5 /grp LIKE **/def$ WHERE EXISTS attrib**
    - select (i.e. read) all datasets found recursively (i.e. stored in groups/sub-groups) starting from root group */* that have a name starting with *abc* from file *file1.h5* and all datasets found recursively (i.e. stored in groups/sub-groups) starting from root group *grp* that have a name ending with *def* from file *file2.h5*
    - only the datasets that have an attribute named *attrib* are selected

15. **SELECT FROM ALL USE FILE dset**
    - select (i.e. read) dataset *dset* from all HDF5 files currently in use (i.e. open)
    - if *dset* is missing in one of the HDF5 files currently in use (i.e. open), an error is raised

16. **SELECT FROM ALL USE FILE dset SKIP**
    - select (i.e. read) dataset *dset* from all HDF5 files currently in use (i.e. open)
    - if *dset* is missing in one of the HDF5 files currently in use (i.e. open), the file is skipped (i.e. no error is raised)

17. **SELECT FROM ALL USE FILE LIKE **/^dset WHERE attrib < 15**
    - select (i.e. read) all datasets found recursively (i.e. stored in groups/sub-groups) that have a name starting with *dset* from all HDF5 files currently in use (i.e. open)
    - only the datasets that have an attribute named *attrib* with a value lower than *15* are selected

18. **SELECT FROM ALL USE FILE /grp LIKE **/^dset WHERE attrib > 100**
    - select (i.e. read) all datasets found recursively (i.e. stored in groups/sub-groups) starting from root group *grp* that have a name starting with *dset* from all HDF5 files currently in use (i.e. open)
    - only the datasets that have an attribute named *attrib* with a value greater than *100* are selected

19. **SELECT FROM ALL USE FILE dset1, dset2 WHERE id >= 20 AND id <= 25**
    - select (i.e. read) datasets *dset1* and *dset2* from all HDF5 files currently in use (i.e. open)
    - only the datasets that have an attribute named *id* with a value between *20* and *25* inclusive are selected
    - if either *dset1* or *dset2* is missing in one of the HDF5 files currently in use (i.e. open), an error is raised

20. **SELECT FROM USE FILE LIKE ^Y2021 dset1, dset2, dset3**
    - select (i.e. read) datasets *dset1*, *dset2* and *dset3* from all HDF5 files currently in use (i.e. open) that have a name starting with *Y2021*
    - if either *dset1*, *dset2* or *dset3* is missing in one of the HDF5 files currently in use (i.e. open), an error is raised

21. **SELECT FROM USE FILE LIKE October|November LIKE ^dset[1|2|3]$**
    - select (i.e. read) all datasets that are named either *dset1*, *dset2* or *dset3* from all HDF5 files currently in use (i.e. open) that have either *October* or *November* in their names

22. **SELECT FROM /data LIKE **/^test.h5$ /grp LIKE dset WHERE color == "Red"**
    - select (i.e. read) all datasets that have *dset* in their names starting from root group *grp* from all HDF5 files found recursively (i.e. stored in directories/sub-directories) that are named *test.h5* starting from root directory *data*
    - only the datasets that have an attribute named *color* with a value equal to *Red* are selected

3. EXTENSION

Canonically speaking, the *SELECT* operation looks as follows with the extension that enables HDFql with batch-processing capabilities:

SELECT FROM [DATASET | ATTRIBUTE] {select | post_process} [WHERE condition]

select := select_list | select_like | select_all_use_file | select_use_file

select_list := {{[file_name] object} | post_process_A} [, {{[file_name] object} | post_process_A}]*

select_like := {[object_container_name] LIKE {object | post_process_B}} | {file_name [, file_name]* LIKE {object | post_process_B}} | {[directory_container_name] LIKE file_name object [, object]*} | {[directory_container_name] LIKE file_name [object_container_name] LIKE {object | post_process_B}}

select_all_use_file := ALL USE FILE {{{object | post_process_B} [, {object | post_process_B}]* [SKIP]} | {[object_container_name] {{LIKE object} | post_process_C}}}}

select_use_file := USE FILE {{{file_name [, file_name]*} | {LIKE file_name}} {{{object | post_process_B} [, {object | post_process_B}]*} | {[object_container_name] {{LIKE object} | post_process_C}}}} post_process := COUNT({select | post_process} [, value]) | SUM(select | post_process) | UPPER(select | post_process) | CAST({select | post_process} AS {TINYINT | SMALLINT | INT | …}) | …

post_process_A := COUNT({{[file_name] object} | post_process_A} [, value]) | SUM({[file_name] object} | post_process_A) | UPPER({[file_name] object} | post_process_A) | CAST({{[file_name] object} | post_process_A} AS {TINYINT | SMALLINT | INT | …}) | …

post_process_B := COUNT({object | post_process_B} [, value]) | SUM(object | post_process_B) | UPPER(object | post_process_B) | CAST({object | post_process_B} AS {TINYINT | SMALLINT | INT | …}) | …

post_process_C := COUNT({{LIKE object} | post_process_C} [, value]) | SUM({LIKE object} | post_process_C) | UPPER({LIKE object} | post_process_C) | CAST({{LIKE object} | post_process_C} AS {TINYINT | SMALLINT | INT | …}) | …

condition := NOT* {condition_type | condition_data_type | condition_exists | condition_values | (condition)} [{AND | OR} NOT* {condition_type | condition_data_type | condition_exists | condition_values | (condition)}]*

condition_type := TYPE {= | !=} {GROUP | DATASET | ATTRIBUTE | [SOFT] LINK | EXTERNAL LINK}

condition_data_type := DATA TYPE {= | !=} {TINYINT | SMALLINT | INT | …}

condition_exists := EXISTS {GROUP | DATASET | ATTRIBUTE | [SOFT] LINK | EXTERNAL LINK}? object_name

condition_values := {VALUES | {{DATASET | ATTRIBUTE}? {object_name | post_process_B}} {= | != | > | >= | < | <=}} value

object := object_name[hyperslab | point | chunk]

hyperslab := [start]:[stride]:[count]:[block] [, [start]:[stride]:[count]:[block]]* [{OR | AND | XOR | NOTA | NOTB} [start]:[stride]:[count]:[block] [, [start]:[stride]:[count]:[block]]*]*

point := coord [, coord]* [; coord [, coord]*]*

chunk := chunk_number [, chunk_number]*