AvroStorage

We want to use Pig to process arbitrary Avro data and store results as Avro files. This page summarizes how to use AvroStorage, a Pig load/store func for Avro data.

1. Introduction

AvroStorage() extends two PigFuncs: LoadFunc and StoreFunc, and is used to load and store Avro data in Pig scripts.

Due to discrepancies of Avro and Pig data models, AvroStorage has:

- Limited support for "record": we do not support recursively defined record because the number of fields in such records is data dependent. For instance, *{"type":"record","name":"LinkedListElem", "fields":[{"name":"data","type":"int"},{"name":"next", "type":["null","LinkedListElem"]}]}*;
- Limited support for "union": we only accept nullable union like ["null", "some-type"].

For simplicity, we also make the following assumption:

- If the input directory is a leaf directory, then we assume Avro data files in it have the same schema;
- If the input directory contains sub-directoies, then we assume Avro data files in all sub-directories have the same schema.

Below is an example pig script using AvroStorage.

```
REGISTER avro-1.4.0.jar
REGISTER json-simple-1.1.jar
REGISTER piggybank.jar
REGISTER jackson-core-asl-1.5.5.jar
REGISTER jackson-mapper-asl-1.5.5.jar

fs -rmr $home/testOut/testCnt;

avro = LOAD '$home/avro/TestInput/case1/part-00000.avro'
USING AvroStorage ();

groups = GROUP avro BY value.member_id;
sc = FOREACH groups GENERATE group AS member_id, COUNT(avro) AS cnt;

STORE sc INTO '$home/testOut/testCnt'
USING AvroStorage ();
```

Users can provide parameters to AvroStorage when used as a LoadFunc or StoreFunc.

2. Input Parameters (as a LoadFunc)

AvroStorage assumes that all avro files in sub-directories of an input directory share the same schema and it by default does schema check. This process may take seconds when the input directory contains many sub-directories. Users can use option "no_schema_check" to disable the schema check.

```
-- does schema check of sub-directories under /testIn/dir
data = LOAD '/testIn/dir' USING storage.avro.AvroStorage ();

-- disable schema check of sub-directories under /testIn/dir
data = LOAD '/testIn/dir' USING storage.avro.AvroStorage ('no_schema_check');
```

Additionally if there are multiple Avro files in different directories having schemas varying by a column. The first schema will be used as an input to read all other Avro files in that directory. This can sometimes result in undesirable results.

3. Input Parameters (as a StoreFunc)

No Parameter

Users can choose not to provide any parameter to AvroStorage and Avro schema of output data is derived from its Pig schema. This may result in undesirable schemas due to discrepancies of Pig and Avro data models or problems of Pig itself:

- The derived Avro schema will wrap each (nested) field with a nullable union because Pig allows NULL values for every type and Avro doesn't. For instance, when you read in Avro data of schema *"boolean"* and store it using AvroStorage(), you will get *["null","boolean"]*.

- The derived Avro schema may contain unwanted tuple wrappers because: 1) Pig only generates tuples; 2) items of Pig bags can only be tuples. AvroStorage can automatically get rid of such wrappers, but sometimes you still see them as in example B.
- Under certain circumstances, for instance when you want to store a Pig map or result of *UNION* operator, Pig doesn't provide concrete schema information and we cannot derive Avro schema.

We thus highly recommend users provide output schema information using the following options.

Pig results are tuples which may contain multiple fields. Users can provide parameters applying to all fields, denoted as global parameters, or specific fields, denoted as field parameters.

Global Parameters

- debug n
  Users can check debug information using this option where *n* represents the debug level:
  1. Show names of some function calls when *n*>=*3*;
  2. Show details when *n*>=*5*.
- index n
  Use this option when you want to store data in different ways. We have to use this hack because Pig (release 0.7) doesn't allow us to access result schema in backend. Please check example A for details.
- nullable boolean
  If users are sure results don't contain *NULL* values, they can use this option. Note that this option is applied recursively to all (nested) fields of results.
- schema str
  Users can provide the full schema using this option, where *str* is a string representation of Avro schema.
- When Pig provides schema information of results, AvroStorage would do a schema compatibility check based on the following table where Y means compatible, N, not compatible and P, potentially compatible and needs further check.

	record	array	map	enum	fixed	bytes	boolean	int	long	float	double	string
tuple	P	P	P	P	P	P	P	P	P	P	P	P
bag		P										
map			Y									
chararray				P								Y
bigchararray				P								Y
bytearray					P	Y						
boolean							Y					
int							Y	Y	Y	Y	Y	
long									Y	Y	Y	
float										Y	Y	
double											Y	
undefined	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

- In the above table, Pig tuple is potentially compatible with any Avro type because users can use this feature to get rid of unwanted tuple wrappers over primitive values, bags and maps. Please refer to example B for more details.
- Pig type *undefined* means Pig doesn't provide schema of such output, like map value and outcome of UNION operator, and users *must* provide corresponding schema information because we cannot derive Avro schema from *undefined* type.
- Conversion of Pig integer to Avro boolean is supported because boolean is not an external type in Pig.
- Users can use Pig to do more type casting, like float=>int.

- same path
  Users can provide the full schema using this option, where *path* points to a directory of Avro files or a single Avro file. AvroStorage will store data using the same schema as in *path*. This option is useful when users are interested in simple processing like filtering and want to store output using the input schema. This option is undefined on the load statement.
- multiple_schemas
  Users can use this feature if they have multiple avro files each having a different schema and they want to reconcile them. The logic for "multiple_schemas" relies on the methods "mergeSchema" and "mergeType" in the class "AvroStorageUtils" to do the merging. For merging complex Avro types, null will be returned unless they are both the same type. Also note that not every primitive Avro type can be merged. For types that cannot be merged, null is returned if the merging does not happen properly. Default values can be merged if they are the same type and if one of them has a default value. The reconciled schema can be obtained by using a "describe" in the Pig script. Merging schemas does not work properly when reconciling nested records. It also does not exhibit the right behavior when merging different primitive types having the same default value.

Field Parameters

Users can provide schema information for specific fields. The index of field, *n*, starts from 0.

- field<n> notnull
  This indicates the *n* th field (and its nested fields) in the output tuple is *notnull*.

- data path
- field<n> def:name
  Users can provide predefined schemas in Avro files using option -data path, where *path* points to a directory of Avro files or a single Avro file. This is used together with field parameter field<n> def:name. AvroStorage internally constructs two maps: map[typeName]=>schema and map [fieldName]=>schema and users can specify which schema to use by providing corresponding *name*. This option is useful when users want to do simple processing of input data (like filtering and projection) and store it using predefined schemas in input. Please refer to example C for more details.

- field<n> str
  Users can directly specify the schema of field *n* where *str* is a string representation of Avro schema. The usage of this option is similar to schema str except that the schema is only applied to the *n* th field.

Put Parameters in a JSON record

Users can put all above parameters in a JSON record. For instance

```
....
STORE data INTO 'out_location'
USING AvroStorage(
'{ "debug": 5,
"index": 2,
"schema": {"type":"record","name":"Y",
"fields":[{"name":"b1","type":"float"},
{"name":"b2","type":"float"}]}
}');
```

4. Examples

A. How to store data in different ways.

Notice that users can pass in parameters as string list or a JSON record.

```
REGISTER avro-1.4.0.jar
REGISTER json-simple-1.1.jar
REGISTER piggybank.jar
REGISTER jackson-core-asl-1.5.5.jar
REGISTER jackson-mapper-asl-1.5.5.jar

fs -rmr testOut/testSplit1_1;
fs -rmr testOut/testSplit1_2;

B = LOAD 'testIn/B' AS (b1:int,b2:int);
DUMP B;
/* get
(2,4)
(8,9)
(1,3)
*/

SPLIT B INTO X IF b1 < 3, Y IF b1 > 7;
STORE X INTO 'testOut/testSplit1_1'
USING org.apache.pig.impl.io.avro.AvroStorage(
'index', '1',
'schema',
' {"type":"record","name":"X",
"fields":[{"name":"b1","type":"int"},
{"name":"b2","type":"int"}]}');

STORE Y INTO 'testOut/testSplit1_2'
USING org.apache.pig.impl.io.avro.AvroStorage(
'{"index": 2,
"schema": {"type":"record","name":"Y",
"fields": [{"name":"b1","type":"float"},
{"name":"b2","type":"float"}
]
}
}');
```

B. How to get rid of unwanted tuple wrappers

```
REGISTER avro-1.4.0.jar
REGISTER json-simple-1.1.jar
REGISTER piggybank.jar
REGISTER jackson-core-asl-1.5.5.jar
REGISTER jackson-mapper-asl-1.5.5.jar

fs -rmr testOut/testPrimitive4;

in = LOAD 'avro/testInput/case2/part-00000.avro' USING AvroStorage ();

B = GROUP in ALL;
sum = FOREACH B GENERATE COUNT(in) ;

STORE sum INTO 'testOut/testPrimitive4'
USING AvroStorage('schema','"long"');
```

Note if no input parameter specified, schema data of output is

```
{"type":"record","name":"TUPLE",
"fields":[{"name":"FIELD","type":["null","long"]}]}
```

C. How to use predefined schemas in data files

```
REGISTER avro-1.4.0.jar
REGISTER json-simple-1.1.jar
REGISTER piggybank.jar
REGISTER jackson-core-asl-1.5.5.jar
REGISTER jackson-mapper-asl-1.5.5.jar

fs -rmr testOut/testFields;

in = LOAD 'inputData'
USING AvroStorage ();

in = FILTER in BY pageNumer > 1;

out = FOREACH in GENERATE impressionDetails.id as id,
impressionDetails.type as type;

STORE out INTO 'testOut/testFields' USING AvroStorage (
'data', 'inputData',
'field0', 'def:impressionDetails.id', /* use field name to look up schema*/
'field1', 'def:ItemType'); /* use type name to look up schema */
```

Suppose *inputData* contains Avro data with the following schema:

```
{
   "name" : "ImpressionSetEvent",
   "type" : "record",
   "fields" : [
      {
         "name" : "pageNumber",
         "type" : [ "int", "null" ]
      },
      {
         "name" : "impressionDetails",
         "type" : {
            "name" : "ImpressionDetailsRecord",
            "type" : "record",
            "fields" : [
               {
                  "name" : "id",
                  "type" : "int"
               },
               {
                  "name" : "type",
                  "type" : {
                     "type" : "enum",
                     "name" : "ItemType",
                     "symbols" : [ "person", "job", "group", "company", "nus",
"news", "ayn" ]
                  }
               },
               {
                  "name" : "details",
                  "type" : {
                     "type" : "map",
                     "values" : "string"
                  }
               }
            ]
         }
      }
   ]
}
```

AvroStorage will construct two maps. One maps from type name to schema as

type name	schema
ImpressionSetEvent	the whole schema
ImpressionDetailsRecord	{"type": "record", "name": ImpressionDetailsRecord","fields" : [{"name":"itemId", "type":"int"}, {"name":"itemType", "type":{"type":"enum", "name":"ItemType","symbols":["person", "job", "group", "company", "nus", "news", "ayn"]}}, {"name":"details","type":{"type":"map","values":"string" }}
ItemType	{"type":"enum", "name":"ItemType","symbols":["person", "job", "group", "company", "nus", "news", "ayn"]}

The other maps from field names to schema as:

field name	schema
pageNumber	["int", "null"]
impressionDetails	ImpressionDetailsRecord
impressionDetails.id	int
impressionDetails.type	ItemType
impressionDetails.details	{"type":"map","values":"string" }

Users can specify which schema to use by field name (as in field 0) or type name (as in field 1).

D. How to store results using schema of existing avro file

```
REGISTER avro-1.4.0.jar
REGISTER json-simple-1.1.jar
REGISTER piggybank.jar
REGISTER jackson-core-asl-1.5.5.jar
REGISTER jackson-mapper-asl-1.5.5.jar

data = LOAD 'input/part-00000.avro' USING AvroStorage ();
ret = FILTER data BY value.member_id > 2000;

STORE ret INTO 'output' USING AvroStorage (
'same', 'input/part-00000.avro');
```

5. Known Issues

- AvroStorage does not load JSon encoded Avro files
- Map data creation for Avro in a Pig script has some issues. It has not been implemented in the AvroStorage. The below script does not work

```
A = load 'complex.txt' using PigStorage('\t') as (mymap:map[chararray], mytuple:(num:int, str:chararray,
dbl:double),
bagofmap:{t:(m:map[chararray])}, rownum:int);
describe A;
store A into 'avro_complex.out' USING org.apache.pig.piggybank.storage.avro.AvroStorage();
```

- Column Pruning is not implemented in the current AvroStorage

6. Related Work

- PIG-794 uses Avro serialization in Pig.
- AVRO-592 does Pig to Avro translation but does not load general Avro data (it only accepts Avro data generated by itself).
- AvroStorageUtils logic for merge multiple schemas.

7 Acknowledgments

This documentation was originally written by Lin Guo, and appeared at http://linkedin.jira.com/wiki/display/HTOOLS/AvroStorage+-+Pig+support+for+Avro+data