Arca Update

Daniel B Davidson

09 January 2013

1 Data Issues

As previously discussed, there are data corruption issues relating to order ids being added multiple times, without having been deleted. I have worked with NYSE support and apparantly there were data issues and the response is they have been resolved. To verify I think we need to re-retrieve the data. I'll see if I can do this, but it may be better for Geoff to, since the site mentioned in the email from Eric requires login, which he probably has.

2 Parsing and Storing In Common Book Format

Our last email discussed these next steps.

2.1 Convert Raw ARCA To HDF5 AMD (Add/Modify/Delete) and Book Format

The script `arca_parser.py` for this has been created. I used the output hdf5 file for convenience in trouble-shooting. It is easier to look at hdf5 data when sizes are large than trying to open in an editor or use scripts to pull out the information. There is now the capability to take the `arcabookftpDATE.gz` files and create directly from them the corresponding h5 files with the same data, less the fields we know we do not use. There may no longer be a need to use this, however.

The next step was to generate the book data by processing the AMD records. This is done in the same script and is now the default behavior. So, the creation of AMD h5 files is not necessary/useful assuming the script works more generally (to be seen). I have run the script on the file in the good range, dated 20110722. The resulting h5 book file has been put in `/datastore/dbd/`. It looks good superficially. We could write a script to compare this data against Geoff's if you feel worth-while.

The standard output of running the script on the original compressed ARCA data file produces the following summary information (described later).

```
Summary of: 2011-07-22_ETF_CSCO_MSFT_HD_LOW.h5
        is_valid: 1
        src_file: arcabookftp20110722.csv.gz
        src_file_size: 2987386971
        src_file_mtime: 2012-12-21 15:08:03.890721-06:00
        parse_start: 2013-01-09 06:49:04.225833-06:00
        parse_stop: 2013-01-09 08:03:55.073053-06:00
        data_start: 2011-07-22 03:00:00.095000-05:00
        data_stop: 2011-07-22 19:00:00.310000-05:00
        processed: 246513779
        irrelevants: 1742877
        total warnings: 19
```

2.2 Book Format

The format of the book data will be the same format discussed previously. The only changes I have made are:

- `timestamp_s` has been added. The `_s` suffix indicates string so a person reading the file can see the timestamps involved. **All string timestamps are Chicago time**.

- `symbol` has been removed. We may want to add this back, depending on how the data is to be used in analysis. Any comments are appreciated.

2.2.1 File/Book Format Details

To allow for analysis on multiple symbols, the h5 file stores each symbol's book data in it's own group.

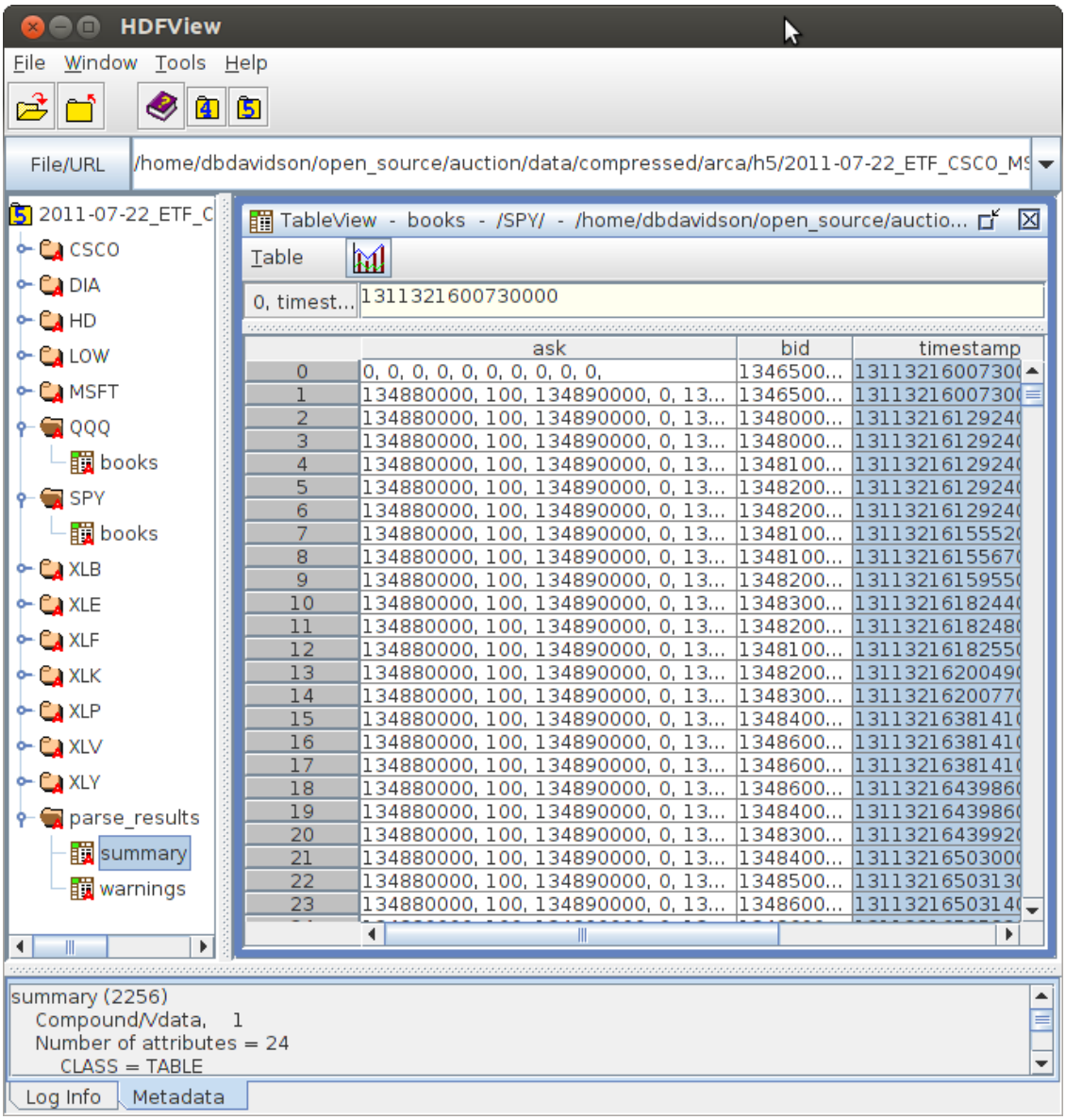

Figure 1: hdfview on Resulting Book Data

So in the example shown this set of symbols (my default) was pulled in:

```
options.symbols = [
    'SPY', 'DIA', 'QQQ',
    'XLK', 'XLF', 'XLP', 'XLE', 'XLY', 'XLV', 'XLB',
    'CSCO','MSFT', 'HD', 'LOW',
    ]
```

In `hdfview`, the folders are the `groups`. Each symbol gets its own `group`, think of it like a directory. Inside each symbol `group` is a single dataset called `books`. The books contain the data defined in the previous script. The fields are as follows:

Field	Data Type	Description
timestamp	Int64Col	The timestamp as UTC.
timestamp_s	StringCol(16)	The timestamp as string in Chicago time for human consumption
ask	Int64Col(shape=(5,2))	Ask data
bid	Int64Col(shape=(5,2))	Bid data

Again, the `symbol` field was removed since all records for a given symbol are stored together in its own dataset. An argument for including it anyway would be that, it may be desirable to store time ordered data of multiple symbols in the same dataset and reuse this existing structure, rather than broken out by symbol. This is where the patterns to be used for analysis are important. I don't think it makes much difference either way, since the idea is to make it easy to write code to combine/merge the data of multiple symbols. Essentially all that is needed is a function that combines all book updates spanning multiple symbols into a serial stream ordered by date. I have created a script `book_file.py` which does this, but I'll need some analysis work requirements to test it out when looking to combine data from mutiple symbols.

2.2.2 Timestamps

All timestamps (i.e. the really big numbers) are stored as UTC. If we do the same for CME and any other data, then no conversion is needed to align data by time. In the process of creating the data the timestamps are converted to Chicago time. So, for example:

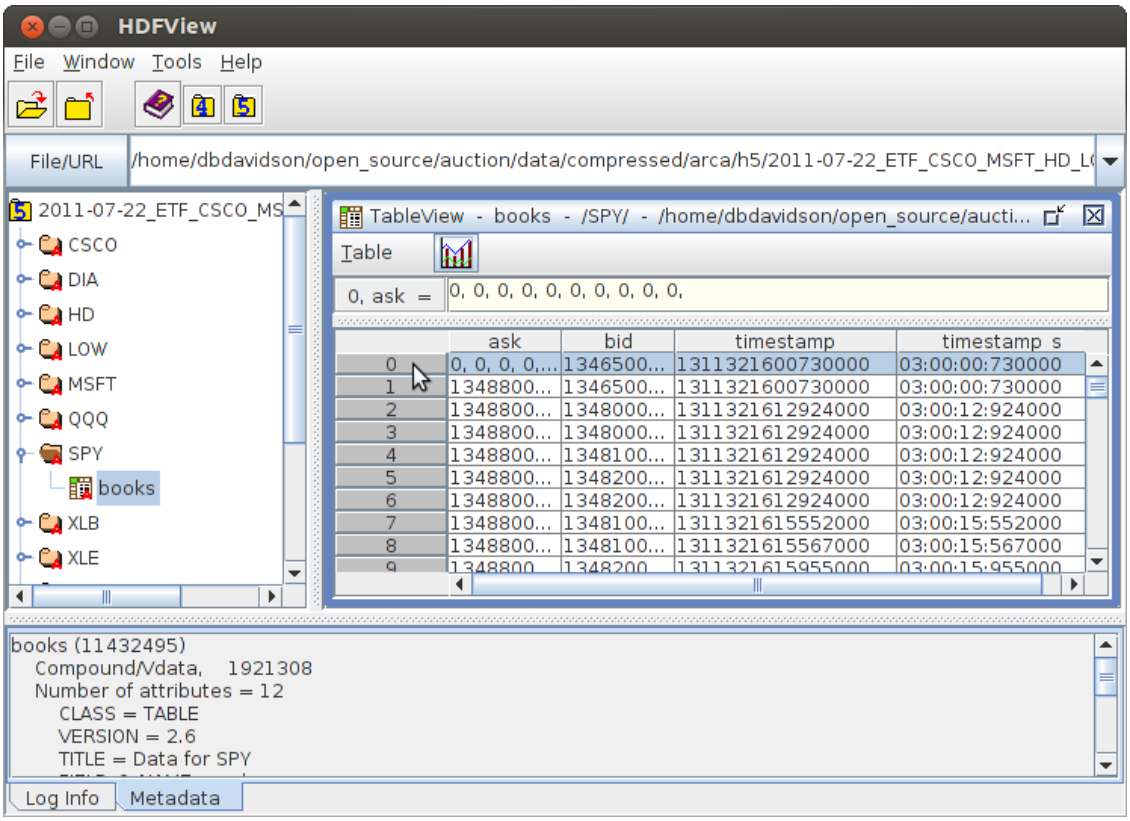

Figure 2: Sample Timestamp

Here you see the `timestamp_s` is a string with value `03:00:00:730000`. This means the first SPY record came in just at 3am Chicago time, or 4am NY time. This corresponds to the first record in the file:

`translate/NYSE_ARCA2/ARCA_SPY.20110722.gz`

produced in the translate phase by Geoff, that starts like:

```
SPY 20110722 04:00:00.730
```

Note it has the same time (only in NY zone).

The timestamps have microsecond resolution and are stored as Int64Col. Therefore, to get the equivalent UNIX time take the large number and divide by 1e6. So, for illustration - from an interactive python shell:

```
>>> 1311321600730000/1e6
1311321600.73
>>> time.ctime(1311321600.73)
'Fri Jul 22 03:00:00 2011'
```

Dividing the timestamp by 1 million gives the seconds since the epoch, which can be used to get back to a string version. This is not convenient for a user, so the string version is also stored. The script `time_utils.py` has methods for dealing with timestamps consistently in this format.

2.2.3 Price/Quantity Data

The price quantity data is of type: Int64Col(shape=(5,2)). That means 5 levels with two values, price and quantity.

All prices are stored with 6 decimal places. That was the largest value I encountered in this file. The tick size for equities we care about is therefore 10,000, which gives the decimal at cents. For example, the SPY price stored as 134880000 means 134.880000 and the next price in the book is 10,000 more (i.e. 134890000 or 134.890000)

In a previous email you mentioned you wanted to store 10 levels of data. Please verify whether this is the case. If so I can try that out and we'll see what happens.

As you would expect, ask prices are increasing and bid prices are decreasing. So `(bid[0], ask[0])` is the top of the book.

2.2.4 Parser Summary Information

If/when the parsing is complete a single summary record is deposited in the file at group `/root/parse_results/`. This group contains two datasets, `summary` with a single summary record and `warnings` with any warnings encountered during the parse of the data. The fields in the `summary` are as follows:

field	meaning
is_valid	1 if the processing was valid. For ARCA, valid means all records were processed correctly and 0 orders remain in the book when done
src_file	The file used to generate the data
src_file_size	The size of the input file
src_file_mtime	The last modified time of the input file
parse_start	The time the parse process started
parse_stop	The time the parse process completed
data_start	The timestamp of the first record in the file
data_stop	The timestamp of the last record in the file
irrelevants	Updates not written to the file because they had no impact on the book. These are the updates in the wings (i.e. levels beyond book), so they are not written to save space. For this first example there were 1742877 irrelevants. Of course the records are tracked to keep an accurate view of the book, they just are not written since they add no new information
total warnings	Number of warnings. The actual content of the warnings are stored in: /root/parse_results/warnings

The fields in the `warnings` are as follows:

field	meaning
msg	The warning message
src_line	The line where the warning was first determined

The purpose of this summary information is to allow a follow-up script to easily interrogate all output files that are generated to see where there are any holes. The code to do that should be a simple script. Currently there are only a handful of warnings that might be encountered. On this data set there were 19 warnings, all for XLE with the similar issue of locked market:

msg	src_line
XLE: Locked :Market XLE:(79080000, 79080000)	76155703

3 Next Steps

- You guys, John, Eric and Geoff review this and the see if you can open the file using hdfview and get a sense of if it works.
- John, as with the book.h5 file, see if you can open this file and read in the data in matlab. This will be a much better test - since it is a big file with loads of data. Also, one change since you tried this on book.h5 is that I added in compression which made a huge difference in space as well as speed. hdfview had no problems and I anticipate matlab will not as well. Let me know.
- Decide if you want 10 levels or just the current of 5.
- Get again from ARCA the data that was bogus.
- If this approach sounds good, we run the script on all compressed ARCA files to create the hdf5 data repository.
- When done, run a script to report on the success of each run as well as any warnings.
- After that move on to CME if desired and then NASDAQ.

Hopefully, Geoff will approve of this format and general approach and if it makes sense take steps to move in that direction. If so, I would be glad to discuss the benefits of Python/hdf5 and help him in any way come up to speed.