# storage_benchmark

September 29, 2021

## 1 Storage Benchmark

In this notebook, we'll compare the following storage formats: - CSV: Comma-separated, standard flat text file format. - HDF5: Hierarchical data format, developed initially at the National Center for Supercomputing Applications. It is a fast and scalable storage format for numerical data, available in pandas using the PyTables library. - Parquet: Part of the Apache Hadoop ecosystem, a binary, columnar storage format that provides efficient data compression and encoding and has been developed by Cloudera and Twitter. It is available for pandas through the `pyarrow` library, led by Wes McKinney, the original author of pandas.

This notebook compares the performance of the preceding libraries using a test DataFrame that can be configured to contain numerical or text data, or both. For the HDF5 library, we test both the fixed and table formats. The table format allows for queries and can be appended to.

### 1.1 Usage

To recreate the charts used in the book, you need to run this notebook twice up to section 'Store Result' using different settings for `data_type` and arguments for `generate_test_data` as follows: 1. `data_type='Numeric`: `numerical_cols=2000`, `text_cols=0` (default) 2. `data_type='Mixed`: `numerical_cols=1000`, `text_cols=1000`

### 1.2 Imports & Settings

[1]:
```
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
from pathlib import Path
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import random
import string
```

[3]:
```
sns.set_style('whitegrid')
```

[4]:
```
results = {}
```

### 1.3 Generate Test Data

The test `DataFrame` that can be configured to contain numerical or text data, or both. For the HDF5 library, we test both the fixed and table format.

[5]:
```
def generate_test_data(nrows=100000, numerical_cols=2000, text_cols=0,␣
↪text_length=10):
    s = "".join([random.choice(string.ascii_letters)
                 for _ in range(text_length)])
    data = pd.concat([pd.DataFrame(np.random.random(size=(nrows,␣
↪numerical_cols))),
                      pd.DataFrame(np.full(shape=(nrows, text_cols),␣
↪fill_value=s))],
                     axis=1, ignore_index=True)
    data.columns = [str(i) for i in data.columns]
    return data
```

[6]:
```
data_type = 'Numeric'
```

[7]:
```
df = generate_test_data(numerical_cols=1000, text_cols=1000)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Columns: 2000 entries, 0 to 1999
dtypes: float64(1000), object(1000)
memory usage: 1.5+ GB
```

### 1.4 Parquet

#### 1.4.1 Size

[8]:
```
parquet_file = Path('test.parquet')
```

[9]:
```
df.to_parquet(parquet_file)
size = parquet_file.stat().st_size
```

#### 1.4.2 Read

[10]:
```
%%timeit -o
df = pd.read_parquet(parquet_file)
```

```
4.86 s ± 134 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

[10]:
```
<TimeitResult : 4.86 s ± 134 ms per loop (mean ± std. dev. of 7 runs, 1 loop
each)>
```

[11]:
```
read = _
```

[12]:
```
parquet_file.unlink()
```

#### 1.4.3 Write

[13]:
```
%%timeit -o
df.to_parquet(parquet_file)
parquet_file.unlink()
```

```
43.5 s ± 1.13 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

[13]:
```
<TimeitResult : 43.5 s ± 1.13 s per loop (mean ± std. dev. of 7 runs, 1 loop
each)>
```

[14]:
```
write = _
```

#### 1.4.4 Results

[15]:
```
results['Parquet'] = {'read': np.mean(read.all_runs), 'write': np.mean(write.
↪all_runs), 'size': size}
```

### 1.5 HDF5

[16]:
```
test_store = Path('index.h5')
```

#### 1.5.1 Fixed Format

**Size**

[17]:
```
with pd.HDFStore(test_store) as store:
    store.put('file', df)
size = test_store.stat().st_size
```

**Read**

[18]:
```
%%timeit -o
with pd.HDFStore(test_store) as store:
    store.get('file')
```

```
2min 7s ± 2.73 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

[18]:
```
<TimeitResult : 2min 7s ± 2.73 s per loop (mean ± std. dev. of 7 runs, 1 loop
each)>
```

[19]:
```
read = _
```

[20]:
```
test_store.unlink()
```

**Write**

[21]:
```
%%timeit -o
with pd.HDFStore(test_store) as store:
    store.put('file', df)
test_store.unlink()
```

```
1min 10s ± 1.47 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

[21]:
```
<TimeitResult : 1min 10s ± 1.47 s per loop (mean ± std. dev. of 7 runs, 1 loop
each)>
```

[22]:
```
write = _
```

**Results**

[23]:
```
results['HDF Fixed'] = {'read': np.mean(read.all_runs), 'write': np.mean(write.
↪all_runs), 'size': size}
```

### 1.5.2 Table Format

**Size**

[24]:
```
with pd.HDFStore(test_store) as store:
    store.append('file', df, format='t')
size = test_store.stat().st_size
```

**Read**

[ ]:
```
%%timeit -o
with pd.HDFStore(test_store) as store:
    df = store.get('file')
```

[ ]:
```
read = _
```

[ ]:
```
test_store.unlink()
```

**Write** Note that `write` in table format does not work with text data.

[ ]:
```
%%timeit -o
with pd.HDFStore(test_store) as store:
    store.append('file', df, format='t')
test_store.unlink()
```

[ ]:
```
write = _
```

**Results**

[ ]:
```
results['HDF Table'] = {'read': np.mean(read.all_runs), 'write': np.mean(write.
↪all_runs), 'size': size}
```

### 1.5.3 Table Select

**Size**

```
[ ]: with pd.HDFStore(test_store) as store:
         store.append('file', df, format='t', data_columns=['company', 'form'])
     size = test_store.stat().st_size
```

**Read**

```
[ ]: company = 'APPLE INC'
```

```
[ ]: %%timeit
     with pd.HDFStore(test_store) as store:
         s = store.get('file')
```

```
[ ]: read = _
```

```
[ ]: test_store.unlink()
```

**Write**

```
[ ]: %%timeit
     with pd.HDFStore(test_store) as store:
         store.append('file', df, format='t', data_columns=['company', 'form'])
     test_store.unlink()
```

```
[ ]: write = _
```

**Results**

```
[ ]: results['HDF Select'] = {'read': np.mean(read.all_runs), 'write': np.mean(write.
      ↪all_runs), 'size': size}
```

## 1.6 CSV

```
[ ]: test_csv = Path('test.csv')
```

### 1.6.1 Size

```
[ ]: df.to_csv(test_csv)
     test_csv.stat().st_size
```

### 1.6.2 Read

```
[ ]: %%timeit -o
     df = pd.read_csv(test_csv)
```

```
[ ]: read = _
```

[ ]:
```
test_csv.unlink()
```

#### 1.6.3 Write

[ ]:
```
%%timeit -o
df.to_csv(test_csv)
test_csv.unlink()
```

[ ]:
```
write = _
```

#### 1.6.4 Results

[ ]:
```
results['CSV'] = {'read': np.mean(read.all_runs), 'write': np.mean(write.
 ↪all_runs), 'size': size}
```

### 1.7 Store Results

[ ]:
```
pd.DataFrame(results).assign(Data=data_type).to_csv(f'{data_type}.csv')
```

### 1.8 Display Results

Please run the notebook twice as described above under `Usage` to create the two `csv` files with results for different test data.

[ ]:
```
df = (pd.read_csv('Numeric.csv', index_col=0)
      .append(pd.read_csv('Mixed.csv', index_col=0))
      .rename(columns=str.capitalize))
df.index.name='Storage'
df = df.set_index('Data', append=True).unstack()
df.Size /= 1e9
```

[ ]:
```
fig, axes = plt.subplots(ncols=3, figsize=(16, 4))
for i, op in enumerate(['Read', 'Write', 'Size']):
    flag= op in ['Read', 'Write']
    df.loc[:, op].plot.barh(title=op, ax=axes[i], logx=flag)
    if flag:
        axes[i].set_xlabel('seconds (log scale)')
    else:
        axes[i].set_xlabel('GB')
fig.tight_layout()
fig.savefig('storage', dpi=300);
```