Blaze
Foundations for Array Computing in Python

Mark Wiebe, Matthew Rocklin
Continuum Analytics

introduction

Motivation

The NumPy NDArray and Pandas DataFrame are *foundational data structures*.

They support the ecosystem

Motivation

But they are restricted to memory.

This is ok for 95% of cases,

what about the other 5%?

Computational Projects

- Many excellent streaming, out-of-core, or distributed alternatives exist
- **NumPy-like**
  - DistArray
  - SciDB
  - Elemental
  - PETSc, Trillinos
  - Biggus
  - ...
- Each approach is valid in a particular situation

Computational Projects

- Many excellent streaming, out-of-core, and distributed alternatives exist
- **Pandas-like**
  - PyTables
  - SQLAlchemy (Postgres, SQLite, MySQL, ...)
  - The HDFS world
    - Hadoop (Pig, Hive, ...)
    - Spark
    - Impala
- Each approach is valid in a particular situation

Data Projects

- Analogous collection exists in data storage techniques
  - CSV - Accessible
  - JSON - Pervasive, human readable
  - HDF5 - Efficient access
  - BLZ - Efficient columnar access
  - Parquet - Efficient columnar access (HDFS)
  - PyTables HDF5 - HDF5 + indices
  - HDFS - Big!
  - SQL - SQL!
  - ...
- Each approach is valid in a particular situation

Spinning up a new technology is expensive

Keeping up with the changing landscape frustrates data scientists

Foundations built to address these challenges must be adaptable

What is Blaze?

- Blaze abstracts array and tabular computation
  - Blaze expressions abstract compute systems
  - Blaze data descriptors abstract data storage
  - Datashape abstracts data-type systems
- These abstractions enable interactions

abstract computation

Abstract Computation

- Symbolic table expressions

```
>>> accounts = TableSymbol('accounts', '{id: int, name: string, balance: int}')

>>> deadbeats = accounts[accounts['balance'] < 0]['name']
>>> deadbeats
accounts[accounts['balance'] < 0]['name']
```

- Computations on Python data types

```
>>> L = [(1, 'Alice', 100),
         (2, 'Bob', -200),
         (3, 'Charlie', 300),
         (4, 'Denis', 400),
         (5, 'Edith', -500)]

>>> compute(deadbeats, L)
['Bob', 'Edith']
```

Abstract Computation

- Symbolic table expressions

```
>>> accounts = TableSymbol('accounts', '{id: int, name: string, balance: int}')

>>> deadbeats = accounts[accounts['balance'] < 0]['name']
>>> deadbeats
accounts[accounts['balance'] < 0]['name']
```

- Computations on Pandas DataFrames

```
>>> df = DataFrame([(1, 'Alice', 100),
                    (2, 'Bob', -200),
                    (3, 'Charlie', 300),
                    (4, 'Denis', 400),
                    (5, 'Edith', -500)],
                    columns=['id', 'name', 'balance'])

>>> compute(deadbeats, df)
1      Bob
4    Edith
Name: name, dtype: object
```

CONTINUUM ANALYTICS

Notebook Demo

uniform data

Data Projects

- Analogous collection exists in data storage techniques
  - CSV - Accessible
  - JSON - Pervasive, human readable
  - HDF5 - Efficient access
  - BLZ - Efficient columnar access
  - Parquet - Efficient columnar access (HDFS)
  - PyTables HDF5 - HDF5 + indices
  - HDFS - Big!
  - SQL - SQL!
  - ...
- Each approach is valid in a particular situation

Spinning up a new technology is expensive

Keeping up with the changing landscape frustrates data scientists

CSV

```
$ cat accounts.csv
id, name, balance
1, Alice, 100
2, Bob, -200
3, Charlie, 300
4, Denis, 400
5, Edith, -500
```

```
>>> csv = CSV('accounts.csv')
>>> csv.columns
['id', 'name', 'balance']

>>> csv.py[:3, ['name', 'balance']]
[('Alice', 100), ('Bob', -200), ('Charlie', 300)]
```

HDF5

```
$ h5dump -H accounts.hdf5
HDF5 "accounts.hdf5" {
GROUP "/" {
   DATASET "accounts" {
      DATATYPE  H5T_COMPOUND {
         H5T_STD_I64LE "id";
         H5T_STRING {
            STRSIZE H5T_VARIABLE;
            STRPAD H5T_STR_NULLTERM;
...
```

```
>>> hdf5 = HDF5('accounts.hdf5', '/accounts')
>>> hdf5.columns
['id', 'name', 'balance']

>>> hdf5.py[:3, ['name', 'balance']]
[('Alice', 100), ('Bob', -200), ('Charlie', 300)]
```

SQL

```
>>> sql = SQL('postgresql://user:pass@hostname/', 'accounts')
>>> sql.columns
['id', 'name', 'balance']

>>> sql.py[:3, ['name', 'balance']]
[('Alice', 100), ('Bob', -200), ('Charlie', 300)]
```

Data API

- Data Descriptors support native Python access
  - Iteration: `iter(csv)`
  - Extension: `csv.extend(...)`
  - Item access: `csv.py[:, ['name', 'balance']]`
- Data Descriptors support chunked access
  - Iteration: `csv.chunks()`
  - Extension: `csv.extend_chunks(...)`
  - Item access: `csv.dynd[:, ['name', 'balance']]`

Data API

- Data Descriptors support native Python access
  - Iteration: `iter(sql)`
  - Extension: `sql.extend(...)`
  - Item access: `sql.py[:, ['name', 'balance']]`
- Data Descriptors support chunked access
  - Iteration: `sql.chunks()`
  - Extension: `sql.extend_chunks(...)`
  - Item access: `sql.dynd[:, ['name', 'balance']]`

Uniformity Facilitates User-Data Interaction

```
>>> csv.py[:3, ['name', 'balance']]
[('Alice', 100), ('Bob', -200), ('Charlie', 300)]

>>> json.py[:3, ['name', 'balance']]
[('Alice', 100), ('Bob', -200), ('Charlie', 300)]

>>> hdf5.py[:3, ['name', 'balance']]
[('Alice', 100), ('Bob', -200), ('Charlie', 300)]

>>> sql.py[:3, ['name', 'balance']]
[('Alice', 100), ('Bob', -200), ('Charlie', 300)]
```

CONTINUUM ANALYTICS

Uniformity Facilitates Data-Data Interaction

- CSV to SQL

```
>>> sql.extend(iter(csv))
```

- SQL to HDF5

```
>>> hdf5.extend_chunks(sql.chunks())
```

`glue`

Boundary Conditions

- There are boundaries between different compute and data backends
  - Naming conventions differ
  - Storage differs
- Need glue to help connect them
  - Uniform way to write the types
  - Efficient intermediate storage and transformation

Datashape

- Datashape provides an array type syntax
- Datashape can map to many backend type systems
  - Python Dynamic Types
  - NumPy DTypes
  - SQL Table Types
  - HDF5

Datashapes

- Scalars
  - `bool`
  - `int`
  - `real`
  - `complex`
- Arrays
  - `100 * 50 * real`
- Tables
  - `var * {name: string, height: real, birthday: date}`

LibDyND

- Provides data glue
  - Uses datashape type system
  - Understands many standard binary/text formats
- Array-oriented storage and compute
  - Similar to NumPy, but more general

Notebook Demo

Conclusion

- Abstractions facilitate interaction
- Blaze connects data scientists to a broader ecosystem
- Try it and get involved

```
$ conda install -c mrocklin -c mwiebe blaze
```

blaze-dev@continuum.io