Data Storage and Handling

Data Storage and Handling

A little excursion on how to store, organize and handle large data sets for analytics …

Data Storage and Handling

A little excursion on how to store, organize and handle large data sets for analytics …

So far we have used:

- ***CSV*** files: text representation of tables
- ***NumPy*** files: binary arrays

Outline

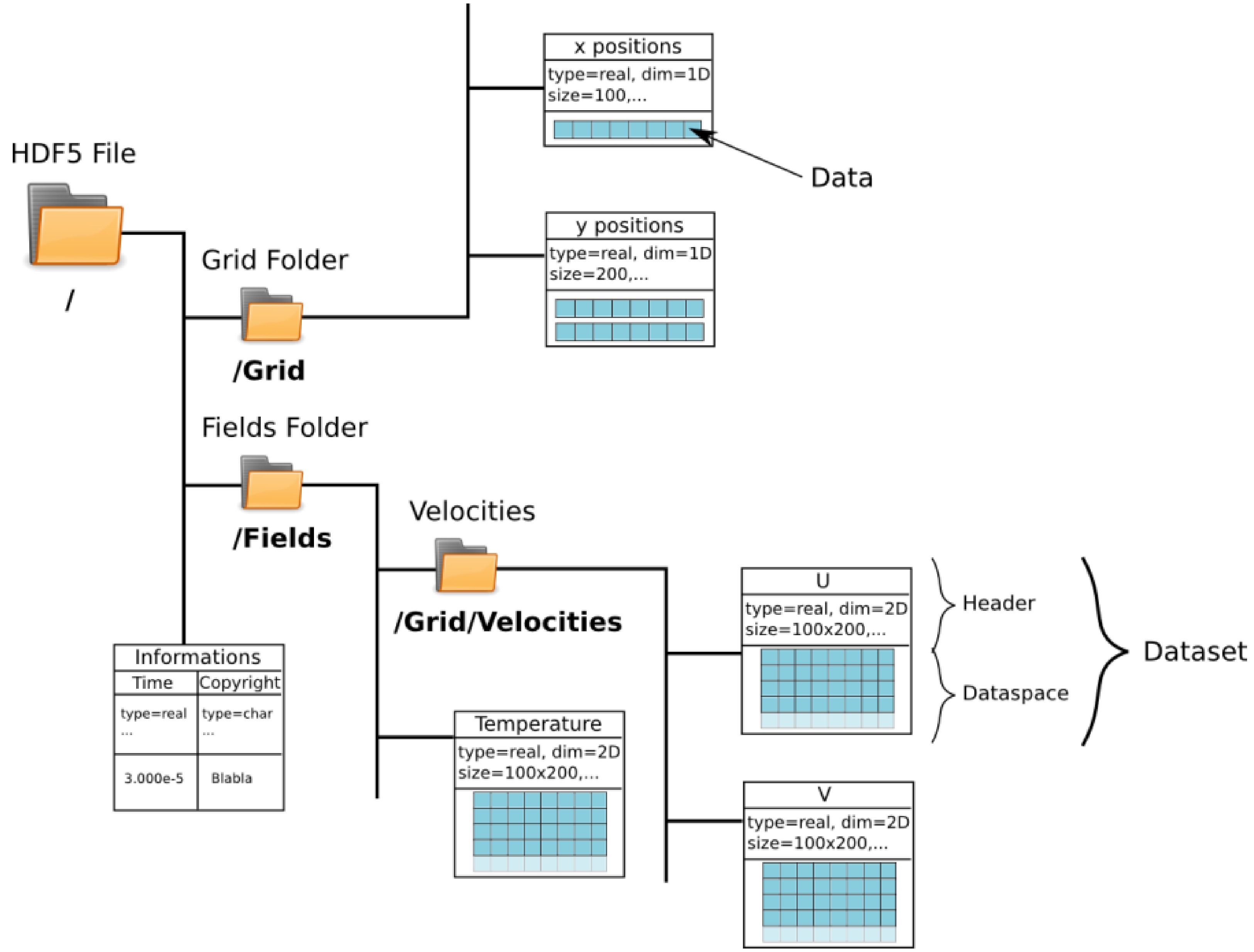
- HDF5
- XML
- JSON
- Relational Data Bases
- NoSQL Data Bases
- Use Case: Restaurant Rating Site

The HDF5 Data Container Format

The HDF5 Data Container Format

Hierarchical Data Format (HDF) is a set of file formats (HDF4, HDF5) designed to store and organize large amounts of data with APIs for many programming languages.

HDF5 Structure



[Image Source: https://www.sphenisc.com/doku.php/software/development/hdf5-phdf5]

HDF5 Key Features:

- POSIX-like syntax for internal data structures /path/to/resource
  - folders
  - meta data
  - comments (even code)
  - arrays
- fast n-D data access
- data compression
- APIs for many programming languages

In Python:

- ***h5py***: http://docs.h5py.org/en/stable/index.html
- ***HDF5 Docs:*** https://portal.hdfgroup.org/display/support

In [2]:
```
import h5py
import numpy as np
d1 = np.random.random(size = (1000,20))
d2 = np.random.random(size = (1000,200))
```

In [3]:
```
#create h5 file
hf = h5py.File('data.h5', 'w')
#write data
hf.create_dataset('dataset_1', data=d1)
hf.create_dataset('dataset_2', data=d2)
hf.close()
```

In [4]:
```python
#read data
hf = h5py.File('data.h5', 'r')
#get dateset
n1 = hf.get('dataset_1')
n1
```

Out[4]:
```
<HDF5 dataset "dataset_1": shape (1000, 20), type "<f8">
```

In [4]:
```python
#read data
hf = h5py.File('data.h5', 'r')
#get dateset
n1 = hf.get('dataset_1')
n1
```

Out[4]:
```
<HDF5 dataset "dataset_1": shape (1000, 20), type "<f8">
```

In [5]:
```python
#convert to NumPy
np.array(n1)
```

Out[5]:
```
array([[0.85419862, 0.99621707, 0.9402049 , ..., 0.63922664, 0.59930167,
        0.74569012],
       [0.61698102, 0.7237691 , 0.71565896, ..., 0.00635492, 0.07563886,
        0.39948412],
       [0.60144952, 0.33529435, 0.7516772 , ..., 0.85664484, 0.46636001,
        0.15232542],
       ...,
       [0.05503757, 0.13383227, 0.43754545, ..., 0.64369297, 0.20736103,
        0.73253284],
       [0.53766212, 0.97982173, 0.79268274, ..., 0.12801816, 0.2375696 ,
        0.82384853],
       [0.15663464, 0.81804727, 0.46839317, ..., 0.49955137, 0.75115427,
        0.10348987]])
```

more on HDF5 in the lab session...

XML

<xml />

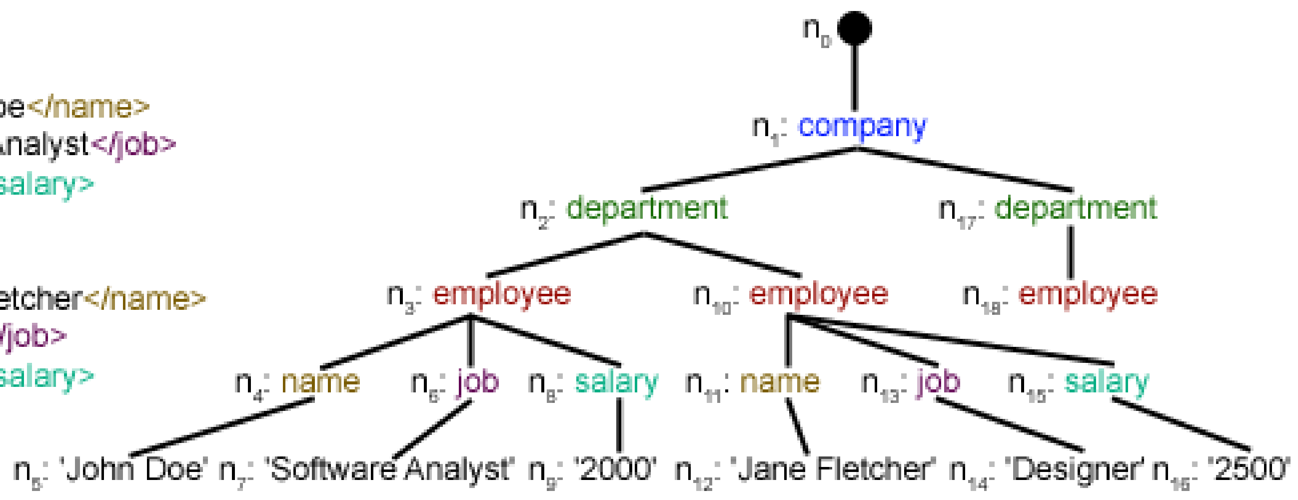
XML

<xml />

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. The design goals of XML emphasize simplicity, generality, and usability across the Internet. It is a textual data format with strong support via Unicode for different human languages. Although the design of XML focuses on documents, the language is widely used for the representation of arbitrary data structures such as those used in web services.

XML Tree Representation of Data

```
<company>
 <department>
  <employee>
   <name>John Doe</name>
   <job>Software Analyst</job>
   <salary>2000</salary>
  </employee>
  <employee>
   <name>Jane Fletcher</name>
   <job>Designer</job>
   <salary>2500</salary>
  </employee>
 </department>
 <department>
  <employee>
  </employee>
 </department>
</company>
```



Another XML Example

```xml
<?xml version="1.0"?>
<data>
    <country name="Liechtenstein">
        <rank>1</rank>
        <year>2008</year>
        <gdppc>141100</gdppc>
        <neighbor name="Austria" direction="E"/>
        <neighbor name="Switzerland" direction="W"/>
    </country>
    <country name="Singapore">
        <rank>4</rank>
```

XML with *Python*

In [6]:
```
import xml.etree.ElementTree as ET
tree = ET.parse(path+'/DATA/example.xml') #parse xml document
root = tree.getroot() #get tree root
```

In [7]:
```python
#get first elements of the tree
for child in root:
    print( child.tag, child.attrib)
```

```
country {'name': 'Liechtenstein'}
country {'name': 'Singapore'}
country {'name': 'Panama'}
```

In [8]:
```
#iterate over the neighbor attribute
for neighbor in root.iter('neighbor'):
    print (neighbor.attrib)
```

```
{'name': 'Austria', 'direction': 'E'}
{'name': 'Switzerland', 'direction': 'W'}
{'name': 'Malaysia', 'direction': 'N'}
{'name': 'Costa Rica', 'direction': 'W'}
{'name': 'Colombia', 'direction': 'E'}
```

In [9]:
```
#get  all country nodes and extract attributes
for country in root.findall('country'):
    rank = country.find('rank').text
    name = country.get('name')
    print (name, rank)
```

```
Liechtenstein 1
Singapore 4
Panama 68
```

```
In [9]: #get  all country nodes and extract attributes
        for country in root.findall('country'):
            rank = country.find('rank').text
            name = country.get('name')
            print (name, rank)

Liechtenstein 1
Singapore 4
Panama 68
```

more on the *Python XML* API: https://docs.python.org/2/library/xml.etree.elementtree.html

JSON

{JSON}
JavaScript Object Notation

JSON

{JSON}
JavaScript Object Notation

JavaScript Object Notation (JSON) is an open-standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types. JSON is a language-independent data format. It was derived from JavaScript, but as of 2017 many programming languages include code to generate and parse JSON-format data.

JSON Document Tree

```
{                                                   Object Starts
    “Title”: “The Cuckoo’s Calling”
    “Author”: “Robert Galbraith”,
    “Genre”: “classic crime novel”,
    “Detail”: {                                     Object Starts
        “Publisher”: “Little Brown”         Value string
        “Publication_Year”: 2013,                   Value number
        “ISBN-13”: 9781408704004,
        “Language”: “English”,
        “Pages”: 494
    }                                       Object ends
    “Price”: [                                      Array starts
        {                                           Object Starts
            “type”: “Hardcover”,
            “price”: 16.65,
        }                                           Object ends
        {                                           Object Starts
            “type”: “Kindle Edition”,
            “price”: 7.03,
        }                                           Object ends
    ]                                       Array ends
}                                                   Object ends
```

JSON in Pythhon

In [10]:

```
import pandas as pd

Data = {'Product': ['Desktop Computer','Tablet','iPhone','Laptop'],
        'Price': [700,250,800,1200]
        }

df = pd.DataFrame(Data, columns= ['Product', 'Price'])

print (df)
```

```
            Product  Price
0  Desktop Computer    700
1            Tablet    250
2            iPhone    800
3            Laptop   1200
```

JSON in Pythhon

In [10]:
```
import pandas as pd

Data = {'Product': ['Desktop Computer','Tablet','iPhone','Laptop'],
        'Price': [700,250,800,1200]
        }

df = pd.DataFrame(Data, columns= ['Product', 'Price'])

print (df)
```

```
            Product  Price
0  Desktop Computer    700
1            Tablet    250
2            iPhone    800
3            Laptop   1200
```

In [11]:
```
#native JSON support in pandas
Export = df.to_json ('Export_DataFrame.json')
```

Use *Jupyter Lab* to browse the JSON file.

Relational Databases

Relational Databases

- Data structure: tables
- Relational Algebra

Relational Databases

- Data structure: tables
- Relational Algebra

Structured Query Language : SQL

Structured Query Language is a domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS), or for stream processing in a relational data stream management system (RDSMS).

In a Nutshell: ACID properties of relational databases

- Atomicity
- Consistency
- Isolation
- Durability

SQL in *Pandas*

Get SQL query as ***pandas*** table

```
#NOTE: not running code - no SQL server here...
df = psql.read_sql(('select "Timestamp","Value" from "MyTable" '
                     'where "Timestamp" BETWEEN %(dstart)s AND %(dfinish)s'),
                   db,params={"dstart":datetime(2014,6,24,16,0),"dfinish":datetime(20
4,6,24,17,0)},
                   index_col=['Timestamp'])
```

NoSQL

NoSQL

NoSQL Data Bases: "Not Only SQL"

Requirements driven by Big Data and Analytics...

- Scalability
- Flexibility
- Throughput

Typical Types of NoSQL Data Bases

- Document based Data Bases
- Key-Value Stores
- Column oriented Data Bases
- Graph Data Bases
- ...

Document based Data Bases

- Data stored in documents (files)
- Flexible structure in documents (like XML)
- Queries like in SQL
- Support distributed operations (***MapReduce***)

Key-Value Stores

riak

- Simple Data Tuple: #Key : Value
- Very high throughput
- Very low latency

Column oriented Data Base

Cassandra

- Data in tables
- Column first data access
    - very good performance for many analytic use cases
    - e.g. aggregation operations
- good compression support

Graph based Data Bases

Neo4j

- Data structure: Graphs {vertex,edges}
- Applications: e.g. Social Networks, ...
- Queries like "find friends of friends" ...

Disadvantages of NoSQL

- Relational DBs have a solid theory
  - ACID
  - mathematical relation algebra
    - allows profs over DB queries

Disadvantages of NoSQL

- Relational DBs have a solid theory
  - ACID
  - mathematical relation algebra
    - allows profs over DB queries

- Is this true for NoSQL DBs ?

Disadvantages of NoSQL

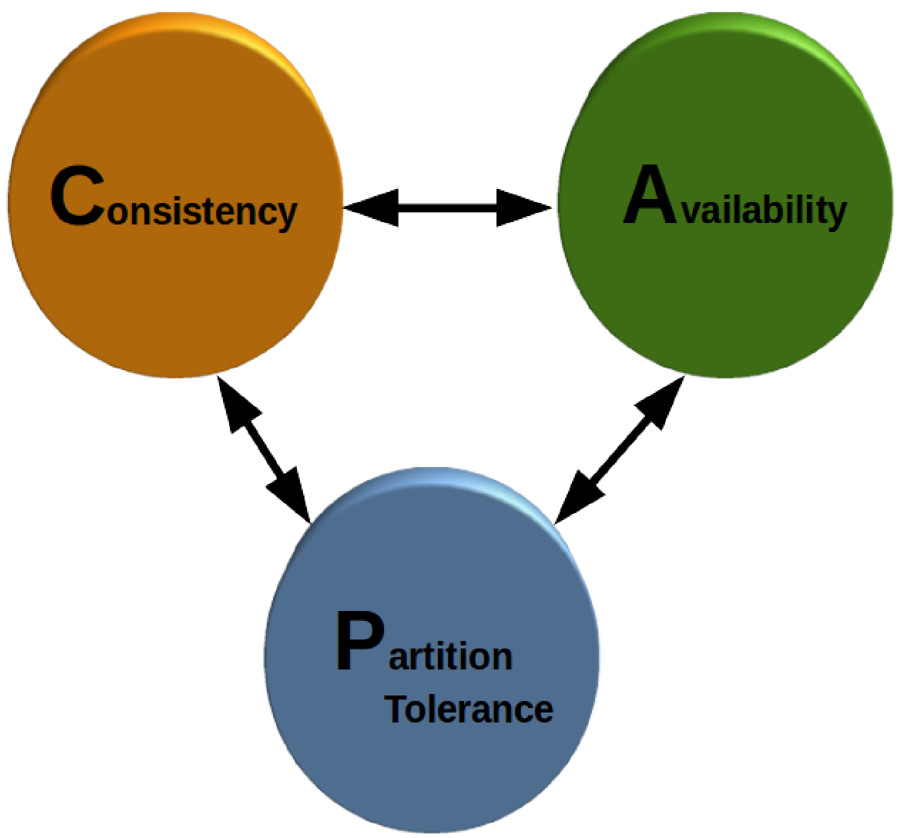
- Relational DBs have a solid theory
  - ACID
  - mathematical relation algebra
    - allows profs over DB queries

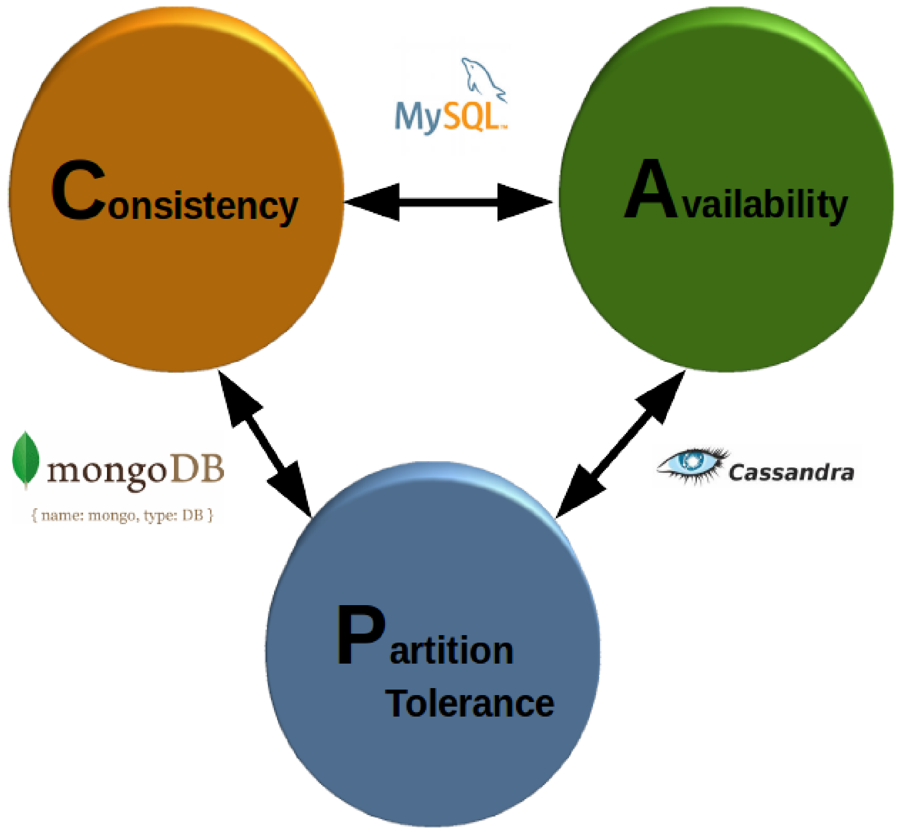
- Is this true for NoSQL DBs ?

Only with constraints !

CAP Theorem

Basic properties for DB systems [Brewer]




Consistency
Availability
Partition Tolerance
MySQL
mongoDB
{ name: mongo, type: DB }
Cassandra

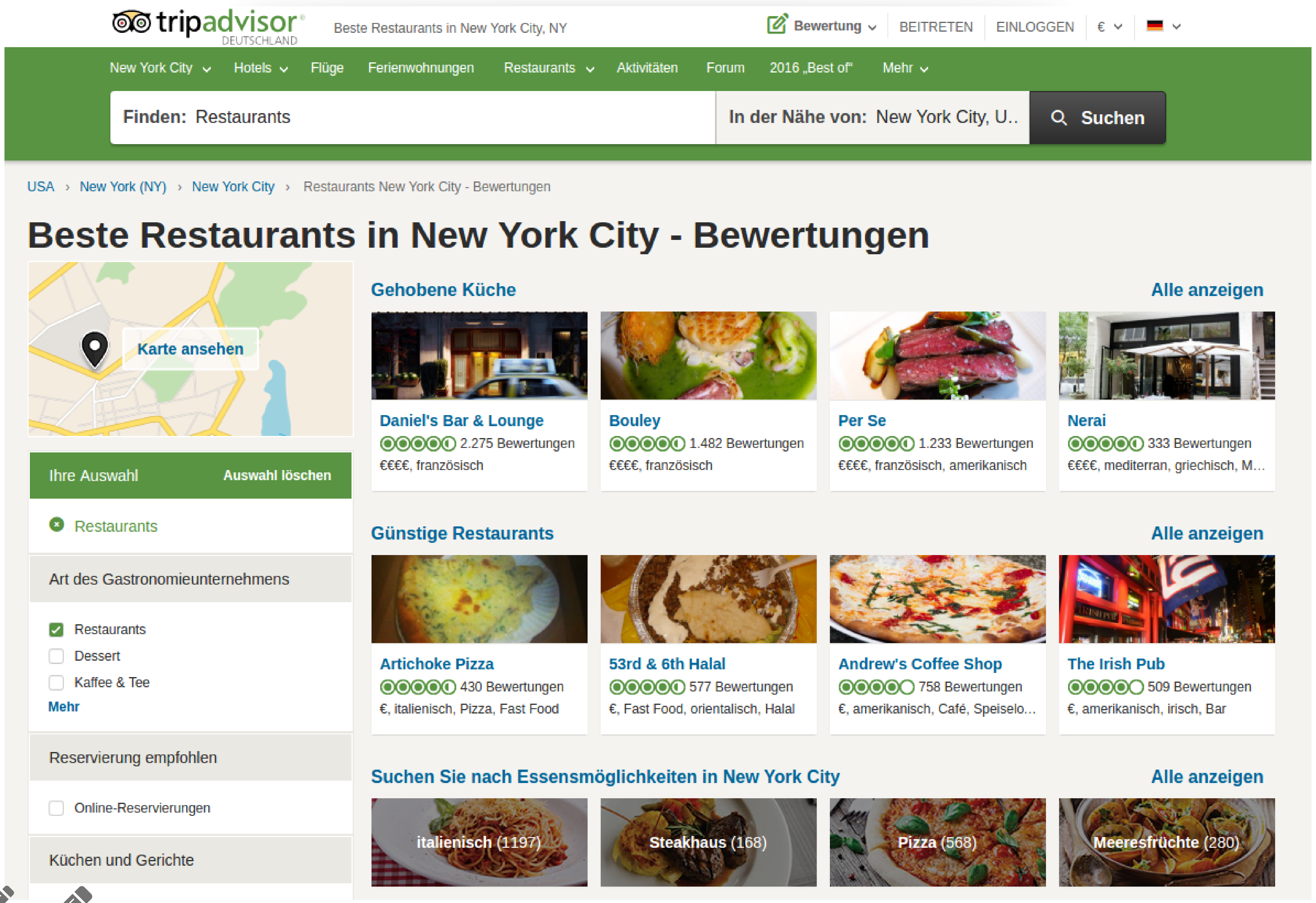
BASE Criteria for (NoSQL) Databases

Basically available, Soft-State, Eventual Consistency

- BASE derived from CAP-Theorem
- Replaces ACID for distributed DBs

Use Case:

A Restaurant rating system:



Implementation with MongoDB

mongoDB
{ name: mongo, type: DB }

- Properties of MongoDB
  - Document oriented DB
    - Structure description in JSON

JSON
JavaScript Object Notation

Implementation with MongoDB

mongoDB
{ name: mongo, type: DB }

- Properties of MongoDB
  - Document oriented DB
    - Structure description in JSON

JSON
JavaScript Object Notation

- Data: open data set with restaurants and ratings:
  - https://raw.githubusercontent.com/mongodb/docs-assets/primer-dataset/primer-dataset.json

Example: JSON Scheme for a restaurant

```
{
  "address": {
     "coord": [ -73.856077, 40.848447 ],
     "street": "Morris Park Ave",
     "zipcode": "10462"
  },
  "borough": "Bronx",
  "cuisine": "Bakery",
  "grades": [
     { "date": { "$date": 1393804800000 }, "score": 2 },
     { "date": { "$date": 1378857600000 }, "score": 6 },
```

Hochschule Offenburg
offenburg.university

Hands on!

In [12]:
```
#NOTE: this will only work if you have a local MongoDB Server running

#import MongoDB client module
from pymongo import MongoClient
import warnings
warnings.filterwarnings('ignore')
#connect to MongoDB on localhost
client = MongoClient()
```

Hands on!

In [12]:
```
#NOTE: this will only work if you have a local MongoDB Server running

#import MongoDB client module
from pymongo import MongoClient
import warnings
warnings.filterwarnings('ignore')
#connect to MongoDB on localhost
client = MongoClient()
```

In [13]:
```
#how many worker nodes are working in th MongoDB Cluster?
client.nodes
```

Out[13]:
```
frozenset({('localhost', 27017)})
```

Hochschule Offenburg
offenburg.university

What Data is on the Cluster?

```
In [14]: #see what databases are available
         client.database_names()

Out[14]: ['admin', 'config', 'demo', 'local', 'mydb', 'test']
```

What Data is on the Cluster?

In [14]:
```
#see what databases are available
client.database_names()
```

Out[14]:
```
['admin', 'config', 'demo', 'local', 'mydb', 'test']
```

In [15]:
```
#generate reference to "demo" database
db = client.demo
```

What Data is on the Cluster?

```
In [14]: #see what databases are available
         client.database_names()

Out[14]: ['admin', 'config', 'demo', 'local', 'mydb', 'test']

In [15]: #generate reference to "demo" database
         db = client.demo

In [16]: #list all collections
         db.collection_names()

Out[16]: ['restaurants', 'myresults']
```

MongoDB Queries

MongoDB Queries

In [17]:
```
db.restaurants.find().count()
```

Out[17]:
```
25359
```

In [18]:
```python
db.restaurants.find()[129]
```

Out[18]:
```
{'_id': ObjectId('5cddbe4287ea9d7fab05db9c'),
 'address': {'building': '26',
  'coord': [-73.9983, 40.715051],
  'street': 'Pell Street',
  'zipcode': '10013'},
 'borough': 'Manhattan',
 'cuisine': 'Café/Coffee/Tea',
 'grades': [{'date': datetime.datetime(2014, 7, 10, 0, 0),
   'grade': 'A',
   'score': 10},
  {'date': datetime.datetime(2013, 7, 12, 0, 0), 'grade': 'A', 'score': 10},
  {'date': datetime.datetime(2013, 2, 11, 0, 0), 'grade': 'A', 'score': 9},
  {'date': datetime.datetime(2013, 1, 10, 0, 0), 'grade': 'P', 'score': 4},
  {'date': datetime.datetime(2012, 7, 27, 0, 0), 'grade': 'A', 'score': 12},
  {'date': datetime.datetime(2012, 2, 27, 0, 0), 'grade': 'A', 'score': 11},
  {'date': datetime.datetime(2011, 8, 12, 0, 0), 'grade': 'B', 'score': 24}],
 'name': 'Mee Sum Coffee Shop',
 'restaurant_id': '40365904'}
```

Structured Queries

* number of restaurants in the city

In [19]:
```
db.restaurants.find({"borough": "Manhattan"}).count()
```
Out[19]:
```
10259
```

Structured Queries

* number of restaurants in the city

```
In [19]: db.restaurants.find({"borough": "Manhattan"}).count()
Out[19]: 10259
```

- All entries with Score>10 and ZIP code 10075

```
In [20]: db.restaurants.find({"grades.score": {"$gt": 10}, "address.zipcode": "10075"}).count()
Out[20]: 79
```

Iterators

- e.g. all iterators in ZIP code 10075

In [21]:
```
cursor=db.restaurants.find({"cuisine": "Bakery","address.zipcode": "10075"})
for doc in cursor:
    print (doc["name"])
```

```
Annelies Pastries
Lady M Confections
Butterfield Express
The Belgian Cupcake
```

Hochschule Offenburg
offenburg.university

Map-Reduce with MongoDB

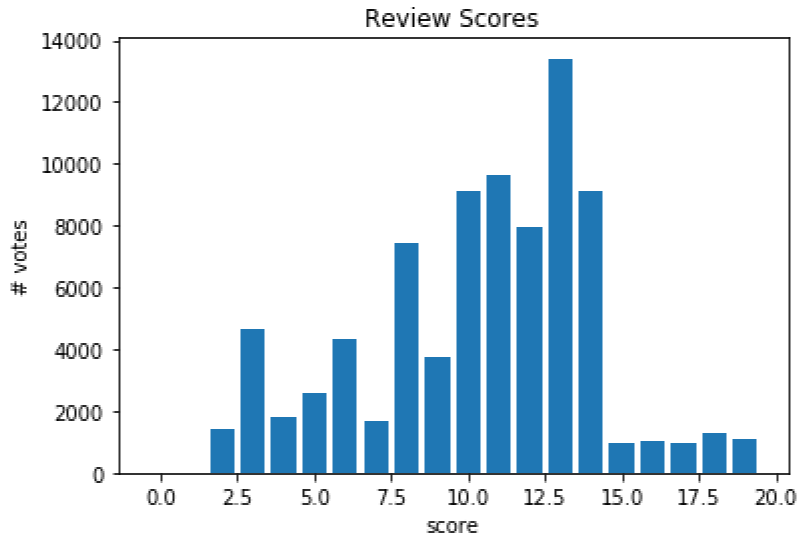
Compute histogram of review scores

In [22]:
```
from bson.code import Code
#map function
map = Code("function () {"
            "  this.grades.forEach(function(z) {"
            "    emit(z.score, 1);"
            "  });"
            "}")

#reduce function
reduce = Code("function (key, values) {"
              "  var total = 0;"
              "  for (var i = 0; i < values.length; i++) {"
              "    total += values[i];"
              "  }"
              "  return total;"
              "}")

result = db.restaurants.map_reduce(map, reduce, "myresults")
```

In [24]:
```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [25]:
```
plt.figure()
df=pd.DataFrame(list(result.find()))
plt.bar(np.arange(20),df[0:20].value )
plt.xlabel('score')
plt.ylabel('# votes')
plt.title('Review Scores')
```

Out[25]: Text(0.5, 1.0, 'Review Scores')



Hochschule Offenburg
offenburg.university

Discussion

```
In [ ]:
```