Distributed Spatial Database

Dr. Junjun Yin

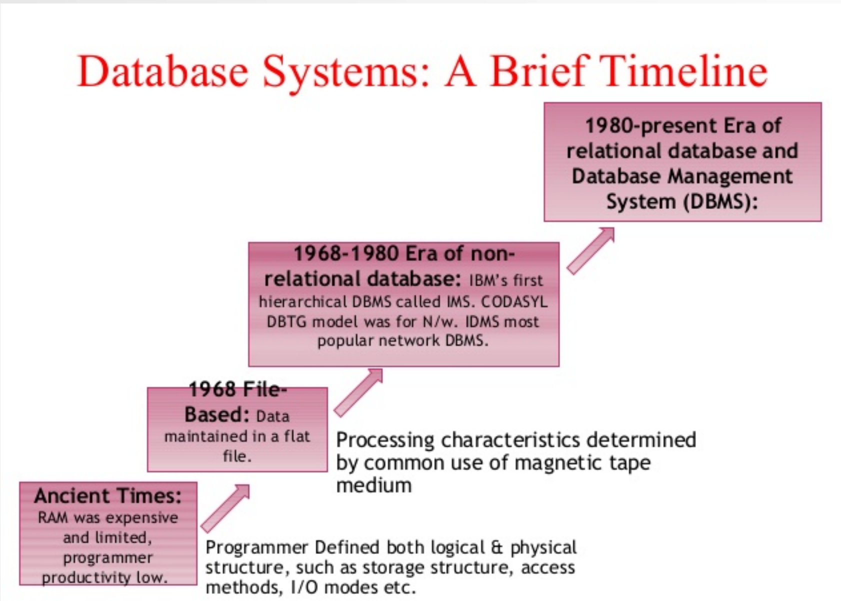
CyberGIS Center for Advanced Digital and Spatial Studies
Department of Geography and Geographic Information Science
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign, IL, 61801, USA
jyn@illinois.edu

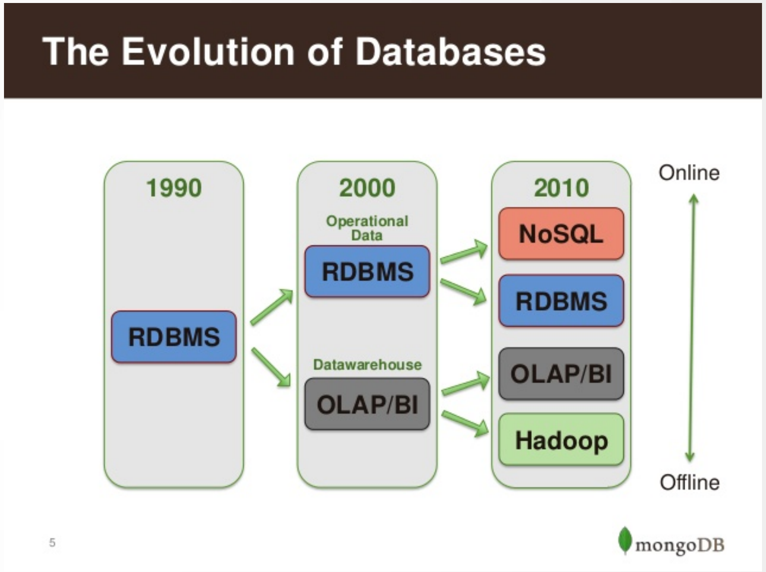
February 9, 2016

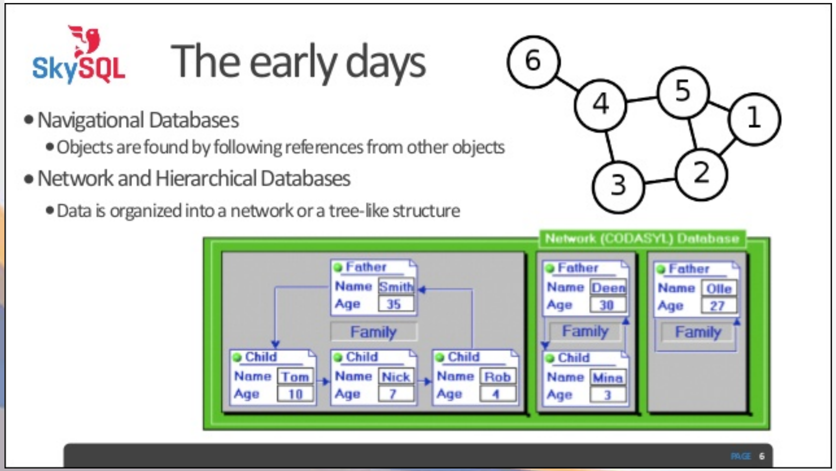
Outline

- What is “distributed databases”
  - File system vs. database
- Relational database vs NoSQL database
  - Relational Database Management System (RDBMS)
  - Spatial Database Management System (SDBMS)
  - Non-relational database (NoSQL)
- Big Data requirements
  - Variety and data structure
  - Velocity and concurrency
- Data management, storage and query performance

Evolution of Databases

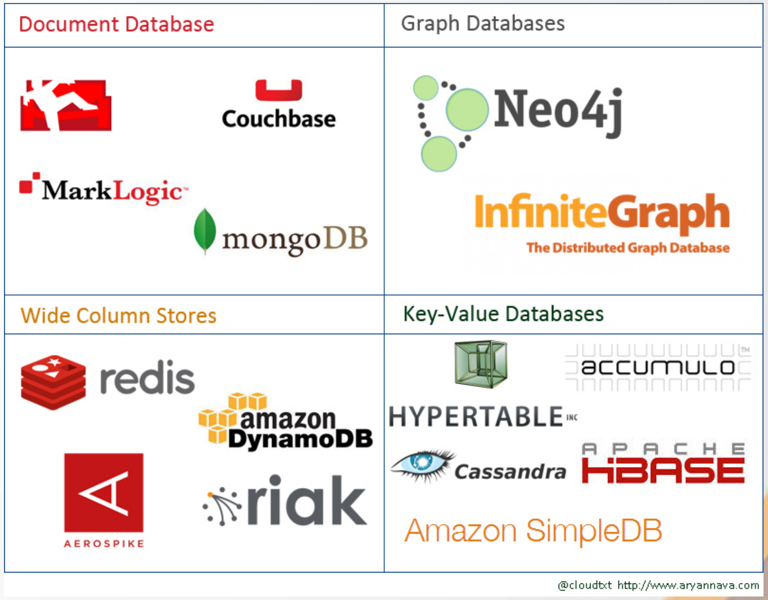






Source:
http://www.slideshare.net/wlaforest/why-nosql-and-mongodb-for-big-data

Evolution of Databases



Source:http://aryannava.com/category/nosql/

Document-oriented Database

```
{
    "_id" : ObjectId("5660d0d92a5d502680eb616f"),
    "loc" : {
        "type" : "Point",
        "coordinates" : [
            -87.9609112,
            42.1526175
        ]
    },
    "message" : "3 more!!!\n",
    "user_id" : "232659889",
    "time" : "Fri Jul 04 20:55:28 CDT 2014"
}
```

- JSON (JavaScript Object Notation) based data format

Column-oriented Database

Row Oriented Database

date	price	size
2011-01-20	10.1	10
2011-01-21	10.3	20
2011-01-22	10.5	40
2011-01-23	10.4	5
2011-01-24	11.2	55
2011-01-25	11.4	66
...	...	...
2013-03-31	17.3	100

Table of Data

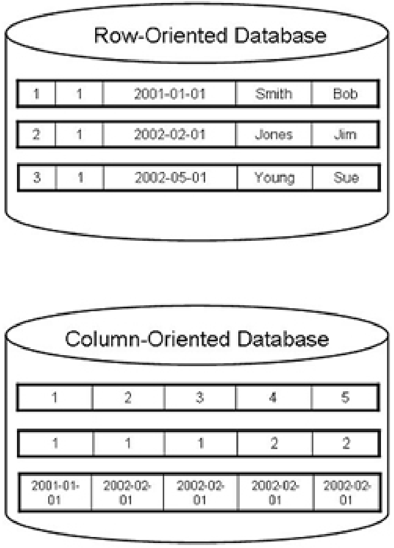
date	price	size
2011-01-20	10.1	10
2011-01-21	10.3	20
2011-01-22	10.5	40
2011-01-23	10.4	5
2011-01-24	11.2	55
2011-01-25	11.4	66
...	...	...
2013-03-31	17.3	100

Column Oriented Database

date	price	size
2011-01-20	10.1	10
2011-01-21	10.3	20
2011-01-22	10.5	40
2011-01-23	10.4	5
2011-01-24	11.2	55
2011-01-25	11.4	66
...	...	...
2013-03-31	17.3	100

Source: http://www.timestored.com/time-series-data/what-is-a-column-oriented-database

Emp_no	Dept_id	Hire_date	Emp_ln	Emp_fn
1	1	2001-01-01	Smith	Bob
2	1	2002-02-01	Jones	Jim
3	1	2002-05-01	Young	Sue
4	2	2003-02-01	Stemle	Bill
5	2	1999-06-15	Aurora	Jack
6	3	2000-08-15	Jung	Laura



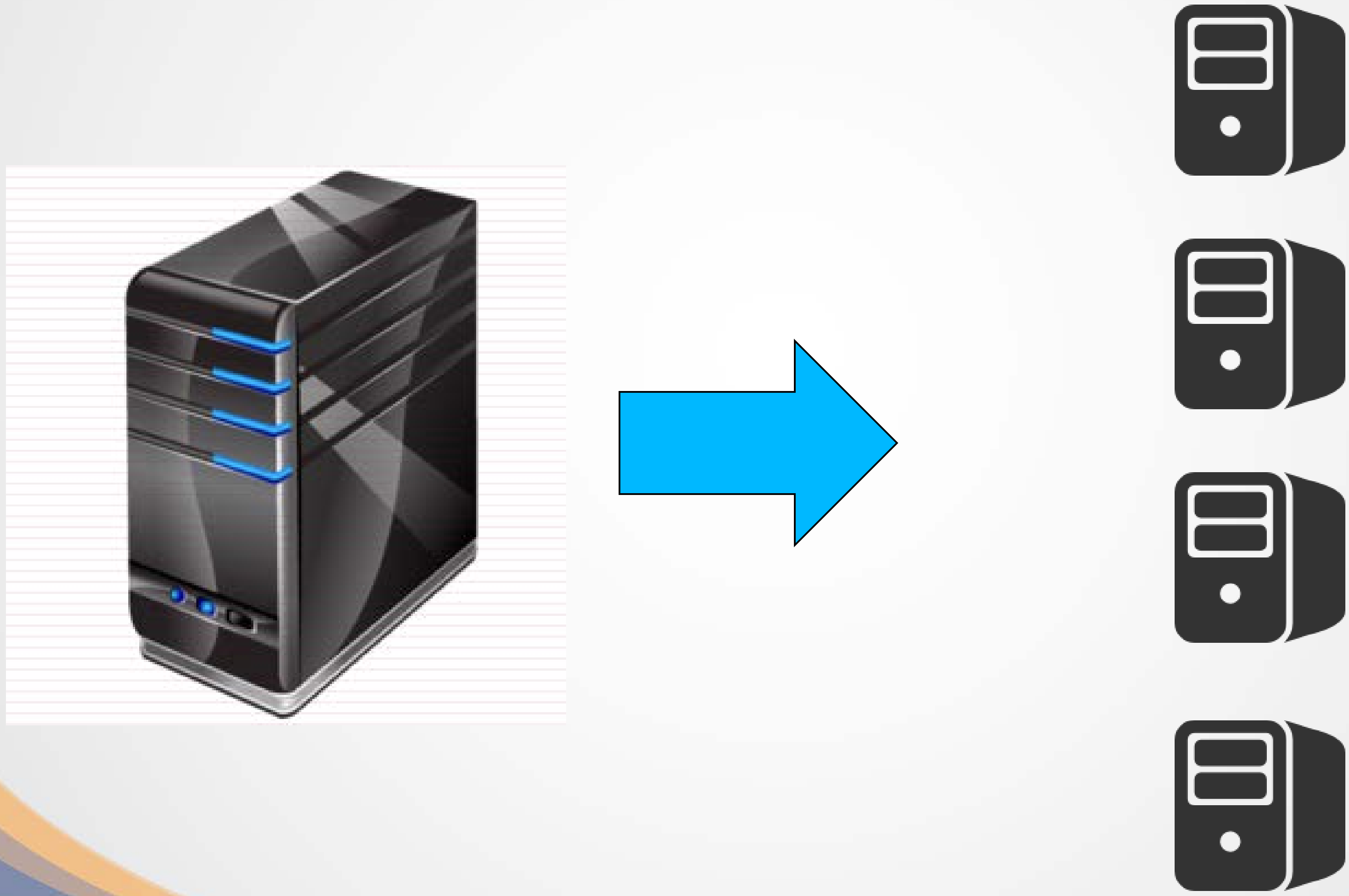
Relational vs. NoSQL

- Relational databases
    - Well designed data schemes
    - Well organized data structures
    - Row-based
- NoSQL databases
    - Document-oriented
    - Column-oriented
    - <key, value> pair-based
    - Graph-based

Spatial Database

- Geometry
  - Point, polyline, polygon, and 3D geospatial objects
- Attributes
- Spatial indexing
  - R-tree (R*-tree), 3D R-tree
  - Quad-tree
- Topological operations
  - Overlapping
  - Intersection
  - K-nearest neighbors
  - Within
  - …

Distributed Databases



Why distributed database

- Your single machine runs out of storage!
  - Easy scalability
- Avoids single point of failure by distributing data among multiple machines.
  - Will not lose any data in case a machine fails.
  - Will not lose the ability to query the data in case a machine fails.
  - Machines that are performance bottlenecks will not affect the overall performance of the system.

Why distributed database

- Reduce query time by “intelligently” distribute data among multiple nodes.
  - A query should look into a smaller subset of data.
  - In write operations the index should be updated for a smaller set of data.
- Distribute workload on multiple nodes in case of simultaneous access.
  - Re-route requests to multiple machines (courtesy of replication).
- Incorporate organizational structure.
  - Each group wants to host their own data!

Challenges

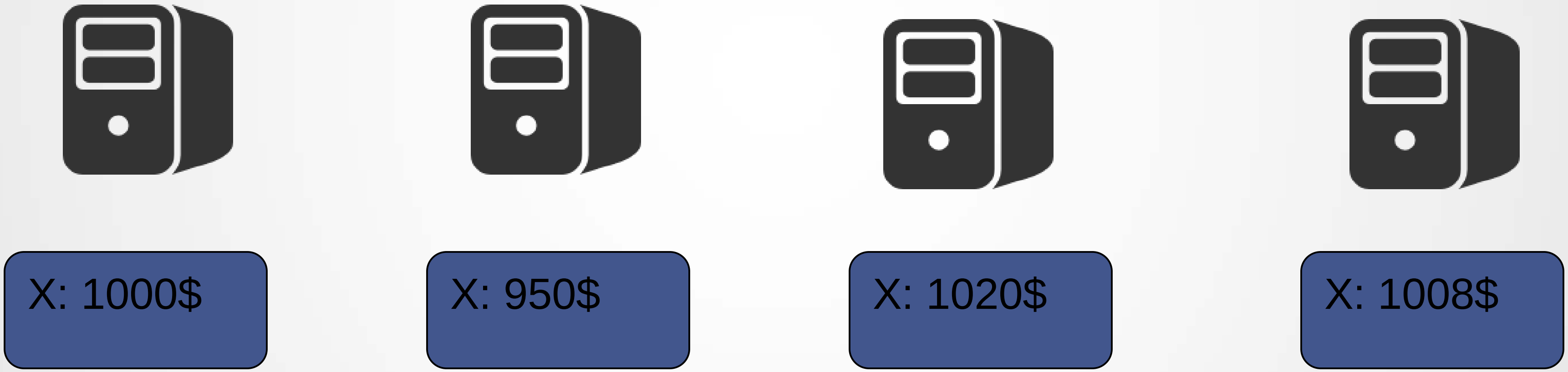
- Require new skills and knowledge for many database admins.
- Relatively new era. Still lots of development is being made.
- Issues with integrity and consistency of the database.
- More issues on ensuring the security of the database cluster.
- Require new database design techniques.

Concepts

- Replication: System maintains multiple identical copies of the data.
  - Increase latency on update.
- Partitioning: The dataset is divided into multiple smaller datasets based on a pre-defined function.
  - Balancing the load on different chunks of data should be considered.
- Consistency: any transaction must only affect data in allowed ways.

Consistency

X : John’s balance



Types of Consistency

- **Strong consistency:** Every client can observe one consistent stage.
- **Weak consistency:** relaxing some constraints on the strong consistency to gain better performance and availability.

CAP Theorem

It is impossible for a distributed computer system to simultaneously provide all of these 3 guarantees:

- Consistency
- Availability
- Partition tolerance
  - The system can continue its operation despite partitioning due to failures.

Examples of NoSQL Distributed Databases

mongoDB

redis

cassandra

MongoDB

- Document-oriented database.
  - Store records in a json-like format.
- Easy-to-use.
- Supports replication and partitioning (sharding).
- Support primary and secondary indexes.
- Support aggregation queries through:
  - Javascript-basd MapReduce structure which can support arbitrary large data but it is slow.
  - C++ based aggregation framework which is limited in size but very fast.

Apache Cassandra

- Highly decentralized architecture.
- Read/write throughput both scale linearly as the new machines are added.
- Very fast write performance.
- The admin can tune the consistency level based on the application.
- Most popular option in the industry for massive and highly distributed datasets.
- Easy integration with Hadoop, Hive, etc.
- Limited support for geospatial operations.

Data Model

Partitioning Key

Clustering Key

<table>
<tr><td rowspan="2">Row ID</td><td>Field 1 | Field 2</td><td>Field 1 | Field 2</td><td>Field 1 | Field 2</td></tr>
<tr><td>Other fields</td><td>Other fields</td><td>Other fields</td></tr>
</table>

<table>
<tr><td rowspan="2">Bus Station</td><td>Time | Bus Id</td><td>Time | Bus Id</td><td>Time | Bus Id</td></tr>
<tr><td># of passenger</td><td># of passenger</td><td># of passenger</td></tr>
</table>

Query Language: CQL

```
INSERT INTO BUSDATA (station, time, id, passengers) VALUES (100, 201504041000, 13222, 22);
```

```
SELECT * FROM BUSDATA WHERE station = 100 AND time >= 201510010000;
```

```
UPDATE BUSDATA SET passengers = passengers + 5 WHERE station = 100 AND time <= 201511010000 AND time >=201510010000;
```

Redis

- In memory key-value stores.
- Supports limited operations but it is significantly faster for smaller datasets.
- Supports replications.
- Provide multiple methods to ensure persistency of the data.
- Main supported data types:
  - Lists
  - Sets and Sorted Sets.
  - Hash Tables.

Basic Commands

SET “13132” “John Smith”

GET “13132” “John Smith”

ZADD “arrivals” 201410100105 “american airlines 3322”

ZRANGEBYSCORE “arrivals” 201410010000 201410052359

HSET “employees” “13132” “John Smith”

HGET “employees” “13132”

Geospatial Commands

GEOADD Illinois -87.6847 41.8369 “Chicago”

GEOADD Illinois -88.2728 40.1150 “Champaign”

GEODIST Illinois Champaign Chicago mi

GEORADIUS Illinois -88.2000 40.3442 200 km WITHDIST

Details: http://redis.io/commands/georadius

Performance Comparison

Solving Big Data Challenges for Enterprise Application Performance Management

Mainly from University of Toronto
VLDB 2012

- Benchmark the databases with different workloads (read-heavy, write-heavy, mixed).
- Benchmark the database with different metrics (latency (sec) and throughput (ops/sec).

Performance Comparison

- Cassandra is a good choice for applications with high insertion rate and applications which requires great scalability.
- Redis performs significantly faster than other database systems in read-heavy loads, however it does not scale well. In addition it comes with a limitation on the size of data.
- MongoDB works fine for smaller clusters, but shows significant performance drop as we move to larger clusters.

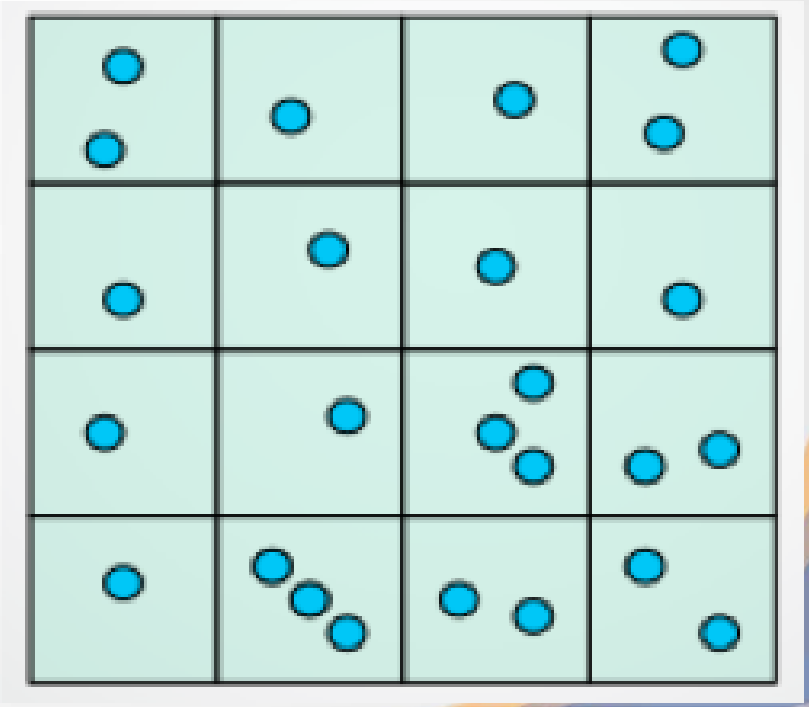
Case Study: Spatial Query Engine

- Establish a generic and scalable query framework for geospatial big data on cyberinfrastructure.
- Support query patterns in a scalable way.
- Use Redis to store the indexing structure in memory and the actual data on Cassandra database.
- Design for integrated spatiotemporal queires.
- Arbitrary input and query shapes.

Case Study: Spatial Query Engine

Polygon Indexing

Point Indexing



Experiments

- 2 weeks of geo-tagged tweets in US
- Rectangular query shape
- 100 queries per level (point and 3 polygon levels)
- Temporal length: one week
- Maximum width/height in miles
  - Point: 2.0
  - Zipcodes: 16.0, County: 64.0, State: 256

Result I

Database write and read -- local performance (ms)

Operation/Node	Average for Point data	Average for Polygon Data
Local Write Latency	0.0445	0.984
Local Read Latency	21.738	3.122

Result II

End-to-end query performance(ms)

Query Type/Latency	Mean	Average	90% Percentile
Polygon Level 0	82	85	160
Polygon Level 1	78	80	142
Polygon Level 2	158	190	322
Point	75	121	200

MongoDB in CyberGIS OpenStack

- Managing MongoDB in OpenStack
  - Create a new instance(s) in OpenStack to host MongoDB server (clusters)
  - For this class, we have created a MongoDB instance
  - Use a GUI client to access MongoDB

- Using MongoDB
  - Choose your favorite programing language to access the databases
  - Using command line interface:

mongo --host **hostserver_name:port (default: 27017)**/database_name

mongo –-host 141.142.168.54

show dbs

use <name> to create a new database

Geo-located Tweets query in Chicago

- Getting started with MongoDB in OpenStack
  - Choose your programming language and setup appropriate drivers
  - List of commands and guidelines:
    - https://docs.mongodb.org/manual/core/crud-introduction/
- Prepare geo-located Tweets as documents to MongoDB
  - MongoDB uses JSON objects to store the data
  - Convert your data structure to JSON
  - Covert geospatial objects to GeoJSON
  - They are essentially <key, value> pairs but with hierarchical structure

Geo-located Tweets query in Chicago

- A GeoJSON object

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [125.6, 10.1]
  },
  "properties": {
    "name": "Dinagat Islands"
  }
}
```

- Geo-located tweets
  - Using Pig script to extract <User_id, latitude, longitude, timestamp, content>

Import Twitter Data into MongoDB

- from pymongo import MongoClient
- import datetime
- import **json**

- def main():
- client = MongoClient('mongodb://141.142.168.54:27017')
- db = client.ddbs
- collection = db[**'chicago'**]

- mFile = open("2014_12_25_chi.txt","rb")
- for mLine in mFile:
- # print mLine
- [uid, lat, lng, tm, msg] = mLine.split("\t")
- geoString = {'user_id': uid, 'time': tm, 'message': msg, 'loc': {'type' : 'Point', 'coordinates' : [lng, lat]}}
- print geoString
- db.chicago.insert(geoString)

- if __name__ == '__main__':
- main()

Geo-located Tweets query in Chicago

- Create spatial index to speed up database querying
  - db.chicago.createIndex({ loc : "2dsphere" })
- Find the geo-located tweets within the search distances from a specified location

  db.chicago.find({loc: {$nearSphere: {$geometry: {type : "Point",coordinates : [-87.639160, 41.878628]},$minDistance: 1000,$maxDistance: 5000}}})
- Find the geo-located tweets within a specified region (polygon)

  db.chicago.find({loc:{$geoWithin: { $geometry: {type : "Polygon", coordinates: [[[-87.639546, 41.878053], [87.639643, 41.879506], [-87.638602,41.879363], [-87.638270,41.878045], [-87.639546, 41.878053]]]}}}})

Geo-located Tweets query in Chicago

- Sort the query result based on distance

```
db.chicago.find({loc: { $nearSphere: { $geometry: { type : "Point", coordinates : [-87.639160, 41.878628]},           $minDistance: 1000, $maxDistance: 5000}}}}).sort({"loc":1})
```

Pick up coordinates from Google Maps