Big Data and NoSQL

Chapter 10

Contents

Contents

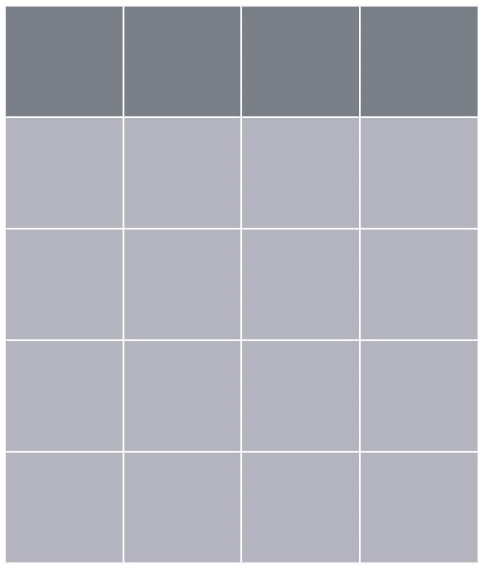
Introduction to Big Data

- Phenomenal growth in data generation
  - Social media
  - Sensors
  - Communications networks and satellite imagery
  - User-specific business data
- What is big data?
  - "Big data" refers to massive amounts of data
  - Exceeds the typical reach of a DBMS
  - Big data ranges from terabytes (10^{12} bytes) or petabytes (10^{15} bytes) to exobytes (10^{18} bytes)

Characteristics of Big Data?

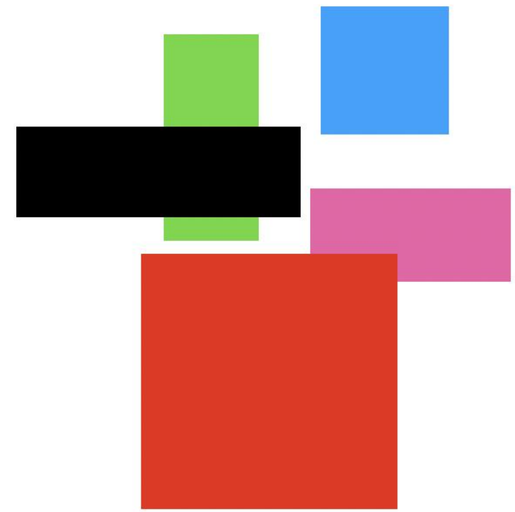
- **Volume:** Refers to size of data managed by the system
- **Velocity**: Speed of data creation, ingestion, and processing
- **Variety**:
  - Refers to type of data source
  - Structured, unstructured
- **Veracity**:
  - Credibility of the source
  - Suitability of data for the target audience
  - Evaluated through quality testing or credibility analysis

How are big data systems different from traditional database systems



Structured

e.g. Database tables



Semi- or Un-structured

e.g. JSON, XML,
Images, Videos …

Big Data Landscape

Vertical Apps
- Predictive Policing
- bloomreach
- Myrrix

Log Data Apps
- splunk
- loggly
- sumologic

Ad/Media Apps
- rocketfuel
- collective[i]
- bluefin
- Recorded Future
- LuckySort
- Media Science
- Turn
- DataXu

Data As A Service
- kaggle
- factual.
- knoema
- GNIP
- DataSift
- Windows Azure Marketplace
- INRIX
- LexisNexis
- Space Curve
- LOQATE

Business Intelligence
- Oracle | Hyperion
- SAP Business Objects
- RJMetrics
- Microsoft Business Intelligence
- IBM Cognos
- birst
- MicroStrategy
- Autonomy
- QlikView
- bime
- Domo
- Chart.io
- GoodData

Analytics and Visualization
- tableau
- Palantir
- Opera
- metaLayer
- dataspora
- Metamarkets
- centrifuge
- Teradata Aster
- SAS
- TIBCO
- Karmasphere
- panopticon
- pentaho
- Datameer
- platfora
- ClearStory
- Cirro
- alteryx
- visual.ly
- Ayata

Analytics Infrastructure
- Hortonworks
- Vertica
- MapR
- cloudera
- Infobright
- ParAccel
- EMC²
- Greenplum
- Netezza
- kognitio
- DataStax
- Exasol
- calpont

Operational Infrastructure
- Couchbase
- 10gen
- Teradata
- Hadapt
- Terracotta
- VoltDB
- MarkLogic
- Informatica

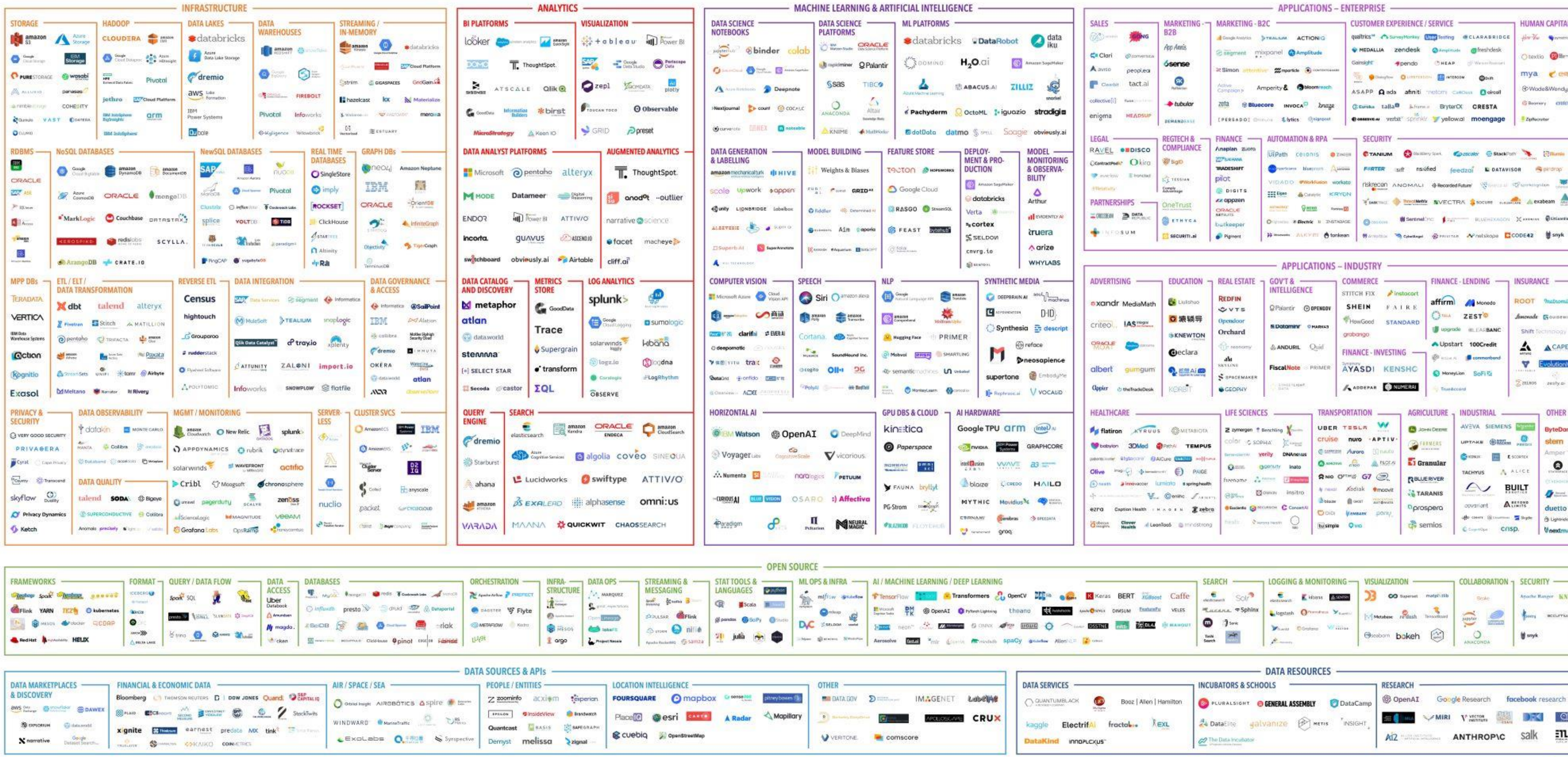
Infrastructure As A Service
- amazon web services
- Windows Azure
- infochimps
- Google BigQuery

Structured Databases
- Oracle
- MySQL
- SQL Server
- PostgreSQL
- IBM DB2
- Sybase
- memsql

Technologies
- hadoop
- hadoop MapReduce
- mahout
- Apache HBase
- Cassandra

dave@vcdave.com
blogs.forbes.com/davefeinleib

MACHINE LEARNING, ARTIFICIAL INTELLIGENCE, AND DATA (MAD) LANDSCAPE 2021



Version 3.0 - November 2021

© Matt Turck (@mattturck), John Wu (@john_d_wu) & FirstMark (@firstmarkcap)

mattturck.com/data2021

FIRSTMARK
EARLY STAGE VENTURE CAPITAL

Contents

Introduction

- NOSQL
  - Not only SQL
- Most NOSQL systems are distributed databases or distributed storage systems
  - Focus on semi-structured data storage, high performance, availability, data replication, and scalability

Introduction

- NOSQL systems focus on storage of "big data"
- Typical applications that use NOSQL
  - Social media
  - User profiles
  - Marketing and sales
  - Posts and tweets
  - Road maps and spatial data
  - Email

Characteristics of NOSQL Systems

- NOSQL characteristics related to distributed databases and distributed systems:
  - NOSQL systems emphasize **high availability**, so replicating the data is inherent in many of these systems.
  - **Scalability** is another important characteristic, because many of the applications that use NOSQL systems tend to have data that keeps growing in volume.
  - **High performance** is another required characteristic, whereas serializable consistency may not be as important for some of the NOSQL applications.

Characteristics of NOSQL Systems

- NOSQL characteristics related to data models and query languages:
  - Not Requiring a Schema: The users can specify a partial schema in some systems to improve storage efficiency, but *it is not required to have a schema* in most of the NOSQL systems.
  - Less Powerful Query Languages:
    - Search queries in these systems often locate single objects in a single file based on their object keys.
    - NOSQL systems typically provide a set of functions and operations as a programming API
  - Versioning: Some NOSQL systems provide storage of multiple versions of the data items, with the timestamps of when the data version was created.

Categories of NOSQL Systems

- Categories of NOSQL systems
  - Document-based NOSQL systems
  - NOSQL key-value stores
  - Column-based or wide column NOSQL systems
  - Graph-based NOSQL systems
  - Hybrid NOSQL systems

Document-Based NOSQL Systems and MongoDB

- Document-based or document-oriented NOSQL systems typically store data as **collections** of similar **documents**.
- Individual documents resemble complex objects or XML documents
  - Documents are self-describing
  - Can have different data elements
- Documents can be specified in various formats
  - XML
  - JSON

MongoDB Data Model

- Documents stored in binary JSON (BSON) format
- Individual documents stored in a collection
- Example command
  - First parameter specifies name of the collection
  - Collection options include limits on size and number of documents

```
db.createCollection("project", { capped : true, size : 1310720, max : 500 } )
```

- Each document in collection has unique ObjectID field called _id

MongoDB Data Model

- A collection does not have a schema
  - Structure of the data fields in documents chosen based on how documents will be accessed
  - User can choose normalized or denormalized design

- Document creation using insert operation

```
db.<collection_name>.insert(<document(s)>)
```

- Document deletion using remove operation

```
db.<collection_name>.remove(<condition>)
```

- Example of simple documents in MongoDB
  - (a) Denormalized document design with embedded subdocuments
  - (b) Embedded array of document references

(a) **project document with an array of embedded workers:**

```
{
      _id:              "P1",
      Pname:            "ProductX",
      Plocation:        "Bellaire",
      Workers: [
                    { Ename: "John Smith",
                      Hours: 32.5
                    },
                    { Ename: "Joyce English",
                      Hours: 20.0
                    }
               ]
);
```

(b) **project document with an embedded array of worker ids:**

```
{
      _id:              "P1",
      Pname:            "ProductX",
      Plocation:        "Bellaire",
      WorkerIds:        [ "W1", "W2" ]
}
      { _id:            "W1",
      Ename:            "John Smith",
      Hours:            32.5
}
      { _id:            "W2",
      Ename:            "Joyce English",
      Hours:            20.0
}
```

- Example of simple documents in MongoDB
  - (c) Normalized documents
  - (d) Inserting the documents into their collections

(c) normalized project and worker documents (not a fully normalized design for M:N relationships):

```
{
    _id:            "P1",
    Pname:          "ProductX",
    Plocation:      "Bellaire"
}
{   _id:            "W1",
    Ename:          "John Smith",
    ProjectId:      "P1",
    Hours:          32.5
}

{   _id:            "W2",
    Ename:          "Joyce English",
    ProjectId:      "P1",
    Hours:          20.0
}
```

(d) inserting the documents in (c) into their collections "project" and "worker":

```
db.project.insert( { _id: "P1", Pname: "ProductX", Plocation: "Bellaire" } )
db.worker.insert( [ { _id: "W1", Ename: "John Smith", ProjectId: "P1", Hours: 32.5 },
                    { _id: "W2", Ename: "Joyce English", ProjectId: "P1",
                      Hours: 20.0 } ] )
```

NOSQL Key-Value Stores

- Key-value stores focus on high performance, availability, and scalability
- Can store structured, unstructured, or semi-structured data
- Key: unique identifier associated with a data item
  - Used for fast retrieval
- Value: the data item itself
  - Can be string or array of bytes
  - Application interprets the structure
- No query language

Examples of Key-Value Stores

- DynamoDB
  - Part of Amazon's Web Services/SDK platforms
  - Table holds a collection of self-describing items
  - Item consists of attribute-value pairs
    - Attribute values can be single or multi-valued
  - Primary key used to locate items within a table.
    - Can be single attribute or pair of attributes
- Oracle key-value store
  - Oracle NOSQL Database
- Redis key-value cache and store
  - Caches data in main memory to improve performance
  - Offers master-slave replication and high availability
  - Offers persistence by backing up cache to disk
- Apache Cassandra
  - Offers features from several NOSQL categories
  - Used by Facebook and others

Column-Based or Wide Column NOSQL Systems

- BigTable: Google's distributed storage system for big data
  - Used in Gmail
  - Uses Google File System for data storage and distribution
- Apache Hbase a similar, open source system
  - Uses Hadoop Distributed File System (HDFS) for data storage
  - Can also use Amazon's Simple Storage System(S3)

NOSQL Graph Databases and Neo4j

- Graph databases
  - Data represented as a graph
  - Collection of vertices (nodes) and edges
  - Possible to store data associated with both individual nodes and individual edges
- Neo4j
  - Open source system
  - Uses concepts of nodes and relationships

Neo4j

- Nodes can have labels
  - the nodes that have the same label are grouped into a collection that identifies a subset of the nodes in the database graph for querying purposes.
  - A node can have zero, one, or several labels.
- Both nodes and relationships can have properties
- Each relationship has a start node, end node, and a relationship type
- Properties specified using a map pattern, which is made of one or more “name : value” pairs enclosed in curly brackets
  - {Lname : ‘Smith’, Fname : ‘John’, Minit : ‘B’}
- Somewhat similar to ER/EER concepts

Neo4j

- Creating nodes in Neo4j
  - CREATE command
  - Part of high-level declarative query language Cypher
  - Node label can be specified when node is created
  - Properties are enclosed in curly brackets

Neo4j - Examples in Neo4j using the Cypher language

(a) creating some nodes for the COMPANY data

```
CREATE (e1: EMPLOYEE, {Empid: ‘1’, Lname: ‘Smith’, Fname: ‘John’, Minit: ‘B’})
CREATE (e2: EMPLOYEE, {Empid: ‘2’, Lname: ‘Wong’, Fname: ‘Franklin’})
CREATE (e3: EMPLOYEE, {Empid: ‘3’, Lname: ‘Zelaya’, Fname: ‘Alicia’})
CREATE (e4: EMPLOYEE, {Empid: ‘4’, Lname: ‘Wallace’, Fname: ‘Jennifer’, Minit: ‘S’})
…
CREATE (d1: DEPARTMENT, {Dno: ‘5’, Dname: ‘Research’})
CREATE (d2: DEPARTMENT, {Dno: ‘4’, Dname: ‘Administration’})
…
CREATE (p1: PROJECT, {Pno: ‘1’, Pname: ‘ProductX’})
CREATE (p2: PROJECT, {Pno: ‘2’, Pname: ‘ProductY’})
CREATE (p3: PROJECT, {Pno: ‘10’, Pname: ‘Computerization’})
CREATE (p4: PROJECT, {Pno: ‘20’, Pname: ‘Reorganization’})
…
CREATE (loc1: LOCATION, {Lname: ‘Houston’})
CREATE (loc2: LOCATION, {Lname: ‘Stafford’})
CREATE (loc3: LOCATION, {Lname: ‘Bellaire’})
CREATE (loc4: LOCATION, {Lname: ‘Sugarland’})
…
```

Neo4j - Examples in Neo4j using the Cypher language

(b) **creating some relationships for the COMPANY data**

```
CREATE (e1) – [ : WorksFor ] –> (d1)
CREATE (e3) – [ : WorksFor ] –> (d2)
...
CREATE (d1) – [ : Manager ] –> (e2)
CREATE (d2) – [ : Manager ] –> (e4)
...
CREATE (d1) – [ : LocatedIn ] –> (loc1)
CREATE (d1) – [ : LocatedIn ] –> (loc3)
CREATE (d1) – [ : LocatedIn ] –> (loc4)
CREATE (d2) – [ : LocatedIn ] –> (loc2)
...
CREATE (e1) – [ : WorksOn, {Hours: '32.5'} ] –> (p1)
CREATE (e1) – [ : WorksOn, {Hours: '7.5'} ] –> (p2)
CREATE (e2) – [ : WorksOn, {Hours: '10.0'} ] –> (p1)
CREATE (e2) – [ : WorksOn, {Hours: 10.0} ] –> (p2)
CREATE (e2) – [ : WorksOn, {Hours: '10.0'} ] –> (p3)
CREATE (e2) – [ : WorksOn, {Hours: 10.0} ] –> (p4)
...
```

Neo4j - Examples in Neo4j using the Cypher language

(c) **Basic simplified syntax of some common Cypher clauses:**

Finding nodes and relationships that match a pattern: MATCH <pattern>
Specifying aggregates and other query variables: WITH <specifications>
Specifying conditions on the data to be retrieved: WHERE <condition>
Specifying the data to be returned: RETURN <data>
Ordering the data to be returned: ORDER BY <data>
Limiting the number of returned data items: LIMIT <max number>
Creating nodes: CREATE <node, optional labels and properties>
Creating relationships: CREATE <relationship, relationship type and optional properties>
Deletion: DELETE <nodes or relationships>
Specifying property values and labels: SET <property values and labels>
Removing property values and labels: REMOVE <property values and labels>

Neo4j - Examples in Neo4j using the Cypher language

(d) Examples of simple Cypher queries:

1. MATCH (d : DEPARTMENT {Dno: '5'}) – [: LocatedIn] → (loc)
   RETURN d.Dname , loc.Lname
2. MATCH (e: EMPLOYEE {Empid: '2'}) – [w: WorksOn] → (p)
   RETURN e.Ename , w.Hours, p.Pname
3. MATCH (e) – [w: WorksOn] → (p: PROJECT {Pno: 2})
   RETURN p.Pname, e.Ename , w.Hours
4. MATCH (e) – [w: WorksOn] → (p)
   RETURN e.Ename , w.Hours, p.Pname
   ORDER BY e.Ename
5. MATCH (e) – [w: WorksOn] → (p)
   RETURN e.Ename , w.Hours, p.Pname
   ORDER BY e.Ename
   LIMIT 10
6. MATCH (e) – [w: WorksOn] → (p)
   WITH e, COUNT(p) AS numOfprojs
   WHERE numOfprojs > 2
   RETURN e.Ename , numOfprojs
   ORDER BY numOfprojs
7. MATCH (e) – [w: WorksOn] → (p)
   RETURN e , w, p
   ORDER BY e.Ename
   LIMIT 10
8. MATCH (e: EMPLOYEE {Empid: '2'})
   SET e.Job = 'Engineer'

Contents


Q
A