NoSQL Databases and Big Data Technologies

Week 9

COS60009: Data Management for the Big Data Age

1

Learning Objectives

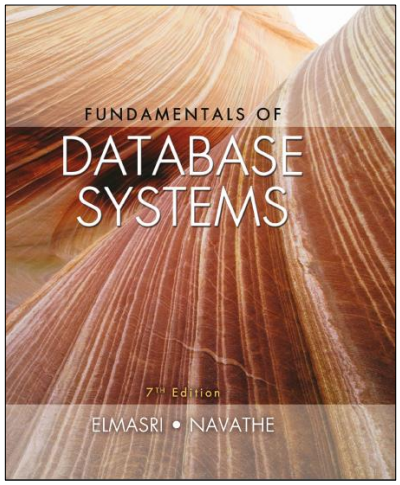
- Introduction to NoSQL Systems
- CAP Theorem
- Document-Based NoSQL Systems and Mongo DB
- NoSQL Key-Value Stores
- Column-Based or Wide Column NoSQL Systems
- NoSQL Graph Databases and Neo4j
- Big Data
- The MapReduce Programming Model
- Hadoop Distributed File System (HDFS)
- Comparisons between SQL and NoSQL Systems

Pearson

2

Fundamentals of Database Systems

Seventh Edition



Chapter 24

NoSQL Databases and Big Data Storage Systems

Chapter 25

Big Data Technologies Based on MapReduce and Hadoop

Other Big Data Materials

3

Introduction to NoSQL Systems

- 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
- NoSQL systems focus on storage of “big data”
- Typical applications that use NoSQL
  - Social media
  - Web links
  - User profiles
  - Marketing and sales
  - Posts and tweets
  - Road maps and spatial data
  - Email

4

Sample NoSQL Systems and Categories

- BigTable
  - Google's proprietary NoSQL system
  - **Column-based** or wide column store
- Dynamo DB (Amazon)
  - **Key-value** data store
- Cassandra (Facebook)
  - Uses concepts from both key-value store and column-based systems
- Mongo DB and Couch DB
  - **Document-based** stores
- Neo4J and GraphBase
  - **Graph-based** NoSQL systems
- Orient DB
  - Combines several concepts
- Database systems classified on the object model
  - Or native XML model

5

Characteristics of NoSQL Systems

- NoSQL characteristics related to distributed databases and distributed systems
  - Scalability
  - Availability, replication, and eventual consistency
  - Replication models
    - Master-slave
    - Master-master
  - Sharding of files
  - High performance data access
- NoSQL characteristics related to data models and query languages
  - Schema not required
  - Less powerful query languages
  - Versioning

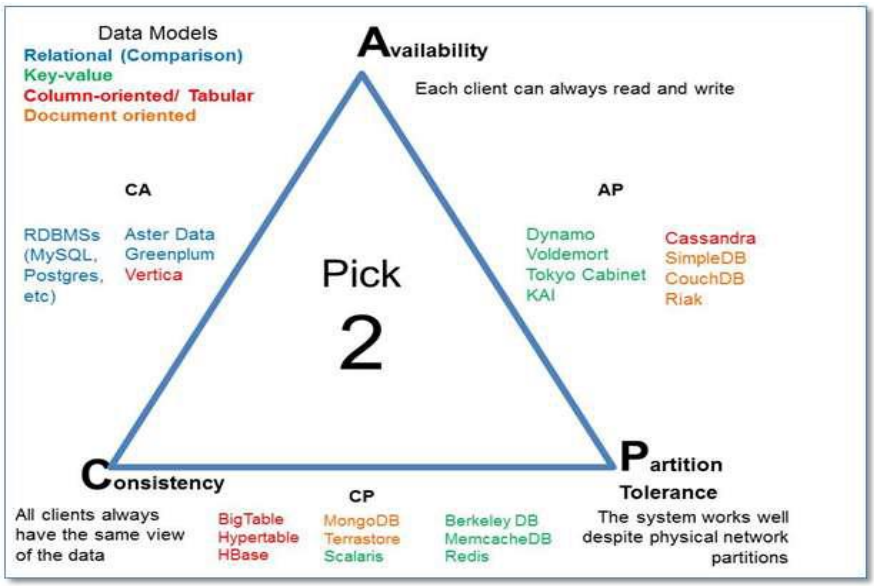
6

The CAP Theorem

- Various levels of consistency among replicated data items
  - Enforcing serializabilty the strongest form of consistency
    - High overhead – can reduce read/write operation performance
- CAP theorem
  - Consistency, Availability, and Partition tolerance
  - **Not possible to guarantee all three simultaneously**
    - In distributed system with data replication
- Designer can choose two of three to guarantee
  - Weaker consistency level is often acceptable in NoSQL distributed data store
  - Guaranteeing availability and partition tolerance more important
  - Eventual consistency often adopted

7

Which data model to choose



8

Document-Based NoSQL Systems and MongoDB

- Document stores
  - 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

9

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 Object ID field called _id
- 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>)
```


10

Figure 24.1 Example of Simple Documents in MongoDB (1 of 2)

(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
}
```

Pearson

11

Figure 24.1 Example of Simple Documents in MongoDB (2 of 2)

c) Normalized documents (d) Inserting the documents in Figure 24.1(c) 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 } ] )
```

Pearson

12

MongoDB Distributed Systems Characteristics

- Two-phase commit method
  - Used to ensure atomicity and consistency of multidocument transactions
- Replication in MongoDB
  - Concept of replica set to create multiple copies on different nodes
  - Variation of master-slave approach
  - Primary copy, secondary copy, and arbiter
  - All write operations applied to the primary copy and propagated to the secondaries
  - User can choose read preference, read requests can be processed at any replica
- Sharding in MongoDB
  - Horizontal partitioning divides the documents into disjoint partitions (shards)
  - Allows adding more nodes as needed
  - Shards stored on different nodes to achieve load balancing
  - Partitioning field (shard key) must exist in every document in the collection
    - Must have an index
  - Range partitioning
    - Creates chunks by specifying a range of key values
    - Works best with range queries
  - Hash partitioning
    - Partitioning based on the hash values of each shard key

13

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

14

DynamoDB Overview

- DynamoDB part of Amazon's Web Services/SDK platforms
  - Proprietary
- 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

15

Other Key-Value Stores

- Voldemort: open source key-value system similar to DynamoDB
- 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

16

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)

17

Hbase Data Model and Versioning

- Data organization concepts: Namespaces, Tables, Column families, Column qualifiers, Columns, Rows, Data cells
- Data is self-describing
- HBase stores multiple versions of data items
  - Timestamp associated with each version
- Each row in a table has a unique row key
- Table associated with one or more column families
- Column qualifiers can be dynamically specified as new table rows are created and inserted
- Namespace is collection of tables
- Cell holds a basic data item

18

Figure 24.3 Examples in Hbase

(a) Creating a table called EMPLOYEE with three column families: Name, Address, and Details (b) Inserting some in the EMPLOYEE table; different rows can have different self-describing column qualifiers (Fname, Lname, Nickname, Mname, Minit, Suffix, … for column family Name; Job, Review, Supervisor, Salary for column family Details). (c) Some CRUD operations of Hbase

(a) creating a table:

```
create 'EMPLOYEE', 'Name', 'Address', 'Details'
```

(b) inserting some row data in the EMPLOYEE table:

```
put 'EMPLOYEE', 'row1', 'Name:Fname', 'John'
put 'EMPLOYEE', 'row1', 'Name:Lname', 'Smith'
put 'EMPLOYEE', 'row1', 'Name:Nickname', 'Johnny'
put 'EMPLOYEE', 'row1', 'Details:Job', 'Engineer'
put 'EMPLOYEE', 'row1', 'Details:Review', 'Good'
put 'EMPLOYEE', 'row2', 'Name:Fname', 'Alicia'
put 'EMPLOYEE', 'row2', 'Name:Lname', 'Zelaya'
put 'EMPLOYEE', 'row2', 'Name:MName', 'Jennifer'
put 'EMPLOYEE', 'row2', 'Details:Job', 'DBA'
put 'EMPLOYEE', 'row2', 'Details:Supervisor', 'James Borg'
put 'EMPLOYEE', 'row3', 'Name:Fname', 'James'
put 'EMPLOYEE', 'row3', 'Name:Minit', 'E'
put 'EMPLOYEE', 'row3', 'Name:Lname', 'Borg'
put 'EMPLOYEE', 'row3', 'Name:Suffix', 'Jr.'
put 'EMPLOYEE', 'row3', 'Details:Job', 'CEO'
put 'EMPLOYEE', 'row3', 'Details:Salary', '1,000,000'
```

(c) Some Hbase basic CRUD operations:

```
Creating a table: create <tablename>, <column family>, <column family>, ...
Inserting Data: put <tablename>, <rowid>, <column family>:<column qualifier>, <value>
Reading Data (all data in a table): scan <tablename>
Retrieve Data (one item): get <tablename>,<rowid>
```


19

Hbase Crud Operations

- Provides only low-level CRUD (create, read, update, delete) operations
- Application programs implement more complex operations
- Create
  - Creates a new table and specifies one or more column families associated with the table
- Put
  - Inserts new data or new versions of existing data items
- Get
  - Retrieves data associated with a single row
- Scan
  - Retrieves all the rows

20

Hbase Storage and Distributed System Concepts

- Each Hbase table divided into several regions
  - Each region holds a range of the row keys in the table
  - Row keys must be lexicographically ordered
  - Each region has several stores
    - Column families are assigned to stores
- Regions assigned to region servers for storage
  - Master server responsible for monitoring the region servers
- Hbase uses Apache Zookeeper and HDFS

21

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

22

Neo4j

- Nodes 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
- Somewhat similar to ER/EER concepts
- 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
- Path
  - Traversal of part of the graph
  - Typically used as part of a query to specify a pattern
- Schema optional in Neo4j
- Indexing and node identifiers
  - Users can create for the collection of nodes that have a particular label
  - One or more properties can be indexed

23

Neo4j – Creating Nodes

(a) creating some nodes for the COMPANY data (from Figure 5.6):

```
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'})
...
```

Figure 24.4 Examples in Neo4j using the Cypher language (a) Creating some nodes

24

Neo4j – Creating Relationships

(b) creating some relationships for the COMPANY data (from Figure 5.6):

```
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)
...
```

Figure 24.4 (b) Creating some relationships

25

The Cypher Query Language of Neo4j

- Cypher query made up of clauses
- Result from one clause can be the input to the next clause in the query

(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>

Figure 24.4 (c) Basic syntax of Cypher queries

26

The Cypher Query Language of Neo4j (Cont'd)

Figure 24.4
(d) Examples of Cypher queries

(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'

27

Big Data

- Phenomenal growth in data generation
  - Social media
  - Sensors
  - Communications networks and satellite imagery
  - User-specific business data
- "Big data" refers to massive amounts of data
  - Exceeds the typical reach of a DBMS
- Big data analytics
- Big data ranges from terabytes (10^{12} bytes) or petabytes (10^{15} bytes) to to exobytes (10^{18} bytes)
- 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
- (Additional) Veracity - credibility of the source, suitability of data for the target audience, evaluated through quality testing or credibility analysis

28

Introduction to MapReduce and Hadoop

- Core components of Hadoop
  - MapReduce programming paradigm
  - Hadoop Distributed File System (HDFS)
- Hadoop originated from quest for open source search engine
  - Developed by Cutting and Carafella in 2004
  - Cutting joined Yahoo in 2006
  - Yahoo spun off Hadoop-centered company in 2011
  - Tremendous growth
- MapReduce
  - Fault-tolerant implementation and runtime environment
  - Developed by Dean and Ghemawat at Google in 2004
  - Programming style: map and reduce tasks
    - Automatically parallelized and executed on large clusters of commodity hardware
  - Allows programmers to analyze very large datasets
  - Underlying data model assumed: key-value pair

29

The MapReduce Programming Model

- Map
  - Generic function that takes a key of type K1 and value of type V1
  - Returns a list of key-value pairs of type K2 and V2
- Reduce
  - Generic function that takes a key of type K2 and a list of values V2 and returns pairs of type (K3, V3)
- Outputs from the map function must match the input type of the reduce function

30

The MapReduce Programming Model (Cont'd)

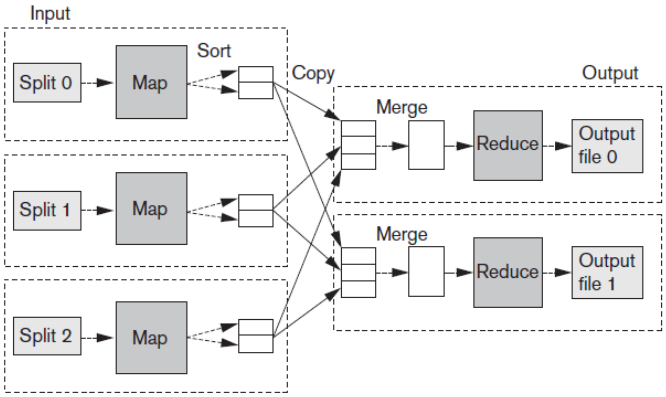


Figure 25.1 Overview of MapReduce execution (Adapted from T. White, 2012)

31

The MapReduce Programming Model (Cont'd)

- MapReduce example
  - Make a list of frequencies of words in a document
  - Pseudocode

```
Map (String key, String value):
    for each word w in value Emitintermediate (w, "1");

Reduce (String key, Iterator values) : // here the key is a word and values are
lists of its counts //
    Int result =0;
    For each v in values :
        result += Parseint (v);
    Emit (key, Asstring (result));
```


32

The MapReduce Programming Model (Cont'd)

- MapReduce example
  - Actual MapReduce code

```
map[LongWritable,Text](key, value) : List[Text, LongWritable] = {
   String[] words = split(value)
   for(word : words) {
      context.out(Text(word), LongWritable(1))
   }
}
reduce[Text, Iterable[LongWritable]](key, values) : List[Text, LongWritable] = {
   LongWritable c = 0
   for( v : values) {
      c += v
   }
   context.out(key,c)
   }
```


33

The MapReduce Programming Model (Cont'd)

- Inverted index
  - Builds an inverted index based on all words present in a document repository
  - Map function parses each document
    - Emits a sequence of (word, document_id) pairs
  - Reduce function takes all pairs for a given word and sorts them by document_id
- Job
  - Code for Map and Reduce phases, a set of artifacts, and properties
- Hadoop releases
  - 1.x features
    - Continuation of the original code base
    - Additions include security, additional HDFS and MapReduce improvements
  - 2.x features
    - YARN (Yet Another Resource Navigator)
    - A new MR runtime that runs on top of YARN
    - Improved HDFS that supports federation and increased availability

34

Hadoop Distributed File System

- HDFS
  - File system component of Hadoop
  - Designed to run on a cluster of commodity hardware
  - Patterned after UNIX file system
  - Provides high-throughput access to large datasets
  - Stores metadata on NameNode server
  - Stores application data on DataNode servers
    - File content replicated on multiple DataNodes
- HDFS design assumptions and goals
  - Hardware failure is the norm
  - Batch processing
  - Large datasets
  - Simple coherency model
- HDFS architecture
  - Master-slave
  - Decouples metadata from data operations
  - Replication provides reliability and high availability
  - Network traffic minimized

35

Hadoop Distributed File System (Cont'd)

- NameNode
  - Maintains image of the file system
    - i-nodes and corresponding block locations
  - Changes maintained in write-ahead commit log called Journal
- Secondary NameNodes
  - Checkpointing role or backup role
- DataNodes
  - Stores blocks in node's native file system
  - Periodically reports state to the NameNode
- e I/O operations
  - Single-writer, multiple-reader model
  - Files cannot be updated, only appended
  - Write pipeline set up to minimize network utilization
- Block placement
  - Nodes of Hadoop cluster typically spread across many racks
    - Nodes on a rack share a switch

36

Hadoop Distributed File System (Cont'd)

- Replica management
  - NameNode tracks number of replicas and block location
    - Based on block reports
  - Replication priority queue contains blocks that need to be replicated
- HDFS scalability
  - Yahoo cluster achieved 14 petabytes, 4000 nodes, 15k clients, and 600 million files

Pearson

37

Advantages of the Hadoop/MapReduce Technology

- Disk seek rate a limiting factor when dealing with very large data sets
  - Limited by disk mechanical structure
- Transfer speed is an electronic feature and increasing steadily
- MapReduce processes large datasets in parallel
- MapReduce handles semistructured data and key-value datasets more easily
- Linear scalability

Pearson

38

Why and Why-not NoSQL

- Why NoSQL?
  - Schema-free
  - Massive data stores
  - Scalability
  - Some services simpler to implement than using RDBMS
  - Great fit for many “Web 2.0” services
- Why NOT NoSQL?
  - RDBMS databases and tools are mature
  - NoSQL implementations often “alpha”
  - Data consistency, transactions (ACID properties)
  - “Don’t scale until you need it”

39

RDBMS vs NoSQL

- Strong consistency vs. Eventual consistency
- Big dataset vs. HUGE datasets
- Scaling is possible vs. Scaling is easy
- SQL vs. MapReduce
- Good availability vs. Very high availability

40

Copyright

Pearson

41