

Introduction to Big Data, NoSQL & MongoDB

Introduction to Big Data, NoSQL & MongoDB

Written By: Rony Keren
Internet Team
John Bryce Training

JOHN BRYCE
Leading in IT Education
a matrix company

Topics

JOHN BRYCE
Leading in IT Education
a matrix company

- Big Data – The challenge
- NoSQL
- MongoDB

Big Data – The Challenge

- Internet grows rapidly
 - More human users
 - More devices
 - More applications
- Since WEB 2.0, users contribute their own share
- Internet world is full of data

Big Data – The Challenge

Internet world is full of data

- Lists of users, profiles, items, user information
- Historical data recording
- Short terms history data (like security video recordings)
- Activity recording, logs
- User posts and shared contents
- Short term, relevant fresh data
- Information (news, movies,...)
- Publication and advertisements
- Meta data for everything



Was taken at one of Facebooks Data Centers

Since there is a huge amount of data we need more sophisticated ways to access it

- Lots of diverse data to store and load
- Intense integration and data transfer
- High, sometimes unpredictable, concurrency peaks



Lots of diverse data to store and load

- There are two main need from today's DBs:
 - Large amounts of data intensively stored and loaded
 - Data structure is very liquid (composed from multiple sources)
- With relational DBs we have the following problems:
 - Relational tables are good for static data types – not for diverse data
 - Too much efforts for converting data to and from SQL



Lots of diverse data to store and load

- According to Gartner:
 - Enterprise Data will grow 650% by 2014
 - 85% of these data will be “unstructured” and streamed data
 - Unstructured data (documents, videos, images...)
 - Streamed data(logs, GPS data, security cameras, sensors...)



Gartner

Lots of diverse data to store and load

It is easy to store records if we know that each record is an employee..

EmpID	Name	Salary
11	Adam	12,000
12	Alexandra	10,000
13	Bob	14,000
14	Chris	13,000
15	Christina	15,000
16	Dana	18,000
17	David	11,000

Employees



We have few tables with lots of records inside - OK

- Lots of diverse data to store and load
 - But what if some employees are also members in some organization, have their own hobbies and connections ?
 - What if some searches totally ignore the fact that they are employees ?
 - like a connection search?
 - hobby search?



Lots of diverse data to store and load

Relational DB solution 1: relationships

EmpID	Name	Salary
11	Adam	12,000
12	Alexandra	10,000
13	Bob	14,000
14	Chris	13,000
15	Christina	15,000
16	Dana	18,000
17	David	11,000

Employees

NetworkID	user
1	Facebook
2	LinkedIn
3	MySpace
4	Tweeter
5	Wiki
6	YouTube
7	Win Live ID

Connections

HobbyID	Name
1	Singing
2	Fishing
3	Hunting
4	Styling
5	Reading
6	Sports
7	Gamer

Hobbies



Problem: not every employee has a connection or a hobby,
DB is not normalized

Lots of diverse data to store and load

Relational DB solution 2: table per type

Employees		
EmpID	Name	Salary
16	Dana	18,000
17	David	11,000

Employees with both hobbies and connections				
EmpID	Name	Salary	Hobbies	Networks
11	Adam	12,000	join	join
12	Alexandra	10,000	join	join
13	Bob	14,000	join	join

Employees with hobbies			
EmpID	Name	Salary	Hobbies
15	Bob	14,000	join

Employees with connections			
EmpID	Name	Salary	Networks
14	Chris	13,000	join
15	Christina	15,000	join



Problem: lots of tables with few records each...

Solution

- NoSQL DB – Not only SQL DB products emerge
 - No usage of SQL in order to store / load data
 - Data is stored in its original format – not as table records
 - No usage of SQL



Intense integration and data transfer

- In most web application data is passed over HTTP
- In order to describe structured data we preferred XML
- Big Data requirements when it comes to XML
 - Lots of XML messages or/and
 - Very large XML documents



Intense integration and data transfer

- XML forces data wrapping with tags: `<greet> hello </greet>`
- We just want to pass 'hello' `<greet> hello </greet>`
- XML describes data accurately and this is great for integration
 - BUT
- XML uses heavy data wrappers – causing network overhead..
- XML forces the usage of parsers which are time consuming..



Solution

- In practice, when we are being asked to choose between standards and performance – we go for performance
- Integration through XML is well-defined but slow
- Using HTTP directly is not standardized – but much faster !



High, sometimes unpredictable, concurrency peaks

- Services might experience periods of peaks
 - Expected or unexpected
 - Intense client activity
 - Growth in resource demand



High, sometimes unpredictable, concurrency peaks

- Peaks requires more
 - CPU
 - Storage
 - Threads
 - Resources
 - Service replications
- But, most of the time we don't really use it...




Solution

- Cloud computing
 - Replicas of complete configurations are allocated on demand
 - Configurations might be
 - Hardware
 - Storage
 - Clusters
 - Services
 - All together



Big Data – in numbers



Bit	0 or 1
Byte	8 Bits
Kilobyte	1024 Bytes
Megabyte	1024 Kilobytes
Gigabyte	1024 Megabytes
Terabyte	1024 Gigabytes
Petabyte	1024 Terabytes
Exabyte	1024 Petabytes
Zettabyte	1024 Exabytes
Yottabyte	1024 Zettabytes (wow)

Big Data, for now...

Examples of typical “Big-Data” scenarios:

- Find the highest temperature in the US during last year
- Find users that might be interested in your hobby
- Track individuals or organizations to do business with
- Track the last year most popular item at e-bay
- Search members in a HUGE user list
- Rent millions of DVD movies by video streaming

So,

- We are dealing with petabytes here..
- Big Data requires
 - NoSQL Databases
 - Cloud Computing
 - Straight and flat usage of HTTP



- As a programmer, did you ever think of how a web search, using Google engine for example, can provide fast results out of billions of web pages in a few milliseconds ?
- Answer is **Map-Reduce** programming model
 - Built for working with huge data sets

Map – Reduce Pattern

- Map – Reduce helps in taking a drop from the ocean
 - Starting with general input, mapping what is relevant
 - Then, shuffling the data to come-up with a result
 - And reducing irrelevant data from it



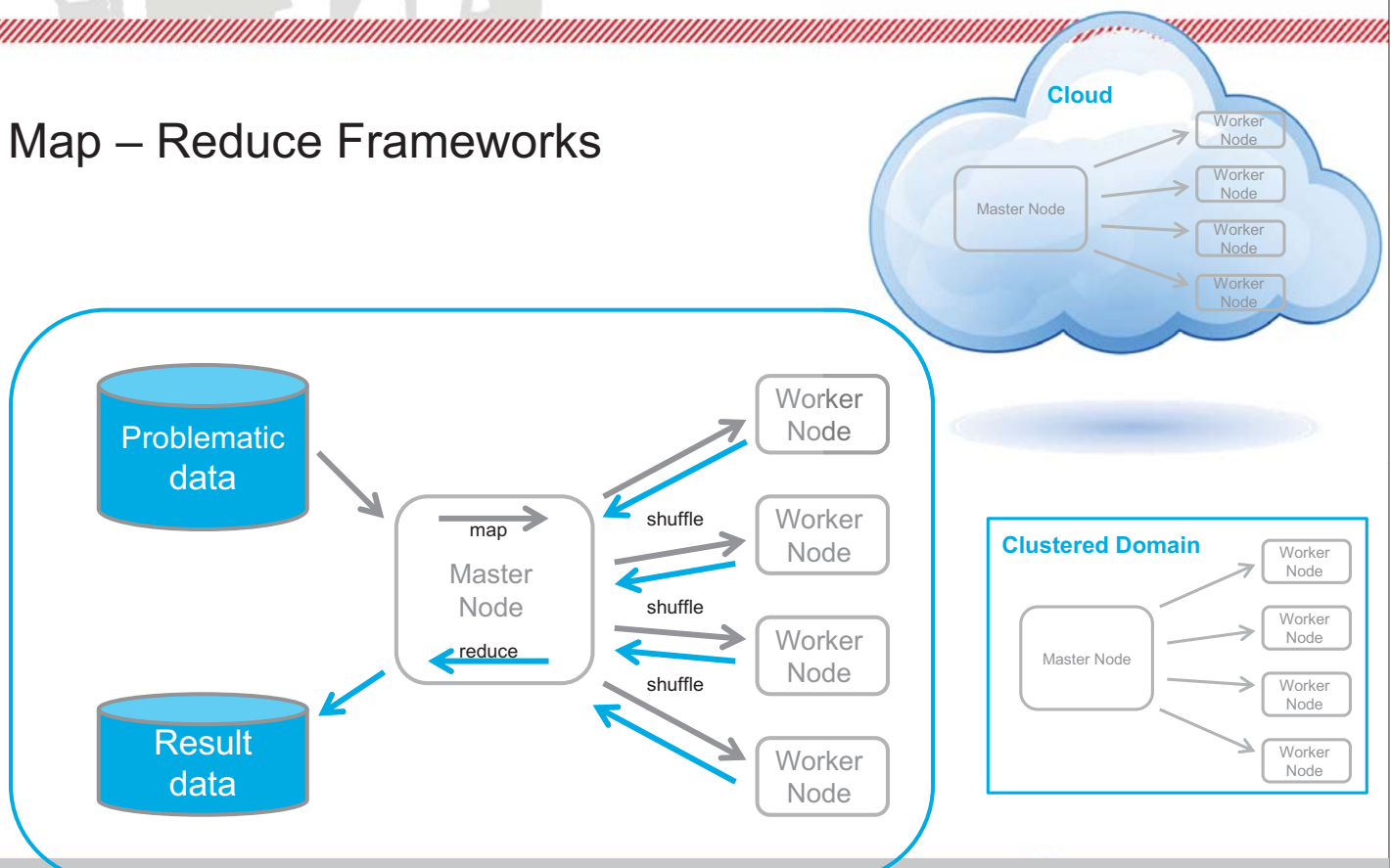
Sounds simple, what's the big deal ?

Map – Reduce Pattern

- Map – Reduce in parallel programming
 - In parallel execution programs the challenge is to access distributed resources
 - Shared resources must be isolated and this is a big headache...
- In other words:
 - Map & Reduce are simple, the pain is Shuffling !
 - Map – just express the input you expect
 - Reduce – we just take what we need out of the result...

- Map – Reduce Frameworks
 - Ease the way programmers can use the pattern
 - Developer focus on the easy parts: Map , Reduce
 - Shuffling is done by the system
 - Across clusters
 - In clouds, when new workers can be allocated
 - Good system shuffles rapidly...

Map – Reduce Frameworks



Map – Reduce Pattern

Example:

Problematic data: Number of citizens in every city

Data row example: **US NY 4m 4.5m**

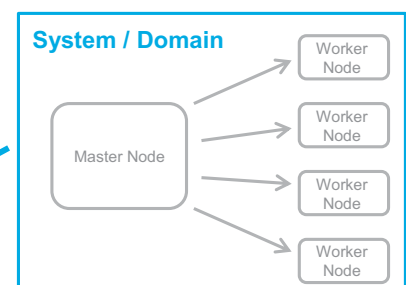
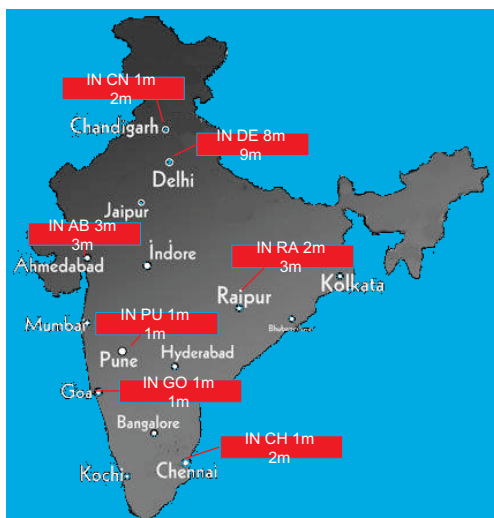
country city estimated no' of males & females

We need the total sum of females in cities in India..

Map – Reduce Pattern

Example:

Step 1 – we map all cities

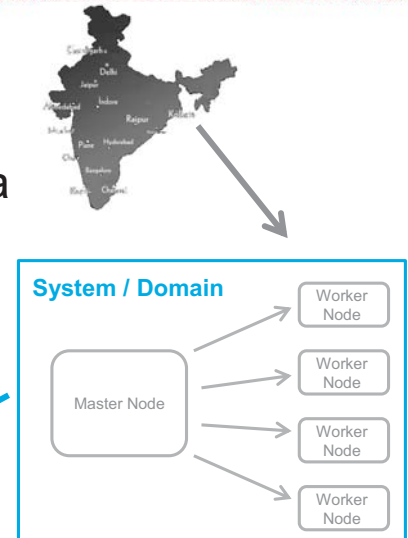
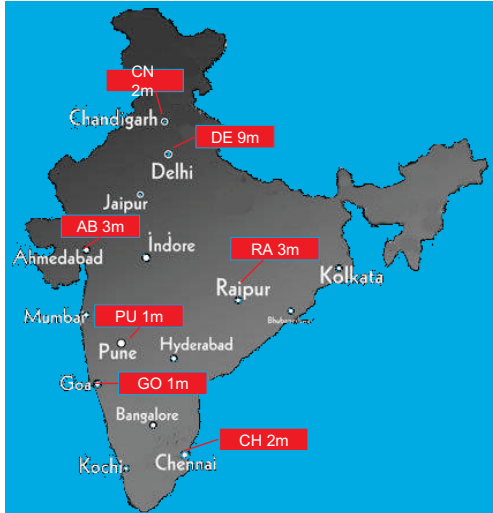


- we got full row data **IN PU 1m 1m** for all cities

Map – Reduce Pattern

Example:

Step 2 – we reduce country & male count data

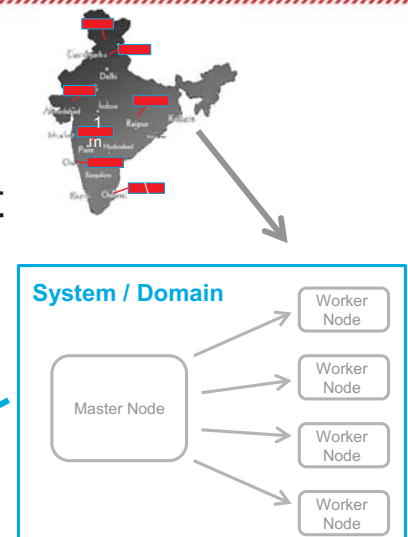
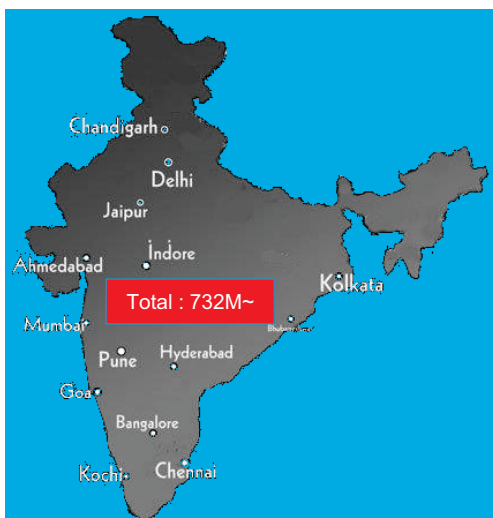


- now, a new lighter structure is formed: **IN PU 1m**
we hold only the data we need

Map – Reduce Pattern

Example:

Step 3 – we calculate the final result
result



- now, it is easy to calculate the result

Map – Reduce Pattern

- 1 When working in large scales, data is structured into some convenient units
 - These units are called '**buckets**'
 - Buckets might be distributed across multiple machines (clustered)
 - The key here is how data is structured and stored (remember noSQL?)
- 2 Map operation works on each bucket concurrently
 - Maps data taken from memory / disk / other node
 - Note: calculation should take place on each node, not on the requesting machine
- 3 Reduce operation may work on each 'bucket result' result concurrently
 - Reduce may do some processing on each value or aggregate it for some general calculation
 - Reduce works on every element in each bucket result
- 4 Fetch / calculate / compose a result

31

© All rights reserved to John Bryce Training LTD from Matrix group

NoSQL

- Not only SQL
- DB which is not a collection of related tables
- Mainly usage for
 - Big data
 - Real-time web (where clients are synchronized with server state)
 - Like Facebook's newsfeed & Twitter
- Usually – faster DB with cloud support
- 'Not Only SQL' – means SQL might be supported

32

© All rights reserved to John Bryce Training LTD from Matrix group

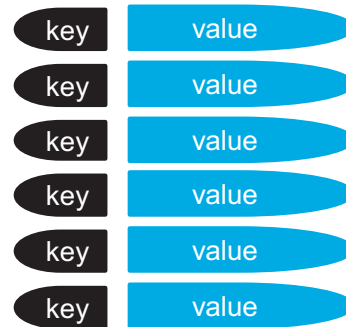
- A.C.I.D – Is not fully implemented in most NoSQL DBs
 - (Atomicity, Consistency, Isolation, Durability)
- Implementation might be
 - Eventually consistent – optimistic locking
 - Immediately consistent – pessimistic locking
 - Ordered – sorted keys
 - RAM/Disk – cache / hard drive
 - Sharding – ability of storing data across multiple machines

NoSQL DB categories

- Key-Value
- Document
- Column
- Graph

Key-Value

- A Map structured DB
- Keys are unique
- Values can be anything



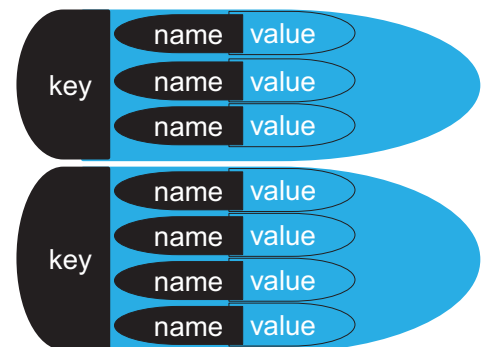
Document

- The document is the data
- The document has a specific format
 - XML/JSON/PROPERTIES
 - PDF/WORD/EXCEL....
- Documents might be organized as
 - Collections, tags, directory, any other meta-data



Column

- Like Key-Value BUT,
- Value is a set of columns
- Each column has a name and value
- Values structure may be different for each key

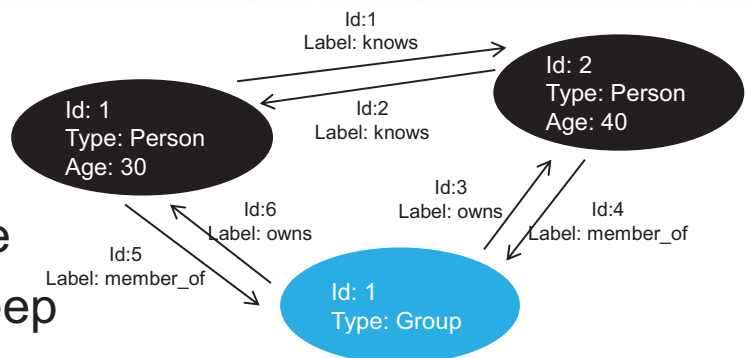


37

© All rights reserved to John Bryce Training LTD from Matrix group

Graph

- Referencing structure
- Can be shallow or deep
- Consist nodes, properties and edges
 - A node holds a direct reference to another – no indexes
 - Each node has properties
 - Edges describes the actual reference (lines between nodes)



38

© All rights reserved to John Bryce Training LTD from Matrix group

Open source document NoSQL DB



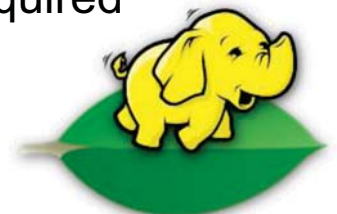
- MongoDB Data Model
 - Single deployment can hold multiple databases
 - A database holds a set of collections
 - Each collection gathers a set of documents
 - Each document
 - Holds the actual data in key-value form
 - Has a dynamic schema
 - Means that documents may vary in their structure even if they belong to the same collection

- Document format is JSON
 - But are stored as BSON (binary representation of JSON)
- Index support
 - MongoDB maintains effective indexes
 - Indexes are used for efficient queries
 - MongoDB doesn't have to scan all documents
 - Index might specify whether a document is relevant for a specific query or not...
- Sharding support
- Rich document-based queries

- Built-in Map-Reduce operations
 - Queries can be assigned with
 - Query string
 - Map function
 - Reduce function
 - Output form
- Grid File System
 - Files of all sizes can be stored
 - Files are automatically split to disks when needed

- Default locking
 - Reader-writer lock which is 'writer greedy'
 - Means that –
 - Readers may share the same lock
 - Writer forces an exclusive lock (locks all other readers & writers)
 - When there are readers and writers in the locking queue – writers comes first

- Mongo in the cloud
 - Support for master and slave nodes
- MongoDB and Hadoop
 - MongoDB can be used as input source and output destination for Hadoop
 - Means that Java developers can use Hadoop map-reduce pattern to integrate real-time data from MongoDB
 - A MongoDB connector for Hadoop is required



- Installing
 - Download mongodb-win32-x86_64plus-XXX.zip
 - Extract to and directory
- Starting MongoDB server
 - Run \MongoDB_HOME\bin\mongod
 - Server uses port 27017 by default