**Part 2**

**Application of NoSQL Database in Web Crawling**

**Principles of Web Crawling**

Web crawling is a process that automatically obtains information from the Web pages through the link relationships between them. It consists of a spider, controller, and the original page library.

**MongoDB:**

MongoDB is one of the most popular NoSQL database, whose main objective is to bridge the gap between key-value stores with high performance and scalability and traditional RDBMS with rich management, and take the advantages of both in one. It provides auto-sharding to achieve mass data storage and supports full indexes. With the powerful query language syntax like the object-oriented language, MongoDB can achieve the most function of single-table query in relational database. It also supports atomic in-place update and two replication mechanisms of Master/Slave and replica set.

**Key comparison points in favor of MongoDB**:

1. Data Structure - Schema-free

2. Querying - nested querying

3. Scalability - auto-sharding of MongoDB can provide easy horizontal scalability with low cost

**Comparing NoSQL MongoDB to an SQL DB**

A performance comparison between SQL server & MongoDB is done on a "not so big" dataset. Results show that MongoDB performs equally as well or better than the relational database, except when aggregate functions are utilized. Apart from that, discussions surrounding the nature data validates the need for either MongoDB or SQL server, i.e. If the database is non-structured and extremely large, then a NoSQL database is a good choice. The typical features of SQL databases, such as the ACID properties, require a certain amount of overhead, and are relaxed or eliminated in NoSQL databases to maximize performance. Many NoSQL databases organize the data into key-value pairs. Finally, after performing the experiments we concluded that MongoDB has better runtime performance for inserts, updates and simple queries. SQL performed better when updating and querying non-key attributes, as well as for aggregate queries. MongoDB could be a good solution for larger data sets in which the schema is constantly changing or in the case that queries performed will be less complex.

**Data Aggregation System**

It’s a heterogeneous ecosystem designed on top of the existing CMS data services for simplifying data look-up for end-user. Below are few of individual concepts in relation to DAS,

**Architecture**:

The DAS consists of such as web server, cache server, analytics, data services and MongoDB (cache and merge). Web servers handle user sessions and query are passed to the cache server for processing. The MD5 hash of the input query is used to identify the correct output. The cache server consists of pool of worker threads which handle the DAS queries. Relevant services are employed to fetch, transform, and insert data into raw cache. All data sharing common keys is merged into a single document and stored in merge cache.

**DAS Query Language:**

Uses a custom text-based query language with a PLY parser for tokenizing & lexical analysis. The queries consists of conditions, filters & aggregators OR map-reduce. The conditions consist of one or more DAS keys. Filters are commands that eliminate or prune data members from a record(s). Aggregators are used for summarizing filtered contents.

**Services**

Services are used by DAS which are accessed over HTTP, returning JSON or XML response. Execution of an query is dependent on identifying the relevant services needed. A mapping document helps us identifying the APIs.

**Caching & Merging**

Records are stored in either raw or merged cache collections. Intermediate response data from APIs & metadata of a query is stored in raw cache, whereas the merged output of a query is stored in merge cache. Larger merged documents are stored in GridFS instead of MongoDB due to size constraints. A reference to these documents is stored in

**Analytics**

Provides critical information to developers to perform cache optimization to reduce latency. The system is divided into query analysis and cache populators. Both of these components enables us to preserve frequently used queries in cache memory.

**Part 3**

1. Create a database for Contact Management System in MongoDB

> use contact\_management\_system

Graphical user interface, application

Description automatically generated

2. Create 5 records with different attributes and values you choose

var doc1 = {

fname: "Aniruddha",

lname: "Tambe",

phone: ["8573768193","8879441961"],

address: {

street\_name: "146 Boylston Street",

unit\_num: "#3L",

city: "Boston",

state: "MA",

zip: "02130"

},

email: "tambe.ani@northeastern.edu"

}

var doc2 = {

fname: "Siddhant",

lname: "Kohli",

phone: "7814989493",

address: {

street\_name: "77 Russel St",

unit\_num: "Apt 2",

city: "Malden",

state: "MA",

zip: "02148"

},

email: "kohli.si@northeastern.edu"

}

var doc3 = {

fname: "Priyanka",

lname: "Tambe",

phone: "8578003604",

address: {

street\_name: "",

city: "Mumbai",

state: "Maharashtra"

},

email: "priyankatambe910@gmail.com"

}

var doc4 = {

fname: "Foram",

lname: "Bhatt",

phone: "4706840790",

address: {

street\_name: "2 Wigglesworth street",

city: "Boston",

state: "MA"

}

}

// Invalid record

var doc5 = {

fname: "liahdiashdpiahdpe2131231",

lname: ";'aosjdpajdpajdpakdna",

phone: "alkdnaokbnoanpoakmdad",

address: {

street\_name: ";klsdhavojsdnlcnjslknsd",

city: ";sdofdjp[adad",

state: "fn;akdn;adma"

}

}

Graphical user interface, application

Description automatically generated

> db.contacts.insertMany([doc1,doc2,doc3,doc4,doc5]);

Graphical user interface, text

Description automatically generated

3. Deleting a record

Deleting the invalid record based on “no first name should contain numbers”

db.contacts.deleteOne({fname:{$regex:"[0213456789]"}})

Graphical user interface, application

Description automatically generated

4. Update one record

Let us update an email-id of a record,

db.contacts.updateOne(

{email: "tambe.ani@northeastern.edu"},

{

$set:{

email:"tambe.aniruddha3110@gmail.com"

}

}

);

Graphical user interface, application

Description automatically generated

**Part 4**

1. Create a collection called as games

use content\_management\_system;

db.createCollection("games");

Graphical user interface, text, application

Description automatically generated

2. Create 5 games

var game1 = {

name: "Prince of Persia",

genre: "Action Adventure",

rating: 91

}

var game2 = {

name: "GTA Vice city",

genre: ["Open world","Crime","Action Adventure","Racing"],

rating: 96

}

var game3 = {

name: "Need for Speed: Most Wanted",

genre: ["Racing","Role-playing"],

rating: 89

}

var game4 = {

name: "Assasins Creed",

genre: ["Open world","Action Adventure","RPG"],

rating: 96

}

var game5 = {

name: "Counter Strike: Global Offensive",

genre: ["Action","Competitive Gaming"],

rating: 98

}

show collections

db.games.insertMany([game1,game2,game3,game4,game5])

Graphical user interface, text

Description automatically generated

3. Find all games & one game

db.games.find({});

db.games.find({name: “Prince of Persia”});

dg.games.findOne({});

Graphical user interface, application

Description automatically generated

5. Write a query that returns the 3 highest rated games.

Graphical user interface, text, application

Description automatically generated

6. Update two favourite games

db.games.update({name: "Need for Speed: Most Wanted"},{$set:{achievement:["Game Master","Speed Demon"]}});

var toUpdate = db.games.find({name: "Assasins Creed"});

toUpdate[0]["achievements"] = ["Game Master","Speed Demon"];

db.games.save(toUpdate[0]);

Graphical user interface, text, application

Description automatically generated

A screenshot of a computer

Description automatically generated

7. Write a query to find records with both achievements

db.games.find({achievements:{$all: ["Game Master","Speed Demon"]}});

Text

Description automatically generated

8. Write a query that returns only games that have achievements

db.games.find({achievements: { $exists: true}});

Graphical user interface, text, application

Description automatically generated

**Part 5**

Execute 5 commands of your choice from each of the following groups, and paste the screenshots in a word document.  
1. mongo> help             [5 commands]

Text

Description automatically generated

2. mongo > db.help()

a. db.stats()

A screenshot of a computer

Description automatically generated

b. db.serverStatus()

A computer screen capture

Description automatically generated with medium confidence

c. db.getName()

A screenshot of a computer

Description automatically generated with medium confidence

d. db.hostInfo()

A screenshot of a computer

Description automatically generated with medium confidence

e. db.hello()

A computer screen capture

Description automatically generated with medium confidence

3. db.mycoll.help()

a. db.mycoll.find().help()

A screenshot of a computer

Description automatically generated with medium confidence

b. db.games.dataSize()

A screenshot of a computer

Description automatically generated

c. db.games.explain().help()

A screenshot of a computer

Description automatically generated with medium confidence

d. db.games.getIndexes()

A screenshot of a computer

Description automatically generated

e. db.games.totalSize()

A screenshot of a computer

Description automatically generated