# Large-Scale and Multi-Structured Databases

## ***Social News***

Biondi Matteo
Burgisi Martina
Cristofani Federico

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB
Innovation for industry 4.0

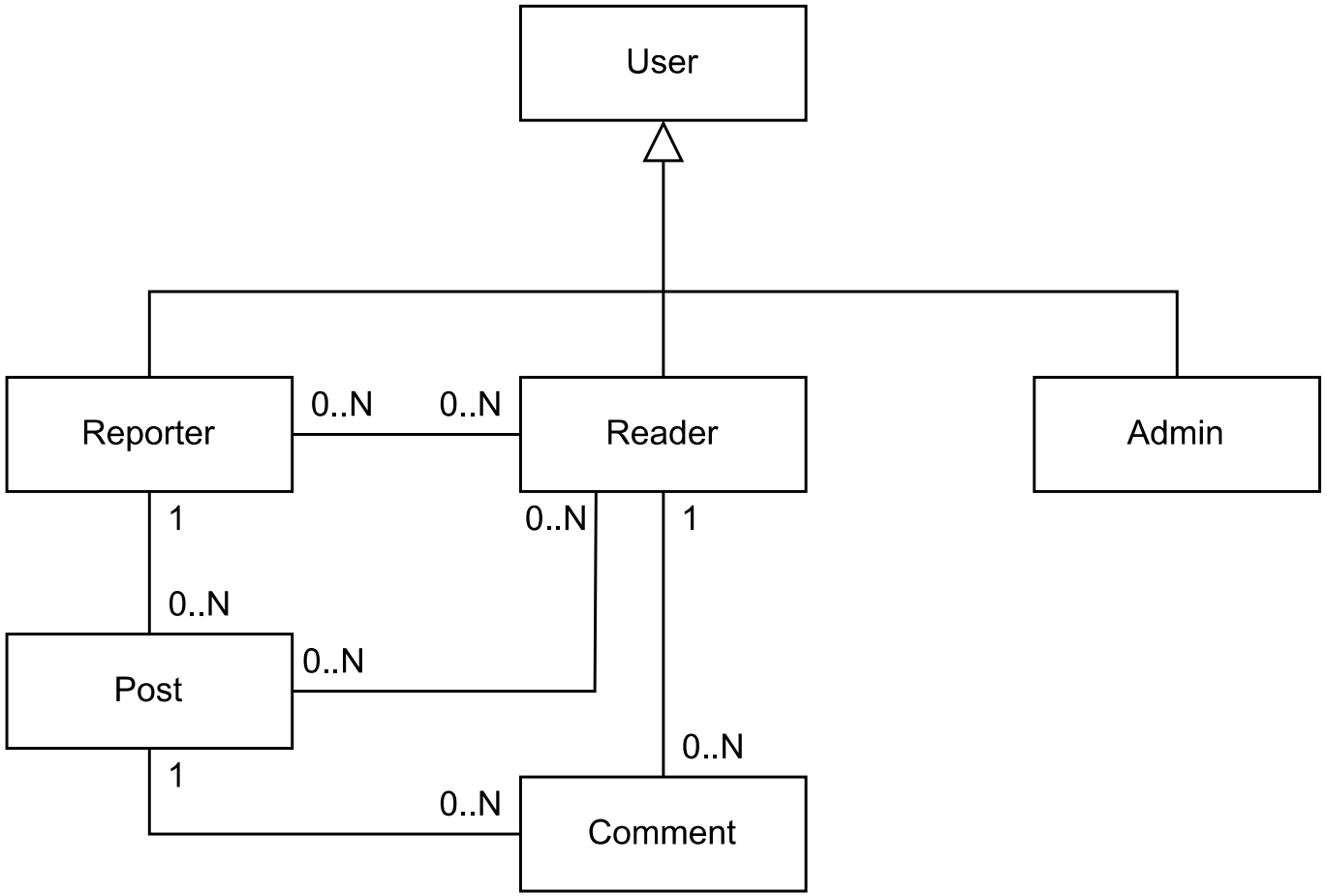
# Application: Idea Introduction

**Objective:** Design and implementation of a WebApp named “SocialNews”. The application fosters interaction among readers, reporters, and administrators, providing relevant information and statistics for an enhanced user experience.

Types of **users**: readers, reporters, and administrators.

- **Readers can** search for reporters or posts, view and comment on posts, delete a comment, follow reporters, view the most influential and popular reporters and view their own profile.
- **Reporters can** post and delete articles, view statistics about most active time of the day and their own most popular posts and view their own profile.
- **Administrators can** view statistics about readers and reporters, register new verified reporters, search and delete registered users and comments/posts.

# UML analysis class diagram



# Dataset Description

***Source:***

- https://github.com/jbencina/facebook-news
- https://www.kaggle.com/datasets/bwandowando/breaking-news-twitter-dataset
- https://randomuser.me/

***Description:***

- Facebook posts and comments on news outlets pages
- Breaking news tweets
- Randomly generated users

***Volume:***

- Post\Tweet and comments after being cleaned at least 104MB
- Generated users 150.000, estimated considering number of real authors of comments and posts

***Variety***:

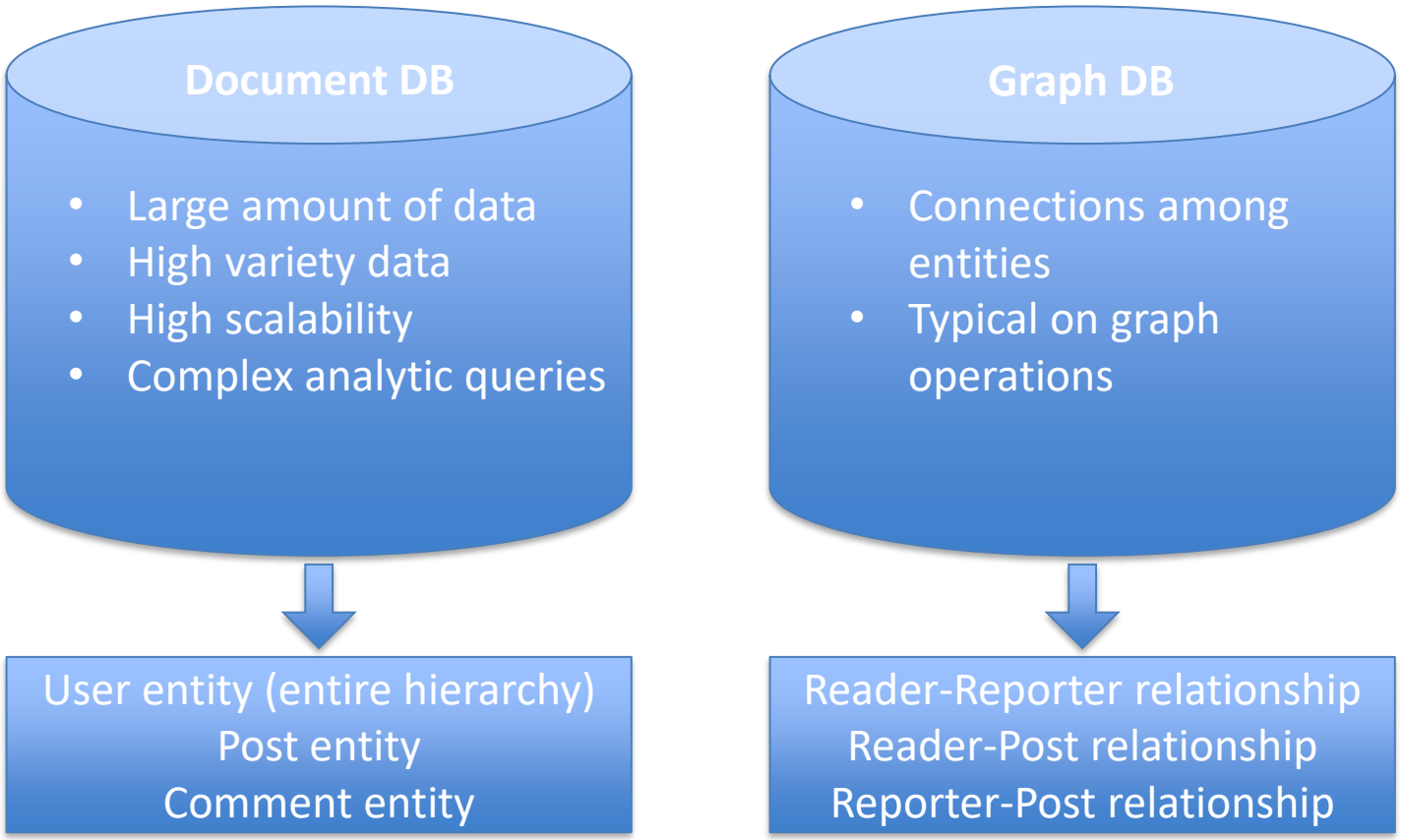
- Data comes from two different sources (Twitter and Facebook)
- Some data attributes are optional

# Non-functional requirements

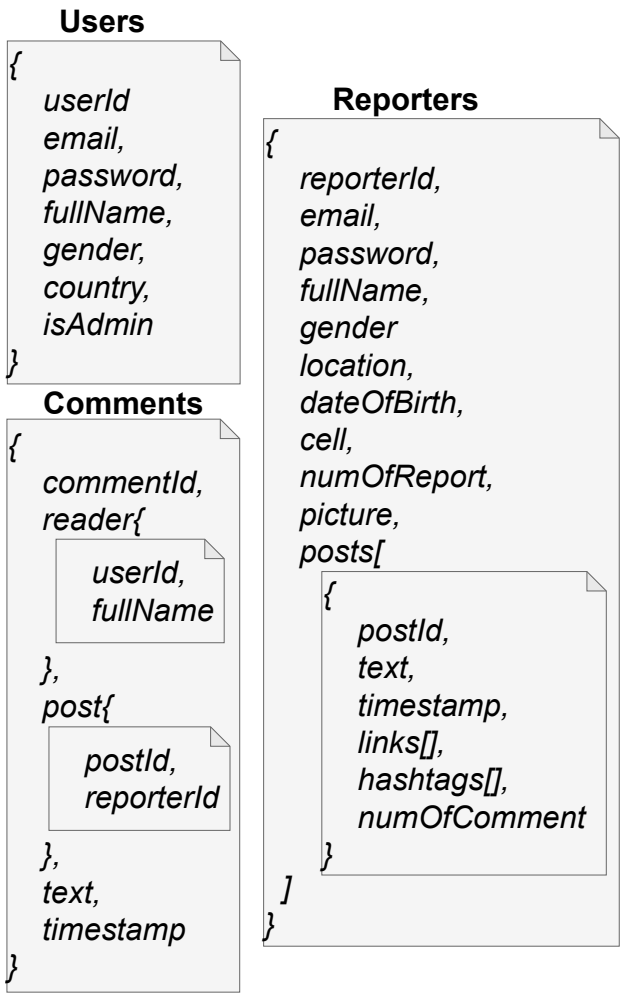
- The system must be a website application
- The system must encrypt users' passwords
- The system must communicate with any user by using a secure communication channel
- The system must be developed by using an OOP language
- The system will be available 24 * 7
- The system will ensure eventual consistency
- The system must be intuitive to use for users without any specific training
- The system must be able to run on at least the following main browsers: Google Chrome, Mozilla Firefox and Edge

# Database design

The system relies on two types of database, each storing the data that present the properties to obtain the maximum advantage from the selected database architecture:



# Document DB: Design




The entities are stored in the document DB inside three collections:

- **Users**, maintains both readers and admins information
- **Reporters**, maintains both reporter's personal info and his published posts as embedded documents
- **Comments**, maintains all comments publish by readers, keeping a reference to original post via document linking and to reader via data duplication

The design of the "reporters" collection includes two redundancies:

- **numOfReport**, overall number of reports on posts published by the reporter
- **numOfComment**, number of comments on the post

# Document DB: Main queries

1. ***Statistic:*** Most active readers
   ***Side:*** Admin
   ***Description:*** The statistic shows the first 10 active readers.
2. ***Statistic:*** Gender statistic
   ***Side:*** Admin
   ***Description:*** The statistic shows a graph about the numbers of women and men readers.
3. ***Statistic:*** Nationality statistic
   ***Side:*** Admin
   ***Description:*** The statistic shows a graph about the nationalities of the readers.
4. ***Statistic:*** Hot posts in the period
   ***Side:*** Reporter
   ***Description:*** The statistic shows the 10 hottest posts (of a certain period of time) of the reporter.
5. ***Statistic:*** Most active moment of the day
   ***Side:*** Reporter
   ***Description:*** The statistic shows the most active time moment of the day

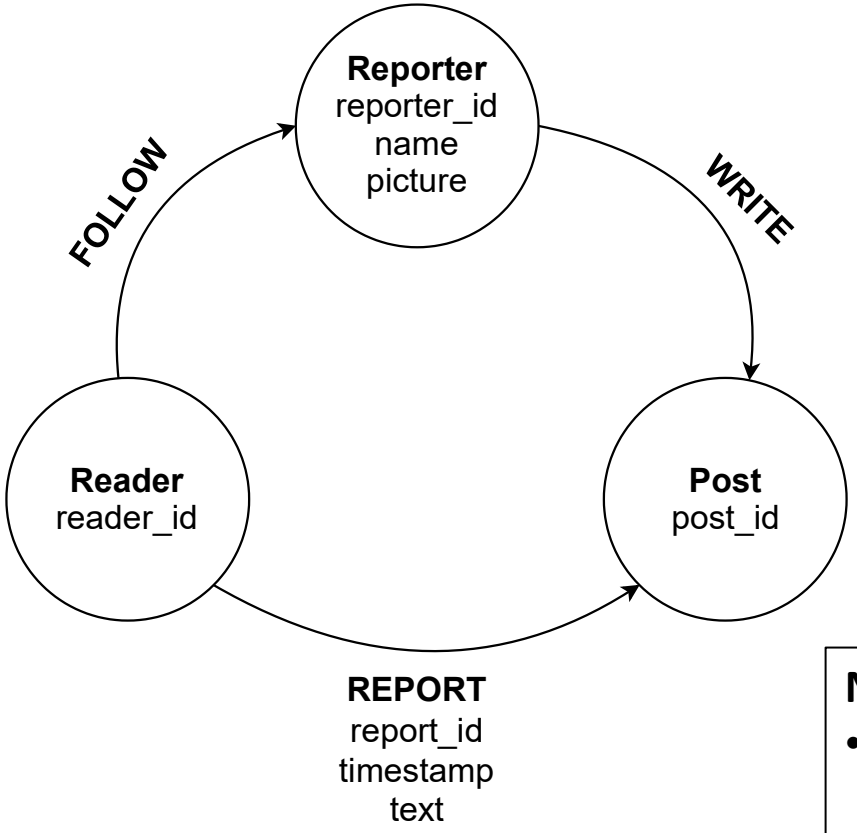
# Document DB: indexes

A read-heavy system can obtain considerable advantages from indexes design:

- Based on implemented base queries, the ones most frequent
- Takes into account the pagination implemented by queries that would otherwise retrieve too many data

| Collection | Index name | Field(s) | Type(s) |
|---|---|---|---|
| Users | sortByFullNamePagination | fullName, _id | Compound |
| Users | emailUnique | email | Single, unique |
| Reporters | searchByFullNamePagination | fullName, reporterId | Compound |
| Reporters | filterByReporterId | reporterId | Single |
| Reporters | filterByPostHashtag | posts.hashtags | Single, sparse |
| Reporters | emailActiveReporterUnique | email | Single, unique, sparse |
| Comments | searchByPostSortByTimestampPagination | post.id, timestamp, _id | Compound |

# Graph DB: Design




**Types of edges:**

- "**FOLLOW**" → unidirectional relationship from a "Reader" to a followed "Reporter"
- "**REPORT**" → unidirectional relationship from "Reader" to reported "Post"
- "**WRITE**" → unidirectional relationship from "Reporter" to their own "Post"

**Notes:**

- "reporter_id", "reader_id", "post_id" come from Document DB
- Reporter nodes contain name and picture to avoid double query (Document and Graph DBs) to take these information

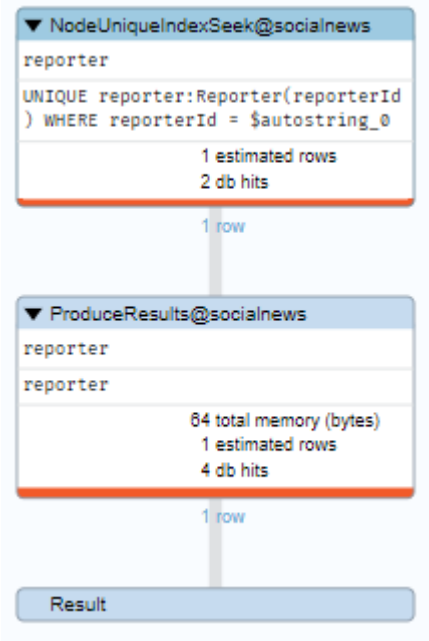
# Graph DB: Main queries

| Domain-Specific | Graph-Centric |
|---|---|
| A reader user reports a post | Add an edge directed from a “Reader” node to a “Post” node |
| How many followers does a specific reporter have? | How many “FOLLOW” edges are incident to a specific reporter vertex? |
| Which reporters are suggested to a specific reader? | Reporter vertexes with highest number of incoming edges that haven’t yet a direct edge with the reader (for which the statistic is computed) |
| Take reports associated with a specific reporter | “REPORT” edges to “Post” nodes linked to the current reporter node with “WRITE” relationship |
| Which reporters are the most popular? | Reporter vertexes with highest number of "FOLLOW” edges |
| Remove a reporter account | Remove the “Reporter” node, all “Post” nodes linked with “WRITE” relationship and all ingoing edges (to deleted posts and reporter) |

# Graph DB: Indexes and contraints

**No indexes**


**With indexes**



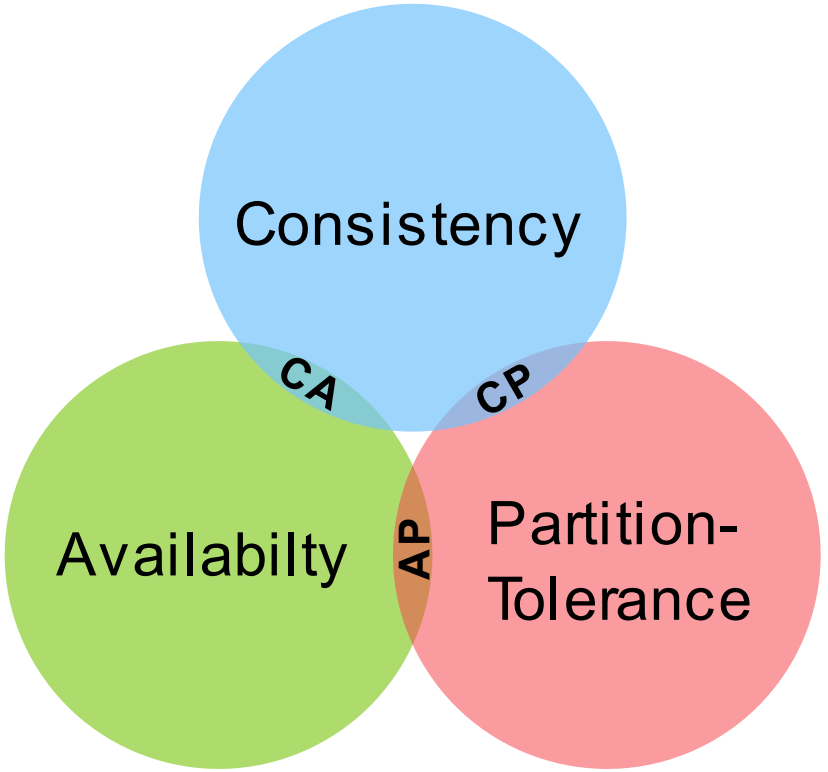

**Indexes e constraints**:

- Default token lookup indexes
- Range index on **report_id** (no associated constraint)
- Range index on **reporter_id** (associated with uniqueness constraint)
- Range index on ***post_id*** (associated with uniqueness constraint)
- Range index on ***reader_id*** (associated with uniqueness constraint)

Limitations due to Neo4J version and edition

# CAP theorem side

- MongoDB cluster in the default configuration lies on CP side
- Configuration to speed up read operations move towards availability sacrificing a part of consistency
- The speed up is obtained setting read preferences as “nearest”
- Reading from nearest replica may return stale data due to the asynchronous replication mechanism



# Data consistency

**Redundancies:**

- *numOfComment* to avoid join operation between different collections
- *numOfReport* to avoid loading data from different databases for a single page

**Drawback:**

Redundancies have to be updated for each comment/report insertion/deletion

**Implemented Mechanism:**

The mechanism is based on separate log files (for comment and report) and a periodic task to update redundant values

**How it works:**

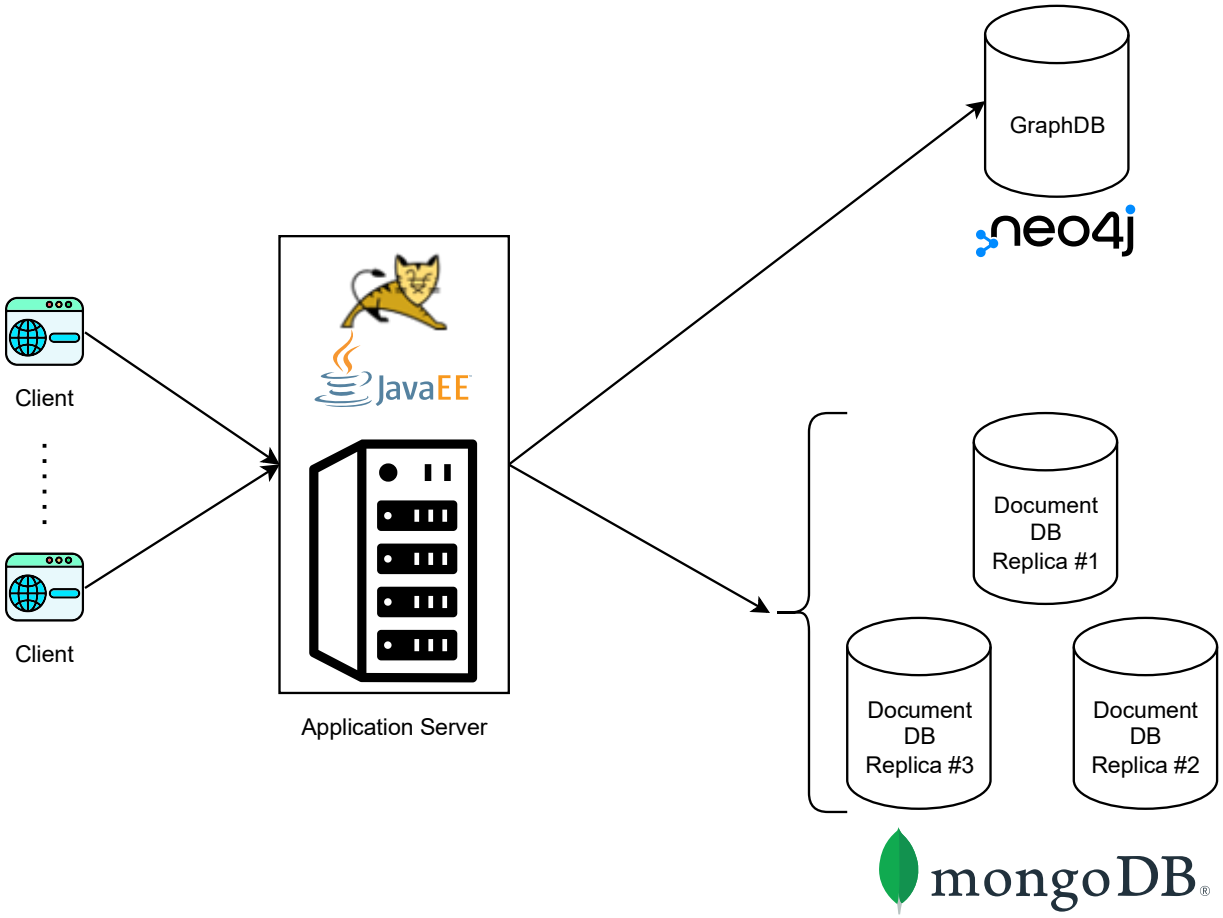
1. Operations performed are saved on separate log files (finer-grained synchronization and thread-safe file access)
2. Periodically, the log files are read, and their contents are extracted.
3. Operations must be summarised to reduce the number of operations in the database.
4. For each report and comment affected, the relative redundancy is updated
5. Unsuccessful update operations must be kept in the log in order to be retried in a subsequent attempt

# Sharding

- Sharding in MongoDB allows to adopt a horizontal scaling approach
- The shard key determines the distribution of data across the shards of the cluster and thus the achievable benefits for the system
- Shard key must be selected taking into account the implemented queries

| Collection | Field(s) | Brief explanation |
|---|---|---|
| **Reporters** | reporterId | Ensure that all the documents about the same reporter will be stored on the same shard |
| **Comments** | post._id | Ensure that all the documents about the comments on the same post will be store on the same shard |
| **Users** | _id | The "_id" field as shard key ensures a balanced distribution of the data |

# System architecture: Software



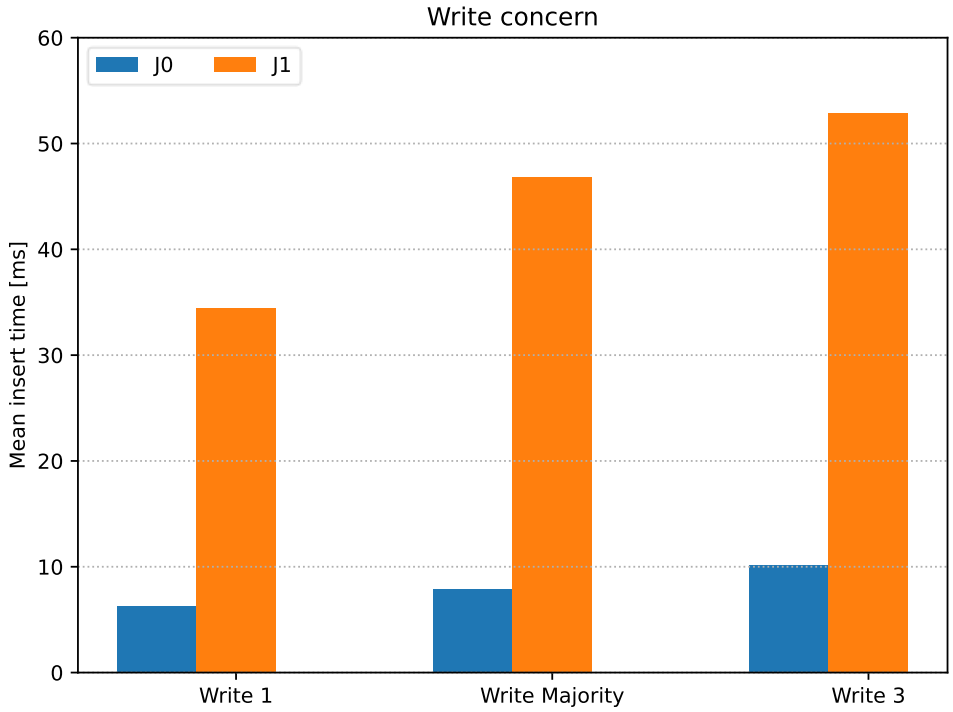
# System architecture: Hardware

| Component | Virtual Machine | Ip address |
|---|---|---|
| Apache Tomcat | Profile2022LARGE4 | 172.16.5.20 |
| Mongo instance #1 | Profile2022LARGE5 | 172.16.5.21 |
| Mongo instance #2 | Profile2022LARGE4 | 172.16.5.20 |
| Mongo instance #3 | Profile2022LARGE6 | 172.16.4.22 |
| Neo4J | Profile2022LARGE6 | 172.16.5.22 |

**Considerations**:

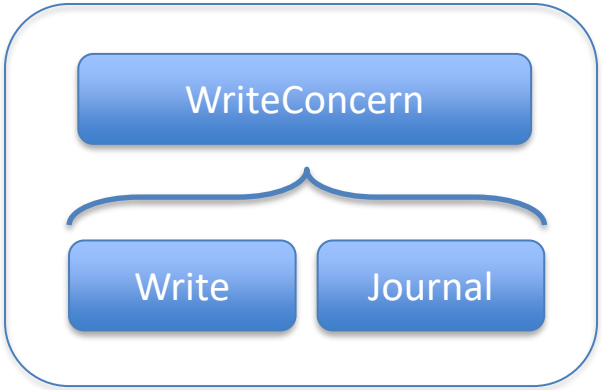
- The cluster available for the test deployment is a composed by three virtual machines, less than the number of application's components
- Mongo cluster has three replicas to be deployed in different machines
- Application server and Neo4J instance should be deployed in two different machines to guarantee the fairest load balancing
- Primary instance of the Mongo cluster should have an entire machine because it should handle a heavier load than the other replicas

# Final considerations and results




The mongoDB replica set con be configured to tune performance and consistency trade-off

***Write concern parameter***




- The write parameter determines the number of acks waited by the primary before send back the result
- The journal parameter determines if the operation is logged only in volatile memory or even on persistent storage