Citations Network Project
ADVANCED DATABASE SYSTEMS
ADB - IT502

Nguyen Hoang Minh - MITIU14005
Chiem Thach Phat - MITIU14006

Fall 2014

Instructor: Nguyen Hong Quang

1 General Terms

This section describes terms used in this study.
CINAS : Citations Network Analysis
GDB : Graph Database
OM : Object-Mapping
ORM : Object-Relational Mapping
ODM : Object-Document Mapping
OGM : Object-Graph Mapping
RDBMS : Relational Database Management System
RDB : Relational Database
USPTO : United States Patent and Trademark Office

2 Conceptual Model (Analysis Task)

We want to build a system that can store and analyze patents from USPTO as described in figure 1. A patent contains an unique Publication number, Publication type, Application number, Publication date, Filling date, Priority date, Fee status (Paid or Free), Title, Length of grant (years), Classification and some of its contents include Abstract, Images, Description and Claims. A patent may or may not cite any other patents; and it may or may not be cited by other patents. A patent must be invented by at least one Inventor, be applied by only one Applicant, be assigned for at least one Assignee and be examined by at least one Examiner. The system will identify Inventor, Applicant, Assignee and Examiner by four attributes, which are their first name, last name, city and country.

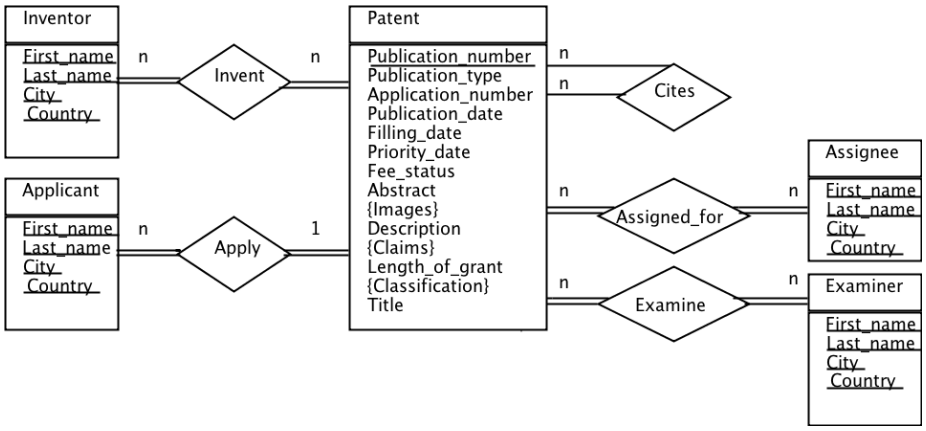

Figure 1: PatentDB Conceptual Model

Due to limited resources, instead of implementing the complete system, we chose the simple modelling to convey the operations of CINAS. The database is called PatentDB which has only two entities, inventor and patent. The relationship between entities and its attribute are illustrated in figure 2.

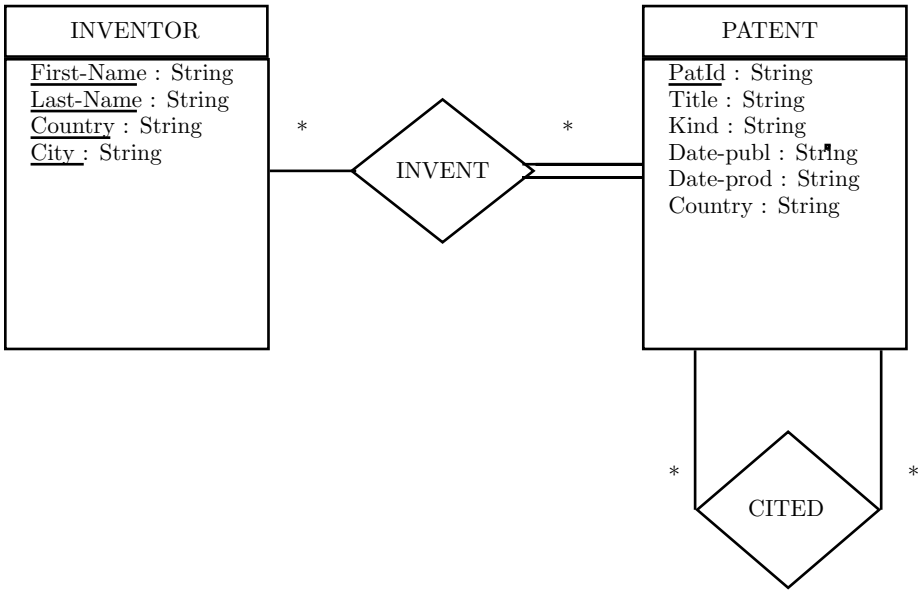

Figure 2: Demo Conceptual Model

Rules are simple as follow:

- An inventor may invent many patents and many inventors can work on the same one
- A patent might be cited by other patents or may refer to many patents

3 Logical Model (Design Task)

3.1 Mapping to Graph Logical Model (Neo4j)

In Neo4J, logical model can be represented by nodes with relationships as in figure 3.
The system has:

- Two types of Node: **Patent** and **Inventor**
- **Patent** has a relationship named **CITEDBY** with **other Patents**
- **Inventor** has a relationship **INVENT** with **Patents**
- Node also has **Properties**, such as **Inventor has first_name** and **Patent has publication_number**

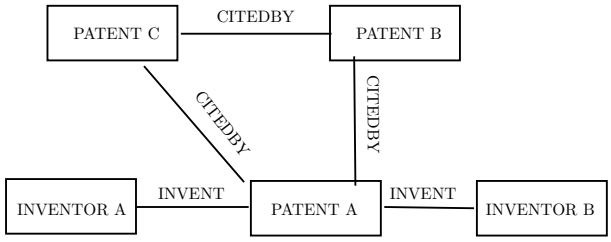

Figure 3: PatentDB Conceptual Model

3.2 Mapping to Document-oriented Logical Model (MongoDB)

With MongoDB, related data is embedded in a single structure or document. These schemas are generally known as "denormalized" models, and take advantage of MongoDB's rich documents. Consider the following schema:

```
patent_schema = {
    kind : string,
    date-published : string,
    number-of-claims : integer,
    country : string,
    patid : string,
    main-classification : string,
```

```
8       title : string,
9       app-number : string,
10      citations : [ { patid : string }, { patid : string }]
11      date-produced : string,
12      inventors:[
13        {city:string, last-name:string, country:string, first-name:
        string},
14        {city:string, last-name:string, country:string, first-name:
        string}
15       ]
16  }
17
18  inventor_schema={
19      city : string,
20      last-name : string,
21      country : string,
22      first-name : string
23  }
```

Listing 1: Document-oriented schema

As we design, the **patent_schema** is **embedded with** the **inventor_schema in inventors** key, as described from line 12 to 15 in listing 1.

In general, embedding provides better performance for read operations, as well as the ability to request and retrieve related data in a single database operation. Embedded data models make it possible to update related data in a single atomic write operation.

However, embedding related data in documents may lead to situations where documents grow after creation. Document growth can impact write performance and lead to data fragmentation. We would use reference for patent's citations. However, client-side applications must issue follow-up queries to resolve the references.

4 Data Storage

In this section, we will describe each component of CINAS in detail and also discuss design methodology. Fully written in Python, CINAS consists of 2 main modules:

1. Firstly, XML Parser Module **parses the input XML file into JSON format**.
2. DB Client would then **load JSON format files to populate it into Database**.

REST services are provided by external resources.

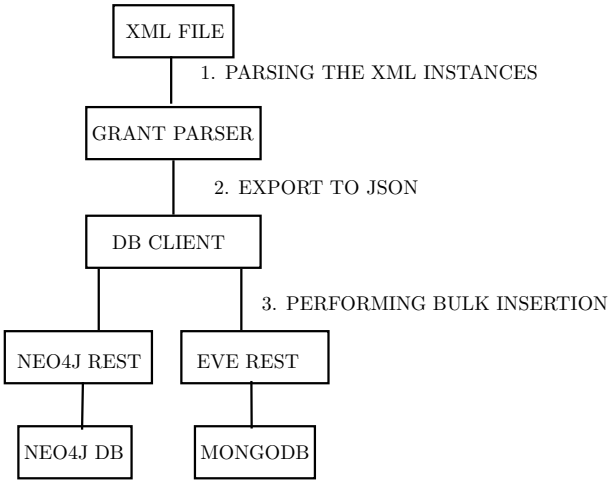

Figure 4: CINAS ARCHITECTURE

4.1 XML Parser Module

There are **two approaches in designing XML Parser**. Either one may choose **DOM** method, which load the entire file onto memory using ElementTree, or line by line processing by **SAX**. Even though the **former method is remarkably simpler and faster, it is susceptible to memory consumption**. As data grows big, implementing parser using DOM method is not plausible. Fortunately, **the latter is well suit for parsing big data by design and thus, will be implemented in CINAS**.

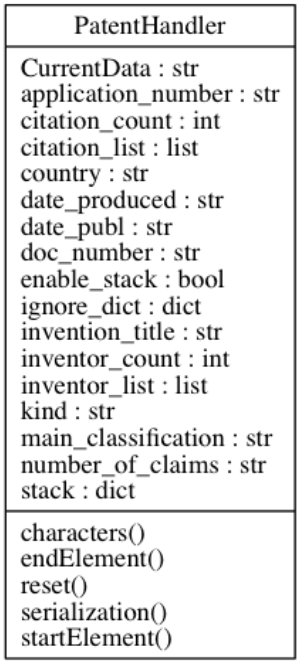

Figure 5: Patent Handler Class Diagram

The core of Grant Parser script is PatentHandler class, class diagram of it is shown in figure 5. It mainly used to hold data that parsed from each XML instance. Since SAX Parser only read XML line by line, it's reasonable to keep track of processing tag using stack. In brief, Grant Parser currently works following this procedure:

1. Load XML file and break it down into separated XML instances
2. Parse each XML instance into patent handler. For each tag with nested tags, enable stack to store adjacent data.
3. Tags that are explicitly defines in ignore list will be left out during parsing. Upon completion, reset the patent handler for new instance
4. Finally, an option to export the entire file to JSON format is available if chosen.

An implementation using DOM tree is also documented solely for benchmark's comparison and thus, detailed explanations are beyond the scope of this study.

4.2 DB Client Module

Instead of using OM for particular database, we chose to **deploy REST services as mean of communication with our backends**. It is complicated

to wrap OM around another OM, but it is simple to wrap data around HTTP response and request before forwarding. Therefore this approach provides more scalable solution than OM. Despite the fact that **this approach might limit the performance speed, it is worth a trade-off for being more extendable**. The figure 6 illustrates class diagram of our DB Client module.

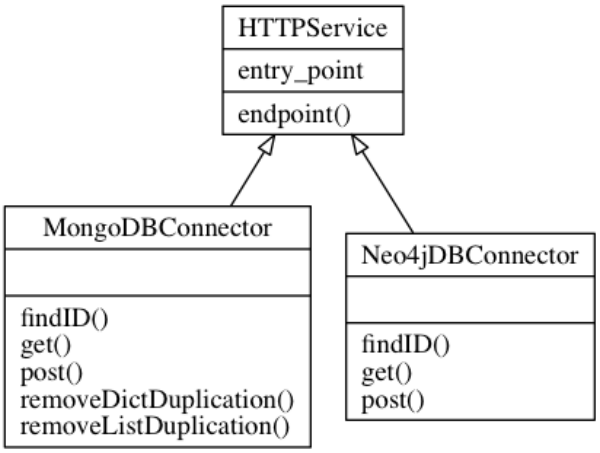

Figure 6: DB Client Class Diagram

The idea is to create an abstract layer to hide the presence of multi-databases systems. Instead of choosing ODM for MongoDB or OGM for Neo4J, we created an data layers to a enhance the transparency. For example, to issue a POST request to whatever the backend is, developers currently only need to concern the differences between resources and query format (commonly JSON). The complexity underlying will be handled by each corresponding connectors. As can be seen from figure 6, **all the fundamental HTTP methods are implemented in HTTPService**. MongoDBConnector and Neo4jDBConnector inherit private methods from its parent.

It's important to note that we need to handle with cares for MongoDB as it's susceptible to data redundancy and inconsistency due to being no schema DB. In addition, MongoDB requires an object ID to make a legitimate references. In specific, for a patent to make references, it should store the objectID of patent instead of proposed patent ID. From our opinions, we think this is unnecessary and generate extra workload for conversions. Hence, we **use embedded to represent the relationships between patent**.

Regarding the duplication in MongoDB, considering this case where patent 08852195 actually refer to WO2005/069809 twice at line 329 and line 330[1], it's inevitable that we might have data redundancies to some extents. Even though it's more acceptable to treat them as one entity, we would consider it in our next stage of implementation due to time constraint.

[1] http://www.directorypatent.com/US/08852195.html

The procedure of **POST in MongoDB** is:

1. Load all data from JSON file
2. Construct list of citations from group of patent
3. Validate uniqueness within the list. When we insert using bulk insertion, 3 patents might references the same patent and this will culminate in data duplication in DB. We can not overlook this problem.
4. Appending statements into query. Then, perform 1st POST these citations
5. If the uniqueness is violated, the response code from EVE server will determine the next step. If error response with duplicates found, we remove it from the list and resend it again.
6. Finally, we perform an overwrite PUT for any individual patent. This step must be linearly execute to guarantee that all the dummy patents, which are created using citations, will be updated with information.

The procedure of **POST in Neo4J is much more simpler**, since Neo4J has an overwrite operation called "MERGE",which will create or merge properties to existing node, and allow developers to execute it through Cypher endpoint. Therefore, it reduces the need for extra validation:

1. Load all data from JSON file
2. Construct list of citations from group of patent
3. Appending statements into query. Then, perform POST these citations
4. Finally, we perform another POST for list of patents.

5 Results (End-User Functionalities Task)

Contemporarily, CINAS has processed up to 110343 so-called patents. Table 1 shows the summarized statistic of patent database.

	Number of Documents
Duplicated Citations	347
Patents	3000
Citations	106996
Total	110343

Table 1: Summarize Table

Class	Descriptions
D99	MISCELLANEOUS
D34	MATERIAL OR ARTICLE HANDLING EQUIPMENT
D32	WASHING, CLEANING, OR DRYING MACHINE
D30	ANIMAL HUSBANDRY
D29	EQUIPMENT FOR SAFETY, PROTECTION, AND RESCUE (1)
D28	COSMETIC PRODUCTS AND TOILET ARTICLES
D26	LIGHTING
D25	BUILDING UNITS AND CONSTRUCTION ELEMENTS
D24	MEDICAL AND LABORATORY EQUIPMENT
D23	ENVIRONMENTAL HEATING AND COOLING; FLUID HANDLING AND SANITARY EQUIPMENT
D21	GAMES, TOYS, AND SPORTS EQUIPMENT
D20	SALES AND ADVERTISING EQUIPMENT
D19	OFFICE SUPPLIES, ARTISTS AND TEACHERS MATERIALS
D17	MUSICAL INSTRUMENTS
D16	PHOTOGRAPHY AND OPTICAL EQUIPMENT
D15	MACHINES NOT ELSEWHERE SPECIFIED
D14	RECORDING, COMMUNICATION, OR INFORMATION RETRIEVAL EQUIPMENT
D13	EQUIPMENT FOR PRODUCTION, DISTRIBUTION, OR TRANSFORMATION OF ENERGY
D12	TRANSPORTATION
D11	JEWELRY, SYMBOLIC INSIGNIA, AND ORNAMENTS
D10	MEASURING, TESTING OR SIGNALING INSTRUMENTS
D9	PACKAGES AND CONTAINERS FOR GOODS
D8	TOOLS AND HARDWARE
D7	EQUIPMENT FOR PREPARING OR SERVING FOOD OR DRINK NOT ELSEWHERE SPECIFIED
D6	FURNISHINGS
D5	TEXTILE OR PAPER YARD GOODS; SHEET MATERIAL
D4	BRUSHWARE
D3	TRAVEL GOODS AND PERSONAL BELONGINGS
D2	APPAREL AND HABERDASERY
D1	EDIBLE PRODUCTS

Table 2: USPTO - Classification Table

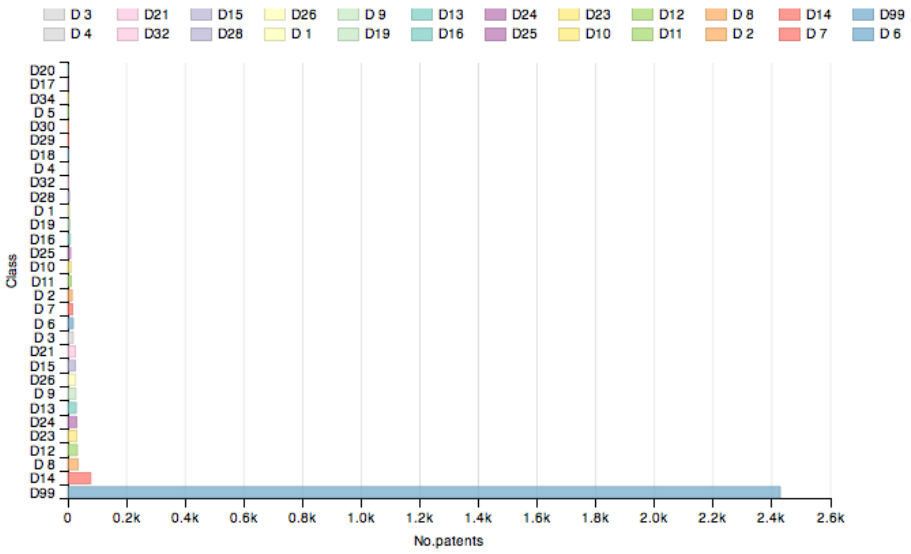

Figure 7: Patent Classification October 2014

From figure 7 and table 2, it is immediately striking that inventions for MISCELLANEOUS (D99) industry were blossomed in October 2014. In specific, nearly 2500 granted patents (accounted for nearly 83%) makes non-trivial goods production industry become the most popular topic among the invention community. In addition, this might signal a competitive field for any inventors that shall want to take part in. On the opposite, classification that are larger than 20 were recorded at lowest level, except for the case of MEDICAL AND LABORATORY EQUIPMENT (D24), ENVIRONMENTAL HEATING AND COOLING; FLUID HANDLING AND SANITARY EQUIPMENT (D23) and LIGHTING(26) where number of inventions are relatively close to those in range of 1x (approx. 100). In general, involving in these fields might be less competitive than the others. Even though coming in second place of the ladder, RECORDING, COMMUNICATION, OR INFORMATION RETRIEVAL EQUIPMENT's inventions (D14) are accounted for merely 6.66%, in other words, 15 times smaller than the dominant D99. Interestingly, this also indicates the need for more robust information retrieval system is significant and thus, was main motivation for this study. Following D14 closely was TOOLS AND HARDWARE (D8) which received 150 attentions. Finally, we also witness the insignificant presence of inventions in various class among 1x and 0x. As promised, statistics and bar chart was computed and generated using R.

5.1 Citations Network

We also demonstrate citations network on 500 patents instead of 3000 patents due to memory constrain on personal computer. A full view of network consists of inventors, patents, citations and their links are shown in figure 8.

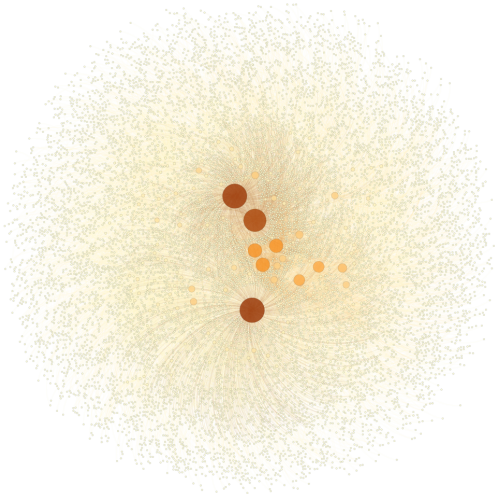

Figure 8: Patents and Citations October 2014

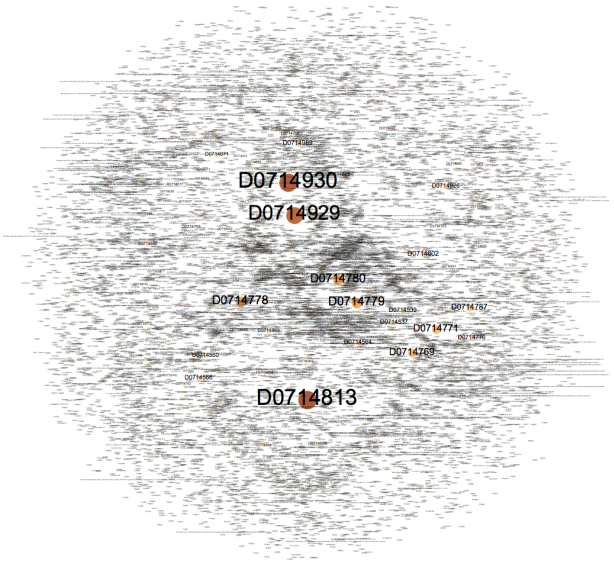

Figure 9: Citations Network October 2014

In this figure, using Gephi as visualization tool to enhance the spatialization within the network, we applied the force-directed layout Fruchterman Reingold to scatter out the node[2] and Force Atlas to distribute nodes without overlapping[3]. Specifically, Bigger nodes will be gather at the central of the graph while nodes with least connections are pushed to the rear area. A closer look at citations network can be observed from figure 9. In particular, we applied ranking by connected components to distinguish group of nodes which have largest number of connections.

Ranking by connected components gives us the importances of particular node since not only many inventors connect to it, but also do many inventions actual refer to it as a fundamental foundation.

ID	Title	No.citations	Class
D0714930	Reservoir.for.water.flosser	495	D24
D0714929	Base.for.water.flosser	457	D24
D0714813	Electronic.camera	517	D14

Table 3: Most connected components patents information

Table 3 shows the information regarding group of most-connected nodes from the graph. Unfortunately, All of them are recently published and thus, we can not observe the case where patent from the past was referred by many

[2]http://en.wikipedia.org/wiki/Forcedirected_graph_drawing
[3]http://webatlas.fr/tempshare/ForceAtlas2_Paper.pdf

patents from this figure. In figure 10, the edges of the network reflect patents or inventors that are connected to only one node of the graph. Following the connected-components ranking algorithm, we might deduce that they are lack of popularity and feasible knowledge.

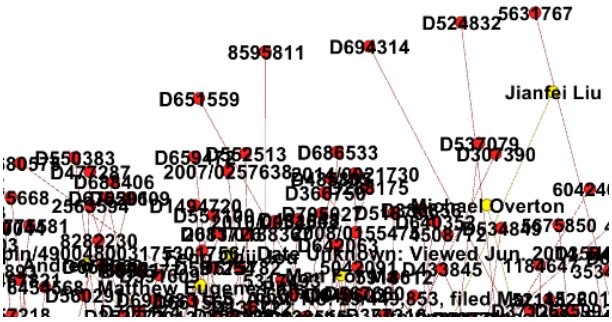

Figure 10: Citations Network Edge October 2014

5.2 Inventors Maps

To measure the activity of inventors community, we classify them based on their regions in figure 11. It's not surprising to witness the outstanding in the field of North America since it's their nation to begin with. However, it's interesting to see that US patents industry also attracts the numerous inventors from different parts of the world, including Asian and Australian. Besides, Russia, despite being second polar of the world in term of economy and currently has oil dispute with American, has no inventors take part in. Overall, the figure might indicate the inventing opportunities in US patent industry from the perspective of the outsiders.

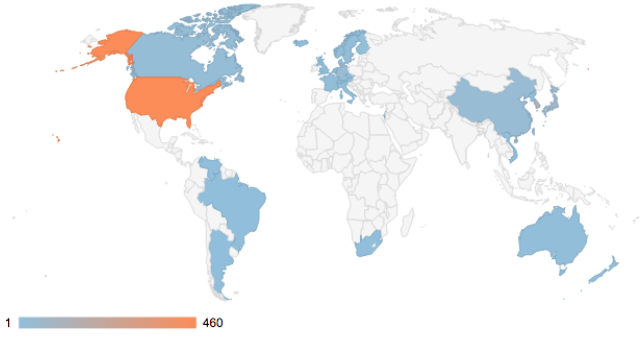

Figure 11: Geographic of inventors October 2014

5.3 Benchmark Neo4J and MongoDB

We perform the time measurements on 500 patents with total of 12016 citations. DOM and OGM are tested on personal computer that can actually handle the entire XML files of 500MB while the rests are tested on limited resources host. The following cases are taken into consideration:

- Parsing XML using DOM
- Parsing XML using SAX
- Insertion of Neo4J throught built-in REST server
- Insertion of MongoDB throught Eve REST server
- Insertion of Neo4J throught OGM

All the statistics are presented in table 8, its corresponding graph is figure 12. Overall, the trade-off between SAX and DOM, OGM and REST are clearly visualized. In particular, as SAX has to process data line by line, it's inevitably slower than DOM which load the whole file onto the memory. In addition, OGM is faster than REST in term of performance, but the simplicity of REST is worth the trade-off to develop more sustainable solutions. As can be seen from the graph, insertions with Neo4J are relatively time-consuming as nodes together with links need to be created. Futhermore, the graph is exponentially increased so Neo4J alone will not suitable solution for big data. On the opposite, MongoDB has higher write performance since everything is stored without constrains and relationships. The drawback of MongoDB is that end-client has to implement the references between each document, enforce data validations and thus, require more works load on the client side.

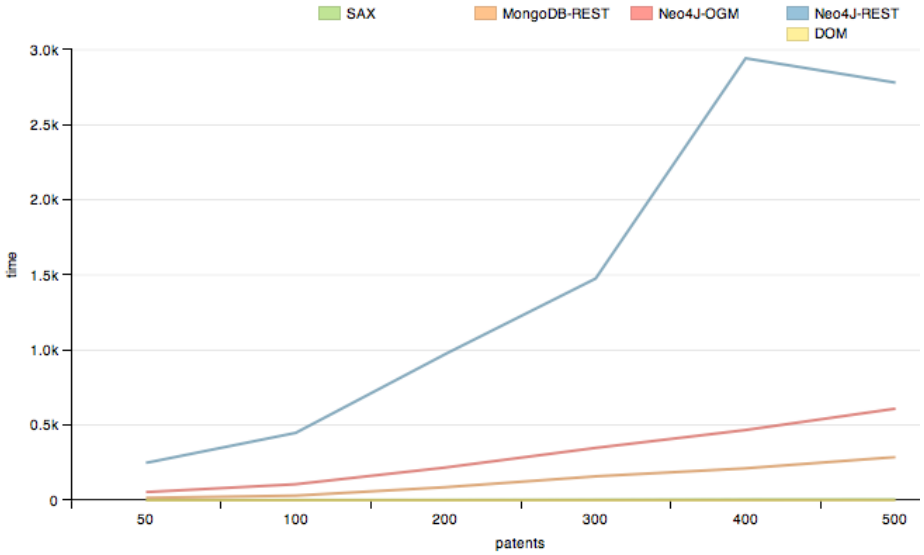

Figure 12: CINAS Benchmark

6 Lessons Learned

CINAS initially aims to provide more flexible and scalable solution for big patent data. Similar to other systems, the process of analyzing patents following CINAS are explained as follow:

- Extracting valuable information
- Inserting into database
- Exporting the information

It's important to note that CINAS is designed to be DBMS-independent. Specifically, the framework has special connectors working as drivers to different databases and thus, can fully take advantages of crossing-database's characteristics. For example, **to analyze a citation network in which contains nested relationships and thousand of nodes, Neo4J is far superior than RDBMS**. Nonetheless, **Neo4J is remarkably slow when it comes to big data** as it stores additional attributes in the relationships between nodes and presents more complicated relationships between each node in comparison to RDBMS. Another example is **to analyze text-based field where MongoDB would take primary place because it has no schema enforced on data**. No schema would also mean no joining cost in term of relational and hence, provide more speed.

In this study, we present the method of obtaining citation network using Neo4j as back-end. Furthermore, we also show how valuable data can be extracted from MongoDB. The dramatical trade-off of this approach is that it would cost more memories, load-balance to maintain two different DBMS system within one host and hence, reducing overall speed. Having thought of that, we implemented CINAS using RESTful APIs to partially address those issues, we also compare the pros and cons with the ORM approach for references in implementation section. Even though it cannot overcome the speed of using ORM with direct-connection to the storage, it's worth mentioned in term of scalability.

6.1 Neo4j

Graph Database differs from RDB in the sense that a node itself is a "thing", not a table that holds many records. This provides a level of flexibility to represent data closer to the real world entity. Considering the following example where GDB overwhelms our traditional RDB, that's in a citation networks a patent is cited by many patents. Traditionally, RDB would maintain only single table, named Patent, with each record has an additional column to store foreign key. However, the fact that many patents can cite a single patent and a patent can be referred by many patents would leave us an M-to-N relationship and require a creation of second table. Suppose citations are limited to 1000 patents, which means cited patents and patents are reachable within our database, and each patent can refer up to 1000 patents. Clearly, 1000000 records should be stored inside our second table. Therefore, representing such a nested relationship with tabular system is not a scalable practice since we have more than millions of patents. The above problem can be addressed with 1000 nodes and 2000 relationship in GDB, this is much similar to what we usually think.

An additional important feature in GDB is relationship's weight, this offers more accurate ways to determine the relationship between each entity.

6.2 MongoDB

MongoDB is an open-source document database, and the leading NoSQL database. MongoDB deploys BSON as an primary data representation and therefore improve speed of traverse through JSON file. BSON, short for Binary JSON, is the binary-encoded version of JSON-like documents. In other words, it adds extensions for data types such as integer or date that are not part of JSON specification. JSON is a self-describing, human readable data format. Originally designed for lightweight exchanges between browser and server, it has become widely accepted for many types of applications. JSON documents are particularly useful for data management for several reasons. A JSON document is composed of a set of fields which are themselves key-value pairs. This means each JSON document carries its own human readable schema design with it wherever it goes, allowing the documents to easily move between database

and client applications without losing their meaning. Detailed explanations for JSON and BSON will not be discussed in this study.

A record in MongoDB is a document, which is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents. The advantages of using documents are:

- Documents (i.e. objects) can be defined with native data types that can be seen in many programming languages. E.x: Date, String, Integer
- Embedded documents and arrays reduce need for expensive joins.
- Dynamic schema supports fluent polymorphism.

6.2.1 Downfall of Normalization

This example was taken from Brian's blog[4] to explain the drawbacks of normalization approach, we also change the data to patent data to illustrate similar problems can be faced in process of designing CINAS . The immediate and fundamental difference between MongoDB and an RDBMS is the underlying data model. A relational database structures data into tables and rows, while MongoDB structures data into collections of JSON documents. The ability to support arrays is an especially helpful feature. Consider the complexity of supporting a repeating group in a properly normalized table structure. To represent the same data object in a single table row would like like this:

PatentID	Title	Classification
1	"Snack"	"D1, D2, D3"

Table 5: Patent Table

Aggregation and updating specific classification for a comma separated list might pose serious problems and extra works. The first normalization was designed to avoid such problems by dividing each class into different records.

PatentID	Title	Classification
1	"Snack"	"D1"
1	"Snack"	"D2"
1	"Snack"	"D3"

Table 6: 1st Normalized Patent Table

We create a data redundancies that make us susceptible to anomalies. An example is to issue update command using SQL query :

[4]http://blog.mongodb.org/post/72874267152/transitioning-from-relational-databases-to-mongodb

```
UPDATE Patent

SET Title = "New Snack"

WHERE PatentID = 1

AND Classification = "D1"
```

Listing 2: Json Example

PatentID	Title	Classification
1	"New Snack"	"D1"
1	"Snack"	"D2"
1	"Snack"	"D3"

Table 7: 1st Normalized Patent Table

This happens because the table fail to qualify a 2nd Normalization Form. Classification is not functionally dependent on Title since there are three titles with different classes. Hence, we need to normalize data into 2 separated tables.

PatentID	Title
1	"Snack"

PatentID	Classification
1	"D1"
1	"D2"
1	"D3"

Table 8: 2nd Normalized Patent Tables

This structure prevents the anomaly, but introduces a new set of problems. The new schema has become much more complex, and we have lost any semantic understanding of stored objects. By adhering to these two normal forms, we've also entered the realm of cross-table joins and the need to enforce ACID transactions across multiple tables. The enforcement of referential integrity in multi-row, multi-table operations will require concurrency controls, increasing overhead and affecting performance.

MongoDB avoids these complications through the use of a document data model. The JSON document forms a more natural representation on the data than the normalized schema. It achieved this by allowing the embedding of related data via arrays and sub-documents within a single document - thus eliminating the need for JOINs, referential integrity and multi-record ACID transactions.

```
{
   {
   "_id" : ObjectId("528ba7691738025d11aab772"),
   "patid" : 1,
   "title": "Snack",
```

```
6     "classification" : ["D1", "D2", "D3"]
7     }
8 }
```

Listing 3: Json Example

6.3 OM and REST services

Object Mapping is an layer over database to assist querying and managing data using an object paradigm. The **pros** of this method is that it hides the database query language syntax away from logic code. Following this manner, data models are gathered in one place and it's easier to update, maintain and reuse the code. Every databases have its different mapping methodology. For example, relational database will need object-relational-mapping while object-document mapping serves as driver between web applications with document-oriented database. The **drawback** of using OM is that OM tools generally stem from the high level of abstraction obscuring what is actually happening in the implementation code. Also, heavy reliance on OM software has been reported as a major factor in producing poorly designed databases. Furthermore, as we stated CINAS is database-indepedent, it's better to exchange data over more popular format such as JSON instead of binding to a specific database's language. Last but not least, OM does not provide neither enhanced security services as it has its own connection to the database, nor load-balancing. This would leave more works on the database's side.

REST, for REpresentational State Transfer, is a simple stateless architecture generally running over HTTP. REST has become the norm in past decades as the majority of developers would prefer this scalable solution[5].

We agree the **boons of REST services** over our CINAS which are described comprehensively in Swaroop Chitlur's blog[6]:

- It's simpler in comparison to OM as it only uses GET, PUT, POST, DELETE and PATCH to communicate with the web services.
- Switching some data from Neo4J to MongoDB or sharding in MongoDB, etc. could be hidden as internals of this layer. This copes well with our requirements for database-independence.
- Providing middle layer would also enhance rate-limiting, monitoring and scalability - and Database's memory leaks, etc. would not affect other parts of the system.

Despite having favors for REST, the services are generally **suffer from performance** as it wraps data through an additional layer and also provide an abstraction over databases.

[5] http://www.theserverside.com/feature/Why-developers-demand-resource-based-RESTful-APIs

[6] http://www.swaroopch.com/2013/04/18/database-as-rest-api/

6.4 Gephi

Unfortunately, **Neo4j visual browser only supports up to 1000 rows**[7] and the **performance becomes inconsistent as data grows**, **we chose Gephi to alternatively visual our network**. Gephi is a collaborative open-source project. It has been built to be easily extended and reused. Like Adobe Photoshop but for data, the user interacts with the representation, manipulates the structures, shapes and colors to reveal hidden properties. The goal is to help data analysts to make hypothesis, intuitively discover patterns, isolate structure singularities or faults during data sourcing[8].

References

[1] *Transitioning from Relational Databases to MongoDB - Data Models* B. Reinero, `http://blog.mongodb.org/post/72874267152/transitioning-from-relational-databases-to-mongodb`, Jan 2014.

[2] *USPTO Patents Database Construction* Torvik, D. Dubin and Q.Liu, August 2012.

[3] *Construction of Japanese Patent Database for Research on Japanese patenting activities* A. Goto, K. Motohashi, last-access December 2014.

[4] *Guide Book for Practical Use of "Patent Map for Each Technology Field"*, Invention Research Institute, Japan Institute of Invention and Innovation, last-access December 2014.

[7] http://stackoverflow.com/questions/26031795/resultset-too-large-over-1000-rows-in-neo4j-browser

[8] http://gephi.github.io/features/