**Birds Migration Map**

**1. Description**

This application is concerning the bird migration phenomena. More precisely, this application will gather information from more resources and it’s going to analyze them.

During this analysis, on one side the processed data will be stored in a database, on the other side they will become useful in creating a map which is going illustrate the actual phenomena, while also indicating hotspots and offering additional information on them.

Last but not least, the previously stored data will be used to be able to observe migration tendencies for the study of bird migration phenomena.

**2. Application modules**

• **Collection Module:** this component will handle the collection of the data that will be used for the map. The data will be collected from the eBird API and from one social network, Twitter, making use of hashtags (e.g #birdmigration, #birdspotting) to discover relevant tweets that can be processed.

• **Processing Module:** the component where all data from the collection module will be processed. For the data coming from the eBird API it will identify the date, the species, the number of birds and location where the birds where seen. As for the data from the social network, first it will check if the tweets contain the necessary information for it to be processed and after confirmation of the data, it will start the processing the information.

• **Storage Module:** this component will handle the storage of the data in the RDF format.

The data can be fetched using SPARQL queries, an RDF query language, that is, a semantic query language for databases, able to retrieve and manipulate data stored in Resource Description Framework (RDF) format.

Examples of queries can be the following:

→ This query fetches all hotspots stored in the database:   
  
PREFIX bimr: <http://xmlns.com/bimr#>

PREFIX user: <http://xmlns.com/bimr/user#>

PREFIX tweet: <http://xmlns.com/bimr/tweet#>

PREFIX location: <http://xmlns.com/bimr/location#>

PREFIX observation: <http://xmlns.com/bimr/observation#>

PREFIX hotspot: <http://xmlns.com/bimr/hotspot#>

PREFIX uid: <http://www.w3.org/2001/vcard-rdf/3.0#UID>

PREFIX name: <http://www.w3.org/2001/vcard-rdf/3.0#NAME>

PREFIX address: <http://schemas.talis.com/2005/address/schema#>

PREFIX screenName: <http://www.w3.org/2001/vcard-rdf/3.0#NICKNAME>

PREFIX address: <http://www.w3.org/2001/vcard-rdf/3.0#ADR>

PREFIX email: <http://www.w3.org/2001/vcard-rdf/3.0#EMAIL>

SELECT ?bird\_species ?hotspot\_id ?how\_many ?observation\_date ?latitude ?longitude ?tweet\_id ?tweet\_text ?tweet\_language ?information\_source\_id ?author ?link ?city ?country ?state ?user\_name ?user\_id ?email ?address ?screen\_name ?has\_geo\_enabled

WHERE {

?hotspot hotspot:hotspotId ?hotspot\_id;

hotspot:observation ?observation.

?observation observation:informationSourceId ?information\_source\_id;

observation:howMany ?how\_many;

observation:date ?observation\_date;

observation:tweet ?tweet;

observation:location ?location;

observation:birdSpecies ?bird\_species.

?location location:latitude ?latitude;

location:longitude ?longitude.

?tweet tweet:tweetId ?tweet\_id;

tweet:text ?tweet\_text;

tweet:language ?tweet\_language.

OPTIONAL { ?location location:country ?country }

OPTIONAL { ?location location:city ?city }

OPTIONAL { ?location location:state ?state }

OPTIONAL { ?tweet tweet:author ?author }

OPTIONAL { ?tweet tweet:link ?link }

OPTIONAL {

?hotspot uid: ?user\_id.

?user uid: ?user\_id;

name: ?user\_name;

}

OPTIONAL { ?user email: ?email }

OPTIONAL { ?user address: ?address }

OPTIONAL { ?user screenName: ?screen\_name }

OPTIONAL { ?user user:hasGeoEnabled ?has\_geo\_enabled }

}

ORDER BY ?hotspot ?bird\_species

→ The following query fetches all the tweets present in the database:

PREFIX bimr: <http://xmlns.com/bimr#>

PREFIX user: <http://xmlns.com/bimr/user#>

PREFIX tweet: <http://xmlns.com/bimr/tweet#>

PREFIX location: <http://xmlns.com/bimr/location#>

PREFIX observation: <http://xmlns.com/bimr/observation#>

PREFIX hotspot: <http://xmlns.com/bimr/hotspot#>

SELECT DISTINCT ?tweet\_id ?tweet\_text ?tweet\_language ?author ?link

WHERE {

?tweet tweet:tweetId ?tweet\_id;

tweet:text ?tweet\_text;

tweet:language ?tweet\_language.

OPTIONAL {

?tweet tweet:author ?author;

tweet:link ?link;

}

}

→ The next query returns the most observed species (by occurrence number):

PREFIX user: <http://xmlns.com/bimr/user#>

PREFIX tweet: <http://xmlns.com/bimr/tweet#>

PREFIX location: <http://xmlns.com/bimr/location#>

PREFIX observation: <http://xmlns.com/bimr/observation#>

PREFIX hotspot: <http://xmlns.com/bimr/hotspot#>

PREFIX uid: <http://www.w3.org/2001/vcard-rdf/3.0#UID>

PREFIX name: <http://www.w3.org/2001/vcard-rdf/3.0#NAME>

SELECT ?bird\_species (COUNT(?bird\_species) as ?occurence\_number)

WHERE {

?hotspot hotspot:observation ?observation.

?observation observation:informationSourceId ?information\_source\_id;

observation:birdSpecies ?bird\_species;

}

GROUP BY (?bird\_species)

ORDER BY DESC(?occurence\_number)

• **Presentation layer:** the component with which the user will interact with it. It will handle the rendering of the map, the creation of areas where the birds have been spotted based on the data provided from the others modules.

**3. Application use cases**

The way this application has been developed allows people to find new places for birds spotting: the ornithology is a hobby and a passion for some of us and the application will permit to view and analyze best locations for live seeing the birds.

People who are willing to use the collected data after the processing part, will have all the collected information will be available to be accessed by other applications in a standard manner.

Students, academic stuff and scientists: ornithology studies will be possible through this application via statistics offered by the application. With a filtering algorithm the user can choose between different species, locations and historical dates and the map will be reloaded with the desired information.

**4. Advantages of this application compared to other solutions**

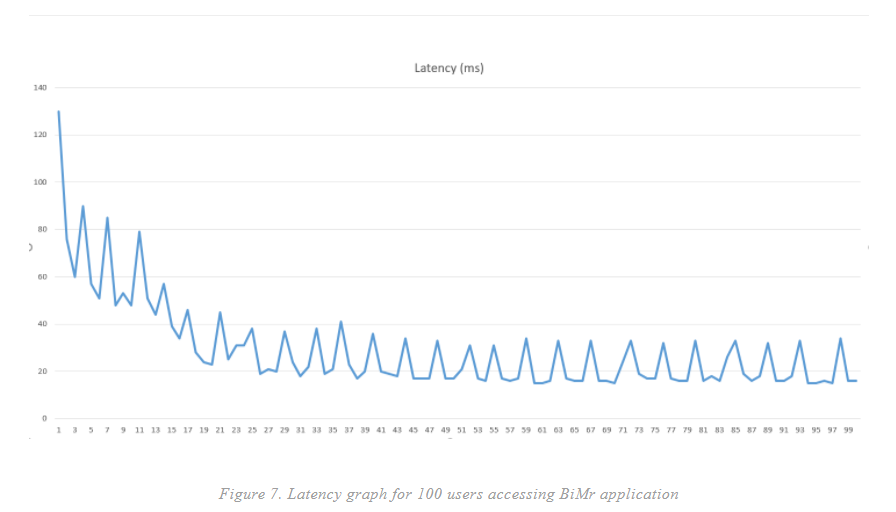
Thinking of getting information from social networks, in some cases those information can be wrongly interpreted. For example a tweet can contain the *#birdmig* hashtag, the location and date, but in the reality the user refers to a hunting plane (MIG).

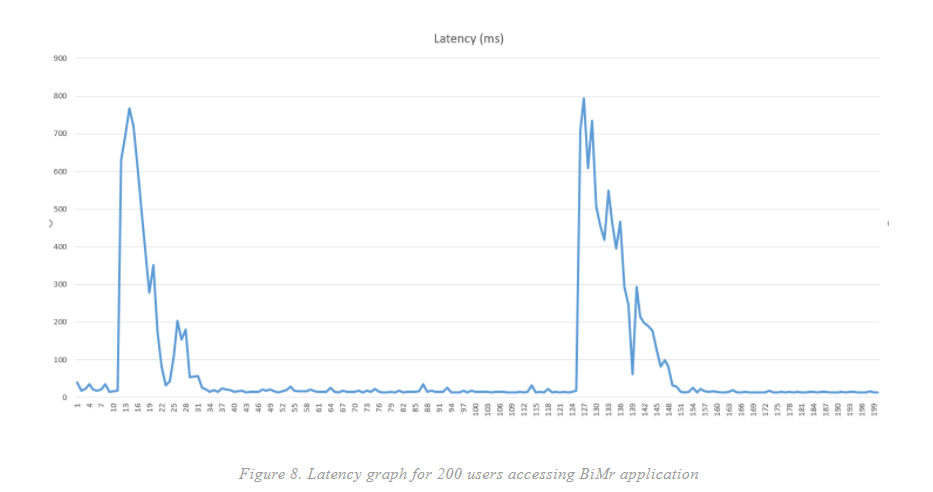
To avoid such situations, our Data Processing module filters those tweets using the Stanford NLP library. This is a team of faculty, postdocs, programmers and students who work together on algorithms that allow computers to process and understand human languages, by covering areas such as sentence understanding, automatic question answering, machine translation, syntactic parsing and tagging, sentiment analysis, and models of text and visual scenes, as well as applications of natural language processing to the digital humanities and computational social sciences.

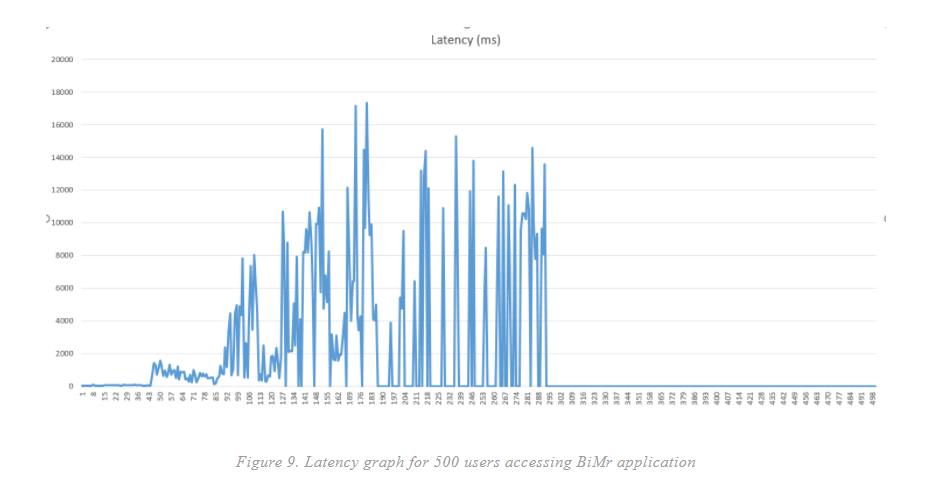
This comes as an advantage over the other existent solutions out there, because it allows us to train our own model using information from an arbitrary number of tweets at first, then use this model for efficient filtering purposes.

Another advantage is that, even though this application is in its early stages of development, stress testing shows that it can successfully treat and respond requests from over 200 users at the same time.

For this matter, we have conducted non-functional testing using tools like Apache JMeter™, which is an open source software, a 100% pure Java application designed to load test functional behavior and measure performance. It was originally designed for testing Web Applications but has since expanded to other test functions.







We can easily observe that handling 100 users requesting access to our application doesn't take too much.

Almost the same goes for the case where BiMr server has to cope with 200 requests at once from 200 users. There are some increase spikes in latency however, especially when our application makes use of the scheduled caller to the Twitter API.

When it comes to handling 500 users at the same time, the application server can handle about one hundred requests before starting to lag (latency spikes to be observed). Moreover, after about 300 requests, the server enters a freeze state where it cannot handle the remaining requests in the waiting queue.

We can easily see that the application could handle a relatively small but important number of users in its incipient phases and could really have improvements on how it can handle the requests more efficiently in the future versions.