Information Retrieval and Web Search - Project Phase 1

Lukas Pfahler Tejas Umakanth

February 12, 2014

1 Collaboration Details

We both do everything. clever...asdfa sdkfjalöksdj fölakdjf öaljdf,ajhdf a abdf afas dasfdghalkjdhfa

adsfadsf

sfg asd

2 Description

For our project, we decided to crawl and index data found on Instagram[1]. Instagram is a popular[2] social media application that allows users to publish photos and videos and search the media published by other users. Each user is identified by a unique user name. Each media item can have a caption, a list of comments by other users or a location. As known from other popular social networks like Twitter and Facebook, captions and comments can contain hashtags.

[1] `http://instagram.com`

[2] Instagram reported 150 million users in September, 2013.

3 Architecture

3.1 Crawler

Instagram offers a developer API that allows us to subscribe to real time updates and crawl new media items as soon as they are posted.[3] There are different options for subscriptions: You can either subscripe to a specific user, hashtag or to a geographic area or location. All communication between the crawler and instagram is done using the HTTP protocol: The crawler is itself a http-server and once we subscripe to media updates at instagram.com, their servers start connecting to our crawler. Everytime there is new media, they issue a http POST request. The post does not contain the actual media, but is merely a notification that there has been an update. We then grab the actual data by requesting all recently added media from their servers in a http GET request.

The response is a JSON file containing a list of media items. We save each of those items in a separate JSON file. We also save a 150×150px jpeg thumbnail image for each media item.

Our crawler is based on the software platform Node.js, which allows us to setup a http-server and request data in an asynchronous fashion with very little javascript code.

> Node.js is a platform built on Chrome's JavaScript runtime. [..] It uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.[4]

[3] http://instagram.com/developer/realtime/

[4] http://nodejs.org/

Instagram allowes developers to issue 5000 requests per hour[5], thus we introduced a politeness factor p: Only every p-th time our crawler is notified we actually request the new data. This has three effects: First, the number of requests is reduced and second, the number of new media items per response is higher. It also means that we probably miss a fraction of the data. However, if one would setup a larger system with multiple crawlers the coverage would increase. Third, for subscriptions with a very high update frequency, this makes sure we don't try to download more than our own bandwidth limit allows us to.

3.2 Indexing

A problem with the data crawled from Instagram is, that it contains only a relatively small amount of caption text - oftentimes the caption is merely a list of hashtags. To get more text, we index a concatenation of the caption and all comments. If the item is supplied with a named location, we also add the name to the indexed text. We call this concatenation *body*. From *body* we remove all #-characters. If the crawler subscribed to a specific hashtag, we also remove this tag, since it is present in all items and thus is useless for indexing.

Furthermore, we introduce a field just for indexing the list of hashtags provided to the media item. Note that this are both hashtags from the caption and from the comments. Also we index the username separately.

Finally, we use Lucene's spatial indexing module to index the media items by their GPS locations. Internally, this builds a inverted index based on Geohash[6]. The spatial module hashes longitude/latitude points to strings, such that close points share a longer prefix. It then builds a prefix tree on those strings and builds a inverted index of its nodes.

[5] `http://instagram.com/developer/endpoints/`

[6] See `http://en.wikipedia.org/wiki/Geohash` for more details

4 Usage

4.1 Crawler

To run our crawler, you first have to install node.js, which you can download at

`http://nodejs.org/download/`

We are also using the request node-module, which you can download and install by running the following command:

```
npm install request
```

Now you can run our crawler by calling

```
node instagramStream.js
```

which will start a http-server on port 1337 of your local machine. Please note that port 1337 of your machine has to be open to the public. To actually start crawling, you have to subscribe to instagram media updates. This is done by calling the following command:

```
curl -F 'client_id=620c46f7e7da46149b0ad5d3cd8268ba' \
     -F 'client_secret=c57beab5c2864e608b0dc4e46e5f8b50' \
     -F 'object=tag' -F 'aspect=media' -F 'object_id=selfie' \
     -F 'callback_url={http://YOUR_IP:1337}' \
     https://api.instagram.com/v1/subscriptions/
```

This will subscribe to all updates of the (very popular) hashtag #selfie; the crawler will start storing media items in a newly created folder selfies.

You can unsubscribe from updates by running

```
curl -X DELETE 'https://api.instagram.com/v1/subscriptions?client_secret=\
     c57beab5c2864e608b0dc4e46e5f8b50&object=all&\
     client_id=620c46f7e7da46149b0ad5d3cd8268ba'
```

You should always unsubscribe before shutting down the crawler (`CTRL-C`), if you fail to do so you might not be able to subscribe to media for a while. We suspect that the servers at Instagram need some time to realize that our crawler is not running anymore and continue to try to send the new subscriptions to our already shutdown crawler.

4.2 Indexer

5 Performance

The time needed to for indexing is dependant on the system on which the indexing is done. This performance analysis was carried out on a laptop with a 2.4 GHz CPU. For performing this test, we use a document containing 3GB worth of JSON files that are returned by our crawler. Then we try to index 10k,20k and so on up to 50k JSON files in the document and calculate the time taken to index in each case. As we can see in figure 1, the indexing time increases almost linearly as the number of files that are indexed increases.

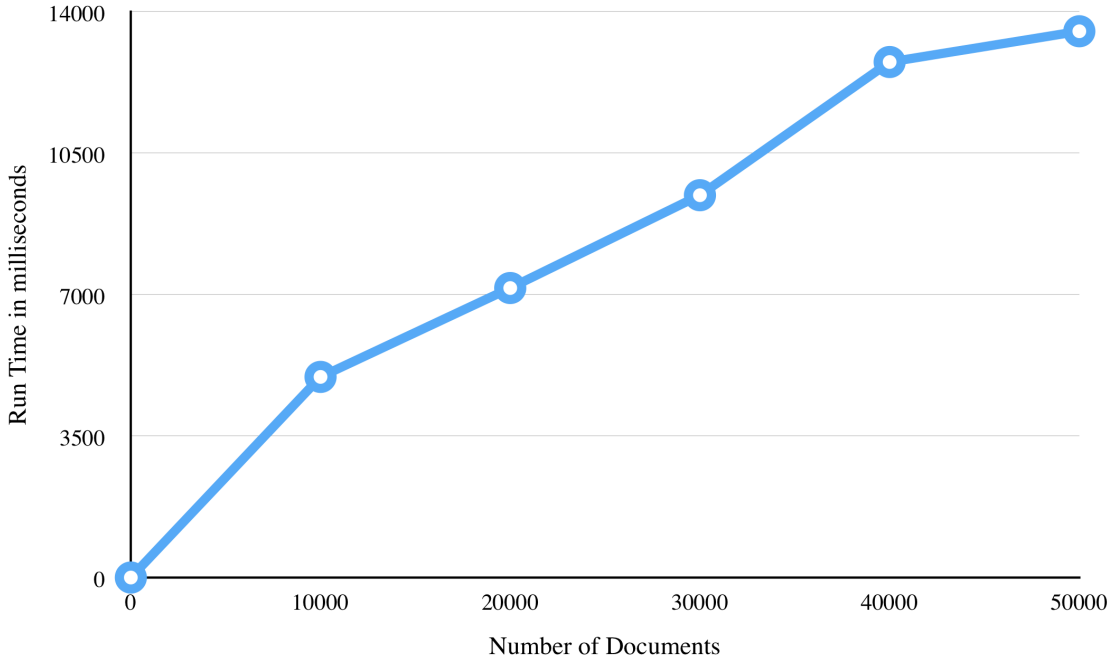

Figure 1: Indexing Performance