Efficient Web Scraping and Database Synchronization

Group 3 - Henrik Hygerth and Michael Håkansson

DD2471 Modern Database Systems and Their Applications,
School of Computer Science and Communication (CSC),
Royal Institute of Technology KTH, Stockholm, Sweden
hygerth@kth.se, michak@kth.se

Abstract. This paper presents a web scraping application that is built in Go with a MongoDB database integration. A web scraper is an application that extracts information from websites or other web resources. By storing the data in a local copy, queries can be made to the local copy rather than to the original data source. The web scraper in this paper is an example on how concurrent programming and theories on database synchronization can be used to build web scapers that are efficient in collecting data and keeping the local copy up-to-date. The particular web scraper in this paper extracts data from three Swedish video on demand services; SVT Play, TV3 Play and Kanal 5 Play where the amount of extracted data is minimized by selecting only relevant data for the end use.

Keywords: database synchronization, web scraping, web harvesting, web data extraction, web crawling, skylark

1 Introduction

Web scraping is a technique used to extract data from websites or other remote web sources. The reasons for extracting data from remote sources and making local copies of the remote data can be many. Typically, the data is collected and stored in order to be able to perform local analysis or to compile data from many different remote sources in once place.

The amount of data on the Internet is constantly increasing, and by the 7th of May 2015, the Indexed Web alone contained at least 4.59 billion web pages according to WordWideWebSize.com. [2]. The seemingly endless increase in the number of web pages puts considerable demands on the web scraping techniques and methods.

When the data has initially been downloaded to the local copy, the data source regularly has to be revisited and checked for updates in order for the local copy to stay up-to-date [1, p.2]. This process is called *synchronization*. To have a notion of which elements that are updated and which are not, the concept of *freshness* is used. Freshness of an element in the local copy is defined as in definition 1.

Definition 1. The *freshness* of a local element e_i at time t is

$$F(e_i, t) = \begin{cases} 1 \text{ if } e_i \text{ is up-to-date at time t} \\ 0 \text{ otherwise.} \end{cases}$$

To decide on when synchronization is to be performed, the age of elements in the database must be known. The age of an element in the local copy is defined as in definition 2.

Definition 2. The age A of the local element e_i at time t is

$$A(e_i, t) = \begin{cases} 0 \text{ if } e_i \text{ is up-to-date (i.e. fresh) at time t} \\ t \text{ - modification time of } e_i \text{ otherwise.} \end{cases}$$

The data scraping problem was illustrated in a paper by Cho et.al. [1], and has been modified in figure 1 to show the selective data fetching. The idea behind selective data fetching is to only select and store the data that is relevant for the end use. By doing this, the amount of stored data can be minimized.

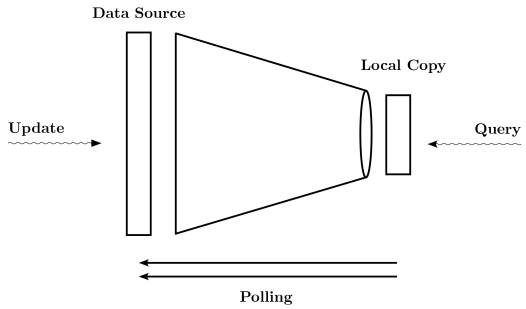

Fig. 1. Illustration of the concept of selective data fetching from data source to local copy. The illustration was inspired by the work of Cho et.al. [1].

By dividing computational work on multiple threads, *concurrent computation* can be achieved. This is useful in systems where computations can complete without the need of the results from other simultaneous computations. Concurrent computation is a good way to fully utilize the computer resources and reduce the total computation time.

2 Implementation

This section describes the implementation of a web scraping application with codename *"Skylark"* and the synchronization policy that was used.

2.1 The Web Scraper

The web scraper part of *Skylark* (which is called *Skywalker*) was built in the Go programming language. The reason for choosing Go was its built-in concurrency primitives and the overall system performance.

The web scraper visits the websites of the video on demand services SVT Play, TV3 Play and Kanal 5 Play. For each site, the scraper downloads information on all shows in the service. For each show, the scraper iterates through all episode of that show and extracts the necessary data such as title, length, broadcast date and the link to the video stream. Since the services differ in structure, the scraper will need customizations for each service. Therefore a module for each service was created, enabling new services to be easily added and modified in the future.

Scraping can be quite slow due to all the external connections and extractions that are needed. Therefore, concurrent computation was used to fully utilize the computational power at hand. A drastic increase in performance was noticed when the scraping was changed from sequential to concurrent execution.

The scraped data is stored in a NoSQL database in the form of MongoDB which will be described more in section 2.2.

2.2 Database

The database used in *Skylark* is a MongoDB database. MongoDB is a NoSQL database that uses JSON documents instead of the traditional tables as in relational databases such as MySQL and

PostgreSQL. The reason for choosing MongoDB for the database is partly because of the speed and partly because of the flexibility.

2.2.1 Speed One reason for MongoDB being fast is that related data is stored together in documents. Therefore, queries can in some cases execute much faster than in relational databases, where data is often spread in different tables that needs to be joined [3]. Another reason for the high speed is that MongoDB does not provide as strong consistency guarantees as SQL databases. It does however prevent multiple clients from modifying the same data piece of data at the same time [4], which is considered to be enough in this particular setup since no critical data is stored and the data is constantly being updated.

2.2.2 Flexibility MongoDB offers great flexibility thanks to the JSON document setup. There are no tables that data has to fit into, and the structure of documents can be altered at any time. Suppose that a play service adds some information that would be interesting to scrape. This new information could then easily be added in future synchronizations without having to worry about the integrity of previously downloaded data.

2.3 Synchronization

To handle the synchronization, a synchronization policy is used. The theory behind the synchronization policy that is used in *Skylark* is presented below, followed by a description on how the policy was implemented in the web scraper.

2.3.1 The Synchronization Policy The synchronization policy decides the *synchronization frequency* for a show, and hence how often the scraper should check the data source for updates. The synchronization frequency for a show S with episodes e_i ($i = 1, 2, ..., N$ sorted on broadcast date in descending order) is defined as the number of synchronizations that should be performed per day for S, and is denoted by λ_S where

$$\lambda_S = \frac{1}{N-1} \sum_{i=1}^{N-1} (e_i - e_{i+1}).$$

The effect of this synchronization policy is that a show that has an average update interval of x updates per week is getting synchronized x times per day.

The purpose of the policy is to reduce the number of times the scraping is performed for each show, with as small impact as possible on the overall freshness of the show.

2.3.2 Using the Synchronization Policy In the web scraper, the synchronization policy for a show S is implemented using a floating point number S_{sf} on how many times a week the show is updated in the data source (based on the *synchronization frequency* - described in section 2.3.1), and the amount of time that has passed since the last synchronization S_{synced}. Depending on how often the synchronizations should be performed, the time frame T can be set to other time periods than weeks.

This pseudocode shows how for each show the decision to synchronize a show or not is performed:

```
for each S in shows:
    maxTimeSinceUpdate = T / S_sf
    if time_now - S_synced > maxTimeSinceUpdate:
        synchronize S
    else:
        skip
```

2.4 Client Implementation Prototype

To test the usage of the results from the web scraper, a web client that works as well on mobile phones as on tablets and desktop computers was developed. The web application was successfully built using HTML5, CSS3 and Go's built-in HTML template package. See figure 2 for screenshots.

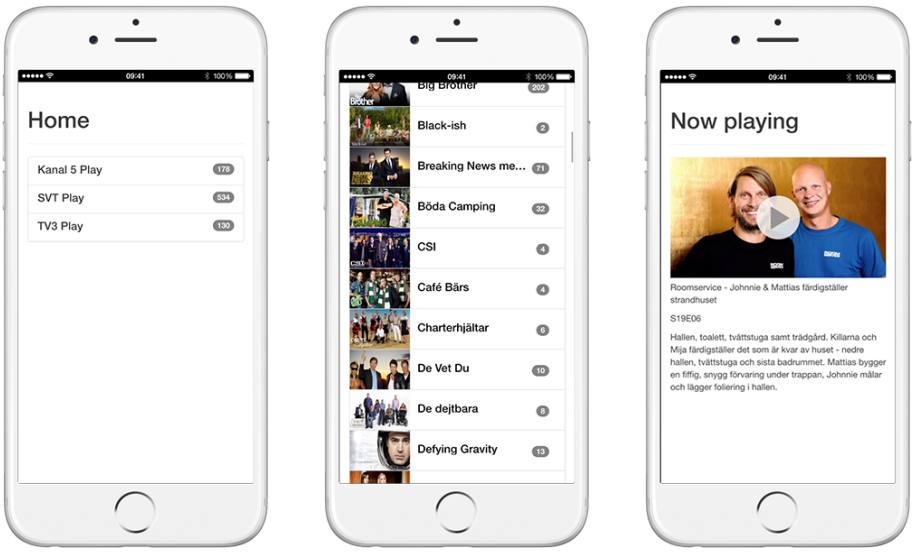

Fig. 2. Screenshots showing the mobile phone experience of the web client. Service overview on the left, show overview in the middle and episode details on the right.

3 Conclusion & Future Work

In this paper, we have described the implementation of a web scraper application called *Skylark*. The concept of selective data fetching has been described, and a synchronization policy has been presented. Both selective data fetching and the synchronization policy are used in the implementation of the web scraper. The result is a web scraper that collects data from three Swedish video on demand services and stores the data in a MongoDB database.

The web scraper application collects data and regularly synchronizes with the service providers. The collected data can be utilized by client applications to provide content from multiple video on demand services in one single view. To test this, a web client was successfully implemented.

The work presented in this paper is only a scratch on the surface when it comes to the possibilities of web scraping. For the particular web scraper implementation put forward in this paper, a more sophisticated client could be developed in contrast to the prototype developed for testing the web scraper application. The application is easily scalable to include more services, and a potential future project would be to expand the number of services to get a more comprehensive application.

References

1. Cho, J., Garcia-Molina, H.: Synchronizing a database to improve freshness. Acm Sigmod Record. 29(2), 117–128 (2000)
2. de Kunder, Maurice: WorldWideWebSize.com. www.worldwidewebsize.com. Accessed: 2015-05-07
3. MongoDB: Introduction. http://www.mongodb.org/about/introduction/. Accessed: 2015-05-19
4. MongoDB: Concurrency. http://docs.mongodb.org/manual/faq/concurrency/. Accessed: 2015-05-19