Cloud Computing
PaaS: Web Crawler

Sonja Biedermann Augusto Jose de Oliveira Martins
Tobias Harald Herbert

December 7, 2018

Contents

1 Project Description

2 Implementation

We decided to implement the project using Python3. Our cloud platform of choice is AWS. The only dependencies the project has are

- `requests`, for issuing HTTP requests,
- `validators`, for validating URLs, and
- `boto3`, the AWS SDK for Python.

We use DynamoDB for storage and SQS for message passing. The workers are realized as AWS Lambda functions. The master can be deployed using ElasticBeanstalk (AWS' traditional PaaS offering), which in turn uses Autoscaling, S3, Load Balancing and monitoring automatically, although this is not needed—the traffic on the master is very low.

The implementation can be found on GitHub.

2.1 Frontend

The frontend is a simple one-page website which communicates with the master node running on EC2 and displays data derived from the crawl and information about the current state of the crawler.

We display how many workers are currently working on the task. Since we use AWS Lambda, whose functions are quite short-lived, this chart sees a lot of activity with workers popping in and out of existence. The purpose of this chart is to "visualize the cloud", and to see how it scales.

The result of the crawl is displayed as a network. Nodes are colored according to the host, however note that colors may be repeated. To draw the network, we use the constraint-based layout algorithm CoLa[1], which basically runs a small simulation and optimizes the layout. The results from this algorithm are excellent, but graph drawing is hard and as a consequence we limit the graph display to 500 edges. The performance is heavily dependent on the graph structure, e.g. networks with very fat hubs (e.g. Wikipedia) have noticeably worse performance than networks with many smaller hubs. This is due to how large the neighborhoods tend to be in the hubs vicinity. It also does not help that it is running in the browser.

The graph is interactive and allows you to drag the nodes around should you not be satisfied with the layout. You can also click on a node to visit the associated webpage.

Figure 1 shows the result of a crawl on the webpage of an ongoing research project. The graph shows 4 hubs: reddit (red), arXiv (green), Wikipedia (orange), and the web presence of the University of California in Riverside (blue), which is split between two involved researchers.

The bottom displays the current number of discovered edges, which is equivalent to the amount of pages crawled, as well as the current depth. Do note that the typical web page has an extremely high fan out at depths of about 3 to 4, so the time spent at consecutive depths grows exponentially. We have thus not opted to display this as a progress bar, since the rate of progress is hard to estimate.

2.2 Backend

2.2.1 Master

The master runs a lightweight web server (using Bottle, a micro WSGI web framework) serving the frontend and offers a small REST API for querying the state of the crawl. This basically consists of starting the crawl and querying the workers and discovered edges.

The master can be deployed to an EC2 instance using ElasticBeanstalk or run locally.

2.2.2 Worker

3 Setup

[1] `https://ialab.it.monash.edu/webcola/`

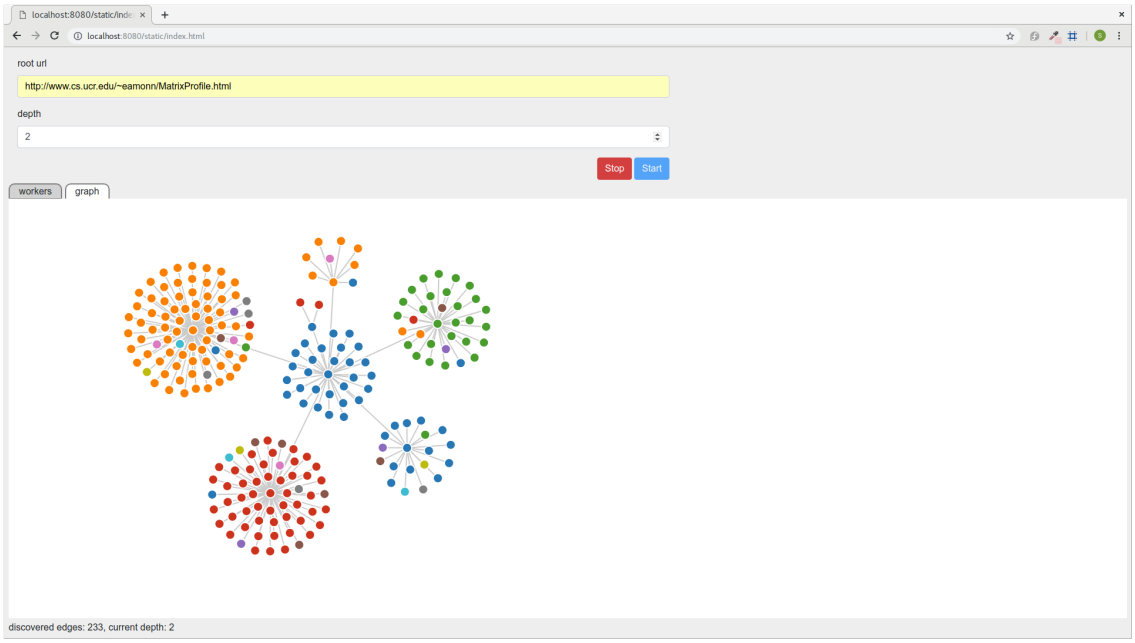

Figure 1: Screenshot of the graph view