

ECE 563 Cloud Computing

URL Shortening Web Service Final Report

Chengzhang Ma, Wanxin Yuan

April 30th, 2018



Introduction

An URL(Uniform Resource Locator) shortening web service is developed in this project. This web service can generate a short alias for a long URL, which can be redirected to the original long URL. Shortened URLs are easier to be remembered, passed around and fit in situations when space is limited. For example, this web service can be used on Twitter when the user wants to share a website link and still keeps tweets under 140-character limit. This web service can also track the click data for each shortened URL, and display top clicked short URLs.

In this project, MEAN (MongoDB, Express, AngularJS and Node.js) stack is adopted to develop the web application. To be more specific, HTML, CSS and AngularJS are used for front-end design, Node.js is used for back-end design, MongoDB is used as database. Moreover, Nginx reverse proxy/load balancer are applied and Redis cache layer is implemented to improve the performance and scalability of the web application. Finally, the server is deployed on Duke Virtual Machine to be tested and used.

Use Cases / Requirements

There are four main use cases in our web service.

1. Enters a long URL for shortening.

The web service can generate a shortened URL when the user submits the original long URL.

2. Customizes user's own shortened URL.

The user can also assign his own shortened URL as an alias to the original URL. Our web service first checks whether the customized URL has already been used, if not, the customized URL will be assigned to the original long URL. If it has been used, the user needs to submit a new customized URL.

3. Redirects the shortened URL to original long URL.

When the user enters the shortened URL in the address bar, it will automatically be redirected the original long URL.

4. Keeps track of click data of a shortened URL.

The web service can track the number of times a shortened URL be clicked since it is generated.

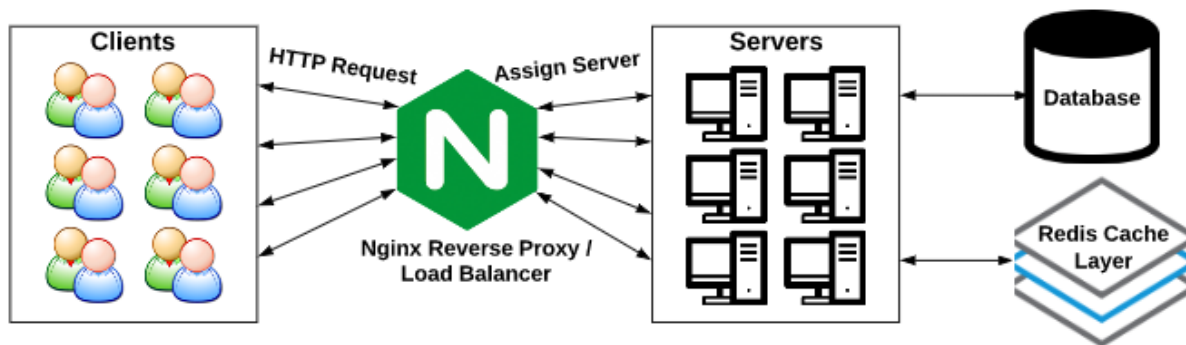
Project Schedule

The following table shows the weekly schedule for our project and we have finished all parts.

Task ID	Description	Owner	Start	Finish	Dependencies	Finished
BasicFrontEndDesign	Design the web page for users to input long URL and return the short URL	Yuan	02/23	02/28	None	✓
DataModelDesign	Design the data model to store the original URL and corresponding short URL	Ma	02/23	03/02	None	✓
UrlRedirect	Redirection function at the server side using Node.js	Yuan Ma	03/03	03/15	None	✓
UrlHashFuc	Determine and implement the hash function of URL transform	Ma	03/16	03/25	DataModelDesign	✓
CustomizedUrlPageDesign	Design the web page for users to customize short URLs	Yuan	03/16	03/20	BasicFrontEndDesign	✓
CustomizedUrlBackEnd	Accomplish the backend logic for customizing URL	Ma	03/26	04/03	UrlRedirect	✓
ClickDataModel	Design the data model to store the click data for each	Yuan	03/21	03/30	None	✓
ClickDataCollectBackEnd	Implement the logic to collect all the click data we need	Ma	04/03	04/10	ClickDataModel	✓
ClickDataPage	Design the web page to display the click data	Yuan	04/01	04/10	BasicFrontEndDesign CustomizedUrlPageDes	✓
Nginx/Redis	Apply the Nginx reverse proxy/load balancer and implement Redis cache layer	Yuan Ma	04/10	04/17	UrlRedirect UrlHashFuc	✓
SystemTest	Deploy the project on Duke Virtual Machine and do the test	Yuan Ma	04/17	04/24	All Above	✓

System Diagram

The following is the system diagram for our web service. It includes five different parts: clients, Nginx reverse proxy/load balancer, servers, Redis cache layer and database.



When clients submit the HTTP request, firstly the Nginx reverse proxy/load balancer assigns the task to one of the servers to deal with the request. The Nginx works on a round robin basis to do load balancing. It also hide logics and topologies from backend to the user, which increases security.

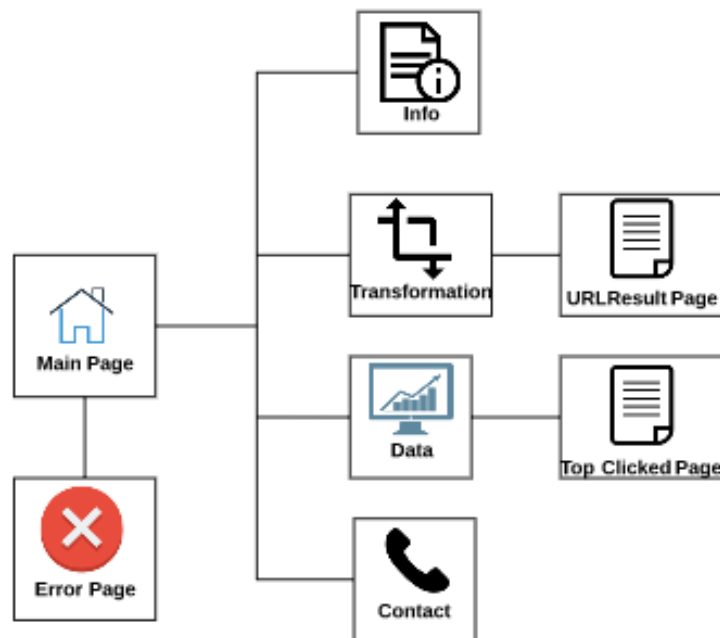
If the request is to redirect the short URL to original URL, the server will check the Redis cache layer first to find corresponding long URL. If such record doesn't exist in cache layer, it will search in the database instead. If corresponding result is found, the server will redirect to original long URL and render the page to the client.

If the request is to generate a short URL, the server will write a new record on short URL and original URL pair in the database. Then the server will render the URL generation result page to the client.

If the request is to find top 5 clicked shortened URLs data, the server will read the record from the database. Then the server will render the top clicked shortened URLs page to the client.

Front-end Design

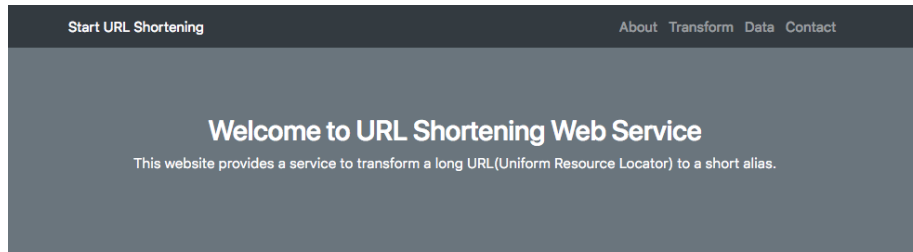
The following is the front-end design diagram for all web pages. The main page includes four section: Info section, URL transformation section, data section and contact section. It also has an error page if the user redirects to wrong website which doesn't exist. The URL transformation section also has a URL result page to show the transformation results, whether it is successful or not. The top clicked data section has a top clicked page to show the table including top 5 clicked shortened URLs and its original URLs. CSS, HTML and AngularJS are adopted in frontend design.



Main Page

1. Info Section

In this part, it contains navigation bar for our main page. The navigation bar contains four part: info, transform, data and contact. If clicked each part, it will navigate to corresponding contents in the main page. This part also has brief introduction about our URL shortening web service, which includes main features and how to use them. As mentioned in previous chapter, our web service has four main use cases. The following is snapshot for this section:



You CAN:

- Enters a long URL for shortening.
- Customizes your shortened URL.
- Redirects the shortened URL to original long URL.
- Keeps track of click data of a shortened URL.

Instructions:

- First you should write the original URL in the "Input URL" textbox.
- By clicking **"Transform"** button, you will get a shortened URL generated by our service.
- You can also customize your own shortened URL. By writing your customized URL in the "Customized URL" text box and clicking the **"Transform"** button, your customized URL is confirmed if it hasn't been used.
- You can only type in letters, numbers and dashes for your customized URL.

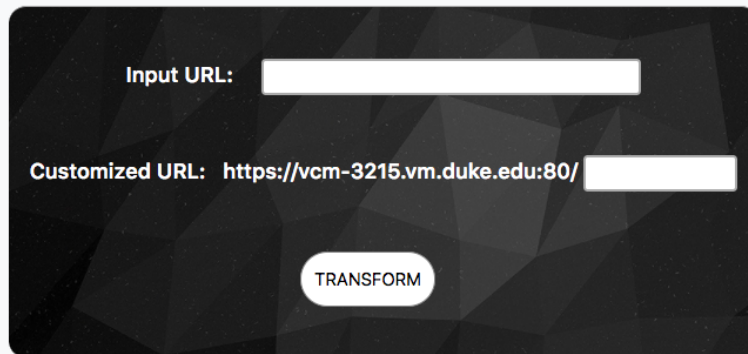
2. URL Transformation Section

In this section, it provides functions to transform from original long URL to a short URL. There are two ways to generate a shortened URL. One way is to generate by our web service. In this way, the user only needs to type in the original URL text box, and type nothing in customized URL text box, then clicking "Transform" button to generate a shortened URL. Then the URL result page will show the successful message and corresponding shortened and long URL.

Another way is to generate a shortened URL by user's customized URL. The user fills out both in original URL text box and customized URL text box, then clicking "Transform" button. If the customized URL hasn't been used, it will be generated successfully. If not, it will ask the user go back and try another customized URL.

The prefix of the shortened URL may change later, depending on the server end design. Currently we used "https://vcm-3215.vm.duke.edu:80/" as the prefix because the server is deployed on Duke virtual machine for testing.

Transform your URL

A screenshot of a web interface for transforming URLs. It features a dark-themed card with a geometric pattern. At the top, it says "Input URL:" followed by a white text input field. Below that, it says "Customized URL:" followed by a pre-filled URL "https://vcm-3215.vm.duke.edu:80/" and another white text input field. At the bottom center of the card is a white rounded button with the text "TRANSFORM".

Input URL:

Customized URL: <https://vcm-3215.vm.duke.edu:80/>

TRANSFORM

The following is URL result page that shows successful message and corresponding URLs pair.

Success



The URL you want to transform is <http://www.google.com>

The shortened URL is [aaaaa5](#)

BACK TO MAIN

3. Top Clicked Data Section

In this section, the user can obtain top 5 shortened URLs that are clicked and how many times they have been called. Our web service has a full record for clicking times on all shortened URLs. Each time the user clicks on “Generate” button, the web page will redirect to a top clicked URL page and display a table showing top 5 records, each includes shortened URL, original long URL and its clicked times. The top 5 clicked URL page is shown as following:

Top 5 clicked URL!


#	Original Url	Shorten Uri	Click Number
1	http://www.baidu.com	aaaaab	4
2	https://nodejs.org/en/	aaaaa3	3
3	http://www.dnvod.tv/Movie/detail.aspx?id=lvziqHQ7igk%3d	aaaaan	2
4	https://www.google.com/	google	1
5	https://superuser.com/questions/421074/ssh-the-authenticity-of-host-host-cant-be-established/421084?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa	000002	1

[BACK TO MAIN](#)


4. Contact Section

In this section, it will just show the developers' contact information, which is shown as below:

Contact us



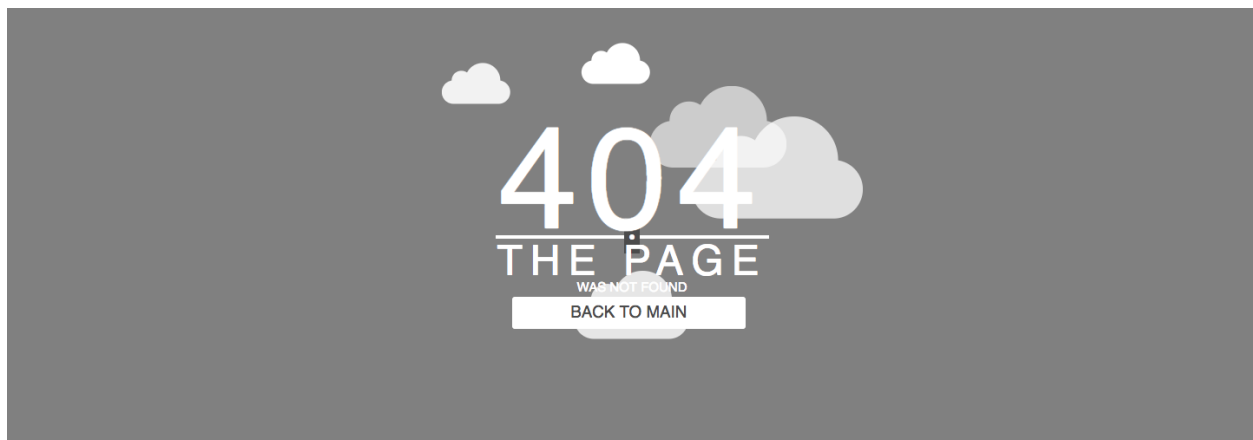
Chengzhang Ma
chengzhang.ma@duke.edu



Wanxin Yuan
wanxin.yuan@duke.edu

Error page

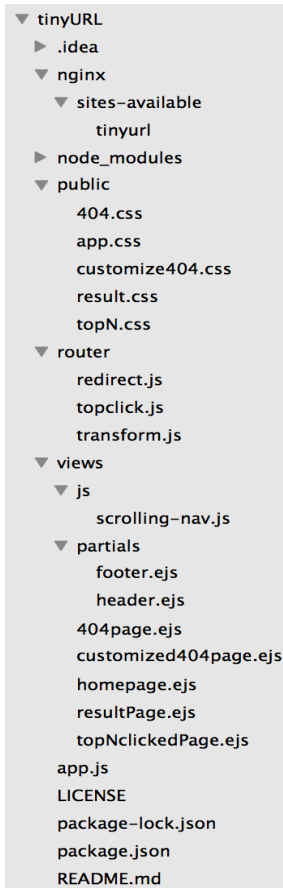
If the user submit a request with an invalid short URL, for example, the shortened URL doesn't exist, it will redirect to an error page and let the user try again.



Back-end Design

Node.js and Express

Since this is a MEAN project, we chose to use Node.js to implement our backend and use Express as the framework, which provides a robust set of features for us to develop web applications. By installing Node.js and npm, then setting up the PATH variable, we wrote JavaScript files to realize the backend logic of our URL shortening service. Below is the code structure of our project.



```
▼ tinyURL
  ► .idea
  ▼ nginx
    ▼ sites-available
      tinyurl
  ► node_modules
  ▼ public
    404.css
    app.css
    customize404.css
    result.css
    topN.css
  ▼ router
    redirect.js
    topclick.js
    transform.js
  ▼ views
    ▼ js
      scrolling-nav.js
    ▼ partials
      footer.ejs
      header.ejs
      404page.ejs
      customized404page.ejs
      homepage.ejs
      resultPage.ejs
      topNClickedPage.ejs
  app.js
  LICENSE
  package-lock.json
  package.json
  README.md
```

Key Backend Logic

1. URL Conversion Logic (<http://vcm-3215.vm.duke.edu:80/urls>)

This function transforms the long URL to a shortened one. We first came up with two solutions in implementing this function. One is hashing, such as MD5 or other hashing functions, then using a hash function to convert long string to short string. However, there may be collisions in hashing, where more than one long URL map to same short URL. Because we need a unique short URL for each long URL so that the redirection can work correctly, this method needs to solve collision issues.

The other method is to keep a counter, which is stored in the database, and convert it into a 6 characters long string. The URL character can be a lowercase alphabet character, an uppercase alphabet character or a digit. There are total $26 + 26 + 10 = 62$ possible characters. There are total $62^6 = 5.6 \times 10^{10}$ permutations in this way, which is enough for our project. So, the task basically is to convert a decimal number to base 62 number.

In our project, we chose the counter way to convert the URL. It is a POST request and need the user input of a URL to be transformed. First, we will check if the user input is valid, and will return to homepage if there is no valid input. Every time a new input is posted to the backend, it will first count the total number of records in database, and then use this number as the counter to generate the short URL for this original URL. After this, it will be stored in the database and also respond to the client with a result page showing their transformation is completed.

To render a customized result page for every different input and output, we use Embedded JavaScript templates (EJS) files so that we can pass parameters into it based on the different user input.

2. URL Redirection Logic (<http://vcm-3215.vm.duke.edu:80/shortURL>)


When the server received a request for the short URL, it will send a redirect response. The “shortURL” in the URL is a variable, which can be taken into the backend function. Then the server will search this short URL in the database and find its corresponding long URL then redirect to that website. If the request is a URL that doesn’t exist in the database, it redirects to an 404 page to remind the user to try again.

3. Customized URL Logic (<http://vcm-3215.vm.duke.edu:80/urls>)

This is a logic need to be integrated into the URL Conversion Logic, since they both take in the same input form from the client. Therefore, to differ these two logic with each other, the server first need to check if the customized URL field is empty. If it’s not empty, which means the user want to have it customized, then it will check with the database to see if this customized URL is used before. If it’s not taken up already, it will be stored to the database and return the result page back to the client. If it is used, it redirects to a 404 page and let the user try it again.

4. Top 5 Clicked URL Logic (<http://vcm-3215.vm.duke.edu:80/topClick>)

After receiving the request to show top 5 clicked shortened URLs data, the server will sort the data in the database by clicking times, then return top 5 records and display those information on the top clicked URL page. However, if with the increasing of size of database, the sorting will cost more time. At that time, we needs to consider some efficient way to keep top 5 records.



To keep track of the click number of each URL, every time a tinyURL is visited, before we redirect it to its original page, we need to get its old click number and add one more time to it. Moreover, we can have much more comprehensive data of click information, such as geographic location, the web page where the link was clicked, and so on. This type of information is invaluable to webmasters and companies. It shows where customers are coming from, when they are coming, and what interests them.

Post/Redirect/Get

During our development, we met a problem that when the user refreshes the result page, the browser will ask you whether you want to submit the form again. If we choose yes, then it will post the input to the server again and the server will generate a new tinyURL or an error may even occur. After studying about this problem, we decided to follow the Post/Redirect/Get(PRG) pattern to deal with it. PRG is a web development design pattern that prevents some duplicate form submissions, creating a more intuitive interface for user.

By using PRG pattern, when the user post an input to the server, the server will first generate a valid short URL, then it will redirect to a Get method and render the result page through this Get method, by passing the new generated tinyURL as a parameter. In this way, when a user refresh this result page, it will access the Get method and will not submit the form again.

MongoDB

MongoDB is adopted as the database for our web service because MongoDB does not has a fixed schema compared to Relational database and it will be flexible if we want to expand more information related to one short URL. For now, the schema of our database is shown as below:

Schema: *short url, original url, clicked number*

We created a new database called tinyurl and had one collection in the database right now named urls. Each record will have the information about the shortened URL, original long URL and the clicked number for shortened URL.

We run the main MongoDB service with *mongod* command and connected node.js server with MongoDB using *mongoose* library. The MongoDB shell can help us check if we have successfully created the collections and stored related data.

Nginx Reverse Proxy/Load Balancer

To handle the possible large amount of request to our server, we decided to use multiple servers to increase its scalability and availability. Meanwhile, to support these multiple servers, we chose Nginx as the reverse proxy and also as load balancer to assign requests for servers.

As a reverse proxy, Nginx can help accept requests from clients, forward it to servers and return the server's response to clients. The benefits of reverse proxy is two-fold. First, it can increase security because no information about our backend servers is visible outside our internal network. Second, it can

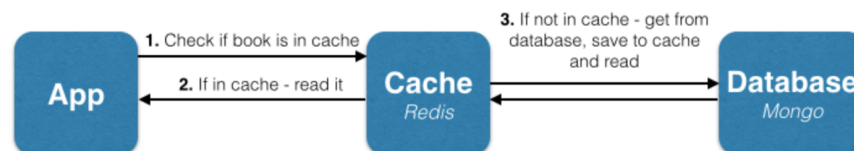
help increase scalability and flexibility because clients see only the reverse proxy's IP address, we are free to change the configuration of our backend infrastructure. This is particularly useful In a load-balanced environment, where we can scale the number of servers up and down to match fluctuations in traffic volume.

As a load balancer, Nginx distributes incoming client requests among a group of servers, in each case returning the response from the selected server to the appropriate client. In our project, Nginx can distribute the workload in a Round Robin style which can make the best use of each server's capacity, prevents overload on any server and results in the fastest possible response to clients.

After downloading and configuring the nginx server, we wrote a config file for our application, to set up the address of our servers and the round robin strategy. First it is in the sites-available folder and then a soft link was created from it to a copy in sites-enabled folder. It finally works after restart the nginx service.

Redis Cache Layer

Performance is one of the most important things we evaluate when developing our project. Since our web servers need to interact with a database, these operations can become a bottleneck. Therefore, we decided to implement a cache layer to speed things up.




In our project, we chose Redis to serve as our cache layer. Redis is an open source, in-memory data structure store, which can work as cache. First, we install and run the redis service on our visual machine. Then we revised the backend logic of redirection to add the cache layer. As the picture shows, First of all, the servers will look in the cache to see if they already brought it there. If they have, that's great because they just had a cache hit, that is finding an item in the cache. If they haven't found it, they had a cache miss. In the case of a missing item, they'll have to look for the original URL in the database. When they find it, they'll return it to clients and also insert it into the cache.

What's more, we can set up the limit amount of memory it can use and also the deletion policy due to the memory limitation. We chose to use LRU(Least Recently Used) mechanism to delete the cache items that were the least recently used.

VM Deployment

At first, we just built all of our project locally, to make it simple and easy to test. After all the features are implemented and all the tests are done, we decided to deploy our project to Duke Visual Machine, to



make this service public so that we can invite more friends to try it out and maybe give us some useful feedbacks.

To deploy it on Duke VM, first we applied for a Ubuntu VM from OIT and installed node and npm on it. Then we downloaded and configured MongoDB, Nginx and Redis as well. Finally we pulled our code from GitHub and ran two instances of it, one is listening to port 3000 and the other listening to port 4000.

Challenges

In the front-end design part, the biggest problem is that neither of us has experience in web development. Hence, we need to take some time to learn basic knowledge in frontend design. At first, we adopted CSS, HTML, and JavaScript to implement the basic front-end design. However, the UI interface design seems not very good at first version. Later, we adopted AngularJS framework to improve UI interface design, which is effective and useful. Besides this, we also meet some specific technical issues such as how to dynamically update contents inside a table, how to fit a webpage size to fit any browser window size, etc. We have solved problems regarding front-end design part by reading and learning related materials.

In the backend part, the first challenge we met is the form resubmission problem. After studying about this problem, we decided to follow the Post/Redirect/Get pattern to deal with it. PRG is a web development design pattern that prevents some duplicate form submissions, creating a more intuitive interface for user. By using PRG pattern, when the user post an input to the server, the server will first generate a valid short URL, then it will redirect to a Get method and render the result page through this Get method, by passing the new generated tinyURL as a parameter. In this way, when a user refresh this result page, it will access the Get method and will not submit the form again.

Another challenge is the asynchronous functions in node.js. At first we are not familiar with this characteristic, so that there are often some errors unexpected. Then we found out that it is mainly because these functions which access the MongoDB database is asynchronous in the JavaScript program. Finally, we got used to this coding style and made sure these functions can be executed in the correct order.

Changes in a second version

Basically, we stuck to our project plan during the whole semester and fulfilled all of our requirements in the end. However, if we got a chance to do a in a second time, we will do some improvements.

Firstly, we will choose another implementation logic for storing and displaying top 5 clicked shortened URL. Currently, we choose to sort the database each time the user press the “generate” button and then find top 5 clicked data. However, this method is not very effective when datasets are large because sorting will take much time. A better way to implement this logic is to maintain a heap with size 5 and the heap is sorted by clicked times. Then the server can access the heap directly to get top 5 clicked data each time and doesn’t need to sort the whole database. In order to make the heap always contain records regarding top 5 clicked data, each time a shortened URL is clicked, it will add one count on its click time and check whether it is needed to update the data in the heap. This will much increase the efficiency when database contains large data.

Secondly, we can have more comprehensive data of clicked shortened URL information. For example, to store user’s geographic location, the web page the link where the link was clicked, and so on. We can also let the user define the number of top clicked URLs they want to see in the table. Currently, we just pre-defined it as 5.

Thirdly, we will consider adopting Apache Cassandra as database instead of MongoDB. Apache Cassandra is a distributed database and has better availability and scalability. Thus, it will show better performance when dealing with larger datasets. It will help our web service to deal with larger number of requests and store large data.

Last but not least, we will have a better schedule on project plan. Currently we have finished the project in time, but it will be better if we could have more time on testing and on adding extra features.

Conclusion

In this project, an URL(Uniform Resource Locator) shortening web service is developed. This web service can generate a short alias for a long URL, which can be redirected to the original long URL. MEAN (MongoDB, Express, AngularJS and Node.js) stack is adopted to develop the web application. To be more specific, HTML, CSS and AngularJS are used for front-end design, Node.js is used for back-end design, MongoDB is used as database. Moreover, Nginx reverse proxy/load balancer are applied and Redis cache layer is implemented to improve the performance and scalability of the web application. Finally, the server is deployed on Duke Virtual Machine to be tested and used.

Appendix

GitHub link: <https://github.com/njyuanwx/tinyURL>