Berry Chen (13519441)
Weiqiang Guo (14499622)
Tianzheng Hu (14125617)

Assignment 1 Report
May 13, 2023

UNIVERSITEIT VAN AMSTERDAM

1 Implementation

In this assignment, we aim to develop a URL shortening service with Python flask-restful package[1] which provides convenient ways to build REST APIs. Our implementation is presented in the following sub-sections.

1.1 Specifications

The (id, URL) pairs are stored in a hash table, which is implemented by the Python dictionary. Since the time complexity of reads and inserts of the hash table is O(1), it provides an efficient and scalable way to deal with large volumes of data.

Path & Method	Functionality	Status code and Return Values
/:id & GET	Return URL corresponding to the given id	301, return the mapped URL if id exists 404, if id does not exist
/:id & PUT	Update URL based on the given id	200, if id exists and URL is valid 400, if id exists and URL is invalid 404, if id does not exist
/:id & DELETE	Delete URL based on the given id	204, delete the (id, URL) from the dict 404, if id does not exist
/ & GET	Return all existing (id, URL) pairs	200, return all (id, URL) pairs
/ & POST	Create a new (id, URL) pair	201, if URL is valid and does not exist, return generated id 400, if URL is valid and existed, return corresponding id 400, if URL is invalid
/ & DELETE	/	404, no operation is allowed

Table 1: Two endpoints with HTTP methods, their functionalities, and the corresponding returned values

1.2 URL Checking

Prior to any *POST* or *PUT* operation on the dictionary, it is necessary to perform a check on the correctness of the input URLs prior to creating or updating them. Our implementation follows a two-step process for this verification. Firstly, we use a regular expression (regex) expression to check the format of the input URLs[1]. If the input URL does not match the required format, it is deemed invalid and rejected. Then we use a Python library "requests" to check if the input URLs are accessible. If both conditions are satisfied, we proceed with the POST or PUT operation. However, if either check fails, we reject the request and return the error status code as specified in Table 1.

1.3 ID Generating Algorithm

While implementing the id-generation algorithm, there are two primary objectives: (1) shortening the newly generated ids while preserving their uniqueness and (2) ensuring the scalability of the id-generating algorithm.

To achieve the shortening goal, we chose to use base62 converter [2], which results in a more compact representation than base 10 numbers. It takes an integer as input and returns the id in the type of string. Base62 is widely used in practice for shortening ids since it only uses digits 0-9, uppercase and lowercase letters. A higher base would further shorten the id, but requires a larger character set and require more computation.

Then, to ensure scalability, our algorithm employs an ID_POOL and a counter to keep track of the shortest possible id. The ID_POOL stores freed ids that have the type of string in the min heap structure, where the shortest available ID is always at the root. Whenever a delete operation performs, the corresponding id is stored in the ID_POOL. The counter is represented as an integer in base 10, which only increases by one when the ID_POOL is empty and a valid URL is to be added. An advantage of this approach is that the time complexity of returning the root of the ID_POOL and using the counter is O(1). Thus, it ensures that the shortest available id is always used for new URLs, while also allowing the algorithm to scale efficiently as the number of URLs increases.

When a new URL is given as a parameter in the POST action, the algorithm first checks whether the URL exists in the dictionary, and then the validity of the URL as described in Section 1.2. To assign the shortest id to the URL, the ID_POOL is checked first. If the ID_POOl is empty, the counter increases and is converted into an id using the base62 converter. Otherwise, the root of the ID_POOL is assigned to the given URL.

[1] Reference for checking validity: `https://www.makeuseof.com/regular-expressions-validate-url/`
[2] Reference of base62 converter: `https://stackoverflow.com/questions/742013/how-do-i-create-a-url-shortener`

2 Use Case: Multiple accesses

To address the scenario caused by multiple users accessing a shared URL dictionary, multi-threading can be applied where each thread is responsible for handling one user request. This enables multiple users to access the service simultaneously. However, it could also lead to problems when users attempt to send the same request or conflict requests. For example, when two users send POST requests concurrently with different URLs or when one sends a PUT request to update a specific id while the other user sends a GET id request at the same time.

A potential solution to this is to implement a lock-in mechanism. This suggests that a lock is needed to be acquired whenever a user tries to perform *POST URL*, *GET id*, *PUT id*, or *DELETE id* actions. If the lock is occupied by another user, one needs to wait until the lock is freed before performing any operations. The addition of a lock may negatively impact the service's performance because it restricts access to the URL repository to only one user at a time. However, it is a necessary measure to prevent incorrect results that may occur due to multiple users accessing the repository concurrently. It is important to note that acquiring a lock is not necessarily needed when reading data such as GET operation.

Another feasible solution to the problem of multiple accesses is to utilize concurrent data structures[3]. Such data structures are designed to enable storing and organizing data for access by multiple threads on shared data. In our case, a concurrent hash table[2] could be implemented. It allows multiple users to access the dictionary simultaneously while maintaining the consistency of the data and preventing incorrect results.

3 Bonus Part

We consider the following as a bonus part of this assignment:

1. URL-checking function: In addition to verifying the format of the URL using regular expressions, we also use the Python library "requests" to check the accessibility of the URL. This ensures that only valid and accessible URLs are stored in our system.

2. Lock mechanism: This mechanism guarantees that multiple users can access the repository concurrently without compromising the correctness of the results. To perform any operation on the URL pairs, external users must first acquire the lock. If the lock is currently held by another user, the requesting user will wait until it becomes available before proceeding with any operations.

3. ID_POOL: The use of an ID_POOL as a "free list" guarantees that the shortest possible id is always used whenever needed. Additionally, the min-heap structure of the ID_POOL allows returning the root node(the minimum value) in the time complexity of O(1), which is efficient in general.

4 Assignment Distribution

- Berry Chen: Basic implementation of the HTTP methods; Base62 converter; Report.
- Weiqiang Guo: Implementation of the Lock mechanism and HTTP methods; Code integration; Report.
- Tianzheng Hu: Code testing; Report.

References

[1] Flask-restful: An extension for flask that adds support for quickly building rest apis. https://github.com/flask-restful/flask-restful. 1

[2] Tobias Maier, Peter Sanders, and Roman Dementiev. Concurrent hash tables: Fast and general?(!). *ACM SIGPLAN Notices*, 51(8):1–2, 2016. 2

[3] Concurrent data structures: `https://en.wikipedia.org/wiki/Concurrent_data_structure`