Scalable Image Recognition Application

Jing Qin, Shuo Liao, Tian Yu, Yujue Xiong
GitHub link: https://github.com/jin-qin/cs655-image-recognition.git
Project on GENI: CS-655-Fall2020/GENI-Mini-Project
Public link: http://pcvm3-23.instageni.cenic.net:5000

Problem Statement

Description

The general purpose of this project is to implement an image recognition service(**scalable**) with a web interface. In our design, we plan to implement a server node which manages all jobs and schedules them to worker nodes. Users will use the web interface to submit jobs to the service node and get results. In worker nodes, we will implement a worker service to interact with the server node and image recognition component.

Motivation and learning outcomes

- We can learn how to :
    - build a **scalable, stable ,fast, easy** to use image recognition system with **Restful APIs**
    - use **NodeJS** to build a http service, handle different types of requests(GET, POST(with binary file), DELETE etc.)
    - build the **responsive** web interface as a **Single Page Application** (SPA) by using **React.js**.
    - deploy a deep learning model.
- We can get familiar with **TypeScript** by using it to build all the JavaScript applications.

Main Features

- Fully scalable system, you can add or remove any worker nodes at any time, the system keeps stable.
- Restful APIs.
- SPA implementation for the web interface.
- Fast prediction, benefit of using MobileNet.
- Automatically reschedule a job if its worker is down.

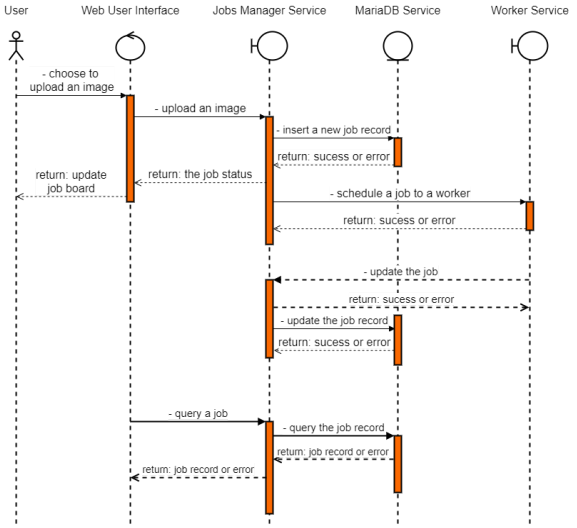
Experimental Methodology

Assumptions

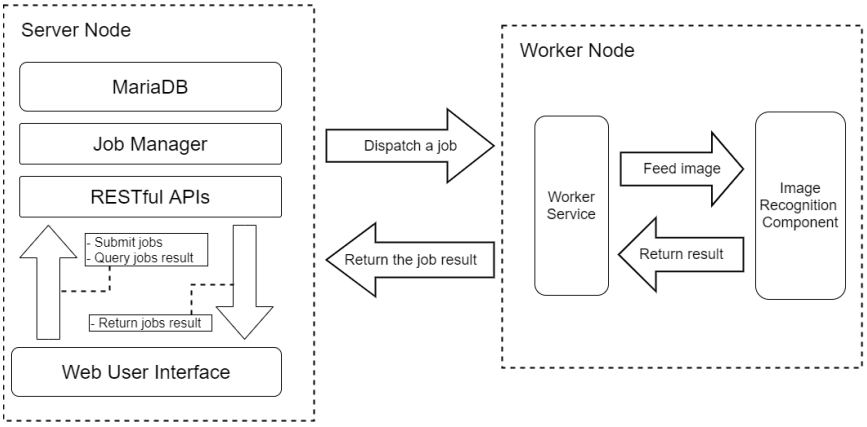
- Maximum Query Per Second (QPS) -- 100 QPS
- Users cannot attack our network, neither adding traffic nor disable the network.
- Users will not do stress testing on our APIs, they can only access the service from the web front-end user interface.

Testing

- Prepare a big image data set: ILSVRC2012 validation images (first 100 images) since this data set cannot be published, so the download link here can only be accessible by Boston University members, log in your BU account to get it.
- Collect the statistics for all images
- Try different loss rates



Sequence Diagram



Architecture Diagram

Result

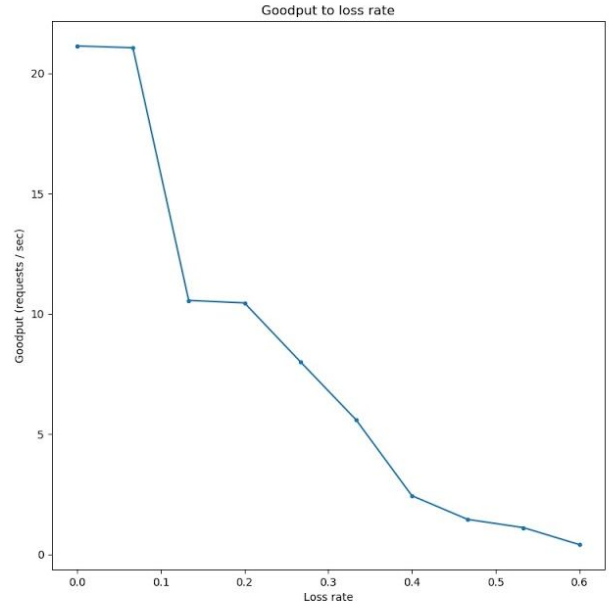
Usage Instructions

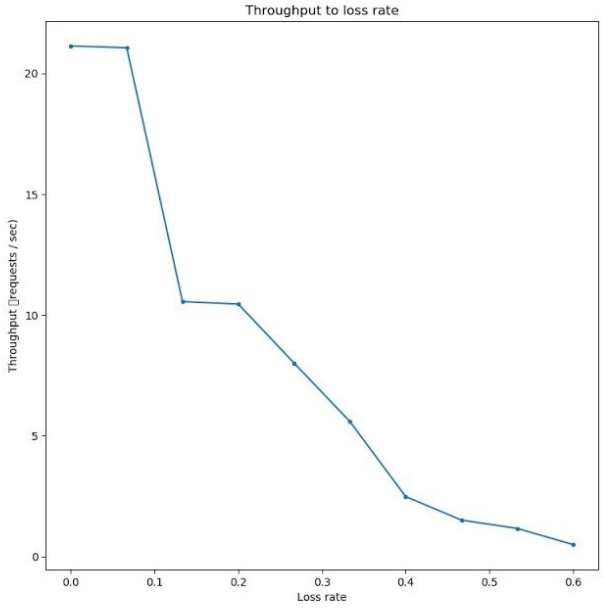
Web interface:

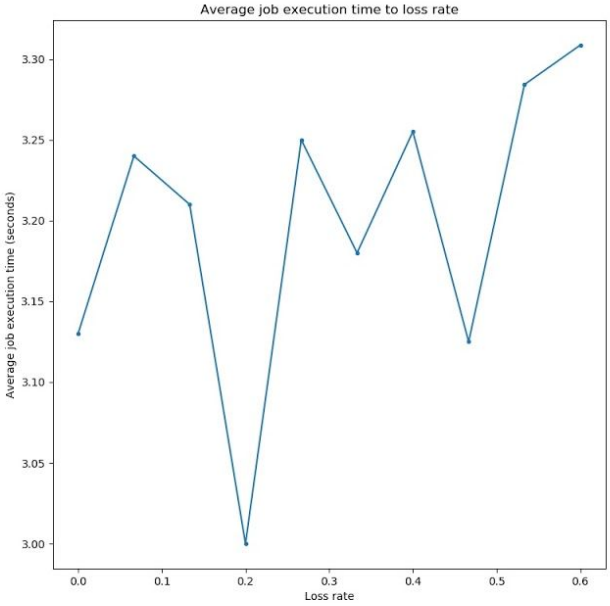
- Open your browser
- Type in the master server's address followed by the port number 5000
- You will see a Job Board first:Job Board Screenshot
- You can submit a job by clicking the blue `SUBMIT A NEW JOB` button.
- You can choose an image from your local machine:Job Submit
- Click `SUBMIT(green button as shown on the left)`
- Back to the Job Board and wait for the result

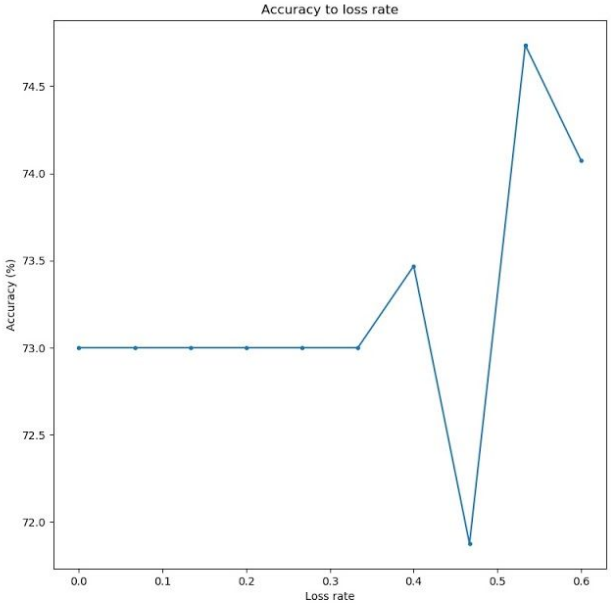
Metrics, graphs and analysis

- **Metrics:**
  - We measured the throughput (number of HTTP requests per second), goodput, accuracy, average job execution time over the loss rates = [0 , 0.067, 0.13, 0.2, 0.27, 0.33, 0.4, 0.47, 0.53, 0.6], and tested with 100 images per loss rate.
  - The loss rates were set on the link between client and the master node.
- **Graphs**:









(a more clear version of graphs: Result graph)

- **Analysis:**

➔ Result table: Result table

➔ According to the first “Goodput-Loss Rate” graph, as shown, goodput (req/sec) decreases as loss rate increases.

- ➔ As the probability of a package being lost, the valid requests asking for image recognition decreases obviously.
- ➔ While the loss rate is between 0.1 and 0.333, the goodput is about 7 requests per second. As the loss rate increases to 0.4 or higher, the goodput keeps around 2 requests per second.
- ➔ The average goodput across 10 tests is 8.223 req/sec.
- ➔ When the loss rate gets larger, there are more packets lost during the transmission and then total time will increase. Thus, the goodput becomes smaller.

- ➔ According to the “Throughput-Loss rate” graph, the trend is almost the same as that of Goodput-Loss Rate but with slight differences in terms of data collected (as shown in table).
    - ➔ The average throughput across 10 tests is 8.249 req/sec.
    - ➔ In our project, we set the timeout of http as 3 seconds. When the loss rate is over 0.4, goodput becomes smaller than throughput, which means under this condition, there exists a timeout http request. Thus, goodput will be smaller than throughput.
- ➔ According to the “Average Job Execution Time-Loss Rate” graph, it is an up-and-down line but the average execution time for each job is between 3.00 and 3.32 seconds.
    - ➔ The average job execution time over 10 tests is 8.3.200 seconds.
- ➔ According to the “Accuracy-Loss Rate” graph, accuracy doesn’t show a significant dependency on the loss rates.
    - ➔ it shows that the line is horizontal when the loss rate is between 0.0 and 0.333, and the accuracy keeps at 73.0%.
    - ➔ As the loss rate increases, the line is up-and-down between 71.9% and 74.7%.
    - ➔ The average accuracy over 10 tests is 73.22%.

Conclusion and extensions

- ➔ Designing a fully scalable image recognition system is challenging since we have to tackle a lot of error status, such as the workers randomly down, and the master need to reschedule those failed jobs and so on.
- ➔ The validation image data set is not easy to get, and we have paid a lot of time on finding the suitable image data set. Besides, the labels inside the image data set are not corresponding to the labels generated by the model, so we also spend a lot of time to find the label mapping file and do the transforming.
- ➔ Loss rate does significantly influence the trend of goodput and throughput because if some job got lost, the valid request or total requests for image recognition during a time period will decrease.
- ➔ The change in the loss rante doesn’t significantly affect the trend of average job execution time because the job execution time is calculated by finish time - schedule time, where schedule time is the time that this job is executed by the worker, this time is not affected by the loss rate.
- ➔ Due to the limitations of time, we did not develop the function of uploading multiple jobs, which can optimize user’s experience. Also, we would consider the optimization of security circumstances and admin dashboard to display statistics.
- ➔ More possible extensions can be viewed in the README.md file.

Division of Labor

- ❏ Server (Jing)
    - ❏ Front-end web user interface
    - ❏ Back-end job manager
    - ❏ Back-end RESTful APIs
    - ❏ Database design
- ❏ Worker (Shuo)
    - ❏ Worker service
        - ❏ Worker registration
        - ❏ Callback to server
    - ❏ Model deployment
- ❏ Testing (Tian, Jing, Shuo,Yujue)
    - ❏ Get the testing data set
    - ❏ Write scripts to test all the images automatically
    - ❏ Gather testing results
- ❏ Data visualization(Jing, Shuo)
- ❏ Project report (Tian, Shuo, Jing, Yujue)