Googloo – SD Final Report

1. Introduction

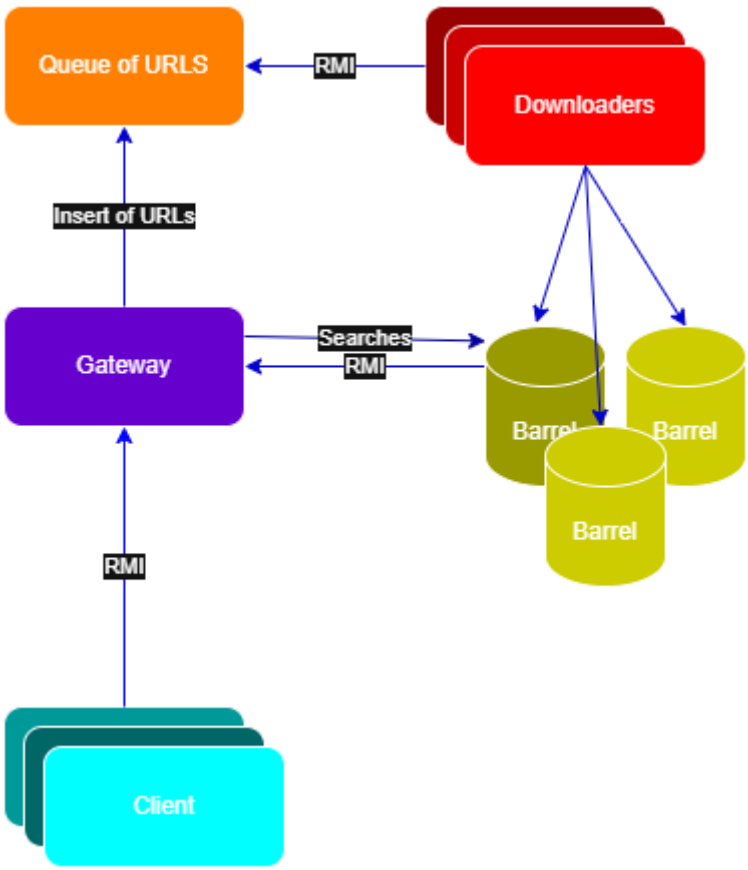
The goal of this project is to develop a search engine with functionalities inspired by a widely used platform “Google”. To achieve this, we divided the system into two main components: backend and frontend.

Backend: Implements the main core of the system including functionalities such as indexing, searching, data management and system statistics.

Frontend: Provides a user-friendly interface for clients to interact with the backend components.

The system uses Remote Method Invocation (RMI) for communication between distributed components, ensuring modularity and scalability. We also use REST to access external APIs such as “Hackernews top stories” and “Openroute”.

2. System Architecture



Backend Software Architecture

2.1 Gateway

Acts as the central controller, coordinating between the Queue, Downloaders, and Barrels. It handles most of the system's logic and is the primary point of contact for clients.

2.2 Barrels

Serve as storage units where indexed data is saved. Each barrel is a separate SQLite database. It is to note that they operate independently and do not communicate directly with each other, the communication and coordination is done through gateway.

2.3 Downloaders

Downloaders act as workers that:

- Retrieve URLs from the queue.
- Download and index the corresponding web pages.
- Forward the processed data to all active barrels using reliable multicast protocol.

For more clarification and a better explanation we decided to explain each component in detail:

Gateway Component

The `Gateway` acts as the **central interface** in a Java RMI-based distributed system, connecting with:

- **Barrel databases**
- A **Queue** (which stores URLs for crawling)
- **Clients** that interact with the system and get real-time system statistics

Key Responsibilities

1. Initialization

- Loads configuration from properties files.
- Connects to the **Queue** and all **active Barrels** via RMI.
- Registers itself on the RMI registry using data from `gateway.properties`.

2. Search Functionality

- **Distributes search queries** to a random Barrel.
- Barrels **update their internal top-word stats** before search.
- Tracks and averages **response times per Barrel**.
- Updates **top searches** and Barrel sizes.
- Notifies subscribed statistics listeners of changes.

3. Queue Interaction

- `insertURL(String url)`: Adds a URL to the crawl queue.
- `addFirst(String url)`: Adds a URL with priority to the front of the queue.

4. Statistics

- Maintains a `SystemStatistics` object:
  - Top search terms
  - Average response time per barrel
  - Active Barrel index sizes
- Provides real-time updates to all **registered listeners**.

5. Connection Retrieval

- `getConnections(String url)`: Requests backlink/connection data for a URL from Barrels.
- Fallback strategy mirrors that of search.

6. Barrel Management

- Detects and connects to Barrels dynamically using properties.
- Supports **manual registration** via `registarBarrel(String rmi)`.
- Maintains `activeBarrels` and response tracking via `BarrelStats`.

7.Barrel Selection Logic

- Randomly selects a Barrel during initialization or refresh.
- If the selected one fails during an operation, tries others in order.

8.RMI Details

- Gateway and all other services (Queue, Barrel) are exposed via RMI.
- Uses `Naming.lookup()` and `Naming.rebind()` to connect/register services.

9.Configuration Files

- `gateway.properties`: Gateway RMI & API key
- `queue.properties`: Queue RMI settings
- `barrel.properties`: List of Barrel RMI entries.

Downloader Component

The Downloader serves as the web crawler and indexer, bridging:

- Queue (gets URLs to process)
- Barrels (sends parsed data)

They also act as worker nodes that:

- Fetch URLs from the Queue.
- Download and parse web pages (using Jsoup).
- Distribute indexed data to all active Barrels using reliable multicast protocol.

Key Features:

-URL Processing:

- Extracts title, text, and links from HTML.
- Normalizes words (removes accents, punctuation) and filters stopwords that are managed in Queue.

-Fault Tolerance:

- Detects and handles failed Barrels (unregisters them via Gateway).
- Retries failed URLs by re-adding them to the Queue.

-Dynamic Stopwords:

- Regularly syncs stopwords from Barrels to optimize indexing.

-RMI Interactions

- Queue: getURL(), addURL(), addStopWords().
- Gateway: getBarrels(), unregisterBarrel().
- Barrels: addToIndex(), containsUrl(), getFrequentWords().

-Threading Model

- Multi-threaded: Parallel processing of URLs across Downloaders.
- Scheduled Task: Updates stopwords in dynamic intervals of time.

-Configuration

Reads RMI endpoints from:

- gateway.properties (Gateway connection).
- queue.properties (Queue connection).

-Error Handling

- Skips locked Barrels (SQLite contention).
- Logs failures when URL cannot be processed because some URLs need authentication or dont exist or.... (RemoteException, parsing errors).

Queue Component

The Queue acts as the URL dispatcher in the Gloogloo system, managing the workflow between:

- Gateway (receives new URLs from clients)
- Downloaders (consumes URLs for processing and get stopwords from a downloader and barrel)

-Central URL buffer that:

- Stores pending URLs for Downloaders to process.
- Manages stopwords (frequent words to ignore during indexing).

Key Features:

RMI Interface:

- addURL(): Adds URLs to the queue.
- getURL(): Retrieves the next URL for processing.
- addStopWords()/getStopwords(): Syncs stopwords across components.

-Persistence:

- Uses in-memory storage using the TXT format to store URLs and Stopwords.

-Load Balancing:

- Distributes URLs evenly to Downloaders using a Queue data structure.

-Configuration

- Reads settings from queue.properties:

-RMI Setup

- Starts RMI registry on specified port.
- Binds service at rmi://[host]:[port]/QueueService.

-Error Handling

- Logs failures (e.g., RMI registry errors, config file issues).

Barrel Component

The Barrel acts as a distributed storage node in the Googloo system, responsible for:

- Storing indexed web content (words, URLs, links,snippet,titles) in SQLite databases.
- Handling search queries from the Gateway
- Providing frequent words (stopwords)

Key Interactions

1. With Gateway:
   - Registers itself via registarBarrel() on startup
   - Receives search queries (query(), getConnections())
2. With Downloaders:
   - Accepts indexed data via addToIndex()
   - Shares stopwords via getFrequentWords()

Key Features:

-Data Storage:

- Each barrel maintains its own SQLite database
- Tables for words (top searched words), URL links (relationships between URLs), URLs (information regarding the title and ranking) and word URL (inverted index)

-Fault Tolerance:

- Unregisters automatically if crashes (via Gateway)

Configuration

Reads from barrel.properties:

-RMI Setup

1. Creates dedicated RMI registry per barrel
2. Binds service at rmi://[host]:[port]/BarrelService[1|2]

3. Requirements

To run the application, the client must have the following installed and configured:

- Maven
- SQLite
- JDK 22
- Gson, Unirest, Jsoup: Present in imports.
- org.json: Also included for JSON handling.
- SpringBoot

Additionally, all necessary packages and dependencies must be correctly configured.

4. Functionalities

4.1 Insert and Index URL

Clients can input a URL through the frontend, which is then:

1. Sent to the Gateway.
2. Queued for processing.
3. Picked up by a Downloader.
4. Indexed and stored in all active Barrels.

4.2 Keywords Search

Clients search for keywords (single or multiple).

1. Routes the query to the Gateway.
2. Queries a random Barrel for indexed terms.
3. Returns URLs/titles/snippet most relevant to the searched terms.

4.3 Related URL Search

Like the keywords search, but:

- Clients input a URL.
- The system returns other URLs most related to it based on indexing.

4.4 Administrative Interface

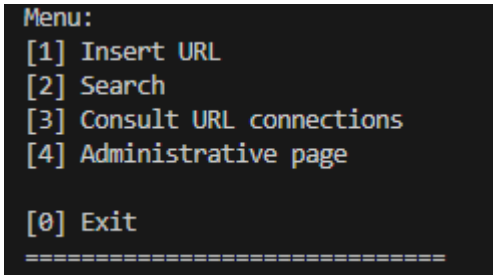
Provides statistics and metadata related to:

- System's performance and search activities.
- It includes lists of searches, barrel sizes, and response times that can be used for analysis or reporting.

5. Execution Instructions (backend first part)

1. Having configured correctly the Ip and necessary ports in the properties files of each component (Gateway,Queue,Barrels)
2. Then using the command terminal, we will go to the directories of
   - ...\Gloogloo\common
   - ...\Gloogloo\backend
3. And we will execute the next command: “mvn clean istall -Dskiptest” in each of the mentioned directories.
4. Afterwards the user must run the following files:
   - QueueServer.java
   - Gateway.java
   - BarrelServer.java
   - Barrel2Server.java
   - Downloader.java
   - Main.java
5. Finally, the user must go to the following directory “...\Gloogloo\frontend”and execute the following command “mvn spring-boot:run -DskipTests”

Once active, the client will be presented with a menu with the folowing format:



6. Testing

We provide the following table as confirmation of the tests we have done to make sure of the correct functioning of our code.

Description	Pass/Fail	Results and outputs
Client inserts entry that does not exist in the menu.	Pass	This message appears “Invalid input. Please choose a valid option” indicating the client to retry using valid inputs.
Client connects without having the Gateway connected.	Pass	This message appears “Failed to connect to Gateway” indicating to client to run the gateway.
Downloader connects without having Gateway connected.	Pass	This message appears “Failed to connect to Gateway” indicating to client to run the gateway.
Barrels connecting without Gateway connected.	Pass	This message appears “Failed to connect to Gateway” indicating to client to run the gateway.
Barrel breaks down, recover status?	Pass	After the shutdown of one of the barrels the downloaders continue working and sending data to the active barrel. After the detection of the shutdown barrel turning back on, the system detects it and proceeds to synchronize the data between barrels.
Load balancing on storage barrel searches Is the information identical across all storage barrels?	Pass	The downloaders send all the information that they possess to all the barrels using reliable multicast protocol. This will guarantee that the

		information across all barrels is the same. From the client side the access to barrels is randomized to guarantee that none of the barrels has to answer to all clients at the same time to reduce the load put on each barrel.
Are indexing requests only answered by one downloader?	Pass	The URLs in queue can be processed distributedly and in parallel using threads.

7. Work Distribution

Development: All team members focused on implementing the backend and frontend functionalities Pedro worked more on implementing the frontend side and Main.java client while Francisca implemented Queue, Downloaders and Barrels implementation logic. Ramyad worked like a fireman while debugging issues in all components of the project while mainly focusing on the documentation part regarding the Final report and the Javadoc documentation and a README file. All members stayed aligned through regular communication and feedback sessions.

8. Documentation

In addition to this final report, we also provide a Javadoc document to guide users in case of any questions regarding the Classes and Interfaces. The document has detailed explanations regarding the Classes and Interfaces. To access the java doc documentation the user must go to the following directory "...\Googloo\docs\apidocs" and open the "allclasses-index.html". In addition there is a README file in the directory:

"...\Googloo"

Instructions

To run the full application the next instructions must be followed:

1. Having configured correctly the Ip and necessary ports in the properties files of each component (Gateway,Queue,Barrels)

2. Then using the command terminal, we will go to the directories of
   - ...\Gloogloo\common
   - ...\Gloogloo\backend
   - ...\Gloogloo\frontend
   - ...\Gloogloo
     And we will execute the next command: “mvn clean istall -Dskiptest” in each of the mentioned directories.

3. Afterwards the user must run the following files:

   - QueueServer.java
   - Gateway.java
   - BarrelServer.java
   - Barrel2Server.java
   - Downloader.java

4. Finally, the user must go to the following directory “...\Gloogloo\frontend”and execute the following command “mvn spring-boot:run -DskipTests”

The End