Distributed Systems (521290S)
Course Project Report
Team Tesla

Member	Student ID	Email	Track
Md Mobusshar Islam	2305578	mislam23@student.oulu.fi	Corporate
Muhammad Ahmed	2304796	mahmed23@student.oulu.fi	Corporate
Muhammad Talha Arshad	2304797	Muhammad.arshad@student.oulu.fi	Corporate

Course project overview

Overview

The project aims to process the request to scrape the data from websites according to the instructions of the requesting nodes (users) and extract useful/required information from the data. The following are the major components of the system.

Distributed System Topics Covered

- **Message Queuing System**:
  - Round Robin Queuing:

    Distributes incoming requests evenly among available server resources in a circular manner.
  - Priority Queue:

    Prioritizes requests based on urgency or importance, ensuring that high-priority requests are processed first (In case of having paid customers in the request list)
- **Load Balancing:**
  - Round Robin Load Balancing:

    Distributes incoming network traffic or requests evenly across multiple servers.
  - Weighted Round Robin:

Assigns different weights to servers based on their capacities, allowing for more efficient resource utilization.

- **Priority Queue Management:**
  - Heap-based Priority Queue:

    Utilizes a heap data structure to efficiently manage and extract the highest-priority requests.
  - Weighted Priority Queue:

    Assigns different weights to requests based on their criticality, influencing the order in which they are processed.
- **Resource Management:**
  - Token Bucket Algorithm:

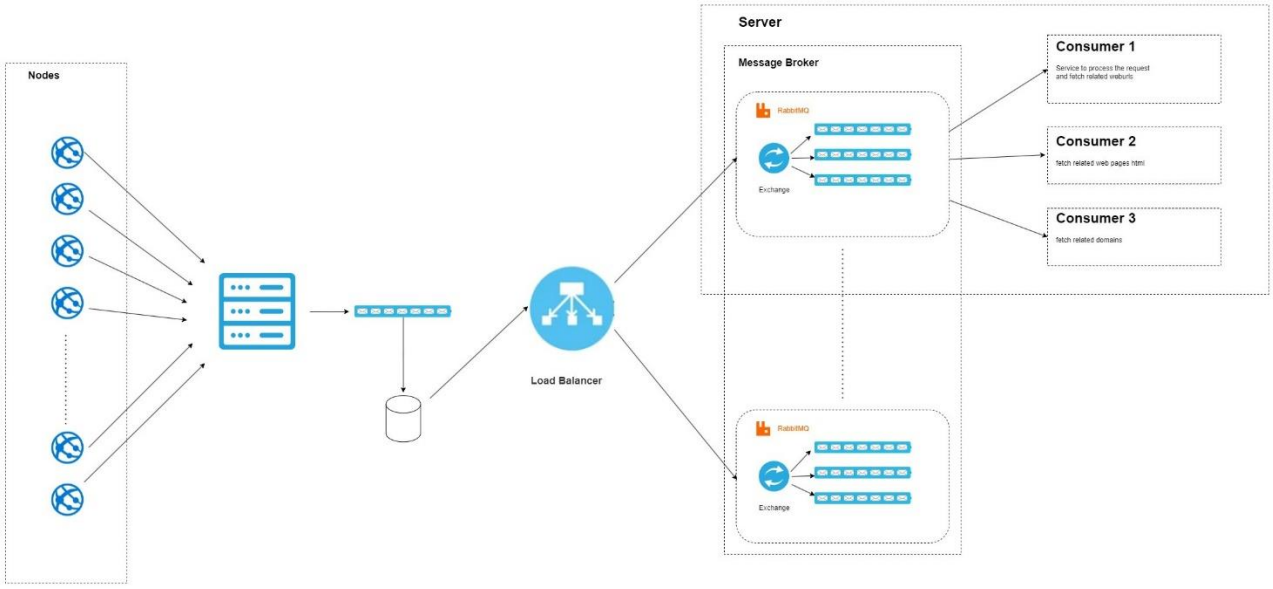
    Controls the rate at which requests are processed to prevent resource exhaustion.
  - Leaky Bucket Algorithm:

    Smooth outbursts of incoming requests, preventing sudden spikes in resource usage.

- **Fault Tolerance and Redundancy:**

In the above-mentioned basic topics covered, we will be trying different approaches to find out the best possible solutions for a specific problem.

Architecture Diagram



1. Architecture

In our proposed **Event-Driven Architecture (EDA) style**, Nodes will be sending the requests to the server and here load balancer will assign the requests to the **message queues** of the servers based on different distributed system load-balancing algorithms in this case, the **load balancer** will act as publisher and different consumers will subscribe to the respective queue maintained by the **RabbitMQ message broker**. Components in the system are **loosely coupled** as they interact with others through events rather than direct connections. Nodes having a connection with the central server are making client-server architecture in the system.

Following are **the roles of the components** of the system.

- **Requesting Nodes:**

Different nodes in the system will request the data to extract the data and, in this project, we will be mocking the role of nodes by some utility/script to generate a good number of requests.

- **Central Server:**

This component of the system will receive the requests from the nodes and return them a response that your request has been accepted and will be processed soon. It will put all the receiving messages in a message queue and all the requests will be registered in the database.

- **Load Balancer:**

This part of the system will pick the requests from the database and will assign them to different servers that are part of the distributed system to process the requests based on the availability of the server.

- **Message Broker**

We are implementing RabbitMQ as a message broker. This will use exchange to assign the requests inside a server to different queues based on the different types of consumers.

- **Consumers**

Based on different jobs to perform in our case, we have different consumers as follows

1) Fetching related text data from the website
2) Fetch related web pages HTML.
3) Fetch related domains.

2. Implementation

Describe how you implemented the functionality for each system component (e.g. client and server), including software solutions and possible hardware.

Advice: For the design, is it already beneficial to think about the implementation of the components. You save a lot of iterations in development, if you, at this point take the design seriously. For example: what are the components' internal architectures, how components are virtualized, what software is needed to realize functionalities, etc?

3. Communication

Design: Describe how the components in your distributed system interact to implement the application, and possible communicate with system-level services.

It pays to describe the following:

- Interaction pattern, e.g. publish / subscribe
- Application-level protocol (if applicable), i.e. the messages / events you use in your application
- Interfaces, e.g. RPC or REST API
- Communication protocol (stack) you used in the project

For example, you are using HTTP atop TCP/IP and the RESTful interface is the following / With MQTT, these events are published..

4. Naming

Design: Describe briefly how the components in your distributed system identify and/or discover other components and resources. For example, is your system using flat / structured /attribute-based naming? What kind address + name scheme you implement? How do peers know their neighbors? etc..

5. Coordination

Design: If your project implements a synchronization / coordination / election / etc scheme or uses a such protocol, describe briefly your solution here. Refer back to section 1 and describe the components and their roles and the utilized algorithm (i.e. election / gossip / etc) in more detail. Again, a picture is worth thousand words, see examples in the course book.

6. Consistency and replication

Design: If your project follows a data- or client-centric consistency model or implements replica management / protocol or such a scheme, describe it here briefly. Refer back to section 1 and describe the components and their roles in more detail. Again, a picture is worth thousand words, see examples in the course book.

7. Fault tolerance

Design: If your project implements a fault detection or tolerance mechanism, solution for reliable communication or distributed commit protocol describe it here briefly. Refer back to section 1 and describe the components and their roles in more detail. Again, a picture is worth thousand words, see examples in the course book.

8. Security

Design: If your project implements security, authentication, authorization mechanism, describe it here briefly. Refer back to section 1 and describe the components and their roles in more detail. Again, a picture is worth thousand words, see examples in the course book.

Evaluation

A table with numeric data speaks thousand words. Include a (very) brief analysis of the evaluation results.. what can be seen, is something missing, etc.

Advice: Already, for the design, is it necessary to think of the evaluation you planned for your project. This way you will have a “placeholders” built-in into your system. enabling easy evaluation once you are at this stage. Think of the test cases and what and how data is collected in your system, do you use a logging tool, etc. General rule is “everything is related to everything”..

Workload distribution

Plan of sharing the workload (wl)and estimated hours. Please fill in the real calculated workload only on the final submission of the report.

Student name	Tasks	Estimated wl. (h)	Real wl. (h)

References

In case you utilized existing work or software in your project, please list the sources here.

Design: You should already have some ideas which (existing) software you are using in your project..