Project Report

Prepared for: Distributed Random Generation of Quiz, Distributed Systems - CS632A

Prepared by: Somesh Kumar Jaishwal , 18111071, MTech, CSE 2018-2020

Naman Narang , 18111043 , MTech, CSE 2018-2020

Supervision: Dr. R.K. Ghosh

Project Group : 22

INTRODUCTION

The goal of our model is to generate MCQ based different sets of question paper for different students. In our approach, we sport distributed servers and organisation of questions to facilitate high availability. Challenges including consistency of databases at multiple servers, load balancing and fault tolerance of servers are also taken care of.

Features

- An instructor can add multiple MCQs, its options, correct option, its marks along with level of difficulties (easy, medium and hard). Instructor will get an access code that represents the questions added by instructor and contains rules associated with it.

- Students need an access code (that will be provided by the instructor) to generate question paper. Each student will exercise different set of quiz. Number of questions in each difficulty level is specified by the instructor in access code as an access code rule.

- Evaluation of quiz is also featured.

PROJECT OUTLINE

1. Data Organisation

MCQ questions are organised in a single table at a server. Schema of the table is

id	question	opt_a	opt_b	opt_c	opt_d	correct_opt	weight	access_code

Access Rules for access code are organised in another table. Schema of the table is

access_code	easy_count	medium_count	hard_count

2. Backend Features

- **Distributed Servers**

  1. **Independent Consistent Servers:**
     Database at independent servers are being synchronised using **RMI**. A server before adding data to its own database invokes the remote method of the others servers for adding the same data into their respective database. This technique helps us to maintain a consistent replicated database among servers.

  2. **Fault Tolerant System:**
     When a server becomes active again after crashing, it adds questions that were added to other servers when it was down. When a server gets a request to add questions, before adding it to the local database it tries to send the same data to the other servers. If one of the remote servers is down then the server sending data to another will figure it out and maintain a log of data that couldn't be sent. When the remote server turns active it requests other servers to send again the unsent data. When the server sends the unsent data it clears its log corresponding to that server. This technique is also implemented using **RMI.**

- **Load Balancing**

  We have used Nginx as an HTTP load balancer to distribute traffic among multiple application servers. Of the three simple load balancing techniques that are, Round Robin, Least Connected and IP Hash, we have used IP Hash for our purpose as it allows a user remain connected to the same server. IP Hash load balancing technique helped us imitate real world geography based user-server connectivity, for example, if a user is in India, he connects with *.co.in domain of a service (if it is available) rather than connecting with a distant *.com domain of the same service (that may be located in USA).

3. Analysis and Results

1. Analysis on number of Instructors that can be simultaneously submit their question batch:

Our model uses four servers and uses four threads to distribute one batch of questions from one server to rest three. The flow goes as follows :

On submission of one batch of questions by an instructor to a server A, server A initiates a thread that distributes the same batch data to the remaining three servers on three separate threads.

We also check the thread threshold (in terms of number of threads that can be created) on personal computer that hosted the server. With configuration of 3GHz processor, 8GB RAM and 4GB swap, it was able to create approximately 3000 threads simultaneously.

Thread threshold being around 3000 and 4 threads contributing to distribution, **around** (3000/4) ~ **750 instructors can submit their questions batch simultaneously***.

** provided machine to be dedicated for the purpose*

2. Analysis on number of Students that can request quiz simultaneously.

Suppose the database has total **N** questions and total number of questions under an access code be **Z**. Also, if we consider the rules of the same access code specifying **e** as the number of easy questions, **m** as the number of medium difficulty questions, and **h** as the number of hard questions to be included in the quiz, we can see from figure 3.2.1 that for 1 process, to generate questions for associated quiz it takes :

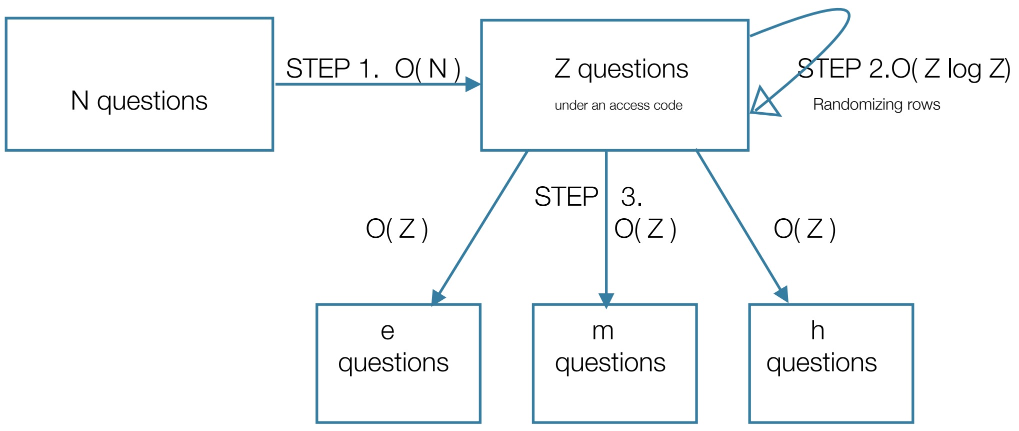


Figure 3.2.1

a) O(N + Z log Z + Z) , if we do STEP 3 that is, reading e, m, h questions parallelly from Z questions.

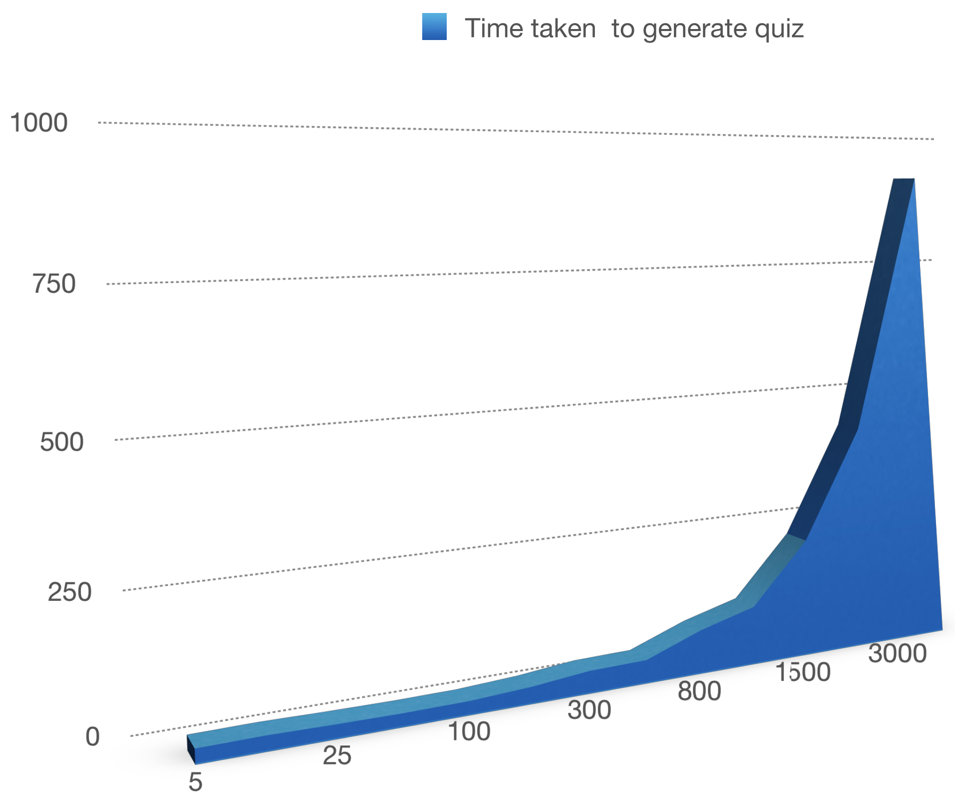
b) O(N + Z log Z + Z + Z + Z) , if we operations in STEP 3 sequentially.

In our model, we have used approach (b) because it does not utilise extra processes in STEP 3. These processes now can serve extra students with their question paper. Also, there is a very negligible difference between performance in above two approaches. As N increases Z becomes relatively quite small, hence time taken will become O(N) in both approaches.

The system hosting the server is capable of creating maximum 700 user processes (testing the server in user space) and 1 process as we saw, can serve a student with the quiz. Hence, **around 700 students can request a quiz simultaneously *.**

** provided machine to be dedicated for the purpose*

3. **Analysis on time to generate quiz when question count in quiz increases.**



Graph 3.3.1

Graph 3.3.1 shows the variation in time taken (in ms on Y axis) to create quiz as the number of questions (on X axis) in the quiz increase. The time shown does not include transmission delay but includes time for generation of html from questions.

4. Limitations

- Users do not require User IDs.
- All the questions under one access code are needed to be added in one session.
- Given a server A is down, if a server B also turns down, server B needs to revive first or before we revive server A (Limitation of Fault Tolerance in our model)

5. Language and Support used

- Java SE8
- Java RMI, Threads, Jooby
- Mariadb Database
- Maven Project Structure, Intellij as IDE
- Jetty as Application Server
- Nginx as Load Balancer
- Configuration of system used for development, testing and analysis of servers :
    - 3GHz processor
    - 8 GB RAM
    - Thread Threshold $\simeq$ 3000
    - Maximum Number of Process that can be created $\simeq$ 700.

6. Future Work

1. Current work can be extended to eliminate independent server that is, no server will have the complete data. When a request comes for a data, in case which is not present on that server, the server will then redirect the request to the another server that contains the needed data (for quiz generation in our case). The server figures out the right server for a request from access code rules tables as it now contains servers' address too.

2. Limitation of Fault Tolerance in our model can be overcome as : If server B, before sending a sync request to server A, checks if it has unclear log of unsent data for A. If it has the unclear log, it sends the unsent data to server A first, then send request for synchronisation to it.

3. More type of questions can be added. They can be organised the same way as MCQs that is, one table for one type of question.

7.References

1. https://jooby.org/apidocs/org/jooby/request
2. https://jooby.org/apidocs/org/jooby/response
3. https://docs.oracle.com/javase/8/docs/api/java/net/HttpCookie.html
4. https://docs.oracle.com/javase/8/docs/api/java/rmi/server/UnicastRemoteObject.html
5. https://www.w3schools.com/xml/ajax_intro.asp