**CS 6314, Fall 2017**

**Dr. Mithun Balakrishna**

**Homework 1**

1.

Yes I would design a web applicaiton using service-oriented architecture(SOA) for my customer to perform 4 different operations whose system resurces are same and have similar load on the server.

Consider a library management web server application of a univeristy that hosts the database of entire library where a student can perform 4 different operations :

* Ordering a new book.
* Blocking a book.
* List the books available.
* Restore a book.

In SOA, a system must consider availability,reliability, scalability, performance, manageabilty and cost.

In this library management system, the request handler takes care of requests from the customer like ordering , blocking a book or lisitng the books and restoring the book and tries to look out for the necessary action as needed.

For ordering a book, it initially takes request from customer, searches for the book in the database and in the eve of unavailablity of the book it orders the book and acknowledges the same to the customer.

While blocking a book, the request handler checks the global cache for availability of the book, if founded it throws the same result to customer else it searches for the book and acknowledges as necessary. Thus the global cache is also updated at the sametime.

In order to find the list of books available in the database through the web application the global cache might have necessary information about the availability of books such that it can list all the books also checks the entire list of books in database, keeps a count and stores in cache to maintain manageability and cost instead of counting all books available everytime. Then the list of books is retrieved from database for the first time and updated in the cache from every next operation.

In order to restore a book by a customer, the request handler takes request and then updates the cache with the book details like availability, name etc. and then in database so that anytime soon the customer can find the book availability soon reducing cost.

* There is no limit to the number of books that will be stored, so storage scalability, in terms of book count needs to be considered.
* There needs to be low latency for book downloads/requests.
* If a user requests a book, the info about book should always be there (data reliability).
* The system should be easy to maintain (manageability).

All the operations have similar load on server, using cache maintains performance, cache updates about availability, data updation of books maintains reliability.

2

In a shared nothing architecture, each node is able to operate independently without having a separate maintanance monitoring activities of other nodes. That way when it comes to horizontal scalability, new nodes can be a added to the database without any special conditions instead of deploying bigger CPU for a single large dataset. Redundancy helps to maintain the dataset incase of any failure while doing operations. Creating redundancy in a system can remove single points of failure and provide a backup or spare functionality if needed in a crisis. There is no exchange between each of the nodes, except some combined result in the end.

For example, let us consider a distributed computing i.e.; parallel computing of creation of new email id in a web application.

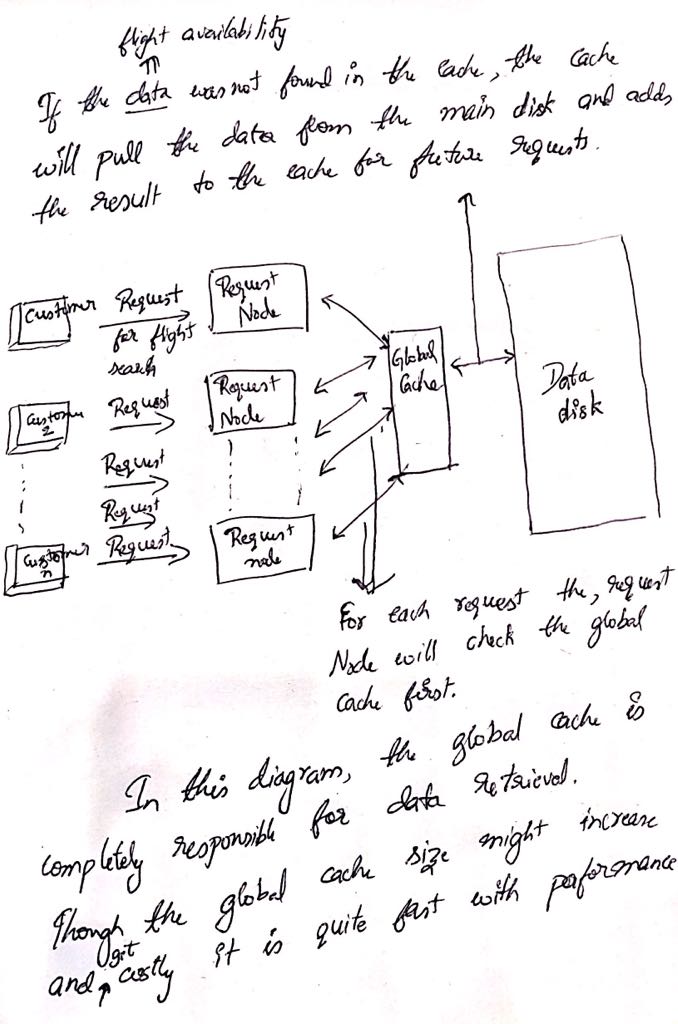
At the basic level, for a user when creates a email id, a new email id is added to a local node and as number of users eventually increase; So a new node must be created to maintian the incoming load and is called horizontal scalability.

Each node is independent to the other nodes, any email/ping from one email id doesn’t effect the reamaining because each node has its own memory and operations. Multiple email ids can be active and send/receive emails proving parallel computing. Redundant email ids in nodes avoid the failure action of one email id with another(mail send/receive/ping) .

3.

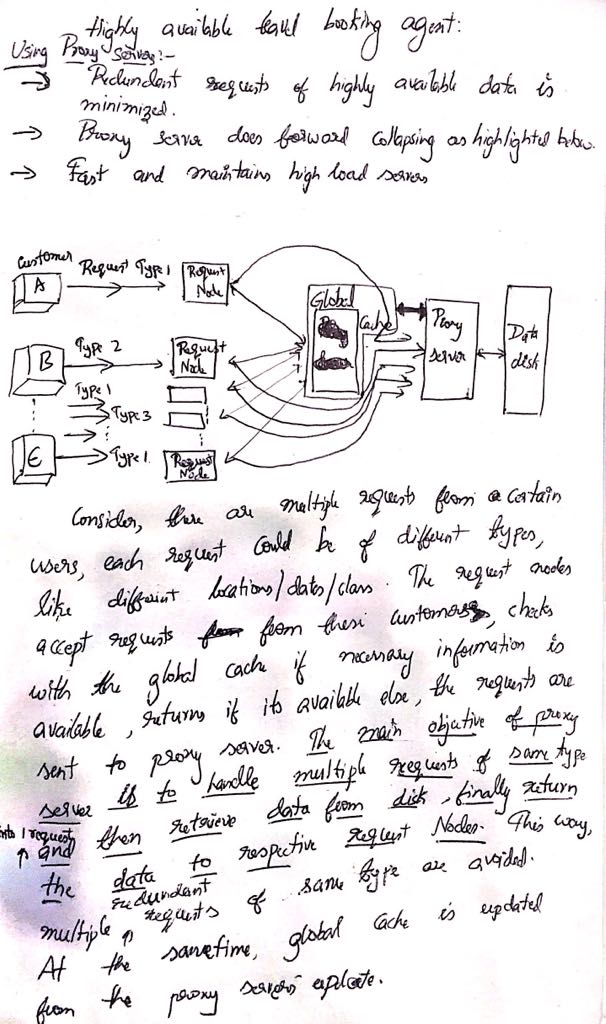
Services to be considered stepwise :

* **Very Fast flight searches**: For the client to have fast flight searches, consider using Global Cache in Service Oriented Architecture. Since in a global cache, the number of requests can be handled easility using a single global cache available for the entire server. Searching for a flight with all the requirements(between places/dates/type) is considered to be a request here and the following diagram depicts the design:

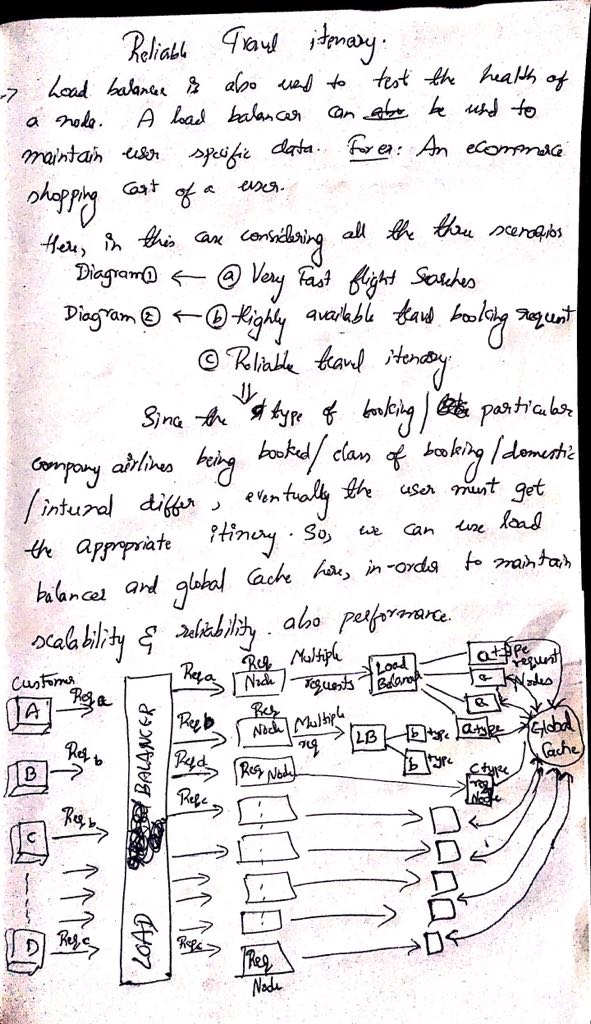


* **Highly available travel booking request**

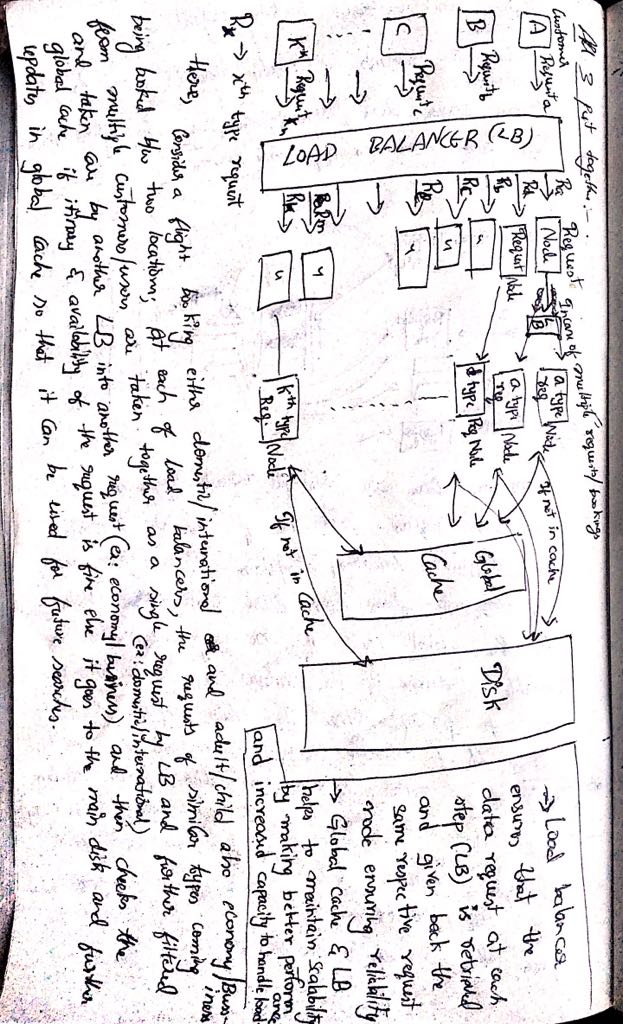
All the assumptions are mentioned in the picture attached below:



* **Reliable travel itenary:**



**Considering all the above 3 steps as listed and to maintian the reliabiliaity and scalability, all the three put together below is the final design:**



4.

HTTP Request methods with example:

**HTTP GET:** It is used while to retrieve any data without causing any other effect (like to) take any action in the web-application. It simply gives the representation of a particular resource requested. It shows whatever information is available in the requested URI.

**Example:**

**GET** Retrieve a resource.

**Request**:

GET /foo/bar HTTP/1.1

Host: example.com

**Response**:

HTTP/1.1 200 OK

Date: …

Content-Type: text/html;charset=utf-8

Content-Length: 12345

<!DOCTYPE …

**HTTP HEAD:** The response received on this request is similar to the GET request except that it doenst contain the request body. Here the metainformation in the HTTP headers in response to HEAD request must be identical to the information sent in response to a GET request. In other words, it’s a get without the body.

Example: **Request:**

HEAD Like GET, but returns just the HTTP header.

Request:

HEAD /foo/bar HTTP/1.1

Host: example.com

**Response**:

HTTP/1.1 200 OK

Date: …

Content-Type: text/html;charset=utf-8

Content-Length: 12345

**HTTP PUT:** This requests that the supplied/attached entity be stored under the supplied request URI. In this case, if requested URI refers to the already existing resource then it should be modeified else a new resource can be created with the supplied URI.

**Example**: **Request**:

PUT /hello.htm HTTP/1.1

User-Agent: MozillaFF/4.0 (compatible; MSIE5.01; Windows NT)

The following example requests the server to save the given entity-boy in hello.htm at the root of the server

Host: www.example.com

Accept-Language: en-USA

Connection: Keep-Alive

Content-type: text/html

Content-Length: 182

<html>

<body>

<h1>Hello, check this word!</h1>

</body>

</html>

The server will store the given entity-body in hello.htm file and will send the following response back to the client:

**Response:**

HTTP/1.1 201 Created

Date: Tue, 21 Sep 2019 09:31:03 CT

Server: Apache/2.2.14 (Win32)

Content-type: text/html

Content-length: 22

Connection: Closed

<html>

<body>

<h1>The file was created.</h1>

</body>

</html>

**HTTP POST**: It requests that the server accept the entity enclosed in the request as a new subordinate of the web resource identified by the URI. The attached URI might contain the resource which has the information of which URI to modify/create whatever as necessary similar to PUT. The data POSTed might be an annotation for existing resources a message for a bulletin board, newsgroup.

**Example**:

**POST** Create a new resource.

Request:

POST /foo/bar HTTP/1.1

Host: example.com

Content-Type: application/x-www-form-urlencoded

**Response:**

action=addentry&subject=Hello,%20World

Response:

HTTP/1.1 201 Created

Date: …

Content-Length: 0

Location: http://example.com/foo/bar