Q1.1: Explain at a high level what technique you would use to map addresses at the single address space to addresses in each individual machine. Are there any centralized components in your design? How scalable is it as we increase the number of machines? What kinds of faults can the design tolerate? (2-3 paragraphs)

* 1. We are using a table to map the addresses with a machine. The table has the titles “Address and Name”. Machines can have different memory sizes this means that that they will require more or less addresses.
  2. The components are centralized on a Server.
  3. In order to scale the design delete and update functions need to be created. The delete function will be used only to eliminate a machine and update will be used in order to update the number of addresses. This is done because the number of addresses is limited.
  4. If a machine goes down temporarily the addresses need to be redistributed. If a machine gets removed the addresses need to be redistributed again. One problem can be is that if the server that assures the mapping goes down then the whole system will fail.

Q1.2: Show the pseudocode and API of the READ and WRITE functions of your memory abstraction. Note that your pseudocode should show both translation of addresses (naming) as well as communication with individual machines to obtain the actual data. (for each function: API + 1-2 sentence description of API + pseudocode)

Read (Address) – read function loads the Address from the table in order to find the corresponding machine; If the address was read correctly a message will be received. You send the address to the machine and receive back the value sent from the machine.

Write (Address, Value) – searches inside the table for the address and finds the respective machine attached to that address; The machine will write the value in the machine memory. In order to find out if the machine works a message will be received to see if the value was written.

Q1.3: Are READ/WRITE operations against regular main memory atomic? Should the READ/WRITE operations against your memory abstraction be atomic? Discuss why or why not. If you believe they should be atomic, argue how you would achieve that property when multiple clients are accessing your abstraction concurrently. If you believe that they should not be atomic, describe what abstraction you would need to provide to clients to implement atomicity when they need it. (1 paragraph)

No, READ/WRITE are atomic only when they meet several conditions. Concurrent readers and writers must be coordinated, and a failure in the middle of an update must leave either all or none of the intended update intact in order for the READ/WRITE functions to be atomic. READ/WRITE operations agains the memory must be atomic, and thus a scheduling plan needs to be done so that it does not starve some clients in favor of others, needs to provide a low turnaround time for each client request and it needs to assure that it has little overhead so that many clients can be served. All thease goals are impossible to be maximised and so trade-offs that favores one class of requests or another.

Q1.4: One important aspect in the design of the naming scheme for this assignment is whether your name mapping strategy makes any assumptions on the number of machines in the cluster. For example, you may consider the number of machines K to be given and known a priori. Alternatively, you may consider that machines may enter and leave the system, and naming must adapt to these changes. Does the name mapping strategy you designed allow for dynamic joins and leaves of machines without making

memory locations unavailable? If so, why? If not, what strategy could you use to achieve that property? (1 paragraph)

Currently it does not support dynamic joins but it updates the addresses and thus making memory locations available.

Q2.1: How does concurrency influence latency in a computer system? Is its influence always positive, always negative, or is there a trade-off? Explain. (2 paragraphs)

Concurrency doesn't always guarantee a decrease in latency, it correlates to how the algorithm could be broken down into several tasks. So even if the task could be broken into n threads or computers that process that data the increase in performance isn't necessarily n fold but less than that, due to the overhead needed to recombine the final result.

And it still depends on the algorithm maybe by making it concurrent unexpected problems occur while separating tasks. The trade-off is always an increase in development and testing time to reduce the concurrency.

Q2.2: Explain the difference between dallying and batching. Provide one example of each. (2 paragraphs)

Dallying is the process of delaying requests in the chance that the initial request is overwritten by the following or that all the requests could be sent together in a process called batching. Batching is the process of sending several requests at a time which saves time instead of sending them individually. Both processes are used to decrease latency.

An example of a combination of these procedures is often used in streaming videos where the first user is dallying in the hope that more people would want to watch the same video and after a period of waiting or enough users are queued batching comes into play and instead of responding to them individually it awaits for the slowest request in the batch and responds to all of them at the same time.

Q2.3: Is caching an example of a fast path optimization? Explain why or why not. (1 paragraph)

Caching is indeed an example of fast path optimization. A simple example can be seen in most Web Browsers when they store the recent accessed Web Pages. The cache is indexed by the name of the page and returned immediately when the user requests that page. The page uses a slow path only the first time when it is accessed. The cache helps reduce the load on services and the load on the network. A considerable number of Web Browsers contain a cache to store the results of different user searches, so the next time the words or sentences are requested it will not require a DSN lookup. In the end the fast path optimization is needed only if there is a consisted difference between the slow path and the fast path latency.

**Programming task**

* Implementation: Regarding the implementation for this assignment, we have written the code for the functions *rateBooks*, *getTopRatedBooks* and *getBookInDemand* as required and for these we have looked at other functions that are already implemented:
* *rateBooks* function in the class CertainBookStore is checking if the ISBN is valid, if the rating has value between 0 and 5 and if the ISBN is in the bookstore(we did not check if there are actually copies available for sale because we thought it is not important). If all these checks are passed, then the rating is added to the book
* *getTopRatedBooks* this function is at first loading all books in store and we used Comparator for being able to sort the books considering the rating. After that the top rated books is returned.
* *getBookInDemand* is implemented almost the same way as the function *getBooks*, only that we check each book if it has any saleMiss and then it is added into the list that will be returned
* We made all these function available via RPC. For this we added new tags in the class BookStoreMessageTag: RATEBOOKS, TOPRATEDBOOKS, BOOKSINDEMAND and managed the RPC functionality in the classes for message handling and proxy servers. We have done this implementation almost the same way as other functionalities were included in those classes.
* Regarding the testing, we used JUnit for testing if we get error in case of rating a book with invalid ISBN, rating a book with value lower than 0 or ask for a negative number of top rated books and we also checked if the books in demand are obtained correctly. All the tests have passed; we did not get any error. Perhaps we should have written more tests and check other situations like trying to obtained a number of top rated books that is larger than the total number of books and do more tests on already existent methods but time did not allow us to play too much with those.

**Questions for Discussion on Architecture**

**Q1.a**: The architecture is strongly modular because the components are implemented separately. First of all the functions for managing the books, the store, the ratings, the stock and all others are well organized in separate classes and functions for each functionality of these elements. This helps with adding new functions very easy. For example, in the assignment we just simply called functions to check if ISBN is valid without taking care how a valid ISBN does is looking like. We also could change the rating (after easy validations) of a book by only calling a method and passing only one parameter(the new rating that is added) without having to take the previous ratings and calculated a new average(functionality that is already implemented in BookStoreBook). Basically each component in the virtual library has the functions to obtain information and change the state about it in its own class.

Secondly, the RPC communication with the server, the customer and store manager and the proxy server functionality is managed in different functions, in different packages by calling the methods of the bookstore. During the RPC communication we only take care if the communication is done correctly and simply call functions of the bookstore components without taking care if the rating is assigned correctly inside the communication classes.

Thirdly, the testing is also managed separately in other 2 classes by using the JUnit framework. In these classes we create test cases and check if functions in bookstore are implemented correctly without looking at their implementation. These helped with doing the tests very easy and focus on finding combinations that will lead to errors.

As a conclusion, the modularity is organized in layers. We have a layer for the functionality of bookstore components, one layer for managing the communication and all these help with adding new functions on each layer and connect them with each other very easily and check the errors and problems specific for each type of layer separately.

**Q1.b**: Regarding the virtualization the client and services are considered to be in different computers. There are used requests and responses to make sure that the client has access to the services that are offered.

**Q1.c**: Even if the clients and services run on the same machine, the isolation is preserved because there are different processes that manage each node in the communication and the bookstore will work in the same way. The client sends requests, the service is processing the request, makes validation and send back the answer.

**Q2.a**: Yes, there is a naming service in the architecture in the class *BookStoreHTTPServerUtiliy* the the *createServer* function which obtains the IP address of the server by name

**Q2.b**: The server is assigning a name for its IP address and then each time the client starts accessing the services of the server is, firstly obtaining the IP address that is associated with that name to be able to communicate with the server using the HTTP protocol

**Q3**: In this architecture it is implemented the semantic exactly-once because it is very possible that several failures and errors to occur. In case of failure the client receives the error message but the operation is not executed again. For each change in the bookstore, there are several parameters that are checked (ISBN, number of items that are requested or valid rating) and if one of these fail then the client will receive message error, but there are no attempts to do changes in the request or system to ensure that the request will succeed.

**Q4**: Yes, it is possible to implement proxy web servers because the Proxy server functionality is implemented for both types of clients: the StockManager and BookStoreClient. By adding proxy servers then scalability will be ensured for the system and more clients will be able to access the bookstore at once. However, the concurrency of clients for the bookstore resources will bring issues that must be solved. It is possible that to have requests for buying a book with limited stock at the same time, and that each client will get his request accepted but the total number of books in store is smaller than the request for all the clients. It is very important to ensure that failures like those will not happen. It is very important that inside the RPC communication to lock the access of other clients for same book until it is released by another client that wants to buy it.

**Q5**: In this architecture, the bottleneck seems to be on the BookStore Interface and on the StockManager Interface because all the clients will have to use these interfaces which will most likely be located on the same machine and it must be ensured that they can satisfy the requests from all the proxy servers. The limitation for the scalability of the architecture is determined by the capacity of these interfaces to provide communication with the clients and to avoid deadlocks which can happen if several clients want to access same resources at the same time.

**Q6**: In case of server failures the proxy servers will be able to continue to provide information for the clients but it depends on which functionalities would the proxy servers miss in case of server crash. Depending on the requests of clients at that moment, they may experience different things. If the client wants to buy books from the store, then the request will fail. But if clients want information about books like name, titles, rating and if these are stored on the proxy server then those clients will have their requests accepted and will receive the information they want.

It is possible to mask all these failures from the clients if there is copy for all information on server on the proxy. These should keep track of all transactions, available stock, all rating changes. This will mean that the proxy server will have to communicate to each other but this will be very hard to happen because in Figure 2 all the proxy servers are connected to the same interface which fails.