CACHE COHERENCY CONTROL BY USING MOESI PROTOCOL IN ONLINE DOCTOR APPOINTMENT

THURA ZAW

M.C.Sc. **SEPTEMBER 2022**

CACHE COHERENCY CONTROL BY USING MOESI PROTOCOL IN ONLINE DOCTOR APPOINTMENT

By

Thura Zaw

B.C.Sc.

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Computer Science

(M.C.Sc.)

University of Computer Studies, Yangon

September 2022

STATEMENT OF ORIGINALITY

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

---------- ----------

Date Thura Zaw

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my sincere thanks to those who helped me with various aspects of conducting research and writing this thesis. To complete this thesis, many things are needed like my hard work as well as the supporting of many people.

First and foremost, I would like to express my deepest gratitude and my thanks to **Dr. Mie Mie Khin**, Rector, University of Computer Studies, Yangon, for her kind permission to submit this thesis.

I would like to express my appreciation to **Dr. Tin Zar Thaw,** Professor, Faculty, of Computer Science, University of Computer Studies, Yangon, for her superior suggestion, administrative supports and encouragement during my academic study.

My thanks and regards go to my supervisor, **Dr. Si Si Mar Win,** Professor, Faculty, of Computer Science, University of Computer Studies, Yangon, for her support, guidance, supervision, patience and encouragement during the period of study towards completion of this thesis.

I also wish to express my deepest gratitude to **Daw Hnin Yee Aung**, Lecturer, Department of English, University of Computer Studies, Yangon, for her editing this thesis from the language point of view.

Moreover, I would like to extend my thanks to all my teachers who taught me throughout the master's degree course and my friends for their cooperation.

I especially thank to my parents, all of my colleagues, and friends for their encouragement and help during my thesis.

ABSTRACT

The rapidly growth of internet and the World Wide Web have made the possible for millions of users to gain access to geographically distributed web content. However, raising the population of online users and the unbalanced use of content accesses, popular objects may cause server and network overload and increase the latency for content access significantly. Caching is a common technique to reduce content access latencies. In caching system, data items are retrieved from the server machines, cached and processed at the client machines, and then write back to the server. This system controls the cache consistency by using MOESI protocol in online doctor appointment. This appointment system allows patients to get an appointment with doctor from any time at anywhere using mobile devices and internet service Wi-Fi, Dial-Up, Broadband, DSL, Cable, Satellite, ISDN. It can control the concurrent processing and save the waiting time not only for the users but also make the appointment process efficient.

Keywords: Cache Consistency, MOESI, Caching.

CONTENTS

LIST OF TABLES

TABLE **Page**

LIST OF FIGURES

FIGURE **Page**

CHAPTER 1

INTRODUCTION

In client server architecture of distributed systems, there are two classes of frameworks, question transportation and information delivering. In question transporting frameworks, like a social client/server DBMS, clients send an inquiry to the server, which processes the question and sends the outcomes back to the clients. Interestingly, the business information base administration frameworks have embraced the information transporting method. At the point when a client needs to get to information, it sends a solicitation for the particular information things (for example articles or pages) to the server. The information things are sent from the server to the clients, with the goal that the clients can run applications and perform procedure on stored information.

The information transporting design is enlivened by the sensational upgrades in PC cost execution and in the presentation and accessibility of organization correspondence. The innovation propels have made it attractive and useful to offload greater usefulness from the server to the client workstations. Regularly, information delivering frameworks can be organized either as page servers, in which clients and servers communicate utilizing actual units of information (for example pages or gatherings of pages), or item servers, which cooperate utilizing intelligent units of information (for example objects). The granularity of the information moved from the server to the clients is the crucial distinction among page and article servers.

Having the information accessible at the clients can diminish the quantity of client-server connections to accordingly free server assets (CPU and plates), hence diminishing client-exchange reaction time. Neighborhood storing permits duplicates of a data set page to live in numerous client reserves. Besides, the exchanges reserving is utilized, a page might remain stored locally when the exchange that has finished. Simultaneousness control for reserved pages should be upheld to guarantee that all clients have steady page duplicates in their stores and that they see a serializable perspective on the data set.

Client reserving brings irregularity into the framework. In the event that two clients all the while read a similar document and, both change it, a few issues happen. As far as one might be concerned, when a third cycle peruses the document from the server, it will get the first form, not one of the two new ones. This issue can be impacts of changing a document should not be noticeable worldwide. Another issue that is the point at which the two records are composed back to the server, the one composed last will overwrite the other one. The lesson of the story is that client reserving must be thought out reasonably cautiously. At the point when a reserve passage (document or block) is changed, the new worth is kept in the store, but on the other hand is sent quickly to the server. As an outcome, when another interaction peruses the record, it gets the latest worth.

1.1 Objectives of the Thesis

Concurrency control can provide the speed of transactions. However, it should resolve the conflicts that occur in multi user systems and to ensure the database transactions execute concurrently without violating the data integrity of individual databases. This system aims to provide the concurrency control for online appointment system based on the following objectives:

- To apply a control algorithm, MOESI in web based appointment system to resolve the conflicts that occur in multi user system
- To ensure the database transactions execute concurrently without violating the data integrity of individual databases
- To book appointments for doctors quickly and effortlessly, making the process less tedious and less time consuming.

1.2 Overview of the System

Nowadays, people are facing many different types of medical problems in the city. The epidemic has not come to COVID-19 as an end, and it is offered to the next major or minor diseases as well. Because of the lockdowns, the appointment for the doctor physically is impossible. By replacing the online appointment with the traditional one,

people can know the best doctor for their medical treatment and they do not need to directly communicate for booking process. Although the online doctor appointment system can provide the efficiency and patient satisfaction, when multiple users simultaneously reserve the same doctor, there may be faced with several concurrency problems as:

- Temporary Update Problem
- Incorrect Summary Problem
- Lost Update Problem
- Unrepeatable Read Problem
- Phantom Read Problem

Keeping in mind these issues, MOESI based online doctor appointment system has been developed.

1.2.1 Related Works

Data centric model using home-based lazy release consistency technique for developing distributed application was presented by [6]. In their system, consistency control and vector timestamp synchronization were implemented using train ticket sales system in Myanmar railway transportation as case study. For data modification, their system used vector timestamps to define the user who can modify the data.

Client centric concurrency control system was presented in [8]. Their work applied monotonic write consistency control and vector clock time synchronization algorithm. To ensure consistency control for the monotonic-write, write operations were propagated in to the all clients at several sites with the correct order.

The simulation of various concurrency control methods were performed by S. Kanungo and M. Rustom [14]. They compared the simulation results of 1. Lock-Based Protocols, 2. Timestamp-Based Protocols, 3. Validation – Based Protocols and 4. Multiversion Schemes based on number of transactions, committed transactions and rollback transactions. They suggested according to their simulations, Locking Protocols

are good for update intensive applications but they are not free from deadlocks and cause the locking overhead. Although Time stamp protocols provide better concurrency than locking protocols, they suffer the numerous rollbacks and may also cause the storage overheads for keeping timestamps. Optimistic protocols may abort more transactions if not prevented in frequent-update systems [5].

1.3 Organization of the Thesis

The organization of this thesis are as follows:

The **Chapter 1** introduces the concurrency control for data shipping, objectives, the research works related to this thesis and thesis organization are described.

The **Chapter 2** presents the theoretical overview of the concurrency control methods, transactions, the general issues between the transactions executed concurrently, the concurrency types and deadlock. The nature of distributed database and the structure of distributed transactions over the distributed databases are also presented in this chapter.

The cache consistency problem and resolution these problems, using active data aware cache consistency approach are discussed in the **Chapter 3**.

The underlying protocol for concurrency control for the proposed online doctor appointment system and the design and implementation of this system are illustrated in the **Chapter 4**.

Finally, the conclusions, the advantages over the proposed system, the limitation and future extension of the proposed system are expressed in the **Chapter 5**.

CHAPTER 2

BACKGROUND THEORY

In the Distributed Database Management Systems (DDBMS), concurrency control is the coordinating activity for concurrent accesses to a database by the multi-users. It allows users to access the distributed database in executing alone on a dedicated system. The main problem is in attempting to prevent database updates performed by one user from interfering with database retrievals and updates performed by another. The concurrency control problem is occurred in DDBMS because of the users may access data stored in many different computers in a distributed system, and a concurrency control mechanism at one computer cannot instantaneously know about interactions at other computers.

Because of the users try to modify the data stored in the different nodes, the concurrency control model at the one node cannot be known the interactions of the other nodes in the distributed system, the concurrency control problem is occurred.

2.1 Distributed Concurrency Control

Distributed Concurrency control is the activities of managing concurrent access to a database in a distributed system. It is the main task of transaction processing in the distributed database systems. Database transactions in the distributed systems should be coordinated so that the result remains the same as if they are executed sequentially. It is called concurrency control in database system [12].

There are various concurrency control mechanisms for managing concurrent access in a distributed system. In the distributed database system, concurrency control is used to resolve conflict with the concurrent access or update of data. The different types of concurrency control are described in the next section.

2.2 Types of Concurrency Control

There are various concurrency control algorithms to manage the detection and resolving of conflicts [8]. Concurrency Control mechanisms can be generally categorized as follows:

- Pessimistic Concurrency Control
- Optimistic Concurrency Control
- Deterministic Concurrency Control

2.2.1 Pessimistic Concurrency Control

A key element of negation methodology is to avoid potential contentions. Exchange access information unless conceivable combat situations arise. If that is absurd, the exchange should hold on until it becomes conceivable. Most cynical computing relies on locks. Old-fashioned cynical computing is the popular two-phase locking (2PL). [5]

Critical Concurrency Control, avoid concurrent executions of exchanges when conflicts that may cause future irregularities are identified. Pessimistic calculations synchronize the concurrent execution of exchanges immediately in the execution lifecycle. Approval (V) Read(R) Computation(C) write (W)

In critical framework, activities may be unnecessarily delayed. Also, the cynical framework can get stuck in a stop situation where, depending on each activity in the gathering, the gathering cannot continue. In a negative framework with a lock, It is important to obtain a fitting lock (by sending a lock request) before reaching the article. So when reading reserved items, there is a round-trip network delay anyway. This delay is important to ensure two things: The activity must carefully read (or change) the latest copy of the article and adhere to the locking rules. Cynical concurrency control can be delegated as follows:

- Two-Phase Locking(2PL)
- Timestamp Ordering

2.2.2 Optimistic Concurrency Control

Optimistic concurrency control executes transactions concurrently and determines whether the results of transaction execution are serializable at commit time. That is, before committing, the DBMS validates the transaction against all transactions that have been committed after the validation process or are currently in the validation process. Checking the serializability of each transaction can negatively impact the performance of systems with low collision rates. In this case, serializable tests are deferred until commit. In this approach, the lifecycle of a transaction is divided into three phases: execution, validation or confirmation, and commit. Transactions bring data items into memory and perform operations during execution.

In the validation phase, the transaction performs the checks to ensure serializability before completing the commit phase, which commits changes to the database. If the transaction can be committed, the DBMS replicates the local write to the database and sends the result back to the client. Otherwise, the transaction will be aborted and the local copy of the data will be lost. Each transaction gets a unique transaction ID before it starts and then appends to the server's local calendar and commit timestamp. When the DBMS executes a transaction, it copies each modified record to a private workspace. This allows the transaction to proceed without delay by checking for conflicts when the transaction is executed. However, having a large number of conflicts can have a significant impact on performance.

2.2.3 Deterministic Concurrency Control

Simple replication strategies are support by the deterministic control model. In this control technique, every clients send query requests to the distributed coordination layer comprised of sequencers. The sequencer ordered the transactions and assigned each transaction a unique transaction ID. Finally, the sequencer batches all the collected transactions and passes them to the server, which manages the partition containing the records the transaction wants to access. Record-level locking for each sequencer on the server is invoked by the scheduler component in a predefined order. That means an entire batch of transactions from the same sequencer is processed by the scheduler in the

transaction order of that sequencer and then the scheduler moves to the next batch of another sequencer.

If the transaction cannot acquire the lock, the DBMS queues it for the lock and the scheduler continues processing. The execution proceeds step by step. First, look at read/write transactions to identify all servers reading or updating data and all active servers performing updates. Then handle all local reads. If data from these local reads is required when executing a transaction on another server, the system passes the data to the server responsible for the transaction. At this point, the server without the update transaction (the inactive server) can finish executing and unlock the data. Write to the local partition are processed when the active server receives messages from other servers from which it expects to receive data. During this phase, the active server commits or aborts the transaction and releases the lock. Responses are sent to the sequencer, and when all responses arrive, it sends an acknowledgment to the client.

2.3 Problems in the Concurrent Executions of Transactions

Concurrency is when different clients access a dataset at the same time. The task of the concurrency control component is to ensure data set consistency while allowing multiple exchanges to run concurrently [1]. Concurrency issues arise in related areas.

2.3.1 Lost or Buried Updates

This problem occurs when two or more exchanges update them with similar information from the quote dataset. All exchanges know nothing about other exchanges. Subsequent exchanges assume that the main exchange understands and then reads what to update before the main exchange commits. Updates are lost whichever change are committed first.

2.3.2 Inconsistent Analysis (Non Repeatable Read)

If the exchange reads similar information at least once or twice, it should continue to read similar values. Non-repeatable reads occur each time a different message is perused, because subsequent exchanges make an information item similar multiple times, while

another exchange refreshes the item, and subsequent exchanges peruse it. Conflicting exams consist of re-reading a large number of similar terms over and over (at least twice)

Figure 2.1, which shows the exchange A plays out a conflicting investigation at time t_8.

ACC1 X := 40;	ACC2 Y := 50;	ACC3 Z := 30;
Transaction A	Time	Transaction B
- SUM := 0; read_item(x); SUM := SUM +x; -	t1	- - - - -
read_item(Y); SUM := SUM +Y; -	t2	- - -
- -	t3	read_item(z); z := z-10;
- -	t4	Write _item(z); -
- -	t5	read_item(x); x :=x+10;
- -	t6	Write _item(x); -
- -	t7	Commit -
read_item(z); SUM := SUM +z;	t8	- -

Figure 2.1: Transaction A Performs an Inconsistent Analysis

Two exchange A and B procedure on account (ACC) records: exchange A is adding account balance, transaction B is moving a sum 10 from account 3 to account 1.

Exchange A has perused thing X before expansion of (10) by exchange B at time t1, and read thing Z after deduction of (10) by exchange B at time t8. All the outcome created by exchange A is clearly wrong; if exchange A was proceed to compose that outcome back into the data set, it would really leave the data set in a conflicting state on

the grounds that the reads thing X by exchange A is not repeatable and exchange B commits its updates before the exchange A has understood thing Z.

2.3.3 Uncommitted Dependency (Dirty Read)

A transaction, on the off chance that it recovers or refreshes the information that has been refreshed by another exchange but has not been dedicated by another exchange. Messy read resembles to conflict checking, where what one exchange reads is committed by another exchange that rolled out improvements.

2.3.4 Phantom Reads

An exchange re-executes a question, tracking down a bunch of information not equivalent to a past one-albeit the hunt condition is unaltered. Ghost peruses may cause when inset or erase activity is performed against a column that has a place with the scope of lines being by an exchange. For instance:

The exchange reruns the question and tracks a set of information that is not equivalent to the past, but the hunt conditions are unchanged. Ghost browsing can occur when an insert or erase activity is performed on a column that has a range of rows placed by an exchange. For example:

Assume exchange A recovers the arrangement of all columns that fulfill some condition (e.g.; all providers the condition that the city in Paris).Suppose that exchange B then starts and embeds another line fulfilling that equivalent condition. If exchange A, presently rehashes its recovery demand, it will see a line that didn't beforehand exist.

2.4 Distributed Database

Distributed Databases stand out enough to be noticed in the data set research local area. Information conveyance and replication offer open doors for further developing execution through equal inquiry execution and burden adjusting as well as expanding the accessibility of information. As a matter of fact, these potential open doors play had a significant impact in propelling the plan of the ongoing age of data set machines.

A disseminated database system (DDBS) is an assortment of a few legitimately related data sets which are truly circulated in various PCs (generally called locales) over a PC network [16]. All destinations taking part in the dispersed data set appreciate neighborhood independence as in the data set at each site has full command over itself as far as dealing with the information. Additionally, the destinations can between work at whatever point required. The client of a conveyed data set has the feeling that the entire information base is nearby with the exception of the conceivable correspondence postpones between the destinations. This is on the grounds that a circulated information base is a logical union of the multitude of destinations and the dissemination is stowed away from the client.

A dispersed database Management system (DDBMS) includes an assortment of destinations interconnected by an organization. Each site runs at least one of the accompanying programming modules: a transaction manager (TM), an information the executives, and a simultaneousness control scheduler (or basically scheduler). In a client-server model, a site can work as a client, a server, or both. A client runs just the TM module, and a server runs just the information executives and scheduler modules. Every server stores a part of the data set. Every information thing might be put away at any server or repetitively at a few servers.

2.5 The Structure of Distributed Transactions

Figure 2.2 shows a general conveyed exchange as far as the cycles engaged with its execution. Every exchange has an expert cycle (M) that runs at its site of start. The expert cycle thus sets up an assortment of Cohort processes (Ci) to play out the genuine handling engaged with running the exchange. Since practically all question handling methodologies for dispersed data set frameworks include getting to information at the site(s)where it dwells, instead of getting to it from a distance.

There is something like one such companion for each site where information is gotten by the exchange. As a general rule, information might be imitated, in which case every partner that update any information things is expected to have at least one update (Uij) processes related with it at different locales. Specifically, a companion will have an

update interaction at every remote site that stores a duplicate of the information things that it refreshes. It speaks with its update processes for simultaneousness control purposes, and it additionally sends them duplicates of the important updates during the primary period of the commit convention portrayed beneath. The concentrated two-stage commit convention will be utilized related to every one of the simultaneousness control calculations inspected.

The convention functions as follows [5]. Distributed Transaction Structure When a companion wraps up executing its Portion of a question, its sends an "execution complete" message to the expert. At the point when the expert has gotten such a message from every companion, it will start the commit convention by sending "plan to commit" messages to all locales. Expecting that a companion wishes to commit, it sends a "ready" message back to the expert, and the expert will send "commit" messages to every partner subsequent to getting arranged messages from all companions.

The convention closes with the expert getting "committed" messages from every one of the partners. On the off chance that any companion cannot commit, it will return a "cannot commit" message rather than a "ready" message in the principal stage, making the expert send "cut off" rather than "carry out" messages in the second period of the convention. At the point when reproduction update processes are available, the commit convention turns into a settled two-stage commit convention. Messages stream between the expert and the partners, and the companions thusly interface with their updaters. That is, every companion sends "plan to commit" messages to its updaters subsequent to getting such a message from the expert, and it accumulates the reactions from its updaters prior to send a "ready" message back to the expert; stage two of the convention is comparatively changed.

2.6 Distributed Transactions

Distributed transactions deal with a physical division of transactions caused by the need to access distributed resources. Special distributed algorithms are needed to handle locking of data and committing of transactions.

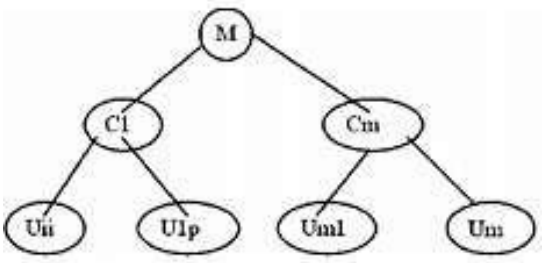


Figure 2.2: Distributed Transaction Structure

2.7 MODELINGA DISTRIBUTED DBMS

Figure 2.3 shows the general design of the model. Each site in the model has four parts: a source, which produces exchanges and furthermore keeps up with exchange level execution data for the site, an exchange director, which models the execution conduct of exchanges, a simultaneousness control chief, which carries out the subtleties of a specific simultaneousness control calculation, and an asset supervisor, which models the CPU and I/O assets of the site. Notwithstanding these per-site parts, the model likewise has an organization chief, which models the way of behaving of the interchanges organization.

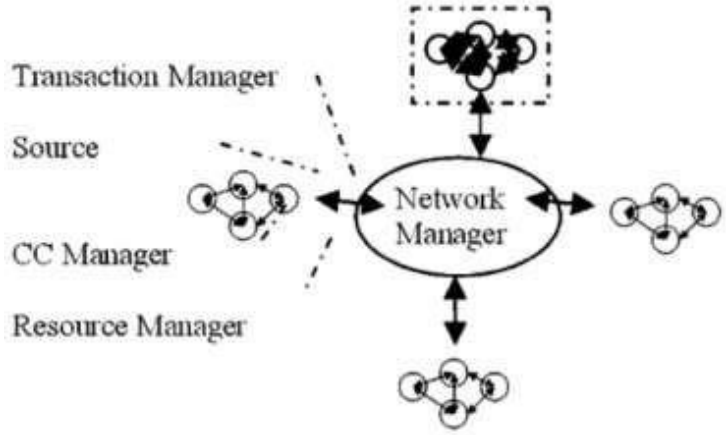


Figure 2.3: Distributed DBMS Model Structure

2.8 Distributed Concurrency Control Algorithm

The four algorithms are

- Distributed Two-Phase Locking (2PL)
- Wound-Wait (WW)
- Basic Timestamp ordering (BTO)
- Distributed Certification (OPT)

2.8.1 Distributed Two-Phase Locking (2PL)

The first calculation is the dispersed "read any, compose every one of the" two-stage locking calculation portrayed in [15]. Transactions set read locks on things that they read, and they convert their read locks to compose locks on things that should be refreshed. To peruse a thing, it gets the job done to set a read lock on any duplicate of the thing, so the nearby duplicate is locked; to refresh a thing, compose locks are expected on all duplicates. Compose locks are gotten as the exchange executes, with the exchange obstructing on a compose demand until the duplicates of the thing to be all refreshed have been effectively locked. All locks are held until the exchange has effectively dedicated or cut off. Halt is plausible, Local stops are checked for any time an exchange impedes, and are settled when fundamental by restarting the exchange with the latest beginning startup time among those associated with the gridlock cycle. (A companion is restarted by cutting

short it locally and sending an "cut off" message to its lord, which thus tells every one of the cycles engaged with the exchange).

Global halt location is taken care of by a "Sneak" process, which intermittently demands hangs tight for data from all destinations and afterward checks for and settle any worldwide stops (involving similar casualty determination models concerning nearby gridlocks). The "Sneak" obligation is not related with a specific site. All things being equal, each site takes a turn being the "Sneak" site and afterward hands this undertaking over to the following site. The "Sneak" obligation in this manner turns among the locales in a cooperative design, guaranteeing that nobody site will turn into a bottleneck because of worldwide halt identification costs.

2.8.2 Wound-Wait (WW)

The second calculation is the circulated wound-stand by locking calculation, again with the "read any, compose all" rule. It varies from 2PL in its treatment of the halt issue: Rather than keeping up with sits tight for data and afterward checking for nearby and worldwide stops, gridlocks are forestalled through the utilization of timestamps. Every exchange is numbered by its underlying startup time, and more youthful exchanges are kept from making more established ones stand by.

On the off chance that a more seasoned exchange demands a lock, and assuming that the solicitation would prompt the more seasoned exchange sitting tight for a more youthful exchange, the Approach for Concurrency Control in Distributed Database System more youthful exchange is "injured" - it is restarted except if it is as of now in the second period of its commit convention (in which case the "injury" isn't deadly, and is just overlooked). More youthful exchanges can sit tight for more established exchanges so the chance of gridlocks is disposed of.

2.8.3. Basic Timestamp Ordering (BTO)

The third algorithm is the fundamental timestamp requesting calculation of [9, 14]. Like injury stand by, it utilizes exchange startup timestamps, yet it utilizes them in an

unexpected way. As opposed to utilize a locking approach, BTO partners timestamps with all as of late gotten to information things and expects that clashing information is gotten by exchanges to be acted in timestamp request. Exchanges that endeavor to perform messed up gets to be restarted. At the point when a read demand is gotten for a thing, it is allowed if the timestamp of the requester surpasses the things compose timestamp.

When a compose demand is gotten, it is allowed assuming the requester's timestamp surpasses the perused timestamp of the thing; if the timestamp of the requester is not exactly the compose timestamp of the thing, the update is essentially overlooked (by the Thomas compose rule) [15]. For duplicated information, the "read any, compose all "approach is utilized, so a read solicitation might be shipped off any duplicate while a compose demand should be shipped off (and supported by) all duplicates. Coordination of the calculation with two stage commit is achieved as follows: Writers keep their updates in a hidden work area until commit time.

2.8.4 Distributed Certification (OPT)

The calculation is the disseminated, timestamp-based, hopeful simultaneousness control calculation from [13], which works by trading confirmation data during the commit convention. For every information thing, a read timestamp and a compose timestamp are kept up with. Exchanges might peruse and refresh information things openly, putting away any updates into a nearby work area until commit time. For each read, the exchange should recall the adaptation identifier (i.e., compose timestamp) related with the thing when it was perused.

Then, when the exchange's all's companions have finished their work, and have detailed back to the expert, the exchange is relegated a worldwide exceptional timestamp. All this timestamp is shipped off every companion in the "plan to commit" message, and it is utilized to locally guarantee its peruses and composes as follows: A read demand is confirmed if (I) the form that was perused is as yet the ongoing variant of the thing, and (ii) no compose with a more up to date timestamp has proactively been privately ensured. A compose demand is ensured if (I) no later peruses have been confirmed and in this manner committed, and (ii) no later peruses have been privately guaranteed as of now

updaters. As depicted before, the expert dwells at the site where the exchange was submitted.

Every companion makes a grouping of perused and composes solicitations to at least one records that are put away at its site; an exchange has one partner at each site where it needs to get to information. Companions speak with their updaters when remote compose access authorization is required for repeated information, and the updaters then, at that point, make the required compose demands for neighborhood duplicates of the information for their associates. An exchange can execute in either a successive or equal design, contingent upon the execution example of the exchange class.

CHAPTER 3

CACHE COHERENCE

This chapter explains the motivation for maintaining cache coherence in a multi-core system. Then, the formal definition of cache coherence will be followed by an overview of cache coherence protocols, including snooping coherence protocols and directory coherence protocols.

3.1 Definition of Cache Coherence

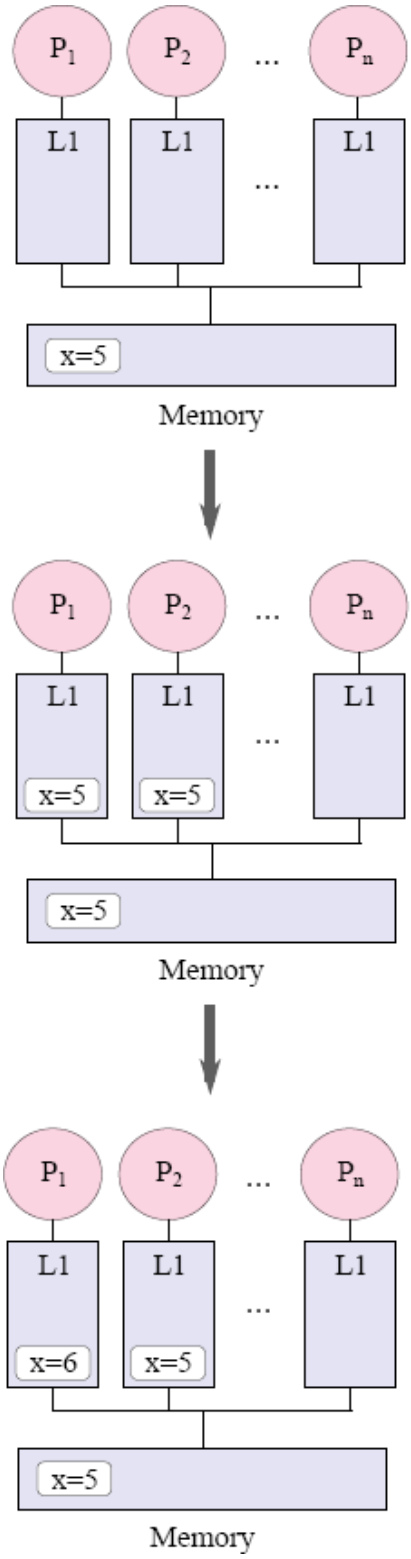
Generally talking, store rationality plans to make the reserves in a multi-center framework imperceptible to programming and act as though the framework is single-center. This implies that every one of the processors ought to peruse something very similar (most recent) esteem in the neighborhood L1 reserves. Circumstances, for example, a processor getting to a lifeless worth, supposed disjointedness, ought to never occur. The justification for why confusion can happen is that there might be more than one processor which gets to the information in the memory. Figure 3.1 shows an illustration of disjointedness which happens in a (improved) multicore framework. In the model, we can see that processor P2 holds the old worth of x 1 which might be returned a while later and subsequently make the program produce wrong outcomes.

Disjointedness can be forestalled by utilizing store cognizance conventions (or reserve conventions or intelligibility conventions for effortlessness). A reserve cognizance convention characterizes the guidelines keeping which processors ought to act. In the model in Figure 3.1, when P1 changes the worth of x, the convention ought to guarantee that, say, P2 is advised that the duplicate of x it holds will get flat and consequently makes a move (e.g., discrediting the duplicate of x in its L1 store). The correct definition of store cognizance should be presented before delving into the specifics of reserve intelligibility rules.

There are various meanings of store intelligence. For example, in [17, 18], there are two circumstances that should be fulfilled for the framework to be viewed as sound:

- each compose should be in the long run apparent to every one of the processors, and
- each processor notices the keeps in touch with a similar memory block in a similar request. One more adaptation is utilized the thought of tokens to characterize cache coherence.

The definition states that there are at most n tokens for each memory block, where n is equal to the number of the processors in the system. A processor is allowed to write the block when it holds all the tokens, and to read the block when it holds one or more tokens. Since there are only n tokens, it can never happen that when one processor writes a memory block, another processor also reads or writes the same block.



At the beginning, the value of x is 5 in the memory.

Then processors P_1 and P_2 load x from the memory in the local L1 caches respectively.

Later on processor P_1 changes the value of x, and now x is equal to 6 in its L1 cache.

However, P_2 is not aware of the change and therefore keeps the stale value of x in the L1 cache, which leads to incoherence.

Figure 3.1: An Example of Cache Incoherence

Coherent time, or legitimate clock, is characterized by Leslie Lamport in [18]. Naturally, for two occasions A and B, occasion An occurs before occasion B in legitimate time assuming that any of the accompanying three circumstances is satisfied: 1) occasion A happens before B in a similar processor, 2) occasion A is the sending of a message and occasion B is getting of the message, or 3) occasion A occurs before occasion C and occasion C occurs before occasion B. On the off chance that occasions A and B are not related with the happens-before-connection, then the request between them can be one way or the other.

The lifetime of every memory block is considered to comprise of a succession of ages, where an age is a timespan during which either 1) a processor can compose (and read) the block, or 2) one or more processors can peruse the block. At the end of the day, when the consent that is a processor can work on the memory block changes, the ongoing age closes and another age begins. Figure 3.2 shows an illustration of ages of a memory block. In the model, processor P1 can peruse and compose the memory block during age a. At the point when P2 is permitted to peruse and compose the block, age a completions and age b begins. Then, at that point, P2, P5 and P6 are permitted to peruse the block during age c, etc.

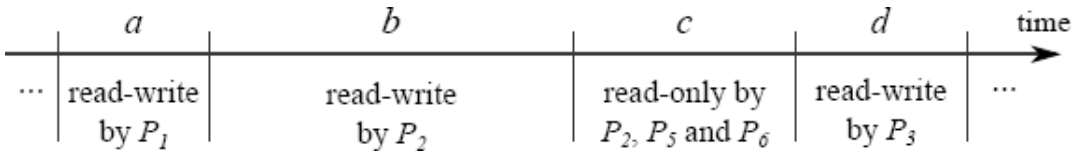


Figure 3.2: An Example of Epochs of a Memory Block

The meaning of reserve intelligence is the accompanying. Officially, there are two invariants of reserve intelligence:

- In the first place, the single-essayist various per user (SWMR) invariant. All the more exactly, it characterizes that at any (sensible) time for any memory block, there is by the same token
  - a single processor which might compose (and read) the block, or
  - one or more processors which might understand it.
- Second, the information esteem invariant. This implies that the worth of a memory block towards the start of an age is equivalent to the worth toward the finish of its last perused compose age.

It is necessary to have a reserve rationality convention to keep up with these two invariants. The following section will explain how these conventions function generally.

3.2 An Overview of Cache Coherence Protocols

As referenced over, a store intelligence convention indicates how the processors and the memory ought to connect to ensure lucidness. All the more exactly, in a store soundness convention see Figure 3.3, the processors and the memory speak with one another through an interconnection organization to keep up with the SWMR and the information esteem invariants, for each memory block and constantly.

Regularly, for example, in the event that a processor gives a heap (read) or a store (compose) guidance however neglects to find the information that it is going to work on, which is a supposed reserve miss, the processor will send a solicitation for the information through the interconnection organization. Either the memory or another processor, contingent upon the store lucidness convention, will answer the solicitation with the information. In some cases, after a processor changes the information in its L1 reserve, it will tell different processors which hold similar information about the adjustment of some way. The memory sends reactions and advances demands when vital. Each store convention has its own principles about how the messages ought to be traded and handled.

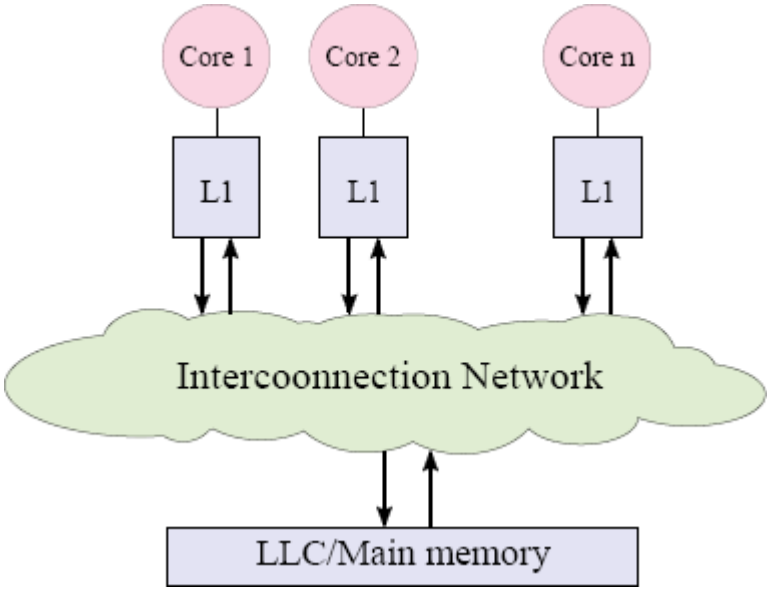


Figure 3.3: An Overview of the Components in a Cache Coherence Protocol

Expect that under a convention for store rationality, a number of indistinguishable and free limited state machines (FSMs) will capture the behaviors of each processor and the memory. Each FSM addresses a memory block or a store line, and the conditions of the FSM demonstrate the states of the block (line) in the memory (processor). At the point when a processor gets a message, e.g., a solicitation for a store line A, it processes the message (say answering the solicitation with the information) and may correspondingly have an impact on the condition of the FSM addressing the line A (or the condition of the line A for straightforwardness). The conditions of each reserve line and the changes between the states can fluctuate in various store soundness conventions. We expect that all the L1 stores contain a similar arrangement of reserve lines, to be specific a similar arrangement of FSMs. A simple model for an innate understanding of store cognizance conventions will be presented below, followed by the presentation of two major categories of reserve intelligence conventions [9].

3.2.1 States and Events

A straightforward reserve soundness convention is displayed in Table 2.1, where the columns demonstrate the conditions of a store line and the sections are the significant

occasions that can occur. Each state addresses an alternate state of a store line in the L1 reserve having a place with some processor. The circumstances incorporate that 1) the line is not open by the processor (state N); 2) the line must be perused by the processor (state R); and 3) the line can be both perused and composed by the processor (state W). An occasion can either be an activity given by the nearby processor (load/store), or a solicitation from another processor (e.g., demand for read-just consent). The passages of the table are the changes between the states. They are as Move/State, where Activity implies the activity made by the processor when an occasion happens to the reserve line, and State is the following state that the store line will move to. For instance, the section "Solicitation for read-just consent/R" shows that when a processor attempts to stack a store line which isn't accessible in its nearby L1 reserve, the processor will give a solicitation which requests the read-just authorization of the line and the condition of the store line in the L1 store of the processor will change to peruse just (R). In the event that the activity is not referenced in a section, then it implies that no move is made upon the occasion; the shortfall of the state implies that the reserve line moves to no other state, and clear passages show that the solicitation is disregarded by the processor.

The states in Table 3.1 are called stable states. Normally, in the execution of a reserve cognizance convention, there are likewise transient states which happen in the change between two stable states. For example, a processor cannot get to a store line when it is in state N. Then, at that point, the processor sends a read-just solicitation for the line and gets the reaction. Presently the store line state is R. A transient condition of this line emerges when the processor is sitting tight for the reaction, i.e., between states N and R. Expect that the occasions of sending a solicitation, receiving a solicitation, responding with information, and modifying the relevant store line state occur on a molecular level in this proposition, unless otherwise specified. This means that researchers just need to focus on stable states. Memory blocks and reserve lines are also distinguished by states. The conditions of a memory block show the conditions of the comparing lines in the L1 stores. That is what a regular model is on the off chance that a store line is unavailable in all the L1 reserves, to be specific it is in state N, then the condition of this block in the memory is N too. Memory block states are additionally out of the extent of this postulation.

Table 3.1: General Behaviors of a Processor Write a Cache Line

States	**Events**			
	Load	Store	Read only request from another processor	Read write request from another processor
Inaccessible (N)	Request for read only permission/R	Request for read write permission/W		
Read only (R)	Read data	Request for read write permission/W		/N
Write only (W)	Read data		Send data to requestor and memory/R	Send data to requestor/N

3.2.2 Snooping and Directory

There are two main categories of cache coherence protocols: snooping and directory:

- Sneaking around convention: In a sneaking around convention, every one of the processors are made mindful of the notices sent on the interconnection organization. They "sneak" on the organization and respond adhering to the convention guidelines. A notice can be a solicitation for a store line or the message that the worth of a reserve line is changed by some processor and so on. At the point when a processor is going to compose a store line, for example,

it initially sends a warning to the organization, constraining different processors hanging tight to nullify it and afterward plays out the compose activity.

- Catalog convention: In a registry convention, the principal memory keeps an index that records the data of each and every store line, for example, which processors have the perused just or perused compose consent of this line and so on. Catalog conventions make every one of the solicitations go through the memory. For instance, when a processor needs to get to a store line, it sends the solicitation to the memory. The memory really looks at the registry and concludes which processor ought to answer the solicitation. Then the solicitation is sent by the memory to that processor, which answers the requestor with the information. In the event that there is no processor holding the information, the actual memory answers the solicitation by sending the information. Another regular model is that a processor needs to compose a line. In such a case, it first tells the memory, which sends messages to every one of the processors holding similar duplicate of the line and powers them to discredit the duplicate. The re-questing processor won't be permitted to compose the line until it gets the affirmations from every one of the processors that hold the duplicate.

The two sorts of conventions have compromises, and it is difficult to close which one is better than the other disregarding the unique situation. Sneaking around conventions don't scale to a major number of processors because of broadcasting, while index conventions might get some margin to deal with demands (demands sent between processor, memory and processor rather than between processors. The MESI convention that we will present in next area can be executed in the two ways.

3.3 The MESI Cache Coherence Protocol

The name MESI stands for the four states that can mark a cache line, namely Modified, Exclusive, Shared, and Invalid. The detailed meaning of each state is as follows.

- Modified: the cache line is only available in the current L1 cache and it is dirty, which means that it has been modified by the processor holding it, but not

updated to the memory yet. In this way the value of the corresponding block in the main memory is stale.

- Exclusive: the cache line is only available in the current L1 cache and it is clean, which means that the value of it is the same as the value of the corresponding block in the main memory.
- Shared: the cache line is available in the current L1 cache, and may also be available in other L1 cache(s). The line is clean in these caches.
- Invalid: the cache line is invalid, which means that it is unavailable in the current L1 cache.

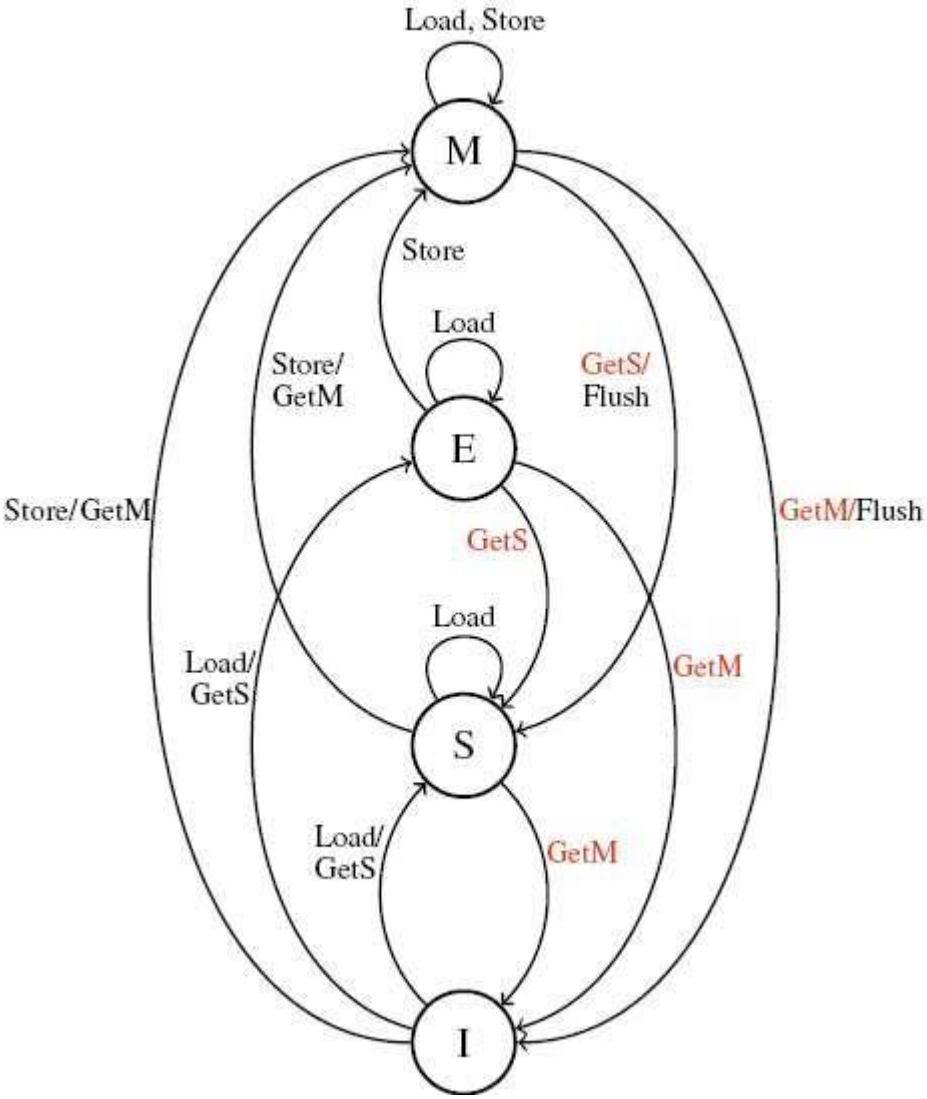


Figure 3.4: Transitions between the States in the MESI Cache Coherence Protocol

Figure 3.4 outlines the changes between the conditions of a store line held by some processor. In the figure, each progress is named with tasks or potentially messages, where the dark messages are the activities and the messages gave by the processor, and the red messages are the messages shipped off the processor. A few changes are of the structure A=B, where A is the activity given by the processor or the message shipped off it, and B is the message or the activity gave by the processor upon A. For instance, the progress from M to I in the figure is named with GetM/Flush, and that implies that the processor gets a GetM message, flushes the information (i.e., refreshes the information into the memory) and moves the store line to the state I. The two messages are presented before delving into the nuances of the MESI convention's advancements:

- GetS: represents GetShared, which compares to mentioning the read just authorization of the reserve line.
- GetM: represents GetModified, which compares to mentioning the read compose authorization of the reserve line.

A processor can peruse or compose information. At the point when a processor is going to work on certain information, there are two cases: 1) reserve hit, and that implies that the information can be found by the processor in the L1 store, or 2) store miss, in particular the processor neglects to find the information in the L1 store. Subsequently, joined with read and compose tasks, there are four potential situations when a processor works on a store line, i.e., read hit, read miss, compose hit and compose miss. Describe the improvements in these four situations in Figure 3.4 as follows. Use the A to B progression structure, where A and B stand for the source and goal states, respectively, and t for the transition mark. Read hit: a read hit happens when the processor figures out how to get to the reserve line and peruses the information. The store state doesn't change upon a read hit. The changes comparing to a read hit are: M Burden M, E Burden E and S Burden S.

- Read miss: a read miss can happen when the store line is in the territory of Invalid. The processor needs to give a GetS demand for the information. There are various changes relating to a read miss from the Invalid state. The changes rely upon the states of different processors. All the more definitively:

- None of different processors holds a duplicate of the mentioned store line. The processor gets the information, peruses the information and moves the reserve line state from Invalid to Selective, since the processor is the one in particular that has the line in its L1 store. The relating change is I Load=GetS ➔E.
- There is one processor holding the store line which is in the state Select. For this situation, this processor will move the store line state from Selective to Shared (E GetS ➔S), since now there will be two processors sharing the reserve line. In the mean time, the mentioning processor peruses the information and moves the store line state from Invalid to Shared (I Load=GetS ➔S).
- There are at least one processors holding the store line in the Common state. The mentioning processor gets the information, peruses the information and moves the store line state from Invalid to Shared as above (I Load=GetS ➔S). The processors which as of now hold the information don't change the reserve line state, since it is now Shared.
- There is a processor holding the store line in the Changed state. After getting a GetS demand, the processor flushes the changed worth to the memory first. Then, at that point, it moves the reserve line state from Altered to Shared. This implies that the processor just has the perused consent of the store line now, since it is imparting the line to another processor. The comparing change in the figure is M GetS = Flush➔S. The mentioning processor, same as above, peruses the line and moves the reserve line state from Invalid to Shared (I Burden = GetS ➔S).

Write hit: a write hit happens when the cache line state is not Invalid in the L1 cache. Different statuses of the line should be considered:

- The store line is in the territory of Adjusted. Then the processor basically composes the information without changing the reserve line state, since it as of now has the compose authorization of the line (M store ➔M).
- The store line is in the territory of Selective. The processor composes the worth and alters the condition of the line from Select to Adjusted (E store ➔ M), implying that it has the perused compose consent of the reserve line now.

- The store line is in the territory of Shared. The processor needs to send a GetM demand first. Different processors which additionally have the reserve line in the Common state meaningfully impact the state from Shared to Invalid upon the GetM message (S GetM ➔I), since the information in their L1 reserves will be flat. The processor which sends the message changes the information and moves the reserve line state from Shared to Adjusted (S Store=GetM ➔ M).

Write miss: a write miss happens when the cache line state is Invalid in the L1 cache. The processor needs to send a GetM request for the read-write permission of the data. It then changes the data and moves the cache line state from Invalid to Modified (I Store=GetM ➔ M) after it receives the response. There are four possible cases that we need to consider:

- None of different processors has the reserve line in the L1 store. Then the requestor gets the information from the memory and doesn't have to nullify any duplicate of the line.
- There is a processor holding the reserve line in the Elite state. Then it requirements to move the condition of the reserve line to Invalid (E GetM ➔ I), since the information will be lifeless.
- There is at least one processors holding the store line in the Common state. Like the case over, the processor(s) move the state from Shared to Invalid (S GetM ➔I).
- There is a processor having the store line in the Changed state. The processor first necessities to refresh the worth of the information to the memory and afterward changes the reserve line state from Adjusted to Invalid (M GetM=Flush ➔I).

In a snooping protocol, messages are shipped off every one of the processors and the memory, and the processors respond as per the messages; while in a registry convention messages are constantly shipped off the memory, which concludes whether itself or some processor ought to answer the solicitation. The MESI convention can be carried out in the two ways. The advances in Figure 3.4 are autonomous of how really the MESI convention is executed. For instance, when a processor gets a Receives message for a store line in state

Selective in its L1 reserve, it will fundamentally have an impact on the state to Shared, regardless of whether the solicitation is sent from the memory or from another processor. The discussion of store rationality rules in this thesis focuses solely on state transitions rather than delving into the nuances of how messages are handled, such as whether they are sent directly to individual processors or first pass through memory. Review that reserve intelligibility comprises of two invariants:

- SWMR, which means that at any time only one processor can write (and read) a memory block, or one or more processors can read the block;
- The data value invariant, i.e., the value of a memory block at the beginning of an epoch is the same as the value at the end of its last read-write epoch.

In the MESI protocol, the meaning of the states guarantees that a reserve line must be composed by each processor in turn, since in the event that a line is changed in one L1 store, it should be Invalid in the wide range of various L1 reserves. Likewise, a store line can be in the Restrictive state in one L1 reserve or in the Common state in at least one L1 stores with the wide range of various L1 stores having the line in the Invalid state. This implies that the convention permits at least one processors to peruse a store line simultaneously.

CHAPTER 4

SYSTEM DESIGN AND IMPLEMENTATION

In the client-server framework, the data copy from the server (worldwide informational index) is taken care of in the client's informational index (close by/store informational index) - saving.

At the point when the client's change trade is successfully devoted in neighborhood informational index (Close by Store) and overall informational index, data in the informational index will be clashing with the data on the contrary side of clients, in light of the fact that other client doesn't have even the remotest clue about the change.

To avoid the situation, after the change is successfully devoted, the changed information ought to be analysis to another client to keep the data consistency (Store Consistency).

4.1 MOESI Protocol

In the storing framework, client brings objects from the cut off machine, work on them locally and send back any acclimations to the server. Such designs further foster system execution by utilizing the taking care of power of client machine. The server's load is reduced by continuing whatever amount of computation as could sensibly be anticipated on client machine.

The consistency control is an essential occupation for the coordinated execution of trade over an informational index. In this manner, client putting away carries abnormality into the structure. In case two clients meanwhile read a comparative record and, both change it, a couple of issues occur. Taking everything into account, when a third cycle examines the report from the server, it will get the main structure, not one of the two new ones. This issue can be effects of changing a record ought not be perceptible globally. Another issue: when the two records are created back to the server, the one formed last will

overwrite the other one. Exactly when a store entry (record or block) is changed, the new worth is kept in the hold, and yet is delivered off the server. Accordingly, when another association scrutinizes the record, it gets the most recent worth.

States of MOESI: In computing, MOESI is a full cache coherency protocol that encompasses all of the possible states commonly used in other protocols [17].

- Modified(M)
- Shared(S)
- Invalid(I)
- Owned (O)
- Exclusive(E)
    - **Modified (M):** The store line is available just in the ongoing reserve, and is grimy - it has been modified (M state) from the value in head memory. The store is supposed to form the data back to key memory eventually, preceding permitting another read of the (right now not authentic) essential memory state. The compose back changes the line to the Share state(S).
    - **Owned (O):** This store is a rare example with a genuine copy of the hold line, yet has the select right to make changes to it — various stores could scrutinize anyway not form the save line. Right when this store changes data on the save line, it ought to impart those changes to any excess stores sharing the line. The introduction of the Guaranteed state licenses muddled sharing of data, i.e., a changed store block can be moved around various stores without reviving head memory. The hold line may be changed to the Adjusted state resulting to refuting each normal copy, or changed to the Common state by composing the alterations back to principal memory. Claimed store lines should answer a sneak solicitation with information.

- **Exclusive (E):** The reserve line is accessible simply in the continuous store, yet is immaculate - it matches essential memory. It very well may be changed to the Familiar state at whatever point, considering a read interest. Then again, it may be changed to the Modified state while staying in contact with it.

- **Shared(S):** Indicates that this cache line may be stored in other caches of the machine and is *clean* - it matches the main memory. The line may be discarded (changed to the Invalid state) at any time.

- **Invalid (I):** Indicates that this cache line is invalid (unused).

4.2. The System Overview

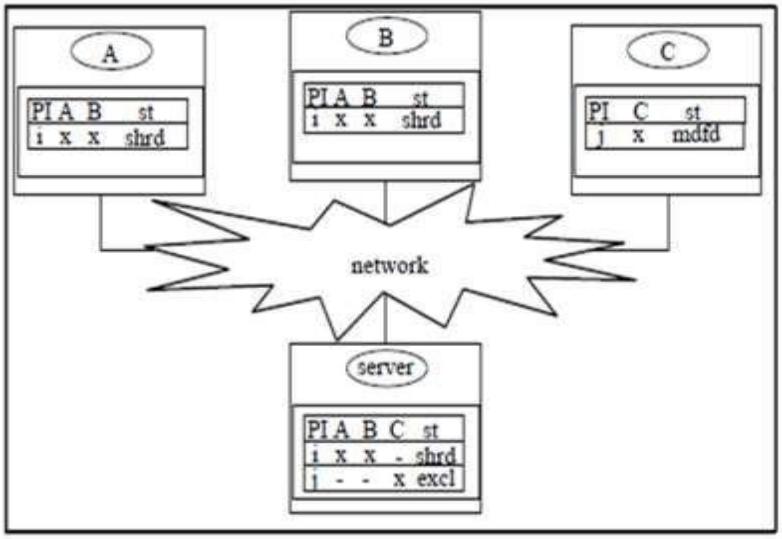


Figure 4.1: The System Overview

Close to the start of the structure, the client requests a record from server and a short time later that is taken care of in the client's store. Right when client examines the chronicle, the system checks the read report is moved by various clients. If the record has more than one per user, the structure set the well-known mode in its client's library. Expecting that the report has recently a solitary per user, the structure set the excc mode in

its client's list. Hence, the client understands that the document is concurrent examining or not.

Exactly when client requests the make grant to the file, the structure checks the referenced report is asserted by various clients. Accepting the file has more than one owner, the system set the "S" mode in its client's vault. Then, at that point, the system checks the late owners and imparted them not to get create grant. The early form request client gets the create grant and change the "M" mode in its client's vault. After commit the create trade, the client's serious data update is similarly update in the server informational index. Then, at that point, the server checks the clients list in its vault for the consistency data replication to other client's informational collection.

Expecting the file has recently a solitary owner, the structure set the "M" mode in its client's library. Then, at that point, the serious data update is taken care of in server anyway try not to duplicate various clients because the document isn't in share mode. Subsequently, the system have some command over the client store consistency and server consistency by the usage of the two level inventory (Client library and Server record). Every one of the clients can know which client own which chronicle and which client surrendered which process mode by the client of client library and server inventory. This is the property of MOESI convention [13].

4.3 Consistency Controlling

Not many clients have a clashing library near with the server for a comparative page. Thusly, those clients could have old presence standards in their libraries. Such library abnormality makes an issue right when those clients need update the page. Right when the server gets a speculative update interest for a typical page, the server differentiates its index and that of the client. Expecting the server recognizes that the client list is outdated, it permits the speculation, but it enlightens the client in regards to the irregularity. Expecting there are a couple of new sharing clients, the speculative client is not allowed to commit before the new sharing clients nullify their copies. At commit time, the client sends the logs to the server, modifies the state of revived pages from involved to changed, and a

while later starts the other client requests that are believing that this client will commit. Right when the server gets a do message, it takes out those segments in the Client Question Table associated with the committing trade, changes the involved select state of those revived pages (with involved client as the committing client) to prohibitive, moves the logs on to a consistent storing locale, and orders the other client trades that are believing that this client will commit.

4.4. Implementation of the System

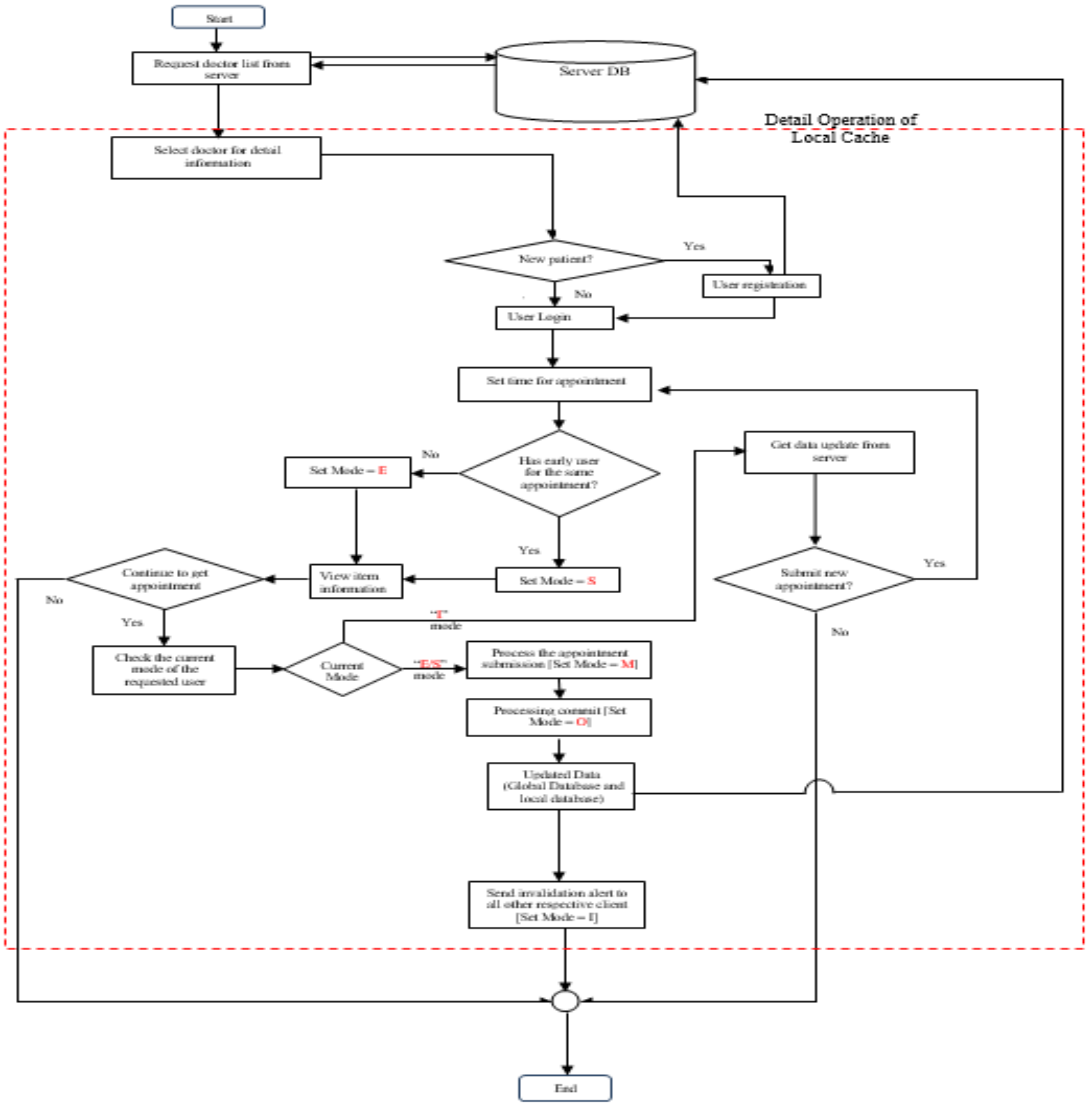


Figure 4.2: The System Flow Diagram of the Proposed System

Arrangement the board frameworks are utilized by all the essential medical care facilities and furthermore emergency clinics for overseeing admittance to specialist organizations. Arrangement chiefs can tremendously affect the outcome of the association, accordingly they merit appropriate consideration. Guaranteeing patients are planned successfully and proficiently is basic to keeping up with and amplifying centers income. While magnificent clinical consideration stays the vitally quiet assumption, medical services purchasers are presently additionally looking for supporting frameworks like arrangement directors that satisfy their necessities. The consistency controlling process stages of proposed doctor appointment system is briefly explained above mentioned Figure 4.2 and the following sequence diagram Figure 4.3.

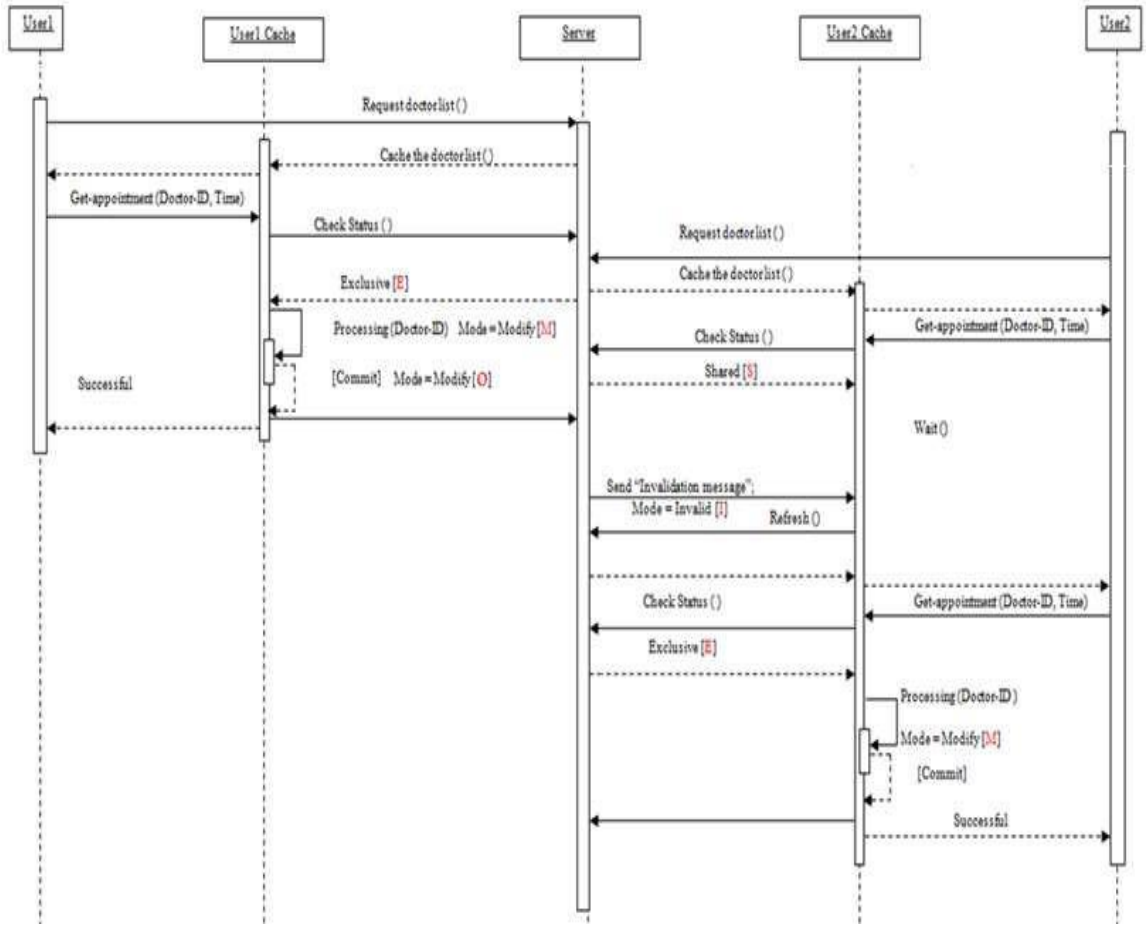


Figure 4.3: Sequence Diagram of the Proposed System

4.4 Home Page

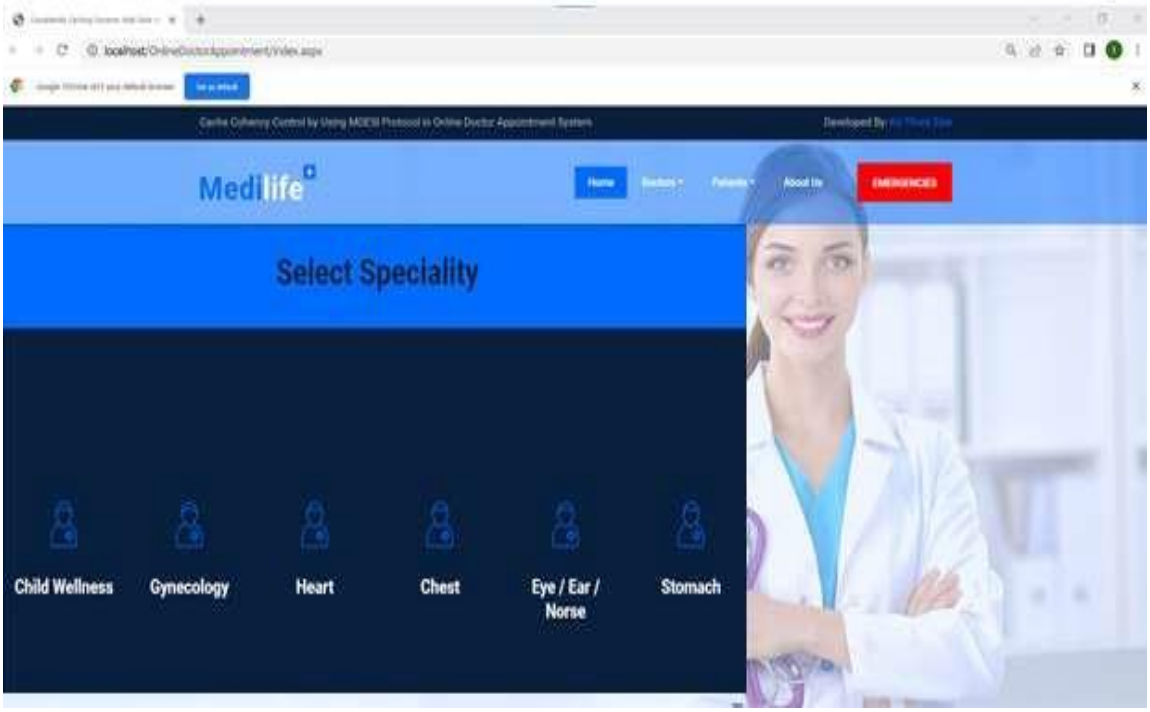


Figure 4.4: Home Page

The proposed system main page is as shown in Figure 4.4. Main page is implemented to search and get appoint from online doctor appointment system by categorized specility.

4.5 Doctor List Page

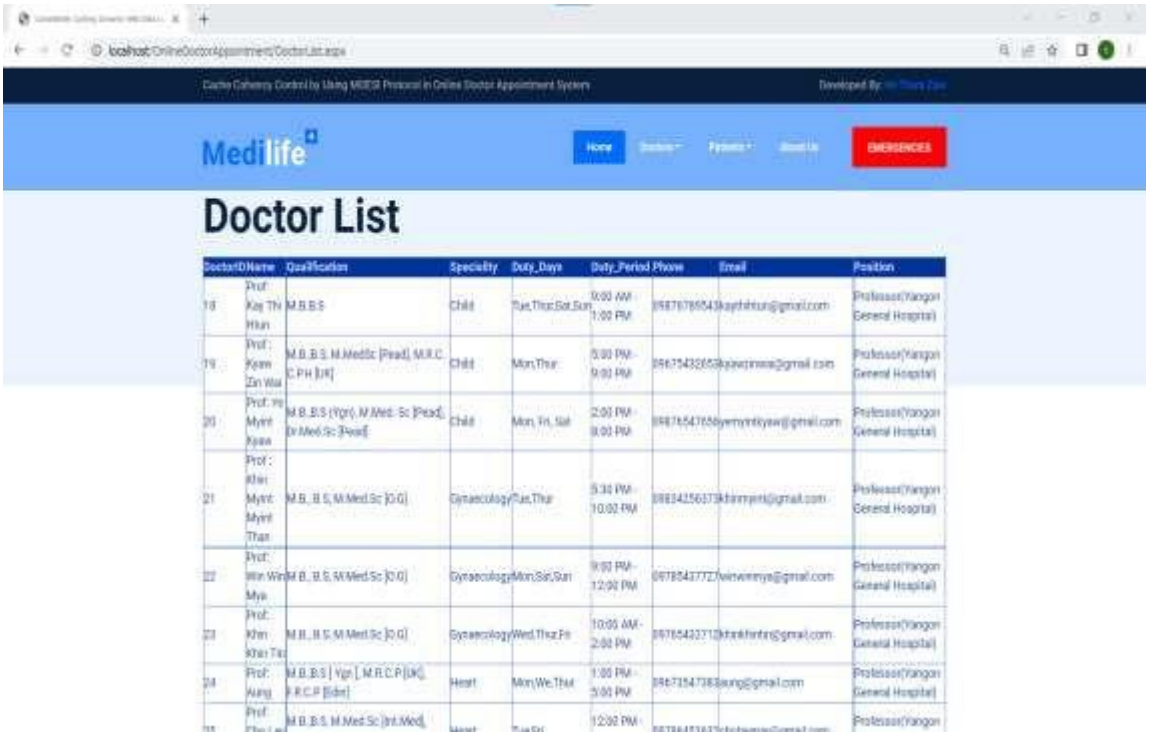


Figure 4.5: Doctor List Page

In this proposed system, the clients/patients can view and search the detail information of the doctors by their specialization, duty_days, duty_period, phone, email and position. So that they can book the doctors according to their sickness or treatment within the doctors' duty days. The sample list of the doctors with their information are shown in the Figure 4.5.

4.6 Patient List and Registration

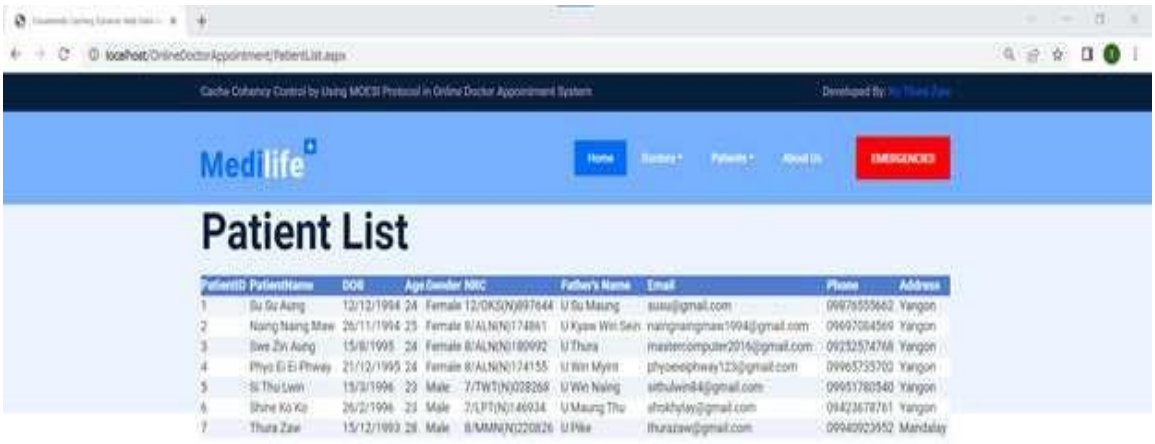


Figure 4.6: Patient List Page

This system only permits the registered user (patients) to enter the system. So, the registered patients information are stored in database and the sample list is shown in Figure 4.6. The patient registration form design is implemented as shown in Figure 4.7.

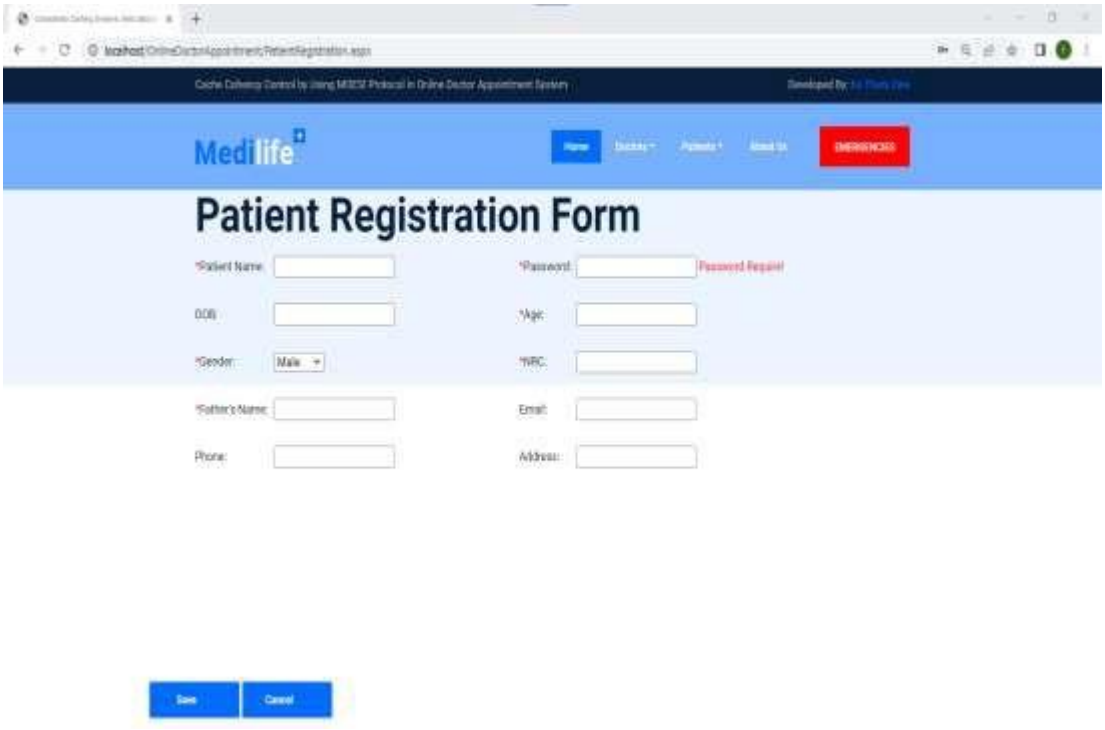


Figure 4.7: Patient Registration Form

4.7 Booking Doctor

Figure 4.8: Doctor List to Make Appointment

Figure 4.8 shows the list of the doctors available in the system. The client/patient can the appointment with the desired doctor via this “Select doctor” page. The schedule of selected doctor and his duty times can be seen as shown in the following Figure 4.9.

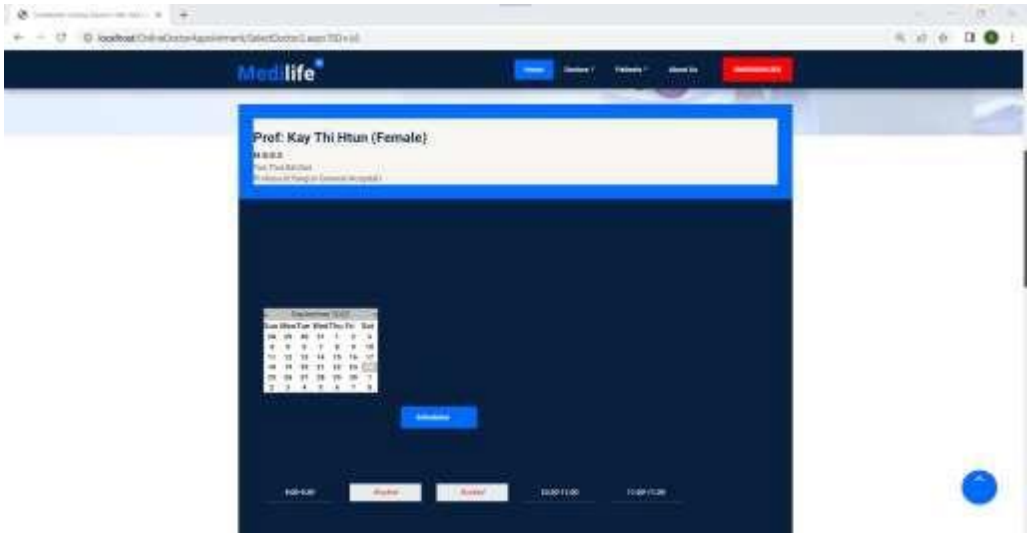


Figure 4.9: Schedule to Book

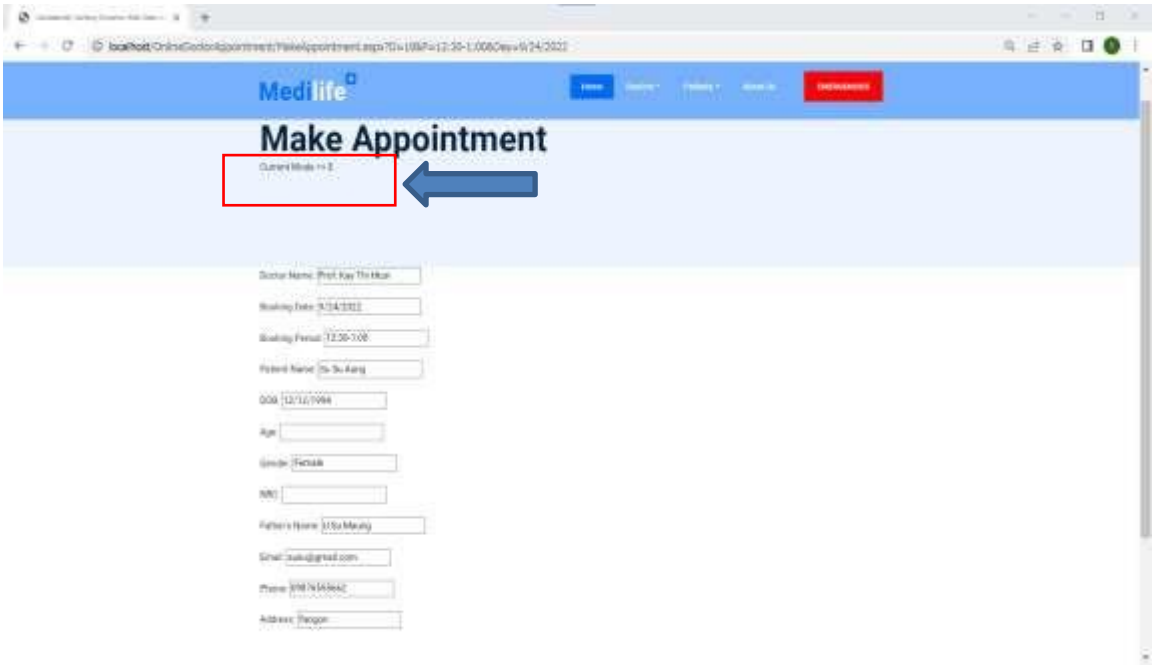


Figure 4.10: Appointment Making (E-Mode)

The main idea of proposed system is controlling consistency and concurrency of get appointment. In case of consistency (consistence view of object) control, the system applied MOESI protocol. In the initial state of making appointment and the appointment getting patient is only one for the selected duty time of period, the consistency alert will be shown “I” as shown in Figure 4.10.

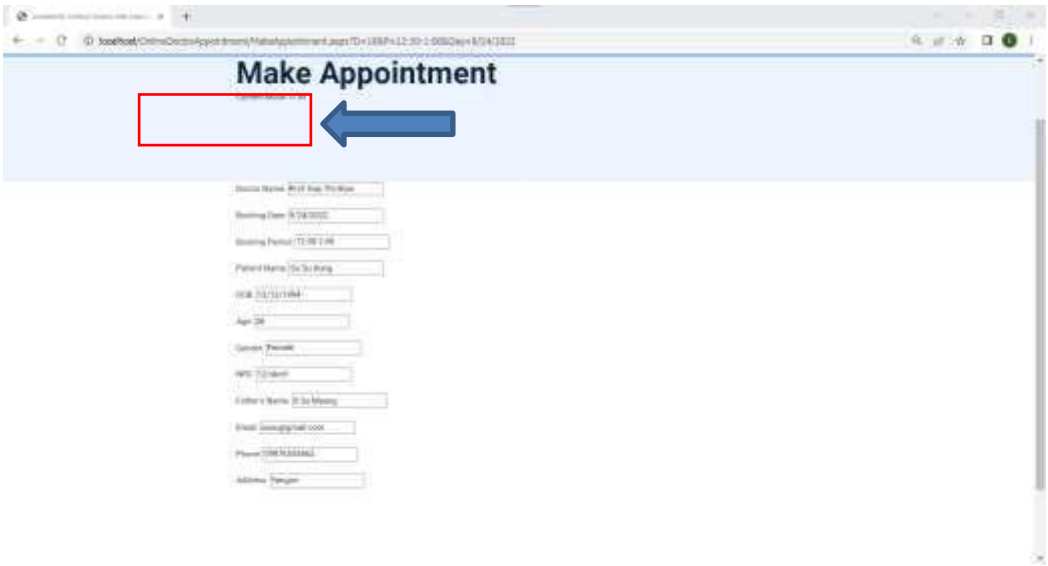


Figure 4.11: Appointment Making (M-Mode)

Then, the user continues to fill requirement information, the system change consistency mode to "M ➔ modify mode". During this phase, another user want to get the same appointment period, the system will show the consistency information as "S ➔ share mode" before the early user is committed as shown in figure 4.13. If the early user finished the transaction processing, the user (the patient) is defined as "O➔ occupy mode / Booked" as shown in Figure 4.12.

Figure 4.12: Appoint Making (O-Mode / Booking Success)

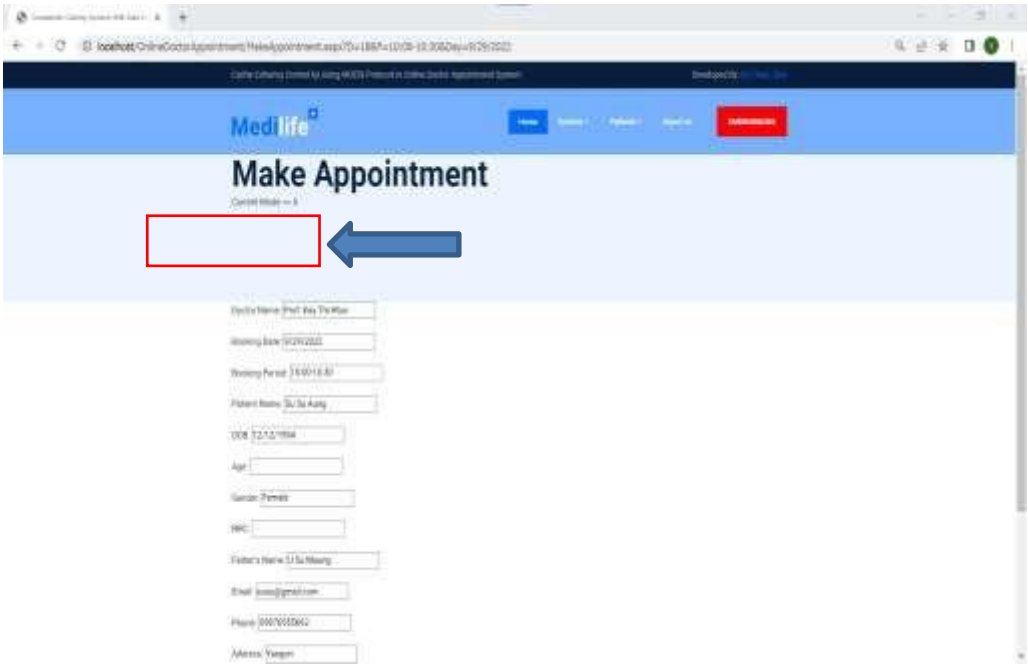


Figure 4.13: Appoint Making (S-Mode)

For the later user who is getting the same appointment period as the early user appointment, the system will be shown the consistence information for the late user is set as “I ➔ invalid mode” as shown in Figure 4.14.

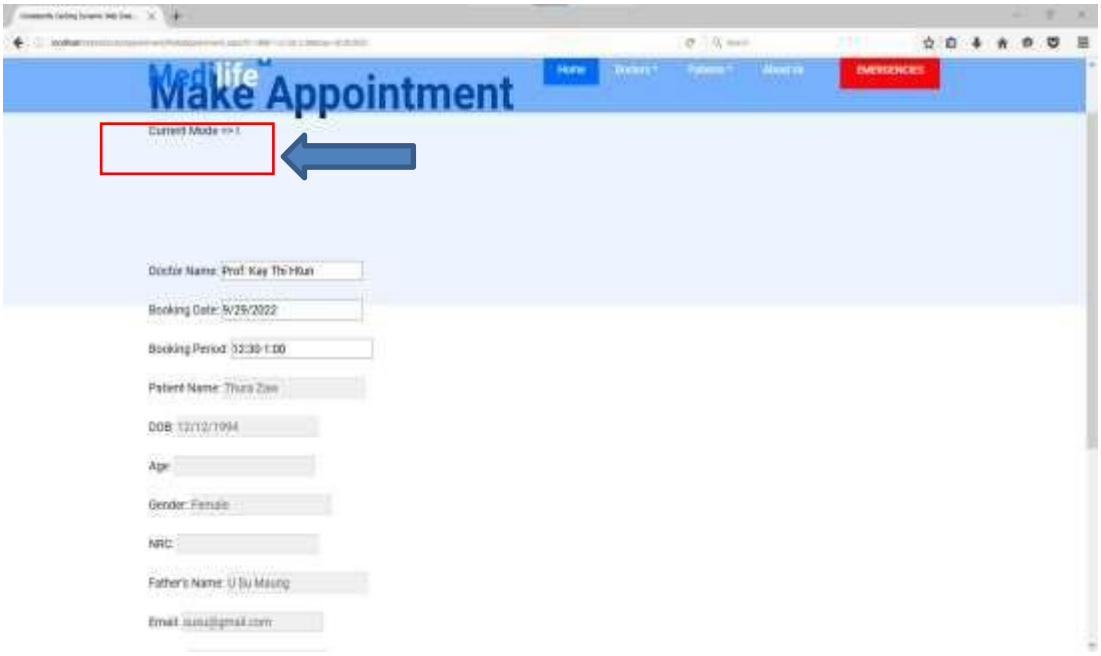


Figure 4.14: Appoint Making (I-Mode)

CHAPTER 5

CONCLUSION, LIMITATION AND FURTHER EXTENSION

The proposed system can control the data consistency by the use of MOESI protocol because of the user can notify the data inconsistency by the “Invalidation” phase. The scheduling systems are used to manage access to service providers. Thus, a proper scheduling system has to be developed by considering all factors which will increase user satisfaction, which in turn increases profit. By allowing clients to be aware of the global state of their cached data, the clients are actively involved in maintaining the cache consistency. The client directory consistency, has been designed to reduce the message overhead for maintaining client directory consistency. This system provides data sharing capabilities – on doctor appointment time schedule - can collaborate with each other effectively.

5.1 Advantages of the System

The system uses a centralized approach for handling deadlocks. The server maintains a Log Table (LT) to keep track of conflicts. When a client commits or aborts, the server removes all entries in the LT related to that client. When the server receives a write request for a page whose server page state is busy–shared or busy–exclusive, the server performs deadlock detection. If the server detects a deadlock, it informs the requester to abort. To avoid starvation, the system ensures that clients can be picked for abortion only a small finite number of times.

For the application point of views, this client server based online doctor appointment system handles with management and appointment issues according to the user's choice/needs. The patient can choose a doctor based on their medical requirements. They can check the details information of doctors and reviews for more details. They can

also see the various booking slots available and select the preferred date and time. Therefore, this system provides the effective solution.

5.1.1. Consistency Controlling

Some clients have an inconsistent directory relative to the server for the same page. As a result, those clients may have outdated presence flags in their directories. Such directory inconsistency causes a problem only when those clients want to update the page.

When the server receives a speculative update request for a shared page, the server compares its directory with that of the client. If the server detects that the client directory is outdated, it grants the speculation, but at the same time it informs the client of the discrepancy. If there are some new sharing clients, the speculative client is not allowed to commit before the new sharing clients invalidate their copies.

5.2 Limitation and Further Extension

This system implements the data sharing system for approving concurrency and consistency control with MOESI. Since it is the shared database system, it depends on the server database and client database. This proposed system can be extended priority based on validation.

AUTHORS PUBLICATION

[1] Thura Zaw, Si Si Mar Win, "Cache Coherency Control by Using MOESI Protocol in Online Doctor Appointment", Parallel and Soft Computing (PSC), UCSY, Yangon, Myanmar, 2022.

REFERENCES

[1] A. Adya, R. Gruber, B. Liskov, and U. Maheshwari, "Efficient optimistic concurrency control using loosely synchronized clocks," in Proceedings of the ACM SIGMOD Conference on Management of Data. San Jose, CA, pp. 23– 34, 1995.

[2] A. McGettrick, M. D. Thies, D. L. Soldan, P. K. Srimani, "Computer Engineering Curriculum in the New Millennium". IEEE Transactions on Education, vol. 46, no. 4, November 2019.

[3] A.M. Devices, 2006. AMD64 architecture programmer's manual vol(2), System programming.

[4] A.N. Mon, and T.T, Naing, Implementation of Client-Centric Consistency Control and Synchronization System For Distributed Replication (For Mobile Clients), Parallel and Soft Computing (PSC), University of Computer Studies, Yangon, Myanmar, 2010.

[5] A.S. Tanembaum, Organización de Computadores. Un enfoque estructurado, 3ª Ed. Prentice Hall, pp. 327-333, 2000.

[6] C. Hamacher, Z. Vranesic, S. Zaky, "Organization of Computers". McGraw-Hill/Interamericana of Spain S.A. 804 p. ISBN: 84-481-3951-8, 2019.

[7] D.A. Patterson, J. L. Hennessy, "Structures and Design of Computers". Interficie circuitería/programación. Edition 2000. Spain. Editorial Reverté, S.A, 758 p. ISBN: 84-291-2617-1, 2018.

[8] F.J. Jiménez Maturana, "Simulador de memoria caché con aplicación del protocolo de coherencia MESI". Escuela Politécnica Superior. Universidad de Córdoba. Spain. 2018.

[9] H. Shukur, S. Zeebaree, R. Zebari, O. Ahmed, L. Haji. and D. Abdulqader, 2020. Cache coherence protocols in distributed systems. Journal of Applied Science and Technology Trends, 1(3), pp.92-97.

[10] J. Handy, “The Cache Memory Book”. 2nd Ed. Academic Press. 1998. 229 p. ISBN: 2020

[11] M.J. Franklin, M.J. Carey, and Livny, M. “Transactional client-server cache consistency: alternatives and performance,” ACM Transactions on Database Systems, vol. 22(3), pp. 315-363, 1997.

[12] P.A. Bernstein, and E. Newcomer, Principles of transaction processing. Morgan Kaufmann, 2009.

[13] S. Dey, and M.S. Nair, Design and implementation of a simple cache simulator in Java to investigate MESI and MOESI coherency protocols. International Journal of Computer Applications, 87(11), 2014.

[14] S. Kanungo, and D.M. Rustom, Analysis and comparison of concurrency control techniques. International Journal of Advanced Research in Computer and Communication Engineering, 4(3), 2015.

[15] W. Stalling, Computer Architecture. 5th Ed. Prentice-Hall., 760 p. ISBN: 84-205-2993-1, 2009.

[16] Z.Z. Moe, and T.T. Naing, Implementation of Home-based lazy release consistency system for a distributed application, Parallel and Soft Computing (PSC), University of Computer Studies, Yangon, Myanmar, 2010.

[17]www.geeksforgeeks.org/cache-coherence-protocols-in-multiprocessor-system/, August, 2022.

[18] Isolation (database systems) - Wikipedia, August 2022.