BAREKENG: Journal of Mathematics and Its Applications
June 2024 Volume 18 Issue 2 Page 1359–1368
P-ISSN: 1978-7227 E-ISSN: 2615-3017
https://doi.org/10.30598/barekengvol18iss2pp1359-1368

PROVING THE CORRECTNESS OF THE EXTENDED SERIAL GRAPH-VALIDATION QUEUE SCHEME IN THE CLIENT-SERVER SYSTEM

Fitra Nuvus Salsabila[1*], Fahren Bukhari[2], Sri Nurdiati[3]

[1,2,3]*Department of Mathematics, Faculty of Mathematics and Natural Sciences, IPB University*
Jln. Meranti Kampus, Babakan, Dramaga, Bogor 16680, Indonesia

*Corresponding author's e-mail: * **fitrasalsabila@apps.ipb.ac.id***

ABSTRACT

Numerous studies have been conducted to develop concurrency control schemes that can be applied to client-server systems, such as the Extended Serial Graph-Validation Queue (SG-VQ) scheme. Extended SG-VQ is a control concurrency scheme in the client-server system that implements object caching on the client side and a locking strategy on the server side. This scheme employs validation algorithms based on queues on the client side and graphs on the server side. This research focuses on the mathematical analysis of the correctness of the Extended SG-VQ scheme using serializability as the criterion that needs to be achieved. Implementing a cycle-free transaction graph is a necessary and sufficient condition to achieve serializability. In this research, the serializability of the Extended SG-VQ scheme has been proven through the exposition of ten definitions, two propositions, three lemmas, and one theorem.

Article History:

Received: 28th February 2024
Revised: 20th March 2024
Accepted: 10th May 2024
Published: 1st June 2024

Keywords:

Client-Server;
Concurrency Control;
Graph;
Locking;
Serializability.

This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution-ShareAlike 4.0 International License.

How to cite this article:

F. N. Salsabila, F. Bukhari and S. Nurdiati., "PROVING CORRECTNESS OF THE EXTENDED SERIAL GRAPH-VALIDATION QUEUE SCHEME WITH MATHEMATICAL ANALYSYS," *BAREKENG: J. Math. & App.*, vol. 18, iss. 2, pp. 1359-1368, June, 2024.

Copyright © 2024 Author(s)
Journal homepage: *https://ojs3.unpatti.ac.id/index.php/barekeng/*
Journal e-mail: *barekeng.math@yahoo.com; barekeng.journal@mail.unpatti.ac.id*

Research Article · Open Access

1. INTRODUCTION

Technology will continue to advance, making it easier for users to connect globally [1]. Work from Home (WFH) or Work from Anywhere (WFA) has rapidly emerged as a trend, supported by advancements in video conferencing applications, file sharing, and cloud-based systems [2]. Geographical distances often separate organizational members and differing work schedules, yet collaboration tools such as real-time collaboration applications are effective solutions [3]. The client-server system architecture can serve as a suitable foundation for applications supporting real-time collaboration [4]. When multiple clients simultaneously access data from the same database, and one of the clients makes changes to the data, this can lead to inconsistency of the data [5]. Therefore, in client-server systems, a mechanism is needed to manage traffic access to shared resources to ensure data consistency, known as concurrency control [6].

Bukhari and Shrivastava [7] introduced an optimistic concurrency control scheme in client-server systems that utilize object caching on the client side. This scheme is called Validation Queue (VQ). However, in the concurrency control of the VQ scheme, complexity may arise in completing transactions as transactions increase on the client side, posing a risk of increasing the server load and impacting the server's performance degradation [8]. Therefore, the VQ scheme is modified, particularly on the server side. This modification aimed to provide an alternative scheme that simplifies the transaction completion (commit) process. The modified scheme is the Extended Serial Graph-Validation Queue (SG-VQ) scheme.

The Extended SG-VQ is a concurrency control scheme in a client-server system environment. Object caching is implemented on the client side, referred to as the cache side. Both the cache side and the server side have their validation algorithms. The Extended SG-VQ scheme retains most of the mechanisms applied in the VQ scheme. Modifications to the VQ scheme are focused on the server-side validation algorithm. In the VQ scheme, both the cache and server sides implement queue-based validation algorithms. In the Extended SG-VQ scheme, the cache side maintains the queue-based validation algorithm. In contrast, the server-side validation algorithm is modified to be graph-based with the implementation of locking strategies.

After modifications are made to the scheme, proving its correctness is necessary to ensure that the modified scheme functions correctly. The expected processing or execution is free from overlapping transactions, also known as serial execution. Serial execution can be achieved by processing transactions one by one. Serial execution guarantees data consistency because there is no overlap between transactions; each transaction views data consistently and is not affected by changes made by other transactions. Therefore, serial execution is considered correct. However, in a concurrent transaction environment, the desired process involves the system's ability to execute multiple transactions simultaneously as if they were executed sequentially, as in serial execution. This process is known as serializable execution [9]. Thus, in systems supporting concurrent data processing, achieving serializable execution becomes a goal because it can improve efficiency and performance without sacrificing data consistency. Additionally, the effect of serializable execution is equivalent to serial execution, making serializable execution also considered correct [10]. Since serializability is an essential criterion for correctness [11], serializability becomes a mathematical tool for proving the correctness of the Extended SG-VQ scheme. A proof of correctness offers a mathematical assurance that an algorithm functions as intended; following its defined specifications and produced the expected outcomes under all possible circumstances. Therefore, this research focuses on proving the correctness of the Extended SG-VQ scheme. In the previous research by Jauhari [8], hypothetical cases have been used to validate the scheme's capability. In this research, the correctness will be proven by demonstrating that the transaction execution produced by this scheme is serializable.

2. RESEARCH METHODS

This research employs formal proving methods to mathematically analyze the correctness of the Extended Serial Graph-Validation Queue scheme. The criterion used to prove the correctness of the Extended SG-VQ scheme is serializability. The serializability of this scheme is examined by:

1. Characterizing History H as a representation of possible executions that occur.
2. Demonstrating that History H is cycle-free, thus making it serializable.

2.1 Extended Serial Graph-Validation Queue Scheme

The scheme discussed in this research is the Extended Serial Graph-Validation Queue (Extended SG-VQ) scheme developed by Jauhari [8]. This scheme is an optimistic concurrency control in client-server systems with the implementation of object caching on the client side, from now on referred to as the cache side. Please refer to Figure 1.

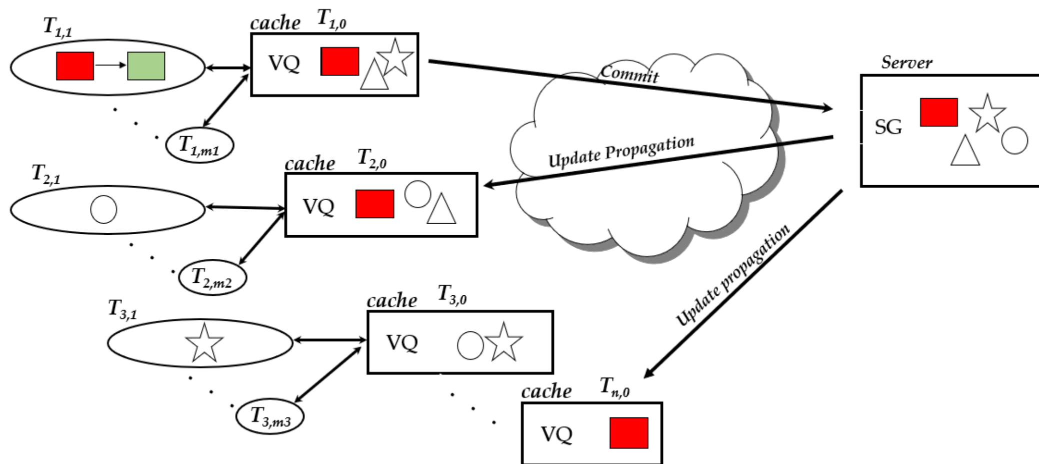

Figure 1. Extended SG-VQ Scheme

On the server side, all objects are stored. Objects accessed by each client connected to that cache are stored on the cache side. Based on Figure 1, we will examine the process of transaction 1 from cache $T_{1,0}$, denoted as transaction $T_{1,1}$. Transaction $T_{1,1}$ updates the red square object to green. This update will first be validated locally on the cache side. If it passes cache-side validation, then the red square object in cache $T_{1,0}$ will change to green according to the update from transaction $T_{1,1}$. This update is then sent to the server via a commit request message. If it passes server-side validation, then the red square object stored on the server will be updated to green according to the changes made by transaction $T_{1,1}$. Then, the cache version of the object will be updated. Transaction $T_{1,1}$ concludes. However, the server still has another task: propagating the update results to other caches that also store the square object. By receiving the Update Propagation element, all caches holding the red square object will update it to green.

Here are the validation algorithms for the cache side and the server side:

1. Cache-side validation algorithm

The cache-side validation algorithm in the Extended SG-VQ scheme is called the VQ algorithm. The VQ algorithm serves as a tool for recording the sequence of local executions. VQ consists of Read, Commit, Validated, Local Validated, and Update Propagation. Before a transaction completes its execution, a commit request is sent to the local cache manager. Upon receiving the commit request, the local cache manager creates a commit element, places it in the VQ, and then the transaction is validated.

Consider the following queue structure:

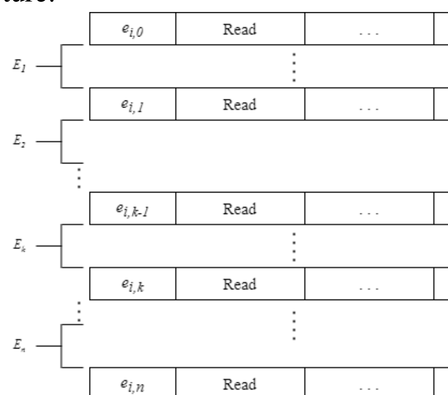

Figure 2. Queue Structure [10]

Transaction T_i comprises a range of elements $e_{i,0}$ through $e_{i,n}$, where n denotes the number of elements in the transaction. Based on **Figure 2**, these elements can exist in one or more collections $E_1, E_2, \ldots, E_n$ that consist of elements between $e_{i,j-1}$ and $e_{i,j}$. Suppose there is a particular element e' in collection E_k that splits E_k into two parts, M and N, such that $E_k = M;\ e';\ N$, where M and/or N can be empty sequences.

For the transaction T_i to successfully pass the validation process, it must satisfy either of two of the following conditions:

a. Condition I: For each $j = 0, 1, \ldots, n-1$, an element or the combined elements $e_{i,0} \cup e_{i,1} \cup \ldots \cup e_{i,j}$ doesn't conflict with any element in the sequence E_{j+1}.
b. Condition II: The elements $e_{i,0} \cup e_{i,1} \cup \ldots \cup e_{i,j}$ (where $j = 0, 1, \ldots, k-1$) must not conflict with any element in the sequence E_{j+1}, and all elements in M, except for e'. For $k = 1, 2, \ldots, n$, the combined elements $e_{i,n}$ or $e_{i,0} \cup e_{i,1} \cup \ldots \cup e_{i,j}$ (where $j = n, n-1, \ldots k+1$) must not conflict with any element in the sequence E_j. Additionally, the combined elements $e_{i,n} \cup e_{i,n-1} \cup \ldots \cup e_{i,k-1} \cup e_{i,k}$ must not conflict with the element e' and all elements in N.

If read-only transactions successfully pass the validation process, all of their elements are merged into Validated elements. Read-only transactions pass the validation process if they satisfy either condition I or condition II. Otherwise, they fail

If update transactions successfully pass the validation process, all elements are merged into Local Validated elements, and the local cache manager sends a commit request to the server. If the server response is positive, Local Validated elements are changed to Validated elements. The local elements are discarded if the server response is an abort message. Update transactions successfully pass the validation process if they satisfy only condition I. Otherwise, they fail.

2. Server-side validation algorithm

The SG algorithm is the validation algorithm on the server side of the Extended SG-VQ scheme. The SG algorithm consists of the commit request process and the validation process. When the server receives a commit request message, it checks whether the message carries the latest cache version. If the cache version carried by the message does not match the latest one, a message will be sent to the original cache manager to verify the cache version of the message and update it first. Then, if the cache version matches the latest one, the validation process will proceed. Before delving further, it's important to clarify the terms writeset and readset. In the context of transaction processing, a writeset ($Wset$) refers to the collection of elements that a transaction intends to modify or write to during its execution. Conversely, a readset ($Rset$) represents the collection of elements that a transaction intends to read from during its execution.

On the server side, a lock-based protocol is employed. In this validation process, note that T_{ij} represents the transaction undergoing validation on the server side. The server maintains a list containing objects and the transactions holding locks on each object. In this scheme, the status of an object is divided into LOCK and UNLOCK. If the status of an object is LOCK, it means that a transaction currently holds an exclusive lock on that object. If the status is UNLOCK, it indicates that any transaction does not exclusively lock the object and can be accessed by anyone.

The validation process begins with checking the status of objects, aiming to prevent multiple transactions from updating the same object simultaneously. If the status of an object is LOCK and another transaction attempts to update the same object, the later arriving transaction will be returned to its originated cache.

If the object accessed by transaction T_{ij} is not in LOCK status, then T_{ij} will be inserted into the serial graph, this means a node containing information about transaction T_{ij} will be formed. The direction of the edge for transaction T_{ij} will be determined to indicate its execution sequence. For example, if transaction T_{kl} exists in the serial graph and $Wset(T_{ij}) \cap Rset(T_{kl}) \neq \emptyset$, a serial graph will be formed as shown in **Figure 3**. This serial graph indicates that transaction T_{ij} will be executed after transaction T_{kl}.

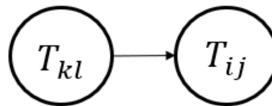

Figure 3. Transaction T_{kl} Precedes T_{ij} in Serial Graph

Furthermore, if $Wset(T_{kl}) \cap Rset(T_{ij}) \neq \emptyset$, a serial graph will be formed as shown in Figure 4. This serial graph indicates that transaction T_{ij} will be executed before transaction T_{kl}.

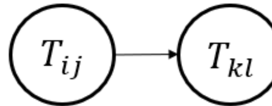

Figure 4. Transaction T_{ij} Precedes T_{kl} in Serial Graph

2.2 Serializability

One way to ensure serializability is by requiring that access to an object be done by applying exclusivity. This means that while a transaction is accessing an object, no other transaction can change that object. The commonly used method to achieve this is the lock-based protocol [12]. Two basic types of locks are used in this method: shared (S) and exclusive (X). In shared locking, a transaction can read an object but not modify it. In exclusive locking, a transaction can both read and modify an object [13].

It is generally accepted that serializability is the strongest property that can define the standard notion of correctness in a database management system (DBMS) [14]. The serializability theory is a mathematical tool used to prove whether a sequence of transaction executions is correct [15]. In serializability theory, the representation of a concurrent execution of a series of transactions is structurally called a history. An execution is considered serializable if it is equivalent to a serial execution of the same transactions. Two histories, H and H', are equivalent if,

1. both histories contain identical transactions and operations,
2. for conflicting operations r_i from transaction T_i and s_j from transaction T_j, where $a_i, a_j \notin H$ and $a_i :=$ aborted element of transaction T_i, $a_j :=$ aborted element of transaction T_j, if $r_i <_H s$, then $r_i <_{H'} s_j$. In other words, in a serializable execution, if operation r_i precedes operation s_j in history H, then operation r_i precedes operation s_j in history H' as well.

A precedence graph, or serialization graph, is commonly used to test for serializability. The serialization graph for H, denoted as $SG(H)$, is a directed acyclic graph (dag) $G = (V, E)$, where V is a collection of vertices v and E is a collection of edges e [15]. A directed graph, also known as a digraph is a graph in which its edges have a direction from one vertex to another [16]. Consider $v_1, v_2 \in V$, if (v_1, v_2) indicates that there is a directed edge from vertex v_1 to vertex v_2, and (v_2, v_1) indicates an edge directed from vertex v_2 to vertex v_1. In a directed graph, the pairs (v_1, v_2) and (v_2, v_1) are considered different because they have opposite directions, in other words, $(v_1, v_2) \neq (v_2, v_1)$, whereas in an undirected graph, they are considered the same. Applying a cycle-free transaction graph is a necessary and sufficient condition to achieve serializability [9].

2.3 Definitions and Propositions

Here are the definitions and propositions used to prove the serializability of the Extended SG-VQ scheme.

Definition 1. [10] An element e_{mn} is the n-th element of transaction T_m, where:

1. $e_{mn} \in \{r_{mn}(x), w_{mn}(x) | r(x) :=$ read operation, $w(x) :=$ write operation, $x :=$ object$\}$;
2. $Rset(e_{mn}) \cap Wset(e_{mn}) = \emptyset$, $Rset(e_{mn}) :=$ readset, $Wset(e_{mn}) :=$ writeset.

Definition 2. [10] If element e_{ip} is a compound element formed by merging element e_{im} and e_{in}, then $Rset(e_{ip}) = Rset(e_{im}) \cup Rset(e_{in})$ and $Wset(e_{ip}) = Wset(e_{im}) \cup Wset(e_{in})$

Definition 3. [10] Element $\boldsymbol{e_{mn}}$ and $\boldsymbol{e_{rs}}$ are in conflict if and only if $\boldsymbol{m \neq r}$ and satisfy one of the following conditions:

1. $Wset(e_{mn}) \cap Wset(e_{rs}) \neq \emptyset;$
2. $Wset(e_{mn}) \cap Rset(e_{rs}) \neq \emptyset;$
3. $Rset(e_{mn}) \cap Wset(e_{rs}) \neq \emptyset.$

Based on Definition 3, it can be concluded that two elements are considered to conflict if and only if they do not originate from the same transaction ($\boldsymbol{m \neq r}$), both access the same object, and at least one transaction performs an update such that the element from the related transaction is a write operation. For example, let $\boldsymbol{T_{11} = \{r(x), w(y)\}}$ and $\boldsymbol{T_{21} = \{r(z), w(y)\}}$. There is an intersection of write operations in transactions $\boldsymbol{T_{11}}$ and $\boldsymbol{T_{21}}$, resulting in a write-write conflict as per Definition 3 (**1**). Furthermore, let $\boldsymbol{T_{11} = \{r(x), w(y)\}}$ and $\boldsymbol{T_{21} = \{r(y), w(z)\}}$. Transaction $\boldsymbol{T_{11}}$ updates an object read by transaction $\boldsymbol{T_{21}}$, resulting in a write-read conflict per Definition 3 (**2**). Conversely, if $\boldsymbol{T_{11} = \{r(x), w(z)\}}$ and $\boldsymbol{T_{21} = \{r(y), w(x)\}}$, then a write-read conflict occurs as per Definition 3 (**3**) because transaction $\boldsymbol{T_{11}}$ reads an object updated by transaction $\boldsymbol{T_{21}}$.

Definition 4. [10] Transaction $\boldsymbol{T_m}$ and $\boldsymbol{T_n}$ conflict if and only if their elements or compound elements conflict.

Definition 5. [10] Transaction $\boldsymbol{T_m}$ is partial order with ordering relation $<_{\boldsymbol{i}}$, where:

1. $T_m = \{e_{m1}, e_{m2}, \dots, e_{mn}\} \cup \{a_m, c_m | a_m := \text{abort}, c_m := \text{commit}\}$;
2. $c_m \in T_m$ only if $a_m \notin T_m$, vice versa;
3. if t is c_m or a_m, then $e_{mn} <_m t$ for all e_{mn} in T_m;
4. if $r_{mn}(x) \in e_{mn}$ and $w_{m\ell}(x) \in e_{m\ell}$, then $e_{mn} <_m e_{m\ell}$ is applied.

Definition 6. [15] A complete history $\boldsymbol{H}$ over $\boldsymbol{T}$ is a partial order with ordering relations $<_{\boldsymbol{H}}$, where:

1. $H = \cup_{m=1}^{k} T_m$;
2. $<_H \supseteq \cup_{m=1}^{k} <_m$;
3. for any two elements $i, j \in H$ in conflict, then either $i <_H j$ or $j <_H i$.

Definition 7. [15] The Serialization Graph SG for a complete history $\boldsymbol{H}$ involving a set of transactions $\boldsymbol{T = T_1, \dots, T_k}$ is denoted as a directed graph $\boldsymbol{SG(H)}$. The nodes correspond to the transactions in $\boldsymbol{T}$, while the edges consist of $\boldsymbol{T_m \to T_n}$ where $\boldsymbol{m \neq n}$ indicating that one of $\boldsymbol{T_m}$'s elements precedes and conflicts with one of $\boldsymbol{T_n}$'s elements in $\boldsymbol{H}$.

Definition 8. [17] Distributed serialization order: A global history $\boldsymbol{H}$ is serializable if a total ordering of $\boldsymbol{T}$ exists in such a way that for every conflicting element $\boldsymbol{e_m \in T_m}$ and $\boldsymbol{e_n \in T_n}$ where $\boldsymbol{m \neq n}$, $\boldsymbol{e_m}$ precedes $\boldsymbol{e_n}$ in any $\boldsymbol{H_1, \dots, H_k}$ if and only if $\boldsymbol{T_m}$ precedes $\boldsymbol{T_n}$ in the total ordering.

The history contains records of committed transactions. Aborted transactions are not recorded in the history. In this study, the overall history is referred to as global history, while the history on the client side is called local history. Definition 8 explains that a transaction is said to be serial if transaction T_m precedes T_n in the global history (total order), then the serial execution with the same order also occurs in all local histories that involve both transactions (partial order). For example, if there exists a set of elements E and a total order $<_H$ over E, then for elements e_1 and e_2 in E, if $e_1 <_H e_2$, then $e_1 <_{H\ell} e_2$ also applies.

Definition 9. [10] Suppose $\boldsymbol{H_\ell}$ is a complete history at cache side $\boldsymbol{\ell}$, where $\boldsymbol{\ell = 1, 2, \dots, n}$ is partial order of $\boldsymbol{T_\ell = \{T_{\ell 1}, T_{\ell 2}, \dots, T_{\ell n}\}}$ with ordering relation $<_{\boldsymbol{H\ell}}$ where:

1. $H_\ell = T_{\ell 1} \cup T_{\ell 2} \cup \dots T_{\ell n_\ell}$;
2. $<_{H\ell} \supseteq <_1 \cup <_2 \cup \dots \cup <_{n\ell}$;
3. for any two elements in conflict such as $m, n \in H_\ell$, then $m <_{Hk} n$ or $n <_{Hk} m$.

Definition 10. [10] Suppose $\boldsymbol{T = \{T_1, T_2, \dots\}}$ is a set of transaction, $\boldsymbol{H}$ is a complete history generated by Extended SG-VQ algorithm, and there exists **k** cache side in the system. History $\boldsymbol{H}$ is a partial order over $\boldsymbol{T}$ with ordering relation $<_{\boldsymbol{H}}$, where:

1. $H = H_1 \cup H_2 \ldots \cup H_k$, where H_ℓ is complete history at cache side ℓ and H_ℓ is partial order over transaction T;
2. $<_H \supseteq <_{H1} \cup <_{H2} \cup \ldots \cup <_{Hk}$;
3. for any two elements in conflict such as $m, n \in H$, then $m <_H n$ or $n <_H m$.

Proposition 1. [10] Suppose $\boldsymbol{H_\ell}$ is the local history on cache side $\boldsymbol{\ell}$ produced by the cache-side algorithm of the Extended SG-VQ scheme. If $\boldsymbol{T_m}$ participates on cache side $\boldsymbol{\ell}$, then the execution of element $\boldsymbol{T_m}$ on cache side $\boldsymbol{\ell}$ is equivalent to the single element $\boldsymbol{e_m}$.

Proposition 1 states that if element $\boldsymbol{T_m}$ participates on cache side $\boldsymbol{\ell}$, the execution of element $\boldsymbol{T_m}$ on that cache side is equivalent to a single element $\boldsymbol{e_m}$. This indicates that when element $\boldsymbol{T_m}$ is processed on the cache side, it is considered a single transaction or atomic unit from the execution perspective on that cache side.

This proposition affirms that the operations performed on the cache side regarding element $\boldsymbol{T_m}$ are indivisible or cannot be further broken down. In transactional database systems, atomicity ensures that a transaction either completes entirely or does not start at all. Thus, in this proposition, when $\boldsymbol{T_m}$ is executed on the cache side, it behaves like an atomic transaction that must be completed fully to maintain data consistency and integrity.

Proposition 2. [10] Suppose $\boldsymbol{H_\ell}$ be the local history on the cache side $\boldsymbol{\ell}$ $(\boldsymbol{\ell = 1, 2, \ldots, n})$, $\boldsymbol{H}$ is the global history, and $\boldsymbol{T} = \{\boldsymbol{T_1}, \boldsymbol{T_2}, \ldots\}$ is a set of transactions. Let $\boldsymbol{T_m}$ and $\boldsymbol{T_n}$ are form cache side $\boldsymbol{\ell}$. If $\boldsymbol{e_m <_{H\ell} e_n}$, then $\boldsymbol{e_m <_H e_n}$.

3. RESULTS AND DISCUSSION

In proving the correctness of the Extended SG-VQ validation algorithm, it is necessary to characterize the set of histories generated by the Extended SG-VQ algorithm, which represents possible executions of transactions synchronized by the Extended SG-VQ algorithm. Therefore, a model needs to be created to characterize the Extended SG-VQ history. Let $T = \{T_1, \ldots, T_n\}$ be the set of transactions in the system, and H be the global history of T. Then, there are n clients in the system. Each client caches the required objects. Each client has a local cache manager that handles local requests. This study defines the local history H_k for client k as a set of partial orders of T.

Lemma 1. Suppose $\boldsymbol{T} = \{\boldsymbol{T_1}, \boldsymbol{T_2}, \boldsymbol{T_3} \ldots\}$ is a set of transactions and there are **n** clients in the system. Based on Extended SG-VQ scheme, each client executes a serial local history $\boldsymbol{H_1, \ldots, H_n}$. If $\boldsymbol{e_m <_H e_n}$, then $\boldsymbol{e_m <_{H_k} e_n}$ for client $\boldsymbol{k}$ that generates both transactions $(\boldsymbol{k = 1, \ldots, \ell})$

Proof. Let m and n be two clients, each making transactions T_m and T_n, respectively. Then, $e_m \in T_m$ and $e_n \in T_n$. We aim to prove that if $e_m <_H e_n$ holds in the global history, then $e_m <_{Hk} e_n$ holds in all local histories of client k that generate both transactions. If $e_m <_{Hk} e_n$, it means e_m conflicts with e_n. Based on Definition 3, three cases indicate e_m conflicts with e_n:

1. $Wset(e_m) \cap Wset(e_n) \neq \emptyset$

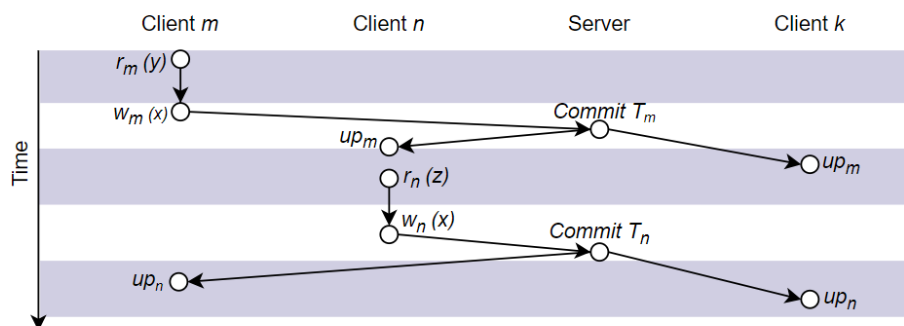

Figure 5. Serialization Graph of Case 1

Based on Figure 5, client m reads object y (denoted as $r_m(y)$) to update object x (denoted as $w_m(x)$), while client n reads object z (denoted as $r_n(z)$) to update object x (denoted as $w_n(x)$). Both clients m and n

update the same object, resulting in a write-write conflict. Since $e_m <_{Hn} e_n$, it implies that at the time when the transactions are still active and undergoing validation, client m already holds an exclusive lock on object x. As a result, client n must wait until object x returns to the UNLOCK state.

In the serialization graph $SG(H)$, the commit of T_m on the server precedes the commit of T_n to ensure that the Update Propagation element of T_m is received by T_n. The cache version of the object x brought by transaction T_n remains valid. Thus, the commit of T_m on the server precedes that of T_n, leading to $e_m <_{Hn} e_n$.

For client k, it is observed that the Update Propagation element of T_m is received before that of transaction T_n. This indicates that for clients $k = 1, \ldots, \ell$ with the same transactions, $e_m <_{Hk} e_n$ holds.

2. $Wset(e_m) \cap Rset(e_n) \neq \emptyset$

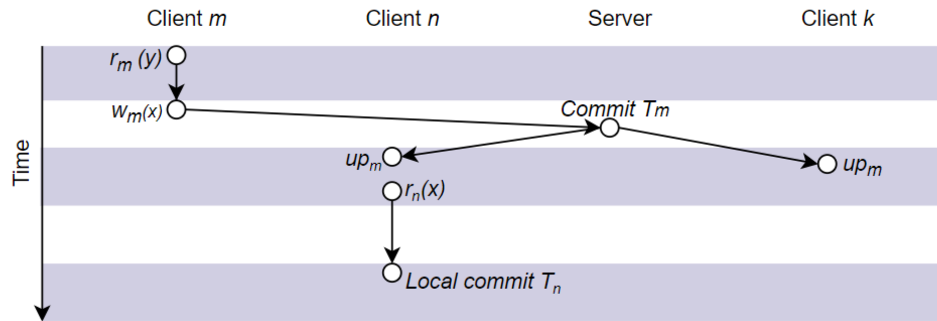

Figure 6. Serialization Graph of Case 2

Based on **Figure 6**, client m reads object y (denoted as $r_m(y)$) to update object x (denoted as $w_m(x)$), while client n performs a read-only transaction, merely reading object x (denoted as $r_n(x)$). A write-read conflict arises because client m updates an object being read by client n, which is object x. While the transactions are active, transaction T_m obtains an exclusive lock on object x during validation at the server.

In the serialization graph, since $e_m <_{Hn} e_n$, it implies that client n reads object x after receiving the Update Propagation element from transaction T_m. Thus, the object x read by client n is the latest version of the object x. Globally, $e_m <_{Hn} e_n$ holds; locally in client n, $e_m <_{Hn} e_n$ holds; and locally in client $k = 1, \ldots, \ell$, $e_m <_{Hk} e_n$ does not hold because transaction T_n is read-only and therefore does not affect client k.

3. $Rset(e_m) \cap Wset(e_n) \neq \emptyset$

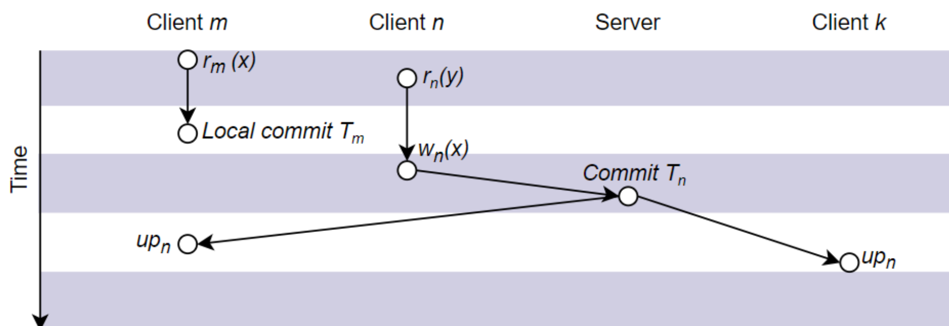

Figure 7. Serialization Graph of Case 3

Based on **Figure 7** client m performs a read-only transaction, merely reading object x (denoted as $r_m(x)$), while client n reads object y (denoted as $r_n(y)$) to update object x (denoted as $W_n(x)$). A read-write conflict arises because client m reads an object being updated by client n.

While the transactions are active, transaction T_n does a local update while client m still reads object x. However, transaction T_n can only obtain an exclusive lock on object x when validated at the server after client m finished reading object x because no transaction is allowed to make changes if an object is being locked. In the serialization graph, globally $e_m <_{Hn} e_n$ holds; locally in client n, $e_m <_{Hn} e_n$ holds, but locally in client $k = 1, \ldots, \ell$, $e_m <_{Hk} e_n$ does not hold because transaction T_m is a read-only transaction and therefore does not affect client k.

Based on the three cases, if $e_m <_{Hn} e_n$ holds for client n, then $e_m <_{Hk} e_n$ holds for client $k = 1, \dots, \ell$ that have the same transactions.

Lemma 2. Let $\boldsymbol{T} = \{\boldsymbol{T_1}, \boldsymbol{T_2}, \boldsymbol{T_3} \dots\}$ be a set of transactions, the Extended SG-VQ algorithm produces a complete history of $\boldsymbol{H}$ over $\boldsymbol{T}$, and the serialization graph $\boldsymbol{SG(H)}$ is defined. If $\boldsymbol{T_m} \rightarrow \boldsymbol{T_n}$ exists in $\boldsymbol{SG(H)}$, then $\boldsymbol{e_m} <_{\boldsymbol{H}} \boldsymbol{e_n}$ due to a conflict between validated elements $\boldsymbol{e_m} \in \boldsymbol{T_m}$ and $\boldsymbol{e_n} \in \boldsymbol{T_n}$ in $\boldsymbol{H}$.
Proof. Based on Definition 8, if $T_m \rightarrow T_n$ holds in the total order H, then there exists $e_m \in T_m$ conflicting with $e_n \in T_n$. Consequently, $e_m <_H e_n$.

Lemma 3. Let $\boldsymbol{H}$ be a complete history generated by the Extended SG-VQ algorithm and there exists a path $\boldsymbol{T_1} \rightarrow \boldsymbol{T_2} \rightarrow \cdots \rightarrow \boldsymbol{T_n}$ in $\boldsymbol{SG(H)}$ with $\boldsymbol{n > 1}$. Element $\boldsymbol{e_1}$ precedes $\boldsymbol{e_n}$ in $\boldsymbol{H}$, or $\boldsymbol{e_1} <_{\boldsymbol{H}} \boldsymbol{e_n}$.
Proof. Mathematical induction is used to prove this statement:

1. Induction Basis: Since $n > 1$, we take $n = 2$ as the induction basis. Based on Lemma 2, on the path $T_1 \rightarrow T_2$ in $SG(H)$, there exists $e_1 \in T_1$ conflicting with $e_2 \in T_2$. Consequently, $e_1 <_H e_2$, therefore Lemma 3 holds true for $n = 2$.
2. Induction Hypothesis: It is assumed that Lemma 3 holds for the case $n = k$ where $k \geq 2$ and $k \in \mathbb{Z}^+$. Based on Lemma 2, in the total order H, $e_1 \in T_1$ conflicts with $e_k \in T_k$. As a result, $e_1 <_H e_k$ because there is a path $T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_k$ in $SG(H)$.
3. Induction Step: We will prove that Lemma 3 also holds for $n = k + 1$ based on the induction hypothesis. This means, we will prove that on the path $T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_k \rightarrow T_{(k+1)}$ in $SG(H)$, e_1 precedes e_{k+1} in the total order H, or can be denoted as $e_1 <_H e_{k+1}$. Assuming that the statement of Lemma 3 holds for $n = k$, then based on Lemma 2, it is known that in the total order H, $e_k \in T_k$ conflicts with $e_{k+1} \in T_{k+1}$, therefore $e_k <_H e_{k+1}$ because there is a path $T_k \rightarrow T_{k+1}$ in $SG(H)$.

Since $\boldsymbol{e_1}$ precedes $\boldsymbol{e_k}$ and $\boldsymbol{e_k}$ precedes $\boldsymbol{e_{k+1}}$, then based on the transitive property of total order, it can be concluded that $\boldsymbol{e_1}$ precedes $\boldsymbol{e_{k+1}}$ in the total order $\boldsymbol{H}$, or can be denoted as $\boldsymbol{e_1} <_{\boldsymbol{H}} \boldsymbol{e_{k+1}}$. By proving this inductively, it has been shown that if Lemma 3 holds for $\boldsymbol{n = k}$, then it also holds for $\boldsymbol{n = k + 1}$, thus proving that Lemma 3 holds for all $\boldsymbol{n > 1}$.

Theorem 1. Every history $\boldsymbol{H}$ of the Extended SG-VQ scheme is serializable.
Proof. Proof by Contradiction: If there exists a cycle in $SG(H)$, it means $T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_n \rightarrow T_1$, where $n > 1$. Based on Lemma 3, one element from T_1 conflicts with another element from T_1 in history H; this contradicts Proposition 1, which implies that the execution of transaction T_1 is equivalent to a single element. Therefore, $SG(H)$ does not have cycles, and H is serializable.

4. CONCLUSIONS

In proving the correctness of transaction execution generated by the Extended SG-VQ scheme, ten definitions, and two propositions have been outlined, along with the proof of three lemmas to establish Theorem 1. Lemma 1 proves that if $e_m <_H e_n$ holds, then locally on client k generating both transactions, $e_m <_{Hk} e_n$ holds. Lemma 2 proves that if $T_m \rightarrow T_n$ exists in $SG(H)$, then $e_m <_H e_n$. Lemma 3 proves that on the path $T_1 \rightarrow \cdots \rightarrow T_n$, $e_1 <_H e_n$ holds for all $n > 1$. The Serializability Theorem of the Extended SG-VQ scheme demonstrates that transactions produced are cycle-free. Based on this proof, it can be concluded that every history H generated by the Extended SG-VQ scheme is serializable. Therefore, it has been theoretically proven that the Extended Serial Graph-Validation Queue scheme can execute transactions correctly.

REFERENCES

[1] S. Ali, R. Alauldeen, and R. A. Khamees, "What is client-server system: architecture, issues and challenge of client-server system (review)," *Recent Trends in Cloud Computing and Web Engineering*, vol. 2, no. I, pp. 1–6, 2020.

[2] J. Gifford, "Remote working: unprecedented increase and a developing research agenda," *Human Resource Development International*, vol. 25, no. 2, pp. 105–113, March 2022.

[3] Q.-V. Dang and C.-L. Ignat, “Performance of real-time collaborative editors at large scale: user persperctive,” in *2016 IFIP Networking Conference and Workshops*, pp. 548–553, May 17-19, 2016.
[4] M. Härtwig and S. Götz, “Mobile Modeling with Real-Time Collaboration Support,” *Journal of Object Technology*, vol. 21, no. 3, pp. 1-5, July 2022.
[5] T. Connolly and C. Begg, *Database Systems: A Practical Approach in Design, Implementation, and Management Database*, 6th ed. Essex: Pearson Education, 2015, 99-100.
[6] M. Kaur and H. Kaur, “Concurrency control in distributed database system,” *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 7, pp. 1443–1447, July 2013.
[7] F. Bukhari and S. Shrivastava, “An Efficient Distributed Concurrency Control Scheme for Transactional Systems with Client-Side Caching,” in *Proceedings of the 14th IEEE International Conference on High Performance Computing and Communications, HPCC-2012 - 9th IEEE International Conference on Embedded Software and Systems, ICESS-2012*, pp. 1074–1081, Jun. 25-27, 2012.
[8] M. F. Jauhari, “Skema serial graph-validation queue (sg-vq) pada sistem klien-server,” Master thesis, IPB Univ., Bogor, Indonesia, 2024.
[9] T. Wang, R. Johnson, A. Fekete, and I. Pandis, “Efficiently making (almost) any concurrency control mechanism serializable,” *International Journal on Very Large Data Bases*, vol. 26, no. 4, pp. 537–562, August 2017.
[10] F. Bukhari, *Maintaining Consistency in Client-Server Database Systems with Client-Side Caching*. Doctoral [Dissertation]. Newcastle upon Tyne: Newcastle Univ., 2012. [Online]. Available: Newcastle University Theses.
[11] A. Mhatre and R. Shedge, “Comparative study of concurrency control techniques in distributed databases,” in *Proceedings - 2014 4th International Conference on Communication Systems and Network Technologies, CSNT 2014*, pp. 378–382, Apr. 7-9, 2014.
[12] Fathansyah, *Basis Data*, 3rd ed. Bandung: Informatika Bandung, 2018, 312-318.
[13] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts*, 6th ed. McGraw-Hill, 2011, 661-667.
[14] T. V. A. Parreira, *Empowering a Relational Database with Lsd: Lazy State Determination*. Master [Thesis]. Lisbon: NOVA School of Science and Technology, 2022. [Online]. Available: Repository of NOVA University (RUN).
[15] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database System*, no. 2. Addison-Wesley, 1987, 1-34.
[16] Y. Heryadi and I. Sonata, *Dasar-Dasar Graph Machine Learning dan Implementasinya Menggunakan Bahasa Python*. Yogyakarta: GAVA MEDIA, 2022, 1-4.
[17] P. A. Bernstein and N. Goodman, “Concurrency control in distributed database systems,” *ACM Computing Survey*, vol. 13, no. 2, pp. 185–221, June 1981.