

A two-part multi-algorithm concurrency control optimization strategy for distributed database systems

Nasser Shebka ^{1, 2, *}¹Department of Computer Science, Northern Border University, Arar, Saudi Arabia²Computer Science College, Al Neelain University, Khartoum, Sudan

ARTICLE INFO

Article history:

Received 19 January 2022

Received in revised form

23 April 2022

Accepted 24 April 2022

Keywords:

Concurrency control

Distributed database systems

Serializability

Tier-based structure

ABSTRACT

In this paper, we propose a novel holistic approach to address the issues of concurrency control after an exhaustive examination of the problem and the various forms it can transpire. The proposed strategy was formulated depending on different perspectives that are based on exploring a wide range of algorithms, methods, and strategies proposed in practice and theory that attempted to address the problem and its forms, but only partially succeeded in doing so. Here we proposed a two-part holistic strategy to optimize concurrency control in distributed environments that address a wide range of concurrency control anomalies by taking advantage of several concurrency control algorithms' strengths while minimizing their weaknesses. The novelty of our approach transpires from two interconnected parts that can be applied regardless of the type of distributed database environment. The first is a structured tier-based data classification system based on data sensitivity with respect to serializability requirements and ranges from strict to very relaxed forms of serializability constraints. The second is a concurrency management algorithm that allocates the appropriate concurrency control algorithm to each transaction depending on the type of transaction and/or type of data being accessed from the aforementioned tier-based classification method. Our proposed method also incorporates a priority allocation mechanism within the concurrency management algorithm. Priority is allocated to different tier transactions depending on the tier's level, which in turn reflects data importance and sensitivity. Although our proposed strategy remains an algorithmic approach as we encountered various challenges regarding performance testing of a novel multi-algorithm approach for handling concurrency control in distributed database systems. However, future work involves testing the performance of our proposed strategy either through real-time systems after considerable adjustments or by constructing an appropriate customized simulation framework. Finally, the potentials of the strategy presented here are very promising, hence, we recommend as we are also optimistic that other scholars are encouraged to further exploit the concept of using multiple concurrency control algorithms within the same distributed database environment.

© 2022 The Authors. Published by IASE. This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>).

1. Introduction


Database systems (DBSs) became ubiquitous since every aspect of daily life activities became highly dependent on accessing some sort of database system. The magnitude upon which our life became

increasingly dependent on databases led to their natural development in size and complexity. DBSs evolved coherently with the emergence of the internet and networking technologies as the necessity for geographically extended databases increased exponentially replacing conventional centralized databases. Databases allow multiple users to access data items residing in the DBS simultaneously. This feature is the fundamental characteristic of databases and one of the strong points that lead to the explosive spread of databases known as 'concurrency'. Concurrency is one of the most prominent characteristic of DBSs. Concurrency

* Corresponding Author.

Email Address: nshebka2004@gmail.com (N. Shebka)

<https://doi.org/10.21833/ijaas.2022.07.016>

 Corresponding author's ORCID profile:

<https://orcid.org/0000-0002-2582-1150>

2313-626X/© 2022 The Authors. Published by IASE.

This is an open access article under the CC BY-NC-ND license

(<http://creativecommons.org/licenses/by-nc-nd/4.0/>)

control is a concept that can be easily explained but is difficult to manage. The main concern of concurrency control is to efficiently manage simultaneous transactions performed on the database with the objective of maintaining the validity and integrity of data while ensuring the ability of all authorized transactions access to all data entities and to perform their required operations. Ironically, the very same feature that signified a forte of DBSs became a main source of concern. And despite the evolvement of distributed database systems (DDBSs) from federate to the grid and then cloud computing, the problem still persists because of the very nature of the concept of concurrent access. However, several forms of anomalies occur as a result of this characteristic. Various methods, algorithms, and strategies have been proposed to efficiently manage concurrent access to DDBSs data resources, yet the problem persists. Moreover, each of these propositions has its own strengths and deficiencies depending on various parameters, such as; type of accessed data items, type of environment in terms of transaction type intensity, design and structure of the DDBS in terms of; site distribution, number of sites, used method of data replication and duplication, etc. Weaknesses of a specific concurrency control algorithm affect the entire DDBS since there is only one. The handling of multiple users' transactions interface with the DBS through intermediary database management system (DBMS) software is achieved by a dedicated concurrency control mechanism. Multiple users' simultaneous access to the same data items can result in problems and discrepancies that we refer to as anomalies (rather than problems) since they are a natural result of the DBS operations. Concurrency control anomalies are not just associated with DDBS but started with centralized DBSs and escalated with DDBS as the geographically spread DBS began growing in terms of scalability and complexity, as did the anticipated problem scenarios that are likely to occur as a result of the simultaneous transactions performed by on the same data. Centralizing concurrency control is very expensive and can even be unfeasible in some cases due to the need to use expensive high processing capacity servers and depend on the traffic state of the transferring network. These requirements in turn translate to concerns about performance and availability. Moreover, operating systems and DBMSs have many similar related characteristics such as concurrency control and deadlocks. Consequently, concurrency control is concerned basically with maintaining database consistency and deadlock avoidance through efficient transaction management and termination respectively. Additionally, It is worth mentioning that an important aspect that contributed greatly to our method design is derived from operating system deadlocks detection and avoidance strategies (Menasce and Muntz, 1979).

Optimization deals with performance measurement and comparisons with other concurrency control algorithms under the same

conditions and measuring and comparing concurrency control algorithms' performance based on various parameters, but initially, we can emphasize the followings:

- Each and every algorithm can be suitable and, hence, successful for a particular set of transaction types and scenarios of interferences but fails in managing concurrency for other types of transactions or scenarios.
- The ever-changing states of a DDBS and transactions are hard to replicate and change instantaneously.

Moreover, there are many proposed hybrid concurrency algorithms that combine two or more of the main concurrency control methods, example include algorithms such as two-phase lock multi-version, timestamp multi-versions, and Hybrid Wait-Die (Rosenkrantz et al, 1978), and many more variants (Batra and Kapil, 2010). Hybridization attempts to harvest the strength points of all involved entities while minimizing or eliminating their weaknesses. This motivated us to start considering using several concurrency control algorithms within the same DDBS, which are managed by a Controlling algorithm that assigns algorithms to transactions depending on a categorization system, built on a tier-based structure of data items classification method according to the sensitivity level of data items with a respect to serializability requirements. In addition, another motivating factor is the fact that concurrency control algorithms are difficult to compare since each is designed to address a specific number of scenarios. Hence, considering a method that combines several different types of these algorithms in order to minimize the weaknesses and increase the efficiency of each of the used concurrency control algorithms. We argue that the anomalies resulting from interfering concurrent transactions in a DDBS cannot be eliminated, although they can be scaled back or optimized.

In our work, we propose a two-part strategy for optimizing the DDBS concurrency control mechanism by using several concurrency control algorithms within the same DDBS. The objective of this strategy is to take advantage of each of the used concurrency control algorithms' strengths and minimize their weaknesses. This is achieved through limiting their effect to specific clusters of data entities, which are also classified; as we also propose; into a tier-based structure system depending on various parameters such as data sensitivity with respect to serializability requirements, type of transactions intensively performed on the specified data entities, etc... The first part is concerned with the following processes:

- Classifying concurrency control algorithms according to the level of serializability they provide.

- Categorizing data entities into a tier-based structuring system according to their importance and sensitivity.

Both processes are interrelated since each algorithm is assigned the task of managing relevant tier sites that contain data entity types and transaction types that reflect that algorithm's strengths and limit its weaknesses to that specific tier data items.

Finally, we demonstrate our proposed strategy by using an example of a DDBS that combines four known concurrency control algorithms and examine whether the proposed algorithm minimizes failure rate and, hence, optimizes concurrency control functionality.

The paper is organized as follows. In section 2, we presented a literature review for comparatively related work regarding various methods and approaches proposed to handle concurrency control issues and anomalies. In section 3, we introduced a brief review of the concept of concurrency control, definitions, types of problems and anomalies, and the ACID rules. We also presented a brief review of some proposed categorization and classification of concurrency control algorithms, as well as introduced the main types of concurrency control algorithms. In section 4, we discussed some problematic issues of concurrency control by raising a few questions. We also questioned some of the proposed methods and strategies to handle it by re-introducing the problem from different angles. In section 5, we presented our proposed two-part strategy for optimizing concurrency control in distributed environments. In the final section, we discussed challenges hindering performance testing of the proposed strategy either in real-time systems or construction of simulation frameworks, future work to be achieved, and finally, our work's conclusion.

2. Related works

Research regarding methods, algorithms, and strategies addressing the problem of concurrency control can be traced back to the emergence of the concept and technology of DBS in the late 1970s (Liu and Özsu, 2009). Various methods were proposed to handle concurrency control and to address the numerous anomalies that can result from different scenarios of concurrent access to DDBSs. These anomalies are second nature to modern DDBSs. Moreover, new forms of anomalies can emerge in contemporary DDBSs as a result of technological developments. Scientists as early as 1981, such as Bernstein and Goodman (1981), for example, introduced and discussed more than 48 principle methods and 20 concurrency control algorithms. Recent work regarding various algorithms, methods, and strategies targeting concurrency control issues can be found in a wide range of research such as Moiz (2015), Akintola et al. (2005), Bakura and Mohammed (2014), Batra and Kapil (2010),

Haapasalo et al. (2008), and Herlihy and Weihl (1991). Works such as Batra and Kapil (2010), Carey and Livny (1988), Geschwent (1994), and Kanungo and Rustom (2015) conducted surveys of concurrency algorithms and compared them against various parameters.

Furthermore, our investigation of related work extended to some commercial DDBMSs concurrency control mechanisms and related literature such as Microsoft's Azure, Apache's Hadoop, Oracle, and Amazon's EC2 (AlKhatib and Labban, 2002; Dean and Ghemawat, 2007; Li and He, 2010), despite the scarcity of information due to proprietary rights.

Likewise, examples of concurrency control algorithms classification research are also numerous. Examples of such work can be derived from works in which comparisons were conducted between various concurrency control algorithms depending on many factors such as serializability and type of transaction type intensity. Examples of such results are; the suitability of locking-based algorithms for update-intensive applications (Geschwent, 1994), the suitability of multi-version-based and certification-based algorithms for read-intensive environments (Kanungo and Rustom, 2015), and the suitability of timestamp-based algorithms for transactions' potential conflict-based environments (Silberschatz et al., 2002). While other researchers discussed a combined classification of concurrency control algorithms in the form of performance advantages of hybrid concurrency control algorithms such as the suitability of the multi-version two-phase locking algorithm for environments that are in-between update and read intensities (Carey and Muhanna, 1986). Hybrid algorithmic approaches that combine advantageous characteristics of optimistic and pessimistic approach strategies for concurrency control are presented by Moiz (2015) and Sheikhan and Ahmadluei (2013). The former paper introduces a hybrid concurrency control algorithm for mobile DBSs in which concurrency access anomalies are addressed depending on a variable transactions priority parameter. This parameter is maintained by the transaction manager for potentially conflicting concurrent transactions, it can be increased to maintain data consistency by reducing the request starvation resulting from conflict resolution strategies. The latter research introduced a hybrid intelligent concurrency control algorithm for centralized DBSs that alternates between the optimistic and pessimistic approaches to manage concurrency depending on the conflict rate value. Another hybrid locking-based concurrency control algorithm is introduced by Herlihy and Weihl (1991) which used properties of type-specific objects to provide more relaxed concurrency access while maintaining the strict serializability of locking-based concurrency control algorithms. A survey of concurrency control algorithms by Batra and Kapil (2010) classified 14 concurrency control protocol variants into two mainstream categorical approaches; optimistic and pessimistic. The survey classifies variants such as;

the divergence control lock model, secured two-phase lock algorithm, clock synchronization by message passing, and optimistic concurrency control (OCC) method. The study examined their performances considering factors such as consistency, reduced blocking, load balancing, and security. Another distributed OCC variant method (DOCC-DATI) is presented by Lindström (2004) that adjusted serialization order dynamically through timestamp intervals. Another variant of OCC is proposed by Kim and Shin (1994). The proposed protocol uses a priority-based method to effectively control concurrent transactions by combining forward and backward validation processes. Other methods of using priority assignment for concurrency optimization can be found in works such as Lam et al. (1997). A lock-free strategy for managing concurrency control in mobile environments through combining features of timestamp ordering and OCC strategies aiming at minimizing transaction abortion rate and response time is discussed by Bakura and Mohammed (2014), in which pre-commit transactions are allowed offline.

3. Concurrency control: A brief review

There are various classifications of concurrency control anomalies resulting from simultaneous access. Early work achieved by Bernstein and Goodman (1981) suggested that all different variations forms of anomalies can be traced to only two sub-problems:

- read-write
- write-write

Further classifications define three mainstream forms of anomalies as follows:

- Lost updates anomaly: As the name suggests, the update of a transaction is lost almost immediately by a successor transaction.
- Inconsistent retrievals anomaly: Also identified as the unrepeatable read problem. This anomaly occurs when two read operations belonging to the same transaction return different values.
- Dirty read anomaly: Also known as Temporary update or uncommitted data problem. It's a result of a transaction's failed attempt to update a data item. But before the failure, the data item is used by another transaction which only reads the failed-uncommitted value.
- Additional exhaustive classifications add the following two forms in addition to the previous list:
- Phantom read anomaly: This results when a transaction is able to read a variable the first time but fails the second with an error 'variable does not exist' message.
- Incorrect Summary anomaly: This anomaly results from concurrent transactions of updating and an aggregate function. This can result in data items being counted or summed before they are

immediately updated, thus, resulting in a faulty aggregate value.

A different notion of the concurrency control problem was presented by Akintola et al. (2005), which extended the definition of the problem in the form of two time-related incurred costs:

- Lost opportunity cost: That results from unnecessary wait under locks to ensure the absence of conflict and interference.
- Restart cost: that results from the unnecessary restart of some transactions due to the inaccuracy of many concurrency control algorithms in preemptively identifying the root cause of a potential anomaly or deadlock, hence, opting for improper transaction termination of all involved transactions.

It is important to indicate that in real-time commercial DDBSs the situation is far more complex, especially when considering anything in between extreme measures to handle concurrency control such as granularity levels of locking algorithms. On one hand, it's always easy to define an operation's method theoretically not considering the actual number of hits a DDBS receives pertaining to the same data items at once and the complexity of completing the operation depending on the selected method of data replication, allocation, and fragmentation (Gray et al., 1996). While on the other hand, to alternate between replication and duplication processes. not to mention additional complexities considering heterogeneous DDBS that use different DBMS software instead of a homogeneous DDBS that employs the same type of DBMS software (Kumar et al., 2013). Conclusively, the root causes of the concurrency control anomalies can easily be identified, as well as classifying the methods used by different concurrency control algorithms into two categories. However, actual scenarios that can occur in practice are numerous and can vary greatly for the aforementioned reasons. Additionally, continuous development of new technologies and devices will eventually lead to an exponential increase in the number of hits a database receives. Ultimately, this led to the existence of numerous concurrency control algorithms while anomalies still persist as a result of this discrepancy between theory and practice and the difficulty of proving the correctness and suitability of concurrency control algorithms (Bernstein et al., 1987). This is why we can only provide limited examples for the method we proposed here.

3.1. Transactions and ACID rules

These are four rules that every centralized or distributed DBS should comply with in regard to every transaction to be performed. They can be summarized as follows:

- **Atomicity:** Each transaction's operations are either entirely committed or entirely aborted, in other words, there is no such thing as a partial commit operation. The atomicity property is better characterized by the "all or nothing" rule. Atomicity is ensured by serializability (Kanungo and Rustom, 2015), a core principle and goal of concurrency control mechanisms.
- **Consistency:** This property is rather a formal regulatory one since the consistency and integrity of the database and its changing states are maintained by the restrictions and transactions operations' permissions enforced by the DBMS software and its programming at the time of the DBS creation.
- **Isolation:** Transactions must always be completely isolated during their execution. This rule is one of the core objectives of concurrency control.
- **Durability:** A record of successfully performed transactions must be persistently maintained at any time by any means necessary for later retrieval in case of failures or crashes, in other words, a committed transaction should never be lost.

It is vital to indicate that isolation in addition to serializability characterizes the ultimate goals of concurrency control (Rahimi and Haug, 2010), in addition to recoverability and distribution. Most concurrency control mechanisms generate schedules that follow the serializability rule in which transactions are sequential and isolated. Although serializability is one of the core goals of concurrency control that ensures correctness it also negatively affects performance in terms of availability.

3.2. Concurrency control algorithms

There are many algorithms that were proposed to address the problem of interference of transactions resulting from simultaneous access to the same data object by more than one user. Some researchers reduce them to only two mainstream techniques; timestamp-based and locking-based mechanisms, while others attempted to analyze and compare these mainstream types of concurrency control algorithms (Carey and Livny, 1988; Kanungo and Rustom, 2015; Bernstein and Goodman, 1981). However, and for the purpose of presenting our algorithm, in the following we briefly examine and discuss different aspects of four of the main algorithms proposed and implemented to handle concurrency control in DDBSs. Mainstream concurrency control algorithms can be summarized in the following categories:

- **Locking-based algorithms:** This type of concurrency control mechanism includes algorithms such as; Two-Phase Locking (2PL) (Gray, 1991), Strict 2PL, and Rigorous 2PL. As the name suggests, locking algorithms enforces a lock or block on a specific data item once an authorized user requests and then was granted access to that item. Hence, access to items is only granted if a lock

can be secured and no other transaction can gain access until the lock is released by a commit or abort or any termination trigger. Locking has many types and granularity levels starting from extreme locking of entire database read or write access permissions. Although lock-based algorithms are highly serializability, and thus, suitable for updating intense environments, they are inefficient in terms of time and processing cost, they are also not deadlock-free.

- **Timestamp-based Ordering algorithms (TO):** Such as Basic Timestamp Ordering (BTO) (Bernstein and Goodman, 1981), each data item is given a timestamp associated with the transaction requesting to access it. A later operation with a read request will be denied access if there is an earlier write access timestamp granted to another transaction and vice versa. Instead of using locks, every transaction is granted access but in the aforementioned manner, taking into consideration that a 'read any - write all' rule must be applied for replicated data stating that a read request can be sent to any replica while a write request should be sent and approved by all replicas.
- **Certification-based algorithms:** Also known as the Optimistic concurrency control algorithm (Sinha et al., 1985). These are also timestamp-based algorithms that use certificate exchange during a transaction commit phase. Each data item is assigned a read and write time stamp. Every transaction is granted free access to read and write on the copy of the data item residing at its site or local workspace. All transactions cohorts upon completion report to the masters which all the transactions, make a decision, and assign a global unique read and write timestamps for the data item and send them to all transaction sites in the 'prepare to commit' message as part of a two-phase commit operation (Alkhatib and Labban, 2002). If the transaction's read timestamp is the same as the global write timestamp then it is certified, else it reads the newer write timestamp that should be already locally certified. Write requests are certified if there are no later reads that have been either certified and committed or locally certified.
- **Multi-version-based algorithms:** This type is used with 2PL and Timestamp ordering in the forms of multi-version Two-phase locking and multi-version Timestamp ordering algorithms respectively. This mechanism depends on accessing older copies in the system in order to increase availability by avoiding any delayed or aborted transaction as an imperative requirement for ensuring serializability which in turn corresponds to the rule of isolation of the ACID protocol. Afterward, the final value of the data item is consolidated and other versions are updated accordingly if necessary based on the timestamp version of that particular data item. However, storing multi-versions of data requires a carefully designed database structure in order to optimize response time (Haapasalo et al., 2008).

Nevertheless, the majority of concurrency control handling algorithms and their variants can be traced to three categorical methods: Timestamp-based (Bernstein and Goodman, 1981), certification-based (Thomasian, 1998; Kung and Robinson, 1981), and locking-based algorithms (Akintola et al., 2005). Hence, if we carefully examine the locking method and investigate the criterion upon which the locks are enforced, we can deduce that prioritization of locking is also with respect to time stamping.

4. Concurrency control: Discussion and re-introducing of the problem

Concurrency control can be simply defined in relation to the objective of preventing transaction interference between users that are granted simultaneous privileged access to the same data items on different sites within a DDBS. Privileged access can be in the form of read or write. It is important to point out that, while anomalies arising from concurrent access to DDBSs persist, most of the algorithms proposed or implemented to handle concurrency control are usually complex and cannot be accurately validated (Bernstein and Goodman, 1981). The main purpose of DDBSs was to lower the cost while maintaining the effectiveness of databases and their applications, in other words; sustaining the efficiency of databases by allowing their extension across distributed locations while lowering the expensive cost of managing such coordination and processing and concurrency management using very expensive servers. The complexity of concurrency control management anomalies escalates exponentially considering methods of data fragmentation, allocation, and replication, since each method used to store data results in its own set of anomalies (Carey and Livny, 1988). However, we propose our method independent of the used techniques, although they can contribute to complexities unnecessarily. Furthermore, we argue that complete replication of the entire database contents on every site serves our purpose of proving that; although it may enhance performance to some extent at a very high cost; yet doesn't guarantee considerable optimization of concurrency control performance or elimination of resulting anomalies. Processing costs can take other forms such as a constant requirement to maintain an optimized network load balancing to ensure timely delivery of control messages and signals between the remotely located sites and prevent any sort of delays that can cause additional problems. The problem of proposing an intricate and complex algorithm to address concurrency control issues in DDBSs depends mainly on the processing factor in terms of capacity and time, not to mention any synchronization requirements between remote sites. Evidently, any inclusion of time delay in any form only increases the problem of concurrency control. Concurrency control can be viewed as a problem of resource allocation (control over data and transaction ability). Hence, the core problem of

concurrency control can be simply characterized as the synchronization procedure of read-write and that of write-write (Bernstein and Goodman, 1981). Both of which, require a sub-algorithm to address each problem. But what if we consider a separate algorithm for each type while segregating data items accordingly?

Consequently, we argue that attempting to exhaustively outline all the possible scenarios in which concurrency control anomalies can occur can be a difficult task since new unanticipated scenarios could emerge as a result of external factors such as a delay in a lock signal due to network congestion. We can also argue that the performance tradeoff of distributed concurrency control remains not properly defined despite the numerous research exhaustively describing different types of algorithms to handle concurrency control (Carey and Livny, 1988). Hence, we propose using more than one concurrency control handling algorithm within the same DDBS in response to one of the questions raised.

Based on or redefinition of the problem of concurrency control as an issue of resource allocation and a derivative of time, we propose extending the method used by contemporary operating systems in memory management through the use and employment of various techniques depending on the state and parameters to be considered in memory management such as the size of the program and data to be executed, the available free memory size and location of these frames and whether there are contingent or not. Another point of concern upon careful investigation of the first part of our proposed strategy is that the suitability of a specific concurrency control algorithm to a particular type of transaction doesn't take into consideration the type of data items being handled, which in turn raises a question on the basis of algorithm-transaction allocation determining factor; Data item type, transaction type, or combination of both?

5. Concurrency control optimization strategy

The optimization strategy we proposed is based on the basic notion that it is not only possible, but it is advisable that geographically distributed sites contain different fragments of data entities. Therefore, if a transaction requires access to two distributed sites to perform one job, then complexity escalates because not only a global concurrency control mechanism is required to manage the data consistency between the two sites, and in the entire DDBS as well, but also a global process handler is required. And although distributed process handling is somewhat easier than handling concurrency control, nevertheless, complexity increases with different distributed concurrency control scenarios in addition to distributed processing management. This relates to our previous partial redefinition of the concurrency control problem as a resource allocation issue. The strategy we propose is a two-

part method. The first part is a structured system to classify data items into tiers according to the importance and sensitivity of data in relation to the transaction type intensity of that environment. The second part is the concurrency control algorithm controller that manages transactions between tiers and sites known as the Concurrency Manager (CM). Database items can be any entities ranging from records up to clusters of tables. For simplicity, we consider tables as the data entities upon which the tier-based structure is identified. Therefore, tables classified as tier1 are the most sensitive data that should be strictly managed in terms of concurrency control. Each tier is assigned a designated algorithm from a previously specified and classified algorithms list. The classification system categorizes selected concurrency control algorithms on a 1.0–0.1 scale depending on the level of relaxation with respect to serializability. There are various variants of algorithms that can be considered for classification under our proposed serializability tier-based structure. However, for demonstrational purposes, we considered a four-tier DDBS structure with four corresponding designated concurrency control algorithms to represent distinct levels of serializability relaxation degrees. Moreover, the number of sites increases with more serializability-relaxed tiers.

The tiered level classification is associated with careful positioning of each tier site geographically, taking into consideration various factors such as:

- Transactions density of each data entity with respect to its geographical location.
- Transactions type density per site with respect to tier level.
- Network status in terms of optimization statistics and congestion.

Therefore, a necessity arises to re-classify concurrency control algorithms according to the degree of relaxation that each can provide in comparison to others, and thereafter, is assigned a simple membership value ranging from (1.0–0.1) to indicate the degree of relaxation to each algorithm. Each transaction pertaining to any data item or entity is classified accordingly and then assigned a concurrency control algorithm depending on the classification of the data item's sensitivity during the creation of that entity. Hence, it is possible to assign 10 or even more data items sensitivity tier levels upon creation.

5.1. The optimized distributed database model

Our proposed system's components structure can be built on top of any concurrency control sub-system of all proposed DDBMSs models. For instance, we can consider a widely used conventional model described by Carey and Livny (1988) as a platform to demonstrate the functionality of our method. The demonstration

model consists of the following components as shown in Fig. 1:

- The Transaction Manager (TM): Models the execution of transactions accepted from the source.
- The Data Manager (DM): Responsible for processing and handling the data by controlling DBMS information access that is stored on disk.
- The Network Manager (NM): Handles the site's communications with other sites.

There are two more components that should be mentioned, although they bare minimal weight in our discussions; the source which generates the workload for a site, and the resource manager which is responsible for managing the site's CPU, I/O, and disks, and provides their services to the transaction manager. As shown in Fig. 1, each site contains a DM, a TM, and an NM. The TM handles the user interface with DDBMS in the form of transactions and communicates with the DM, while the DM manages the actual database and local transactions. TMs communicate with local DMs.

The second part of our model involves a Concurrency Manager (CM), which is an algorithm that determines which algorithm to allocate to which transaction depending on a specified range of parameters such as source of transaction, targeted environment transactions type intensity, type on the transaction, etc. The concurrency control algorithm controller operates under a tier-based classification system; it provides a form of tradeoff but not in the convenience that has been previously applied in DDBMSs concurrency control practices.

At this stage, we propose a simple design for The CM algorithm, although more intricate functionalities can be embedded later. Logically, The NM is part of the tier's CM. The DM communicates with the tier site's CM. furthermore, no tier site CMs are required for same-tier remote sites as shown in Fig. 2.

The 4-tier structure system set as an example here is composed of the following tiers:

- Tier1 sites: Contains the most sensitive data entities (eg; account balance tables), are stored in a limited number of sites (only 2 locations) carefully geographically located, and assigned a strict 2PL algorithm as a concurrency control algorithm to ensure strict serializability.
- Tier2 sites: Contain lesser sensitive data than tier1, and therefore are assigned the 2PL algorithm that provides a more relaxed form of serializability. The number of Tier3 sites is slightly more than Tier1 sites (3 locations).
- Tier3 sites: Contain data that can tolerate a relatively more relaxed form of serializability. This tier offers moderate levels of availability in the form of performance. Tier3 sites are assigned a Multi-version 2PL algorithm.
 - Tier4 sites: Contain the most relaxed form of serializability, which translates to the highest level

of availability permitted in DDBSs. It is assigned a

multi-version Timestamp ordering algorithm.

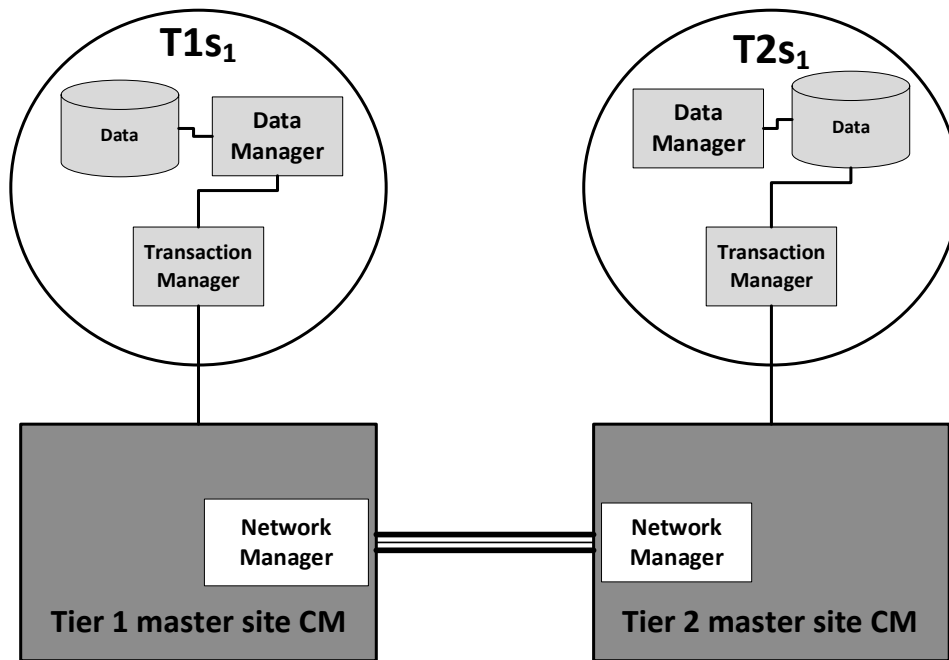


Fig. 1: Site components

Both networks in Fig. 2 and Fig. 3, which represent a single site-multitier scheme, are both reliable point-to-point (PTP) networks, however, it's possible to use VPNs through the internet as a medium of communication to reduce costs, although this may raise delay and security concerns.

5.2. Two scheme tier-based site distribution

We propose two tier-based site distribution schemes for our proposed optimization strategy. The DDBS in both schemes consists of multiple distributed sites. But the distribution and locating of tiers differ between the two schemes, for which each has its own merits and weaknesses as follows:

- **Single Site-Single-Tier scheme:** As shown in Fig. 2, each geographically distributed site contains only a single-tier. One site of each tier level is designated as the master site for that tier and controls inter-tier transactions between same tier sites. Additionally, the tier master site also contains the CM that handles transactions requesting access to that tier's data items. This scheme design is agile and simple but depends greatly on the quality and status of network communications.
- **Single site-Multi-tier scheme:** As shown in Fig. 3, each geographically distributed site can contain different multiple tiers. Tier master sites are alternately distributed so that no single site contains the master tier site of two tiers. Similarly, the tier master site contains the CM that handles transactions requesting access to that tier's data items. The advantage gained from this decentralized scheme is that each tier master site CM can process and perform any other tier's

transaction if its tier data is available at the same site taking into consideration the version of data items with respect to replication. Otherwise, it will classify and forward it to the corresponding tier master site CM. This scheme design doesn't rely greatly on network communication like the former scheme, but consequently, imposes a higher processing burden on the tier master site CM adjacent to the multi-tier cluster site.

5.3. Concurrency manager: Algorithm's design and operation method

All data items are important parts of the DDBS, but performance-wise, Our algorithm suggests a categorical classification of data items in terms of serializability requirements into crucial data items that should strictly follow the serializability scheduling mechanism, examples of such data include banks customers' credit, money, and balance transactions, and into those that can follow a more relaxed form of serializability for employee information in the form of a snapshot isolation mechanism such as the mechanism used by Haapasalo et al. (2009) with a concurrent multi-version B+-tree to allow multiple concurrent read-only transactions to access historical states of the database, or even updating transactions using relaxed forms of serializability scheduling while maintaining security in both categories. In each tier, hence, the concurrency control algorithm has a master site to minimize cost in terms of processing capacity and synchronization messages as much as possible by increasing the autonomy of concurrency control algorithms or the mechanism that oversees its handling.

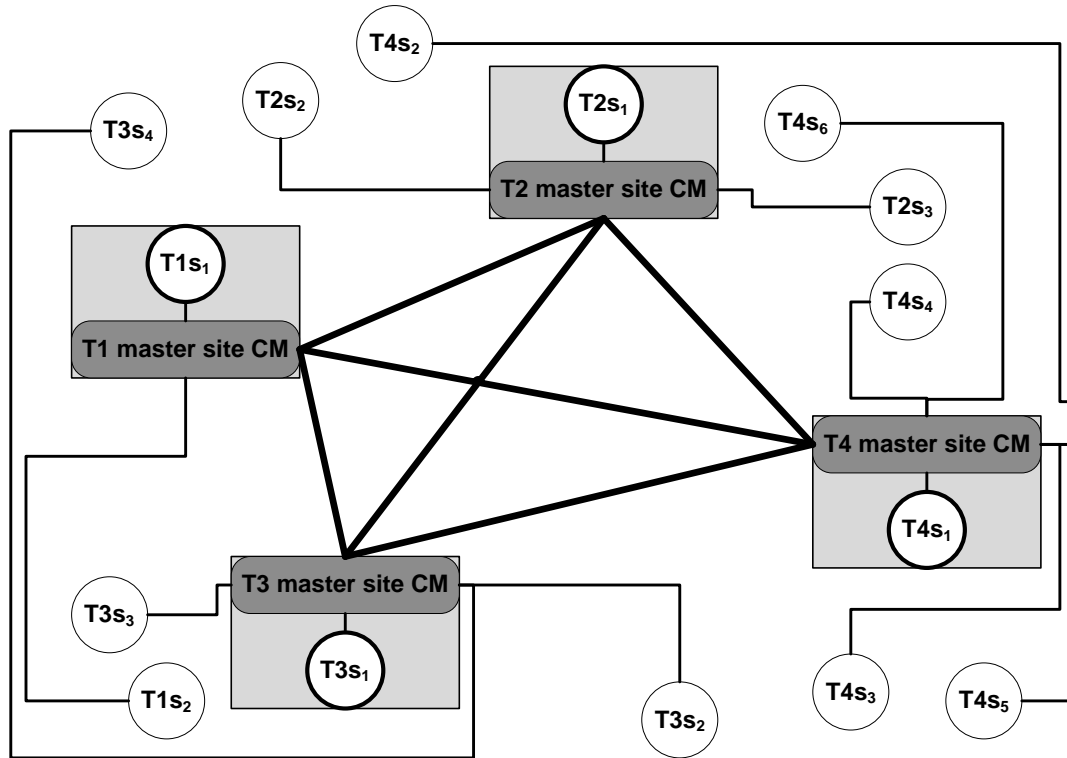


Fig. 2: Single site-single-tier scheme

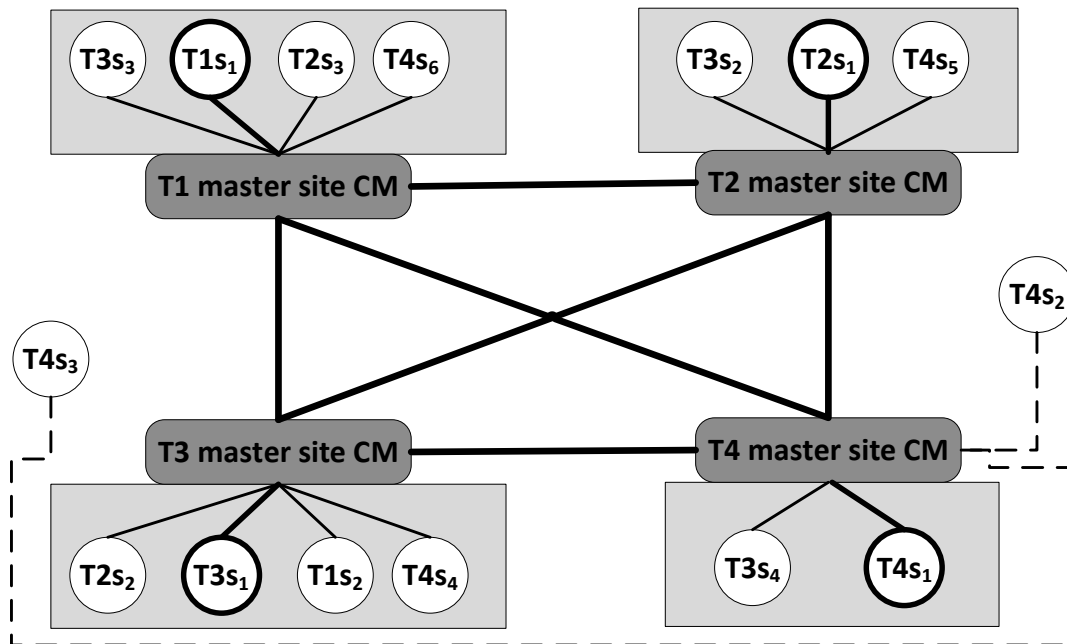


Fig. 3: Single site-multi-tier scheme

As shown in Fig. 4, the designation of a specific algorithm for every site or tier-classified data item optimizes concurrency control in the following manner:

- Each tier, hence, the concurrency control algorithm has a master site, which minimizes cost in terms of processing capacity and amount of synchronization messages as much as possible by increasing the autonomy of concurrency control algorithms or the mechanism that oversees its handling.
- Priority of every algorithm's control over all transactions requesting access to its site's data,

while acquiring a lesser priority value for other sites.

- Strength and weakness properties of each algorithm are mainly limited to the sites and data it manages. Hence, weaknesses are considerably minimized since classifying data items and aligning them with an algorithm is achieved on the basis of suitability.

The CM algorithm doesn't shift from algorithm to algorithm in the same manner that concurrency control algorithms handle control to sub-routines. It allocates a concurrency control algorithm to the transactions requesting access.

This differs from the implemented scheduling practice of using modular or global serializability and distributed serializability (Bernstein and Newcomer, 2009), although it uses a similar method to the Commit Ordering mechanism (CO) (Raz, 1992) to effectively distribute concurrency control information to maintain the compatibility between the chronological order of transactions commit events and between their precedence order. The main functions of the CM algorithm are as follow:

- Determine transactions tier according to pre-set parameters.
- Allocate the corresponding concurrency control algorithm
- Forward transactions to the appropriate master tier CM algorithm.
- Handle transactions' requests for access to data items within the site.
- Handle transactions' requests for additional access to data on other tier sites with the aid of the network manager.

The proposed algorithm's primary concern is establishing and maintaining a clear priority system for allocating algorithms between transactions, which in turn raises a question about whether; the once specified concurrency control algorithm; is assigned on the bases of site location or per transaction. Apparently, assigning concurrency control algorithms on the basis of transaction type and size for example; infers a higher cost in terms of processing capacity and delay time. The broad operation methodology of the CM adheres to the following directives:

- Transactions are handled by the concurrency control algorithm allocated to each type of data item or site's tier classification.
- Each master tier site contains a CM algorithm that handles control of requests from/to other master tier sites.
- Each CM of the master tier site handles that tier's data items requests from other master tier sites and not directly from user transactions.
- If any authorized transaction requests access to specified tier data, it is allocated to that tier's designated CM.

For the crucial purpose of deadlock avoidance, it is imperative that every local concurrency control algorithm should ensure the termination of every transaction in case of any delay, timeout, or failure of the designated handling concurrency control algorithm, provided that its tier level is higher than that of the transaction. As shown in Fig. 2, we have the following tier sites:

- Primary sites: These are tier master sites such as $T1s_1$, $T2s_1$, $T3s_1$, and $T4s_1$. they contain the concurrency manager algorithm.
- Secondary sites: These are non-master tier sites. The rest of the sites such as $T1s_2$, $T2s_2$, $T3s_2$, $T4s_2$,

and so on, fall under this category. They contain versions of the tier's data items for redundancy and performance concerns in the same manner as traditional DDBSs. Multi-versions of each tier's data entities are used since they significantly allow update and read-only transactions (Haapasalo et al., 2008).

Hence, according to the proposed tier-based structure system and its method of operation, transactions fall into one of two categories:

- Conclusive Transactions (tc_n): These are transactions requesting data items sufficiently available only to a specific tier.
- Inconclusive Transactions ($ti_{m,n}$): These are transactions that require data available in two or more different tier sites.

It is imperative to point out that each transaction is initially assumed to be conclusive (tc_n), and therefore, is assigned one of the four concurrency control algorithms according to the type of data located at its corresponding site. The operation method of the concurrency manager is shown in Fig. 4 which is a flowchart that illustrates the concurrency manager operation method,

And can be further explained considering the following scenario: Suppose a transaction is initiated, it is received by the nearest CM which classifies it as tc_1 , allocates its concurrency control algorithm type, and forwards it to its designated tier's master site. Let's suppose it is assigned a tier1 site $T1s_1$ which means being allocated a strict 2PL algorithm to handle it, and thus, follows its operation method. The transaction is terminated thereafter, once the transaction's requests are satisfied from only accessing $T1s_1$. However, If tc_1 requests are not fulfilled and require access to additional data items on a different tier site, the additional data items request is returned to the T1 master site CM, which classifies it as $ti_{1,1}$, and determines the tier of the additional requested data items site; let's say tier2, and assigns a 2PL algorithm to handle it and forwards it to T2 master site CM. transaction $ti_{1,1}$ follows the new tier directives and thereafter, is terminated once the transaction's requests are satisfied from accessing $T2s_1$ or one of the T2 secondary remote sites, thus, returning the results to the T2 master site CM, which returns it back to the T1 master site CM, complete tc_1 results, and then returns the final results to the user. Similarly, if $ti_{1,1}$ requires access to additional data items on a different tier site, the additional data items request is returned to the T2 master site CM, which then classifies it as $ti_{1,2}$, and determines the tier of the additional requested data items site; let's say tier3; and assigns a multi-version 2PL algorithm to handle it and forwards $ti_{1,2}$, which should now follow the new tier directives and its associated algorithm, to T3 master site CM, and so on. Before each inconclusive transaction is processed at its corresponding tier master site, the CM algorithm

assigns it a priority level depending on the tier it's being transferred from. In the previous example, $(ti_{1.1})$ will assign a higher priority than tier2 conclusive transactions, or transactions that only require operations and data that can be sufficiently provided by tier2. On the other hand, if a tier3 inconclusive transaction is transferred to $(T2s_1)$, it is placed on the normal queue of transactions.

Therefore, the priority mechanism reflects two issues:

- The importance and sensitivity of transactions according to data items classification.
- Availability in the form of the increased number of tier sites as data requirements with respect to serializability becomes more relaxed.

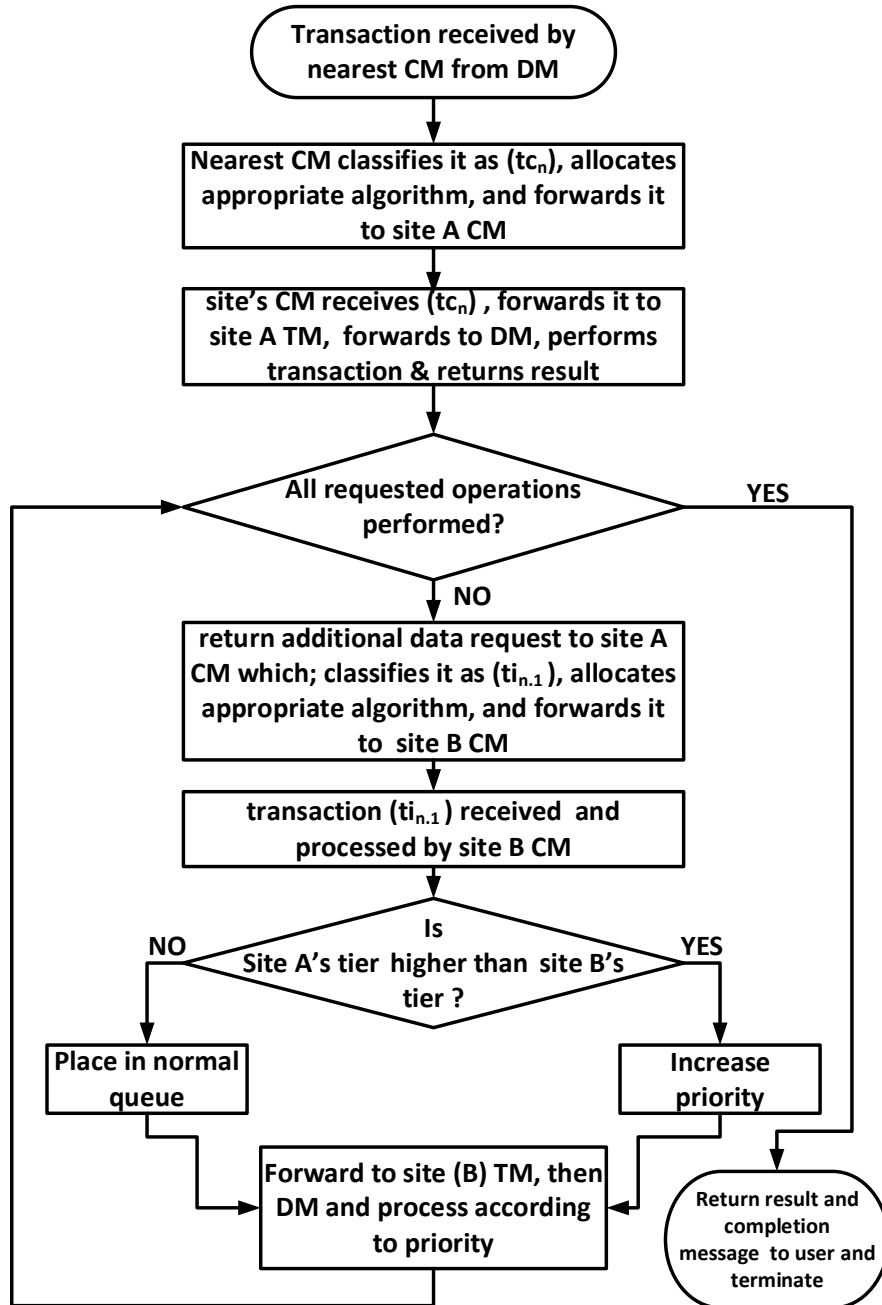


Fig. 4: The concurrency manager operation method

6. Challenges and performance testing

Up to this point, our proposed strategy remains an algorithmic approach due to its nature. Moreover, we're in the process of designing and constructing a hybrid multi-concurrency algorithm simulation framework for testing the cumulative performance of the DBS environment which incorporates multiple concurrency algorithms. Various challenges face this

task since the notion is a novel one because all DDBS in practice or theory applies only one concurrency algorithm. However, several simulation frameworks methods are being investigated and considered on the basis of the quality of their generated results, such as; hybrid concurrency simulator and distributed hybrid simulator (Bakura and Mohammed, 2014; D'Angelo et al., 2018), real-time database system simulation model presented by Kim

and Shin (1994) with a DDBS oriented environment, and real-time object-oriented database architecture (Taina and Raatikainen, 1996; Lindström, 2004) despite the fact that it's a telecommunication architecture but the concurrency principle is the same. For demonstrational purposes, we used four known types of concurrency control algorithms with relatively distinct characteristics, but empirically, discrepancies between major concurrency control variants can be vague (Batra and Kapil, 2010), thus, adding to the challenges of reaching a crisp classification of concurrency control algorithms. This can reflect/translate into the performance tradeoffs of distributed concurrency control (in too many or too less parameters and their qualitative nature) in either selecting the appropriate algorithms or classifying them in concurrence to the alignment process with tier-based classified data items. Finally, it is important to indicate for comparison purposes, that after the revision and examination of similar work and research in literature, we were not able to find any similar endeavors. Alternatively, the nearest methods to the approach presented were characterized by two works, each presenting a partial aspect of our strategy and only to some extent. The first one of these researches introduces the concept of using more than one algorithm within a hybrid intelligent concurrency control algorithm for centralized DBSs, but it only alternates between optimistic and pessimistic approaches to manage concurrency depending on the conflict rate value (Sheikhan and Ahmadluei, 2013). The second research discusses transaction priority within a hybrid concurrency control algorithm for mobile DBSs in which concurrency access anomalies are addressed depending on a variable transactions priority parameter (Moiz, 2015).

7. Conclusion and future work

Our work concentrates on designing an optimized distributed database system. We proposed a two-part strategy for optimizing distributed databases' concurrency control mechanisms. Our strategy can be easily adapted to any distributed environment, grid, or cloud. The first part of our strategy acknowledges two factors; first, the differences between data items with regard to importance and sensitivity translate into serializability degree requirements. Second, the strengths and weaknesses of each implemented concurrency control algorithm affect the entire distributed database system regardless of different types of data entities and their different requirements. We proposed using multiple concurrency algorithms within the same distributed database system. These algorithms are classified according to a proposed tier-based structure system that considers the type of transactions and data items being processed among other factors. The second part of our strategy is a concurrency manager algorithm that is designed to operate on the basis of the first part of our proposed strategy, with a simple priority mechanism that is also based on the

first part. Our intentions for future work are mainly targeted towards accomplishing two objectives; establishing a well-defined system for classifying concurrency control algorithms according to degrees of fitness to data items' types and constructing an appropriate simulator to test the performances of all possible varieties of the proposed strategy in terms of; used algorithms and their variants, structure of the system, and parameters used to classify data items. With respect to the latter objective and due to the unconventional nature of our proposal, we're considering a hybrid method of simulator construction for performance measurement. Another concern that we aim to further express and investigate is the scenarios in which conflicting data is present in the form of two values for the same data item attribute that are equal in priority as a result of using different algorithms to manage concurrency locally from two different sites.

Evidently, this paper alone is not sufficient to investigate the potentials of all different possible combinations of two or even more concurrency control algorithms considering the number of variants proposed, but nevertheless, our work may motivate other researchers to further investigate the potentials.

Acknowledgment

The author gratefully acknowledges the approval and the support of this research study by grant no. COM-2018-3-9-F-7921 from the Deanship of Scientific Research at Northern Border University, Arar, K.S.A.

Compliance with ethical standards

Conflict of interest

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

References

- Akintola AA, Aderounmu GA, Osakwe AU, and Adigun MO (2005). Performance modeling of an enhanced optimistic locking architecture for concurrency control in a distributed database system. *Journal of Research and Practice in Information Technology*, 37(4): 365-380.
- Alkhatib G and Labban RS (2002). Transaction management in distributed database systems: The case of oracle's two-phase commit. *Journal of Information Systems Education*, 13(2): 95-104.
- Bakura SA and Mohammed A (2014). Lock-free hybrid concurrency control strategy for mobile environment. In the IEEE 6th International Conference on Adaptive Science and Technology, IEEE, Ota, Nigeria: 1-5.
<https://doi.org/10.1109/ICASTECH.2014.7068146>
- Batra N and Kapil AK (2010). Concurrency control algorithms and its variants: A survey. *AIP Conference Proceedings: American Institute of Physics*, 1324: 46-50.
<https://doi.org/10.1063/1.3526261>

- Bernstein PA and Goodman N (1981). Concurrency control in distributed database systems. *ACM Computing Surveys*, 13(2): 185-221. <https://doi.org/10.1145/356842.356846>
- Bernstein PA and Newcomer E (2009). Principles of transaction processing. Morgan Kaufmann Publishers, Burlington, USA. <https://doi.org/10.1016/B978-1-55860-623-4.00004-4>
- Bernstein PA, Hadzilacos V, and Goodman N (1987). Concurrency control and recovery in database systems. Volume 370, Addison-Wesley, Boston, USA.
- Carey MJ and Livny M (1988). Distributed concurrency control performance: A study of algorithms, distribution, and replication. In: Bancilhon F and DeWitt DJ (Eds.), Proceedings of the 1988 VLDB Conference: 14th International Conference on Very Large Data Bases: 13-25. Morgan Kaufmann Publishers, Burlington, USA.
- Carey MJ and Muhanna WA (1986). The performance of multiversion concurrency control algorithms. *ACM Transactions on Computer Systems*, 4(4): 338-378. <https://doi.org/10.1145/6513.6517>
- D'Angelo G, Ferretti S, and Ghini V (2018). Distributed hybrid simulation of the internet of things and smart territories. *Concurrency and Computation: Practice and Experience*, 30(9): e4370. <https://doi.org/10.1002/cpe.4370>
- Dean J and Ghemawat S (2007). Distributed programming with MapReduce. In: Oram A and Wilson G (Eds.), Beautiful code: 371-384. O'Reilly Media, Inc., Sebastopol, USA.
- Geschwent P (1994). A survey of traditional and practical concurrency control in relational database management systems. Technical Reports, Miami University, Oxford, USA.
- Gray J, Helland P, O'Neil P, and Shasha D (1996). The dangers of replication and a solution. In the 1996 ACM SIGMOD International Conference on Management of Data, Association for Computing Machinery, Montreal, Canada: 173-182. <https://doi.org/10.1145/235968.233330>
- Gray JN (1991). Notes on database operating systems. *Operating Systems: An Advanced Course*, 60: 397-405.
- Haapasalo T, Sippu S, Jaluta I, and Soisalon-Soininen E (2009). Concurrent updating transactions on versioned data. In the 2009 International Database Engineering and Applications Symposium, Association for Computing Machinery, Cetraro-Calabria, Italy: 77-87. <https://doi.org/10.1145/1620432.1620441>
- Haapasalo TK, Jaluta IM, Sippu SS, and Soisalon-Soininen EO (2008). Concurrency control and recovery for multiversion database structures. In the 2nd Ph.D. Workshop on Information and Knowledge Management, Association for Computing Machinery, Napa Valley, USA: 73-80. <https://doi.org/10.1145/1458550.1458563>
- Herlihy MP and Weihl WE (1991). Hybrid concurrency control for abstract data types. *Journal of Computer and System Sciences*, 43(1): 25-61. [https://doi.org/10.1016/0022-0000\(91\)90031-Y](https://doi.org/10.1016/0022-0000(91)90031-Y)
- Kanungo S and Rustom DM (2015). Analysis and comparison of concurrency control techniques. *International Journal of Advanced Research in Computer and Communication Engineering*, 4(3): 245-251. <https://doi.org/10.17148/IJARCCCE.2015.4360>
- Kim J and Shin H (1994). Optimistic priority-based concurrency control protocol for firm real-time database systems. *Information and Software Technology*, 36(12): 707-715. [https://doi.org/10.1016/0950-5849\(94\)90042-6](https://doi.org/10.1016/0950-5849(94)90042-6)
- Kumar N, Bilgaiyan S, and Sagnika S (2013). An overview of transparency in homogeneous distributed database system. *International Journal of Advanced Research in Computer Engineering and Technology*, 2(10): 2677-2682.
- Kung HT and Robinson JT (1981). On optimistic methods for concurrency control. *ACM Transactions on Database Systems*, 6(2): 213-226. <https://doi.org/10.1145/319566.319567>
- Lam KY, Lee VC, Hung SL, and Kao BC (1997). Priority assignment in distributed real-time databases using optimistic concurrency control. *IEEE Proceedings-Computers and Digital Techniques*, 144(5): 324-330. <https://doi.org/10.1049/ip-cdt:19971496>
- Li JM and He GH (2010). Research of distributed database system based on Hadoop. In The 2nd International Conference on Information Science and Engineering, IEEE, Hangzhou, China: 1417-1420. <https://doi.org/10.1109/ICISE.2010.5689141>
- Lindström J (2004). Performance of distributed optimistic concurrency control in real-time databases. In the International Conference on Intelligent Information Technology, Springer, Hyderabad, India: 243-252. https://doi.org/10.1007/978-3-540-30561-3_26
- Liu L and Özsu MT (2009). Encyclopedia of database systems. Volume 6, Springer, New York, USA. <https://doi.org/10.1007/978-0-387-39940-9>
- Menasce DA and Muntz, RR (1979). Locking and deadlock detection in distributed data bases. *IEEE Transactions on Software Engineering*, SE-5(3): 195-202. <https://doi.org/10.1109/TSE.1979.234181>
- Moiz SA (2015). A hybrid concurrency control strategy for mobile database systems. *Journal of Advanced Research*, 3(4): 374-381. <https://doi.org/10.9734/AIR/2015/12406>
- Rahimi SK and Haug FS (2010). Distributed database management systems: A practical approach. John Wiley and Sons, Hoboken, USA. <https://doi.org/10.1002/9780470602379>
- Raz Y (1992). The principle of commitment ordering or guaranteeing serializability in a heterogeneous environment of multiple autonomous resource managers using atomic commitment. In the 18th International Conference on Very Large Databases (VLDB'92), Morgan Kaufmann Publishers, Vancouver, Canada: 292-312.
- Rosenkrantz DJ, Stearns RE, and Lewis PM (1978). System level concurrency control for distributed database systems. *ACM Transactions on Database Systems*, 3(2): 178-198. <https://doi.org/10.1145/320251.320260>
- Sheikhan M and Ahmadluei S (2013). An intelligent hybrid optimistic/pessimistic concurrency control algorithm for centralized database systems using modified GSA-optimized ART neural model. *Neural Computing and Applications*, 23(6): 1815-1829. <https://doi.org/10.1007/s00521-012-1147-3>
- Silberschatz A, Korth HF, and Sudarshan S (2002). Database system concepts. Volume 5, McGraw-Hill, New York, USA. <https://doi.org/10.1016/B0-12-227240-4/00028-9>
- Sinha MK, Nandikar PD, and Mehndiratta SL (1985). Timestamp based certification schemes for transactions in distributed database systems. *ACM SIGMOD Record*, 14(4): 402-411. <https://doi.org/10.1145/971699.318990>
- Taina J and Raatikainen K (1996). RODAIN: A real-time object-oriented database system for telecommunications. In: Soparkar N and Ramamritham K (Eds.), Proceedings of the workshop on databases: Active and real-time: 10-14. University of Massachusetts, Boston, USA. <https://doi.org/10.1145/352302.352306>
- Thomasian A (1998). Distributed optimistic concurrency control methods for high-performance transaction processing. *IEEE Transactions on Knowledge and Data Engineering*, 10(1): 173-189. <https://doi.org/10.1109/69.667102>