Adaptive Lockable Units to Improve Data Availability in a Distributed Database System

Khaled Maabreh
Faculty of Information Technology, Zarqa University, Jordan

***Abstract*—Distributed database systems have become a phenomenon and have been considered a crucial source of information for numerous users. Users with different jobs are using such systems locally or via the Internet to meet their professional requirements. Distributed database systems consist of a number of sites connected over a computer network. Each site deals with its own database and interacts with other sites as needed. Data replication in these systems is considered a key factor in improving data availability. However, it may affect system performance when most of the transactions that access the data contain write or a mix of read and write operations because of exclusive locks and update propagation. This research proposes a new adaptive approach for increasing the availability of data contained in a distributed database system. The proposed approach suggests a new lockable unit by increasing the database hierarchy tree by one level to include attributes as lockable units instead of the entire row. This technique may allow several transactions to access the database row simultaneously by utilizing some attributes and keeping others available for other transactions. Data in a distributed database system can be accessed locally or remotely by a distributed transaction, with each distributed transaction decomposed into several sub-transactions called participants or agents. These agents access the data at multiple sites and must guarantee that any changes to the data must be committed in order to complete the main transaction. The experimental results show that using attribute-level locking will increase data availability, reliability, and throughput, as well as enhance overall system performance. Moreover, it will increase the overhead of managing such a large number of locks, which will be managed according to the qualification of the query.**

Keywords*—*Granularity hierarchy tree; Lockable unit; Locks; Attribute level; Concurrency control; Data availability; Replication

I. INTRODUCTION

Distributed database system (DDBS) may be defined as a collection of multiple, logically interrelated databases distributed over a computer network [12]. This system stores a huge amount of data that have been accessed by a large and increasingly growing number of users. Distributed database system is a crucial source of information for numerous users who access the database locally or via the Internet for different tasks. To meet the professional requirements of users, data must be available at all times, because data availability plays a major role in the success of information systems.

Data can be accessed by a local transaction when it does not require other sites, or by a distributed transaction in which two or more database sites are involved [12]. Each site has a local transaction manager responsible for coordinating transactions across one or more database resources [1, 2]. During transaction execution, the lock manager locks the required database items by sending a message to the central site (in case a central lock manager location is used). If the requested lock is granted, then the lock manager sends a message to the requested site; otherwise, it waits. In case of a write operation, the lock manager must lock all copies of the requested database item in all sites where it exists, but in a read operation, the transaction is executed at a local copy that exists or at any copy at any available site [4,13,16].

In the study of locking techniques, the size of the lockable units clearly has a major effect on the concurrency control and the availability of data, because while the database unit is locked, it will be unavailable for a time. Thus, if the locked unit is a table, then no other transaction can access that table in a conflict mode until the lock is released. According to this problem, the study on the means to reduce database units, which in turn increases the database resources, became necessary.

The present research proposes a new approach for increasing the data availability by suggesting the attribute as a new lockable database unit. This technique may be implemented by increasing the database hierarchy tree by one more level down to include the attributes as lockable units instead of the entire row. The proposed approach may allow several transactions to access the same database row simultaneously, which may increase the degree of concurrency and the availability of data. This research uses three-phase locking instead of two-phase locking protocol. Three-phase locking protocol has a pre-commit phase to prevent the blocking state. To simplify the implementation, a central locking approach is considered, which means there is one site that has a lock manager and must coordinate with other sites in the system. Locking can be granted on some attributes of a row, including the key of that row if no conflicts among transactions could occur as the compatibility matrix adheres [4,12,13], as shown in Table I.

TABLE I. COMPATIBILITY MATRIX

	IS	**IX**	**S**	**SIX**	**X**
IS	T	T	T	T	F
IX	T	T	F	F	F
S	T	F	T	F	F
SIX	T	F	F	F	F
X	F	F	F	F	F

This research is funded by the Deanship of the Research and Graduate Studies in Zarqa University /Jordan

TABLE II. SYSTEM PARAMETERS

Parameter	Description	Values
Num-of-sites	Number of sites in the system	M
Num-of-DB	Number of databases in each site	1
Rep-degree	Degree of replication	0.2–0.8
Num-of-tables	Number of tables in a database	50
Num-of-trans	Number of transactions in the system	5000
Min-trans-size	Minimum number of operations	1
Max-trans-size	Maximum number of operations	20
OP-Mode	Operation mode	R, RW, W
Queue-Length	Maximum queue length	30
CheckT	Mean time to check a lock	1 *ms*
SetT	Mean time to set a lock	1 *ms*
RelT	Mean time to release a lock	1 *ms*
Ex-Time	Mean time to process a data object	20–150 *ms*

The remainder of this paper is organized as follows. Section 2 presents the background and literature review. Section 3 presents the proposed approach. Section 4 discusses the experiments and the produced results. Section 5 contains the conclusion.

II. BACKGROUND

The problem of data availability and the degree of concurrent transactions have been discussed by several researchers [2, 3, 8, 10] who concentrated on a strategy of dividing the database into variable size units. The size of such units is dynamically managed by the lock manager based on user needs and competition. This competition increases more in a distributed database system than in a centralized one because of the higher number of users.

A proposed simulator for a distributed object-oriented database to evaluate the concurrency control and performance of the system is presented by Norvag et al. [7]. Their simulation results show the comparison of performance and response times for two concurrency control algorithms, namely, timestamp ordering and two-phase locking. Their results show that two-phase locking outperforms timestamp ordering, specially in long transaction workload, because of the very high abortion rate in timestamp ordering. Defining new lock types and their compatibility matrix in DDBS is presented by Zhangbing et al. [17]. These types are produced to overcome the disadvantages of the traditional locking mechanism in a DDBS. Their experimental results show the enhancement of control and flexibility of locks with improved 2PL protocol and multi-granularity locking. Their improved protocol effectively ensures the serializability of scheduling transactions and decreases the communication costs while locking. It also obtains better transaction concurrency than the traditional mechanism.

Sorapak et al. [15] studied the evaluation of distributed database system performance by using MySQL cluster. Eight nodes are used to test their system. The results of their research showed the relations between query processing time and number of system nodes, which indicates that the processing time is improved when the number of system nodes is increased.

A Distributed Database Performance Tradeoff among fairness, isolation, and throughput features is studied by Jose and Daniel [6]. Their study showed that only two of the three features can be fulfilled simultaneously. Fairness means the received transactions are processed immediately without delay, isolation means a transaction cannot block or abort another transaction, and throughput means that the system will run a transaction without interference among them because of synchronization independence. Maabreh and Al-Hamami [14] implemented a study on the approach to increase the database hierarchy tree. Their study was based on a two-phase locking protocol with three sites, which represent a limited number of sites in terms of the possibility of producing insufficient results.

III. PROPOSED APPROACH

A. Data Set

To investigate the proposed idea of decreasing the size of a lockable unit, a homogeneous distributed database system consisting of ***m*** equivalent sites will be implemented. Each site has its own database, which can be accessed locally or remotely. The number of sites is extended to *m* instead of three as we studied in [14], and the numbers of database objects and concurrent transactions are increased to obtain more significant results and conclusions. In case of update operations, one copy of each object will be selected as a master copy and will be located at a specific site. The sample tables in this study have to be replicated over the system as 1-D partial replication (some objects to all sites) [10]. These tables are randomly filled with 10,000 rows as a sample of virtual data and distributed across the system. Each table has one master copy placed at one site, while the other copies are considered as replicas. To simplify the analysis of the produced results, an example of 40 sites is chosen for study. Transactions with different operation modes are also randomly generated. Table II shows the details of the system parameters that will be used in the simulation program.

B. Approach Methodology

Fig. 1 shows a sample of a multiple granularity tree, which represents the organization of data within a database. Granularity is a dynamic size of the database item, which may be locked by a transaction. Multiple granularity locking will allow several transactions to be executed at different sizes, ranging from whole database to a specific row, because transactions often do not need to lock at the higher granularity and therefore, free up objects that would otherwise still be locked. This research presents an attribute as a new granule size, because whenever the granule is being as small as possible, then more transactions could be executed simultaneously, which increases concurrency in the system.

Before building the simulation program, three essential components were implemented: the transaction manager, the lock manager, and a tree to represent the database objects. The transaction manager is responsible for keeping track of the executed transactions from the start state to the terminated state, whether its commit or abort. The lock manager is responsible for acquiring the locks needed by the transaction and ensuring that no conflict may occur among transactions by

adhering to the compatibility matrix (Table I). The lock manager uses the three-phase locking protocol by assuming that one site could fail in the system, because multiple site failures could cause a problem as proved in [11]. Three-phase locking protocol is a nonblocking protocol because it includes a pre-commit phase. This phase is reached if all transaction participants have voted to commit; in this case, the transaction is committed unless a site fails as a timeout period may indicate. Otherwise, and if this state is not reached, the participant will be aborted and the blocked resources will be released. Finally, a hierarchy tree representing the multiple granularity of the database objects (Fig. 1) is implemented by using NetBeans IDE 8. Deadlock can be detected by using a predefined timeout. Data used in this approach are virtual, so the total execution time for a database object can be guessed in advance according to the system parameters shown in Table II and by using the following formula:

Total-processing-time= CheckT + SetT + RelT + Max (Ex-Time). Given that the timeout is considered a time limit on how long a transaction may be active, when a transaction exceeds this time, the transaction has a deadlock.

As an example, the maximum total execution time for a specific database object could be 153 *ms* (i.e., 1+1+1+150) by assuming that the object has a maximum processing time. Thus, if the waiting time for the database object that has been locked by a transaction exceeds 153 *ms*, then the system has a deadlock and must be solved. A drawback of this deadlock detection approach is the long time taken to detect a deadlock.

C. Approach Description

This research aims to include the attributes that would be the new lockable units for allowing several transactions to access the same database row concurrently. This approach may increase the database resources, which would increase the concurrency and throughput in the system and decrease deadlock occurrences. In contrast, the overhead may be increased, but it will be managed as will be explained in a later part of the research. Fig. 1 shows the suggested attributes as new lockable units; the attributes may be locked individually when a transaction requires only some attributes of the database row. This "locking" can be performed as explained in [9] and as follows.

1) The database row is locked in an intent exclusive mode (IX).

2) The key of that row is locked in a shared mode (S).

3) The required attributes can be locked by the requested transaction in read or write.

4) Consistency constraints between attributes are considered. For example: if A=B+C is a constraint among the attributes A, B, and C, then when any transaction needs any of those attributes, the lock manager locks all of them to satisfy that constraint.

The following example may illustrate the importance of the idea.

Let R ($\underline{A_1}$, A_2, A_3, A_4, ... , A_n) be a database relation (table). $A_1, A_2, ..., A_n$ are the attributes of R, A_1 is the key, and let t=<v_1, v_2, v_3, v_4, ..., v_n) be any tuple in R. Furthermore, consider the following three transactions T_1, T_2, and T_3, which need to access the same database row according to the following scenarios:

T_1: Update R Set A_2= A_2+N, where A_1=v_1;

T_2: Update R Set A_4= A_4+M, where A_1=v_1; and

T_3: Select A_n from R, where A_1=v_1;

where v_1 is the value of the attribute A_1 in the same row, and N and M are some values. In the current situation of a database, these transactions cannot be executed simultaneously because the database row is considered as one block and can be locked by only one transaction; the other transactions will be waiting until the locked row is released. In this example, T_2 has to wait for T_1 to finish (because the data requested by T_2 are not yet available), and T_3 has to wait for both T_1 and T_2 to finish for the same reason of T_2. This scenario is an example of increasing the average waiting time and reducing the data availability.

When the proposed idea is implemented, all the three transactions in this example will be executed at the same row simultaneously. T_1, T_2, and T_3 will lock the database row in an intent exclusive mode (IX), and thus the key of that row will be locked as a shared (S), T_1 will lock A_2 as an exclusive (X), T_2 will lock A_4 as an exclusive (X), and T_3 will lock A_n as shared (S). This technique may reduce the waiting time and increase the average response time and data availability.

To implement the suggested approach, an event-driven simulation program was built by using Java Netbean IDE version 8.1. The simulator contains the necessary components for a distributed database system, as shown in Fig. 2. Transactions are treated as threads to be executed concurrently and managed by the transaction manager. The lock manager is responsible for locking and releasing the database items as the transaction need arises, as well as communicates with the transaction manager and with the database through a network. For simplicity of analysis, the network model is assumed to be LAN of 5 *ms* inter-processing time based on Gray and Reuter measurement [5].The database is implemented as a hierarchy tree representing the granules. Transactions are then requested by the granule size as needed.

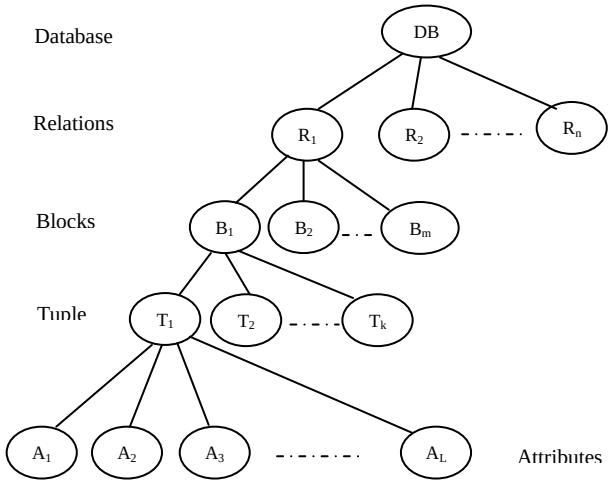

Fig. 1. Sample of Multiple Granularity

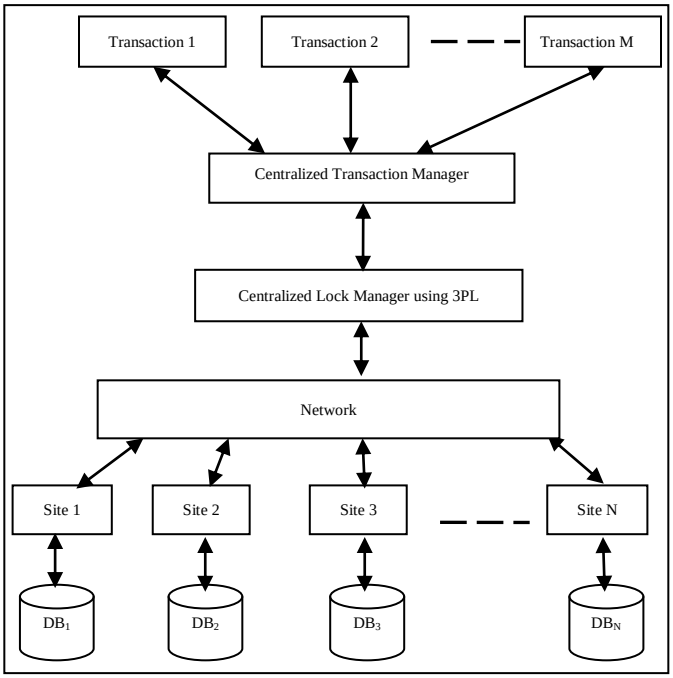

Fig. 2. Simulation Program Architecture

IV. DISCUSSION

This section demonstrates the results of the series of experiments conducted to evaluate the effectiveness of the proposed approach. Five hundred transactions are generated randomly as a sample run with different operation modes. Experiments are repeated several times with different random numbers of transactions under different degrees of replication to observe and evaluate the system behavior. The simulation program is running in both cases: one when the database row has the lowest granule, and the other when allowing the transaction to access specific attribute(s) of the row. Then, the results concerned with average execution time, average waiting time, and the overhead are collected and analyzed.

Fig. 3 shows the average execution time of the system. The average execution time of the proposed approach is less than the average execution time of the current lock mechanism, which means that the transaction almost does not need much time to acquire its requested operation because of the attribute sharing among transactions. The locks are incrementally and dynamically acquired, so the transaction may request an attribute of the row while the other attributes may be available or acquired by another transaction. This finding means that the transactions access the needed attributes without delay, which reduces the average execution time.

The average waiting time of the proposed approach is also less than that of the current lock mechanism (Fig. 4) for the same reason. The overhead that occurs because of attribute-level locking is managed by locking the database row when the transaction requires numerous attributes of the same row. Fig. 5 shows the increasing overhead and the management of locks at the row level instead of the locking of many attributes. An investigation of Figures 3 to 5 reveals that the system performance of 270, 280 and 290 transactions as a workload seems to be the same in terms of average execution time, average waiting time, and overhead (i.e., The two lock mechanisms operate the same, which is the lock manager's decision). This outcome is attributed to the numerous attributes of the same row a transaction requires, causing the lock manager to return one level up on the tree and attempt to lock the row. This finding is the same result as the one that occurred when a transaction needs multiple rows of the same table; in this case, the lock manager locks the whole table in order to reduce the overhead.

Reducing the average execution time and average waiting time increased data availability because the times to lock and access the data are decreased. Transactions in this case may obtain the data immediately or by less waiting time as much as possible. For example, by using the company schema, Fig. 9.2 in Elmasri and Navathe (2015) [4], the following two transactions will be processed together at the same database row.

T_1: Update Employee Set Salary=Salary+N, where SSN=M;

T_2: Update Employee Set Super-ssn=new Super-ssn, where SSN=M; where M is the same employee number. This example shows that the lock manager locks SSN because both transactions are shared, and then the salary is locked exclusively for T_1 and the super-ssn as an exclusive for T_2. This result means that, both transactions T_1 and T_2 will be executed concurrently because the data are available.

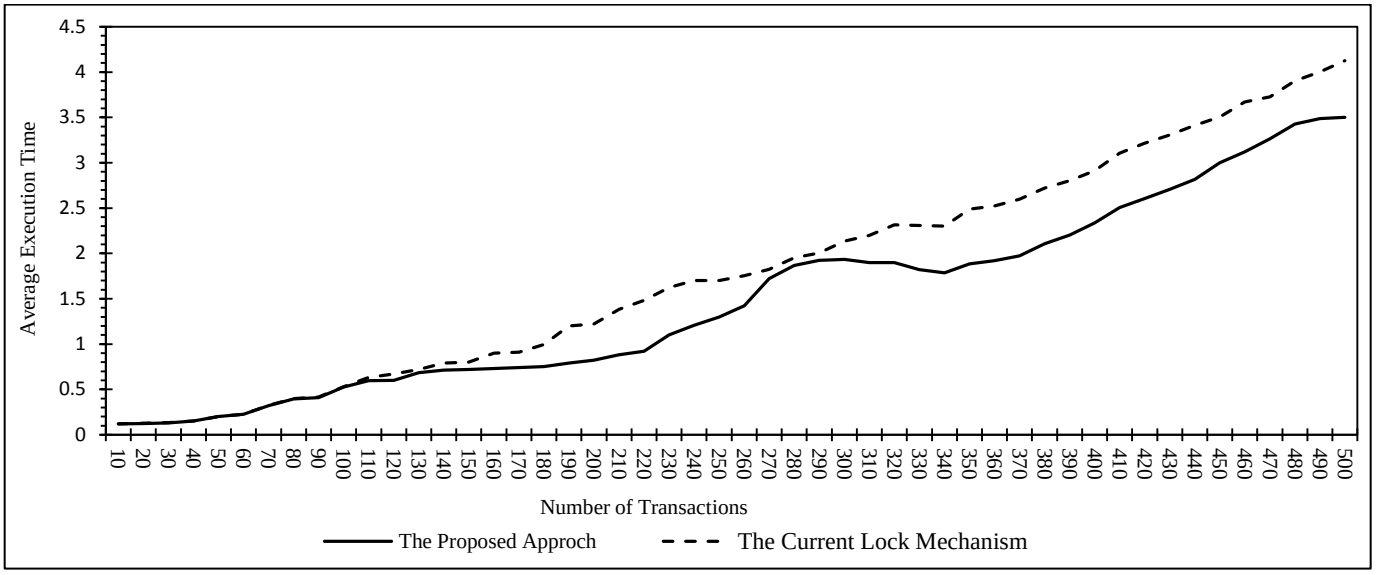

Fig. 3. Average Execution Time

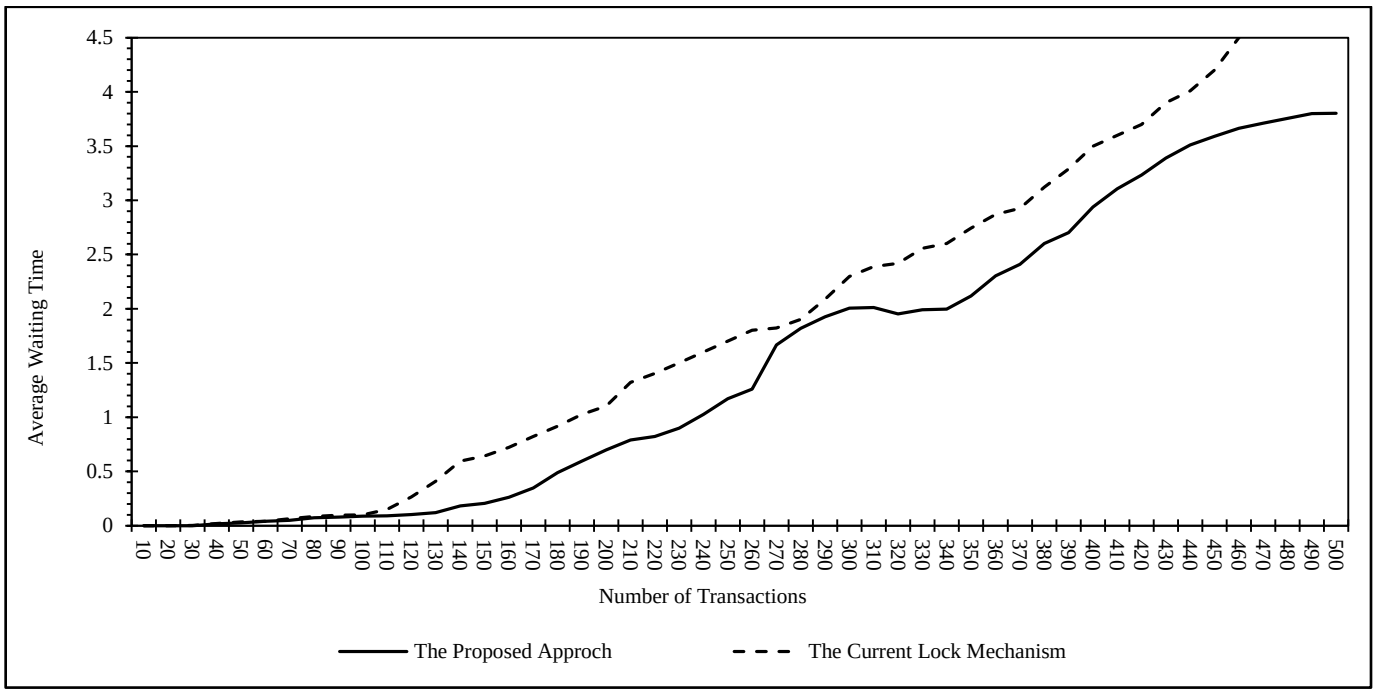

Fig. 4. Average Waiting Time

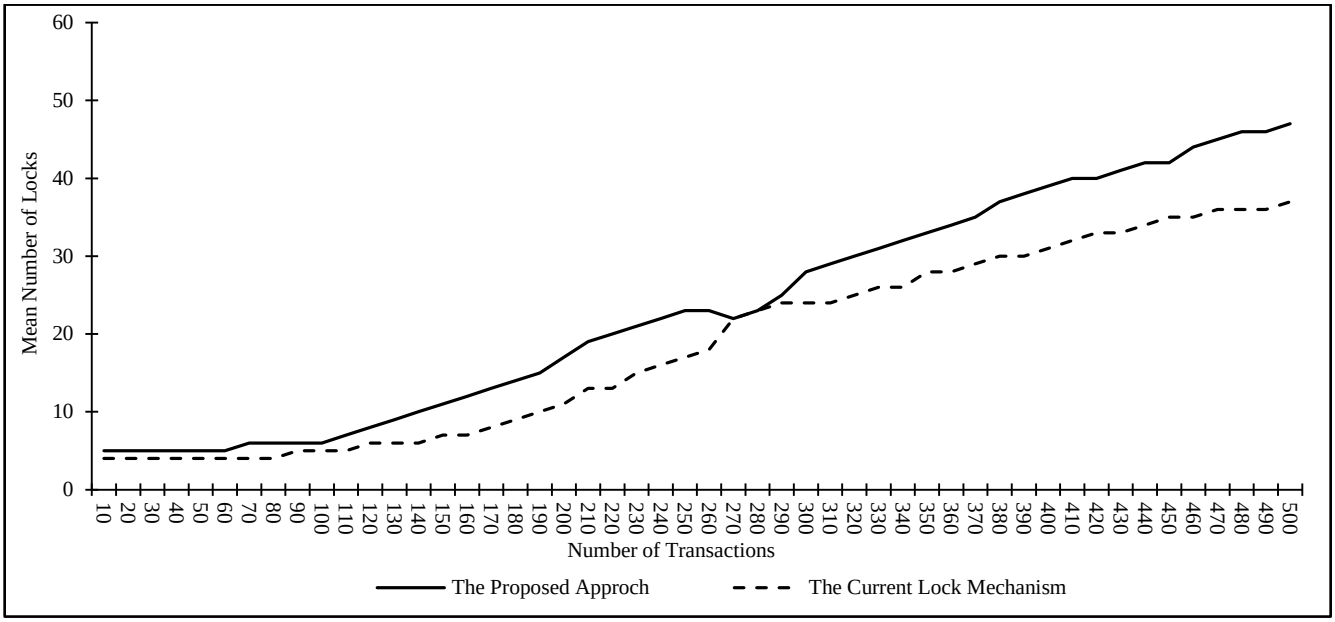

Fig. 5. System Overhead

V. Conclusions and Future Work

Distributed database systems are considered crucial sources of information. Therefore, data contained in such systems must be available at all times as much as possible to satisfy the user's professional needs. To increase the data availability, this paper proposes a new adaptive approach to increase the database items by reducing the size of the lockable units. This reduction can be carried out by locking the attributes instead of the database row, which remains as the other attributes become available for other transactions. The experimental results showed that using attribute-level locking increases the degree of concurrency by increasing the data availability. The overall system performance is also improved because the average waiting time is decreased. The increasing overhead is managed by returning the lock at the row level when a transaction requires many attributes of the same row. The proposed approach is suitable with short transactions of mixed read and writes operations, especially when the degree of replication is less than 50%.

Further work for studying the proposed approach could be implemented by having more sites, larger data set and a higher workload, as well as more practical examples, experiments and comparison with other technologies will also be studied as a future work to improve the quality of the research.

References

[1] Bernstein P. and Newcomer E., "Principles of Transaction Processing for the Systems Professional", 2nd ed. Morgan Kaufmann Publisher, 2009.

[2] Chandy K., Misra J. and Hass L., "Distributed Deadlock detection", *ACM Transactions on Computer Systems*, Vol. 1, No. 2, 1983.

[3] Croker, A. "Improvements in Database Concurrency Control with Locking", *Journal of Management Information Systems*; Vol. 4 Issue 2, 2001.

[4] Elmasri R. and Navathe S. "Fundamentals of Database Systems", *Pearson Addison Wesley*, 7th edition, 2015.

[5] Gray J. and Reuter A., "Transaction Processing: Concepts and Techniques", San Francisco, Calif.:Kaufmann, 2011.

[6] Jose M. Faleiro and Daniel J. Abadi. "FIT: A Distributed Database Performance Tradeoff", IEEE Data Engineering Bulletin, 38(1): 10-17, 2015.

[7] Kjetil Norvag, Olav Sandsta, and Kjell Bratbergsengen, "Concurrency Control in Distributed Object-Oriented Database Systems", Advances in Databases and Information Systems, 1997.

[8] Maabreh K. and Hamami A., "Increasing database concurrency control based on attribute level locking", *on the proceedings of International Conference on Electronic Design, ICED, IEEE*, pp1-4, Issue 1-3, Malaysia, Penang. Dec. 2008.

[9] Maabreh K. and Hamami A., "*Implementing New Approach for Enhancing Performance and Throughput in a Distributed Database*", The International Arab Journal of Information Technology, Vol. 10, No. 3, May 2013.

[10] Matthias N. and Matthias J., "Performance Modeling of Distributed and Replicated Databases", *IEEE transactions on knowledge data engineering*, Vol.12 No.4, pp 645-672, July 2000.

[11] Muhammad Atif, "Analysis and Verification of Two-Phase Commit &Three-Phase Commit Protocols", International Conference on Emerging Technologies (ICET), pp:326-331, Islamabad,19-20 Oct. 2009.

[12] Ozsu T. and Valduriez P., "Principles of distributed database systems", *Springer science and business*, 3rd edition, New York, 2011.

[13] Silberschatz A., Korth H. and Sudarshan S. "Database System Concepts", *McGraw-Hill*, New York, 6th edition, 2010.

[14] Sinha M. " Constraints: consistency and integrity", *ACM SIGMOD*, Vol. 13, Issue 2, New York, 1983.

[15] Sorapak Pukdesree, Vitalwonhyo Lacharoj and Parinya Sirisang, "Performance Evaluation of Distributed Database on PC Cluster Computers using MySQL Cluster", Proceedings of the World Congress on Engineering and Computer Science Vol. I WCECS San Francisco, USA. 2010.

[16] Weikum G. and Vossen G., "Transactional Information Systems, Theory, Algorithms and the Practice of Concurrency Control and recovery", *Morgan Kaufman Publishers*, 2002.

[17] Zhangbing Li, Zilan Zhu and Shaobo Zhang, "Locking Mechanism for Distributed Database Systems", Journal of Networks, Vol. 9, No. 8, 2014.