A NOVEL MULTI GRANULARITY LOCKING SCHEME BASED ON CONCURRENT MULTI - VERSION HIERARCHICAL STRUCTURE

Ms. Swati[1], Dr Shalini Bhaskar Bajaj[2], Dr Vivek Jaglan[3]
[1]Research Scholar, CSE Department, Amity University Haryana, Gurgaon, India
[2]Professor, CSE Department, Amity University Haryana, Gurgaon, India
[3]Professor and Dean Research, CSE Department, Graphic Era Hill University, Dehradun, India
Email: swattiguptta@gmail.com, sbbajaj@ggn.amity.edu, jaglanvivek@gmail.com

Abstract: We present an efficient locking scheme in a hierarchical data structure. The existing multi-granularity locking mechanism works on two extremes: fine-grained locking through which concurrency is being maximized, and coarse grained locking that is being applied to minimize the locking cost. Between the two extremes, there lies several pare to-optimal options that provide a trade-off between the concurrency that can be attained. In this work, we present a locking technique, Collaborative Granular Version Locking (CGVL) which selects an optimal locking combination to serve locking requests in a hierarchical structure. In CGVL a series of version is being maintained at each granular level which allows the simultaneous execution of read and write operation on the data item. Our study reveals that in order to achieve optimal performance the lock manager explore various locking options by converting certain non-supporting locking modes into supporting locking modes thereby improving the existing compatibility matrix of multiple granularity locking protocol. Our claim is being quantitatively validated by using a Java Sun JDK environment, which shows that our CGVL perform better compared to the state-of-the-art existing MGL methods. In particular, CGVL attains 20% reduction in execution time for the locking operation that are being carried out by considering, the following parameters: i) The number of threads ii) The number of locked object iii) The duration of critical section (CPU Cycles) which significantly supports the achievement of enhanced concurrency in terms of the number of concurrent read accesses.

Keywords: Hierarchical structure, Scalability, Concurrently, Synchronization, Performance.

I. Introduction

Synchronization mechanism is a fundamental building blocks for designing applications that are being run concurrently .The applications such as storage system [1], operating system [2, 3, 4, 5], network system [6, 7] and database system [8] completely rely on these synchronization mechanism which plays an integral role in tuning their performances [9].A commonly used mechanism for obtaining synchronization mechanism in the database domain [10] is locking. The locking mechanism ensures that each data item is controlled by a single thread of control, each time available to whosoever holds the lock. This is turn guarantees data integrity, by not allowing conflicting locks to take place together[10].The different implementation exist depending on various factors such as the way the lock request is granted and released, as well as the granularity of the data item[10].

In order to support locking operation at different hierarchical structure Multiple Granularity Locking (MGL) scheme was introduced which poses scalability challenges for both coarse grain and fine grain [11].This hierarchical locking thereby involves the following schemes: shared lock, exclusive lock and intension exclusive lock. The presence of intension locks provides notification to other transactions thereby reduces the cost of finding the entire element for performing locking operations. Even though MGL effectively gains serializability it still fails to achieve higher concurrency due to pseudo-conflicts [32] that exist among multiple transactions.

With the view to reduce the traversal cost for MGL locking, there have being several attempts made by [11, 12, 30] which quickly checks for overlaps regions. Most of the work is concentrated only on the parameter of reducing the traversal cost. Unfortunately the scheme does not focus on achieving higher concurrency among multiple transactions that perform data access operations in hierarchical structure.

In order to support enhanced concurrency without any phantom avoidance we proposed Collaborative Granular Version Locking(CGVL) scheme which is an extension of the MGL framework that supports multiple versions of the data items that support better scalability in terms of throughput. In case of read-write conflicts (IS-X,IX-S,S-SIX,S-X) mode unlike MGL,CGVL avoids synchronous waiting for data item reclamation by providing suitable version for each data item[31].It works on timestamp ordering principle to provide a consistent snapshot of each data item. Each

ISSN (Print): 2204-0595
ISSN (Online): 2203-1731

requesting thread selects the suitable version of the data item by using the last committed timestamp of each version(when the version was committed).This is well known approach that exist in multi version control database scheme [14-18] that provide multiple version for each transactional operations to perform their work.

However, it is difficult to design a scalable synchronized framework that is based on MVCC concepts which include the following 1) a global timestamp allocation scheme 2) An efficient garbage collection scheme.

In particular the work comprises of following contributions.

- The scalability issues has being explored with the proposed locking mechanism that shows consistent performance in a hierarchical data structures.
- It offers maximum degree of concurrency by supporting simultaneous execution of read-write operations which is not possible in the existing MGL locking protocols.

The proposed CGVL algorithm have being evaluated against existing intention-lock-based locking mechanism and we have study their relative performances. The study presents scenarios under which CGVL outperform existing intension locking protocol. The results were obtained by using Java Sun JDK platform that shows significant performance in the system.

The organization of the paper is as follows: In Section 2 literature review is being covered that comprises of existing locking approaches and their associated pros-cons. In Section 3 Phases of CGVL algorithm is being discussed. Section 4 quantitatively evaluate the effectiveness of CGVL and compare its performance with the existing intension based locking protocol. Section 5 provides conclusion.

II. Related Work

Some of the important work carried out in the field of multiple granularity and multi version locking protocols are discussed in table 1.

Table 1: Literature Review

Locking Technique	Approach	Pros	Cons
Multiple Granularity(MGL)	Dom lock[11]	The number of nodes being locked is being reduced. Thereby, graph traversal cost is being reduced by selecting an arbitrary dominator in the hierarchical tree structure.	It does not support concurrency as the number of nodes being locked is being reduced.
	Numlock[12]	In order to serve any MGL request it selects a locking combination which is optimal by using a greedy algorithm. It lies on the concept of interval locking to generate a subset for the given locking option. That generate a subset of the pare to-optimal options	Concurrency does not increases as it is completely dependent on efficient thread synchronization.
	Hi-fi lock[30]	In order to quickly checks the overlap region a novel indexing technique is being used. It quickly checks the overlap between two threads requested in a hierarchy.	It is best suited when the number of transactions are less.
	Fine grained Locking[32]	It considered objects as lowest level of granularity by creating an object graph which is used to create the parallel procedures.	The memory overhead of the locks is being increased.
	Automatic Lock Placement Policy[19,20]	A system that is being used for carrying out the lock placements is being used. A mapping function is generated between the lock and the data item that guards multiple nodes.	Increase in Concurrency is not taken into account.
	Concurrent multi way tree algorithm[21]	In order to represent concurrent left-child right-sibling tree Packed Objects are being used to represent the ordering relationship.	It does not reduces the overhead of recursive tree traversal
	Multiversion-Gist[22]	It proposed a concurrent index structure based on MVCC and the Gist which provides long-lasting read sessions.	It has increased memory consumption
	Multi-Version read-log-update (MV-RLU) [23]	In the case of any update conflict, the MV-RLU avoids the waiting condition for an object reclamation by providing them the existing versions of a given object.	Increased Memory consumption
	Starvation Freedom in Multi-Version OSTM (SF-MVOSTM)[24]	In order to perform starvation-freedom the latest K-versions is being maintained corresponding to each key.	Does not handle deadlock condition
	Time stamp based multi version[25]	It ensures read-only transaction does not aborts by using timestamp to decide which version should be given to the transaction to perform its read operation.	Inefficient garbage collection

ISSN (Print): 2204-0595
ISSN (Online): 2203-1731

Multi -Version Locking	Time-Warp Multi-version algorithm (TWM)[26]	It is based on the principle of time-wrap commit by allowing any update transaction to perform stale reads.	Look into account timestamp counters for each transaction operations thereby increasing the computational overhead
	Novel MVCC implementation[27]	It verifies that the (extensional) writes of any transaction that has committed recently do not overlap with the intentional read predicate space that belongs to some other transaction.	Maintenance process becomes complex.
	Cicada [28]	In order to support fast serializable concurrency control it relies on the concept of optimistic form of multi-version.	Garbage collection technique is inefficient
	Multi-Version Concurrency Control with Closures (MV3C)[29]	It handles the conflict among simultaneously running multiple transactions by partially aborting and restarting them instead of aborting.	Restarting the transaction increases computational overhead

III. Design of CGVL Locking Protocol

CGVL supports hierarchical locking mechanism that provides high robust performance under read and write intensive workloads. As depicted in figure 1 it adopts different phases: read, write and maintenance phase.

Read Phase

In the read phase each transaction entering into the system is being assigned a unique timestamp. In order to ensure that transaction T is serializable it must satisfy the following properties [30]:

- Read Stability: During the execution of the transaction if a transaction T reads a particular version V1 of a record. We must ensure that the value of V1 will not be changed even at the end of the transaction. That is by no means V1 will not be replaced by any other version. This can be ensured by providing the suitable version to be read by the requesting transaction and avoiding any update on the particular version.
- Phantom avoidance: It ensures that for each transaction scan an additional new version is not created.

In our work we have created a specific function to process the read request as shown below:

```
    If (writeInProgress.contains
(mulitGranularityVersionTableRow)) {
    Return
mulitGranularityVersionTableRow.getPreviousD
ataVal () ;}
    ThreadContextKeeper.getContext
().getWaitTimeList ()
    .add (System.currentTimeMillis () -
ThreadContextKeeper.getContext
().getWaitStartTime ());
    Return
mulitGranularityVersionTableRow.getData ();
```

Write Phase

In this phase each write request consider two transactional request. The current transaction record which points to record that represent the version with which the transaction started and last commit transaction record which specify the record created when the transaction commits. To perform valid write transactional request any transaction Ta checks the following conditions: It ensures that there exist no other transaction Tm which started after Ta that tries to update the same data element then in that case Ta aborts. The pseudo code given below shows the working of the write phase.

```
ThreadContextKeeper.getContext
().getWaitStartTime (System.currentTimeMillis
());
Synchronized
(mulitGranularityVersionTableRow.getData ()) {
Logger. Debug ("Locked by {} by thread {}",
mulitGranularityVersionTableRow.getRouNum
(), Thread.currentThread ().get Name ());
writeInProgress.add
(mulitGranularityVersionTableRow);
 mulitGranularityVersionTableRow.getData
(mulitGranularityVersionTableRow.getData ()
mulitGranularityVersionTableRow.getPreviousD
ataVal
(mulitGranularityVersionTableRow.getData ());
writeInProgress.remove
(mulitGranularityVersionTableRow);
Logger. Debug ("Released by {} by thread {}",
mulitGranularityVersionTableRow.getRouNum
(), Thread.currentThread ().get Name ());
ThreadContextKeeper.getContext().getWaitTime
List ().add (System.currentTimeMillis () -
ThreadContextKeeper.getContext
().getWaitStartTime ());
```

Maintenance Phase

In this case we validate the record for the last commit record onwards. Each last commit record is associated with an increment version number which is being committed. Basically validation of a transaction T comprises of three main steps:

- It checks if T's read set is updated with the current global state
- For each write request it looks for an intersection with T's write set.
- An obsolete version is being removed by the process known as garbage collection.

ISSN (Print): 2204-0595
ISSN (Online): 2203-1731

The pseudo code for the maintenance phase is given below:-

```
Synchronized
(mulitGranularityVersionTableRow.getData ()) {
Logger. Debug ("Locked by {} by thread {}",
mulitGranularityVersionTableRow.getRouNum
(), Thread.currentThread ().get Name ());
writeInProgress.add
(mulitGranularityVersionTableRow);
```

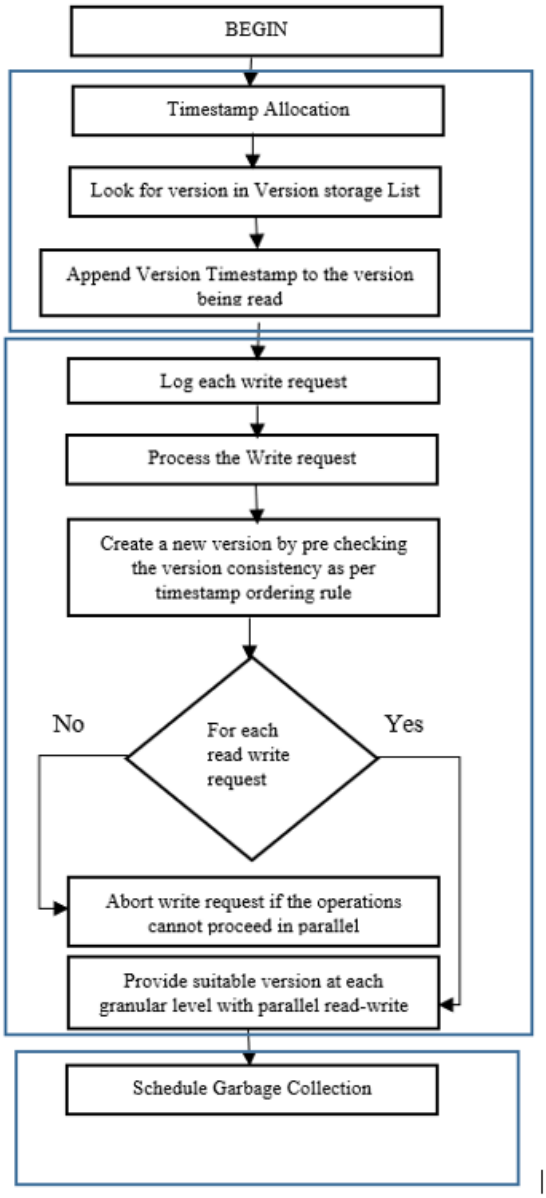

Fig 1: Phases of CGVL Locking Protocol

IV. Evaluation

While In this section the performance of CGVL has being evaluated. As platform we have used 4 GB of RAM, comprising of a 64 bit windows XP installation. For our implementation we have used Sun JDK 1.5.0. In this environment the different java threads are being mapped to the kernel threads that are being scheduled by the operating system. In order to ensure consistent system conditions the background processes are disabled as much as possible so as to maintain consistent system conditions.

Implementation using Java: The thread executes in a looping construct which performs locking and unlocking operation on the data item. On every run we configure the following parameters: i) The number of threads ii) The number of locked object iii) The duration of critical section (CPU Cycles).For each looping iteration the thread selects the locked object. Our result uses the average value obtained after 10 repetitions.

We compare CGVL against state-of-the-art Intention Locking [7] protocol. Basically our test driver creates multiple threads under different scenarios which operate concurrently on the underlying data structure. We further capture the advantage of Multi version locking and blend in our proposed work. In order to evaluate the performance of Multi version locking we implement it under varying workload read/write ratio and number of concurrent transactions.

Results :Implementing Multi version and Single version

We first discussed the results of implementing Multi version locking and compare its performance with single version locking by considering the following parameters.

a) **Effects of varying workload read/write ratio:** A key property of update transactions is the ratio between reads and writes. We explore the spectrum from a read-intensive workload having read/write ratio of 10:0 to a write-intensive workload having read/write ratio of 0:10.We designed a scenario which comprises of 10 transactions that perform their transactional operations depending on their number of read-write request. The detailed scenario covered is being shown in table 2. Interestingly, in table 3 we observe that for the read only transaction, the total execution time is same for both multi version and single version.

However, the writes into the mix, where the lock avoidance logic fails, and acquiring read locks with the exclusive locks (resulting in an extended lock wait) that are held for the entire duration of the transaction in the case of single version database.

ISSN (Print): 2204-0595
ISSN (Online): 2203-1731

While in case of multi-version database the clashes between readers and writers are eliminated by acquiring only update locks during transaction life-time, and holding exclusive locks for a much shorter period primarily during the commit time. As shown in the figure 2 and 3 the execution time for the single version database keeps on increasing as we execute more of update transactions as compared to multi-version database. So, we conclude that the performance of multi version is better than single version database primarily because the readers and writers don't conflict as when an updater writes a new version of a record, a read could continue reading the currently committed version of the record without any blockage.

Table 2: Detailed Scenario for Single and Multi-version Locking (W: Write, R: Read)

Case No.	**Mode of operation**	**Transaction 1**	**Transaction 2**	**Transaction 3**	**Transaction 4**	**Transaction 5**	**Transaction 6**	**Transaction 7**	**Transaction 8**	**Transaction 9**	**Transaction 10**
1	All write operations	W	W	W	W	W	W	W	W	W	W
2	1 Read,9 write operations	W	W	W	W	R	W	W	W	W	W
3	2 Read,8 Write operations	R	W	W	W	R	W	W	W	W	W
4	3 Read,7 Write operations	R	W	W	R	W	W	W	R	W	W
5	4 Read,6 Write operations	R	W	R	W	W	W	R	W	R	W
6	5 Read,5 Write operations	R	W	R	W	R	W	R	W	R	W
7	6 Read,4 Write operations	R	W	R	W	R	R	W	R	W	R
8	7 Read,3 Write operations	R	W	R	W	R	R	W	R	R	R
9	8 Read,2 Write operations	R	W	R	R	R	R	R	W	R	R
10	9 Read,1 Write operations	R	R	R	R	R	W	R	R	R	R
11	All Read operations	R	R	R	R	R	R	R	R	R	R

ISSN (Print): 2204-0595
ISSN (Online): 2203-1731

Table 3: Execution time taken by Single and Multi-Version

Case No.	Cases	Single Version(msec)	Multi Version(msec)
1	0_10(0 Read 10 Updates)	118.50	115.20
2	1_9(1 Read 9 update)	103.51	95.20
3	2_8(2 Read 8 Update)	92.82	82.14
4	3_7(3 Read 7 Update)	83.98	73.34
5	4_6(4 Read 6 Update)	77.12	65.19
6	5_5(5 Read 5 Update)	66.91	54.60
7	6_4(6 Read 4 Update)	57.30	44.34
8	7_3(7 Read 3 Update)	49.68	37.79
9	8_2(8 Read 2 Update)	41.82	31.19
10	9_1(9 Read 1 Update)	36.64	27.65
11	10_0(10 Read 0 Update)	15.24	13.44

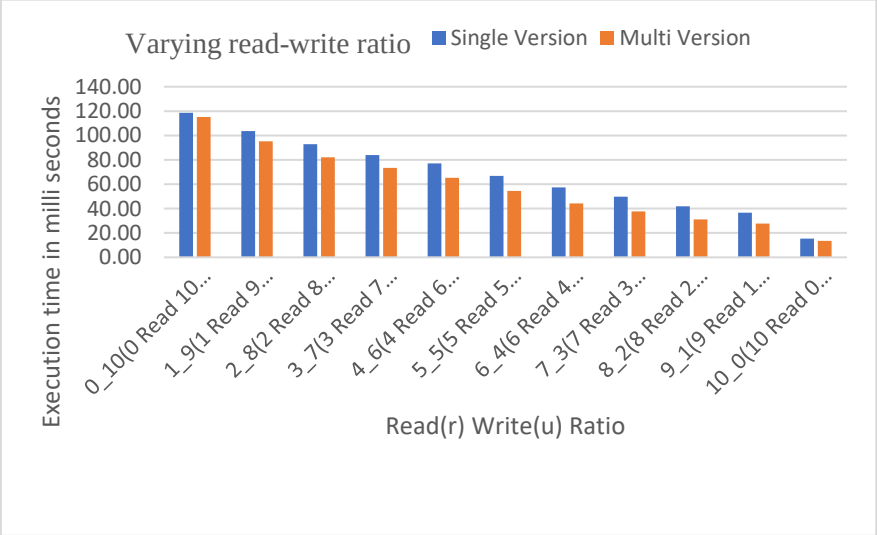

Fig 2: Varying read-write ratio

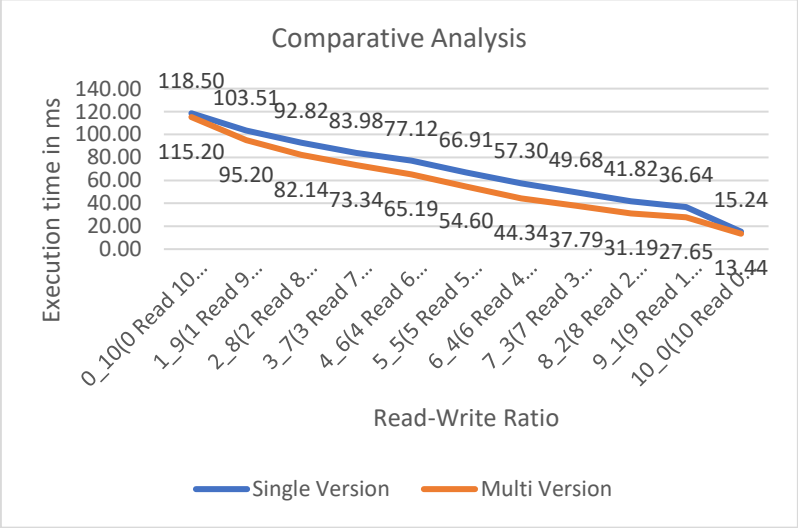

Fig 3: Comparative Analysis

b) Effect of varying contention: We control the degree of contention by varying the transactions size in case of both single version and multi version database. As we increases the number of transactions from 10 to 30, we observed that existing multi version outperforms the single version database

ISSN (Print): 2204-0595
ISSN (Online): 2203-1731

primarily because with the increase in number of transactions as shown in figure 4, 5, 6 the contention is increased significantly between the read locks of long running read-only transactions and the write locks (exclusive locks) of update transactions in a single version database. This increased contention is less significant in case of multi-version as a result their execution time is significantly less for mix of read-write tranasctions.Thus, we performed a comparative analysis as being shown in figure 7, 8, 9 that validate our work by showing the effectiveness of multi version over single version locking under different workloads.

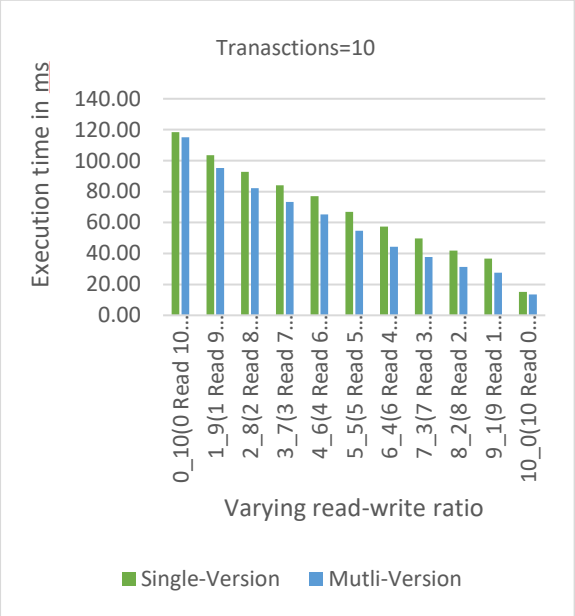

Fig 4: Varying Workload for 10 transactions

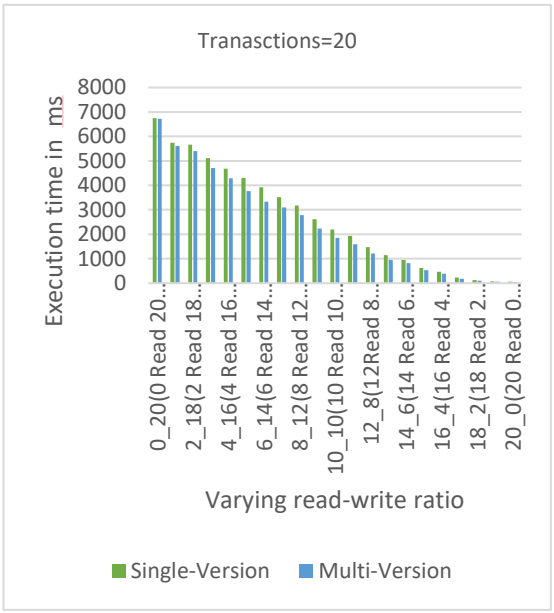

Fig 5: Varying Workload for 20 transactions

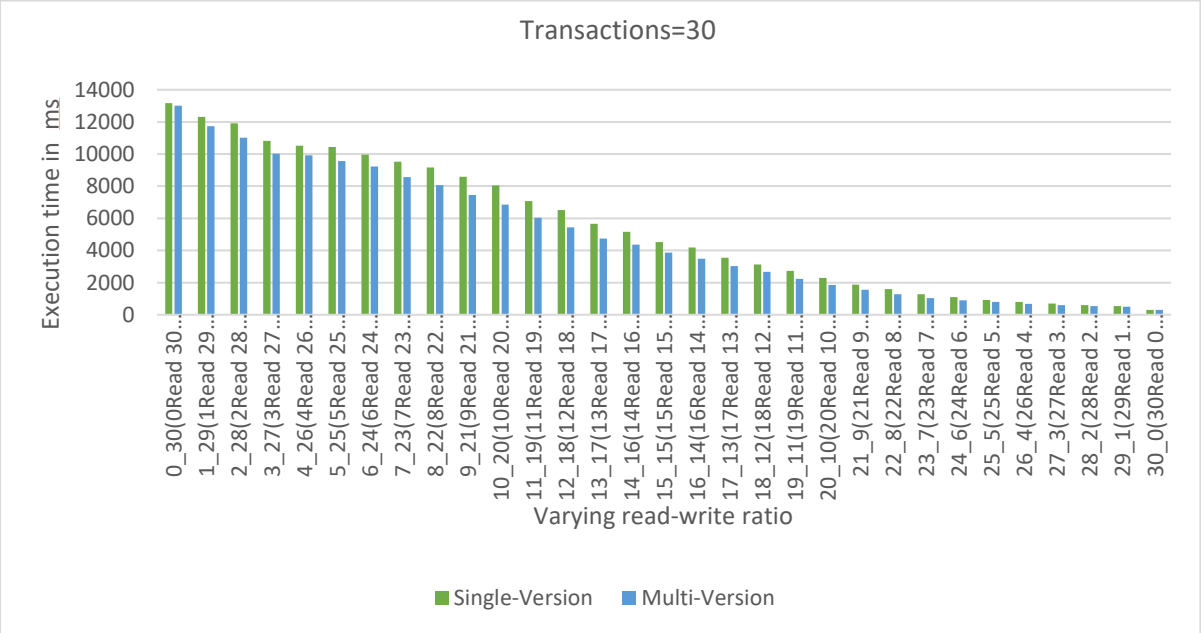

Figure 6: Varying Workload for 30 transactions

ISSN (Print): 2204-0595
ISSN (Online): 2203-1731

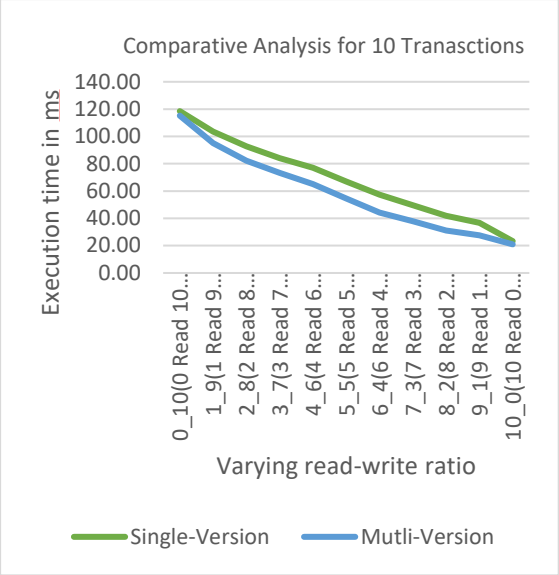

Fig 7: Comparative Analysis for 10 transactions

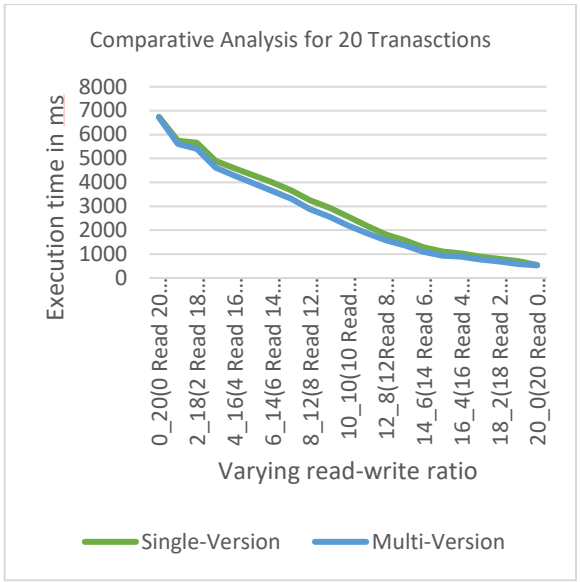

Fig 8: Comparative Analysis for 20 transactions

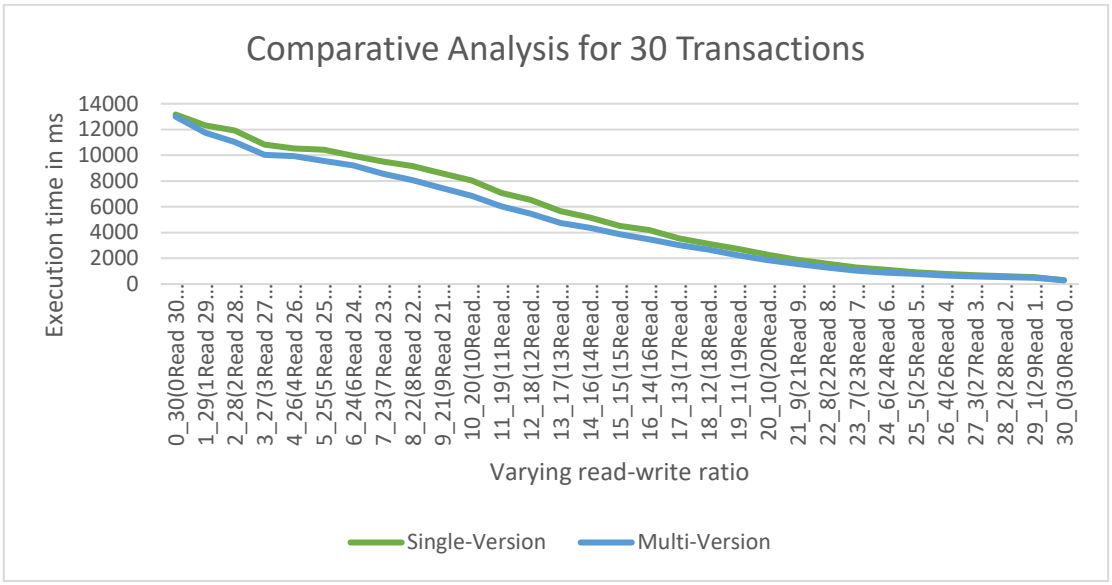

Fig 9: Comparative Analysis for 30 transactions

Results: Implementing CGVL and MGL

Next we implement CGVL and existing MGL by considering a scenario which comprises of 10 transactions with 15 different cases where each case represent the different locking modes. The system conditions and resources are kept same for both the protocols. We then implement them to compare their performance by considering the following parameters.

a) **Effect of varying contention:** We study the effect of varying contention by changing the read-write ratio and then analysing performance of the system by considering its total execution time in each cases. The results are derived by executing 10 transactions on 15 different cases as shown in table 4. The two protocols are executed in JDK environment where CGVL support certain operations that are executed concurrently which MGL doesn't permit. This thereby lead to the improvement of existing compatibility matrix of MGL thereby converting the non-supporting locking modes into supporting locking modes as shown in table 5.

ISSN (Print): 2204-0595
ISSN (Online): 2203-1731

Table 4: Detailed Scenario for CGVL and Multiple Granularity Locking (W: Write, R: Read)

<table>
<tr><th>Case No.</th><th>Lock Modes</th><th>Supporting Status</th><th>Thread 1</th><th>Thread 2</th><th>Thread 3</th><th>Thread 4</th><th>Thread 5</th><th>Thread 6</th><th>Thread 7</th><th>Thread 8</th><th>Thread 9</th><th>Thread 10</th></tr>
<tr><td>1</td><td>IS-IS</td><td>Yes</td><td>R|1</td><td>R |2</td><td>R|3</td><td>R|4</td><td>R|5</td><td>R|6</td><td>R |7</td><td>R|8</td><td>R|9</td><td>R|10</td></tr>
<tr><td>2</td><td>IS-IX</td><td>Yes</td><td>R|1</td><td>W|2</td><td>R|3</td><td>W|4</td><td>R|5</td><td>W|6</td><td>R|7</td><td>W|8</td><td>R|9</td><td>W|10</td></tr>
<tr><td>3</td><td>IS-S</td><td>Yes</td><td>R|1,2,3,4,5</td><td>R|1</td><td>R|5,6,7,8,9,10,11,12</td><td>R|2</td><td>R|3</td><td>R|4</td><td>R|1,2,3,4,5</td><td>R|4</td><td>R|5</td><td>R|1,2,3,4,5,6</td></tr>
<tr><td>4</td><td>IS-SIX</td><td>Yes</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|1</td><td>R|1</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|2</td><td>R|2</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|3</td><td>R|3</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|4</td><td>R|4</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|5</td><td>R|5</td></tr>
<tr><td>5</td><td>IS-X</td><td>No</td><td>R|1</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td><td>R|2</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td><td>R|3</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td><td>R|4</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td><td>R|5</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td></tr>
<tr><td>6</td><td>IX-IX</td><td>Yes</td><td>W|1</td><td>W|2</td><td>W|3</td><td>W|4</td><td>W|5</td><td>W|6</td><td>W|7</td><td>W|8</td><td>W|9</td><td>W|10</td></tr>
<tr><td>7</td><td>IX-S</td><td>No</td><td>W|1</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12</td><td>W|2</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12</td><td>W|3</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12</td><td>W|4</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12</td><td>W|5</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12</td></tr>
<tr><td>8</td><td>IX-SIX</td><td>No</td><td>W|1</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|1</td><td>W|2</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|2</td><td>W|3</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|3</td><td>W|4</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|4</td><td>W|5</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|5</td></tr>
<tr><td>9</td><td>IX-X</td><td>No</td><td>W|1</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td><td>W|2</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td><td>W|3W</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td><td>W|4</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td><td>W|5</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td></tr>
<tr><td>10</td><td>S-S</td><td>Yes</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12</td></tr>
<tr><td>11</td><td>S-SIX</td><td>No</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|1</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|2</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|3</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|4</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|5</td></tr>
<tr><td>12</td><td>S-X</td><td>No</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12</td></tr>
<tr><td>13</td><td>SIX-SIX</td><td>No</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|1</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|1</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|1</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|1</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|1</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|1</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|1</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|1</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|1</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|1</td></tr>
<tr><td>14</td><td>SIX-X</td><td>No</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|1</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|1</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|1</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|1</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td><td>R|1,2,3,4,5,6,7,8,9,10,11,12-W|1</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td></tr>
<tr><td>15</td><td>X-X</td><td>No</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td><td>W|1,2,3,4,5,6,7,8,9,10,11,12</td></tr>
</table>

ISSN (Print): 2204-0595
ISSN (Online): 2203-1731

Table 5: Improved Locking Modes in CGVL

Locking Modes(Ti)	Locking Modes (Tj)	Existing MGL approach	Proposed CGVL approach
IS	X	No	Yes
IX	S	No	Yes
S	(IX,SIX,X)	No	Yes
SIX	S	No	Yes
X	(IS,S)	No	Yes

The total execution time taken in each case has being calculated as shown in table 6.The figure 10 shows the improvement of CGVL over existing MGL.

Table 6: Execution time for MGL and CGVL

Case No.	Lock Modes	MGL (msec)	CGVL(msec)
1	IS-IS	152	145
2	IS-IX	199	187
3	IS-S	130	124
4	IS-SIX	361	345
5	IS-X	2178	1800
6	IX-IX	370	350
7	IX-S	551	459
8	IX-SIX	560	545
9	IX-X	2550	2534
10	S-S	320	302
11	S-SIX	445	380
12	S-X	2600	2100
13	SIX-SIX	2211	2200
14	SIX-X	4111	3997
15	X-X	4321	4254

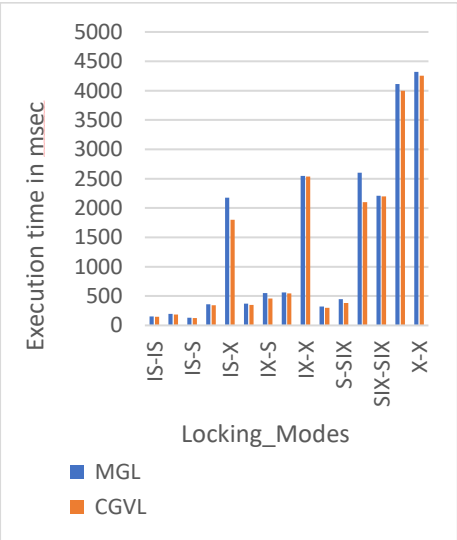

Fig 10: Effect of varying contention

b) Stress Test Implementation: .In order to test the scalability of CGVL against intention locks (IL) we next came up with the designing of a synthetic test bench that execute the two protocols by varying the number of transactions and comparing their performances. The same amount work is being assigned to each thread and we measure the overall execution time taken for the threads. The number of threads in our implementation varies from 10 to 50. We have fragmented our results into two cases comprising of 1) Improved Locking Modes 2) Non-Improved Locking Modes.

1) Improved Locking Modes:

We now present the effect of changing the number of transactions on the system in case of improved locking modes. It is clearly observed that there is an enhanced performance of the applications as we vary the underlying locking mechanisms for the operations of similar type. Conventional multiple granularity does not support higher degree of concurrency as compared to CGVL in case of (IS-X, IX-S, S-SIX, and S-X) under different set of transactions being executed (10-50). Basically, as the number of threads increases, the overhead incurred gets compensated by the improved degree of concurrency (on an average 20% performance improvement over MGL) has being obtained.

Figure 11-15 depicts the execution time which varies with respect to the different distributions. In order to control the distribution, the threads are restricted to access only a particular set of data items. The X-axis shows the improved locking modes while Y-axis represent the execution time taken by the transactions. We conclude that CGVL is one of the most promising locking protocol in hierarchical structure.

As it exhibits enhanced performance compared to the existing multiple granularity locking protocol our work and we conclude that which does not support the locking operations of concurrent transactions in this particular locking modes. Figure 16-17 shows the percentage of improvement attained in each case which validate our work and we conclude that proposed CGVL is better than the existing MGL locking protocol.

ISSN (Print): 2204-0595
ISSN (Online): 2203-1731

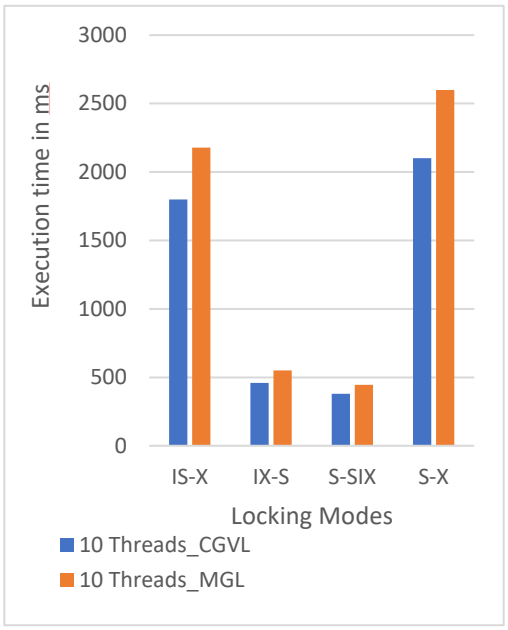

Fig 11: Improved Locking Modes for 10Transactions

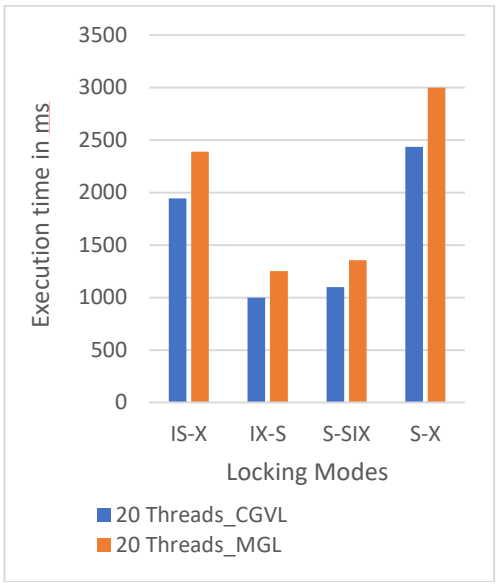

Fig 12: Improved Locking Modes for 20Transactions

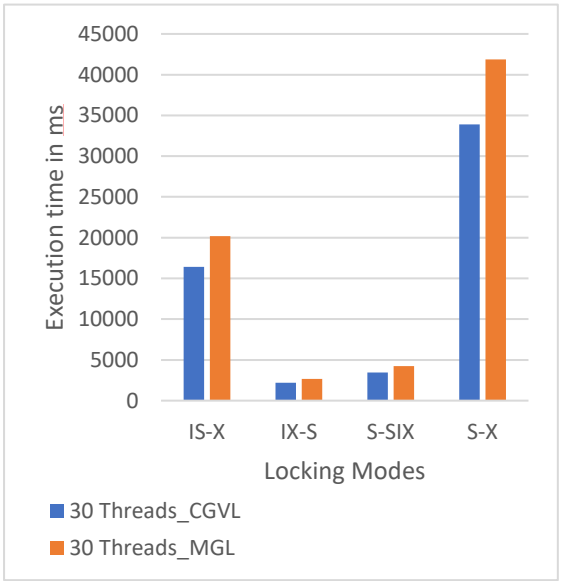

Fig 13: Improved Locking Modes for 30 transactions

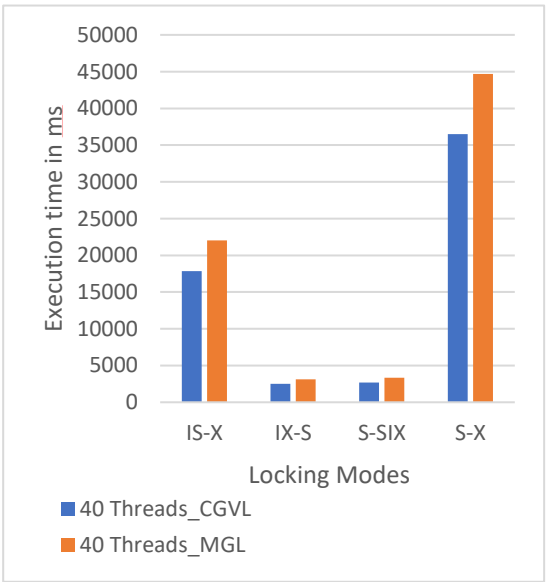

Fig 14: Improved Locking Modes for 40 transactions

ISSN (Print): 2204-0595
ISSN (Online): 2203-1731

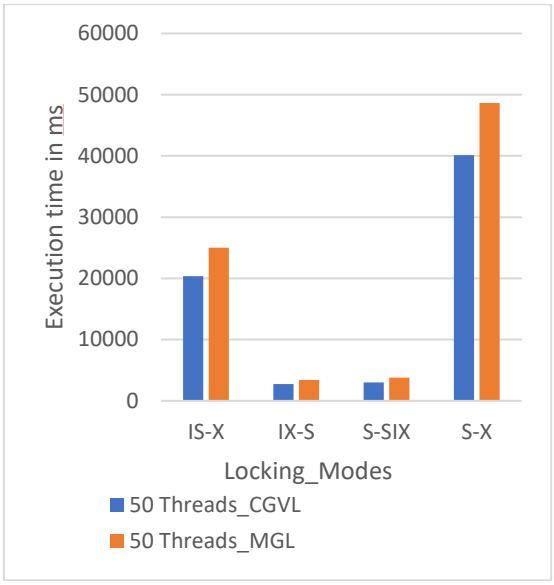

Fig 15: Improved Locking Modes for 50 transactions

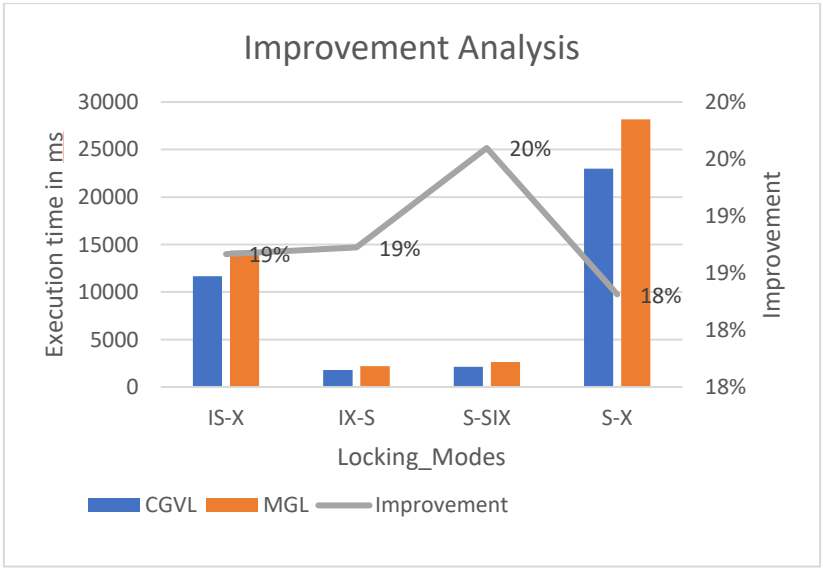

Fig 16:Improvement Analysis

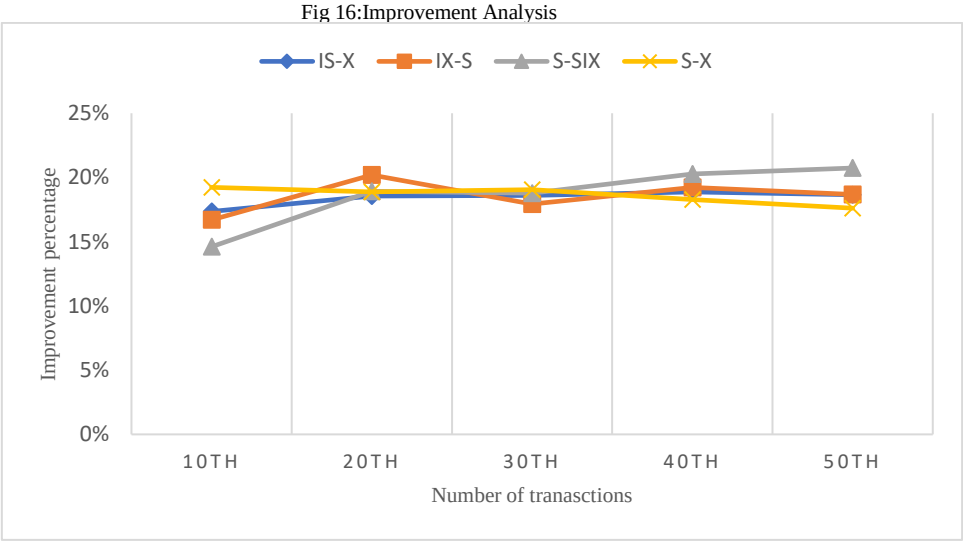

Fig 17: Trend line graph for Improved Locking Modes

ISSN (Print): 2204-0595
ISSN (Online): 2203-1731

2) Non-Improved locking modes:

In this section we took up the (S-S,X-X,IS-IS,IS-S,IX-X) locking modes and study their effect on the two locking environment as shown in figure 18-22.We noticed that they takes almost same time to execute for the transactional operations that are being executed in X-X and IX-IX locking modes. However, for the other locking modes (IS-IX, IS-SIX, IX-SIX, SIX-SIX, SIX-X) there is slight considerable improvement on an average of 5% is being noticed. The figure 24 provides the analysis of different locking modes in CGVL environment that provides a conclusion that our proposed approach is better than the MGL locking protocol.

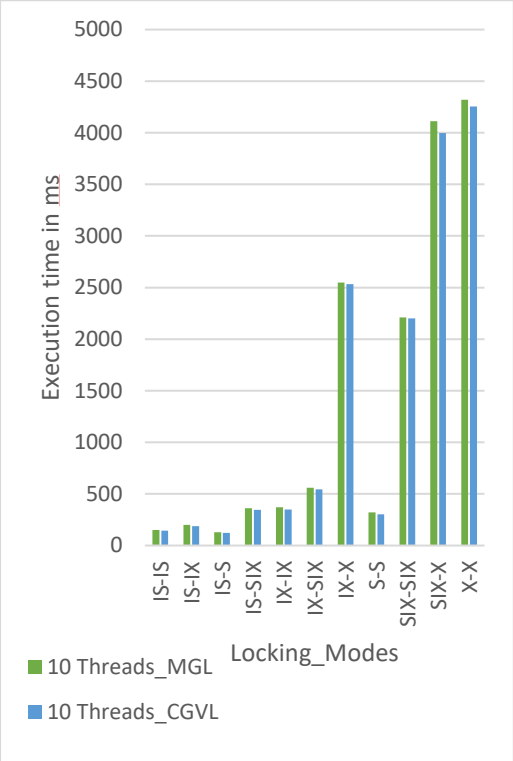

Fig. 18: Non-moved Locking Modes for 10 transactions

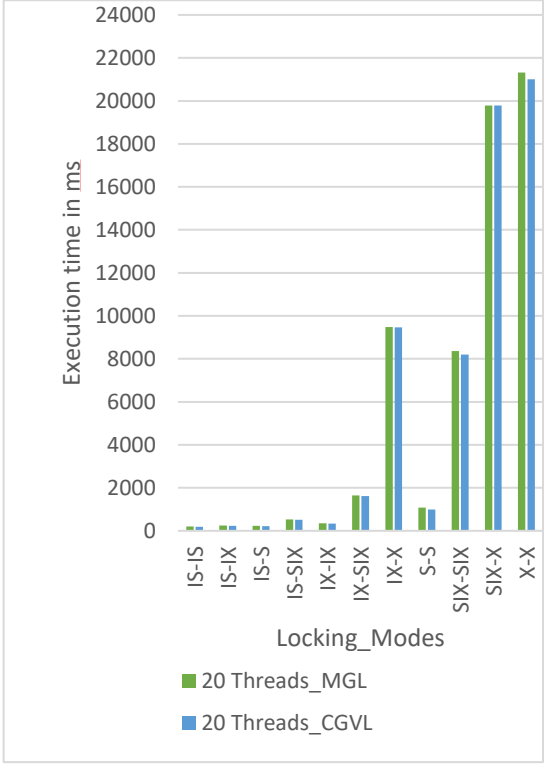

Fig 19: Non-moved Locking Modes for 20 transactions

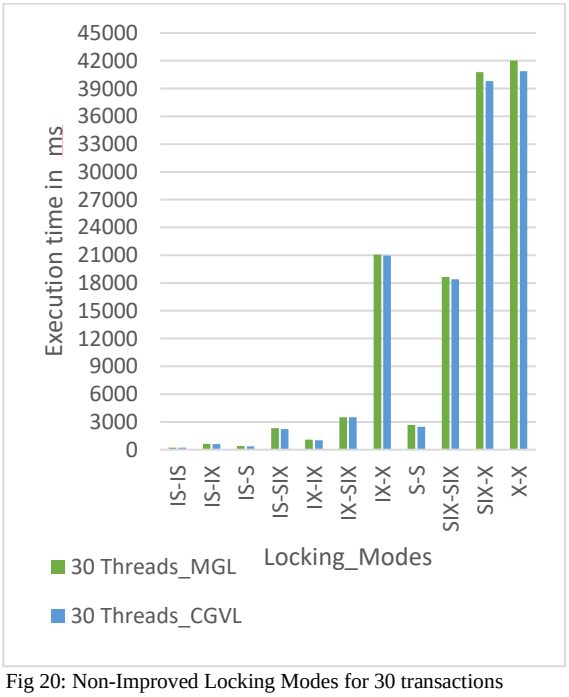

Fig 20: Non-Improved Locking Modes for 30 transactions

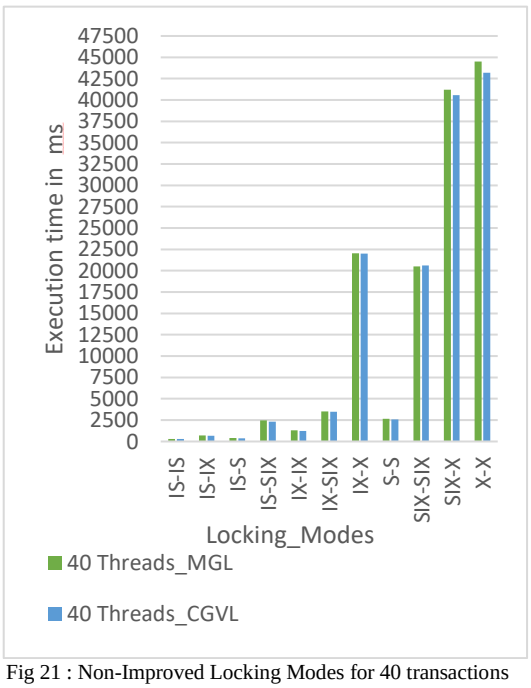

Fig 21 : Non-Improved Locking Modes for 40 transactions

ISSN (Print): 2204-0595
ISSN (Online): 2203-1731

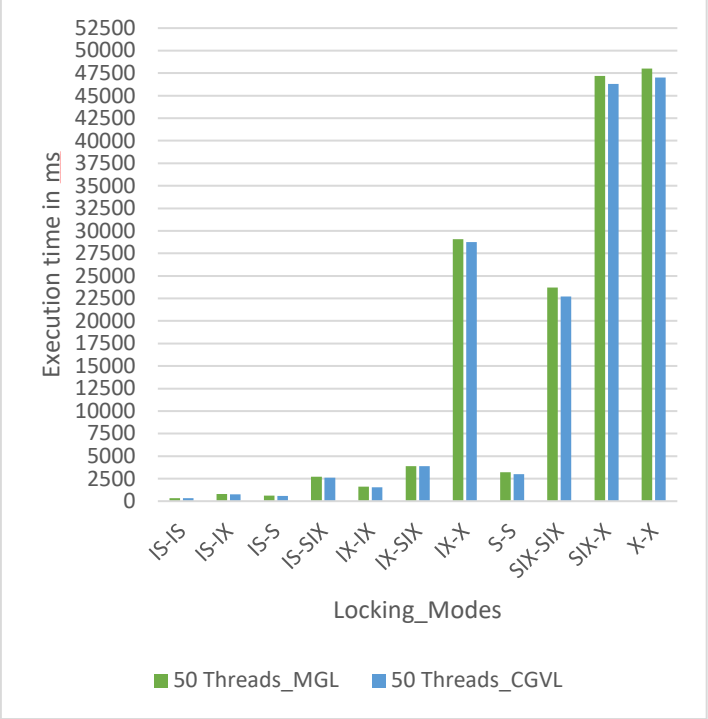

Fig 22: Non-Improved Locking Modes for 50 transactions

Conclusion

We proposed an effective locking scheme namely Collaborative Granular Version Locking (CGVL) which works on the principle by combining the query capabilities and flexibility features from the Multiple Granularity locking (MGL) that is being successfully integrated with the high concurrency offered by Multi Version. The concurrency of the system has been improved by maintaining a series of version at each granular level so as to allow the concurrent execution of read and write operation on the data item. In this work the model predicts the performance of the system which is based on several parameters such i) The number of threads ii) The number of locked object iii) The duration of critical section (CPU Cycles) which significantly supports the achievement of enhanced concurrency. We have been able to validate our work by using a Java Sun JDK environment, which shows that our CGVL perform better compared to the existing MGL methods. In particular, CGVL attains 20% reduction in execution time for the locking operation as it is able to convert some of the non-supporting locking modes into supporting locking modes thereby improved the compatibility matrix.

References

[1] HyeontaekLim, DongsuHan, DavidG.Andersen, and Michael Kaminski" MICA: A Holistic Approach to Fasting-memory Key-value Storage". In Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI).Seattle, WA, 429–444, 2014.

[2] Silas Boyd Wickizer, AustinT.Clements, Yandong Mao, Aleksey Pesterer, M.Frans Kaashoek, RobertMorris, and Nickola Zeldovich "An Analysis of Linux Scalability to Many Cores. In Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI)."Vancouver, Canada,16, 2010

[3] AustinT.Clements, M.FransKaashoek, and Nickolai Zeldovich "Scalable Address Spaces Using RCU Balanced Trees". In Proceedings of the 17th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). London, UK, 2012

[4] AustinT.Clements,M.FransKaashoek, and NickolaiZeldovich." Radix: Scalable Address Spaces for Multithreaded Applications". In Proceedings of the 8th European Conference on Computer Systems (Euros).Prague, CzechRepublic, 2013.

[5] Morris, and Eddie Kohler "The Scalable Commutatively Rule: Designing Scalable Software for Multicore Processors". In Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP). Farmington, PA, 2013.

ISSN (Print): 2204-0595
ISSN (Online): 2203-1731

[6] Sanguine Han, Scott Marshall,Byung-Gon Chunand Sylvia Ratna"MegaPipe:ANewProgrammingInterfacef orScalableNetwork I/O".In Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI).USENIXAssociation,Hollywood, CA,135–148,2012.

[7] Aleksey Pesterev, Jacob Strauss, Nickolai Zeldovich, and RobertT. Morris ImprovingNetworkConnectionLocalityonMultic ore Systems.InProceedings of the 7th European Conference on Computer Systems (Euros).ACM, Bern, Switzerland, 337–350, 2012

[8] Stephen To, Wenting Zheng, Eddie Kohler, Barbara Liskov, and SamuelMadden.SpeedyTransactionsinMulticore In-memory Databases. In Proceedings of the 24th ACM Symposium of Operating Systems Principles (SOSP).ACM, Farmington, PA, 18–32, 2013

[9] Jaeho Kim, Ajit Mathew, Sanidhya Kashyap,Madhava Krishnan Ramanathan, Changwoo Min" MV RLU:ScalingRead-Log-Updatewith Multi-Versioning" published in the proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, April 13 - 17, 2019.

[10] Constantine's Papadopoulos" A Multiple Granularity Locking Protocol FOR CSCW" International Journal of Cooperative Information Systems Vol. 11, Nos. 1 & 2 (2002) 21–50, 2012

[11] SaurabhKalikarandRupeshNasre.2016. Dom Lock: A New Multi-granularity Locking Technique for Hierarchies. In Pop 2016. ACM, Article 23, 23:1– 23:12pages. https://doi.org/10.1145/2851141.2851164

[12] SaurabhKalikarandRupeshNasre.2017" NumLock: TowardsOptimalMulti-GranularityLockingin Hierarchies" .Inapt 2018: 47th International Conference on Parallel Processing, August 13–16, 2018, Eugene, OR, USA.ACM, and New York,NY,USA,10pages.https://doi.org/10.1145/3225058.3225141.

[13] Yingjun Wu, Joy Arulraj, Jiexi Lin, Ran Xian ,Andrew Pavlo "An Empirical Evaluation of In-Memory Multi-Version Concurrency Control "published in the Proceedings of the VLDB Endowment, Vol. 10, No. 7,2017.

[14] Lisping George, Anastasios Andronidis, CristianCadar "FreeDA: Deploying In compatible Stock Dynamic Analyses in Production via Multi-Version Execution" published in the proceedings of ACM May 8–10, 2018, Ischia, Italy.

[15] MarcosK.Aguilera,Tudor David, Rachid Guerraoui, Junxiong Wang "Locking timestamps versus locking objects" published in the proceedings of the ACM Symposium on Principles of Distributed Computing, Pages 367-376, 2018.

[16] Chirag Juyal, Sandeep Kulkarni, Sweta Kumari, Sathya Peri, and Archit Somani "An Innovative Approach to Achieve Compositionality Efficiently using Multi-Version Object Based Transactional Systems" Published in Distributed, Parallel, and Cluster Computing Journal,2018.

[17] Jaeho Kim, Ajit Mathew, Sanidhya Kashyap,Madhava Krishnan Ramanathan, Changwoo Min" MV-RLU:ScalingRead-Log-Updatewith Multi-Versioning" published in the proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, April 13 - 17, 2019.

[18] Chirag Juyal, Sandeep Kulkarni, Sweta Kumari, Sathya Peri, and Archit Somani "Achieving Starvation- Freedom with Greater Concurrency in Multi-Version Object-based Transactional Memory Systems" Published in Distributed, Parallel, and Cluster Computing Journal,2019.

[19] Guy Golan-Gueta,Nathan Bronson,Alex Aiken,G.Ramalingam,Mooly Sagiv, and EranYahav.2011. AutomaticFine-grainLocking Using Shape Properties.In OOPSLA 2011.ACM, 225–242. https://doi.org/10.1145/2048066.2048086

[20] Peng Liuand Charles Zhang "UnleashingConcurrencyforIrregularData Structures"In ICSE 2014 .ACM, 480–490. https://doi.org/10.1145/2568225.2568277,2014

[21] Yang, KentKB, AubanelE, MacKayS, AgilaT.Amulti-granularitylocking scheme for java packed objects based on a concurrent multiwaytree.ConcurrencyComputatPractExper. 2018;e5024.https://doi.org/10.1002/cpe.5024.

[22] Walter Binder,Adina Mosincat,Samuel Spycher, Ion Constantinescu and Boi Faltings" Multiversion concurrency control for the generalized search tree "Published in Concurrency and computation: practice and experience Concurrency Computat.:Pract.Exper.2009;21:1547–1571 Publishedonline13February2009inWileyInterScience

[23] Jaeho Kim, Ajit Mathew, Sanidhya Kashyap,Madhava Krishnan Ramanathan, Changwoo Min" MV-RLU:ScalingRead-Log-Updatewith Multi-Versioning" published in the proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, April 13 - 17, 2019.

[24] Chirag Juyal, Sandeep Kulkarni, Sweta Kumari, Sathya Peri, and Archit Somani "An Innovative Approach to Achieve Compositionality

ISSN (Print): 2204-0595
ISSN (Online): 2203-1731

Efficiently using Multi-Version Object Based Transactional Systems" Published in Distributed, Parallel, and Cluster Computing Journal,2018.

[25] Priyanka Kumar, Sathya Peri, K.Vidyasankar **Fig** "A Timestamp Based Multi-Version STM Algorithm" Published in the International Conference on Distributed Conference on Distributed Computing and Networking, pp 212-226, 2014.

[26] Nuno Diegues Paolo Romano" Time-Warp: Lightweight Abort Minimization in Transactional Memory" published in the 19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP),, At Orlando, Florida, USA,2014.

[27] Thomas Neumann ,Tobias Mühlbauer and Alfons Kemper" Fast Serializable Multi-Version Concurrency Control for Main-Memory Database Systems" published in ACM SIGMOD'15, May 31–June 4, Melbourne, Victoria, Australia,2015.

[28] Hyeontaek Lim Carnegie Mellon, Michael Kaminsky, and David G. Andersen "Cicada: Dependably Fast Multi-Core In-Memory Transactions "published in ACM SIGMOD '17, May 14–19, Chicago, IL, USA, 2017.

[29] Mohammad Dashti, Sachin Basil John, Amir Shaikhha, and Christoph Koch "Transaction Repair for Multi-Version Concurrency Control "published in the proceedings of the ACM International Conference on Management of Data, Pages 235-250,USA,May 14-19,2017.

[30] Per-Åke Larson1, Spyros Blanas, Cristian Diaconu, Craig Freedman, Jignesh M. Patel, Mike Zwilling" High-Performance Concurrency Control Mechanisms for Main-Memory Databases"Published in the 38th International Conference on Very Large Data Bases, August 27th - 31st 2012, Istanbul, Turkey. Proceedings of the VLDB Endowment, Vol. 5, No. 4,2011

[30] K. Ganesh, Saurabh Kalikar(B), and Rupesh Nasre" Multi-granularity Locking in Hierarchies with Synergistic Hierarchical and Fine-Grained Locks" 24th International Conference on Parallel and Distributed Computing Turin, Italy, August 27–31, 2018 Proceedings, Springer International Publishing AG, part of Springer Nature, pp. 546–559,2018.

[31] Ms. Swati,Dr Shalini Bhaskar Bajaj" CGVL:An Hierarchical Locking Mechanism "published in International Journal of Control and Automation Vol. 12, No. 6, pp. 725-743,2019.

[32] Peng Liu and Charles Zhang. 2014. Unleashing concurrency for irregular data structures. In Proceedings of the 36th International Conference on Software Engineering (ICSE 2014). ACM, New York, NY, 480–490. DOI:http://dx.doi.org/10.1145/2568225.2568277.

ISSN (Print): 2204-0595
ISSN (Online): 2203-1731