An Efficient Data Replication Technique with Fault Tolerance Approach using BVAG with Checkpoint and Rollback-Recovery

Sharifah Hafizah Sy Ahmad Ubaidillah[1], A. Noraziah[2], Basem Alkazemi[3]
Faculty of Computing, University Malaysia Pahang, Pahang, Malaysia[1, 2]
IBM Center of Excellence, University Malaysia Pahang, 26300 Kuantan, Pahang, Malaysia[2]
College of Computer and Information System, Umm Al-Qura University, Saudi Arabia[3]

***Abstract*—Data replication has been one of the pathways for distributed database management as well as computational intelligence scheme as it continues to improve data access and reliability. The performance of data replication technique can be crucial when it involves failure interruption. In order to develop a more efficient data replication technique which can cope with failure, a fault tolerance approach needs to be applied in the data replication transaction. Fault tolerance approach is a core issue for a transaction management as it preserves an operation mode transaction prone to failure. In this study, a data replication technique known as Binary Vote Assignment on Grid (BVAG) has been combined with a fault tolerance approach named as Checkpoint and Rollback-Recovery (CR) to evaluate the effectiveness of applying fault tolerance approach in a data replication transaction. Binary Vote Assignment on Grid with Checkpoint and Rollback-Recovery Transaction Manager (BVAGCRTM) is used to run the BVAGCR proposed method. The performance of the proposed BVAGCR is compared to standard BVAG in terms of total execution time for a single data replication transaction. The experimental results reveal that BVAGCR improves the BVAG total execution time in failure environment of about 31.65% by using CR fault tolerance approach. Besides improving the total execution time of BVAG, BVAGCR also reduces the time taken to execute the most critical phase in BVAGCRTM which is Update (U) phase by 98.82%. Therefore, based on the benefits gained, BVAGCR is recommended as a new and efficient technique to obtain a reliable performance of data replication with failure condition in distributed databases.**

Keywords*—Data replication; computational intelligence; fault tolerance; binary vote assignment on grid; checkpoint and rollback-recovery

I. INTRODUCTION

Data replication is a useful technique for a Distributed Database System (DDS) as it can provide high availability and efficient access to required data and can be applied in a grid computation situation to improve the efficiency of the system [1, 2]. Besides, data replication technique can be one of the influential techniques that can expand the usefulness of computational intelligence structure. Data replication involves frequent, incremental copying of data from one database to another database in a continuous manner which can increases availability, provide low response times and allows fast local access of the system [3, 4]. Despite the goodness of data replication techniques in handling the distributed database, still, it has some weakness when dealing with failure cases.

Handling data replication in the failure cases is very crucial in order to preserve the effectiveness of the systems. The main challenges of data replication are that the replica has to be kept consistent when updates occur despite having any failure during the transaction's running [4]. The only way to solve these problems is by enabling fault tolerance. Fault tolerance approach is a crucial issue in distributed computing; it keeps the transaction in an operational condition in subject to failure. The most important point of it is to keep the transaction working even if any of its part goes off or faulty [5]. Fault tolerance is the dynamic approach that's used to keep the interrelated transaction together, put up with reliability and availability in DDS. Efficient fault tolerance approaches help in detecting of faults and if possible recovers from it [6].

Based on previous studies, the combination of any data replication technique with Checkpoint and Rollback-Recovery (CR) fault tolerance approach in a distributed database is infrequently analyzed irrespectively of its individual promising potential to lessen the total execution time in failure-prone situations [7]. As an example, research done by [7] explored the performance of transaction process using CR only, replication only and the combination of both techniques in linear workflow with the presence of failure. The result obtained reveals that the conditions in which each techniques lead to improved performance. Besides that, paper done by [8] concludes that the CR approach is essential for not only transaction process replication but also for security issues.

Despite good performances, there are only few researchers who had interest in exploring the effectiveness of combining data replication technique with CR fault tolerance approach. It is a common practice to utilize a Checkpoint and Rollback-Recovery (CR) to facilitate an adequate failure recovery for improving transaction reliability [9]. Mainly, the checkpoint is performed to save information linked with the completed portion of a transaction. When a transaction failure occurs, through rollback and information retrievals, the transaction can be resumed from the last successful checkpoint. Instead, without implementing the checkpoint technique, the transaction has to repeat the execution of the entire transaction from the very beginning [10]. Hence, the data replication

transaction might be time consuming without the CR approach if any failure happened.

Therefore, in this study, a data replication technique called as Binary Vote Assignment on Grid (BVAG) is combined with CR with the proposed of evaluating the efficiency of hybridizing data replication technique with fault tolerance approach for a better performance of a single data replication transaction in the presence of failure. The proposed method, BVAGCR is implemented in Binary Vote Assignment on Grid with Checkpoint and Rollback-Recovery Transaction Manager (BVAGCRTM).

The paper is arranged as follows. In the next section, Literature Review is detailing about BVAG data replication technique and CR fault tolerance approach. In Section 3, Methodology describes the procedure of BVAGCR technique which is employed via BVAGCRTM. The Result and Discussion section discussed the outcome obtained from standard BVAG and BVAGCR. Also presented in this section is the comparison of both techniques in terms of execution time while managing data replication transaction with the occurrence of failure. Finally, the conclusion of this research and suggestion for future research are provided in Conclusion.

II. Literature Review

A. Binary Vote Assignment on Grid (BVAG)

The concept of Binary Vote Assignment on Grid (BVAG) is replicating the data from the primary replica to the neighbours' replica which is located at the adjacent sites of the primary replica [11]. Full replication can result in a huge waste of storage space and consume a lot of bandwidth [12]. By using this technique, the execution time of the replication process in a distributed database can be reduced as it only replicates data at the specified sites [12]. The query expansion process involves augmenting initial user query with additional terms that are related to user requirements [13], while BVAG focus challenge to increase write query availability through replication. BVAG is striding a new track in replication that helps to maximize the write availability with little communication cost as a result of minimum number of quorum size needed. Furthermore, the replication is interconnected with transaction procedure [14].

In BVAG, all sites are logically organized in the form of two-dimensional grid structure. For example, if a BVAG consists of nine sites, it will logically organize in the form of 3 x 3 grid structure (Fig. 1) as shown. As can be seen in Fig. 1, site A is neighbours to site B and site D, if A is logically located adjacent to B and D. Hence, four sites on the corners of the grid have only two adjacent sites, other sites on the boundaries have only three neighbours and the site located in the middle of the grid formation has four neighbours [14].

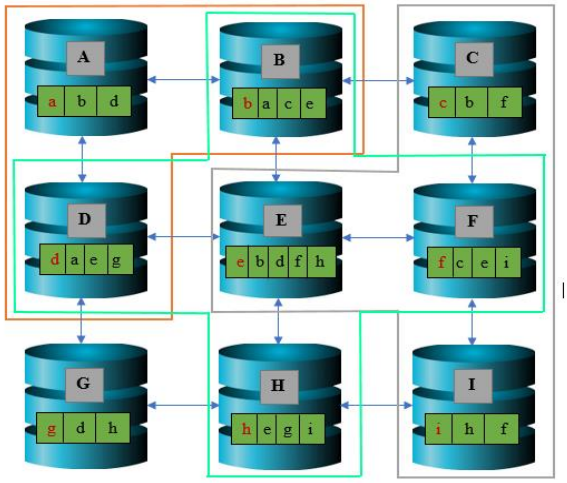

Fig. 1. Binary Vote Assignment on Grid (BVAG).

Each site has a premier data file. Data will be replicated to the neighbours sites from the primary site [11]. For simplicity, the primary site of any data file and its neighbours are assigned with vote one (1) or vote zero (0) otherwise. A neighbour binary vote assignment on grid, *B*, is a function such that $B(i) \epsilon \{0,1\}, 1 \leq i \leq d$ *where* $B(i)$ is the vote assign to site *i*. This assignment is treated as an allocation of replicated copies and a vote assigned to the site results in a copy allocated at the neighbour. Due to the data that will be replicated to neighbours, the possible number of data replication from each site, d, should then be:

$d \leq$ quorum (the number of neighbours + a data from the primary site itself).

For example, primary data from the site A which is called as 'a' are replicated to site B and site D which are their neighbours. Site E which holds the primary data 'e' has four neighbours, namely, sites B, D, F and H that will get the replicated data of 'e'. As such, the site E has five replicas. Meanwhile, primary data 'f' from site F are replicated to site C, E and I. The number of quorums used are based on the total number of replicated data and the primary data, d, which can be three, four or five [11,14].

The transaction procedure in BVAG is called as Binary Vote Assignment on Grid Transaction Manager (BVAGTM). The BVAGTM is applied to control the transaction of each data replication process. The primary site of any primary data file and its replica are assigned with different votes depends on their condition. There are two types of votes used in this study as shown in Table I. Zero (0) specified that the site is available (free). Meanwhile, one (1) displayed that the site is unavailable (busy). The status of each site is statistically independent of others. The status of each site is statistically independent of others. When a site is available, the copy at the site is available too; otherwise, it is unavailable [11,14].

TABLE I. TYPE OF STATUS

Type of Status	Definition
0	Available (Site is free or not in used)
1	Unavailable (Site is busy or in used)

There are seven main phases involve in BVATM; Initiate Lock (IL), Propagate Lock (PL), Obtain Quorum(OQ), Update(U), Commit (C), Unlock (UL) and Release Lock (RL) [11,14]. IL phase involves locking the primary site if the primary site is in available (0) status. If the primary site is busy (status = 1), then the primary site will be release (RL). After the primary site has been locked, the PL phase determines the status of each neighbours site. All neighbours sites are locked if they are in available (0) status. Otherwise, the neighbours' sites will be release (RL). Then, OQ phase declares that the quorum obtained is enough for the transaction to be continued. Next, the primary data will be updated in the U phase. Afterward, the updated primary data which is also called as new primary data is replicated to the neighbours' sites in C phase. Last but not least, the transaction will unlock (UL) all the sites that are involved in the transaction. The summary of BVAGTM is shown in Fig. 2.

B. Checkpoint and Rollback-Recovery (CR)

Checkpoint with Rollback-Recovery (CR) is a renowned fault tolerance approach. Checkpoint is a process which stores the recent state of a transaction in stable (non-volatile) storage [9]. It is recognized through the normal execution of a transaction occasionally. The information related to the transaction is saved on a stable storage with the intention of using it in case of site failures. The saved information comprises of the transaction state, its environment, the value of registers, etc. When an error is spotted, the transaction is roll backed to the last saved state [15].

Fig. 3 shows the summary of CR approach. The checkpoint mechanism takes a snapshot of the transaction state and stores the information on some non-volatile storage medium [16]. When failures occur, the restore mechanism copies the last known check pointed state of transaction back into memory and continue processing. The basic idea behind CR is the saving and restoration of transaction state. By saving the current state of the transaction occasionally or before critical code sections, it delivers the baseline information needed for the restoration of lost state in the event of a transaction failure. CR is one of various time efficient fault tolerant approaches [17]. Besides reducing the execution time, CR can also lessen computing resources [18].

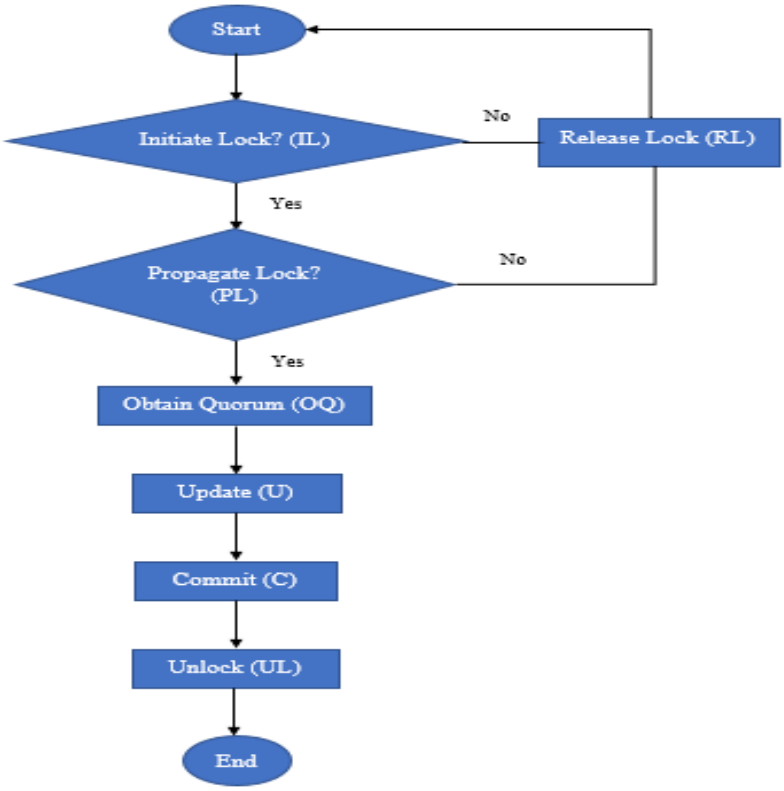

Fig. 2. Binary Vote Assignment on Grid Transaction Manager (BVATM).

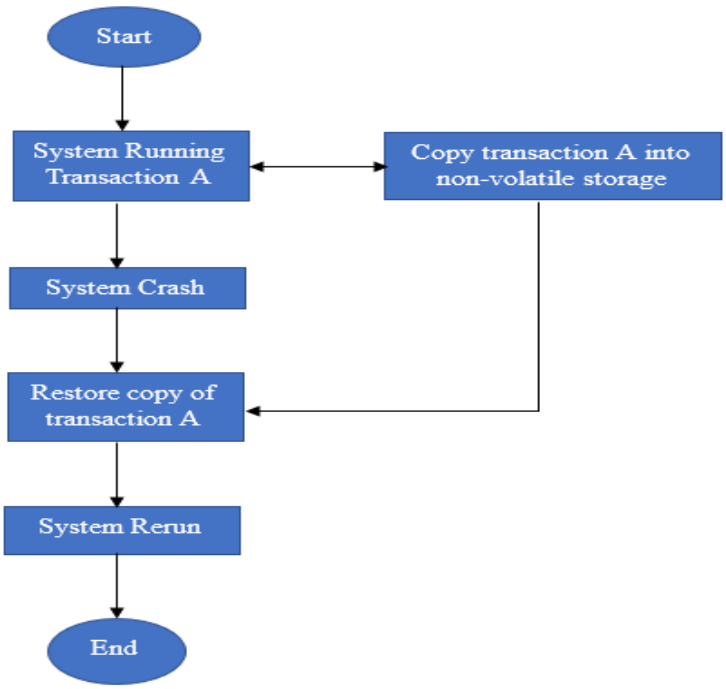

Fig. 3. Checkpoint and Rollback-Recovery (CR).

III. Methodology

In this section, the methodology of the proposed technique called as BVAGCR is described. Fig. 4 illustrated the algorithm of BVAGCR technique applied in Binary Vote Assignment on Grid with Checkpoint and Rollback-recovery Transaction Manager (BVAGCRTM) for a single data replication transaction.

First, the following notations are defined as:

1) T is transaction,
2) C is checkpoint transaction,
3) λ represents different group of transaction T (before and until get quorum). λ can be either α or β,
4) T_λ is transaction of group λ
5) X is the data to be update
6) q_i is the number of queue for transaction T_λ. i = 1,2,3, …
7) $T_{\lambda_{\chi q_1}}$ is transaction of λ for data χ in queue 1
8) $C_{\lambda_{x_{q1}}}$ is checkpoint transaction of λ for data χ in queue 1
9) CP is checkpoint file
10) $CP_{\lambda_{\chi q_1}}$ is checkpoint file for transaction of λ for data χ in queue 1
11) S is the status of the required site
12) PR stands for Primary Replica site
13) NR stands for Neighbour Replica site
14) S_{PR} is status for PR
15) S_{NR} is status for NR
16) S_{PR_χ} is the status of PR which hold data χ
17) S_{NR_χ} is the status of NR which hold data χ
18) Q is the amount of quorum needed to continue the transaction of $T_{\lambda_{\chi q_1}}$
19) D is database
20) D_{PR} is the database for PR
21) D_{NR} is the database for NR
22) D_{PR_χ} is the database of PR that consists of data χ
23) D_{NR_χ} is the database of NR that consists of data χ

A data replication transaction can request to update any data file at any replica. The BVAGCRTM will firstly check whether there is any checkpoint file, $CP_{\lambda_{x_{q1}}}$ that has been saved in BVAGCRTM. If there is none $CP_{\lambda_{x_{q1}}}$ file, BVAGCRTM will accept a new data replication transaction.

A new data replication transaction named as $\left(T_{\lambda_{x_{q1}}}\right)$ which request to update data (x) is in first queue $(q1)$ in BVAGCRTM. The transaction will check the status of primary replica $\left(S_{PR_x}\right)$ which hold data (x) whether it's status is free (0), busy (1) or having a failure (-1). If the primary replica is free to be used in the transaction, the status will be lock as 1. Else, the primary replica will be released because it is unavailable. The status of $\left(S_{PR_x}\right)$ and data (x) are save in the checkpoint file named as $\left(CP_{\lambda_{x_{q1}}}\right)$ file. Next, the transaction will request to lock all the neighbours' replicas, $\left(S_{NR_x}\right)$. If all $\left(S_{NR_x}\right)$is in free status, then it will be lock as 1. However, if one or any $\left(S_{NR_x}\right)$ is busy, all of it will be released for other transactions. The status of each $\left(S_{NR_x}\right)$ is then saved in $CP_{\lambda_{x_{q1}}}$ file.

Afterward, the total of Q is declared and saved in the $CP_{\lambda_{x_{q1}}}$ file. After that, the Update and Commit () function is executed. Data (x) will be update at $\left(D_{PR_x}\right)$ which is the primary database that hold data(x). $\left(D_{PR_x}\right)$ is then saved in

$CP_{\lambda_{x_{q1}}}$ file. Next, $\left(T_{\lambda_{x_{q1}}}\right)$ will commit the replication at all neighbours database, $\left(D_{NR_x}\right)$. All $\left(D_{NR_x}\right)$ is then saved in $\left(CP_{\lambda_{x_{q1}}}\right)$ file. Lastly, $\left(S_{PR_x}\right)$ and all $\left(S_{NR_x}\right)$ will be unlock (status set to 0).

Meanwhile, if there are any $\left(CP_{\lambda_{x_{q1}}}\right)$ that has been saved in BVAGCRTM, then a transaction called as $\left(C_{\lambda_{x_{q1}}}\right)$ will retrieve any information saved in $\left(CP_{\lambda_{x_{q1}}}\right)$. The information that has been saved in $\left(CP_{\lambda_{x_{q1}}}\right)$ are the status of $\left(S_{PR_\chi}\right)$ and all $\left(S_{NR_\chi}\right)$, data (x), (Q), $\left(D_{PR_\chi}\right)$ and $\left(D_{NR_\chi}\right)$. $\left(S_{PR_\chi}\right)$ and $\left(S_{NR_\chi}\right)$ is then set as 1 to declare that it is in used for the transaction. Then, the Update and Commit () function is executed based on the information that has been recover.

Finally, all the locks for $\left(S_{PR_\chi}\right)$ and $\left(S_{NR_\chi}\right)$ will be unlock and are sets to 0. This algorithm can be implemented in the case of either primary site or any neighbours' sites are having a failure. The transaction can only be continued if the failed site is back to normal status (0). This is because the proposed algorithm is not constructed to repair the failure but somehow to let the failed transaction to continue running not from the very beginning after the failure is solved.

Main Algorithm

{

If exist $CP_{\lambda_{x_{q1}}}$ file,

Begin transaction $C_{\lambda_{x_{q1}}}$,

Read $CP_{\lambda_{x_{q1}}}$,

Write S_{PR_χ} = 1,

Commit S_{PR_χ},

Write all S_{NR_χ} = 1,

Commit S_{NR_χ},

Update and Commit (x),

Write S_{PR_χ} = 0,

Write S_{NR_χ} = 0,

Else

Begin transaction $T_{\lambda_{x_{q1}}}$,

If S_{PR_χ} = 0,

Write S_{PR_χ} = 1,

Commit S_{PR_χ},

Save S_{PR_χ} in $CP_{\lambda_{x_{q1}}}$ file,

Else

Write S_{PR_χ} = 0,

End

If all S_{NR_χ} = 0,

Write all S_{NR_χ} = 1,

Commit all S_{NR_χ},

Save all S_{NR_χ} in $CP_{\lambda_{x_{q1}}}$ file,

Else

Write all S_{NR_χ} = 0,

End

Q= majority,

Update and Commit (x),

Write S_{PR_χ} = 0,

Write S_{NR_χ} = 0,

End

}

Function Update and Commit (x)

{

Update x in D_{PR_χ} ,

Commit x in D_{PR_χ} ,

Save D_{PR_χ} , in $CP_{\lambda_{x_{q1}}}$ file,

Commit replication x in all D_{NR_χ},

Save all D_{NR_χ}, in $CP_{\lambda_{x_{q1}}}$ file,

}

Fig. 4. Algorithm of BVAGCRTM.

In the next section, an experiment considering failure condition has been conducted in order to evaluate the performance of BVAG and BVAGCR. The results obtained and the discussions about the results are also explained in the next section.

IV. Result And Discussion

An experiment of a single transaction with failure condition occurred at the primary replica was conducted in this research using MATLAB simulation. The time between failure and recovery is assumed as 10 seconds. The transaction is continued after failure recovery. In this transaction, the site E is considered as the primary replica holding primary data e. Meanwhile, sites B, D, F, H are the neighbor's replica which will be receiving the copy of data e from site E. In this case, the transaction, $\left(T_{\lambda_{e_{q1}}}\right)$ requests to update data (e) and replicate the data into the neighbors' replica. A transaction failure is considered to occur in the Update (U) phase seeing that it is the critical phase in BVAG and BVAGCR.

Fig. 5 and Fig. 6 demonstrated the flow of the transaction, $\left(T_{\lambda_{e_{q1}}}\right)$ with failure condition for BVAG and BVAGCR. As can be seen in Fig. 5 (BVAG), the information related to the transaction in BVAGCRTM was not being saved in a checkpoint file ($C_{\lambda_{e_{q1}}}$). Thus, when a failure occur in T_{10}, the transaction needs to be started all over again as there is no information recovery can be done if failure occurs.

Meanwhile, in BVAGCR (refer Fig. 6), the information related to the transaction is being saved in the checkpoint file

(highlighted with grey) for each phase of BVAGCRTM. Thus, once the transaction $\left(T_{\lambda_{e_{q1}}}\right)$ failed as in T_{13}, the information can be retrieved from the checkpoint file $\left(C_{\lambda_{e_{q1}}}\right)$ and the transaction can be resume from the Update phase (U) as the failure occurred in that particular phase.

The execution times for each phase involved in BVAG and BVAGCR methods are recorded before and after failure occurred as shown in Table II and Table III. As presented in Table II, the overall time taken to complete a transaction using BVAG is 15.8574 seconds which include the estimation time duration of failure (10 seconds). The transaction had run four phases which are IL, PL, CQ, OQ that took 0.4931 seconds to be executed before failure occurred. After failure recovery, the transaction needs to be run again from the start due to no checkpoint file, ($C_{\lambda_{e_{q1}}}$) found. The time taken to rerun the transaction is 5.3643 seconds. For the critical phase U, the time needed to finished it is 4.7809 seconds.

As displayed in Table III, BVAGCR need 10.8381 seconds of time to finish a transaction before failure occurred and after failure recovery which also takes account of the estimate time duration of failure (10 seconds). The transaction had performed four phases same as BVAG which are IL, PL, CQ, and OQ that acquired 0.7443 seconds before failure happened. Once the failure had been recovered, the transaction only needs to rerun at U phase onwards because it had retrieved the information about the transaction which is save in a checkpoint file ($C_{\lambda_{e_{q1}}}$). Based on the ($C_{\lambda_{e_{q1}}}$) file, the last saved information of the current transaction is in phase U. The time used to rerun the transaction from U phase until the data has been replicated to all neighbors' sites is 0.0938 seconds. For the critical phase (U), the time needed to finish it is 0.0516 seconds.

Fig. 7 shows the comparisons of time taken for total time, update phase, execution time before failure and execution time after failure between BVAG and BVAGCR. Based on Fig. 7, the proposed method, BVAGCR used more time (0.7443 seconds) than BVAG (0.4931 seconds) in execution time before failure because it took extra time to save the information about the current transaction into a checkpoint file. However, after failure recovery, BVAGCR spend less time to complete the transaction than BVAG as it does not have to rerun the transaction from the beginning.

REPLICA	E	B	D	F	H
TIME					
T_1	unlock S_{E_e}	unlock S_{B_e}	unlock S_{D_e}	unlock S_{F_e}	unlock S_{H_e}
T_2	begin transaction $T_{\lambda_{e_{q1}}}$				
T_3	initiate lock: S_{E_e}				
T_4	write lock S_{E_e} = 1				
T_5	get lock S_{E_e}				
T_6	propagate lock: S_{B_e}, S_{D_e}, S_{F_e}, S_{H_e}				
T_7		write lock S_{B_e} = 1	write lock S_{D_e} = 1	write lock S_{F_e} = 1	write lock S_{H_e} = 1
T_8	get lock : S_{B_e}, S_{D_e}, S_{F_e}, S_{H_e}				
T_9	Obtain majority quorum (Q)				
T_{10}	update (e) in D_{E_e} and failure occurred				
T_{11}	unlock S_{E_e}	unlock S_{B_e}	unlock S_{D_e}	unlock S_{F_e}	unlock S_{H_e}
T_{12}	begin transaction $T_{\lambda_{e_{q1}}}$				
T_{13}	initiate lock: S_{E_e}				
T_{14}	write lock S_{E_e} = 1				
T_{15}	get lock S_{E_e} Σ quorum $\left(S_{q_e}\right)$ = 1				
T_{16}	propagate lock: S_{B_e}, S_{D_e}, S_{F_e}, S_{H_e}				
T_{17}		write lock S_{B_e} = 1	write lock S_{D_e} = 1	write lock S_{F_e} = 1	write lock S_{H_e} = 1
T_{18}	get lock : S_{B_e}, S_{D_e}, S_{F_e}, S_{H_e}				
T_{19}					
T_{20}	Obtain majority quorum (Q)				
T_{21}	update (e) D_{E_e}				
T_{22}	Commit update in D_{E_e}				
T_{23}		Commit replication in D_{B_e}	Commit replication in D_{D_e}	Commit replication in D_{F_e}	Commit replication in D_{H_e}
T_{24}	write lock S_{E_e} = 0	write lock S_{B_e} = 0	write lock S_{D_e} = 0	write lock S_{F_e} = 0	write lock S_{H_e} = 0

Fig. 5. BVAG Transaction's Flow (Failure Occurred While Updating Data).

REPLICA	E	B	D	F	H
TIME					
T_1	unlock S_{E_e}	unlock S_{B_e}	unlock S_{D_e}	unlock S_{F_e}	unlock S_{H_e}
T_2	begin transaction				
T_3	initiate lock: S_{E_e}				
T_4	write lock S_{E_e} = 1				
T_5	Save S_{E_e} in $C_{\lambda_{e_{q1}}}$ file				
T_6	get lock S_{E_e}				
T_7	propagate lock: S_{B_e}, S_{D_e}, S_{F_e}, S_{H_e}				
T_8		write lock S_{B_e} = 1	write lock S_{D_e} = 1	write lock S_{F_e} = 1	write lock S_{H_e} = 1
T_9	save S_{B_e}, S_{D_e}, S_{F_e}, S_{H_e} in $C_{\lambda_{e_{q1}}}$ file				
T_{10}	get lock S_{B_e}, S_{D_e}, S_{F_e}, S_{H_e}				
T_{11}	obtain majority quorum (Q)				
T_{12}	Save S_{q_e} in $C_{\lambda_{e_{q1}}}$ file				
T_{13}	update (e) in D_{E_e} and failure occurred				
T_{14}	begin transaction $C_{\lambda_{e_{q1}}}$				
T_{15}	Retrieve information from $CP_{\lambda_{x_{q1}}}$ file				
T_{16}	write lock S_{E_e} = 1	write lock S_{B_e} = 1	write lock S_{D_e} = 1	write lock S_{F_e} = 1	write lock S_{H_e} = 1
T_{17}	update (e) in D_{E_e}				
T_{18}	Commit update in D_{E_e}				
T_{19}	Save D_{E_e} , in $CP_{\lambda_{x_{q1}}}$ file				
T_{20}		Commit replication in D_{B_e}	Commit replication in D_{D_e}	Commit replication in D_{F_e}	Commit replication in D_{H_e}
T_{21}	Save D_{B_e} , D_{D_e} , D_{F_e} , D_{H_e} in $CP_{\lambda_{x_{q1}}}$ file				
T_{22}	write lock S_{E_e} = 0	write lock S_{B_e} = 0	write lock S_{D_e} = 0	write lock S_{F_e} = 0	write lock S_{H_e} = 0

Fig. 6. BVAGCR Transaction's Flow (Failure Occurred While Updating Data).

TABLE II. EXECUTION TIME TAKEN FOR BVAG DATA REPLICATION TRANSACTION (A) BEFORE FAILURE OCCUR, (B) AFTER FAILURE OCCUR

PHASES	**TIME(SECONDS)**
INITIATE LOCK[A]	0.2320
PROPAGATE LOCK[A]	0.2408
OBTAIN QUORUM[A]	0.0203
UPDATE AND FAILURE OCCUR	10.0000
INITIATE LOCK[B]	0.2331
PROPAGATE LOCK[B]	0.2354
OBTAIN QUORUM[B]	0.0300
UPDATE[B]	4.7809
COMMIT[B]	0.0602
UNLOCK[B]	0.0247
TOTAL	15.8574

TABLE III. EXECUTION TIMES TAKEN FOR BVAGCR DATA REPLICATION TRANSACTION (A) BEFORE FAILURE OCCUR. (B) AFTER FAILURE OCCUR

PHASES	**TIME(SECONDS)**
INITIATE LOCK[A]	0.2390
PROPAGATE LOCK[A]	0.4795
OBTAIN QUORUM[A]	0.0258
UPDATE AND FAILURE OCCUR	10.0000
UPDATE[B]	0.0516
COMMIT[B]	0.0149
UNLOCK[B]	0.0273
TOTAL	10.8381

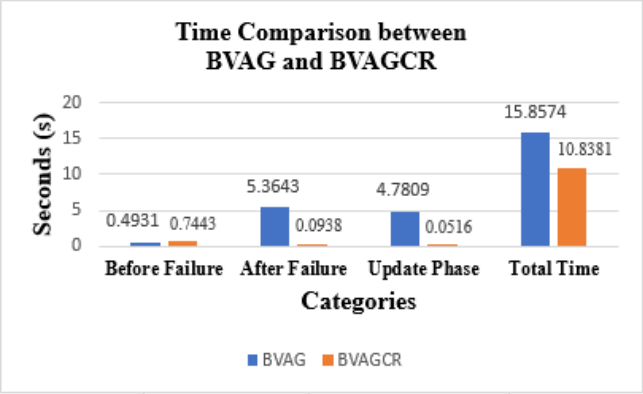

Fig. 7. Time Comparison between Standard BVAG and BVAGCR.

Meanwhile, the total time taken for a single transaction of BVAGCR (10.8381 seconds) is shorter than BVAG (15.8574 seconds). BVAGCR has improved the standard BVAG method by 31.65 % in terms of total execution time. Besides that, the efficiency of BVAGCR can also be seen when it successfully improved the performance of standard BVAG in critical phase which is the Update (U) phase by 98.82%. Thus, based on the results obtained, it can be said that the objective of this study which is to improve the efficiency of standard BVAG by proposing a new data replication technique with fault tolerance approach (BVAGCR) has been successfully achieved.

V. Conclusion

This study has explored a new combination of data replication and fault tolerance approach called as BVAGCR. The performance of BVAGCR is tested using a simulation of MATLAB. A comparison between standard BVAG and BVAGCR has been done in order to evaluate the effectiveness of implying the CR fault tolerance approach in BVAG data replication technique. The result gained from this study shows that the proposed BVAGCR has outperformed standard BVAG in terms of total execution time, time taken to execute the U phase and time taken to rerun the transaction after failure recovery.

Therefore, BVAGCR can be proposed as an alternative technique which is efficient and reliable to replicate data in failure condition. To test the robustness of the proposed BVAGCR, future work should explore the application of this proposed method on big data. Besides that, BVAGCR can also be implemented with data mining method in order to get more competent performance of the data replication technique.

Acknowledgment

This study is supported by the Fundamental Research Grant Scheme (RDU190185) with Reference no: FRGS/1/2018/ICT03/UMP/02/3 that sponsored by Ministry of Higher Education Malaysia (MOHE). In addition, this study is also supported by Grant Code: (20UQU0074DSR) under the Deanship of Scientific Research at Umm Al-Qura University. Appreciation is also conveyed to University Malaysia Pahang for project financing under UMP Short Term Grant RDU1903122 and UMP PGRS RDU170329.

References

[1] N. Dogra and S. A. Singh, “A survey of dynamic replication strategies in distributed systems,” International Journal of Computer Applications., vol. 110, no. 11, Jan 2015.

[2] S. H. S. A. Ubaidillah and N. Ahmad, "Fragmentation Techniques for Ideal Performance in Distributed Database–A Survey." International Journal of Software Engineering and Computer Systems., vol. 6, no. 1, pp. 18-24, May 2020.

[3] Kumar, Nirmal, and Girish V. Mattur. "Data replication in a database environment." U.S. Patent Application No. 15/813,828.

[4] B. Kemme, “Data Replication,” In: Liu L., Özsu M. (eds) Encyclopedia of Database Systems. Springer, New York, NY, 2017.

[5] A. Sarı and E. Çağlar, “Performance Simulation of Gossip Relay Protocol in Multi-Hop Wireless Networks,” Owner: Girne American University Editor: Asst. Prof. Dr. İbrahim Erşan Advisory Board: Prof. Dr. Sadık Ülker Assoc. Prof. Dr. Zafer Ağdelen Cover Graphic Design: Asst. Prof. Dr. İbrahim Erşan., pp. 145, 2015.

[6] A. Sari and M. Akkaya, “Fault tolerance mechanisms in distributed systems,” International Journal of Communications, Network and System Sciences., vol. 8, no. 12, pp. 471, Dec 2015.

[7] A. Benoit, A. Cavelan, F. M. Ciorba, V. Le Fevre and Y. Robert, “Combining checkpointing and replication for reliable execution of linear workflows,” in IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2018, pp. 793-802.

[8] S. Desai, N. Pendharkar and Veritas Technologies LLC, “Data replication techniques using incremental checkpoints,” United States patent US 9,495,264. 15 Nov 2016.

[9] Y. Mo, L. Xing, Y. K. Lin and W. Guo, “Efficient Analysis of Repairable Com-puting Systems Subject to Scheduled Checkpointing,” IEEE Transactions on Dependable and Secure Computing., Sep 2018.

[10] R. Garg and A. K. Singh, “Fault tolerance in grid computing: state of the art and open issues,” International Journal of Computer Science & Engineering Survey (IJCSES)., vol. 2, no. 1, pp. 88-97, Feb 2011.

[11] A. Noraziah, A. A. Fauzi, S. H. Ubaidillah, Z. Abdullah, R. M. Sidek, M. A. Fakhreldin, “Managing Database Replication Using Binary Vote Assignment on Grid Quorum with Association Rule,” Advanced Science Letters., vol. 24, no. 10, pp. 7834-7837, Oct 2018.

[12] J. P. Yang, “Elastic load balancing using self-adaptive replication management,” IEEE Access., vol. 22, no. 5, pp. 7495-7504, Nov 2016.

[13] Muhammad Ahsan Raza, M. Rahmah, A. Noraziah, Mahmood Ashraf, “Sensual Semantic Analysis for Effective Query Expansion “, International Journal of Advanced Computer Science and Applications, vol. 9, no. 12, pp. 55-60, 2018.

[14] A. A. Fauzi, A. Noraziah, T. Herawan, Z. Abdullah, R. Gupta, “Managing Fragmented Database Using BVAGQ-AR Replication Model,” Advanced Science Letters., vol. 23, no. 11, pp. 11088-11091, Nov 2017.

[15] S. Veerapandi, S. Gavaskar and A. Sumithra, “A Hybrid Fault Tolerance System for Distributed Environment using Check Point Mechanism and Replication,” International Journal of Computer Applications., vol. 975, pp. 8887, 2017.

[16] J. Wu. “Checkpointing and recovery in distributed and database systems,” 2011.

[17] D. Poola, M. A. Salehi, K. Ramamohanarao and R. Buyya, “A taxonomy and survey of fault-tolerant workflow management systems in cloud and distributed computing environments,” in Software Architecture for Big Data and the Cloud., pp. 285-320, Jan 2017.

[18] T. Sterling, M. Anderson and M. Brodowicz, “High performance computing: modern systems and practices,” Morgan Kaufmann., Dec 2017.