VOL. 11, NO. 9, MAY 2016 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

BERKELEY SYNCHRONIZED ALGORITHM BASED FAULT TOLERANT MECHANISM FOR COMPUTATIONAL DATA GRID IN CLOUD ENVIRONMENT

Ramachandra V. Pujeri[1], S. N. Sivanandam[2] and N. Suba Rani[3]
[1]MIT College of Engineering, Kothrud, Pune, Maharashtra, India
[2]Educational Advisor, Karpagam Group of Institutions, Coimbatore, Tamil Nadu, India
[3]Pollachi Institute of Engineering and Technology, Coimbatore, Tamil Nadu, India

ABSTRACT

Computational Data Grid provides massive resource sharing and aggregated computing resources in a dynamic manner. Due to the limitation of available heterogeneous resources distributed through several networks in computational data grid, occurrence of failure poses severe problem. Providing efficient fault tolerance mechanism is a key optimization technique for improving scalability and attain QoS based fault tolerant dynamic replication in a wise manner. In this paper, Merkle Damgard Clock Synchronized based Fault Tolerant (MDCS-FT) mechanism is developed to overcome the fault occurrence in computational data grid. MDCS-FT in cloud services allows for concurrent transaction without relying on a centralized grid component, which amounts for better scalability. Clock Synchronized with Berkeley algorithm uses the optimized different data grid sequences to attain QoS based fault tolerant dynamic replication. Berkeley algorithm in MDCS-FT mechanism is more suitable for easily identifying the fault with time server. Time server in MDCS-FT Mechanism periodically fetches the time from all the clients and averages the results on cloud zone to secure data objects replication in cloud data grid by removing the fault. Experimental results demonstrate that the proposed mechanism achieves better performance by improving the scalability and QoS (error rate and transmission delay) and minimizes the fault tolerance compared to the state-of-the-art works.

Keywords: computational data grid, fault tolerance, clock synchronized, berkeley algorithm, dynamic replication, time server.

1. INTRODUCTION

Fault tolerance mechanism for computational data grid has received great attention due to the distributed nature of the data. Due to the inaccessibility of network, development difficulty, faulty resources, fault may take place in the result or performance of the system may be corrupted. A fault tolerant service identifies errors and recovers them without involvement of any external agents, such as humans. Many research persons have contributed in this field and also developed a lot of fault tolerance mechanism. Enhanced Dynamic Hierarchical Replication in Data Grid (EDHR-DG) [1] minimized the data access time by applying Weighted Scheduling Strategy. However, it experience maintains issues and fail to plan real data grids. Optimized Approach on Cloud Storage (OACS) [2] efficiently improved the storage and access efficiency of small files by applying file merging and pre-fetching scheme. But, it has less storage efficiency with the aim of optimizing the file merging/grouping strategy and the peak size of a merged file/logic unit.

To avoid the fault occurrence in cloud, Merkle DamgardClock Synchronized based Fault Tolerant (MDCS-FT) mechanism is proposed in this paper. MDCS-FT mechanism has two strategies. First a novel MerkleDamgard Hash Dynamic Replication (MDHDR) model is developed with objective of improving the scalability. The MDHDR model considers the number of transaction blocks to be delivered, the initial vector and the intermediate function in order to obtain the MerkleDamgard Hash function. Second a clock synchronization strategy, called Clock Synchronized with Berkeley algorithm is designed with the aim of improving the QoS in an efficient manner.

The rest of the paper is organized as follows. We present the issues of fault tolerance in Section 2 and present the proposed mechanism with the aid of block diagram and algorithmic steps. Section 3 provides with the experimental setup to conduct experiments. The simulation results are analysed and discussed in Section 4. Finally, Section 5 concludes the paper.

2. RELATED WORK

Fault tolerance mechanism is used to preserve the transmission of expected services in spite of the fault existence caused the errors in the system itself. Fault tolerance mechanism avoids the failures in the presence of faults. A novel architecture, Advanced Cloud Protection System (ACPS) [3] was introduced with the objective of minimizing the resilient against attacks using security management layer. Though storage and attacks were addressed, but the job execution time was compromised. Pre-fetching based Dynamic Data Replication Algorithm (PDDRA) [4] aiming at reducing the job execution time was introduced with the aid of pre-fetching and replacement algorithm. Another method to address job execution time was introduced in [5] for smart grid applications.

Privacy preserving the collusion tolerance is the two main issues to be tackled for data grid in cloud environment. An external aggregator protocol was introduced in [6] aiming at reducing the attack and therefore minimize the collision rate. Though collision rate was addressed, but the computational cost remained

unsolved. With the objective of reducing the computational cost in [7] a novel Partition Solution Space based approach was introduced. This approach not only reduced the computational cost but also the rate of scalability. In [8], energy-efficient fault tolerance mechanism was introduced with the aid of k-out-of-n-computing. With the increase in the data size of the user, fault also increases. To address the faulty rate a mechanism for distributed accountability was introduced in [9].

Current public cloud implementations mainly concentrate on providing scaled-up and scaled-down computing power and storage. In [10], Decentralized Self-Adaptation Mechanism was introduced using marked-based heuristics to minimize the cost of decentralization. However, scalability remained unaddressed. Fault tolerant load balancing algorithm was designed in [11] to attain minimum response time and optimal computing node utilization. However, this method does not consider the security related issues. A system-level fault-tolerant mechanism [12] was developed for message passing applications where it able to detect node faults and recovering the processes automatically. But, it requires special attention when a process loses several connections. Sparse Grids and a Fault-tolerant Combination Technique [13] were presented to reduce the computational complexity and its inherent ABFT properties. Multi agent System Architecture of the SETL Grid Control Module is illustrated in [14] for the control and monitoring of the SETL process. However, it requires appropriate tools for an efficient storage, analysis, and visualization of the available data. Based on the aforementioned methods and techniques stated, a novel Fault Tolerant mechanism is designed in cloud using Clock Synchronized with Berkeley algorithm for efficient cloud service provisioning which detailed described in the forthcoming subsections.

3. PRELIMINARIES

In this section, we first describe the system model and then a MerkleDamgard Hash Dynamic Replication model without relying on a centralized grid component which is important for achieving good performance and scalability.

3.1 System model

In this work, we presume a MerkleDamgard Hash failure model where faulty users (i.e. client or servers) in cloud services act in an arbitrary manner and that at most 'r' replicas are faulty out of a total of '$n = 3r + 1$'replicas. We also assume that the users are connected by an untrustworthy network that may dwindle while delivering data packets, distort them, or deliver the data packets in an irregular manner.

3.2 Design of MerkleDamgard hash dynamic replication model

The MerkleDamgard Hash Dynamic Replication model is designed in such a manner that the cloud services allows for concurrent transaction without relying on a centralized grid component. Let us consider that the secure data objects replica in cloud is implemented by 'r'replicas and executes the processes as requested by the users (i.e. clients). The replicas and the users (i.e. clients) run in different nodes that are connected by a network. Figure-1 shows the MerkleDamgard Hash Dynamic Replication model with separate replicas and clients.

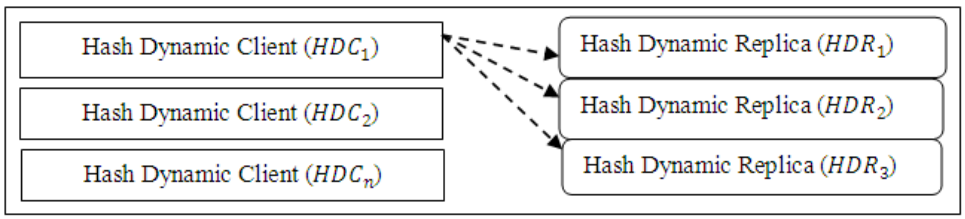

Figure-1. MerkleDamgard hash dynamic replication model with separate clients and replicas.

The MerkleDamgard Hash Dynamic Replication model Replicas use a MerkleDamgardHash function 'MDH()' to evaluate transaction blocks and uses message authentication codes (MACs) to authenticate transaction data packets 'DP_i' coming from all user (i.e. client) requests. Let us consider the transaction blocks as 'TB_i' where 'V' denotes the initial vector with 'f_i' representing the intermediate function obtainedas transaction blocks are appended with 'ZP' representing the zero pads. Figure-2 given below shows the Construction of MerkleDamgard Hash function.

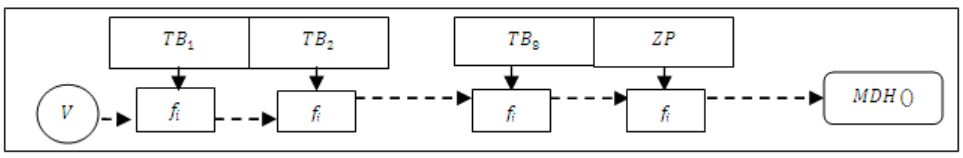

Figure-2. Construction of MerkleDamgard hash function.

As shown in the figure, for each transaction block, the function 'f_i' obtains the initial value, in addition to the initial vector value 'V' integrates with the transaction block 'TB_i' to obtain intermediate result and then zero padded with 'ZP' to produce the final MerkleDamgard Hash function '$MDH()$' respectively. The '$MDH()$' is mathematically formulated as given below:

$$MDH() = \sum_{i=1}^{n} f_i \cup TB_i \cup ZP_i \quad (1)$$

From (1), 'f_i' represents the intermediate function, 'TB_i' represents the transaction blocks and 'ZP_i' the zero padding inserted resulting in the MDH function. Once the MDH function is evaluated, message authentication codes (MACs) uses session key pairs for each replica pair. Let us consider replica pair 'a' and 'b'. Then, the mathematical formulation for MAC evaluation to perform authentication is as given below:

$$MDH(A_{i,j}) \rightarrow Transaction\ Block\ sent\ from\ i\ to\ j \quad (2)$$

$$MDH\ (A_{j,i}) \rightarrow Transaction\ Block\ sent\ from\ j\ to\ i \quad (3)$$

From (2) and (3), '$A_{i,j}$' evaluates the MAC transaction block sent from 'i' to 'j' whereas '$A_{j,i}$' evaluates the MAC transaction block sent from 'j' to 'i'.Each replica shares a secret key with each client that is used for efficient communication between the replica and the client and is formulated as given below:

$$Secret_{key} = \sum_{i=1}^{n} (User_i, Key_i t) \quad (4)$$

From (4), the secret key '$Secret_{key}$' is obtained for each user (i.e. client) '$User_i$' and assigned with a key 'Key_i' for each time interval 't' respectively. The key is updated in a periodic manner by the client (i.e. at time t). If the key updating is not performed by the client, the replica discards the current key for that client. As a result the client refreshes the key.

Input: Transaction Blocks '$TB_i = TB_1, TB_2, \ldots, TB_n$', Zero Padding '$ZP$', Transaction Data Packets '$DP_i = DP_1, DP_2, \ldots, DP_n$', Vector '$V$', User '$User_i = User_1, User_2, \ldots, User_n$', Key = '$Key_i = Key_1, Key_2, \ldots, Key_n$', Time '$t$'
Output: Scalable addressability of concurrent transaction
Step 1: Begin Step 2: For each Transaction Blocks 'TB_i' Step 3: For each Transaction Data Packets 'DP_i' Step 4: Measure MerkleDamgard Hash function '$MDH()$' using (1) Step 5: Perform key sharing using (4) Step 6: End for Step 7: End for Step 8: End

Figure-3. MerkleDamgard Hash Replication algorithm.

From the Figure-3 given above the MerkleDamgard Hash Replication algorithm performs two important steps. First, for each transaction blocks and data packets, MerkleDamgard Hash function is evaluated. Followed by this, the second step performs key sharing for efficient data packet communication between the users with the objective of improving the scalability which in turn reduce the fault occurrence significantly.

3.3 Design of clock synchronization with berkeley

Clock Synchronized with Berkeley algorithm uses the optimized different data grid sequences to attain QoS based fault tolerant dynamic replication. A clock 'C' is said to be non-faulty if there said to exists a threshold (i.e. drift value for non-faulty clock) 'α_1' such that for time 't_1' and 't_2' with different data grid sequences 'DGS_i' attain QoS based fault tolerant and is mathematically formulated as given below:

$$1 - \alpha_1 < \frac{C(t_2) - C(t_1)}{t_2 - t_1} < 1 + \alpha_2 \quad (5)$$

The error rate is measured by each user on the basis of its own clock synchronized with MDH function. Two clocks 'C_1' and 'C_2' of two users '$User_1$' and '$User_2$' are said to be synchronized at time 't' if '$C_1(User_1[t]) - C_2(User_2[t]) < \delta$'.From this, set '$n$' clocks are said to be synchronized with 'n' users if the following condition is said to be satisfied.

$$if\ C_1(USer_1[t]) - C_2(User_2[t]) \ldots\ldots C_n(User_n[t]) < \delta \quad (6)$$

From (6), 'C_1' and 'C_2' are clock synchronized with user '$User_1$' and '$User_2$' respectively for a specified constant 'δ'. Due to the non-zero drift values of all clocks, the clock rate does not remained synchronized without certain periodic cycle. So, in the proposed mechanism, the users periodically resynchronize clock values in order to maintain an efficient global time with the objective of reducing the error rate.

The periodical resynchronization of clock values for each user with respect to global time is performed between successive grid sequences is called the resynchronize clock intervals (i.e. a constant) denoted by

'RCI'. Let us consider a time 't_0' when user '$User_1$' began its process, then the time of 'i^{th}' resynchronize value is then formulated as given below:

$$t_i = t_0 + iRCI \quad (7)$$

Let 'a' and 'b' be two non-faulty nodes and 't' be the clock time for node 'a'. Then, when a node 'a' wants to broadcast transaction block to node 'b' at time '$C_a(t)$', this transaction block '$C_a(t)$' reaches 'b'after certain amount of time with a delay 'D'. Therefore, the transaction block message is delivered to node 'b' with a delay and is formulated as given below:

$$(T_i, a, b) \rightarrow C_a(t) + D \quad (8)$$

Input: Threshold 'α_1', Time 't_1, t_2', Data Grid Sequences '$DGS_i = DGS_1, DGS_2, \ldots, DGS_n$', User '$User_i = User_1, User_2, \ldots, User_n$', Transaction Blocks '$TB_i = TB_1, TB_2, \ldots, TB_n$'
Output: Minimized error rate during transaction block delivery between users
Step 1: Begin Step 2: For each data grid sequences DGS_i Step 3: Measure QoS based fault tolerant using (5) Step 4: End for Step 5: For each user $User_i$ Step 6: Measure clock synchronization using (6) Step 7: Measure resynchronize clock intervals using (7) Step 8: Deliver transaction blocks between nodes using (8) Step 9: End for Step 10: End

Figure-4. Clock synchronization algorithm.

This error is small because of the non-faulty nature of the nodes and the communication between the broadcast of transaction block is established. Therefore, the error rate during the broadcast of transaction block is reduced in a significant manner. Figure-4 shows the clock synchronization algorithm for efficient transaction block delivery between users. As shown in the figure 4, the Clock Synchronization algorithm performs two important steps. During the first step for each data grid sequences, QoS based fault tolerant value is measured. Followed by which for each user in the second step, clock synchronization and resynchronize clock intervals are measured. Finally, transaction blocks between users are delivered. This QoS based fault tolerant clock synchronization helps in reducing the error rate.

3.4 Design of berkeley algorithm

Finally, Berkeley algorithm in MDCS-FT mechanism is more suitable for easily identifying the fault with time server. Time server in MDCS-FT Mechanism periodically fetches the time from all the clients and averages the results on cloud zone to secure data objects replication in cloud data grid by removing the fault. MDCS-FT Mechanism with Berkeley algorithm highlights the work with fault tolerance and also improves the cloud service provisioning. The client whose clock differs by a value outside of a given tolerance in MDCS-FT Mechanism is disregarded as faults. The users involved in the synchronization each execute a time daemon process that is responsible for delivering the transaction block to other users. One of these users is designated to be the master. The others users are referred to as the slaves. The master user polls each node's periodically, asking it for the time. When all the results are obtained the master user computes the average time in addition to its own time and correspondingly decides the faulty and non-faulty nodes with time server. Figure-5 shows the Berkeley Clock Synchronized model.

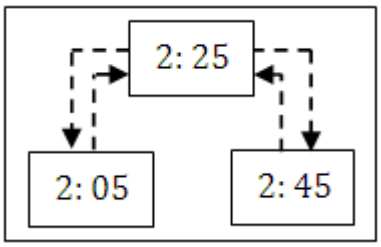

Figure-5. Berkeley clock synchronized model.

Let us consider three nodes (i.e. users) '$User_1$' have timestamp 'TS_1' '2: 25', '$User_2$' have timestamp 'TS_2' '2: 05' and '$User_3$' have timestamp 'TS_3' '2: 45' respectively. As shown in the figure, with the Berkeley Clock Synchronized model, the highest timestamp with '2: 25' is elected as the master user (i.e. '$User_1$'). The other two nodes (i.e. '$User_2$' and '$User_3$') are the slave users. The master user '$User_1$'sends a synchronize query to two other users '$User_2$' and '$User_3$'. As a response the two slave users sends the time stamp to the master user. The master user performs the averages the results on the cloud zone using three time stamps (i.e. of '$User_2$' and '$User_3$' and its own time stamp '$User_1$') and is formulated as given below:

$$BS = [Timestamp\ (User_2) + Timestamp\ (User_3) + Timestamp\ (User_1)/3] \quad (9)$$

From (9), the Berkeley Synchronized value is obtained using the timestamp of '$User_1, User_2\ and\ User_3$'. The three synchronized values are then averaged (i.e. (2.25 + 2.05 + 2.45)/3 = 2.25). This average value forms the threshold 'Th'. The value lesser than 'Th' is considered as the non-faulty node whereas the value greater than 'Th' is considered as the faulty node and disregarded from the network. Figure-6 shows the Berkeley Synchronized algorithm.

Input: Users '$User_i = User_1, User_2, \ldots, User_n$', Timestamp '$TS_i = TS_1, TS_2, \ldots, TS_n$', Threshold 'Th'
Output:
Step 1: Begin Step 2: For each users $User_i$ Step 3: For each timestamp TS_i //Identify master user and slave user Step 4: Measure timestamp Step 5: Assign highest timestamp user as master user Step 6: Send synchronize query by master user to slave users Step 7: Slave users reply by sending their timestamp Step 8: Measure average timestamp (i.e. Thusing (9)) Step 9: If 'Th' <Timestamp ($User_i$) Step 10: User is a non-faulty node Step 11: End if Step 12: If 'Th' >Timestamp ($User_i$) Step 13: User is a faulty node Step 14: Disregard $User_i$ Step 15: End if Step 14: End for Step 15: End for Step 16: End

Figure-6. Berkeley Synchronized algorithm.

As shown in the above Figure, the Berkeley Synchronized algorithm performs the secure data objects replication in cloud data grid by removing the faulty nodes in the cloud zone. This is achieved through Berkeley Synchronized value. This is turn efficiently identifies the faulty node and disregards them resulting in the reduced transmission delay.

4. Experimental settings

In this section, the investigational setup for designing MDCS-FT Mechanism is explained and the experiments were conducted on using GridSim simulator performed on Cloud environment. The GridSim simulator performed on Cloud environment offers distinct resource configurations for several virtual machine instances. Each virtual machine instance type is configured with a specific amount of memory, CPUs, and local storage. The MDCS-FT Mechanism is equipped with two quad core 2.33-2.66 GHz Xeon processors (8 cores total), 7 GB RAM, and 1690 GB local disk storage.

The performance evaluation tests aimed at comparing the Enhanced Dynamic Hierarchical Replication in Data Grid (EDHR-DG) [1] and Optimized Approach on Cloud Storage (OACS) [2] with the proposed MerkleDamgard Clock Synchronized based Fault Tolerant (MDCS-FT) mechanism. So to study the MDCS-FT mechanism using the simulator, we proposed a simulation environment that has the following parameters: the number of users varies between 20 and 140 with transaction block size of ranging from 5 to 35, the transaction data packets is equal to 5 packets to 35 packets with packet size of l pieces of data with a size 10 MB. The following parameters including scalability, error rate, fault tolerance rate and transmission delay in cloud serviceare evaluated.

5. DISCUSSION

The MerkleDamgardClock Synchronized based Fault Tolerant (MDCS-FT) mechanism is compared against with the existing Enhanced Dynamic Hierarchical Replication in Data Grid (EDHR-DG) [1] and Optimized Approach on Cloud Storage (OACS) [2].The experimental results using Grid Sim simulator in Cloud environment are compared and analysed through table and graph form given below.

5.1 Impact of scalability

Scalability is one of the most important standard metrics used to measure the performance of fault tolerant systems for computational data grid in cloud environment. Scalability is mathematically formulated as given below;

$$S = (TB * Time_{TB}) \quad (10)$$

From (10), scalability 'S' is measured on the basis of the product of total number of transaction blocks submitted 'TB' to total amount of time necessary to complete the transaction block '$Time_{TB}$'. Scalability is used to measure the ability of the computational data grid to accommodate the transaction blocks.

Table-1. Tabulation for scalability.

No. of Transaction Block (TB)	Scalability (ms)		
	MDCS-FT	EDHR-DG	OACS
5	0.68	0.89	0.96
10	0.75	0.94	1.00
15	0.82	1.01	1.07
20	0.85	1.04	1.10
25	0.91	1.10	1.16
30	0.99	1.20	1.26
35	1.10	1.31	1.37

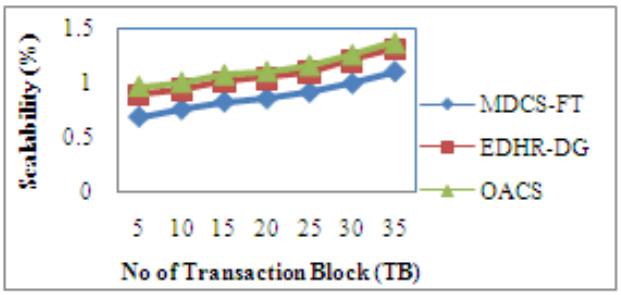

Figure-7. Measure of scalability.

To support transient performance, in Table-1 we apply a MerkleDamgard Hash Replication algorithm and comparison made with two other existing methods EDHR-DG [1] and OACS [2]. From the above tabulation, scalability refers to the number of transaction blocks addressed at minimum time interval. Figure-7 show that the MerkleDamgard Clock Synchronized based Fault Tolerant (MDCS-FT) mechanism provides higher amount of scalability when compared to EDHR-DG [1] and OACS [2]. With the application of MerkleDamgard Hash Replication algorithm, MerkleDamgard Hash function evaluates transaction blocks in an efficient manner using intermediate function and zero padding. This in turn authenticates transaction data packets using Message Authentication Code using session key pairs for each replica pair helps to improve the number of transaction blocks being addressed and therefore the scalability rate for computational data grid is improved using MDCS-FT mechanism by 23.27% compared EDHR-DG [1] and 30.52% compared to OACS [2] respectively.

5.2 Impact of error rate

Error rate is measured on the basis of the difference between the users' in different data grid sequences to be synchronized with the number of users synchronized.

$$ER = \sum_{i=1}^{n} User_i - No\ of\ users\ synchronized \quad (11)$$

From (11), the error rate 'ER' is measured based on the number of users in different grid sequences '$User_i$'. When the error rate is lower, the method is said to be more efficient and it is measured in terms of megabytes (MG).

Table-2. Tabulation for error rate.

No. of users	Error rate (MB)		
	MDCS-FT	EDHR-DG	OACS
20	23.1	33.9	41.5
40	38.7	46.3	53.8
60	49.6	57.2	66.7
80	55.8	63.4	72.9
100	69.3	77.1	85.6
120	78.4	86.2	94.7
140	82.3	90.1	99.6

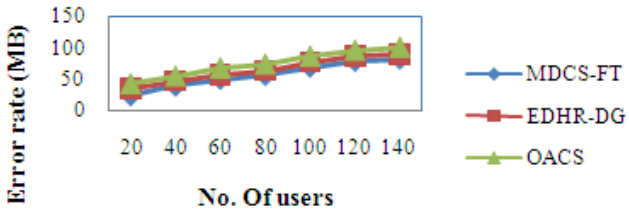

Figure-8. Measure of error rate.

The targeting results of error rate using MCDS-FT mechanism with two state-of-the-art methods [1], [2] in Table-2 presented for comparison based on the number of users for measuring fault tolerance in cloud environment. From Figure-8, it is evident that the error rate is reduced using the proposed MDCS-FT mechanism. The Clock Synchronization for different users with optimized data grid sequences results in the reduced error rate in MDCS-FT mechanism. With the application of Clock Synchronization 'n' clocks are said to be synchronized with 'n' users, resulting in the minimization of error rate. At the same time, in MDCS-FT mechanism, the users periodically resynchronize clock values due to the non-zero drift values of all clocks the efficient separation of single event detection to multiple event detection is made in an efficient and therefore minimizing the error rate. With the resynchronized clock values, an efficient global time is maintained and sends the results of resynchronized clock values between successive grid sequences. MCDS-FT mechanism reduces the error rate by 18.00% compared to EDHR-DG [1] and 35.58% compared to OACS [2] respectively.

5.3 Impact of transmission delay

The transmission delay is the time taken to transmit the transaction block with respect to the data packets ready for transmission.

$$TD = \sum_{i=1}^{n} DP_i * Time\ (DP_i) \quad (12)$$

From (12), the transmission delay 'TD' is measured using the number of data packets ready for transmission 'DP_i' and time for each transaction block '$Time\ (TB_i)$' respectively. Lower the transmission delay, more efficient the method is said to be.

Table-3. Tabulation for transmission delay.

Data Packets (DP)	Transmission delay (ms)		
	MDCS-FT	EDHR-DG	OACS
5	13.1	14.09	16.3
10	21.8	23.11	26.11
15	35.6	37.09	40.12
20	41.3	43.06	46.16
25	59.4	61.07	64.17
30	68.3	71.06	74.15
35	74.2	76.05	79.14

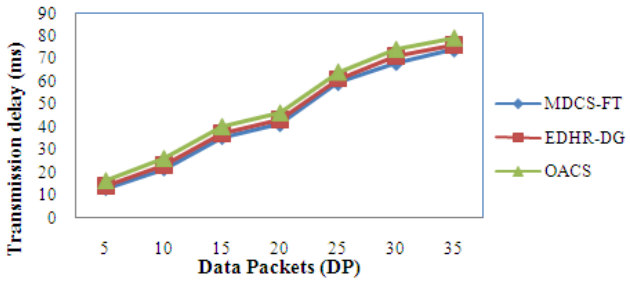

Figure-9. Measure of transmission delay with respect to data packets.

As listed in Table-3, MDCS-FT mechanism measures the transmission delay on cloud zone to secure data objects replication in cloud data grid with respect to data packets. It is measured in terms of milliseconds (ms). Figure-9 presents the variation of transmission delay with respect to data packets to measure fault tolerant dynamic replication. All the results provided in Figure-9 confirm that the proposed MDCS-FT mechanism significantly outperforms the other two methods, EDHR-DG [1] and OACS [2]. The transmission delay is reduced in the MDCS-FT mechanism using the Berkeley Synchronized algorithm. With the application of Berkeley Synchronized algorithm, execute a time daemon process that is responsible for delivering the transaction block to other users. Followed by this, a master slave block is established using the Berkeley Synchronized value. This master slave block in MDCS-FT mechanism in turn reduces the transmission delay by 4.47% compared to EDHR-DG [1]. As a result, secure data objects replication is performed in cloud data grid by removing the faulty nodes in the cloud zone in the MDCS-FT mechanism using the threshold value. The synchronized value greater than the threshold are disregarded as faulty nodes. This efficient segregation of fault and non-faulty nodes in MDCS-FT mechanism helps in reducing the transmission delay by 13.13% compared to OACS [2].

6. CONCLUSIONS

In this paper, MerkleDamgard Clock Synchronized based Fault Tolerant (MDCS-FT) mechanism is provided based on the Clock Synchronized with Berkeley algorithm for efficient cloud service improves the scalability and fault tolerance rate in computational data grid. As the mechanism uses MerkleDamgard Hash Replication algorithm in a dynamic manner, it improves the scalability through efficient delivery of transaction block using hash function. Finally, the Berkeley Synchronized algorithm is designed that improved the fault tolerance rate in a significant manner. Different data packets with varied transaction block sizes on computational data grid in cloud environment analyze the faulty and non-faulty nodes. A series of simulation results are performed to test the scalability, error rate, transmission delay and fault tolerance rate and therefore to measure the effectiveness of MCDS-FT mechanismwith an improvement of scalability by 26.90% and reduces the error rate by 26.79% compared to state of the art works.

REFERENCES

[1] NajmeMansouria, Gholam Hosein Dastghaibyfard, "Enhanced Dynamic Hierarchical Replication and Weighted Scheduling Strategy in Data Grid," Journal Parallel Distributed Computing, Elsevier Journal, 2013

[2] Bo Dong, Qinghua Zheng, Feng Tian, Kuo-Ming Chao, RuiMaa, Rachid Anane, "An optimized approach for storing and accessing small files on cloud storage," Journal of Network and Computer Applications, Elsevier Journal, 2012

[3] Flavio Lombardi, Roberto Di Pietro, "Secure virtualization for cloud computing", Elsevier, Journal of Network and Computer Applications, volume 34, issue 4, july 2011, pp. 1113–1122.

[4] Nazanin Saadat, Amir MasoudRahmani, "PDDRA: A new pre-fetching based dynamic data replication algorithm in data grids", Elsevier, Future Generation Computer Systems, volume 28, issue 4, April 2012, pp. 666-681.

[5] Zhuo Lu, Wenye Wang, and Cliff Wang, "Camouflage Traffic: Minimizing Message Delay for Smart Grid Applications under Jamming", IEEE

transactions on dependable and secure computing, volume 12, issue 1, January/February 2015, pp. 31-44.

[6] Taeho Jung, Xiang-Yang Li, and Meng Wan, "Collusion-Tolerable Privacy-Preserving Sum and Product Calculation without Secure Channel", IEEE transactions on dependable and secure computing, volume 12, issue 1, January/February 2015, pp. 45-57.

[7] Dong Yuan, Xiao Liu, and Yun Yang, "Dynamic On-the-Fly Minimum Cost Benchmarking for Storing Generated Scientific Datasets in the Cloud", IEEE transactions on computers, volume 64, issue 10, October 2015, pp. 2781-2795.

[8] Chien-An Chen, Myounggyu Won, RaduStoleru, and Geoffrey G. Xie, "Energy-Efficient Fault-Tolerant Data Storage and Processing in Mobile Cloud", IEEE transactions on cloud computing, volume 3, issue 1, January/march 2015, pp. 28-41.

[9] SmithaSundareswaran, Anna C. Squicciarini, and Dan Lin, "Ensuring Distributed Accountability for Data Sharing in the Cloud", IEEE transactions on dependable and secure computing. Vol. 9, issue 4, July/august 2015, pp. 556-568.

[10] VivekNallur, Rami Bahsoon, "A Decentralized Self-Adaptation Mechanism For Service-Based Applications in The Cloud", IEEE transactions on software engineering, volume 39, issue 5, may 2013, pp. 591-612.

[11] JasmaBalasangameshwara, NedunchezhianRaju, "A hybrid policy for fault tolerant load balancing in grid computing environments", Journal of Network and Computer Applications, Elsevier Journal, volume 35, 2012, pp. 412-422.

[12] Marcela Castro-León, Hugo Meyer, Dolores Rexachs, Emilio Luque, "Fault tolerance at system level based on RADIC architecture", Journal of Parallel and Distributed Computing, Elsevier Journal, volume 86, 2015, pp. 98-111.

[13] J. W. Larsona, M. Heglanda, B. Hardinga, S. Robertsa, L. Stalsa, A. P. Rendellb, P. Strazdinsb, M. M. Alib, C. Kowitzd, R. Nobesc, J. Southernc, N. Wilsonc, M. Lic, Y. Oishic, "Fault-Tolerant Grid-Based Solvers: Combining Concepts from Sparse Grids and MapReduce",International Conference on Computational Science, ICCS 2013, volume 18, pp. 30-139.

[14] BoubakerBoulekrouchea, Nafaâ Jabeurb, Zaia Alimazighia, "An Intelligent ETL Grid-Based Solution to Enable Spatial Data Warehouse Deployment in Cyber Physical System Context", The 12th International Conference on Mobile Systems and Pervasive Computing, Elsevier journal, volume 56, 2015, pp. 111-118.