VOL. 12, NO. 23, DECEMBER 2017 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences
©2006-2017 Asian Research Publishing Network (ARPN). All rights reserved.

www.arpnjournals.com

AN EFFECTIVE FAULT TOLERANCE METHOD FOR COLLABORATIVE EDITING WITH FICKLE OPERATIONS

G. Sekar.[1] and V. Vasanthraj[2]
[1]PG and Research Department of Computer Science, Dr. Ambedkar Govt. Arts College, Vyasarpadi, Chennai, India
[2]Dr. Ambedkar Govt. Arts College, Vyasarpadi, Chennai, India
E-Mail: gsekarg@yahoo.co.in

ABSTRACT

Collaborative editing refers to the editing groups which produce work products as the collection of individual contribution. We present an implementation model on how to increase fault tolerance for collaborative editing systems with fickle operations. Some of the recent research revealed that collaborative editing systems were constructed by the Conflict free Replicated Data Types (CRDT). This new approach is shown to avoid the fault on the every user's replicas updates that should not affect the owner's document. Every user can update their own document with some updates. At the end, all the updates were transferred to the owner's original document. In some case some of the user's replica can update with some mismatch updates that can also be reflected on the owner's document. So these updates badly reduce the reliability and integrity of the collaborative systems. The mismatch updates and the faults cause the whole document lead to lessen its integrity and quality. In this paper, we carefully analyze, find the mismatch updates and the replica's faults towards this type of systems and reduce the fault tolerance. We define algorithms to find such fault mismatch and remove that fault replica. Then we produce the original document without any fault updates.

Keywords: fault tolerance, fickle operations, distributed systems, editing systems.

INTRODUCTION

A distributed system [1] is a model in which components located on networked computers communicate and coordinate their actions by passing messages. The components interact with each other in order to achieve a common goal. A collaborative editor is a form of collaborative software application that allows several people to edit a computer file using different computers; a practice is called collaborative editing [2]. A collaborative editing system used to facilitate the multiple users can edit a document over the Internet.

The different users can update the stored shared document that relies on centralized server. The well known Wikipedia is the best example for the most large scale collaborative editing system. Here every user can update some changes towards the document. That time it's not assured to give fully original same updates [3]. In some of the times the user can update with mismatch data with compared to others updates. These all updates with mismatch data can transfer to the owner's main document that causes the fail of integrity and consistency. Every collaborative editing system has to manage the integrity and consistencies over the shared document. Conflict updates affect the editing system if it contains mismatch and wrong updates. Every local copy of the shared document is maintained by the every replica using some commutative or fickle operations [4].

Most of the research in collaborative editing system has focused on these sequence or independent operations to prove the stable reliability over the shared document [5]. In every update on the local copies has a relation over the concurrent operations and original updates. This relationship achieves the state convergence between the different replicas updates. In the centralized server approach, mismatching updates are identified and detected and compared with shared local copies updates. Many of the researches over collaborative editing systems prove the stability of convergence manage space and time complexities and simultaneity and locking properties.

Every malicious attack over the collaborative editing systems can be identified after the update was made by the replica. It is not so easy to identify and detect the replica as malicious at initial by server. Every replica is doing concurrently some update over the local copies. The owner document can reflect by all these local copies updates. The fault updates and mismatch updates over on the original document can lead into inconsistencies on the centralized place [6]. But the important challenge on collaborative editing system is to prove the stable of convergence and integrity. A conflict-free replicated data type (CRDT) [7] is a data structure which can be replicated across multiple computers in a network, where the replicas can be updated independently and concurrently without coordination between the replicas [8], and where it is always mathematically possible to resolve inconsistencies which might result. Concurrent updates to multiple replicas of the same data, without coordination between the computers hosting the replicas, can result in inconsistencies between the replicas, which in the general case may not be resolvable. Restoring consistency and data integrity when there are conflicts between updates may require some or all of the updates to be entirely or partially dropped. A possible approach is optimistic replication, where all concurrent updates are allowed to go through and the results are merged or "resolved" later, with consistency between the replicas eventually re-established. While optimistic replication might not work in the general case, it turns out that there is a significant and practically useful class of data structures, CRDTs, where it does work and mathematically always possible to merge or resolve concurrent updates on the data structure without conflicts. This makes CRDTs ideal for optimistic replication [9].

In this paper, we present algorithms to find the mismatch updates and faulty replica while the replica is doing concurrent updates and remove that replica from the group. In this way, we make the efficient fault tolerance for collaborative systems to provide the more reliable data with improved integrity. The mismatch update is done by some replica those updates were find and removed by the server with the help of algorithms. When the replica is started to do update, time server can detect the update to check whether the update is right or not. The primary challenge is to handle an update done by the replicas to find the malicious updates. We mitigate such threats and ensure eventually consistency of the shared document. We also give the maximum throughput of the shared document without any mismatch updates.

BACKGROUND AND RELATED WORK

A. Fault tolerance

Fault tolerance is the property that enables a system to continue its operation in the event of failure of some of its components. Fault tolerance is particularly sought in high availability or life critical systems. A **fault** in a system is some deviation from the expected behavior of the system: a malfunction. Faults may be due to a variety of factors, including hardware failure, software bugs, operator (user) error, and network problems.

Faults can be classified into one of three categories [1]:

- **Transient faults**

These occur once and then disappear. For example, a network message doesn't reach its destination but does when the message is retransmitted.

- **Intermittent faults**

Intermittent faults are characterized by a fault occurring, then vanishing again, then reoccurring, then vanishing. These can be the most annoying of component faults. A loose connection is an example of this kind of fault.

- **Permanent faults**

This type of failure is persistent (ie) it continues to exist until the faulty component is repaired or replaced. Examples of this fault are disk head crashes, software bugs, and burnt-out power supplies.

Approaches to faults

We can try to design systems that minimize the presence of faults. **Fault avoidance** is a process where we go through design and validation steps to ensure that the system avoids being faulty in the first place. This can include formal validation, code inspection, testing, and using robust hardware. Fault removal is an *ex post facto* approach where faults were encountered in the system and we managed to remove those faults. This could have been done through testing, debugging, and verification as well as replacing failed components with better ones, adding heat sinks to fix thermal dissipation problems, etc.

Fault tolerance is the realization that we will always have faults (or the potential for faults) in our system and that we have to design the system in such a way that it will be tolerant of those faults. That is, the system should compensate for the faults and continue to function [10].

B. Fickle or commutative operations

In mathematics, an operation is commutative if the order of the numbers used can be altered with the result remaining the same. For example, addition and multiplication are commutative operations. Operational transformation was proposed to facilitate the convergence of the states of different replicas by transforming conflicting operations into commutative operations [11].

Fickle is another name of commutative operations. The state based CRDTs are called Convergent Replicated Data Types, where the states are merged by a function which must by commutative, associative and idempotent. The merge function provides a join for any pair of replica states, so the set of all states forms a semi lattice. The update function must monotonically increase the internal state, according to the same partial order rules as the semi lattice.

C. Collaborative editing systems

Collaborative editing is the editing of groups producing works together through individual contributions. Effective choices in group awareness, participation, and coordination are critical to successful collaborative writing outcomes. Collaborative editing systems were worked by the principle of CRDT is abbreviated of Conflict Replicated Data Types. However the recent researches over the collaborative editing systems many of the algorithms have been found like Logoot [12], Treedoc [13] and WOOT [14].

In the Logoot algorithm, it uses the scalable optimistic replication for collaborative editing on the P2P networks. According to the WOOT (With out Operational Transformation) algorithm is the framework that ensure the consistency without operational transform on the group editors community. A real time group editor has used these frameworks for the editing. The Treedoc is also a mechanism for the commutative replicated data type systems.

In the author of WOOT algorithm, proposed to include the immutable identifiers for the position of inserted or deleted, instead of using the index to the position, which could be affected if another user has submitted an insert or delete operation for an earlier position. Likewise the Logoot algorithm uses a simpler data structure for unique identification; however, it relies on a casually ordered reliable multicast service for the algorithm to work, which could limit its use in practical systems. A tree data structure is used to generate totally ordered unique identifiers for each element in the shared document by according to the Treedoc algorithm.

Many of the researches provide mechanism for the collaborative editing but however the real challenge for every mechanism to ensure the consistency and integrity [15]. This is the primary challenge for every Collaborative editing system using operational transformations. The Operational Transformations are simply denoted as the OT in the editing systems [16]. OT has a reputation of being hard to understand. This reputation has to rise even faster now to that Google has published the Drive Realtime API, which is based on OT and let's third party apps use the same collaboration as Google Docs.

WORKING OF PROPOSED ALGORITHM

In this paper, there are three algorithms - server update, replica update and Write main updates. Each algorithm has separate process to determine some conditions satisfied.

Algorithm1: Server Update

```
1.  function SERVER (nRep, id,stp)
2.     /* nRep-No Of Replicas,id-Replica Id,tstp-TimeStamp*/
3.     if nRep=Available then
4.        for i= 1 to nRep do
5.           tsp=EndTime-StartTime
6.           ReceiveUpdate from AllReplica Data,id,TimeStamp
7.           WritOn MainDocument With DataandRepId
8.           /*Copy These Updates on Main Document*/
9.        end for
10.    end if
11. end function
```

Algorithm 2: Replica Update

```
1.  function RepData (nRep, id,stp)
2.     /* nRep-No Of Replicas,id-Replica Id,tstp-TimeStamp*/
3.     for i=1 to nRep do
4.        if nRepUpdate= Mismatch then
5.           Dispose the Replica
6.           /*Removes the Faulty Replica*/
7.           Exit the Group
8.        else
9.           Send Update to Server
10.          /*Send Correct Updation Only*/
11.          Write Correct Update
12.          /*Write All Replicas Update*/
13.       end if
14.    end for
15. end function
```

Algorithm 3: Write Main Updates

```
1.  function RepData (data, id,stp)
2.     /* data – Replicas Data,id-Replica Id,tstp-TimeStamp*/
3.     for i=1 to nRep do
4.        Sort All Replica's Updation(data,id,tstp)
5.        Write Update(data,Id)
6.        /*Write Only Data and Replica id*/
7.        Write Update(data,ReplaceAll("[^a-zA-Z_ ]"," "))
8.        /*Contains Only Data, No Special Char Allowed*/
9.     end for
10. end function
```

All these algorithms perform with some conditions regarding server available and replica updates. The algorithms tell the process of the server and the replicas update over the group of editing system. It is the main process to find faulty replica and remove from the group. Each process did the updates with perfect match with other replica updates. The algorithm check every update whether it is right or not with other update. Three algorithms are namely as Server Update, Replica Update and Write main update. Lets we see these algorithm process in detail.

A. Server update

The function of the server update algorithm tells about the server process that means the check of every replica is available or not. When the entire replica is ready to do the update on the local copies, this algorithm did the availability or not at the time. The replica start time and end time has been calculated in the nanoseconds to transfer the updates to the main document. The exact time is taken by the subtraction of the start time in the end time in nanoseconds. After the checking the availability, the server take the response of update transform to the main document if and only if it is only right match update.

B. Replica update

The second algorithm said about the every replica update with respect to the server actions. Here we have to check the updates are right or not. In every replica update, the replica update algorithm checks that replica's update is mismatch or correct match. Sometimes the replica can did any mismatch update on the local copies. Those mismatch updates were found and removed and also that replica is also removed from the group. So it saves the entire document from failure and integrity faults. Then the server gets suitable updates from the respective replicas and writes over the main document. This is applicable for all replicas update.

C. Write main updates

In this part, updates of our all replicas contain rep id, timestamp and data. But we want to extract only data from the all the updates, however all the updates were arrived and stored at the local copies but the server wants only suitable update not at every replica updates. This algorithm did all the updates were sort and transfer to the main document. Here rep id and timestamp are stored at local copies only for their future responses. Only the data from the all the replica updates were write on the main document. Here we cannot see any special characters on our updates. Our whole update contains only pure data. This is done by this algorithm for every replica update.

The working of every algorithm is so efficient and simple, since they do not require any complex data types and data structures.

D. Finding threats

Threat is meant by in our updates wrong data, mismatch data, unauthorized access, document loss [17], unavailability like this etc,. Our three algorithms have supported to find these threats and remove from our process. Threat detection is much important in the collaborative editing. Threat causes the whole document leads to fail. Threat analyzes and removes process in collaborative editing systems lead the good editing system and good integrity. More threat and mismatch updates lead to document into the inconsistencies and contains wrong updates [18]. Lets we see what are the threats available and the ways to treat them as follows.

MISMATCH UPDATES AND THREATS

A. Unauthorized access

The unauthorized access is one of the threats in the collaborative editing systems. When the replica is identified by as unauthorized by the owner, it is immediately removed by the server. Because when the inaccessible replica can have a chance to update the main document it doesn't know to do what and when. This will lead to inconsistencies and faulty updates. When the replica is found as the unauthorized of the group our algorithm is used to check the access and their updates. The update is mismatch or wrong that will automatically remove from the group.

B. Mismatch updates and invalid data

The mismatch is meant by when one replica can update some data in the main document as totally different as others update then it is considered as a mismatch update. In some cases for example four replicas is allowed to update some data on the main document but the third one did the different than other replica update. Then it will reflect in the main document as different update. So the main document is not valid due to the mismatch updates. In another case, every replica can do with same data type update for some update but there may one replica did invalid data type update on the document cause the inconsistencies and unexpected loss. So the validation of every updates can be verified as true update and good consistencies over the main document.

C. Malicious attack on the replica's update

If some attacks over the replica during the update process, the replica will fail to give the update. It's the malicious access or attack over the group communication. When one replica can be accessed by the unauthorized it would as block listed in the general of collaborative editing systems [19]. In our mechanism when the replica has ready to start the update every nanoseconds of the update will check and drop by the algorithm. If there any malicious updates and attacks over the replica, it would be removed at initially then it will no longer to access the main document. When the malicious user has given any mismatch update to the main document, it will spoil the document as faulty. So the every replica is identified at initial update where made at the group of editing. A list of blocked or malicious replicas where identified by the server that they cannot have permission to access the main document or any other further updates over the main document.

These are some threats in the collaborative editing systems. The threat analysis and threats should be removed in the well manner on the collaborative system to prevent the inconsistencies. Our process mechanism is also identified and analyzed the malicious access and updates. The above threats are presented in the every collaborative editing system and some of the recent researches give some possible solutions to maintain the integrity and consistency.

RESULTS AND DISCUSSIONS

The figure (Figure-1) shows that four (R1, R2, R3, R4) replica has a local copy updates. All these updates are transferred to the main document. Here the every replica can do some update with respect to the correct updates. Each replica can have a local copy to maintain their updates. All the updates are matched correctly then the updates are transferred to the main document. In every update the replica can have own copy for their future reference.

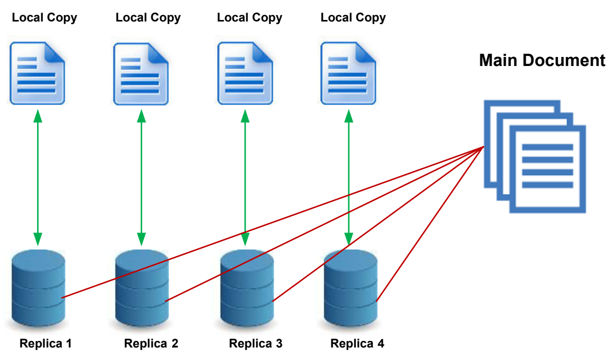

Figure-1. Collaborative editing using multiple replicas.

If all the updates are ready to arrive at the main document the main copy has sort all the updates. If the update is in mismatch or wrong that replica is identified and removed at the group. Then it doesn't access no longer on the main document. When the replica has command to start up their update the server is identified that update is right or wrong. When the wrong updates are made by the replica and then it is removed from the group. In the diagram the main document has owner replica that is the response for all the replica updates. When R1, R2, R3, R4 has given some updates to the main document the server has to identify firstly check all the replica is available or not. To avoid the confusion for multiple replica the main document replica is absent at the Figure-1. If the unauthorized access and malicious updates were found from R1, R2, R3 and R4 by the server update algorithm, it is identified and removed at the initial stage. Further it has no access to do update over the main document. When all the replicas were started to do their updates the start and end time is calculated in nanoseconds. It is the timestamp of every replica. Timestamp is very important to consider sorting every updates. After the timestamp is calculated the server update algorithm correspondingly receives the updates from the all replicas. In this update has all the data along their id and timestamp in nanoseconds. Then all the updates were verified as the perfect update, it is transferred to the main document, since it should have perfect updates. When the mismatch and malicious updates were found at the replica update the server immediately removed from the group. The malicious access and the updates at the while replica update the updates are ignored by the main document. So the original updates were sort and transfer to the main document.

In the write on document algorithm is used to check the update has no special characters and rep id. It has only the update of the replica data only. It is done by the command of <ReplaceAll ("[^a-zA-Z_]"," ")>. It is used to determine the number of special characters allowed in the main document. The write main update algorithm is the response to write the whole update on the main document. It has initially timestamp, id and data. But later the document is filtered only replica data only. The replica id and the timestamp has stored at only in the replica local copies only. In our mechanism all the updates transferred by one by one at the replicas, finally every replica updates were sort and stored at the main copy. Every timestamp is calculated at the nanoseconds that are partially calculated as start time and end time. The start time is meant by the when the replica is started and the end time is denoted as the update is done at the local copy. The throughput of the every process in the replica update is denoted by some process. The total elapsed time is calculated in the nanoseconds. A graph is drawn for the replica update and the document throughput. Document throughput is defined by how the document has only pure update without any faults on the minimum time manner. We sort all the updates from the replica updates. Here all replicas can give the updates to the main document but however which one gives correct update in the minimum time units is considered. We sort all their updates according to the timestamps only. The timestamp is taken in the nano seconds to calculate accurate update time.

Normally, the throughput should be maximum for every main document. It is based on the replica's performances over the main document. In our mechanism, we take six replicas for the replica update graph. Six replicas namely R1, R2, R3, R4, R5, and R6 are considered. The arrival time of the six replicas has been taken in the nanoseconds (NS). Separately the update time of the R1, R2, R3, R4, R5 and R6 timestamps has been taken in the nanoseconds. Here we can draw the line graph for the six replicas update and arrival performances.

Table-1. Replica update time.

Replicas	**Arrival time** (in Ns)	**Update time** (in Ns)
R1	1048	1800
R2	2068	2800
R3	2880	3180
R4	3148	3520
R5	3600	3848
R6	5042	5640

The table (Table-1) shows that the arrival time and the update time of the six replicas. The graph has been plotted according to the nano second values. All the replicas can arrive in the different time slots and make the updates over the main document. Here if there any maximum time interval between the arrival and the update, that replica is ignored by the main document. There is the some sequence time intervals all the replicas did the update over the main document. Here there are two lines are plotted, one for the arrival time and another one for the update time. The entire replica produces the update in the time increasing flow because here the no more mismatch updates and the malicious attacks on the replicas. If there any some attacks and faults the line has been fall from high to low manner.

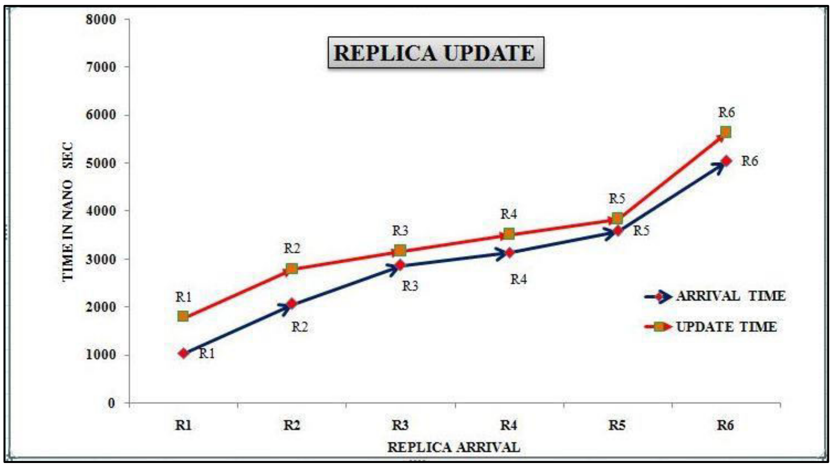

Figure-2. Replicas update line chart.

Using this replica update graph (Figure-2), the throughput of the document has been calculated in the milliseconds. The timestamp of nano seconds is converted into milliseconds. In our mechanism the throughput is

VOL. 12, NO. 23, DECEMBER 2017 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences
©2006-2017 Asian Research Publishing Network (ARPN). All rights reserved.

www.arpnjournals.com

defined as the fraction of the update time and arrival time in milliseconds. First of all, the conversion of the nanoseconds to millisecond is 1 nano seconds=1e-6. The method of the throughput is defined as:

$$\text{Throughput} = \frac{\text{Update time in Nano Sec}}{\text{Arrival Time in Nano Sec}} \text{ X } 1000000 \text{ ms}$$

In the above calculation, we can utilize our document throughput in milliseconds. The throughput chart of the six replicas update over the main document has been plotted below. The throughput of the main document is used to determine that which replica has more perfect update over the main document. For example, R1 arrived as early and give immediate update than others. So R1 has the maximum throughput than other replicas. The main document of the throughput is plotted in Figure-3.

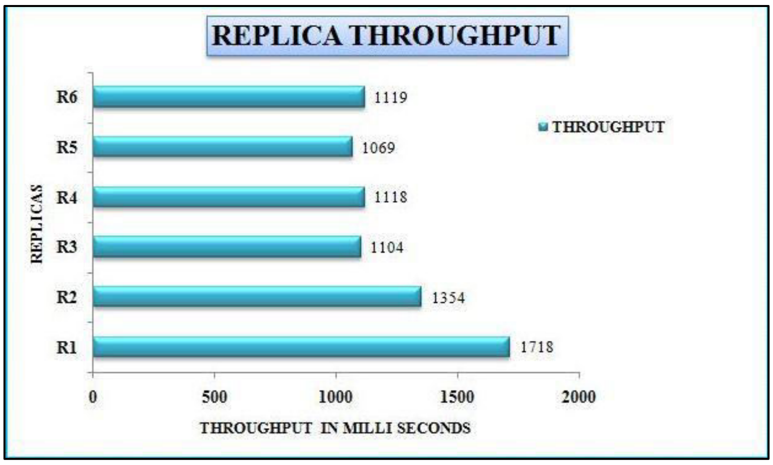

Figure-3. Replica throughput.

So R1 has the highest throughput on the main document and has perfect updates over the main document. Other replica has some delay and causes some fault update and recovers later. Let's compare R3, R4, R5, R6 with its sequence arrival and update; but they are delayed with compared to the R1. So they are same level of throughput with less than the R1. Thus the way of the multiple replica update through this method. In some case of the any malicious attack and fault update that also can reflect in the graph. It is also identified in the line which falls from high to low. If there is no more error that replica will give the high throughput.

CONCLUSIONS

The recent researches have proposed about the collaborative editing system to participate with multiple operations on same time with many mechanisms. However all the research show that the importance and concept of integrity and consistencies. Out work also showed that the reliability and integrity of the editing system is maintained in high order. We can also consider the removal of the replica in the group of editing which is vulnerable to attacks and threats. The CRBT based system has revolutionized by the many recent researches and new mechanisms. We proposed three basic algorithms to find and evaluate the mismatch update and perfect update over the main document. Here we can also prove that which replica can give the high and low throughput over the main document. The throughput between the main and local copies has been found and analyzed by the line charts. Our mechanism has proved to detect and removal the malicious and mismatch update over the multiple replica update. Here we can satisfy our mechanism by finding and removing malicious attacks, saving perfect updates and most importantly we can maintain the strong integrity and more reliable performance over the main document. Thus the approach of algorithm to maintain consistency and reliability on the fickle operations at Collaborative editing systems has been implemented successfully. There are many researches is going on the group of editing systems. As a future enhancement, we can consider the protection of collaborative editing system from other left over threats and challenges. We can also implement the new mechanism to fast the update with accuracy. We can do the session management for each replica to do automatic update over the main copies. We can also define one garbage collector to correct the mismatch update and recover the updates from the fails. We also implement the firewall booster over the each replica to avoid the unexpected crashes. The new encryption and decryption method for separately for every replica to do session update between local and main copies. New hardware approaches and mechanism will arrive for the strong fault tolerance at this type of editing system at soon. These will lead to the ongoing research over the distributed computing environment. The advanced future enhancement on the Collaborative Editing is ongoing research on the computer networking sites. They will arrive at the future on the editing systems platforms.

REFERENCES

[1] Tanenbaum Andrew S. 1995. Distributed Operating Systems. Englewood Cliffs, N.J: Prentice Hall. ISBN 0-13-219908-4.

[2] Brenes J. A., López G. & Guerrero L. A. 2017. Development and evaluation of augmented object prototypes for notifications in collaborative writing environments. In Advances in Human Factors and System Interactions (pp. 301-312). Springer International Publishing.

[3] Sila Ozen Guclu, Tanir Ozcelebi, Johan Lukkien. 2016. Distributed Fault Detection in Smart Spaces Based on Trust Management. In: Proceedings of the 2016 Elsevier Science Direct 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016). pp. 66-73.

[4] Dunne J., Jiang M. Z., Shao H. & Xue Z. Y. 2016. U.S. Patent No. 9, 471, 897. Washington, DC: U.S. Patent and Trademark Office.

[5] Zibin Zheng, Michel R. Lyu. 2015. Selecting an Optimal Fault Tolerance Strategy for Reliable Service-Oriented Systems with Local and Global Constraints. In: Proceedings of the 2015 IEEE Transactions on Computers. 64: 219-232.

[6] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. 2007. Zyzzyva: Speculative byzantine fault tolerance. In: Proceedings of 21st ACM Symposium on Operating Systems Principles.

[7] https://en.wikipedia.org/wiki/Conflict-free_replicated_data_type

[8] Adrian W. T., Nalepa G. J. & Ligęza A. 2016. Usefulness of Inconsistency in Collaborative Knowledge Authoring in Semantic Wiki. In Knowledge, Information and Creativity Support Systems: Recent Trends, Advances and Solutions (pp. 13-25). Springer International Publishing.

[9] Lv X., He F., Cai W. & Cheng Y. 2016, May. An efficient collaborative editing algorithm supporting string-based operations. In: Computer Supported Cooperative Work in Design (CSCWD), 2016 IEEE 20th International Conference on (pp. 45-50). IEEE.

[10] G. Oster, P. Urso, P. Molli and A. Imine. 2005. Proving correctness of trans-formation functions in collaborative editing systems. INRIA, Rapport de recherche RR-5795.

[11] Cheng Y., He F., Wu Y. & Zhang D. 2016. Meta-operation conflict resolution for human–human interaction in collaborative feature-based CAD systems. Cluster Computing. 19(1): 237-253.

[12] Wenbing Zhao, Mamdouh Babi, William Yang, Xiong Luo, Yueqin Zhu, Jack Yang, Chaomin Luo, Mary Yang. 2016. Byzantine fault tolerance for collaborative editing with commutative operations. In: proceeding of the of the IEEE International Conference on Services Computing. IEEE. pp. 0246-0251.

[13] S. Weiss, P. Urso and P. Molli. 2009. Logoot: a scalable optimistic replication algorithm for collaborative editing on p2p networks. In: Proceedings of the 29th IEEE International Conference on Distributed Computing Systems. IEEE. pp. 404-412.

[14] N. Preguica, J. M. Marques, M. Shapiro and M. Letia. 2009. A commutative replicated data type for cooperative editing. In: Distributed Computing Systems, 2009. ICDCS'09. 29th IEEE International Conference on. IEEE. pp. 395-403.

[15] Lie Chen, Wei Zhou. Byzantine Fault Tolerance with Window Mechanism for Replicated Services. In: Proceedings of the 2015 fifth IEEE International Conference on Instrumentation and Measurement, Computer, Communication and Control (IMCCC). pp. 1255-1258.

[16] P A Alsberg, J D. Day. 1976. A principle for resilient sharing of distributed resources. Proceedings of the 2nd international conference on Software engineering. pp. 562-570.

[17] Wenbing Zhao, Mamdouh Babi, William Yang, Xiong Luo, Yueqin Zhu. 2016. Enable Concurrent Byzantine Fault Tolerance Computing with Software Transactional Memory. in Proceedings of the 2016 IEEE International Conference on Electro Information Technology (EIT). pp. 675-720.

[18] M A Naseen, Amal Ganesh, Sunitha C. 2016. A Study on Byzantine Fault Tolerance Methods in Distributed Networks. In: Proceedings of 2016 Elsevier Science Direct Fourth International Conference on Recent Trends in Computer Science & Engineering Elsevier Science Direct. pp. 50-54.

[19] Rong Ding, Xiaoguang Li, Tongyu Zhu, Zhenhan Zhong, Jufu Zhang. 2011. Research on Intelligent Fault Location for Multistage in Large-scale Distributed Measurement Environment. In: Proceedings of the 2011 Elsevier on Advanced in Control Engineering and Information Science. pp. 2403-2407.