A fault tolerant approach for reaching consensus in a distributed system

Anirban Roy
Computer Science and Engineering
NIT Durgapur
Durgapur, India
anirbanroy88@gmail.com

Nishad T M
science and Humanities
National College of Engineering
Tirunelveli, India
searchofme@gmail.com

Sangeetha. K. G
Computer Science and Engineering
National College of Engineering
Tirunelveli, India
yavanchitra@gmail.com

R.Muthamil Selvi
Computer Science and Engineering
National College of Engineering
Tirunelveli, India
rmg.selvi@gmail.com

Arpan Mondal
Electronics and Communication
NIT Durgapur
Durgapur, India
arpamondal.nitdgp@gmail.com

Abstract—**This paper addresses solution to the problem of reaching an agreement (consensus) in a distributed system. The aim is to maximise fault tolerance as well as reduce the message exchange overhead. The proposed algorithm uses as few as two rounds of message exchange. This better efficiency is achieved through early disposal of faulty processes. The network partitioning scheme proposed later further reduces message exchange overhead and thus network traffic**

***Keywords*— Consensus, Byzantine Agreement, Reaching Agreement, Early Stopping, Decision Vector**

I. Introduction

In a distributed system it is often required to reach a common agreement (consensus) among the processes[6].The participating process must co-operate among themselves for such an agreement. This operation becomes all the more difficult when the behaviour of the processes become unpredictable.

A number of solutions[1],[2] has been reported for reaching this agreement with the target maximising fault tolerance and reducing message exchange. However optimality is yet to be achieved considering both these clauses. The Quick consensus algorithm described in [11] addresses a viable solution to this problem but with some assumptions. The proposed scheme is set to achieve better efficiency through early disposal of faulty processes in exactly two rounds of message exchange. The algorithm ensures minimum message exchange considering all kinds of faults[4] [6].The algorithm applies a upper bound on the number of faulty process as $n \geq 2t$ where n is the number of processes and t is the number of faulty processes.

II. Previous Work

<u>The Byzantine Agreement Problem</u>

In Byzantine agreement, one process (designated as initiator) broadcasts a value that is to be agreed upon. All processes then communicate with each other through message exchanges. All non-faulty processes finally agree on the same value. If the initiator is non-faulty then all the non faulty processes agree on the value of the initiator.

The algorithm proposed in [1] solves the Byzantine agreement problem for n processes but with the upper bound of $n \geq 3t+1$ where t is the maximum number of tolerable faulty processes .It is found in [1] and [2] that any conventational Byzantine agreement algorithm would require t+1 rounds of message exchange in the worst case.

The Synchronous Mortal Byzantine Tolerant Consensus (SMBTC) approach [3] improves fault tolerance by $n \geq 2t$ but with the assumption that the faulty processes must crash within a finite time. A system of 5 processes($P_1, P_2, ..., P_5$) out of which P_3 and P_5 are faulty has been cited as an example. Processor P_1 initiates the procedure by sending its decision value to all other processes. Then it passes through certain phases with each phase consisting of two rounds.

In the first round of a phase, each process sends its proposed and decision values to all. If any non-faulty process (P_2) identifies that the message from a process (P_5) is lost, the non-faulty process (P_2) detects it (P_5) as faulty. In the second round, each process sends the records it is having, at the end of first round, to all others.

A non-faulty process checks the received information from processes considered non-faulty and tries to decide.

If a faulty process (P_3) is not yet crashed (i.e. to be detected) and sends conflicting values, the non-faulty process (P_2) cannot decide and a new phase is started.

The agreement process ends when all the non-faulty processes decide on the same value.

The Quick Consensus Algorithm [11] reaches an agreement in two rounds of message exchange. Each process has an initial decision value. In the first round of message exchange each process sends its decision value to every other process. The faulty processes may behave maliciously by sending different decision value to different processes. Thus at the end of first round every process site has a decision vector.

In the second round of message exchange each process sends the decision vector formed in round one to every other process. It is assumed that the faulty processes would not behave maliciously in the second round and must send the correct decision vector to each process. At the end of round two each process site has a $n \times n$ matrix which it uses to calculate the decision value.

This algorithm is optimal in terms of message exchange overhead but not completely fault tolerant. The algorithm shows high degree of fault tolerance in round one but not in round two. The solution proposed in the current work reaches an agreement through early disposal of faulty processes as well as partitioning of the network while it tries to reach an agreement. Further, we have considered a broad fault model which maximises the fault tolerance of our algorithm. The algorithm is best suited for a star topology although topologies and communication delays do not affect the decision making procedure [5].

III. PROPOSED SCHEME

In a network with large number of processes the message exchange overhead is huge. Due to presence of faulty processes several rounds of message exchange[2]are required to ensure that the decision taken by the non-faulty processes are correct. An early identification of the non faulty processes can quicken the decision making procedure .Our proposed scheme computes the consensus value in exactly two rounds of message exchange. The algorithm takes care of all kinds of fault models[6][7].A network partitioning scheme is introduced with a aim of further reducing the number of message exchanges. It reduces network congestion in a large distributed system. The following subsection reports the proposed scheme.

*A. **Consensus Reaching Procedure in a group of processes***

The proposed scheme uses exactly two rounds of message exchange. Every process site must possess a initial decision value (0/1) before participating in the consensus procedure. A system of five processes (P_1,P_2,..P_5) is considered among which P_2 and P_5 is assumed faulty.

Round 1: All process site exchange their initial values.

Round 2: At the beginning of round two each process P_i is aware of the initial value of every other process. A vector V_i is stored at every process site to store these values. The vector V_i stored at each process site is shown in Fig.1(a)

Each process P_i then sends its vectors (V_i) to every other process. Thus at the end of round two every process posses a matrix (n×n array where n is the number of participating processes)

Four different situations may arise depending upon the malicious behaviour of the faulty processes. The four different situations are discussed below:

i) If none of the processors are faulty the matrix received at every processor site is the same. Such a situation is shown in Fig.2(a).The values in every column of the matrix is identical. A majority function is applied to every row to compute the decision value of each process.

ii) The faulty processes behave maliciously in round 1 but not in round two. Process P_5 has been assumed to have crashed. The 2-D matrix is identical at every processor site and has been shown in Fig.2(b)

iii) The faulty processes behave maliciously in round 2 but not in round one. The 2-D formed at different process site are shown in Fig.3

iv) The faulty processes behave maliciously in both rounds of message exchange. The 2-D matrix formed at different process sites are shown in Fig.4

The solution to reach a consensus among the processes is described next. The following assumptions are taken while devising the solution.

(1)The entire network is a fully connected one. The links among the processes/nodes are reliable and do not introduce any delay.

(2)A dormant faulty process can send different values (0/1) to different processes in round one. It can distort the vector formed in round one before sending it in round two except its own decision value. d is the total number of dormant faulty process.

(3) c is the total number of crash faulty process. Such a fault occurs when a process site crashes permanently.

(4) f= d + c is the total number of faulty processes.

Algorithm 1:

(i) PL- List of process ids of participating processes maintained at each process maintained at each process site

(ii)V_i - Initial vector received after round one at each process site P_i

(iii)M_i- n×n matrix formed after round two at each process site P_i

(iv)D_i- Decision vector at each process site P_i

Input: processes ,initial value and PL for each process

Output: Decision Value

```
1. First Round: Each process sends its initial value to all other processes.
            If  a process $P_i$ does not receive any message from a
                process $P_j$
            then  update its own vector $V_i[j]$=C
2. Second Round: Each process $P_i$ sends its own vector $V_i$ formed after round two to every other process.
            If  a process $P_i$ does not  receive any vector from a
                process $P_j$
            then update the matrix $M_i$ as :for i=1 to n
                  { M[j][i]=C}
3. Construction of decision vector D at each process site P:
    For  i=1 to n
       decision value, k=M[i][i]
      If  column i contains the value C or does not contain a
          majority of the value ,k
      then the corresponding row and column of the matrix M
              are cancelled out and update D[i]=X
      else
         column i in matrix M contains a majority of the value k.
         We update D[i]=decision value, k
       A sub list is created at D[i] consisting of process ids of those processes which have not received the actual decision value of process $P_i$. Such an arrangement for Fig.4(c) is shown in Fig.5(a).Process $P_5$ is cancelled out as the fifth column in matrix $M_3$ does not contain a majority of 0 which is the original decision of processor $P_5$.
    end for
```

4. Computation of decision value

The vector D obtained in step 3 is processed in a descending order of the length of the sub-list attached to every process site.

```
For each process site selected(process site with a sublist0
    If the column i in the matrix M does not have all
       identical values
    then Update D[i]=X
         Cancel out the corresponding row and column in
         matrix M
         Cancel i from the sub-lists of all other processes
         where ever it appears(The above procedure is
         shown in Fig.5(b).Process $P_2$ is cancelled out as
         the length of sub list at D[2] was the longest
    else (the column i in matrix M contains all identical
    values)
         cancel the processes that appear in the sub-list of D[i]
         update vector D and matrix M accordingly(The
         above procedure is shown in Fig.5(c).
end for
```

If a process does not have a sub list in the vector D it is considered to be non faulty process. Finally a majority function is applied to the decision matrix to compute the final decision value.

1	0	1	0	1

V_1

1	0	1	0	1

V_2

1	1	1	0	1

V_3

1	0	1	0	0

V_4

1	1	1	0	0

V_5

Fig. 1 Decision Vector V at different process sites formed after round 1

1	0	1	0	0
1	0	1	0	0
1	0	1	0	0
1	0	1	0	0
1	0	1	0	0

Fig. 2(a) 2-D matrix at each process site when none of the processes are faulty

1	0	1	0	C
0	0	0	0	C
1	1	1	0	C
1	0	1	0	C
C	C	C	C	C

Fig.2(b) 2-D matrix at every process site when P_2(dormant) and P_5(crash) are faulty

1	0	1	0	0
0	0	1	0	1
1	0	1	0	0
1	0	1	0	0
1	1	0	1	0

1	0	1	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	0	0
0	1	0	1	0

1	0	1	0	0
1	0	0	0	1
1	0	1	0	0
1	0	1	0	0
1	1	0	1	0

1	0	1	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	0	0
1	1	1	0	0

1	0	1	0	0
0	0	1	1	1
1	0	1	0	0
1	0	1	0	0
0	1	1	0	0

Fig. 3 2-D Matrix M at different process sites(P_1 to P_5) when process P_2 and P_5 behaves maliciously in round 2 only

1	0	1	0	1
1	1	0	0	0
1	1	1	0	1
1	0	1	0	0
1	0	1	0	0

1	0	1	0	1
0	0	0	0	1
1	1	1	0	1
1	0	1	0	0
0	1	0	1	0

1	0	1	0	1
1	0	0	1	1
1	1	1	0	1
1	0	1	0	0
1	1	0	1	0

1	0	1	0	1
0	0	1	0	0
1	1	1	0	1
1	0	1	0	0
0	1	1	0	0

1	0	1	0	1
0	0	1	0	1
1	1	1	0	1
1	0	1	0	0
1	1	0	0	0

Fig. 4 2-D Matrix M at different process sites(P_1 to P_5) when process P2 and P5 behave maliciously in in both rounds of message exchange

1	0	1	0	1
0	0	0	0	1
1	1	1	0	1
1	0	1	0	0
0	1	0	1	0

Matrix M_3 after cancelling out process P_5

1	0	1	0	X
	1	2		
	3			

Matrix D_3 along with the sub-lists constructed from Matrix M_3
Fig. 5(a)

1	0	1	0	1
0	0	0	0	1
1	1	1	0	1
1	0	1	0	0
0	1	0	1	0

Matrix M_3 after cancelling out process P_2

1	X	1	0	X
		2		

Vector D_3 after cancelling process P_2
Fig. 5(b)

1	0	1	0	1
0	0	0	0	1
1	1	1	0	1
1	0	1	0	0
0	1	0	1	0

1	X	1	0	X

Fig. 5(c) Final matrix M_3 and vector D_3

B. Network Partitioning Scheme

Message exchange overhead can be further reduced using a network partitioning scheme. It is particularly applicable for large networks where the network traffic is huge. A network consisting of n processors is partitioned in g number of groups. A particular processor (L_i) is selected as the co-ordinator of each group G_i[11].The co-ordinator of each group uses algorithm 1 to compute the decision value of that particular group. Once the local rounds for all the groups are completed, the co-ordinator from each group participates in a global decision making round. Before the commencement of the global round, each processor must possess a value of the form (d_i,w) where d_i is the decision value of the that group and w is the number of non-faulty processors in that group.

At the initialization phase of network partitioning, a randomly selected process L_i (co-ordinator) broadcasts an initialization message[8][9]. After receiving it, each process of the system initializes a counter to '1' and starts incrementing. The L_i then further broadcasts g tokens. Each token can be received by one and only one process. The processors holding the tokens are the leaders. Each leader logically forms a group of n/g-1 processes.

Algorithm 2:

```
Input: leaders of the groups and their weighted local
decision-value (d,w).
Output: global-decision-value.
1. If for any I, $1 < i < g$ (g is the number of groups),
local round for group $G_i$ is finished leader L of $G_i$ sets a
random timer RT[L] and starts decrementing it
2. If L doesn't receive any advertisement from an
initiator of global round and RT[L] = 0
then L sends local-decision d and weight (i.e. no.
of non-faulty processes in $G_i$) to all the leaders,
else go to step 2
3. If a leader $Q \in G_i$ receives initialization message
from L then Q resets RT[Q]=0
4. L initiates Algorithm 1 (quick-consensus) considering
the set of leaders as a group and decision
value computed is the global-decision-value
5. the leader L of each group conveys global-decision
value to all processes belonging to its group G
6. return global-decision-value
```

In global round (Algorithm 2), if weighted local decision-value is sent by the faulty leader of group G, there may be a chance of mishap. However, the proposed scheme can mask off such faults as the faulty leader is detected in the local round. Once the faulty leader is detected, a new leader is selected from G following a cellular automaton based election algorithm reported in [10]. The new leader participates in global round by sending a reply to the initiation message [11].

IV. ANALYTICAL AND SIMULATION RESULTS

This section reports performance evaluation of the proposed scheme in terms of number of message exchanges to reach a consensus. The analytical results are shown in the following sub-sections.

A. *Analytical Results*

The number of message exchanges required in the proposed scheme (Algorithm 1 and 2) is

$$m = \frac{2n(n-1)}{g} + 2g(g-1) + (n-g-f)$$

where the 1st(2n(n-1)/g) term and the 2nd (2g(g-1)) term represents the number of message exchanges required in local and global rounds respectively. The 3rd (n-g-f) represents the number of message exchanges required by the leaders of each group to inform the global decision to all the non-faulty processes belonging to that group.

B. *Simulation Results*

The performance of the proposed scheme is compared with the SMBTC [3] and agreement-at-partition [12] in terms of message exchange overhead while reaching an agreement. Table I compares the message exchanges required by the SMBTC and the proposed scheme without network partitioning. The first column shows the number of participating processes. The second column represents the number of allowable faulty processes. To take the random behaviour of faulty processes into account, three sets of observations have been taken for different crash times. This is shown in column three. Column four shows the number of message exchanges and column five shows their average. Finally column five represents the number of message exchanges in the proposed scheme. Fig 6 shows the comparison between these two schemes.

Table II compares the performance of the proposed scheme (global-agreement) and agreement-at-partition. The observations are recorded for a fixed number of 50 processes partitioned into different number of groups. The first column shows the number of groups. The second column shows the number of allowable faulty processes. The third column shows the number of message exchanges for different crash times of the faulty processes. The fifth column shows the average number of message exchanges and the last column represents the proposed algorithm with partitioning scheme.

The results shown in the tables indicate that the proposed algorithm shows maximum fault tolerance while achieving optimality in terms of message exchanges. Both SMBTC and agreement-at-partition take arbitrary number of message exchanges to reach an agreement. The proposed

TABLE I

PERFORMANCE EVALUATION OF PROPOSED SCHEME AND SMBTC

No. of Process	No. of faulty process	**SMBTC [3]**			**Prop-osed Schm**
		Obs. Time for diff. Crash times	No.of Msg.	Avg. Msg.	No. Of Msg.
15	4	1	1800	2100	420
		2	2250		
		3	2250		
20	6	1	4000	3467	760
		2	1600		
		3	4800		
25	12	1	7500	6667	1200
		2	6250		
		3	6250		
50	24	1	65000	58333	4900
		2	65000		
		3	45000		

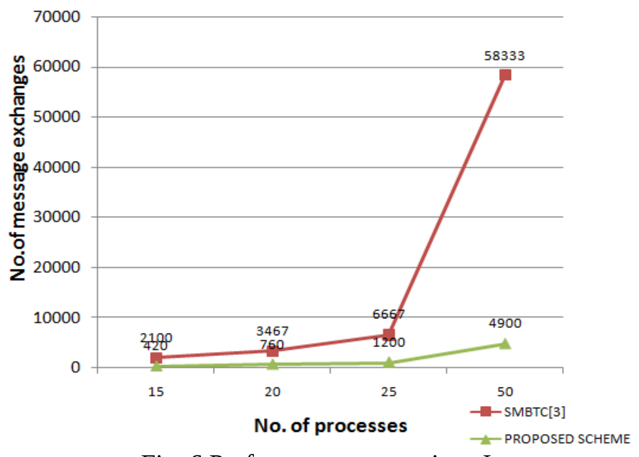

Fig. 6 Performance comparison I

TABLE II

PERFORMANCE COMPARISON BETWEEN PROPOSED SCHEME AND AGREEMENT -AT -PARTITION

No. of Groups	No. Of faulty processes	**SCHEME [12]**		**Proposed Scheme**
		No. of message exchanges	Avg. Msg.	
4	16	4768	4562	1279
		4376		
		4512		
5	16	2650	2817	1040
		3100		
		2700		
10	16	1400	1300	694
		1300		
		1200		

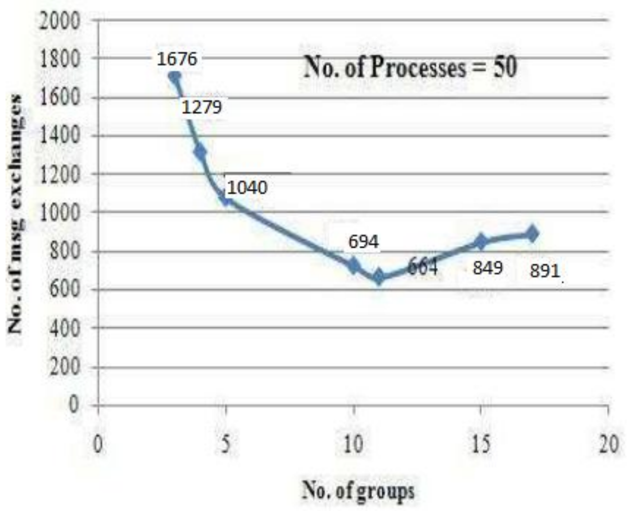

Fig. 7 Performance Comparison II

scheme on the other hand reaches consensus in exactly two rounds of message exchange. The proposed algorithm overcomes the assumptions of the Quick Consensus [13]. Fig. 7 shows the impact of partitioning on the proposed scheme. We can conclude that that the number of message exchanges reduces with the increase in the number of partitions but only up to a threshold limit.

V. Conclusions

This work addresses the issue of reaching an agreement in a distributed system. An early disposal of faulty processes/processors makes the decision making processes so quick. A network partitioning scheme reduces message exchange overhead further. The algorithm is highly fault tolerant. It can be of great use in any real time distributed system. Thus it can be concluded that two rounds of message exchange is sufficient to reach an agreement in a fully connected link fallible network when the total number of process failure is less than or equal to (N-1)/2 where ‘N’ is the total number of processors in the given network. This is the experimental result which got through the simulation of the program and is supported by the theoretical formula.

.

References

[1] Lamport L.,Pease M.,Reaching Agreement in Presence of Faults', Journal of ACM,Vol.27 pp.228-234,Apr,1980

[2] A.W.Krings, T.Feyer 'The Byzantine Agreement Problem: Optimal Early Stopping' in Proceedings of the 32nd Hawaii International Conference on System Science,1999

[3] J.Widder,G.Gridling,B.Weiss,'Synchronous Consensus with Mortal Byzantine' in Proceedings of the 37th Annual IEEE/IFP International Conference on Dependable Systems and Networks

[4] P.Lincoln, J.Rushby 'A Formally Verified Algorithm for Interactive Consistency Under a Hybrid Fault Model' in Fault-Tolerant Computing Symposium FICS 23,Toulouse,France June 1993

[5] O.Cihan and M.Akar 'Effect of Bounded Delay on Convergence Speed of Distributed Consensus Algorithms' in 2009 IEEE International Conference on Control and Automation Christchurch, New Zealand, December 2009

[6] S.Ghosh 'Distributed Systems An Algorithmic Approach' , Chapman and Hall/CRC, 2006

[7] N.A.Lynch 'Distributed Algorithms' ,Morgan Kaufmann Inc. , 1996

[8] M.Singhal, N.G. Shivaratri 'Advanced concepts in operating systems: distributed, database, and multiprocessor operating systems', Tata McGraw-Hill,2001

[9] Raymond K.,'A Tree Based Algorithm for Distributed Mutual Exclusion', ACM Trans.Computer Systems,1989

[10] Kalyan Mahata, Meghnath Saha, and Sukanta Das, ‘Cellular Automata Based Coordinator Selection Scheme in Distributed System’, accepted in CSC’09, USA, 2009.

[11] M. Dalui, B Chakraborty, B. K. Sikdar, ‘Quick Consensus through early disposal of faulty processes’, Submitted in Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics San Antonio,TX, USA - October 2009

[12] M.Dalui, B Chakraborty, B.K.Sikdar, ‘An Efficient Scheme for Quick Consensus In Partitioned Adhoc-Network’, Submitted in 34th IEEE Conference on Local Computer Networks, Switzerland, 2009