A 'liub($\log_2(N)$)-2' Resilient Decentralized Commit Protocol

Shyan-Ming Yuan, Member IEEE
National Chiao Tung University, Hsinchu

Key Words — **Distributed database system, commit protocol, decentralized algorithm, resilience, hypercube, message complexity**

Reader Aids —
General purpose: Present a new commit protocol
Special math needed for explanations: None
Special math needed to use results: None
Results useful to: Distributed-database designers

Summary & Conclusions — **In distributed database systems, commit protocols are used to ensure the transaction atomicity. In the presence of failures, nonblocking commit protocols can guarantee the transaction atomicity without blocking the transaction execution. A (resilient) decentralized nonblocking commit protocol (RDCP) is proposed for distributed database systems. This protocol is based on the hypercube network topology and is 'liub($\log_2(N)$)−2' resilient to node failures (N = number of system-nodes). The number of messages sent among the N nodes is $O(N\cdot\log_2^2(N))$ which is only a factor of $\log_2(N)$ over the message complexity lower bound $O(N\cdot\log_2(N))$ of decentralized commit protocols. Furthermore, RDCP is an optimistic nonblocking protocol. It aborts the transaction only when some nodes want to abort or some nodes fail before they make local decisions.**

1. INTRODUCTION

In a distributed database system, maintenance of database consistency usually requires a transaction to be atomic, *viz*, all actions of a transaction successfully complete or none of them does. *Commit* protocols ensure this behavior, and are needed to maintain the logical atomicity of a transaction.

Several commit protocols have been proposed [1 - 7], of which the simplest is 2-phase commit protocol in which [1] one node which starts the protocol is *coordinator* and others the are *slaves*. In phase 1, upon receiving a transaction request from a user, the coordinator sends a *start* message to all slaves, and waits for responses from them. If all slaves and the coordinator agree to commit the transaction then the coordinator sends a *commit* message to all the slaves; otherwise it sends an *abort* message. Since there is 1 coordinator in the 2-phase commit protocol, it is *centralized* protocol.

A decentralized implementation of the 2-phase commit protocol requires that every node, at each phase, communicate with all other nodes in the system. Thus, $N\cdot(N-1)$ messages are sent among the N nodes. Lakshman & Agrawala [4] proposed a decentralized commit protocol that requires $O(N^{1.5})$ messages. Their solution is based on finite projective planes [8], and requires 2 rounds of message exchanges. Yuan & Agrawala [5] proposed a family of decentralized commit protocols that requires $O(k\cdot N^{1+1/k})$ messages in k rounds of message exchanges. In these algorithms, there is no central coordinator and all nodes are considered to be coordinators. They all execute an identical program. This type of protocol is *decentralized* and is characterized by successive rounds of message interchanges.

Although the algorithms [4,5] have smaller message complexities than the decentralized 2-phase commit protocol (DTPCP), both of them and the DTPCP guarantee the transaction atomicity by blocking the execution of transactions in the presence of failures. Once some nodes fail, all other nodes have to block the transaction execution until failed nodes recover. Because a blocked transaction can hold some shared resources which are needed by other transactions, the resource availability is appreciably affected. Therefore, commit protocols which do not block transactions in the presence of failures are desirable. The first nonblocking commit protocol was introduced by Skeen [6]. It is 3-phase commit protocol (3PCP) which extends the centralized 2-phase commit protocol. The major advantage of nonblocking commit protocols is that the resource availability is not affected by failures. The decentralized version of 3PCP requires twice the messages of DTPCP. Similarly, in [4,4], the messages needed for the nonblocking versions are twice the blocking ones.

Wolfson & Segall [14] discussed the communication complexity of blocking & nonblocking centralized commit protocols, and proposed a Tree-commit protocol which performs better than other centralized commit protocols. Ramarao [7] proposed distributed commit protocols for ring & tree networks with message complexity of $O(N)$. However, these protocols are not truly decentralized because the initiator node (for ring) and the root (for tree) know the final decision first then the final decision is propagated to other nodes. Yuan & Agrawala [15] extended their decentralized commit protocol [5] to tolerate some special cases of multiple node failures.

This paper presents a truly decentralized nonblocking commit protocol with $O(N\cdot\log_2^2(N))$ message complexity. It guarantees the transaction atomicity without blocking the execution of a transaction in the presence of all kinds of multiple node failures as long as the number of failed nodes is not greater than 'liub($\log_2(N)$) −2'. The protocol is based on the hypercube network topology in [8]. Ref [13] showed that the lower bound of message complexity for any truly decentralized protocol is $O(k\cdot N^{1+1/k})$ for k rounds of message exchanges. Since the lower bound of message complexity for $\log_2(N)$ round decentralized protocol is $O(N\cdot\log_2(N))$, the proposed commit protocol is sub-optimal with respect to the message complexity because it is based on the hypercube and requires $k=\log_2(N)$ rounds of message exchanges.

Section 2 describes the system model. Section 3 summarizes the hypercube architecture. Section 4 describes the communication scheme used by the new protocol. The communication scheme

considers the nodes of a distributed system as vertices in a hypercube. Section 5 presents the 'liub($\log_2(N)$)-2' resilient decentralized commit protocol (RDCP). All proofs are in the appendix. Standard notation is given in "Information for Readers & Authors" at the rear of each issue.

Acronyms

CPS communication partner set
FSA finite state automaton
RDCP resilient decentralized commit protocol

2. SYSTEM MODEL

Assumptions

1. The distributed system consists of a set of autonomous processing nodes connected by a hypercube network which provides error free, in sequence, and guaranteed message delivery.
2. The message delays are bounded. It is impossible for any system to maintain consistency even under 1 node-failure if the message delay is unbounded [10].
3. A node detects a *timeout* when the messages it is waiting for are guaranteed not to arrive because of node failures.
4. Nodes fail by stopping [11], *ie*, a node does not receive or send any message after it fails.
5. Local transition is atomic whether in the absence or presence of failures.
6. Each node has an internal timer; message delays between any two nodes are bounded.
7. The underlying communication protocol can detect the node failure and append the *timeout* message to the normal message such that all receivers of the failed node receive a *timeout* message. ◄

The model used to formalize commit protocols and node failures in this paper is based on the Skeen model [12] wherein the execution of a commit protocol at each node is modeled as a FSA. Each vertex in the FSA represents a local state. Local state transition of each FSA consists of receiving and/or sending of zero or more messages and is represented by a labeled edge. Each label contains a numerator and a denominator where the numerator represents the received messages and the denominator represents the sending messages during the state transition. This implies that nodes have the capability to send any number of messages in each local state transition. Each FSA has 2 final states: *commit & abort*.

If a commit protocol is correct, then either all FSA reach their commit states or all reach their abort states. When no failure occurs, the final decision is commit if all nodes want to commit. If at least one node wants to abort, then the final decision is abort. If some nodes fail before announcing their local decisions (yes or no) then the final decision is abort. Otherwise, if all failed nodes have announced their local decisions, then their local decisions are used for the final decision.

Example 1

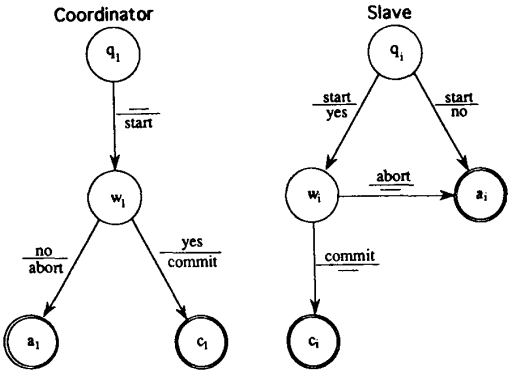

Figure 1. FSA of 2-Phase Commit Protocol

Figure 1 shows the FSA of the 2-phase commit protocol in a 2-node system. The FSA to express the commit protocol has been simplified. In figure 1, the coordinator proceeds to state a_1 if it receives at least one *no* message, and to state c_1 if it receives $N-1$ *yes* messages. ◄

In general, a node failure can be detected by the absence of an anticipated message. Therefore, when the timer expires, the node has *timed out* and can take appropriate actions such as declaring the failure of the anticipated sender. In [12], it is modeled by a *timeout* message that is received like any other normal message and can cause a state transition.

Node failures are modeled by a *failure transition*. A failure transition begins at the failed node on the instance of node failure and ends when the failed node recovers. Since the local state transition is atomic in the presence of failures, a failure cannot occur in the middle of a transition and interrupt the sending of messages. Thus, a failure transition is defined as reading all outstanding messages and sending a *timeout* message to all desired receivers. Although, in real systems, local state transitions are not atomic and nodes can fail after sending only some of the associated messages, [12] argued that by allowing a failure transition to send the normal messages followed by the *timeout* messages is sufficient to model the behavior of nonatomic state transitions; see assumption 7. Therefore, a real nonatomic local state transition can be modeled as an atomic action.

3. HYPERCUBE TOPOLOGY

Notation

HD(u,v) Hamming distance between any 2 binary sequences, u & v: number of positions where the bit values of u & v differ.

An m-dimensional hypercube can be considered as an undirected graph containing $k=2^m$ vertices labeled from 0 to 2^m-1 in such a way that there exists an edge between any 2

vertices A & B iff the Hamming distance between the two binary representations of A & B is 1.

An important property of the m-dimensional hypercube is that it can be constructed recursively from lower dimensional hypercubes. An m-dimensional hypercube can be derived from 2 $(m-1)$-dimensional hypercubes by connecting every vertex of the first $(m-1)$-dimensional hypercube to the corresponding vertex of the second $(m-1)$-dimensional hypercube. The labels of all vertices x, $0 \le x \le 2^{m-1}-1$ in the first $(m-1)$-dimensional hypercube are unchanged and the labels of all vertices y, $0 \le y \le 2^{m-1}-1$ in the second $(m-1)$-dimensional hypercube are renamed to $y+2^{m-1}$.

The following properties of the hypercubes [9] are necessary to establish correctness of the RDCP.

Theorem 3.1 The node connectivity of an m-dimensional hypercube is m.

Theorem 3.2 The diameter of an m-dimensional hypercube is m.

Theorem 3.3 There exist at least $m-1$ disjoint paths whose lengths are at most m between any 2 nodes in an m-dimensional hypercube.

Theorem 3.4 There exist m disjoint paths whose lengths are at most $m+1$ between any 2 nodes in an m-dimensional hypercube. ◄

4. COMMUNICATION STRUCTURE

Notation

S_j^i	CPS for node j in round i
N	number of physical nodes
k	liub($\log_2(N)$): number of rounds
M	2^k: number of logical nodes.

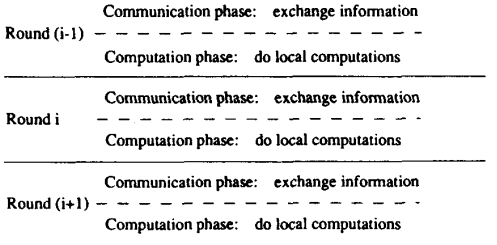

Figure 2. General Structure of Rounds for Decentralized Algorithms

The operation of a decentralized protocol requires that each node exchange information and do computations. A general structure of a decentralized algorithm is defined in terms of rounds. Each round consists of communication & computation phases as in figure 2. The computations at each node in round i are based on the information available at that node up to and including the communication phase of round i. In a protocol, if a node j exchanges information with a set of nodes S_j, then S_j is the CPS of node j. In general the CPS for a node j can be different in every round.

The complete specification of message exchanges for a protocol is defined by:

S_j^i, for all $i \in [1, k]$ and $j \in [0, N-1]$.

Ref [4, 5] used the finite projective planes and the k-dimensional array as basic structures to define the CPS of each node in every round of message exchanges. Here, the CPS of each node is defined according to the hypercube network topology. Choose $M = 2^k$; then add $M-N$ virtual nodes into the system. In practice, randomly select $M-N$ physical nodes to act as 2 logical nodes; then,

$$\tfrac{1}{2}M = 2^{k-1} < N \le 2^k = M.$$

Thus, the number of virtual nodes needed is always less than N. In the worst case, where $N=2^{k-1}+1$, $N-2$ virtual nodes are needed. Let the logical M-node system be a k-dimensional hypercube:

$S_X^i = \{Y | \mathrm{HD}(X,Y) = 1\}$, for all logical nodes $X \in [0, M-1]$, and all rounds $i \in [1, k]$.

Since the S_X^i are the same for all $i \in [1, k]$, let S_X represent the set of logical nodes with which logical node X exchanges information in all rounds. In other words, every logical node X exchanges messages with all logical nodes which are adjacent to X in the corresponding logical k-dimensional hypercube at each round of message exchange.

Example 2

Consider a 7-node system ($N=7$). Then $k = 3 = \mathrm{liub}(\log_2(7))$, and $M = 2^3 = 8$. Add $8 - 7 = 1$ virtual node into the system to form a logical 3-dimensional hypercube. Without loss of generality, let physical node 0 act as 2 logical nodes: 0 & 7. Since $M=8$ and $k=3$,

$S_0 = \{1,2,4\}$, $S_1 = \{0,3,5\}$,

$S_2 = \{0,3,6\}$, $S_3 = \{1,2,7\}$,

$S_4 = \{0,5,6\}$, $S_5 = \{1,4,7\}$,

$S_6 = \{2,4,7\}$, $S_7 = \{3,5,6\}$.

It is easy to check that the local information of any logical node can reach all other logical nodes after $k=3$ rounds of message exchanges and there exist at least $k-1$ disjoint paths whose lengths are at most k between any 2 logical nodes; *eg*, the local information of logical node 1 can reach,

logical nodes 0, 3, 5 after round 1,

logical nodes 2, 4, 7 after round 2,

logical node 6 after round 3.

There exist 2 disjoint paths with length ≤ 3 between nodes 0 & 3:

0-1-3 and 0-2-3. ◄

Lemma 4.1 For an M-node system, let every node exchange information with all nodes in its CPS set (see beginning of section 4). The local information of all nodes can reach all other nodes after k rounds. ◄

Lemma 4.2 For an M-node system, if every node exchanges information with all nodes in its CPS set (see beginning of section 4) by removing any $k-2$ nodes from the system, lemma 4.1 is still true among operational nodes. ◄

Lemma 4.3 The total number of messages sent among M nodes in k rounds of message exchanges, according to the CPS set (defined in beginning of section 4), is $M \cdot \log_2^2(M)$. ◄

5. RESILIENT COMMIT PROTOCOL

This section describes RDCP. Since the communication structure described in section 4 is used in this discussion, the following phrases are defined:

send messages: messages are sent to all logical nodes in S_X for node X

receive all messages: the receipt of messages from all logical nodes in S_X

receive a message: the receipt of a message from one of the logical nodes in S_X for X.

5.1 Description of RDCP

The FSA for the RDCP is shown in figure 3. The actions in each state for an M-node system are listed here.

- State a_k (abort): On receiving a *recovery*(Z) message, send an *abort* message to node Z.
- State c (commit): On receiving a *recovery*(Z) message, send a *commit* message to node Z.
- State r (recovery): Send *recovery*(X) messages and move to state $r-w$, where X is the address of the recovered node.
- State $r-w$ (recovery-waiting): On receiving a *commit* message, move to state c. On receiving an *abort* message, move to state a_k.
- State q: Upon receiving a transaction, unilaterally decide to commit or abort the transaction. If abort is decided, send *no* messages and move to state a_1. If commit is decided, send yes^1 messages and move to state w_1. For virtual nodes, commit is always decided because any abort decision leads to the abortion of the whole transaction. If failed, send *timeout* messages and move to state a_k on recovery.
- State w_1: If all yes^1 messages are received, send yes^2 messages and move to state w_2. If a *no* message or a *timeout* message is received, send *no* messages and move to state a_2. If failed, send *timeout* messages and move to state r on recovery.
- State w_i, $i \in [2, k-1]$: If all received messages are either yes^i or *timeout*, send yes^{i+1} messages and move to state w_{i+1}. If a *no* message is received, send *no* messages and move to state a_{i+1}. If failed, send *timeout* messages and move to state r on recovery.
- State w_k: If all received messages are either yes^k or *timeout*, move to state c. If a *no* message is received, move to state a_k. If failed, move to state r on recovery.
- State a_i, $i \in [1, k-1]$: Send *no* messages and move to state a_{i+1}. If failed, send *timeout* messages and move to state a_k on recovery.

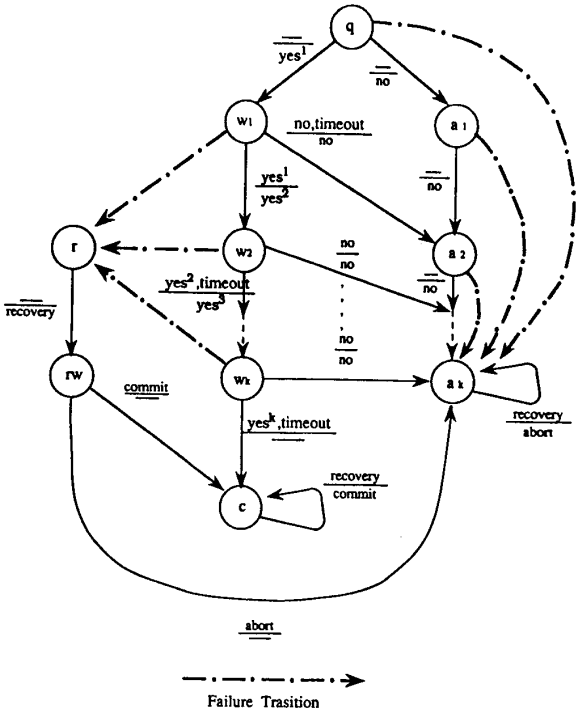

Figure 3. FSA for the RDCP Protocol

5.2 Properties of RDCP

Definition 5.1 A protocol is p-resilient to node failures if it works correctly even in the presence of at most p node failures. ◄

Theorems 5.1 - 5.3 establish the correctness of the RDCP in the absence of node failures.

Lemma 5.1 If a node is in one of the a states, it eventually aborts.

Lemma 5.2 If a node decides to abort, all nodes eventually abort.

Lemma 5.3 If no nodes decide to abort, all nodes eventually commit. ◄

Theorem 5.1 In the absence of node failures, the RDCP ensures that all nodes are either all aborted or all committed. ◄

Lemma 5.4 RDCP is $(k-2)$-resilient to node failures if some nodes fail in state q.

Lemma 5.5 RDCP is $(k-2)$-resilient to node failures if no nodes fail in state q. ◄

Theorem 5.2 RDCP is $(k-2)$-resilient to node failures.

If N is not a power of 2 and some virtual nodes are added to form an M-node logical system, then for $N=2^{k-1}+1$ (the worst case), $N-2$ virtual nodes are needed. Since each physical node at most acts as 2 logical nodes, each physical node failure can lead to at most 2 logical node failures. Therefore, the resilience of the RDCP is at least $\frac{1}{2}k-1$. Thus, theorem 5-2 can be modified to:

Corollary 5.1 For an N-node system where N is not a power of 2, the RDCP is p-resilient to node failures, $p \in [\frac{1}{2}k-1, k-2]$, $k = \text{liub}(\log_2(N))$. ◄

Theorem 5.3 For an N-node system, the total number of messages required for RDCP is $O(N \cdot \log_2^2(N))$. ◄

Since the resilience of RDCP is based on the existence of at least $k-1$ node disjoint paths of length $\leq k$ between any two nodes in a k-dimensional hypercube, if the number of rounds can be increased to $k+1$, from theorem 3-4, there exist k node disjoint paths of length $\leq k+1$ between any two nodes. RDCP with $k+1$ rounds is $(k-1)$-resilient when $N=2^k$.

APPENDIX

A.1 Proof of Lemma 4.1

Since every node X, $0 \leq X \leq M-1$, exchanges information with all nodes in S_X for all rounds and S_X is defined according to the neighboring nodes of X in the k-dimensional hypercube, from theorem 3-2, the local information of node X reaches all other nodes after k rounds. *Q.E.D.*

A.2 Proof of Lemma 4.2

Since there exist at least $k-1$ node disjoint paths whose lengths are at most k between any 2 nodes in a k-dimensional hypercube (from theorem 3.3), removing any $k-2$ nodes can only destroy at most $k-2$ paths between any two nodes. Therefore, there still exists at least one path whose length is at most k between any two nodes. Thus, by removing any $k-2$ nodes in the system, the remaining nodes can still propagate their local information to all other nodes after k rounds of message exchanges. *Q.E.D.*

A.3 Proof of Lemma 4.3

Since the node degree of a k-dimensional hypercube is k and there are k rounds of message exchanges, the total number of messages sent among the M nodes is $M \cdot k^2 = M \cdot \log_2^2(M)$. *Q.E.D.*

A.4 Proof of Lemma 5.1

If a node is in one of the a states, then it can perform only the transition of sending *no* messages and move to the next a state. Thus, eventually, the node moves to state a_k. *Q.E.D.*

A.5 Proof of Lemma 5.2

Let node X decide to abort the transaction. Since each node sends its local decision to nodes which are adjacent to itself in the corresponding hypercube in every round, nodes in S_X should receive a *no* message from node X in round 1 and send *no* messages in round 2. Because the diameter of a k-dimensional hypercube is $k=\log_2(M)$ after k rounds, all other nodes should receive at least one *no* message and move to one of the a states. From lemma 5-1, all nodes abort the transaction eventually. *Q.E.D.*

A.6 Proof of Lemma 5.3

If no nodes want to abort the transaction, they send yes^1 messages in round 1 and move to state w_1. Since all messages are eventually delivered, each node receives all yes^1 messages, send yes^2 messages, and move to state w_2 in round 2, *etc.* Therefore, eventually each node receives all yes^k messages and moves to state c. *Q.E.D.*

A.7 Proof of Theorem 5.1

The proof follows from lemmas 5.2 & 5.3. *Q.E.D.*

A.8 Proof of Lemma 5.4

Let node X fail in state q. From the actions defined in states w_1 & a_1, nodes in S_X receive a *timeout* message due to the failure of node X. Thus, they send *no* messages in round 2 and move to state a_2. Because there exist at least $k-1$ disjoint paths whose lengths are at most k between any two nodes in the k-dimensional hypercube, if there are no more than $k-3$ nodes other than node X which can fail, then every operational node receives at least one *no* message within k rounds and eventually aborts the transaction.

For the failed nodes, they recover to state a_k directly if they fail in state q or in one of the a states already. Otherwise, they recover to state r, send *recovery* messages and wait to receive an *abort* message, then move to state a_k. There are k nodes in its CPS, so at least one operational node replies with an *abort* message. *Q.E.D.*

A.9 Proof of Lemma 5.5

Since no nodes fail in state q, all nodes have successfully sent out their local decisions in round 1, *viz*, failures can only occur after round 1. There are 2 cases:

- All Nodes Want to Commit the Transaction. All nodes make a commit decision, send yes^1 messages, and move to state w_1 in round 1. Because messages are guaranteed to be delivered, all operational nodes after round 1 receive either *yes* messages or *timeout* messages (if some nodes fail after

round 1). According to the RDCP protocol, these *timeout* messages produce no *no* messages if the timed out nodes are in a *w* state. Since no *no* messages are generated, all operational nodes eventually move to state *c*. For those nodes failed after round 1, they recover to state *r*, send *recovery* messages and wait to receive a *commit* message (at least one operational node in its CPS), then move to state *c*.

- Some Nodes Want To Abort the Transaction. If no more than $k-2$ nodes can fail and there are at least $k-1$ disjoint paths whose lengths are at most k between any 2 nodes, all operational nodes receive at least 1 *no* message and eventually move to state a_k. Similar to lemma 5-4, all failed nodes eventually abort. *Q.E.D.*

A.10 Proof of Theorem 5.2

The proof follows from lemmas 5-4 & 5-5. *Q.E.D.*

A.11 Proof of Theorem 5.3

Since the M logical nodes are obtained by adding $M-N$ virtual nodes into the system and $M=2^k$, $k=\text{liub}(\log_2(N))$, then $M \in [N, 2N]$. From theorem 4-3, the total number of messages sent among the M logical nodes is $M \cdot k^2 \leq 2N \cdot \text{liub}(\log_2(N)) = O(N \cdot \log_2^2(N))$. *Q.E.D.*

REFERENCES

[1] J.N. Gray, "Notes on database operating systems", *Operating Systems: An Advanced Course*, 1979; Springer-Verlag.

[2] M. Hammer, D. Shipman, "Reliability mechanisms for SDD-1: A system for distributed databases", *Technical Report*, 1979 Jul; Computer Corporation of America.

[3] C. Mohan, B. Lindsay, "Efficient commit protocols for the tree of processes model of distributed transactions", *Proc. 2nd ACM SIGACT/SIGOPS Symp. Principles of Distributed Computing*, 1983, pp 76-80.

[4] T.V. Lakshman, A.K. Agrawala, "Efficient decentralized consensus protocols", *IEEE Trans. Software Eng'g*, vol SE-12, num 5, 1986, pp 600-607.

[5] S. Yuan, A.K. Agrawala, "A class of optimal decentralized commit protocols", *Proc. 8th Int'l Conf. Distributed Computing Systems*, 1988, pp 234-241; IEEE.

[6] D. Skeen, "Nonblocking commit protocols", *Proc. ACM SIGMOD Conf. Management of Data*, 1982, pp 133-147.

[7] K.V.S. Ramarao, "Design of transaction commitment protocols", *Information Sciences*, vol 55, 1981 Jun, pp 129-149.

[8] A. Albert, R. Sandler, *An Introduction to Finite Projective Planes*, 1969; Holt, Rinehart & Winston.

[9] H. Sullivan, T. Bashkow, "A large scale homogeneous, fully distributed parallel machine", *Proc. 4th Symp. Computing Architectures*, 1977 Mar, pp 105-117; ACM.

[10] M.J. Fischer, N.A. Lynch, M.S. Paterson, "Impossibility of distributed consensus with one faulty process", *J. ACM*, vol 32, 1985 Apr, pp 374-382.

[11] R.D. Schlichting, F.B. Schneider, "Fail-stop processor: An approach to designing fault-tolerant computing systems", *ACM Trans. Computer systems*, vol 1, 1983 Aug, pp 222-238.

[12] D. Skeen, M. Stonebraker, "A formal model of crash recovery in a distributed system", *IEEE Trans. Software Eng'g*, vol SE-9, 1983 May, pp 219-228.

[13] S. Yuan, "The communication complexity for decentralized evaluation of functions", *Information Processing Letters*, vol 35, num 4, 1990, pp 177-182.

[14] O. Wolfson, A. Segall, "The communication complexity of atomic commitment and of gossiping", *SIAM J. Computing*, vol 20, 1991 Jun, pp 423-450.

[15] S. Yuan, A.K. Agrawala, "Fault-tolerant decentralized commit protocols", *J. Parallel & Distributed Computing*, vol 13, 1991, pp 299-311.

AUTHOR

Dr. Shyan-Ming Yuan; Dept. of Computer & Information Science; National Chiao Tung University; 1001 Ta Hsueh Road; Hsinchu 30050, TAIWAN - R.O.C.

e-mail (Internet): smyuan@tiger.cis.nctu.edu.tw

Shyan-Ming Yuan (M'89) was born 1959 July 11 in Mauli, Taiwan ROC. He received the BSEE (1981) from National Taiwan University, the MS (1985) and PhD (1989) both in Computer Science from University of Maryland. He joined the Electronics Research and Service Organization, Industrial Technology Research Institute as a Research Member in 1989 Oct. Since 1990 Sep, he has been an Associate Professor at the Institute & Department of Computer & Information Science, National Chiao Tung University. His research interests include distributed system design, fault-tolerant computing, computer supported cooperative works, and multimedia application environments. Dr. Yuan is a Member of IEEE.

Manuscript received 1994 February 23.

IEEE Log Number 94-01523 ◄TR►