International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:8, No:1, 2014

Fault Tolerance in Distributed Database Systems

M. A. Adeboyejo, O. O. Adeosun

***Abstract*—**Pioneer networked systems assume that connections are reliable, and a faulty operation will be considered in case of losing a connection. Transient connections are typical of mobile devices. Areas of application of data sharing system such as these, lead to the conclusion that network connections may not always be reliable, and that the conventional approaches can be improved. Nigerian commercial banking industry is a critical system whose operation is increasingly becoming dependent on information technology (IT) driven information system. The proposed solution to this problem makes use of a hierarchically clustered network structure which we selected to reflect (as much as possible) the typical organizational structure of the Nigerian commercial banks. Representative transactions such as data updates and replication of the results of such updates were used to simulate the proposed model to show its applicability.

***Keywords* —**Dependability, reliability, data redundancy.

I. Introduction

THE type, quantity and quality of information kept by an organization and, how the information is stored and exchanged keep changing as new and more efficient methods of information management evolve. The idea behind the design of a particular technology dictates how it will be used to solve problems. The design of the mainframe computers was meant for business systems in which centralized computing power matched the mode of operation of the business. Central method of control, efficiency and economy are some of the motivational factors for an organization to adopt a centralized information system [1]. On the other hand, excessive control, still according to [1], may result in bureaucratic bottle necks and, the resultant inflexibility of such systems can also cause operational costs to escalate. Efficiency and economy of a centralized information system may be favorable where the business organization is relatively small.

However, in a multi-branched organization, benefits of economy of scale and higher throughput outweigh the suitability of a centralized information system [1]. The idea of data redundancy which is usually avoided as much as possible in database parlance [2] is also a reason for keeping data in a location. On the other hand, the risk of absolute failure of the system as a result of software or hardware error is high. Moreover, it is practically not possible to implement all database normal form rules to the letter [3]. Redundancy and efficiency of data access has to be balanced.

M. A. Adeboyejo is with the Directorate of Information and Communication Technology, National Open University of Nigeria, Lagos State, Nigeria (phone: 08037636445; e-mail: phadamoses@gmail.com).

O. O. Adeosun is with Department of Computer Science and Engineering, Ladoke Akintola University, Ogbomoso, Osun State, Nigeria (phone: 07038866844; e-mail:hootadeosun@yahoo.com).

II. Basic Concept

A. Database (DB) Models

Information with relevant attributes for a given situation is usually fetched from a source that is systematically organized in line with the overall business objective of the organization (such as a bank) that owns the collection of information [4]. Hence, a database (DB) is a collection of pieces of information that is organized so that it can easily be accessed, managed, and updated [5]. It follows that a DB is an organized collection of specific related and integrated data representing some aspect of the real world with the purpose of meeting a set of specific information needs.

The ability of a system to make required pieces of information available as at when needed with little or no need for further modification is one of the main objectives of a database management system (DBMS). Though there are several models (networked, hierarchical, relational model etc.) by which data can be organized and maintained to meet the requirements of a particular organization or an individual [2], [6], [3] yet, some of the models are more commonly used than others. One of the approaches to database modeling is to simulate real live objects as the basic data elements in a database. Here, the object of interest on which data is being stored is the main focus. This is the object oriented database model (OODM) and according to [3], it is a model in which information is represented in the form of objects as used in object oriented programming. However, we observed that the most natural way for humans to begin to study objects is to initially group them according to some criteria of interest. This natural tendency is what informed the hierarchical and relational database model with the relational model being the commoner of the two. The relational database (RDB) model enables the classification and mapping of the attributes of one or more object of interest into the model of the RDB using its semantics. The adopted model is usually incorporated into a much larger DBMS.

In the RDB approach, data elements are organized into a collection of stacked horizontal sets of attributes, each of which represents an entity (e.g. bank account) in the collection. This in physical terms effectively forms a tabulated set of data elements. Reference [7] formally defines a relation R as $R (A_1, A_2, A_3, A_4,..., An)$ where R is defined over attributes $(A_1, A_2, A_3, A_4,..., A_n)$ and A_i is the *ith* attribute in R for all integer $i \in \{1, 2, ..., n\}$. Each corresponding attribute of the entities belongs to a domain D of a set of atomic values which are indivisible as far as the RDB model is concerned [2]. The *nth* attribute A belongs to a domain defined as $dom(A_i)$ which refers to the collection of properties or conditions that must be satisfied by whatever value A_i will hold [3].

B. Distributed DB Management System (DDBMS) Architecture

A DBMS comprises of integrated and interdependent components that interacts in an organized and coordinated way with the aim of achieving a central objective. Reference [8] describes a DBMS as being made up of the DB, the DB engine, the application program and the user. A single DBMS is one in which all these components are located closely together or in a single hardware unit. For the various system components to work together towards achieving global system objectives, relevant data and/or information need to be exchanged. When this coordination is to be done amongst remotely located components, then activities have to be synchronized and coordinated properly via well established communication network of suitable coverage (local-, wide-, and/or metropolitan-area network etc.) and topology (ring, star, mesh, tree etc.). The extent of similarity and/or dissimilarity of these components across the network define the degree of heterogeneity and/or homogeneity of the DBMS [9].

The major components of the architecture of a distributed DBMS (DDBMS) are the underlying network of the distribution and the structure of the DBMS. The relationship of interaction (service request and rendition) between the components on the network irrespective of the type and topology could refer to client-to-server and/or peer-to-peer architecture [10]. When DBMS architecture refers to the distribution of application logic (presentation, processing and storage) on the network, there can be one-, two- or three-tier architecture, depending on how many major hardware units are used to host which portion of the application logic.

C. Distributed Data Storage (DDS)

To improve availability and reliability in a DDBMS, storage analysis typical of single DBMS are extended. This extension results in concepts such as fragmentation and replication which can either be combined or implemented separately. One of the means by which a DDBMS is made to appear as a single unit to the users is by ensuring that data are easily accessible across the entire network by either fragmenting and/or replicating data, that is ensuring single system image [11]. Data fragmentation can be horizontal, vertical or mixed. In horizontal partitioning, table of data is split along the row into two or more fragments. The splitting is done such that the only difference between the original and resulting smaller tables is the table size. Every other property of the parent table is inherited by the fragments [3]. Vertical partitioning is done along columns while mixed partitioning combines the two. On the other hand, data replication is the process of copying and maintaining database objects in multiple databases that make up a distributed database system [3].

D. Distributed Data Transaction

The commands issued by the DB users cause the DB engine to create a new record, update, delete and/or retrieve an existing record [12]. For a DB engine to execute these commands, it must break them into logical units of processing and then execute these as a string of serialized interleaved (from different transactions) operations [3]. As in the single DBMS, transaction properties (atomicity, consistency, isolation and durability) are upheld. These however, are accomplished via local transaction manager (TM) at every site in a distributed system. To ensure a single image system, the TM at a site coordinates with all TMs at other sites. These interactions according to [3], are synchronized by a transaction coordinator (TC) whose specific responsibilities according to [13] are to:

- Start the execution of transactions that originate at the site.
- Distribute sub-transactions at appropriate sites for execution.
- Coordinate the termination of each transaction that originates at the site, which may result in the transaction being committed or aborted at all sites.

E. Transparency in DDBS

System transparency generally refers to the separation of the higher-level functional details of the system from the lower-level implementation issues [14]. A transparent DDBMS is one in which users are able to use the system without being involved in the intricacies of ensuring consistency, availability and reliability of the system. This refers to the degree to which users are unaware of the various elements of the system. There are various aspects to transparency. When a DB system is developed independently of other DB locations, particularly in a distributed system, and it is not aware of the design decisions and control structures adopted at other sites, such database is said to have local autonomy [15]. The following [13] are the criteria for ensuring local autonomy:

- Every data item must have a system-wide unique name.
- It should be possible to find the location of data items efficiently.
- It should be possible to change the location of data items transparently.
- Each site should be able to create new data items autonomously.

According to [3], replication and fragmentation transparency ensures that users need not refer to a specific replica of a data item when issuing queries. Instead, the name of the relation is all that the users need to know. The system should determine which replica of the relation mentioned in the user query to reference. According to [16] location transparency is a property of a DDBMS that should ensure that users do not need to be aware of the location of data in a distributed database. And, that data should be accessible at a remote site just as easily as it is at a local site.

III. Purpose of Study

The Nigerian commercial banking industry is a critical system whose operation is increasingly becoming dependent on technology driven information system [17]. Though the users (banks) use it as a tool to gain marketing advantage yet,

a down time of few minutes caused by this same tool almost always result in financial and/or physical damages to the customer and loss of revenue to the bank with a result wavering customer loyalty [18]. The discomfort through which customers have to pass makes them to quickly forget the conveniences afforded by the technology.

The dependence of the commercial banking system (such as that of Nigerian) on the global inter-network of computers and related accessories has become a critical factor in determining the quality of service delivery [19] [17]. Therefore, outages in customer services of these banks with respect to its underlying DDBMS need to be minimized as much as possible. One of the ways of achieving this is building fault tolerance (FT) into the system [11] via redundancy concepts such as data replication.

IV. FT IN DDBMS

According to [20], a system is said to be fault-tolerant (FT) if it can mask the presence of one or more faults in the system by using redundancy while performance may be degraded. That is, FT allows a system to continue to behave according to design objectives. Redundancy implies the presence of parts or modules of similar configuration to the one that is functioning but whose purpose is to form error checking quorum and possibly take over the functions of the active module when it fails. To achieve a reliable, high-availability system, [21] suggests two very different approaches which are fault-avoidance and fault-tolerance. Fault-avoidance is prevention of fault-occurrences by construction, while fault-tolerance, according to him is the use of redundancy to avoid failures due to faults [21]. According to [21], fault-avoidance is difficult, and close to impossible in large and complex systems. Therefore, FT is the only realistic alternative for the class of system (DDBMS) under consideration.

Implementing FT in a DDBMS with respect to the underlying network may take the form of hardware redundancy. Thus, it is important that a failing component (site) stops functioning and transfer its responsibility (via well established algorithms) to an otherwise redundant and similar component meant for the same/similar purpose in the system. To achieve redundancy, modules (sites) can be in duplicate in the form of duplex module redundancy (fig. 1) in which two identical copies of a module are connected to a comparator that checks their respective output. When the output differs, a fault is detected. Since the fault is detected immediately (albeit, with a small latency), it is therefore called fail-fast [21]. In the form of triple module redundancy (TMR) as in fig. 2, the expectation is that outputs from all components should be similar. If one (or more) is different, a fault signal is sent to the user and the system continues to function as expected though the failing component would have been decommissioned [22].

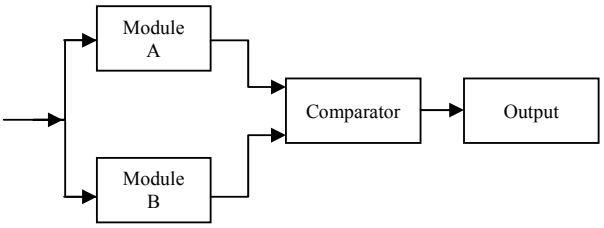

Fig. 1 Duplex module redundancy

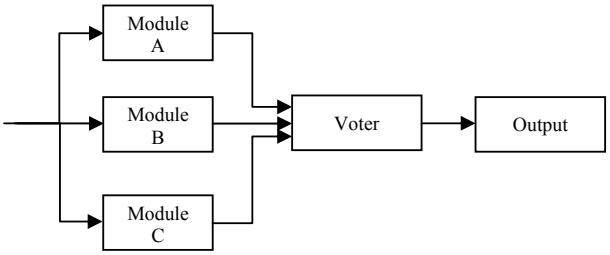

Fig. 2 Triple module redundancy

This is fail-vote setup since it requires a majority vote to determine the correctness of the outputs of the respective modules [21].

Unlike software systems meant for small, self-contained DB systems, it can be extremely difficult to find and diagnose more unusual bugs in software systems meant for DDBMS. Approaches to developing software that can tolerate software design (programming) errors or user fault have made use of static and dynamic redundancy approaches similar to those used for hardware faults. Some of these are N-version programming [23], recovery blocks [22] and N-self checking programming [24]. Another means by which FT can be built into a (distributed) system via redundancy is by adding extra information (meta data) to data, to allow error detection and correction. These are typically error-detecting codes, error-correcting codes (ECC), and self-checking circuits [21] [25].

Data replication which is a form of information redundancy represents a particular simple instance of the more general FT technique of introducing redundancy. Reference [26] defines data replication as the making of straightforward copies of meaningful units of data, processing or communication. The two main replication techniques are primary backup replication and active replication. Replication could be described as active when a primary server (RM) processes the client's requests and replies the client immediately (fig. 3). On the other hand, replication is described as passive when primary server (RM) processes client's requests, propagates updates to other backup replica servers and then responds to the client's request (fig. 4).

Whichever technique is adopted in a particular system dictates the mode of inter component communication within the system. The modes are group and point to point communication. Generally FT system architecture is made up of the client (C), front end (FE) that is, the client interface and replica manager (RM) which is the service provider [27]. The

basic model for managing replicated data as stated by [27] is described in the following algorithm:

a. Request: Client issue requests to a frond end.
b. Server coordinates: The replica server coordinate (through the FE) with each other to synchronize the execution of the operation (ordering of concurrent).
c. Execution: The FE contacts one or more RMs to retrieve or store the data.
d. Agreement coordination: The RMs interacts to ensure that data is consistent.
e. Response: Outcome of operation is transmitted back to C.

V. System Design

The proposed model is designed with respect to the fundamental objectives (data consistency, availability and reliability) of data redundancy towards ensuring a fault tolerant DDBMS. The model comprises of components (users, database storage, database engine, application logic, user interface, communication links, sites or nodes and messaging) that are typical of a DBMS. Some of these components such the DB engine, user interface, application logic, users etc. are implied in the model. While others such as the communication links, nodes and inter-node relationships are obvious from the design, considering the fact that being a model, only the generic features of the design has to be explicitly featured. This is ensured through guiding protocols in the form of formats, rules and procedures for performing design functions. Such activities may include managing (member joining or leaving a group) the clustering of sites, assumptions and specifications on which inter node communications are based, relationship (parent or child node) amongst nodes etc.

A site can only communicate directly with a limited number of other sites. This, in line with design objectives, ensures minimal cost (heavy traffic, congestion, component failure) of data transmission thereby ensuring the availability of data with minimal delay. With respect to group membership, established protocols helps to ensure that all sites have the correct global map of the network of the system. This ensure that there are no unnecessary data transmission to failed sites and this in turn speeds up transmission because there is no waiting for response from a failed sites. Also, the adopted replication protocols for the proposed design ensure that data remains globally consistent, reliable and available irrespective of system condition.

The sites of the proposed model are homogeneous because the design is with reference to a field of application (banks) in which installed DBMS, user application and network structure are the same across sites. The database of the proposed model design is based on the relational database model. This is because it is easy to implement and maintain. Moreover, it is compatible with the data management requirements of the reference field of application. Database operations are on real time basis. The proposed model uses temporal data items. That is, there is emphasis on their timeliness.

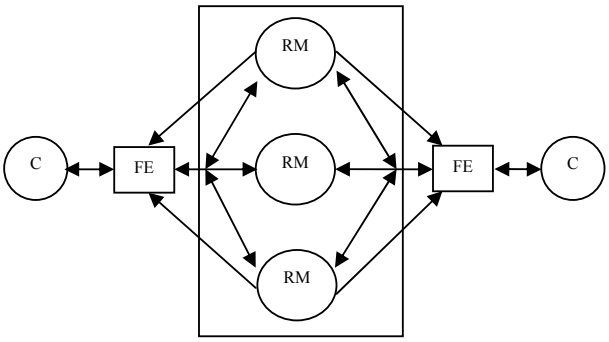

Fig. 3 Active replication model

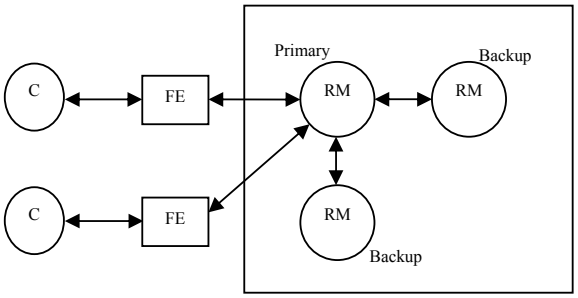

Fig. 4 Passive replication model

A. Architecture of Model

Observations have shown that business organizations are inherently centralized but with varying degree of decentralization hence, the adoption of the network structure in Fig. 5 [33] for the proposed replication model. Objects in the proposed model interact by passing messages (Table I) which could be a request for data update or retrieval in either user or system table(s). In the proposed replication model, database D is distributed over N nodes. Each database d_i *(for* $0 < i \leq N$*)* is resident at a node N called a site (Fig. 5). Each site hosts a set of temporal user and system data objects. The site is called the PS for those data objects. Every site has a front end (FE) made up of transaction coordinator (TC) and transaction manager (TM) with which distributed processes are managed.

The proposed model adopts cluster approach to replication in which sites are grouped into metropolitan area networks (MAN), each made up of a primary and one or more secondary sites. The MANs are hierarchically connected to form a tree shaped WAN (Fig. 5). This structure agrees with the usual geographical distribution of the reference field of application (banks). The structural model is made up of one headquarters (HQs) site S_1, two regional branches (RB) S_2 and S_3, four zonal branches (ZB) S_4 to S_7 and eight branches (B) S_8 to S_{15}. Here, a collection of Bs is headed by a ZB. A collection of the ZBs is in turn headed by higher level RB. The hierarchy builds up until it gets to the HQs of the bank. The ZB-B relationship is that of primary to secondary site (SS).

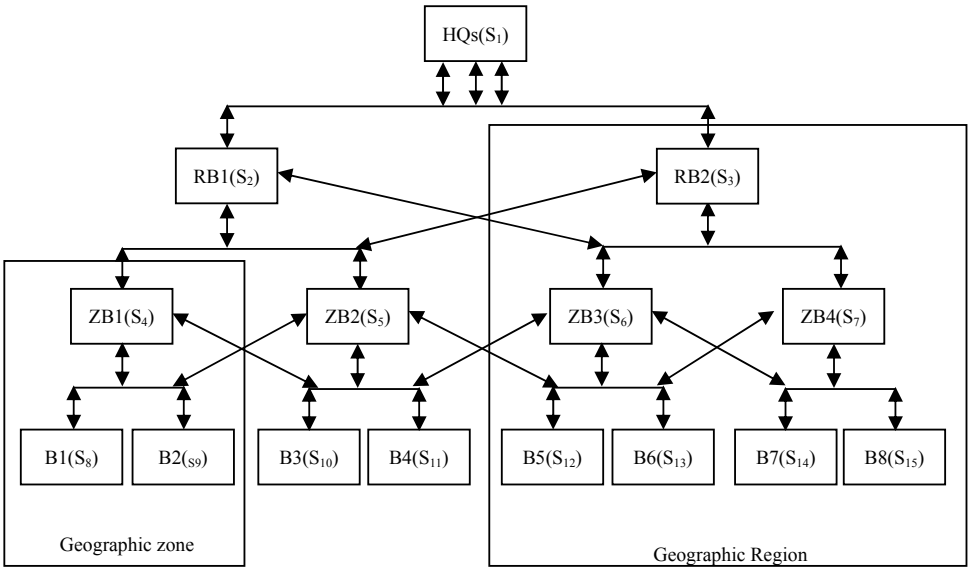

Fig. 5 Schematic diagram of a clustered networked organization

Similarly, there is a PS to SS relationship between the RBs and the ZBs and so on. The processing and storage capacity at a site is directly proportional to its level on the hierarchy so that S_1 has the highest capacity while the branches have the lowest capacity. For a specific data item, the data copy at the PS is called primary copy and the copies that are replicated are called the replicas. The database size is the total size of the databases at all sites defined as follows:

$$D = \sum_{i=1}^{N} d_i \quad (1)$$

where $N = 15$ and i is the *ith* database at the *ith* site.

B. Design Assumptions

To ensure that only generic features are considered and that details of reality do not introduce unnecessary and avoidable level of design details, assumption about the practical application of the model and the specifications under which it can be implemented were considered subject to design objectives. The assumption and specification guiding the global state of the network, inter-site mode of communication and interaction and global system behavior are thus presented:

- Message passing is structured because of the existence of a physical (and logical) order in the form of a hierarchy along which communication flows [28].
- The DDBMS is modeled as a set of services implemented by server (PS) processes and invoked by client (SS) or local RM processes [29] through message passing.
- No two adjacent PSs within a region should be down at the same time. This ensures that at no time are PSs of a given hierarchy level loaded beyond their maximum capacity.
- Redundancy in connection there are errors due to break in communication links.
- Replication is symmetrically done on two adjacent sites within a region and upwardly from SSs to PSs.
- A complete communications sequence includes source and destination site ID, message sequence number to avoid unnecessary re-transmission, timer to know if/when to retransmit, and acknowledgement (ACK) of reception to the sender. The last three are usually handled at system level.
- PSs are configured for contingency extra load from secondary sites of adjacent zone or region.

The Bs are the most accessible to customer thus, there is no PS to B replication and this keeps the Bs as light as possible in terms of quantity of data they hold. Hence, model is based on 1-dimensional partial replication [30].

C. Managing Group Membership

For the various sites to interact effectively there has to be well established communication modes to ensure synchronization of site activities. Atomic broadcast (ABCAST) provides atomicity and total order. Reference [29] illustrates thus: let *m* and *m'* be two messages that are ABCAST to the same group *g* of servers (sites). The atomicity property ensures that if one member of *g* delivers *m*, then all (not crashed) members of *g* eventually deliver *m*. The order property ensures that if two members of *g* deliver both *m* and *m'*, they deliver them in the same order. On the other hand, view synchronization broadcast (VCAST) handles communication issues by treating *g* as a view and handling changes in such view as processes join or leave *g*.

TABLE I
INTERPRETATION OF INTRA AND INTER SITE COMMUNICATION MESSAGES

Sn	Format	Message	Parameters	Intra Site (I), Inter Site (E)
1	(CATUPDATE, a, b)	CATUPDATE	a = failed site, b = what happened to site	I
2	(FAIL, a, b, c)	FAIL	a = sender, b = failed site, c = receiving site	E
3	(ELECTREQ, a, b, c)	ELECTREQ = Request to be elected	a = requesting site, b = failed site, c = receiving site	E
4	(CATUPDATE, 4, 2, ELECTREQ)	CATUPDATE = update local catalo	a = requesting site, b = failed site, c = request	I
5	(SELECTED, a, b, c, d)	SELECTED = A site has been selected	a = sender, b = selected site, c = failed sites, e = receiving site	E
6	(CATUPDATE, a, b, SELECTED)	CATUPDATE = update local catalo	a = requesting site, b = failed site, c = a site has been selected	I

The proposed model ensures that failure of a site before, during, and after data update does not affect the communication of the record updates and retrievals to relevant sites. This is achieved via group (zonal or regional) membership communication. The group membership process model is as follows:

- Sites are identified by their hierarchical numbers while database objects are named to ensure transparency and local autonomy using aliases. In this way, users can be unaware of the physical location of a data item. Furthermore, the user is unaffected if the DB administrator should decide to move a data item from one site to another.
- Sites are grouped into multicast groups. A zone is a group of sites within a region which in turn is a group of sites within the wan.
- Membership changes are proactively monitored via status check (gossip) signal [27].
- Only a site that is an SS to the failed site can initiate the notification process.
- If the failed site is a B, only its peer can initiate the notification process.

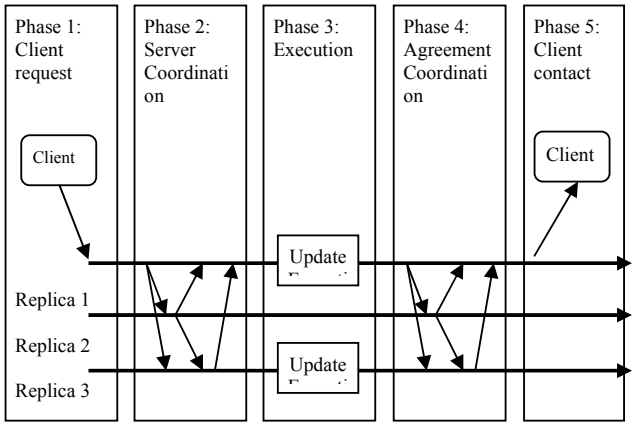

Fig. 6 Proposed replication model

D.Proposed Replication Model

The proposed model adopts a variant of replication techniques as proposed by [29]. The model begins with a logical data item x and its physical copy x_i on the different sites. The basic unit of replication is the data item. x_i belong to a tuple t_i which in turn belongs to a replica r_i. Clients access the data by submitting transactions. An operation, $o_i(x)$, of a transaction, T_i, can be either a read or a write access to a logical data item, x in the database. Hence, the two basic transaction type considered are update and retrieval transaction. Delete is treated as an update that marks a record as deleted while addition of a new record is treated as an update of a record whose fields are empty. With respect to fig. 5, direction of replication is as follows:

- Vertically in both directions, excluding the Bs
- Horizontally between two adjacent ZBs of the same region
- Horizontally between two adjacent RBs
- Between the ZB and the RB closest to each other

The model (fig. 6) divides client-server interaction into five phases as explained below:

a. Request (RE): the client submits an operation to one (or more) replicas.
b. Server coordination (SC): the replica servers coordinate with each other to synchronize the execution of the operation (ordering of concurrent operations).
c. Execution (EX): the operation is executed on one or more replica servers.
d. Agreement coordination (AC): the replica servers agree on the result of the execution (for example, to guarantee atomicity, consistency etc.).
e. Response (END): the outcome of the operation is transmitted back to the client.

E. Adaptation of Model

Two representative transactions are illustrated below which involve sites using inter- and intra-site messages as interpreted in Table I.

1. Replication of Local Update to Next Closest PS.

This is typical of a (credit or debit) T_i at a terminal branch S_i for $8 \geq i \leq 15$ (fig. 5) on account A. The effect of this update has to be replicated upwards to a PS S_j for $4 \geq j \leq 7$. Model (Fig. 6) is adapted as follows for $i = 9$ and $j = 4$ (Fig. 7):

a. Request (RE): S_9 requests for a global lock over X_i. When granted, it initiates an update transaction T_9 that reduces credit balance of A because of a withdrawal transaction, and submit T_9 to its local replica manager (RM9) coordinated.
b. Server coordination (SC): There is no need for this phase because RM9 is the only RM involve.

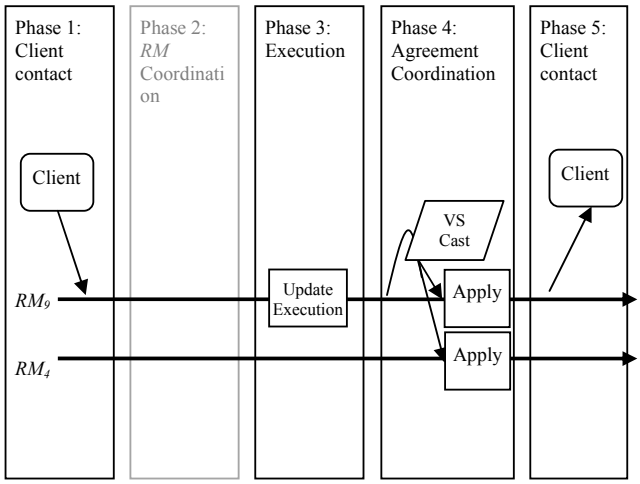

Fig. 7 Propagation of local changes to immediate PS

c. Execution (EX): the operations of T_9 are executed on RM9.
d. Agreement coordination (AC): RM9 and RM4 agree on the result of the execution (to guarantee atomicity, consistency etc.) and commit it, while at the same time ensuring that the other is alive.
e. Response (END): the outcome of the operation is transmitted back to the client from RM_9.

The algorithm for replicating local updates to the next closest PS is as shown in appendix.

2. Propagation of Changes in Account (A) To Other Sites

Being a clustered network, S4 only has to replicate changes to S2 and S5 while the lock over X_i is still been maintained. The steps to accomplish this, illustrated in Fig. 8 follow:

a. Request (RE): There is no need for this step because RM4 is the initiator.
b. Execution (EX): the operation is executed on RM2 and RM5.
c. Agreement coordination (AC): the RM4, RM2 and RM5 send agreement and acknowledgement (ACK) message amongst themselves (coordinated by the TC) on the result of the execution (for example, to guarantee atomicity, consistency etc.).
d. Response (END): the outcome of the operation is transmitted back to the RM4.
e. Steps a to d are iterated till it gets to S1 then the pattern is reversed starting from S3 downwards till it gets to S6 and S7.
f. Lock over X9 is released.

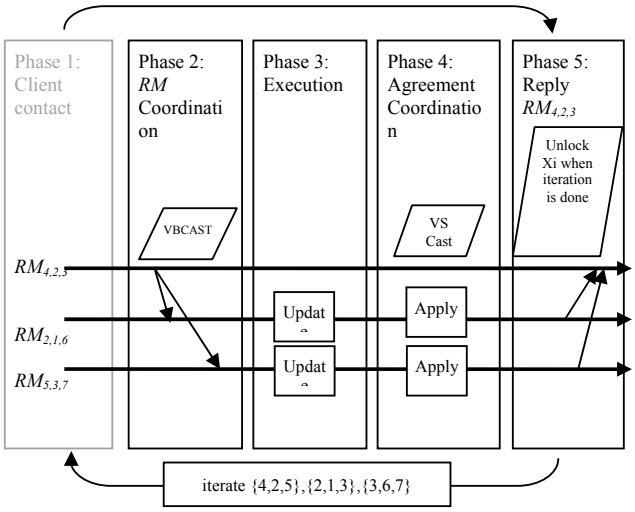

Fig. 8 Propagation from PS to other sites

The algorithm for replicating updates in account A to other sites is as shown in the appendix.

F. Performance Evaluation

Performance of the system (model) was measured in terms of its availability and reliability, subject to the design objectives. To achieve this, the metrics of the system that were monitored were used to assess its behavior (performance). In quantitative assessments, relevant system attributes are most often qualified probabilistically and modeled to simplify the process [31]. For example, reliability, R(t) which is the conditional probability that the system can perform its design function at time t given that it was operational at time $t = 0$ is used. Likewise, availability, $A(t)$ which is defined as the probability that the system is operational at time t could also be used. To ensure a system remains reliable and available in the presence of faults, there has to be timed automatic and/or manual repairs.

Two of the metrics used to evaluate the performance of the proposed model are statistical mean time values of the system failure and repairs times [31]. Mean time to failure (MTTF) is the expected time of failure of the system while mean time to repair (MTTR) is the expectation of the time repair a failure. A combination of these two parameters gives steady state availability as:

$$A_{steady\ sate} = \frac{MTTF}{MTTF + MTTR} \quad (2)$$

A highly available system means that availability is close to 1. This will be if MTTF is relatively large. A smaller MTTF implies that availability varies significantly with repair time. Therefore, the performance of the proposed model can be compared with that of other models (for example, centralized systems) in terms of MTTF and MTTR. As an illustration, let x_0 be the value of MTTF of the proposed model and let the model network structure (Fig. 5) be further abstracted to S1, S2 and S3. The resulting four possible sates of the system with respect to either healthy or failed state are shown in Fig. 9.

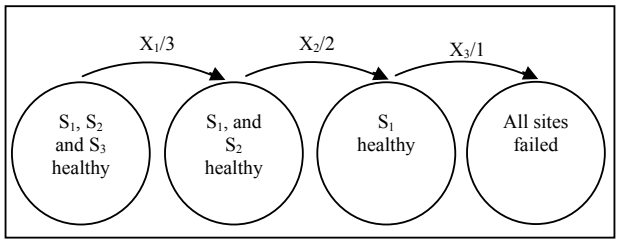

Fig. 9 Transition diagrams to estimate the MTTF (x) of proposed model

The more the sites to which data is replicated, the higher the value of MTTF is (Fig. 9). MTTF initially had a value of x_0 with all sites healthy but started reducing as sites failed down to x_3 ($x_0 > x_1 > x_2 > x_3$). System state with only S1 healthy could be compared to a system in which all data are held in a location with no copies anywhere else that is, a centralized system. Therefore, it can be concluded that the replicated model is more dependable (availability and reliability) than the non-replicated and centralized system.

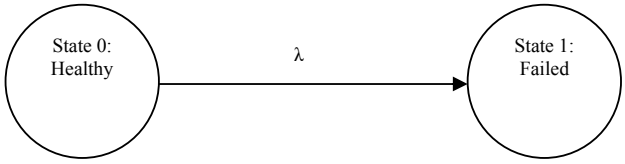

Fig. 10 Transition diagrams to estimate rate of change of state of proposed model

With reference to [32], another model that was used to assess the performance of the proposed model is the Markov's model (named after the mathematician Andrei Markov). Assuming that each site in the proposed model represents a component in the system, the probability of the state of one component at $t = t_0$ gradually changes to the probability of the state of the same component at $t = t_1$ (Fig. 10) where λ is the parameter of the rate of transition. If $P_j(t)$ is the probability of the proposed model being in state j at time t and the site in question is known to be healthy at some initial time $t = 0$, then the initial probabilities of the two states are $P_0(0) = 1$ and $P_1(0) = 0$. Thereafter the probability of state *0* decreases at a constant rate (not always the case in reality) λ, which means that if the system is in state *0* at any given time, the probability of making the transition to state *1* during the next increment of time dt is λdt. Therefore, the overall probability that the transition from state *0* to state *1* will occur during a specific incremental interval of time dt is given by multiplying the probability of being in state *0* at the beginning of that interval, the probability of the transition during an interval dt given that it was in state *0* at the beginning of that increment. This represents the incremental change dP_0 in probability of state *0* at any given time as modeled below:

$$dP_0 = (P_0)(\lambda dt) \quad (3)$$

Dividing both sides by dt gives

$$\frac{dP_0}{dt} = -\lambda P_0 \quad (4)$$

This implies that a transition path from a given state to any other state reduces the probability of the source state at a rate equal to the transition rate parameter λ multiplied by the current probability of the state. Now, since the total probability of both states must equal *1*, it follows that the probability of state *1* of the component must increase at the same rate that the probability of state *0* is decreasing. The actual Markov model of the designed system will include a "full-up" state (that is, the state with all sites operating) and a set of intermediate states representing partially failed condition, leading to the fully failed state, that is, the state in which the system is unable to perform its design function. The model may include repair transition paths as well as failure transition paths.

One of the factors that determine the rate of transition (λ) from healthy to failed state depends on how much redundancy there is in the system. Given the same conditions, for example, a DMR takes shorter time to transit (takes two failed components to disable the system) while a TMR takes longer time. That is

$$-\frac{dP_0}{dt} \alpha \frac{1}{x} \quad (5)$$

Therefore, again, it can be concluded that the proposed distributed and replicated DB model will take a longer time to fail than a non-replicated centralized system.

VI. Conclusion

Having studied the application of FT in the deployment of IT in the Nigerian commercial banking industry, the resultant proposed design model presents some implications and some areas of further studies. Site-wide data replication enhances *MTTF* thereby reducing frequency of service down time. As a result, loss of revenue due to leaving of dissatisfied customers will be minimized. The bank in question will be able to increase its share of the market continually as more customers will naturally want to do business with such bank. On the other hand, there is an initial cost of ensuring an FT system. However, the effect of such cost is eventually off-set by the initially mentioned implications.

Appendix

1. Algorithm for replicating local update to next closest PS is as follows:

```
BEGIN
  LET j = 4; i = 9;
  PROPAGATE (i, j)
  {
        While lock request on di.ri.ti.Xi is pending{request lock on
        di.ri.ti.Xi;}
        While di.ri.ti.Xi is locked {
              Si READ Xi FROM di.ri.ti WHERE di.ri.ti.accounID =
        'CustomerId';
              Si SET Ti = di.ri.ti.Xi - withdrawn_amount;
              Si submit Ti to TMi;
              TMi BEGIN Ti of oi(x);
              TMi submit resulti of oi(x) to RMi;
```

```
                IF !passMessage (UPDATE, result, T, i, j) {
        PROPAGATE (i, j+1);)}}
                IF RMi AND RMi-5 alive AND RMi agrees with RMi-5
                on resulti-5 of oi(x) THEN COMMIT Ti of oi(x) at RMi
                AND RMi-5;
                Release lock on di.ri.Xi ti;
                RMi send resulti to Si;
        }

        BOOLEAN passMessage (a, b, c, i, j)
        {
                FOR (timeCount = 0; timeCount ≤ 4; timeCount++) {
                  WHILE !timeOut AND no ACK from {
                    RMi UNICAST MESSAGE (a, resulti , Ti) to RMj;
                        IF timeOut THEN BREAK; ELSE
                RETURN ACKi;
                  }
                  timeCount++;
                }
                RETURN FALSE;
  }
 }
END
```

2. Algorithm for replicating updates in account A to other sites is as follows:

```
BEGIN
   FOR (i = 4; i ≤ 2; i -= 2) {
                LET di.ri.ti.Xi = obji
                IF i = 4 THEN {
                        IF RMi is alive THEN {
                                RMi ABCAST obji to RMi-2 AND
                        RMi+1;
                                IF RMi-2 is alive THEN {RMi-2
                                UPDATE obji-2; RMi-2 ACK RMi;
                                }
                                IF RM i+1 is alive THEN {RMi+1
                                UPDATE obji+1; RMi+1 ACK
                                RMi;}
                        } ELSE {
                                RM i+1 UNICAST obji+1 to RMi-2;
                    IF RMi-2 is alive THEN {
                                        RMi-2 UPDATE obji-2;
                                RMi-2 ACK RMi+1;
                } ELSE {
                                   RMi+1 UNICAST obji +1to
                        RMi-1;
                                   RMi-1 UPDATE obji-1; RMi-1
                        ACK RMi+1;
                                }
            }
     }ELSE IF RMi is alive THEN {
     {
        RMi ABCAST obji to RMi-1 and RMi+1;
                   IF RMi-1 is alive THEN RMi-1 UPDATE obji-1; RMi-1
                ACK RMi;
                        IF RM i+1 is alive THEN RMi+1 UPDATE
                obji+1; RMi+1 RMi;
     }
          }

          FOR (i = 1; i ≤ 3; i += 2) {
                LET di.ri.ti.Xi = obji
                IF i == 1 AND RMi is alive THEN {
                        IF RMi+1 is alive AND obji +1 <> obji THEN
                {
                                RMi UNICAST obji to RMi+1;
                                RMi+1 UPDATE obji+1; RMi+1
                        ACK RMi;
                        }
                        IF RMi+2 is alive AND obji +2 <> obji THEN
                {
                                RMi UNICAST obji to RMi+2;
                                RMi+2 UPDATE obji+2; RMi+2
                        ACK RMi;
                        }
                }ELSE IF i == 3 AND RMi is alive THEN {
                        IF RMi+3 is alive AND obji +3 <> obji THEN
                {
                                RMi UNICAST obji to RMi+3;
                                RMi+3 UPDATE obji+3; RMi+1
                        ACK RMi;
                        }
                        IF RMi+4 is alive AND obji +4 <> obji THEN
                {
                                RMi UNICAST obji to RMi+4;
                                RMi+4 UPDATE obji+4; RMi+4
                        ACK RMi;
                        }

              }
            }
END
```

REFERENCES

[1] M. Beck, "Centralized versus Decentralized Information Systems in Organizations", Emporia state university, 2010, p. 33.

[2] E, F. Codd, "A relational model for large shared data banks", *Communications of the ACM*. Vol. 13, No. 6, 1970, pp. 377-387.

[3] O. Folorunso, "CIT844 Advanced database management system", National Open University of Nigeria, Lagos, Nigeria, 2009, pp. 82 – 103, 207.

[4] K. Jahangir, "Improving organizational best practice with information systems". Knowledge Management Review, 2005.

[5] M. K. David and J. A. David, *Database concepts*, New Jersey, Prentice Hall, 2008.

[6] H. V. Jagadish, L. V. S. Lakshmanan, and D. Srivastava, "Revisiting the Hierarchical Data Model", *IEICE Trans. Inf. & Syst.Vol. E82-D, No. 1,* 1999, p. 3.

[7] W. Zhang, "CSS5443 Database management syatem – Relational database", University of Texas, San Antonio, 2011, p. 5.

[8] S. K. Singh, *Database systems. Concept, design and application*, Dorling Kindersly, India, PVT. Ltd., 2011, pp, 20, 441.

[9] C. Ray, C., "Distributed database system", Pearson Education India, 2009, p 36.

[10] O. J. Oyelade, "CIT853 Advanced database management system", National Open University of Nigeria, Lagos, Nigeria. 2013, pp, 12 – 20.

[11] P. Krzyzanowski, "Distributed systems – Fault tolerance – Dealing with an imperfect world", 2009.

[12] P. Mancier, "Managing database operations using ADO and C++, Part 1: Introduction to SQL", 2011.

[13] Silberschatz, Korth, and Sudarshan, "Database system concepts", 2005, pp. 16, 60, 69.

[14] J. Gamper, "Distributed databases", 2009, pp 17, 211

[15] D. Taniar, H. C. Clement, R. W. Leung, and S. Goel, "High-Performance Parallel Database Processing and Grid Databases", 2008, p 6.

[16] S. K. Rahim, and F. S. *Haug, Distributed database management system*, John Wiley & Sons, Inc., 2010, p. 68.

[17] M. C. Elder, "Fault tolerance in critical information system", University of Virginia, 2001, p. 1.

[18] K. Mats, "Distributed systems basics – Handling failure: Fault tolerance and monitoring", 2011.

[19] O. T. Ekanem, "Productivity in the Banking Industry in Nigeria", *Journal of Economics and Social Studies*, Vol. 3, No. 7, 2003, p 24.

[20] J. Guo, "Fault tolerant computing", The University of Michigan-Dearborn, 2004, p. 7.

[21] K. Nørv°ag, "An introduction to fault-tolerant systems", *IDI Technical Report*, Vol. 6, No. 99, 2000.

[22] B. L. C. Ramos, "Challenging malicious input with fault tolerance techniques", Black Hat Europe, 2007, p 3.

[23] A. A. Aviz˘ienis, "The methodology of N-version programming", University of California, 1995, pp. 24-46.

[24] L. L. Pullum, "Software fault tolerance. Technique and implementation", *Artec House Inc.*, 2001, pp. 132, 150.

[25] D. P. Siewiorek and R. S. Swarz, *Reliable Computer Systems: Design and Evaluation*, Digital Press, 1992.

[26] A. H. Abdelsalam, B. B. Abdelsalam, and B. B. Bharat, *Replication techiques in distributed systems*, Kluwer Academic Publishers, 1996, p 108.
[27] M. Aksu, "Fault tolerance in distributed systems", 2005, p 21, 23.
[28] J. Weijia, Z. Wanlei, *Distributed network systems: From concepts to implementations*, Springer, 2006, p. 44.
[29] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso, "Understanding replication in databases and distributed systems", Proceedings of the 20th international conference on distributed computing systems, 2000.
[30] N. Matthias, J. Matthias, "Performance modeling of distributed and replicated databases", *IEEE Transactions On Knowledge And Data Engineering*, Vol. 12, No. 4, 2000, pp. 645-672.
[31] V. P. Nelson, "Fault tolerance computing. Fundamental concept", 1990, p 19.
[32] P. Pukite and J. Pukite, *Markov Modelling for Reliability Analysis*, Wiley-IEEE Press, 1998.
[33] M. Kaiser, M. Görner, and C. C. Hilgetag, "Criticality of spreading dynamics in hierarchical cluster networks without inhibition", *New Journal of Physics*. Vol. 9, 2007.