Asian Journal of Scientific Research

ISSN 1992-1454

science alert
http://www.scialert.net

ANSInet
an open access publisher
http://ansinet.com

Asian Journal of Scientific Research 1 (2): 90-102, 2008
ISSN 1992-1454

Strategic Planning for Fault-Tolerant Internet Connectivity Using Basic Fault-Tolerant Architectural Design as Platform

[1]O.O. Adeosun, [2]E.R. Adagunodo, [3]I.A. Adetunde and [4]T.H. Adeosun
[1]Department of Computer Science and Engineering,
Ladoke Akintola University of Technology, Ogbomoso, Nigeria
[2]Department of Computer Science and Engineering,
Obafemi Awolowo University, Ile-Ife, Nigeria
[3]Department of Applied Mathematics and Computer Science,
University for Development Studies, Navrongo Ghana, Nigeria
[4]Department of Banking and Finance,
Osun State College of Technology, Esa-Oke, Nigeria

Abstract: Present focus in this study is to provide Internet connectivity without any interruption even at the presence of faults/failures thereby enhancing Internet services performance. To achieve this, the deployment and redeployment of faulty component(s) are done using Basic Fault-Tolerant (BFT) architectural design. A framework to provide enhanced performance in terms of confidentiality, integrity and availability in clusters is suggested using BFT, considering all sources of vulnerabilities including operating system/software, communication hardware, user-level communication and network protocols.

Key words: Deployment, redeployment, fault-tolerant, reliability, availability, accessibility, dependability, observability, fault trajectory, propagation

INTRODUCTION

Conventionally, a system can be equated with the combination of computer hardware and software (Adeosun, 2003). In real sense of it, a system is the entire set of components (computer and non-computer related) that provides a service to a user. Figure 1 represents a system exists in an environment and has operators and users. The system is structured to provide feedback to the operator and services to the user. In Fig. 1, operators are shown inside the system because operator procedures are usually a part of the system design and many system functions, including fault recovery, may involve operator action. Eqnally important are the system's designers and maintainers.

Reasonable systems are developed to satisfy a set of requirements that meet a need. A requirement that is important in some systems is that they be highly dependable in terms of reliability. Fault tolerance is a means of achieving dependability. A system that can tolerate fault would be highly stable, reliable, available and dependable.

Internet nodes and links are prone to failures (Helal *et al*., 1996). Failures may be of different nature, comprising hardware component crashes, incorrect software behaviour, performance overload, human errors, physical damages of wires, connectors, etc. The experimentally obtained frequencies of various fault types are shown in Fig. 2.

Corresponding Author: O.O. Adeosun, Department of Computer Science and Engineering,
Ladoke Akintola University of Technology, Ogbomoso, Nigeria

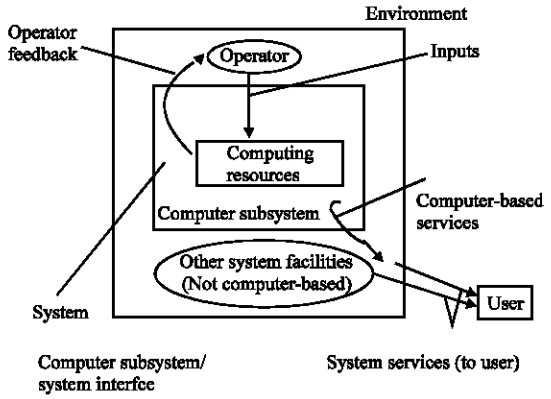

Fig. 1: Conceptual representation of computing system

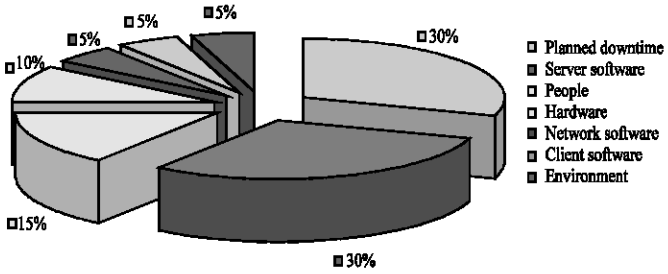

Fig. 2: Statistical distribution of different failure types (Stem and Marcus, 2000)

General classification of failure in an Internet service provisioning is as:

Host Failure

The corresponding server stops providing its service(s), due to a hardware/software crash of a subsystem. It is also known as server, site or node failure.

Network Failure

The corresponding virtual links stop serving as a medium to convey messages. It is also known as link or communication failure.

In presence of both types of failures, the requirements for high reliability and availability are straight forward; the service users must be unaware of the failures, i.e., they must continue being served in a transparent fashion. Introducing fault tolerance in this study fulfills this requirement. Fault-tolerance is achieved when another operational component of the system has to take over the functionality of the failed counterpart (Schlichting and Schneider, 1983). This is an approach, which makes use of redundant resources that contain enough functionality and state information to enable uninterrupted operation.

Rationale for the Study

Individuals, industries, schools, agencies and governments are now heavily relying on computer and Internet systems services. It means that the system should be made highly dependable in terms

of availability, reliability, accessibility and stability. This can only be possible using fault-tolerant platform to model the designing and connectivity of Internet system. To accomplish this, we are going to make use of BFS architectural model for our design.

Faults and Failures in Computer Systems

The terms failure and fault are key to any understanding of system reliability that would crave its dependability. They are often misused. One describes the situation (s) to be avoided, while the other describes the problem (s) to be circumvented. The term fault in its broadest sense indicates a condition that causes a device, component or element to fail to perform in a required manner. The fault may be either physical or logical e.g., a short-circuit, a broken wire, or an intermittent counection.

Failure on its own can be defined in terms of specified service delivered by a system. A system is said to have a failure if the service it delivers to the user deviates from compliance with the system specification for a specified period of time (Adeosun *et al*., 2007). Thus, failure is the deviation of the service delivered by a system from the system specification angle. The specification can be considered as a boundary to the system's region of concern. It is important to note that every system has an explicit specification, which is written and an implicit specification that the system should at least behave.

Faults are better defined in terms of failure (s). A fault is recognized to be the adjudged cause of a failure. The advantages of viewing faults as failures of component/interacting systems are:

- One can consider faults without the need to establish a direct connection with a failure, in that wise, we talk about faults that do not cause failures, i.e. the system is naturally fault tolerant.
- The definition of a fault is the same as the definition of a failure with ouly the boundary of the relevant system or subsystem being different. This means that we can consider an obvious internal defect to be a fault without having to establish a causal relationship between the defect and a failure at the system boundary.

From the aforementioned, a fault will be defined as the failure of

- Any component of the system.
- Any subsystem of the system.
- Any other system which has interacted or is interacting with the considered system.

In this study, we believe, that every fault is a failure from some point of view mentioned above. It is possible for a fault to lead to another faults, or to a failure, or neither (i.e., fault tolerant system).

Errors in Computer Systems

Error is often used in addition to the terms fault and failure to describe the state of a system in computing. In the real sense of it, errors are defined to be the result of faults leading to failures. This is a passive concept associated with incorrect values in the system state. It is extremely difficult to develop unambiguous criteria for differentiating between faults and errors. The connection between error and failure is even more difficult to describe.

Fault Attributes

Observability

Faults normally originate in a system component or subsystem, in the system's enviroument, or in an interaction between the system and a user, operator, or another subsystem. A fault may ultimately have one of several effects:

- It may disappear leaving no perceptible effect.
- It may remain in place with no perceptible effect.
- It may lead to a chain of additional faults that result in a failure in the system's delivered service (i.e., propagation to failure).
- It may lead to a chain of additional faults with no perceptible effect on the system (i.e., undetected propagation).
- It may lead to a chain of additional faults that have a perceptible effect on the system but do not result in a failure in the system's delivered service (i.e., detected propagation without failure).

The first step in fault tolerance is fault detection. Even if there are facilities to prevent a failure by compensating for a fault, it is important to detect and remove faults to avoid the exhaustion of systems fault tolerance resources.

A fault is said to be observable if there is information about its existence and such information should be made available at the system interface. The information indicating the existence of a fault is a symptom. A symptom may be a directly observed fault or failure, or it may be a change in system behaviour such that the system still meets its specifications. A fault tolerated by a fault tolerance mechanism of a system is said to be detected. Otherwise it is latent, whether it is observable or not. The definition of detected is independent of whether or not the fault tolerance mechanism is able to successfully deal with the fault condition. For a fault to be detected, it must be known to the system.

Propagation

A fault is said to be active if it can propagate to other faults or failures. A non-propagating fault is referred to as dormant. When a previously dormant fault becomes active, it is said to be triggered. It is also possible for an active fault to become dormant, awaiting a new trigger. The sequence of faults, each successive one triggered by the preceding one and possibly ending in a failure, is known as a fault trajectory (Fig. 3).

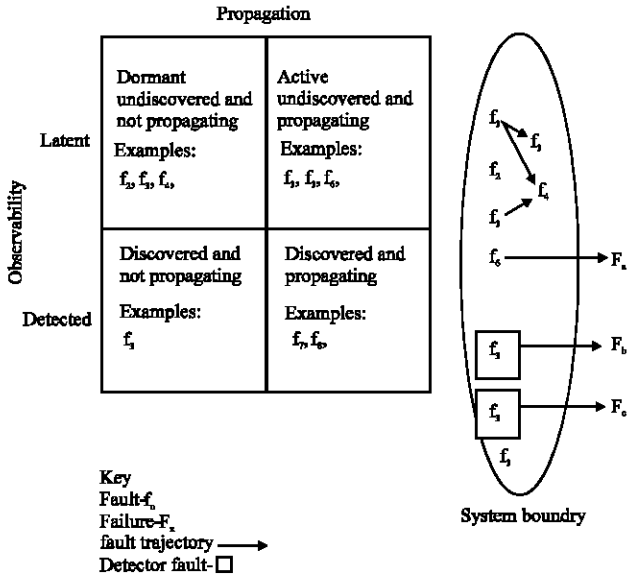

Fig. 3: Relationship between detected, latent, dormant and active faults

Fault Classifications

No system can be made to tolerate all possible faults, so it is essential that the faults be considered throughout the system requirements definition and its design process. To enumerate all of the faults to be tolerated by a system is not possible also; faults must be aggregated into manageable fault classes.

Fault may be classified based on

- Locality (atomic component, composite component, system, operator, envirourment)
- Effect (timing, data)
- Cause (design, damage)

Other possible classification criteria include

- Duration (transient, persistent).
- Effect on System State (crash, amnesia, partial amnesia, etc.)

Since the location of a fault is so important, fault location is a logical starting point for classifying faults.

Locality

An atomic component fault is a fault at the fault floor, that is, in a component that cannot be subdivided for analysis purposes.

For instance, in a computer system, substrate faults can appear in diverse forms. A fault in a memory bit is not an atomic component fault if the details of the memory are below the current span of concern. Such a fault may or may not appear as a memory fault; depending upon the memory's ability to mask bit faults.

A composite component fault is one that arises within an aggregation of atomic components rather than in an atomic component. It may be the result of one or more atomic component faults.

A disk drive failure in a computer system is an example of a composite component failure. If the individual bits of memory are considered to be in the span of concern, a failure of one of those would be a component failure as well.

A system level fault is one that arises in the structure of a system rather than in the system's components. Such faults are usually interaction or integration faults, that is, they occur because of the way the system is assembled rather than because of the integrity of any individual component. We are to note that an inconsistency in the operating rules for a system may lead to a system level fault. System level faults also include operator faults, in which an operator does not correctly perform his or her role in system operation. Systems that distribute objects or information are prone to a special kind of system fault called replication faults. Replication faults occur when replicated information in a system becomes inconsistent, either because replicates that are supposed to provide identical results no longer do so, or because the aggregate of the data from the various replicates is no longer consistent with system specifications. Replication faults can be caused by malicious faults, in which components such as processors lie by providing conflicting versions of the same information to other components in the system. Malicious faults are sometimes called Byzantine faults.

As an example, consider the computer systems in an automobile. Suppose the airbag deployment computer and the antilock brake computer are both known to work properly and yet fail in operation because one computer interferes with the other when they are both present. This would be a system fault.

External Faults

Here, the faults arise from outside the system boundary, the environment, or the user. Environmental faults include phenomena that directly affect the operation of the system, such as temperature, vibration, or nuclear or electromagnetic radiation or that affects the inputs provided to the system. User faults are created by the user in employing the system. Note that the roles of user and operator are considered separately; the user is considered to be external to the system while the operator is considered to be a part of the system.

Effects

Faults may also be classified according to their effect on the user of the system or service. Since computer system components interact by exchanging data values in a specified time and/or sequence, fault effects can be cleanly separated into timing faults and value faults. Timing faults occur when a value is delivered before or after the specified time. Value faults occur when the data differs in value from the specification.

Value Faults

Computer systems communicate by providing values. A value fault occurs when a computation returns a result that does not meet the system's specification. Value faults are usually detected using knowledge of the allowable values of the data, possible determined at run time.

Timing Faults

A timing fault occurs when a process or service is not delivered or completed within the specified time interval. Timing faults cannot occur if there is no explicit or implicit specification of a deadline. Timing faults can be detected by observing the time at which a required interaction takes place; no knowledge of the data involved is usually needed.

Since time increases monotonically it is possible to further classify timing faults into early, late, or never (omission) faults. Since it is practically impossible to determine if never occurs, omission faults are really late timing faults that exceed an arbitrary limit. Systems that never produce value faults, but only fail by omission are called fail-silent system. If all failures require system restart, the system is a fail-stop system.

Duration

Persistent faults remain active for a significant period of time. These faults are sometimes referred to as hard faults. Persistent faults usually are the easiest to detect and diagnose, but may be difficult to contain and mask unless redundant hardware is available. Persistent faults can be effectively detected by test routines that are inter-leaved with normal processing. Transient faults remain active for a short period of time. A transient fault that becomes active periodically is a periodic fault (also known as intermittent fault). Because of their short duration, transient faults are often detected through the faults that result from their propagation.

Immediate Cause

Faults can be classified according to the operational condition that causes them. These include resource depletion, logic faults, or physical faults.

Resource depletion faults occur when a portion of the system is unable to obtain the resources required to perform its task. Resources may include time on a processing or communications device, storage, power, logical structures such as a data structure, or a physical item such as a processor.

Logic faults occur when adequate resources are available, but the system does not behave according to specification. Logic faults may be the result of improper design or implementation. Logic faults may occur in hardware or software.

Physical faults occur when hardware breaks or a mutation occurs in executable software. Most common fault tolerance mechanisms deal with hardware faults.

Ultimate Cause

Faults can also be classified as to their ultimate cause. Ultimate causes are the things that must be fixed to eliminate a fault. These faults occur during the development process and are most effectively dealt with using fault avoidance and fault removal techniques. This is where the pivot of this study is rotating.

A common ultimate cause of a fault is an improper requirements specification which leads to a specification fault. Technically, this is not a fault, since a fault is defined to be the failure of a component/interacting systems and a failure is the deviation of the system from specification (Keidar, 2004). For instance, radiation causing a bit to flip in a memory location would be a value fault which would be considered as an external fault. But, if the fault propagates inside the system boundary the ultimate cause is a specification fault because the system specification did not foresee the problem.

Design Fault

A design fault occurs when the system design does not correctly match the requirements.

Implementation Fault

An implementation fault occurs when the system implementation does not adequately implement the design.

Documentation Fault

A documentation fault occurs when the documented system does not match the real system. The validation process is specifically designed to detect these faults in an attempt to provide viable solutions.

Methodology of the Design

The execution of the work is divided into phases. These include an extensive survey of existing model on Internet counectivity with a view to exploring previous works and to attempt designing a fault-tolerant model for the system. An extensive study of faults, failures and errors on system were made to acquit ourselves with their symptoms on the system. Basic Fault-Tolerant System (BFS) architecture was study in-depth in order to understand the basic rudiments of the fault-tolerant system design so as to get our focus right. Finally, a conceptual design of a fault-tolerant Internet counectivity is done. Generation of a quality test plan for the system and performing the test specification by comparing reliability of fault-tolerant system together with that of non fault-tolerant system.

Correct vs Faulty Processes

A process that does not fail in a run is correct in that run. Otherwise, the process is faulty in the run. Note that a process that fails any time in the run is faulty throughout the entire run.

Threshold Failure Model

Given n processes, it is possible that t out of n processes may fail. t in this case is usually given as a function of n, e.g.,

$$t < n$$
$$2t < n$$
$$3t < n$$

Models of Link Failures

Failures can be attributed to links instead of processes. Links may be reliable or unreliable. In a reliable links, every message sent is eventually delivered at the stipulated destination at the appropriate time while this is not so in the case of unreliable links. Failure types may be crash, loss (omission), timing or Byzantine (Fig. 4).

Conceptual Models of Process Failures

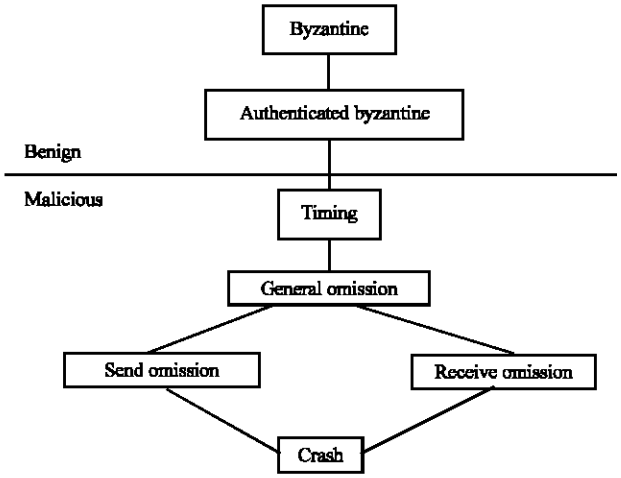

Fig. 4: Models of process failures

Methodology of Fault-Tolerance

Program disruptions can be avoided or corrected by means of protective redundancy.

Protective Redundancy

This is introduced into the computer system in three forms:

- Additional hardware (i.e., hardware redundancy)
- Additional programs (software redundancy)
- Repetition of machine operations (time redundancy)

A program restart (or rollback) is sufficient to correct errors caused by transient faults while replacement or reconfiguration of hardware is needed to eliminate permanent faults from the system. Devices, modules, or entire system complexes are provided in n-modular form to provide hardware redundancy. A correction network compares the results of the n-modules and forms a majority decision as to the correct result. Software redundancy on its own means:

- N-tuple programming of identical software modules and comparison of the various solutions.
- Hierarchically structured system software
- Repeat operations
- Organization of software for protecting programming modules and restart data
- Error-detecting and error-correcting codes

Note that Time redundancy is suitable only for processes that are not critical with respect to time.

BFS-Realization of a Fault-Tolerant Internet Connectivity Architecture

In this study, a fault-tolerant architecture will be implemented by BFS (Basic Fault-Tolerant System), which is based on a multiserver structure with functional decomposition to model the Internet connectivity. It is a partially meshed ring structure with each server linked to the next one and next-but-one neighbour, similar to an optimal diagnosable system (Preparata *et al*., 1977; Kuhl and

Reddy, 1980; Friedman and Simoncini, 1980). This architecture permits decentralized control and a reconfiguration by graceful degradation (Siewiorek, 1977).

Concept of the Architecture

In designing the BFS architecture for Internet connectivity, basic requirements will be made for a high degree of fault tolerance and for increasing availability and reliability cum stability of the overall system (Schmitter, 1988), such as:

- A high degree of intrinsic redundancy for efficient fault elimination without any substantial need for special hardware components.
- High flexibility within the structure to implement necessary redundancies.
- The possibility of majority decisions within the system to improve fault localization.
- Simple limitation of the process being executed in the overall system by functional decomposition of the software modules to their own hardware modules.
- Self-diagnosis of the system with multiple faults.
- Automatic reconfiguration according to the criteria of graceful degradation upon failure of a system component.

BFS Internet Connectivity Conceptual Design

The BFS architecture of a partially meshed ring can be designed (Fig. 5) in which

- Each server is kept informed about the status of its immediately adjacent server (i.e., direct connection).
- If a system component and hence, the corresponding server fails, the overall Internet system and the structure remain operative, because a connection to the next-but-one server always remains.
- The communication paths are bi-directional and each server has two input and two output ports.
- System components are largely homogeneous and no master-slave relationships exist between them.

The chosen structure has features that support a fault-tolerant system. For instance:

- Neighbouring servers in a process allow various forms of majority decision-making without hardware voters.
- Standard servers assure high economic efficiency in both system development and maintenance.
- Functional and modular decomposition of hardware and software within the ring structure improves reliability significantly and simplifies the maintainability of the overall system.
- Utilization of redundancies present in the system and the support of fault diagnosis by the facilities offered by servers greatly increase system reliability and availability.

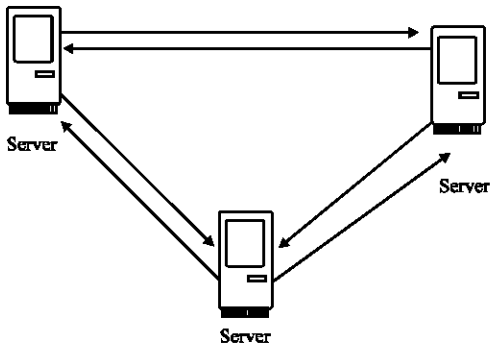

Fig. 5: BFS architecture

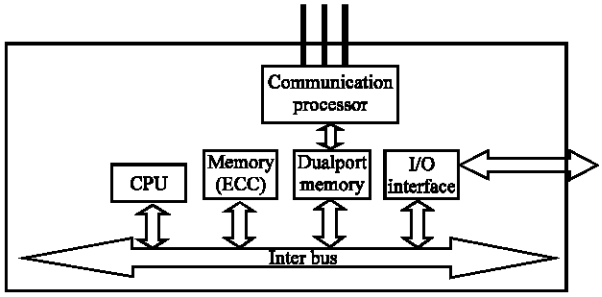

Fig. 6: System component of server for BFS internet connectivity

Hardware Design of BFS

Each server of the BFS architecture comprises the following standard elements (Fig. 6).

- A Central Processing Unit (CPU) with on-board PROM and RAM together with on-die cache memory
- A memory with error detection and correction possibilities
- A communication processor with two input and two output ports and a serial data transfer
- A dual-port RAM
- An interface adequate to its function and its periphery
- An internal bus

Where, there is no central element or global supervisor, each server is autonomous in respect to its functions, its resources and its interfaces.

Additioual hardware such as expensive devices is required for associating one peripheral device to several microcomputer modules, to reconfigure. This peripheral switch is controlled by three modules to get a 2-out-of-3 majority decision about a faulty periphery unit.

In generalization of the interprocessor communication, CPU and the communication processor are separated. This separation makes possible an interface that permits point-to-point communication up to a local distributed microcomputer net (Sauer and Schwaertzel, 1980). The first implementation of BFS possesses point-to-point communication links and its optimal diagnosable architecture (Preparata *et al*., 1977; Friedman and Simoncini, 1980) is also its physical structure.

System Software and Operating System Structures

With this BFS architecture, an operating system with fault-tolerant features is realized that allows true parallel operation. Typical fault-tolerant features are as follows:

- Communication between unit servers is safeguarded and the reaction delay is checked.
- Each server reports its status to its two neighbours.
- The ring structure is preserved despite failure of a neighbouring server.
- Owing to status information reporting, the duties of any server that fails can be either taken over by a neighbour or delegated otherwise.
- Breakpoints can be included in programs to report the status of the server to its neighbouring servers periodically so that in the event of failure, the program can be resumed at the last breakpoint by another server.
- Self-tests are performed when the system is started and at regular intervals thereafter.
- Neighbouring servers are tested at regular intervals.

Table 1: Truth table representing the BFS internet connectivity

Server			
1	2	3	Internet connectivity (Output)
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

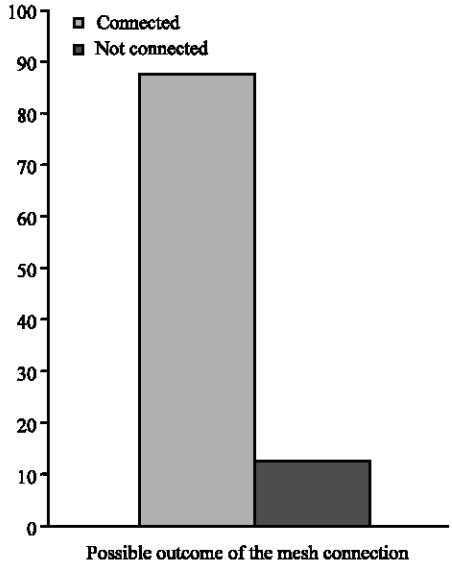

Fig. 7: Availability and non-availability of Internet system

Concept of the Operating System

The operating system is democratic in structure in relation to the nonhierarchical structure of the hardware. Its functions are distributed among two levels: The operating system nucleus and the other operating system functions.

The operating system nucleus has to be permanently correct, because of fault tolerance. It is not yet possible to verify the permanent correctness of complex software, so the operating system nucleus has to be modularized and limited in size so that its correctness can be verified. The portion of this nucleus that is used for starting the system and that embodies the system initialization and self-test and a miniloader, resides in PROM of each server.

The other portion of the nucleus can be loaded and is fetched from the backing storage by the miniloader. This loadable portion embodies the time supervision, the communication routines and the program loader.

The program loader can be used for reloading the other operating system functions; namely, system services, periphery drivers, interrupt servicing and table management. To optimize parallel interserver data communication, a communication system with a certain standard message format that is also suitable for peripherals is provided. To assure the reliable operation of the system, restart, computer configuration and message tables are stored and continuously updated in each server.

Analysis of BFS Internet Connectivity

For any structure, reliability is a function of the number and organization of the interconnections. The system reliability is under the assumption of independence of failure of the servers. In this research, we replicate servers for BFS Internet connectivity.

Table 1 shows the various possibilities of connecting triplicated servers forming a mesh ring. From Table 1 eight different scenarios can be seen and this is gotten from 2n. Seven out of the eight scenarios would enable Internet connectivity while only one will not. That is, connection to the Internet will be established even if just only one server out of the replicated ones is working. Figure 7 shows the availability and non-availability of internet system when triplicated servers are used in Internet connectivity.

RESULTS

The output in Table 1 shows that there will be reliable Internet availability improvement hence the performance of Internet will be enhanced. The ratio of the improvement according to results in Table 1 is 7:1. This will give availability of $\frac{7}{8}\times 100 = 87.50\%$. This percentage of availability achieved would definitely enhance the Internet system reliability by bring it closer than never before to the targeted system reliability of 99.999%. This would make the Internet system more available, accessible, reliable and dependable even at the presence of fault (s). Also, Internet throughput will be enhanced. The Internet system becomes more stable and its scalability improved through connectivity as a result of this high availability.

This study integrates fault-tolerant technology vis-à-vis BFS architectural design into Internet connectivity design using deployment and redeployment of faulty server as platform.

CONCLUSIONS

Present results show that the replicated servers used in Internet connectivity will provide higher and better Internet performance compared to the non-replicated server used in Internet connectivity. This approach as shown in this work will definitely enhance Internet system availability to users even at the presence of failures/faults. Faults and failures can now be masked when noticed with the presence of spares of servers (i.e., replicated servers). One of the spares will just take over if the one in use is faulty.

If well implemented, Internet system becomes more reliable and self-stabilized by local checking and correction. This is the characteristic of a fault-tolerant system that enhances Internet service performance.

REFERENCES

Adeosun, O.O., 2003. The advent of information technology. J. Pure Applied Sci., 4: 67-73.

Adeosun, O.O., E.R. Adagunodo, I.A. Adetunde and T.H. Adeosun, 2007. Strategic modeling and analysis of internet connectivity using markov process to enhance internet performance. Asian J. Inform. Technol., 6: 379-384.

Friedman, A.D. and L. Simoncini, 1980. System-level fault diagnosis. Computer, 13: 47-53.

Helal, A., A. Heddaya and B. Bhargava, 1996. Replication Techniques in Distributed Systems. Kluwer Academic Publishers, pp: 231-240.

Keidar, I., 2004. Principles of Reliable Distributed Systems. Technion EE, pp: 1-30.

Kuhl, J.G. and S.M. Reddy, 1980. Some extensions to the theory of system level fault diagnosis. Digest of Papers: FTCS, 17: 291-296.

Preparata, F.P., G. Metze and R.T. Chien, 1977. On the connection assignment problem of diagnosable systems. IEEE Trans. Electron. Comput., 16: 848-854.

Sauer, A.M. and H.G. Schwaertzel, 1980. A local distributed microprocessor net with decentralized access to an option bus. Proceedings of Compcon.

Schlichting, R.D. and F.B. Schneider, 1983. Fail-stop processors: An approach to designing fault-tolerant computing systems. ACM Trans. Comput. Syst., 1: 222-238.

Schmitter, E., 1988. Structure Principles for Fault-Tolerant Multiprocessor Systems. Siemens Forschnngs-und Entgwicklungs-Berichte, 7: 328-333.

Siewiorek, D.P., 1977. Multiprocessors: Reliability modeling and graceful degradation. Infotech State of the Art Conference: System Reliability, pp: 47-73.

Stern, H. and E. Marcus, 2000. Blueprints for High Availability: Designing Resilient Distributed Systems. Wiley.