Sub-Sahara African Academic
Research Publications
December, 2020, Editions

JPAS
Journal of Pure and Applied Science
Vol..17, No.9, ISSN 2318-9315

RELIABILITY MODELS AND APPLICATIONS IN DISTRIBUTED SYSTEMS

VICTOR IBOYI[1], FATIMA SHITTU[2], MUSTAPHA LAWAL ABDULRAHMAN[3]
[1,2]Department of Computer Science, Federal Polytechnic, Damaturu
[3]Mathematical Science, Abubakar Tafawa Balewa University Bauchi, Nigeria

ABSTRACT
One of the potential advantages of distributed systems is improved reliability and availability. This, however, is not a feature that comes automatically. It is important that mechanisms be provided to ensure the consistency of the database as well as to detect failures and recover from them. The implication for Distributed Database Systems (DDBS)s is that when a failure occurs and various sites become either inoperable or inaccessible, the databases at the operational sites remain consistent and up to date. Furthermore, when the computer system or network recovers from the failure, the DBSs should be able to recover and bring the databases at the failed sites up-to-date. This may be especially difficult in the case of network partitioning, where the sites are divided into two or more groups with no communication among them. In this mini project, we discuss the various reliability protocols operational in a distributed database system. We also consider different types of reliability models and then implement and test the validity of a two-phase commit protocol in an object oriented environment.

Keywords: *Reliable distributed database, Reliability protocol, 2-Phase Commit model, failure model, poisson distribution*

Introduction
A reliable distributed database management system is one that can continue to process user requests even when the underlying system is unreliable. In other words, it is a fault tolerant and a fault forecasting

system (Ahmed and Wu, 2013) . This means that even when components of the distributed computing environment fail, a reliable distributed DBMS should be able to continue executing user requests without violating database consistency. State based reliability model includes to a system that consists of a set of components with a State, which changes as the system operates. The behaviour of the system in providing response to all the possible external stimuli is laid out in an authoritative specification of its behaviour. The specification indicates the valid behaviour of each system state. Any deviation of a system from the behaviour described in the specification is considered a failure (Katan, 2014). Each failure obviously needs to be traced back to its cause. Failures in a system can be attributed to deficiencies either in the components that make it up, or in the design, that is, how these components are put together. Each state that a reliable system goes through is valid in the sense that the state fully meets its specification. However, in an unreliable system, it is possible that the system may get to an internal state that may not obey its specification. Further transitions from this state would eventually cause a system failure. Such internal states are called erroneous states; the part of the state that is incorrect is called an error in the system (Boudali, Crouzen, & Stoelinga, 2007). Any error in the internal states of the components of a system or in the design of a system is called a fault in the system. Thus, a fault causes an error that results in a system failure (Ahmed and Wu, 2013). It is the concern of this work to outline the reliability protocols of a distributed DBMS, implement and test the validity of the protocol using a two-phase commit methodology. The Sun Microsystem's Java programming language was used.

Overview of Fundamental Concepts

A. Reliability: refers to the probability that the system under consideration does not experience any failures in a given time interval. Reliability of a system or component R (t) is defined as the probability that the system or component can perform a required function under given condition for the time interval [0,t] (Troubitsyna and Tarasyuk, n.d.)

$R(t) = Pr\{0 \text{ failures in time } [0,t] | \text{no failures at } = 0\}$

If we assume that failures follow a Poisson distribution (Zhu and Pham, 2018), which is usually the case for hardware, this formula reduces to
R(t) =Pr{0 failures in time [0,t]}
Under the same assumptions, it is possible to derive that

$$\Pr\{k \text{ failures in time } [0, t]\} = \frac{e^{-m(t)}[m(t)]^k}{k!}$$

Where $$m(t) = \int_0^t z(x)dx$$

Here z (t) is known as the hazard function (Pham and Pham, 2019), which gives the time-dependent failure rate of the specific hardware component under consideration. The probability distribution for z (t) may be different for different electronic components.

B. Availability: A(t), refers to the probability that the system is operational according to its specification at a given point in time t (Hößler, Scheuvens, Franchi, Simsek, & Fettweis, 2017). A number of failures may have occurred prior to time t, but if they have all been repaired, the system is available at time t. Obviously, availability refers to systems that can be repaired.

The repair rate expresses the probability that a system failed for a time t, and recovers its ability to perform its function in the next time unit (Troubitsyna and Tarasyuk, 2015).

Repair rate of the system is defined by

$$\mu(t) = \frac{g(t)}{1 - M(t)}$$

where g(t) is the repair probability density function and M(t) is the system maintainability

C. Failure: The deviation of a system from the behavior that is described in its specification.

D. Erroneous state: The internal state of a system such that there exists circumstances in which further processing, by the normal algorithms of the system, will lead to a failure which is not attributed to a subsequent fault.

E. Error: The part of the state which is incorrect.

F. Fault: An error in the internal states of the components of a system or in the design of a system.

G. Commit Protocols: In distributed data base and transaction systems a distributed commit protocol is required to ensure that the effects of a

distributed transaction are atomic, that is, either all the effects of the transaction persist or none persist, whether or not failures occur. Several commit protocols have been proposed and studied over the years. These include 1-phase commit (1PC), 2-phase commit (2PC), 3-phase commit (3PC) or quorum-based commit QBC.

i. 1-Phase Commit: is the simplest commit protocol. Each site locally completes its transaction, then send "Done" to the coordinator and wait for "Commit" or "Abort" from the coordinator. After receiving messages from all sites, it then makes the final decision to "Commit" or "Abort".

ii. 2-Phase Commit: reduces the vulnerability of one-phase commit protocols. In the basic form of 2PC, there is a coordinator and subordinates where the coordinator is the site that has initiated the transaction. In the first stage, the coordinator tries to get a uniform decision of either committing or aborting out of the subordinates and in the second stage, the coordinator relays the decision back to them.

iii. 3-Phase Commit: is similar to 2-phase commit. The difference between 3PC and 2PC can be seen when the coordinator fails before sending "prepare commit" message. In 2PC, the subordinates will wait indefinitely until the coordinator comes back again, but in 3PC, a new coordinator is chosen.

iv. Quorum Based Commit is the solution to 3PC's problem as it does not guarantee consistency in its basic form. The solution is to use a quorum of subordinates and the coordinator. If the coordinator fails and a new coordinator is chosen, the coordinator starts collecting information from the subordinates (Chase, 2017). An addition to the 3PC is the "prepare to abort." Unlike the original 3PC, sites that respond to "prepare to abort" can still commit later on. What "prepare to abort" means is that the site wants to abort but in the same case it doesn't have to and can commit if the situation requires it. This is possible because the quorum is built after all sites have researched "prepare" state and in this state, the sites do not care if they commit or abort. Quorum-based protocol also introduced VA and VC, where these are the

constants that are defined during the transaction. If V is the total number of sites in a transaction, then VA+VC=V+1 to ensure that commit or abort in performed.

Termination protocol

This is intended to terminate a transaction when one or more sites fail. This protocol is invoked by all operational sites after a failure of a site is detected. The termination protocol tries to commit or abort the transaction after a failure as soon as possible to release data resources locked by the sites involved in the transaction. There are two termination protocols that are activated when failures occur. The first protocol is activated by the coordinator when one or more slaves fail. The second protocol is activated by the slaves when the coordinator fails.

Recovery protocols

Protocols that specify the steps that a failed site must take to terminate the transaction it was running at the time of the failure after it is repaired. The recovery protocols must make sure that a failed site terminates its transaction in exactly the same way the working sites terminated it. There are two possible ways of doing this.

Recovery by Discovery: The first approach requires the repaired site to ask around about the fate of the transaction. The site then terminates the transaction accordingly. If the failed site is a slave, it can ask the coordinator about the transaction. If the failed site is the coordinator, it can broadcast the question to all slaves. Either all the other slaves or the newly elected coordinator will send the answer to the requesting site. In order for this protocol to work, the information about the termination of a transaction must be held by each site until all failures are repaired.

Independent Recovery: The second approach relies on the site and the state of the site at the time of the failure to terminate the transaction properly. A site can determine the state it was in when the failure happened by examining its local log. In the following, we discuss the recovery process for the coordinator and the slaves based on this approach.

Reliability Protocol Overview

One of the critical functions that database systems have to ensure is correctness and availability. During the course of the database operations, the database system may stop running or some transactions may have to be aborted before the transactions commit. In those situations, atomicity and durability would be compromised if a committed transaction is not written to the disk or if aborted transaction is written to the disk (Kemper & Neumann, 2011). It is the role of the transaction manager to guarantee correctness of database system - all actions in the transaction happen or none happen and if a transaction commits then its effects persist. Furthermore, recovery manager has to be efficient enough so that downtime of the database is minimized and the availability of the database is maximized.

Earlier in (Braden et al., 1998) mention has been made that to ensure correctness, a trivial solution would be to use no steal-force method. In no steal-force method, the effects of transactions are held in the buffer pool until the transaction commits and after the transaction commits the effects are forced (written) to the disk. There are two problems with this method. In a long transaction, no steal prevents the content of the buffer pool to be replaced until the transaction commits. This will lead to poor throughput in the database system as fewer items get to be placed into the buffer manager and disk has to be continuously accessed. Force writes requires writes to the disk after the transaction but this leads to poor response time. For example, if there is a page that is updated frequently, the page has to be forced to the disk at every update. It would be more efficient to have the page in the buffer pool and then write to the disk after most of the updates are finished. However, force is necessary to guarantee durability because what would happen if the database were to crash before writing the data to the disk? So, a desired method is steal-no force method that will also guarantee atomicity and durability.

Methodology

Figures 1& 2 below show technique and framework used in this work, the description follows in the coming subsections

Two-Phase Commit

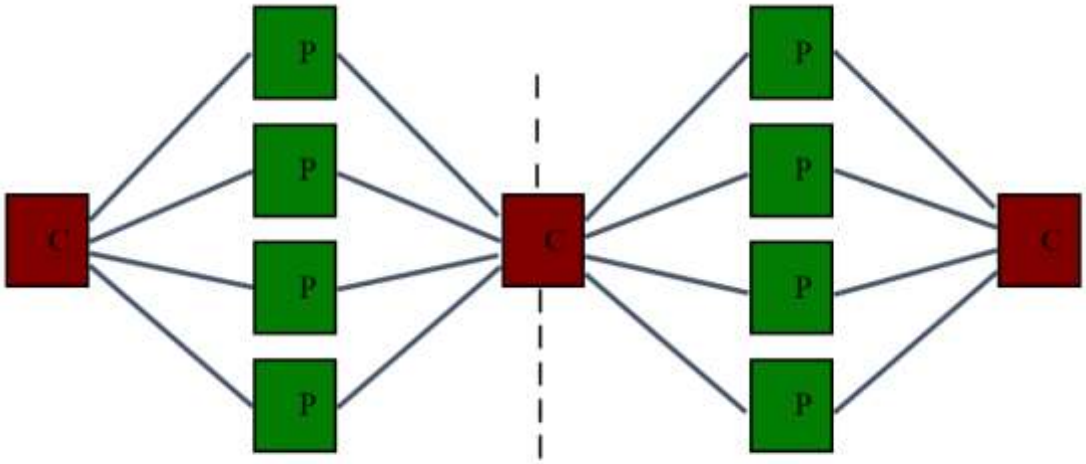


Figure 1: Centralized 2PC

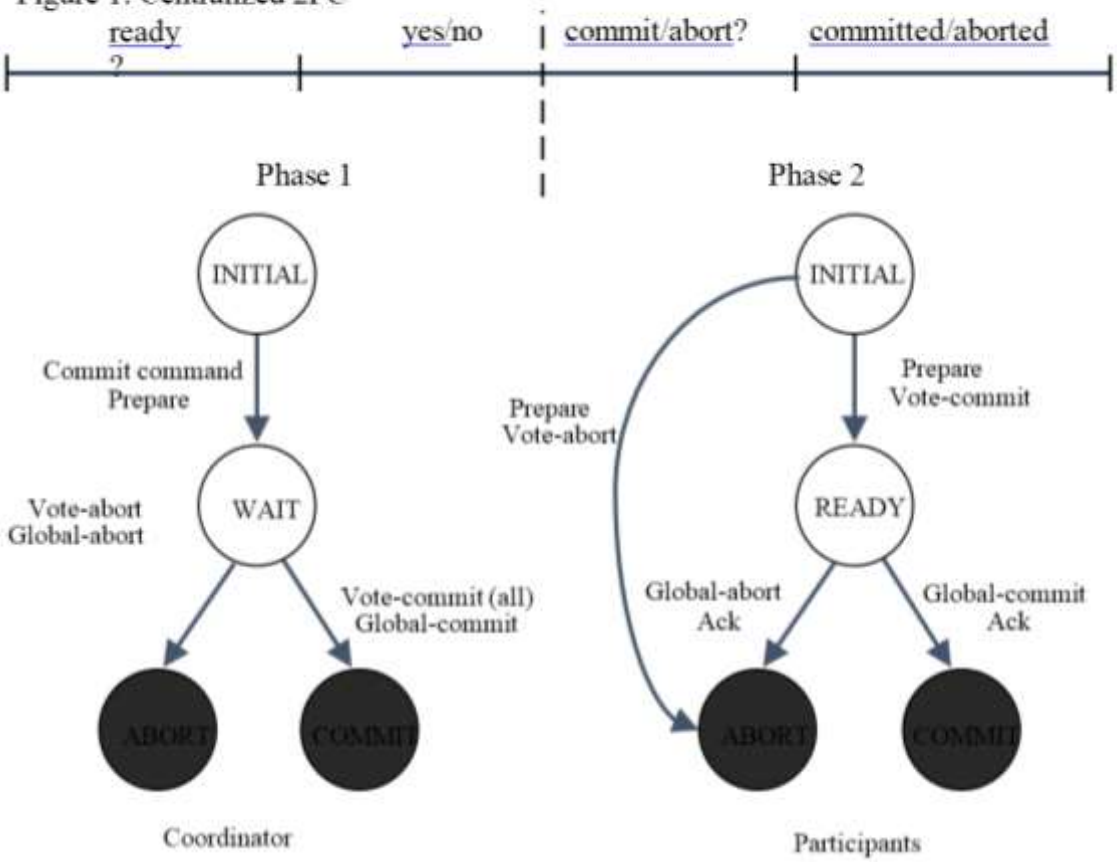


Figure 1: 2PC State Transition

Algorithm used

The algorithm of the 2-phase commit used in this work is depicted below

```
begin
repeat
        wait for an event ;
        switch event do
        case Msg Arrival
                Let the arrived message be msg ;
                switch msg do
                case Commit \\commit command from scheduler
                        write begin commit record in the log ;
                        send "Prepared" message to all the involved participants ;
                        set timer
                case Vote-abort \\one participant has voted to abort-unilateral abort
                        write abort record in the log ;
                        send "Global-abort" message to the other involved participants ;
                        set timer
                case Vote-commit
                        update the list of participants who have answered ;
                if all the participants have answered then \\all must have voted
                                                                        \\ to commit
                    write commit record in the log ;
                    send "Global-commit" to all the involved participants ;
                    set timer
                case Ack
                        update the list of participants who have acknowledged ;
                        if all the participants have acknowledged then
                            write end of transaction record in the log
                        else
                            send global decision to the un-answering participants
        case Timeout
            execute the termination protocol
    until forever;
end
```

2PC Coordinator Termination Protocol

The termination steps that the coordinator takes after the failure of a slave has been detected depending on the state that the coordinator is in at the time of the failure detection.

The coordinator handles the failure of a slave as follows:

Case 1: The slave's failure is detected when the coordinator is in the "Before Commit" state. In this case, the slave must have failed during transaction

execution. To terminate the transaction, the coordinator will log "Aborting" and send a "Global Abort" message to all slaves.

Case 2: The slave's failure is detected when the coordinator is in the "Preparing" state. In this case, the slave could have been in any of the "Before Commit," "Aborting," or "Prepared" states. If the slave failed in the "Before Commit" state, it never received the "Prepare" message. If the slave failed in the "Abort" state, it must have received the "Prepare to Commit" message and voted "Not Ready." Finally, the slave must have failed in the "Prepared" state, if it voted "Ready" but failed before it sent out the message. In any case, the decision by the coordinator is to assume that the slave was not initiated to commit the transaction, which necessitates the global abort process.

Case 3: The slave's failure is detected when the coordinator is in the "Committing" state. This signifies the fact that the coordinator did not receive the "Commit ACK" message from the slave. In this case, the slave must have failed in either the "Prepared" state—it failed before it received the "Global Commit" message—or in the "Committed" state—it failed after committing and before it sent out the "Commit ACK" message. In either case, the coordinator has to continue polling the slave for acknowledgment of the action before it ends the transaction globally.

Case 4: The slave's failure is detected when the coordinator is in the "Aborting" state. This signifies the fact that the coordinator did not receive the "Abort ACK" from the slave. In this case, the slave could have been in any of the "Before Commit," "Prepared," "Aborting," or "Aborted" states. Regardless of the state in which the slave failed, the coordinator must poll the slave for acknowledgment of the abort before it ends the transaction globally.

2PC Slave Termination Protocol:

Each slave detecting the coordinator's failure activates this protocol to terminate the current transaction. Each slave that detects the failure of the coordinator takes the following steps:

Case 1: The coordinator's failure is detected when the slave is in the "Before Commit" state. In this case, the coordinator must have failed when it was in any of the "Before Commit," "Aborting," or "Preparing" states. In this

case, the slaves elect a new coordinator which will abort the transaction globally.

Case 2: The coordinator's failure is detected when the slave is in the "Aborting" state. In this case, the coordinator could have been in the "Preparing" or "Before Commit" state. In either case, the transaction needs to be aborted, which is achieved by slaves deciding to elect a new coordinator.

Case 3: The coordinator's failure is detected when the slave is in the "Prepared" state. In this case, the coordinator could have failed in any of the "Preparing," "Aborting," or "Committing" states. The slaves deal with the "Committing" and "Aborting" states of the coordinator similarly, and, therefore, we will consider only two cases. Let's discuss these cases in more detail. In the first case, the coordinator dies in the "Preparing" state before it sends out the "Global Abort" or the "Global Commit" message. The coordinator that dies in the "Preparing" state and does not get a chance to send out the "Global Abort" or "Global Commit" message leaves the slaves in the dark about the final decision on the transaction. This is possible when the coordinator makes the decision to commit (or abort) the transaction and makes the transition to the "Committing" (or "Aborting") state, but before the message leaves the site's message queue, the site fails. Keeping in mind that the coordinator's decision could have been applied to the local copy of the database at the coordinator's site, the slaves cannot do anything about this transaction—therefore, they are blocked. In the second case, the coordinator dies in the "Committing" (or "Aborting") state. For this case, it is possible that the message to globally commit the transaction did not make it out of the coordinator's site or that it did make it to one or more slaves. If the message did not make it out of the coordinator's site, none of the slaves will know about the decision. On the other hand, if the message made it out of the coordinator's site before the site failed, one or more of the slaves may know the decision. For the latter case, after the failure is detected, the slaves elect a new coordinator and try to discover the coordinator's state at the time of the failure. To do so, the new coordinator sends a message to all other slaves asking them to reply with the last message they received from the failed coordinator.

2PC Coordinator Recovery Process

After a coordinator is repaired, it will read the local log, determine the state it was in at the time of the failure, and take the necessary steps for recovery.

Here are the possible cases:

The coordinator was in the "Before Commit" state when it failed. In this case, the coordinator did not send the decision to commit or abort the transaction to slaves. The termination protocol for this case forces the slaves to abort the transaction. Therefore, the coordinator must also abort the transaction. As part of this process, the coordinator will send an "Abort ACK" to the newly elected coordinator.

The coordinator was in the "Preparing" state when it failed. In this case, according to the termination protocol the slaves are blocked. The coordinator restarts the voting process to terminate the transaction. This is necessary since some of the slaves' responses may have been lost in the coordinator's message queue when the site failed.

The coordinator was in the "Aborting" state when it failed. In this case, the decision was to abort the transaction because either one or more slaves voted to abort, or the coordinator decided to abort. The recovery process in this case requires the coordinator to communicate with the new coordinator, which was elected as part of the termination protocol. Two sub-cases are possible. If all the slaves were ready to commit, but the coordinator had decided to abort, and none of the slaves had received the "Global Abort" message, then the slaves are blocked. For this sub-case, the coordinator will have to inform all slaves that the transaction is being aborted by sending a "Global Abort" message to terminate the transaction. In the second sub-case, one or more slaves had received the "Global Abort" message. Here, the new coordinator has aborted the transaction according to the termination protocol. Therefore, the repaired coordinator will simply abort the transaction and acknowledge the action to the new coordinator.

The coordinator was in the "Committing" state when it failed. In this case, the decision was to commit the transaction, and all slaves were ready to do so. The recovery process in this case requires the coordinator to communicate with the new coordinator, which was elected as part of the

termination protocol. Two cases are possible. If the "Global Commit" message was received by one or more slaves, then the new coordinator has committed the transaction. If the "Global Commit" message was not received by any transaction, then the slaves are blocked. If the slaves are blocked, the old coordinator will commit the transaction globally. If slaves have already committed the transaction, then the old coordinator commits the transaction as well.

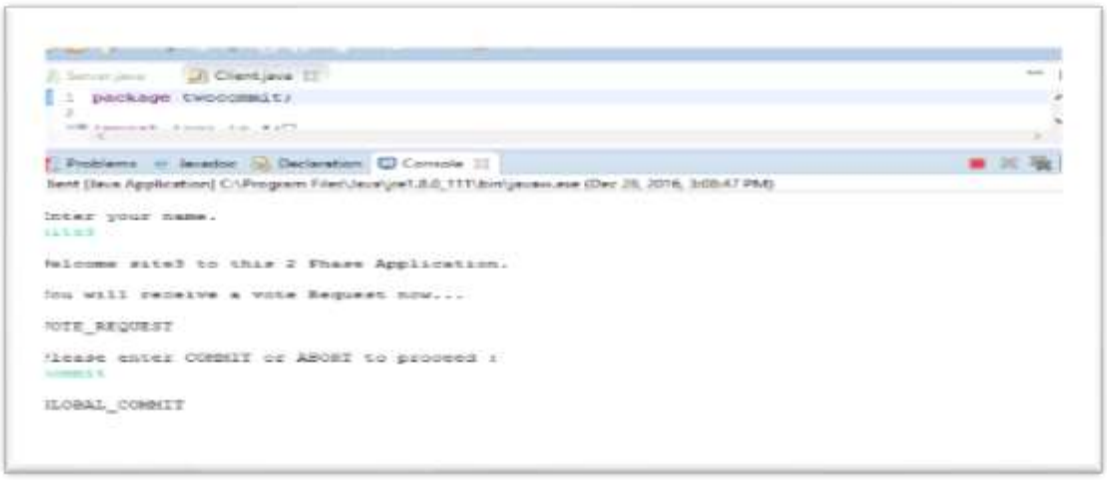
The coordinator was in the "Aborting" state when it failed. In this case, the decision was to abort the transaction. Again, either the slaves are blocked or they have aborted the transaction. The recovery process in this case requires the coordinator to communicate with the new coordinator, which was elected as part of the termination protocol. If the "Global Abort" message was received by one or more slaves, then the new coordinator has aborted the transaction. In this case, the old coordinator will abort the transaction as well. If slaves are blocked, the old coordinator will abort the transaction globally.

Recovery in the states "Committed" and "Aborted" does not require any action from the coordinator.

Implementation and Results

The 2-phase commit protocol was implemented using java programming language introduced by Sun Microsystems. The language possesses features, which make it attractive for using in computational modeling. Java is a simple language (simpler than C++). It has rich collection of libraries implementing various APIs (Application Programming Interfaces). With Java it is easy to create Graphical User Interfaces and to communicate with other computers over a network. Java has built-in garbage collector preventing memory leaks. Another advantage of Java is its portability. Java Virtual Machines (JVM) are developed for all major computer systems. JVM is embedded in most popular Web browsers. Java applets can be downloaded through the internet and executed within Web browsers. Useful for object-oriented design Java features are packages for organizing classes and prohibition of class multiple inheritance. This allows cleaner object-oriented design in comparison to C++. Despite its attractive features, Java is rarely used in finite element analysis. Just few

publications can be found on object-oriented Java finite element codes. Figures 4 & 5 depict the result obtained after simulating the work in the java environment.



Conclusions

Failures in a system can be attributed to deficiencies either in the components that make it up, or in the design, that is, how these components are put together. Each state that a reliable system goes through is valid in the sense that the state fully meets its specification. One of the potential advantages of distributed systems is improved reliability and availability. This, however, is not a feature that comes automatically. It is important that mechanisms be provided to ensure the consistency of the database as well as to detect failures and recover from them. The implication for DDBSs is that when a failure occurs and various sites

become either inoperable or inaccessible, the databases at the operational sites remain consistent and up to date. Furthermore, when the computer system or network recovers from the failure, the DBSs should be able to recover and bring the databases at the failed sites up-to-date. This may be especially difficult in the case of network partitioning, where the sites are divided into two or more groups with no communication among them. In this mini project, we considered different types of reliability models and we implemented a two-phase commit protocol. The simulation results have demonstrated further the significance of reliability models and its applications in distributed systems

References

Ahmed, W. and Wu, Y. W, (2013). A survey on reliability in distributed systems. Journal of Computer and System Sciences, 79 (8), 1243-1255. Elsevier. doi:10.1016/j.jcss.2013.02.006.

Braden, B., Clark, D. D., Crowcroft, J., Davie, B. S., Deering, S., Estrin, D., . . . Partridge, C. (1998). Recommendations on Queue Management and Congestion Avoidance in the Internet. RFC, 2309, 1-17.

Boudali, H., Crouzen, P., & Stoelinga, M. (2007). Dynamic fault tree analysis using input/output interactive markov chains. Paper presented at the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07).

Chase, J. (2017). Distributed Systems, Failures, and Consensus. Retrieved from cs.duke.edu.

Hößler, T., Scheuvens, L., Franchi, N., Simsek, M., & Fettweis, G. P. (2017). Applying reliability theory for future wireless communication networks. Paper presented at the 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC).

Katan, D. (2014). Translating cultures: An introduction for translators, interpreters and mediators. Routledge.

Kemper, A., & Neumann, T. (2011). HyPer: A hybrid OLTP & OLAP main memory database system based on virtual memory snapshots. Paper presented at the 2011 IEEE 27th International Conference on Data Engineering.

Pham, T, and Pham, H. A. (2019). Generalized software reliability model with stochastic fault-detection rate. Ann Oper Res 277, 83–93. doi:10.1007/s10479-017-2486-3

Troubitsyna, E. and Tarasyuk, A. (2015). Software Safety. Lecture 8 on system reliability. Abo Akademi University. Also part of 34th International Conference on Computer Safety, Reliability and Security (SAFECOMP2015), Lecture Notes in Computer Science 9337, 29–43, Springer International Publishing Switzerland, 2015.

Zhu, M. and Pham, H. (2018). A two-phase software reliability modeling involving with software fault dependency and imperfect fault removal. Computer Languages, Systems & Structures, 53, 27-42. doi:10.1016/j.cl.2017.12.002.