UNIVERSITY DATA RECOVERY SYSTEM USING SEED BLOCK ALGORITHM

MYAT PHOO NGE

M.C.Sc. **SEPTEMBER 2022**

UNIVERSITY DATA RECOVERY SYSTEM USING SEED BLOCK ALGORITHM

By

Myat Phoo Nge

B.C.Sc.

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Computer Science

(M.C.Sc.)

University of Computer Studies, Yangon

September 2022

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my sincere thanks to those who helped me with various aspects of conducting research and writing this thesis. To complete this thesis, many things are needed like my hard work as well as the supporting of many people.

First and foremost, I would like to express my deepest gratitude and my thanks to **Dr. Mie Mie Khin,** Rector, the University of Computer Studies, Yangon, for her kind permission to submit this thesis.

I would like to express my appreciation to **Dr. Si Si Mar Win and Dr. Tin Zar Thaw,** Professor, Faculty of Computer Science, University of Computer Studies, Yangon, for their superior suggestions, administrative supports and encouragement during my academic study.

My thanks and regards go to my supervisor, **Dr. Yu Wai Hlaing,** Lecturer, Faculty of Computer Science, University of Computer Studies, Yangon, for her support, guidance, supervision, patience and encouragement during the period of study towards completion of this thesis.

I also wish to express my deepest gratitude to **Dr. Mya Thandar Aung**, Assistant Lecturer, Department of English, University of Computer Studies, Yangon, for her editing this thesis from the language point of view.

Moreover, I would like to extend my thanks to all my teachers who taught me throughout the master's degree course and my friends for their cooperation.

I especially thank to my parents, all of my colleagues, and friends for their encouragement and help during my thesis.

STATEMENT OF ORIGINALITY

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

…… …..…….

Date Myat Phoo Nge

ABSTRACT

Large amount of data is generated in electronic form. The data recovery services are required to maintain this data very efficiently. Since organizations (include universities) have become very dependent on digital data processing, a breakdown may disrupt the business' regular routine and stop its operation for a certain amount of time. In order to prevent data loss and minimize disruptions there have to be well-designed file backup and recovery procedures. The recovery process can rebuild the system when it goes down. In this paper we have proposed a smart remote data backup algorithm, Seed Block Algorithm (SBA). The main aim of proposed algorithm is two types: the first one is to help the users to collect information from any remote location in the non-presence of or loss of network connectivity. Another one is to recover the files if files get deleted mistakenly or if the server gets destroyed due to any reason. The time related problems are also solved by proposed SBA such that it will take minimum time for the recovery procedure. Without using any of the existing encryption techniques, the proposed paper concentrates on security features for the backup files stored at remote servers. This system is implemented by using ASP.Net (C#) programming language with Microsoft SQL Server 2008.

Keywords: data recovery, smart remote data backup algorithm, SBA

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Nowadays large amount of data is stored in the storage/ backup and becoming very important to all the organization. Because it is the age of technology, almost all organizations are based on transaction processing. So, data backup and recovery are very important for data reliability and data availability.

Since organizations such as universities have become very dependent on digital data processing, a breakdown may disrupt the business' regular routine and stop its operation for a certain amount of time. To prevent data loss and minimize disruptions, there must be well-designed file backup and recovery procedures. The recovery process can rebuild the system when it goes down. The data recovery services are required to maintain this data very efficiently.

Every university, there has a lot of data about students, teachers, staff, and other school related activities. As technology advances with the times, many changes are needed in everyday lives. Education is very important for all ages. Therefore, all the information about university should have backup and recovery planning. In “University Data Recovery System”, Excel, Power Point and Microsoft Word Files which are mostly used in university have been recovered in a short time using Seed Block Algorithm. When the system breaks down, it stops the regular routines of the business and stops its operation for a certain amount of time. To recover incomplete transactions, to prevent data loss, and to minimize disruption, the well-designed backup and recovery procedure is put into use. This system uses a very efficient algorithm for data backup called **Seed Block Algorithm (SBA)**. It is one of the smart data backup algorithms for remote data access. The contribution of the proposed SBA is twofold; 1-SBA helps the users to collect information from any remote area in the failure of network connection. 2-Recover the files in case if it gets deleted due to any reason like by mistake or intentionally or if the main server gets destroyed. As a result, data which is accurate and available when needed may improve the customer’s satisfaction.

1.1 Objective of Thesis

The objectives of thesis are as follows:

- To prevent data loss in University Data Recovery System
- To restore the database to a previous consistent state when the current state is inconsistent
- To deliver accurate data and timely services to the users
- To understand how data consistency is important in database system
- To implement a data reliable system by using multiple backup servers
- To support data availability (anytime, anywhere) by the aid of backup servers
- To provide the flexibility for the user to recover their lost data from any backup server

1.2 Related Work

Exchange Handling Framework (TPS) is the information lifeline for a specific business association since it is the wellspring of information for data frameworks like MIS (The board Data Framework) and DSS (Choice Emotionally supportive networks). TPS can be utilized for the different associations, where information plays the most noteworthy need. In the improvement of Software engineering, exchange handling framework are advancing in numerous areas, for example, carrier reservation, banking, inn, the travel industry, industry, shopping center and medical services. This framework is quick and dependable enough to deal with the hourly bank exchanges [4].

Exchange The board Framework (TMS) gave admittance to their transmission framework to advance culmination in discount power market [5]. Their TMS is imagined mechanizing and incorporate the market interface and to facilitate security processes. Their framework upheld the administration of transmission and subordinate assistance is quick and dependable enough to deal with the hourly market exchanges. Exchanges are without a doubt the unit of recuperation [10]. Business associations for the most part use recuperation strategies for information security and dependability.

1.3 Organization of the Thesis

This thesis is organized in five chapters. In Chapter 1, the introduction of the system, objectives and background theory of the transaction processing system (TPS) used in this system. In Chapter 2 presents the Transaction Processing System and the data recovery system. Chapter 3 describes the data recovery procedures. Chapter 4 expresses the design and implementation of the system. Finally, Chapter 5 presents the conclusions of this thesis, limitations and further extension of the system.

CHAPTER 2

THEORICAL BACKGROUND

Since business organizations have become very dependent on transaction processing, a breakdown may disrupt the business' regular routine and stop its operation for a certain amount of time. In order to prevent data loss and minimize disruptions there have to be well-designed backup and recovery procedures. The recovery process can rebuild the system when it goes down.

2.1 Recovery Process

A Transaction Processing System (TPS) may fizzle for some reasons, for example, framework disappointment, human blunders, equipment disappointment, erroneous or invalid information, PC infections, programming application mistakes or regular or man-made fiascos. As it's impractical to forestall all disappointments, a TPS should have the option to recognize and address blunders when they happen and adapt to disappointments. A TPS will go through a recuperation of the information base which might include the reinforcement, diary, designated spot, and recuperation director [4]:

- Journal: A journal keeps a review trail of exchanges and information base changes. Exchange logs and Database change logs are utilized; an exchange log records every one of the fundamental information for every exchange, including information values, season of exchange and terminal number. An information base change log contains when duplicates of records that have been changed by exchanges.

- Checkpoint: The reason for check pointing is to give a preview of the information inside the data set. A designated spot, by and large, is any identifier or other reference that distinguishes the condition of the data set at a specific moment. Changes to data set pages are acted in memory and are not really written to circle after each update. Subsequently, occasionally, the data set framework should play out a designated spot to compose these updates which are held in-memory to the capacity plate. Composing these updates to capacity circle makes a moment in which the data set framework can apply the

changes contained in an exchange log during recuperation after a surprising shut down or crash of the data set framework. In the event that a designated spot is interfered with and recuperation is required, the data set framework should begin recuperation from a past effective designated spot. Checkpointing can be either exchange steady or non-exchange reliable (called additionally fluffy checkpointing). Exchange predictable checkpointing produces a persevering data set picture that is adequate to recuperate the information base to the express that was remotely seen right now of beginning the checkpointing. A non-exchange steady checkpointing brings about a determined information base picture that is lacking to play out a recuperation of the data set state. To play out the data set recuperation, extra data is required, normally contained in exchange logs. Exchange reliable checkpointing alludes to a predictable data set, which doesn't be guaranteed to incorporate every one of the most recent committed exchanges, however all changes made by exchanges, that were committed at the time designated spot creation was begun, are completely present. A non-predictable exchange alludes to a designated spot which isn't really a reliable data set, and can't be recuperated to one without all log records produced for open exchanges remembered for the designated spot. Contingent upon the kind of data set administration framework executed a designated spot might consolidate files or capacity pages (client information), records and capacity pages. Assuming that no files are integrated into the designated spot, records should be made when the data set is reestablished from the designated spot picture.

- Recovery Manager: A recuperation director is a program which reestablishes the data set to a right condition which permits exchange handling to be restarted.

Depending on how the system failed, there can be two different recovery procedures used. Generally, the procedure involves restoring data that has been collected from a backup device and then running the transaction processing again.

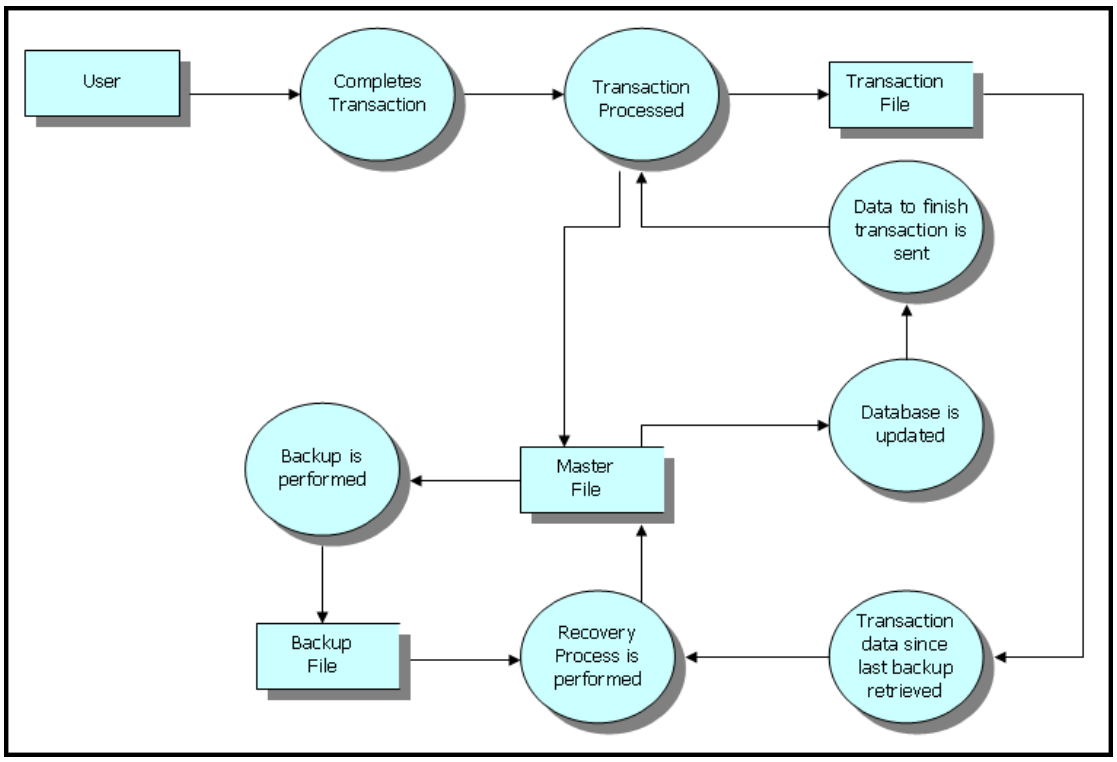


Figure 2.1 A Dataflow Diagram of Backup and Recovery Procedures

2.2 Backup

In information technology, a backup, or the process of backing up, refers to the copying and archiving of computer data so it may be used to *restore* the original after a data loss event. The verb form is to **back up** in two words, whereas the noun is backup [4].

Backups have two particular purposes. The basic role is to recuperate information after its misfortune, be it by information cancellation or defilement. Information misfortune can be a typical encounter of PC clients; an overview saw that as 66% of respondents had lost documents on their home PC. The optional reason for reinforcements is to recuperate information from a previous time, as per a client characterized information maintenance strategy, ordinarily designed inside a reinforcement application for how long duplicates of information are required. However reinforcements address a straightforward type of catastrophe recuperation, and ought to be important for any fiasco recuperation plan, reinforcements without help from anyone else ought not to be viewed as a total calamity recuperation plan. One justification behind this is not all reinforcement frameworks can reconstitute a

PC framework or other complex setup, for example, a PC group, dynamic registry server, or data set server by basically reestablishing information from reinforcement.

Since a reinforcement framework contains no less than one duplicate of all information considered worth saving, the information stockpiling necessities can be critical. Coordinating this extra room and dealing with the reinforcement interaction can a confounded embrace. An information storehouse model might be utilized to give design to the capacity. These days, there are a wide range of sorts of information stockpiling gadgets that are helpful for making reinforcements. There are likewise various manners by which these gadgets can be sorted out to give geographic overt repetitiveness, information security, and versatility.

Before information is shipped off their capacity areas, they are chosen, extricated, and controlled. A wide range of strategies have been created to streamline the reinforcement methodology. These incorporate advancements for managing open documents and live information sources as well as pressure, encryption, and de-duplication, among others. Each reinforcement plan ought to incorporate run-throughs that approve the unwavering quality of the information being supported. It is essential to perceive the restrictions and human elements associated with any reinforcement conspire [7].

There are two primary sorts of back-up systems: **grandfather-father-son** and **partial backups**:

2.2.1 Grandfather-father-son

This system includes taking total reinforcements of all information at normal stretches – every day, week by week, month to month, or whatever is suitable. Different ages of reinforcement are held, frequently three which leads to the name. The latest reinforcement is the child, the past the dad, and the most established reinforcement is the granddad. This strategy is ordinarily utilized for a bunch exchange handling framework with an attractive tape. In the event that the framework fizzles during a bunch run, the expert document is reproduced by reestablishing the child reinforcement and afterward restarting the cluster. Nonetheless, in the event that the child reinforcement falls flat, is undermined or obliterated, the past age of reinforcement (the dad) is utilized.

Moreover, in the event that that fizzles, the age of reinforcement past to the dad (for example the granddad) is required. Obviously the more seasoned the age, the

more the information might be obsolete. Sort out just of records that have changed. For instance, a full reinforcement could be performed week by week, and fractional reinforcements taken daily. Recuperation utilizing this plan includes reestablishing the last full reinforcement and afterward reestablishing all halfway reinforcements to deliver a forward-thinking data set. This cycle is faster than taking just complete reinforcements, to the detriment of longer recuperation time.

2.2.2 Backup Plus Journal

This technique is additionally utilized related to customary complete reinforcements. The expert document is upheld at normal spans. Finished exchanges since the last reinforcement are put away independently and are called diaries, or diary records. The expert record can be reproduced by reestablishing the last total reinforcement and afterward going back over exchanges from the diary documents. This will create the most forward-thinking duplicate of the information base, however recuperation might take longer in view of the time expected to handle a volume of diary records.

2.3 Checking Points in Distributed System

In distributed computing, check pointing is a procedure that endures disappointments that in any case would drive long-running application to restart all along. The most fundamental method for carrying out check pointing is to stop the application, duplicate every one of the expected information from the memory to dependable capacity (e.g., Parallel record framework) and afterward go on with the execution. In the event of disappointment, when the application restarts, it doesn't have to begin without any preparation. Rather, it will peruse the most recent express ("the designated spot") from the steady stockpiling and execute from that.

There are two fundamental methodologies for check pointing in such frameworks: facilitated check pointing and ungraceful checks pointing. In the planned check pointing approach, processes should guarantee that their designated spots are predictable. This is typically accomplished by some sort of two-stage commit convention calculation. In clumsy check pointing, each cycle designated spots its own state freely. It should be focused on that just compelling cycles to designate spot their state at fixed time stretches isn't adequate to guarantee worldwide consistency. The

requirement for laying out a predictable state (i.e., no missing messages or copied messages) may compel different cycles to move back to their designated spots, which thus might make different cycles roll back to significantly prior designated spots, which in the most outrageous case might imply that the main steady state found is the underlying state (the purported cascading type of influence).

Designated spot/Restart: As cluster applications dealt with tens to a huge number of exchanges where every exchange could handle one record from one document against a few unique documents the requirement for the application to be restart-capable eventually without the need to rerun the whole occupation without any preparation became basic. Subsequently the "designated spot/restart" capacity was conceived, in which after various exchanges had been handled, a "depiction" or "designated spot" of the condition of the application could be taken, so, all in all in the event that the application bombed before the following designated spot it very well may be restarted by giving it the designated spot data and the last spot in the exchange document where an exchange had effectively finished. The application could then restart by then.

Check pointing will quite often be costly, so it was by and large not finished with each record, but rather at some sensible split the difference between the expenses of a designated spot versus the worth of the PC time expected to go back over a clump of records. Subsequently the quantity of records handled for every designated spot could go from 25 to 200, contingent upon cost factors and the overall intricacy of the application and the assets expected to restart the application effectively.

2.3.1 Optimization of Recovery Actions by Checkpoints

An proper mix of overt repetitiveness given by log conventions and planning procedures is essentially all that we require for executing exchange arranged information base recuperation as depicted. In genuine frameworks, nonetheless, there are various significant refinements that diminish how much log information required and the expenses of crash recuperation. In the middle, there is the brief log containing UNDO and REDO data and extraordinary sections telling the start and end of an exchange (BOT and EOT, separately). Underneath the transitory log, the exchange history going before the accident is shown, or more it, recuperation handling for worldwide UNDO and halfway REDO is connected with the log passages. We have

not expected a particular spread methodology. There are two inquiries concerning the expenses of crash recuperation:

On account of the appeared DB being changed by inadequate exchanges, how much does the log need to be handled for UNDO recuperation?

In the event that the DBMS doesn't utilize a FORCE discipline, what portion of the log needs to handle for REDO recuperation? The primary inquiry can without much of a stretch be replied: If we realize that updates of fragmented exchanges can have impacted the emerged data set (STEAL), we should filter the impermanent log document back to the BOT passage of the most established deficient exchange to be certain that no invalid information are left in the framework. The subsequent inquiry isn't as straightforward. Retry is begun at a point that is by all accounts picked randomly.

Why would that be no REDO recuperation for object A? By and large, we can expect that on account of a FORCE discipline changed pages will be composed ultimately in light of cradle substitution. One could expect that main the items in the most as of late changed pages must be revamped - in the event that the change was brought about by a total exchange. Be that as it may, take a gander at a cushion action record. The circumstance portrayed is average of numerous enormous information base applications. The greater part of the changed pages will have been changed "as of late," however there are a couple of problem areas like Pi, pages that are adjusted over and over, and, since they are referred to so habitually, have not been composed from the cradle. Inevitably such pages will contain the updates of many complete exchanges, and REDO recuperation will accordingly need to return exceptionally far on the brief log. This makes restart costly. As a general rule, how much log information to be handled for halfway REDO will increment with the time frame between two resulting crashes. As such, the higher the accessibility of the framework, the more exorbitant recuperation will turn into. This is unsuitable for huge, requesting applications. Thus extra measures are expected for making restart costs autonomous of interim between disappointments. Such arrangements will be called designated spots, and are characterized as follows. Producing a designated spot implies gathering data in a protected spot, which characterizes and restricting how much REDO recuperation expected after an accident.

2.4 Classification of Log Data

Depending on which of the compose and proliferation plans presented above are being carried out, we should gather log data with the end goal of eliminating invalid information (adjustments affected by deficient exchanges) from the appeared data set and enhancing the emerged data set with updates of complete exchanges that were not held back in it at the hour of crash.

In this part, we momentarily portray what such log information can resemble and when such information is appropriate to the accident condition of the appeared data set. Log information is repetitive data, collected for the sole motivation behind recuperation from an accident or a media disappointment. They don't go through the planning system of the information base items, yet are gotten on a specific level of the planning order and composed straightforwardly to nonvolatile capacity, that is to say, the log records. There are two unique, though not completely symmetrical, standards for grouping log information. The first is worried about the sort of objects to be logged. Assuming some piece of the actual portrayal, or at least, the bit pat-tern, is kept in touch with the log, we allude to it as actual logging; on the off chance that the administrators and their contentions are recorded on a more significant level, this is called legitimate logging. The subsequent rule concerns whether the condition of the information base previously or after a change or the progress causing the change is to be logged.

Actual State Logging on Page Level - The most fundamental technique, which is as yet applied in numerous business DBMSs, involves the page as the unit of log data. Each time a piece of the direct location space is changed by some change, inclusion, and so on; the entire page containing this piece of the straight location space is kept in touch with the log. If UNDO logging is required, this will be finished before the change happens, yielding the alleged before picture. For REDO purposes, the subsequent page state is recorded as an after picture.

Actual Transition Logging on Page Level - This logging method depends additionally on pages. Be that as it may, it doesn't expressly record the old and new conditions of a page; rather it composes the distinction between them to the log. The capability utilized for figuring the "distinction" between good for nothing strings is dynamic and acquainted as expected by the recuperation calculation. In the event that this distinction is applied to the old condition of a page, again utilizing the selective

or, the new state will result. Then again, applying it to the new state will yield the old state.

Actual State Logging on Access Path Level - Actual logging can be applied to the objects of the entrance way level, in particular, actual records, access way structures, tables, and so forth. The log part must know about these capacity designs and record just the changed section, as opposed to aimlessly logging the entire page around it. The upside of this prerequisite is self-evident: By logging just the actual items really being changed, space necessities for log documents can be definitely decreased. One can save much more space by taking advantage of the way that most access way structures comprise of completely excess data. For instance, one can totally reproduce a B*-tree from the record events to which it alludes. In itself, this sort of reproduction is positively excessively costly to be-come a standard strategy for crash recuperation. Yet, if by some stroke of good luck the changes in the records are logged, after an accident the comparing B* tree can be recuperated reliably, gave that a fitting compose discipline has been noticed for the pages containing the tree.

Change Logging on the Access Path Level - On the entrance way level, we are managing the sections of capacity structures, yet don't have any idea how they are connected with one another concerning the objects of the data set diagram. This kind of data is kept up with on more significant levels of the planning order. In the event that we take a gander at the actual section portrayal (physical progress logging), state change on this level implies that an actual record, a table passage, and so on is added to, erased from, or altered in a page. The contentions relating to these activities are the actual passages, thus there is little contrast among this and the past methodology.

On account of actual state signing on the entrance way level, we set the actual location along with the section portrayal. Here we place the activity code and item identifier with a similar sort of contention. Consequently actual change signing on this level gives nothing basically unique. We can likewise consider coherent progress logging, endeavoring to take advantage of the grammar of the stockpiling structures carried out on this level. The sensible expansion, another record event, for instance, would incorporate every one of the excess table updates, for example, the record id file, the free space table, and so on, every one of which was unequivocally logged with the actual plans. Thus we again have a likely saving of log space. In any case, it is critical to take note of that the coherent changes on this level by and large influence more than one page. If they (or their opposite administrators for UNDO) are to be

applied during recuperation, we should be certain that all impacted pages have a similar state in the emerged data set. This isn't true with direct page portion, and utilizing the more costly aberrant plans can't be legitimate by the nearly couple of advantages yielded by consistent progress signing on the entrance way level. Consequently intelligent change signing on this level can commonly be precluded, however will turn out to be more alluring on the following more elevated level.

Consistent Logging on the Record-Oriented Level - At one level higher, it is feasible to communicate the progressions performed by the exchange program in an exceptionally minimized way by essentially recording the update DML proclamations with their boundaries. Regardless of whether a nonprocedural inquiry language is being utilized over this level, its updates will be decayed into updates of single records or tuples comparable to the single-record updates of procedural DB dialects. Accordingly signing on this level implies that main the INSERT, UPDATE, and DELETE activities, along with their record ids and property estimations, are kept in touch with the log. The planning system recognizes which sections are impacted, which pages should be changed, and so forth. In this manner recuperation is accomplished by re-executing a portion of the recently handled DML explanations. For UNDO recuperation, obviously, the opposite DML explanation should be executed, that is to say, a DELETE to remunerate an INSERT as well as the other way around, and an UP-DATE got back to the first qualities.

These opposite DML proclamations should be created consequently as a feature of the customary logging action, and hence this approach isn't feasible for network-situated DBMSs with data bearing between record relations. In such cases, it very well may be incredibly costly to decide, for instance, the opposite for a DELETE. Details can be found in Reuter [1981]. Framework R is a genuine illustration of a framework with legitimate signing on the record-situated level. All update tasks performed on the tuples are addressed by one summed up alteration administrator, which isn't unequivocally recorded. This administrator changes a tuple distinguished by its tuple identifier (TID) from an old worth to another one, the two of which are recorded. Embedding a tuple involves changing its underlying invalid worth to the given worth, and erasing a tuple involves the opposite progress. Subsequently the log contains the data displayed in Figure 2.2. Intelligent progress logging clearly re-quires an emerged data set that is predictable up to Level 3; that is, it must be joined with ATOMIC spread plans. Albeit the quantity of log information

composed are tiny, recuperation will be more costly than that in different plans, since it includes the going back over of some DML explanations, this should be possible more efficiently than the first handling.

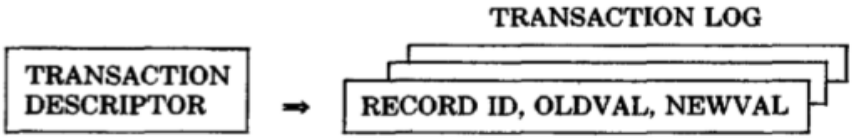


Figure 2.2 Logical Transition Logging as Implemented In System R

2.5 Recovery Procedures

The recovery procedures are backward recovery, Forward recovery and Write-Ahead Logging.

2.5.1 Backward Recovery

Backward recovery is used to undo unwanted changes to the database. It reverses the changes made by transactions which have been aborted. It involves the logic of reprocessing each transaction, which is very time-consuming.

2.5.2 Forward Recovery

It starts with a backup copy of the database. The transaction will then reprocess according to the transaction journal that occurred between the time the backup was made and the present time. It's much faster and more accurate. In forward recovery, the user has to execute unexpected tasks to recover the fault. Forward recovery is commonly the only way to recover from technical failure or human error in critical systems.

2.5.3 Write-Ahead Logging (WAL)

Write-Ahead Logging (WAL) is the fundamental rule. It ensures that a record of every change to the DB is available while attempting to recover from a crash. The old value of a data item on disk has been overwritten by the new value on disk. To facilitate the recovery process, the recovery system may need to maintain a number of lists.

- List of active transactions: transactions started but not committed yet
- List of committed transactions

- List of aborted transactions

WAL protocol for a recovery that requires both UNDO and REDO. The WAL log data is a sequence of log records and includes the following:

Redo records, used for updating disk blocks and insuring file system consistency during restarts. It needs to redo the effect of the operations from the log.

Undo records used for transaction rollback. It needs to undo the effect of the operations from the log.

CHAPTER 3

THE CONTROLLING FOR RECOVERY

Understanding the concepts of database recovery requires a clear comprehension of two factors:

• The type of failure the database has to cope with, and

• The notion of consistency that is assumed as a criterion for describing the state to be reestablished.

Before beginning a discussion of these factors, the contents of this section rely on the description of failure types and the concept of a transaction [1].

3.1 Which Failures Have to Be Anticipated

In order to design and implement a recovery component, one must know precisely which types of failures are to be considered, how often they will occur, how much time is expected for recovery, etc. One must also make assumptions about the reliability of the underlying hardware and storage media, and about dependencies between different failure modes [14]. However, the list of anticipated failures will never be complete for these reasons:

- For each set of failures that one can think of, there is at least one that was forgotten.
- Some failures are extremely rare. The cost of redundancy needed to cope with them may be so high that it may be a sensible design decision to exclude these failures from consideration. If one of them does occur, however, the system will not be able to recover from the situation automatically, and the database will be corrupted. The techniques for handling this catastrophe are beyond the scope of this paper.

The following subjects of failure should be considered:

Transaction Failure: The transaction of failure has already been mentioned in the previous section. For various reasons, the transaction program does not reach its normal commit and has to be reset back to its beginning, either at its own request or on behalf of the DBMS. Within one application, the ratio of transactions that abort

themselves is rather constant, depending only on the amount of incorrect input data, the quality of consistency checking performed by the transaction program, etc. The ratio of transactions being aborted by the DBMS, especially those caused by deadlocks, depends to a great extent on the degree of parallelism, the granularity of locking used by the DBMS, the logical schema (there may be hot spot data, or data that are very frequently referenced by many concurrent transactions), and the degree of interference between concurrent activities.

For classification, it is sufficient to say that transaction failures occur 10-100 times per minute, and that recovery from these failures must take place within the time required by the transaction for its regular execution.

System Failure: The system failures that are considering can be caused by a bug in the DBMS code, an operating system fault, or a hardware failure. In each of these cases processing is terminated in an uncontrolled manner, and we assume that the contents of main memory are lost. Since database-related secondary (nonvolatile) storage remains unaffected, we require that a recovery take place in the same amount of time that would have been required for the execution of all interrupted transactions. If one transaction is executed within the order of 10 milliseconds to 1 second, the recovery should take no more than a few minutes. A system failure is assumed to occur several times a week, depending on the stability of both the DBMS and its operational environment.

Media Failure: Besides these more or less normal failures, we have to anticipate the loss of some or all of the secondary storage holding the database. There are several causes for such a problem, the most common of which are

- bugs in the operating system routines for writing the disk,
- hardware errors in the channel or disk controller,
- head crash,
- Loss of information due to magnetic decay.

Such a situation can only be overcome by full redundancy, that is, by a copy of the database and an audit trail covering what has happened since then. Magnetic storage devices are usually very reliable, and recovery from a media failure is not likely to happen more often than once or twice a year. Depending on the size of a database, the media used for storing the copy, and the age of the copy, recovery of this type will take on the order of 1 hour.

3.2 Summary of Recovery Actions

As we mentioned in Section 3.1, the notion of consistency that uses for defining the targets of recovery is tied to the transaction paradigm, which have encapsulated in the "ACID principle." According to this definition, a database is consistent if and only if it contains the results of successful transactions. Such a state will be called transaction consistent or logically consistent. A transaction, in turn, must not see anything but effects of complete transactions (i.e., a consistent database in those parts that it uses), and will then, by definition, create a consistent update of the database. What does that mean for the recovery component? Let us for the moment ignore transactions being aborted during normal execution and consider only a system failure (a crash). It might then encounter the situation depicted in Figure 3.1. Transactions T1, T2, and T3 have committed before the crash, and therefore will survive. Recovery after a system failure must ensure that the effects of all successful transactions are actually reflected in the database. But what is to be done with T4 and T5? Transactions have been defined to be atomic; they either succeed or disappear as though they had never been entered. There is therefore no choice about what to do after a system failure; the effects of all incomplete trans- actions must be removed from the database. Clearly, a recovery component adhering to these principles will produce a transaction- consistent database. Since all successful transactions have contributed to the data- base state, it will be the most recent trans- action-consistent state. We now can distinguish four recovery actions coping with different situations:

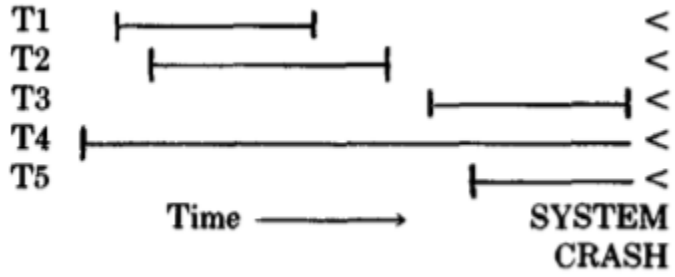


Figure 3.1 Scenario for Discussing Transaction-Oriented Recovery

Transaction UNDO: If a transaction aborts itself or must be aborted by the system during normal execution. By definition, UNDO removes all effects of this transaction from the database and does not influence any other transaction.

Partial REDO: When recovering from a system failure, since execution has been terminated in an uncontrolled manner, results of complete transactions may not yet be reflected in the database. Hence they must be repeated, if necessary, by the recovery component.

Global UNDO: When recovering from a system failure, the effects of all incomplete transactions have to be rolled back. The database is assumed to be physically destroyed; therefore, it must start from a copy that reflects the state of the database some days, weeks, or months ago. Since transactions are typically short, we need not consider incomplete transactions over such a long time. Rather we have to supplement the copy with the effects of all transactions that have committed since the copy was created.

With these definitions we have introduced the transaction as the only unit of recovery in a database system. This is an ideal condition that does not exactly match reality. For example, transactions might be nested, that is, composed of smaller sub-transactions. These sub-transactions also are atomic, consistent, and isolated but they are not durable. Since the results of sub-transactions are removed whenever the enclosing transaction is undone, durability can only be guaranteed for the highest transaction in the composition hierarchy. A two-level nesting of transactions can be found in System R, in which an arbitrary number of save points can be generated inside a transaction. The database and the processing state can be reset to any of these save points by the application program. Another extension of the transaction concept is necessary in fields like CAD. Here the units of consistent state transitions, that is, the design steps, are so long (days or weeks) that it is not feasible to treat them as indivisible actions. Hence these long transactions are consistent, isolated, and durable, but they are not atomic. It is sufficient for the purpose of taxonomy to consider "ideal" transactions only.

3.3 THE MAPPING HIERARCHY OF A DBMS

There are numerous techniques and algorithms for implementing database recovery, many of which have been described. It needs to reduce these various methods to a small set of basic concepts, allowing a simple, yet precise classification of all reasonable implementation techniques; for the purposes of illustration, we need a basic model of the DBMS architecture and its hardware environment. This model, although it contains many familiar terms from systems like INGRES, System R, or those of the CODASYL [1973, 1978] type, is in fact a rudimentary database architecture that can also be applied to unconventional approaches like CASSM or DIRECT although this is not purpose here.

3.3.1 The Mapping Process: Objects and Operations

As shown in Table 3.1, the major steps of dynamic abstraction from the level of physical storage up to the user consists of some billions of bits stored on disk, which are interpreted by the DBMS into meaningful information on which the user can operate. With each levels of abstraction (proceeding from the bottom up), the objects become more complex, allowing more powerful operations and being constrained by a larger number of integrity rules. The uppermost interface supports one of the well-known data models, whether relational, network like, or hierarchical. Note that this mapping hierarchy is virtually contained in each DBMS, although for performance reasons it will hardly be reflected in the module structure. We shall briefly sketch the characteristics of each layer, with enough detail to establish our taxonomy. For a more complete description see Haerder and Reuter.

Table 3.1 Description of the DB Mapping Hierarchy

Level of abstraction	Objects	Auxiliary mapping data
Nonprocedural or algebraic access	Relations, views, tuples	Logical schema description
Record-oriented, navigational access	Records, sets, hierarchies, networks	Logical and physical schema description
Record and access path management	Physical records, access paths	Free space tables, DB- key translation tables
Propagation control	Segments, pages	Page tables, Bloom filters
File management	Files, blocks	Directories, VTOCs, etc.

File Management. The lowest layer operates directly on the bit patterns stored on some nonvolatile, direct access device like a disk, drum, or even magnetic bubble memory. This layer copes with the physical characteristics of each storage type and abstracts these characteristics into fixed- length blocks. These blocks can be read, written, and identified by a (relative) block number. This kind of abstraction is usually done by the data management system (DMS) of a normal general-purpose operating system.

Propagation Control. This level is not usually considered separately in the current become clear in the following sections we strictly distinguish between pages and blocks. A page is a fixed-length partition of a linear address space and is mapped into a physical block by the propagation control layer. Therefore a page can be stored in different blocks during its lifetime in the database, depending on the strategy implemented for propagation control.

Access Path Management. This layer implements mapping functions much more complicated than those performed by sub- ordinate layers. It has to maintain all physical object representations in the database (records, fields, etc.), and their related access paths (pointers, hash tables, search trees, etc.) in a potentially unlimited linear virtual address space. This address space, which is divided into fixed-length pages, is provided by the upper interface of the sup- porting layer. For performance reasons, the partitioning of data into pages is still visible on this level.

Navigational Access Layer. At the top of this layer we find the operations and objects that are typical for a procedural data manipulation language (DML). Occurrences of record types and members of sets are handled by statements like STORE, MOD- IFY, FIND NEXT, and CONNECT [CO- DASYL 1978]. At this interface, the user navigates one record at a time through a hierarchy, through a network, or along logical access paths.

Nonprocedural Access Layer. This level provides a nonprocedural interface to the database. With each operation the user can handle sets of results rather than single records. A relational model with high-level query languages like SQL or QUEL is a convenient example of the abstraction achieved by the top layer .On each level, the mapping of higher objects to more elementary ones requires additional data structures, some of which are shown in Table 3.1.

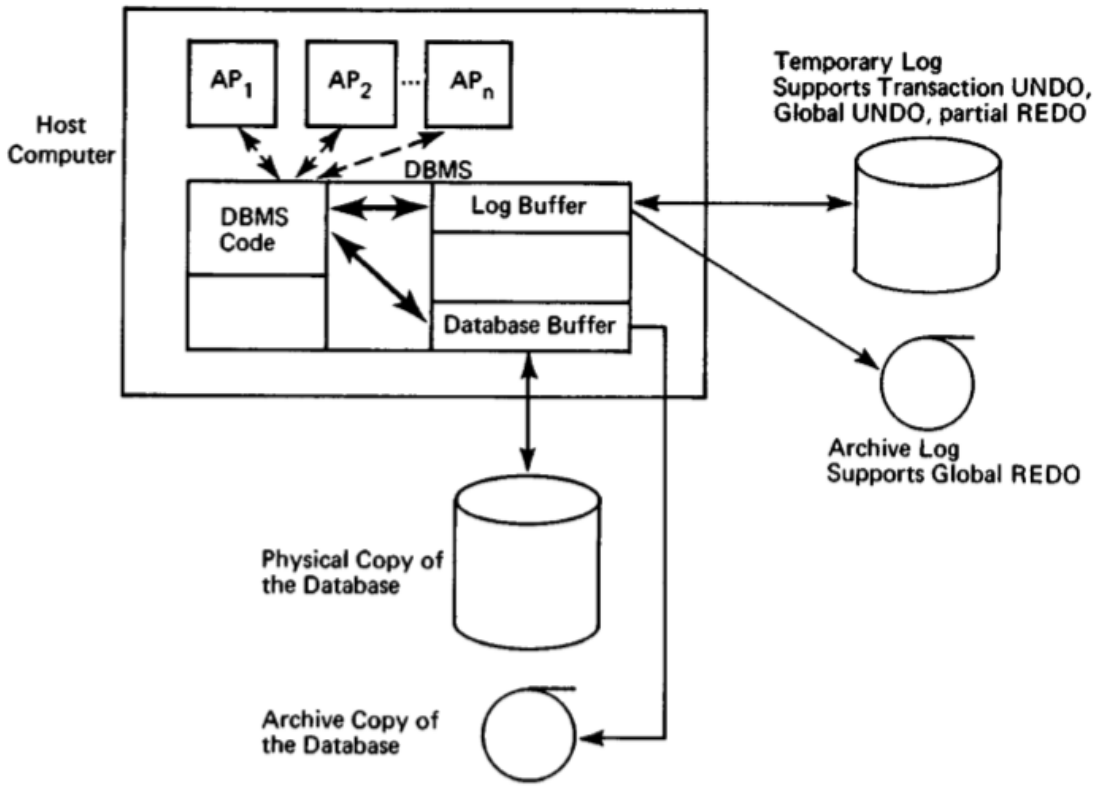


Figure 3.2 Storage Hierarchy of A DBMS During Normal Mode of Operation

3.3.2 The Storage Hierarchy: Implementation Environment

Both the number of redundant data required to support the recovery actions described in Section 3.2 and the methods of collecting such data are strongly influenced by various properties of the different storage media used by the DBMS. In particular, the dependencies between volatile and permanent storage have a strong impact on algorithms for gathering redundant information and implementing recovery measurement. As a description framework we shall use a storage hierarchy, as shown in Figure 3.2. It closely resembles the situation that must be dealt with by most of today's commercial database systems. The host computer, where the application programs and DBMS are located, has a main memory, which is usually volatile. 8 Hence we assume that the contents of the database buffer, as well as the contents of the output buffers to the log files, are lost whenever the DBMS terminates abnormally. Below the volatile main memory there is a two-level hierarchy of permanent copies of the database. One level contains an on-line version of the database in direct access memory; the other contains an archive copy as a provision against loss of the on-line copy. While both are functionally situated on the same level, the on-line copy is almost always up-to-date, whereas the archive copy can contain an old state of the database. Our main concern here is data- base recovery, which, like all provisions for fault tolerance, is based upon redundancy. We have mentioned one type of redundancy: the archive copy, kept as a starting point for reconstruction of an up-to-date on-line version of the database (global REDO)[6]. This is discussed in more detail in Section 4. To support this, and other recovery actions introduced in Section 3.1, two types of log files are required:

Temporary Log: The information collected in this file supports crash recovery; that is, it contains information needed to reconstruct the most recent database (DB) buffer. Selective transaction UNDO requires random access to the log records. Therefore, we assume that the temporary log is located on disk.

Archive Log: This file supports global REDO after a media failure. It depends on the availability of the archive copy and must contain all changes committed to the database after the state reflected in the archive copy. Since the archive log is always processed in sequential order, we assume that the archive log is written on magnetic tape.

3.3.3 Different Views of a Database

In Section 3.3.1, we indicated that the database looks different at each level of abstraction, with each level using different objects and interfaces. But this is not what we mean by "different views of a database" in this section. We have observed that the process of abstraction really begins at Level 3, up to which there is only a more convenient representation of data in external storage. At this level, abstraction is dependent on which pages actually establish the linear address space, that is, which block is read when a certain page is referenced. In the event of a failure, there are different possibilities for retrieving the contents of a page. These possibilities are denoted by different views of the database:

The current database comprises all objects accessible to the DBMS during normal processing. The current contents of all pages can be found on disk, except for those pages that have been recently modified. Their new contents are found in the DB buffer. The mapping hierarchy is completely correct. The materialized database is the state that the DBMS finds at restart after a crash without having applied any log information. There is no buffer. Hence some page modifications (even of successful transactions) may not be reflected in the on-line copy. It is also possible that a new state of a page has been written to disk, but the control structure that maps pages to blocks has not yet been updated. In this case, a reference to such a page will yield the old value. This view of the database is what the recovery system has to transform into the most re- cent logically consistent current database. The physical database is composed of all blocks of the on-line copy containing page images--current or obsolete. Depending on the strategy used on Level 2, there may be different values for one page in the physical database, none of which are necessarily the current contents. This view is not normally used by recovery procedures, but a salvation program would try to exploit all information contained therein.

With these views of a database, we can distinguish three types of update operations-all of which explain the mapping function provided by the propagation control level. First, we have the modification of page contents caused by some higher-level module. This operation takes place in the DB buffer and therefore affects only the current database. Second, there is the write operation, transferring a modified page

to a block on disk. In general, this affects only the physical database. If the information about the block containing the new page value is stored in volatile memory, the new contents will not be accessible after a crash; that is, it is not yet part of the materialized database. The operation that makes a previously written page image part of the materialized database is called propagation. This operation writes the updated control structures for mapping pages to blocks in a safe, nonvolatile place, so that they are available after a crash. If pages are always written to the same block (the so-called "update-in-place" operation, which is done in most commercial DBMS), writing implicitly is the equivalent of propagation. However, there is an important difference between these operations if a page can be stored in different blocks.

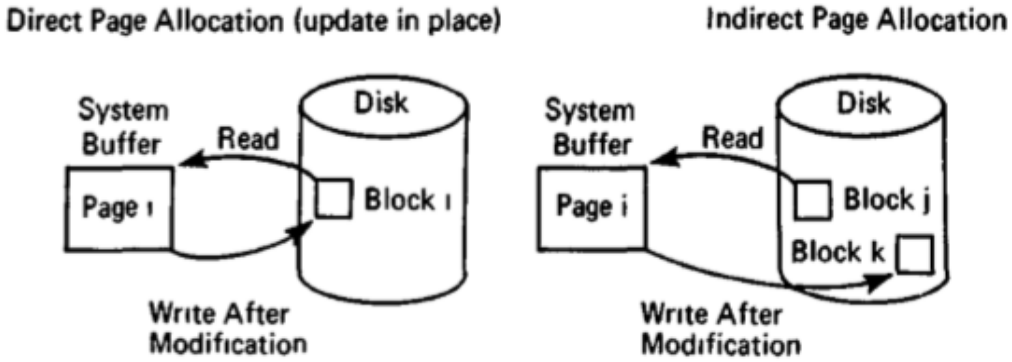


Figure 3.3 Page Allocation Principles

3.4 Crash Recovery

In order to illustrate the consequences of the concepts introduced thus far, we shall present a detailed discussion of crash recovery. First, we consider the state in which a database is left when the system terminates abnormally. From this we derive the type of redundant (log) information required to reestablish a transaction-consistent state, which is the overall purpose of DB recovery. After completing our classification scheme, we give examples of recovery techniques in currently available database systems. Note that the results in this section also apply to transaction UNDO--a much simpler case of global UNDO, which applies when the DBMS is processing normally and no information is lost [2].

3.4.1 State of the Database after a Crash

After a crash, the DBMS has to restart by applying all the necessary recovery actions described in Section 3.2. The DB buffer is lost, as is the current database, the only view of the database to contain the most recent state of processing. Assuming that the on-line copy of the database is intact, there are the materialized database and the temporary log file from which to start recovery. We have not discussed the contents of the log files for the reason that the type and number of log data to be written during normal processing are dependent upon the state of the materialized database after a crash. This state, in turn, depends upon which method of page allocation and propagation is used.

In the case of direct page allocation and ~ATOMIC propagation, each write operation affects the materialized database. The decision to write pages is made by the buffer manager according to buffer capacity at points in time that appear arbitrary. Hence the state of the materialized database after a crash is unpredictable: When recent modifications are reflected in the materialized database, it is not possible (without further provisions) to know which pages were modified by complete transactions (whose con- tents must be reconstructed by partial REDO) and which pages were modified by incomplete transactions (whose contents must be returned to their previous state by global UNDO). Further possibilities for providing against this situation are briefly discussed in Section 3.2.1.

In the case of indirect page allocation and ATOMIC propagation, we know much more about the state of the materialized database after crash. ATOMIC propagation is indivisible by any type of failure, and therefore we find the materialized database to be exactly in the state produced by the most recent successful propagation. This state may still be inconsistent in that not all updates of complete transactions are visible, and some effects of incomplete transactions are. However, ATOMIC propagation ensures that a set of related pages is propagated in a safe manner by restricting propagation to points in time when the current database fulfills certain consistency constraints. When these constraints are satisfied, the updates can be mapped to the materialized database all at once. Since the current database is consistent in terms of the access path management level--where propagation occurs--this also ensures that all internal pointers, tree structures, tables, etc. are correct. Later on, it also discusses the schemes that allow for transaction-consistent propagation.

The state of the materialized database after a crash can be summarized as follows:

-ATOMIC Propagation. Nothing is known about the state of the materialized database; it must be characterized as "chaotic." ATOMIC Propagation. The materialized database is in the state produced by the most recent propagation. Since this is bound by certain consistency constraints, the materialized database will be consistent (but not necessarily up-to-date) at least up to the third level of the mapping hierarchy.

In the case of ~ATOMIC propagation, one cannot expect to read valid images for all pages from the materialized database after a crash; it is inconsistent on the propagation level, and all abstractions on higher levels will fail. In the case of ATOMIC propagation, the materialized database is consistent at least on Level 3, thus allowing for the execution of operations on Level 4 (DML statements).

3.4.2 Types of Log Information to Support Recovery Actions

The temporary log file must contain all the information required to transform the materialized database "as found" into the most recent transaction-consistent state (see Section 1). As we have shown, the materialized database can be in more or less defined states, may or may not fulfill consistency constraints, etc. Hence the number of log data will be determined by what is contained in the materialized database at the beginning of restart. We can be fairly certain of the contents of the materialized database in the case of ATOMIC propagation, but the result of "~ATOMIC schemes have been shown to be unpredictable. There are, however, additional measures to somewhat reduce the degree of uncertainty resulting from ATOMIC propagation, as discussed in the following section.

3.4.2.1 Dependencies between Buffer Manager and Recovery Component

There are different dependencies between Buffer Management and UNDO Recovery actions and Buffer Management and REDO Recovery actions.

3.4.2.1.1 Buffer Management and UNDO Recovery Actions

During the normal mode of operation, modified pages are written to disk by some replacement algorithm man- aging the database buffer. Ideally, this happens at points in time determined solely by buffer occupation and, from a consistency

perspective, seems to be arbitrary. In general, even dirty data, that is, pages modified by incomplete transactions, may be written to the physical database. Hence the UNDO operations described earlier will have to recover the contents of both the materialized database and the external storage me- dia. The only way to avoid this requires that the buffer manager be modified to prevent it from writing or propagating dirty pages under all circumstances. In this case, UNDO could be considerably simplified:

If no dirty pages are propagated, global UNDO becomes virtually unnecessary that is, if there are no dirty data in the materialized database. • If no dirty pages are written, transaction UNDO can be limited to main storage (buffer) operations.

The major disadvantage of this idea is that very large database buffers would be required (e.g., for long batch update transactions), making it generally incompatible with existing systems. However, the two different methods of handling modified pages introduced with this idea have important implications with UNDO recovery.

STEAL modified pages may be written and/or propagated at any time. STEAL modified pages are kept in buffer at least until the end of the transaction (EOT).

The definition of STEAL can be based on either writing or propagating, which is not discriminated in ATOMIC schemes. In the case of ATOMIC propagation variants of STEAL are conceivable, and each would have a different impact on UNDO recovery actions; in the case of STEAL, no logging is required for UNDO purposes.

3.4.2.1.2 Buffer Management and REDO Recovery Actions

As soon as a transaction commits, all of its results must survive any subsequent failure (durability). Committed updates that have not been propagated to the materialized database would definitely be lost in case of a system crash, and so there must be enough redundant information in the log file to reconstruct these results during restart (partial REDO). It is conceivable, however to avoid this kind of recovery by the following technique. During Phase 1 of EOT processing all pages modified by this transaction are propagated to the materialized database; that is, their writing and propagation are enforced. Then we can be sure that either the transaction is complete, which means that all of its results are safely recorded (no partial REDO), or in case of a crash, some updates are not yet written, which means that the transaction is not successful and must be rolled back (UNDO recovery actions). Thus,

it has another criterion concerning buffer handling, which is related to the necessity of REDO recovery during restart:

FORCE: All modified pages are written and propagated during EOT processing. FORCE: No propagation is triggered during EOT processing.

The implications with regard to the gathering of log data are quite straightforward in the case of FORCE: No logging is required for partial REDO. While FORCE avoids partial REDO, there must still be some REDO log information for global REDO to provide against loss of the on-line copy of the database.

CHAPTER 4

SYSTEM DESIGN AND IMPLEMENTATION

Since business organizations are very dependent on data. Transaction Processing System (TPS) is used to handle data accuracy and data loss prevention. In TPS, there are three main functions: data loss prevention, data locking and deadlock detection and resolution. Since most of the business organizations - concerned on the accuracy of data, TPS keeps the data accuracy - prevent data loss during transactions. Data consistency is mainly relied on secure transactions to show reliability to its customers. This transaction processing system is based on seed block recovery information and prevents data loss during transaction period. During the transactions, due to various errors, the transaction is interrupted and the connection with the other breaks down.

When system breaks down, it stops the regular routines of the business and stopping its operation for a certain amount of time. To recover incomplete transaction, to prevent data loss, to minimize disruption the well-designed backup and recovery procedure is put into use. This system is proposing a very efficient algorithm for data backup. In this system, this algorithm is called as Seed Block Algorithm (SBA). In Seed Block Algorithm, there are two types of objectives which we are trying to achieve, these are as: First one is from any distinct location user collecting information when network connectivity is absent. The process flow of the proposed system is shown in Figure 4.1.

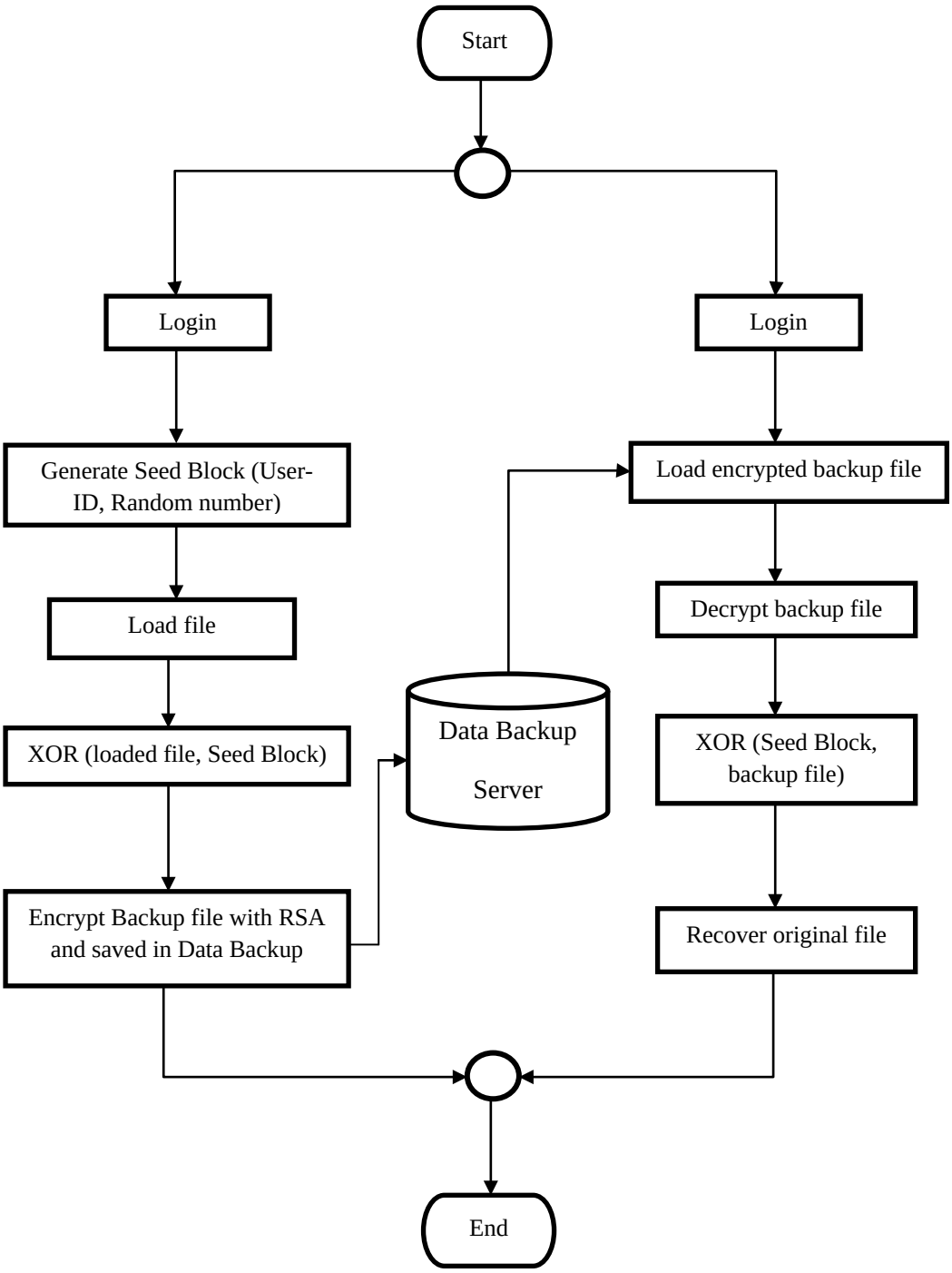


Figure 4.1 The System Flow Diagram

4.1 Proposed Data Recovery Technique - Seed Block Algorithm (SBA)

1. BEGIN
2. Set a random number in the main storage and unique client id for every client
3. Whenever the client id is being register in the main storage, then client id and random number is getting EXORed with each other to generate seed block for the particular client.
4. Whenever client creates the file in cloud first time, it is stored at the main storage.
5. When it is stored in main storage (blob), the main file of client is being EXORed with the Seed Block of the particular client.
6. It is also encrypted using public key RSA
7. That output file is stored at the backup storage (blob) in the form of file' (pronounced as File dash).
8. During Retrieval, check if data present in main storage If present then EXOR with seed block and retrieve data If not present, retrieve data from backup storage.
9. During Retrieval from backup storage, the private key of the user will used to decrypt file'
10. The user will get the original file by Exoring on decrypted file' with the seed block of the corresponding client to produce the original file and return the resulted file in case of crash.
11. END

4.2 Benefits of Remote Backup Services

The following issues must be covered in Remote Backup services:

- **Data Integrity**: Server's whole structure along with all complete states tells us about Data integrity of server. At the time of transmission and reception, data which resist to any kind of change in it. Such type of data is verifies using Data Integrity. Validity of Data on Remote server is also checked by Data integrity.
- **Data security**: The Remote servers have primary priority to provide total security to data of user. And either intentionally or non-intentionally, only particular user should have access to that data or not any other users.
- **Data Confidentiality**: In certain times, the system has to kept user's data files to be secret as if number of users are simultaneously accessing the cloud, when other users accessing files on the cloud should unable to see particular data file that belongs to only that particular user. This is also known as Data Confidentiality characteristic.
- **Genuine Characteristic**: Trustworthiness is the important characteristic of the Remote cloud. Remote cloud should possess that because every user are having their private also confidential data on cloud. Therefore, Trustworthiness characteristic should be present in remote also in cloud backup.
- **Cost efficiency**: The cost of processing of data recovery should be efficient so that large number of companies along with users can take benefit of back-up and recovery service. There are many large numbers of methods that have focused on these issues. The fore said issues occur at the time of recovery also in back-up of domain of cloud computing is discussed.

4.3 Example Operation of System

Step-1 Start

Step-2 Assume there are five clients in the system

Let clientid for client1 = cid01

clientid for client2 = cid02

clientid for client3 = cid03

clientid for client4 = cid04

clientid for client5 = cid05

Random number = 12345

Step-3 ClientID for client2 XOR random number

cid02⊕ 12345 = 09910510002 ⊕ 12345

= 9910502347

(Seed block for client2)

Step-4 Client2 creates file in cloud

(E.g., ‘myat’ = 6d796174)

Change text to hexadecimal with converter

Step-5 Client2’s file XOR with Client2’s seed block

6d796174 ⊕ 9910502347 = 997d294230

(Result stores in main storage)

Step-6 Encrypt by public key RSA (Using Online RSA Encryption, Decryption And key Generator Tool)

Step-7 Encrypted file stored at backup server.

Step-8 When the wanted file is retrieved, result in Step-5 XOR with Client2’s seed block.

997d294230 ⊕ 9910502347 = 6d796174

(Result show as hexadecimal, change to text again)

Step-9 If not present, we retrieve data from back up storage. Using private key with RSA calculation, Client2 will get decrypted file.

Step-10 To get original file, decrypted file XOR with Client2’s seed block.

Step-11 Stop

ASCII code for c, i, d

c=099

i=105

d=100

RSA Calculation

Let p = 3

q = 5

r = p * q = 3*5 = 15

Encryption key, e = 11 (Public key, greater than p & q)

Decryption key, d =? (Private key)

d * e = 1 modulo (p-1) * (q-1)

d * 11 = 1 modulo 2 * 4

d * 11 = 1 modulo 8

If d = 1 => 1 * 11 = 1 modulo 8

Divide 8 into 11, and the answer is 1 with remainder 3(1≠ 3)

If d = 2 => 2 * 11 = 1 modulo 8

Divide 8 into 22, and the answer is 2 with remainder 6(1≠ 6)

If d = 3 => 3 * 11 = 1 modulo 8

Divide 8 into 33, and the answer is 4 with remainder 1(1 = 1)

Therefore, d = 3.

Let Plaintext, P = 13 (integer)

Cipher text, $C = P^e$ modulo r

$= 13^{11}$ modulo 15

= 1,792,160,394,037 modulo 15

= 7

$P = C^d$ modulo r

$= 7^3$ modulo 15

= 343 modulo 15

= 13

4.4 RSA Encryption Algorithm

RSA Encryption Algorithm is the most commonly used public key encryption algorithm. It can be used both for encryption and for digital signatures. The security of RSA is generally considered equivalent to factoring, although this has not been proved.

RSA computation occurs with integers modulo n = p * q, for two large secret primes p, q. To encrypt a message m, it is exponentiated with a small public exponent e. For decryption, the recipient of the cipher text

$$c = m^e \pmod{n}$$

computes the multiplicative reverse

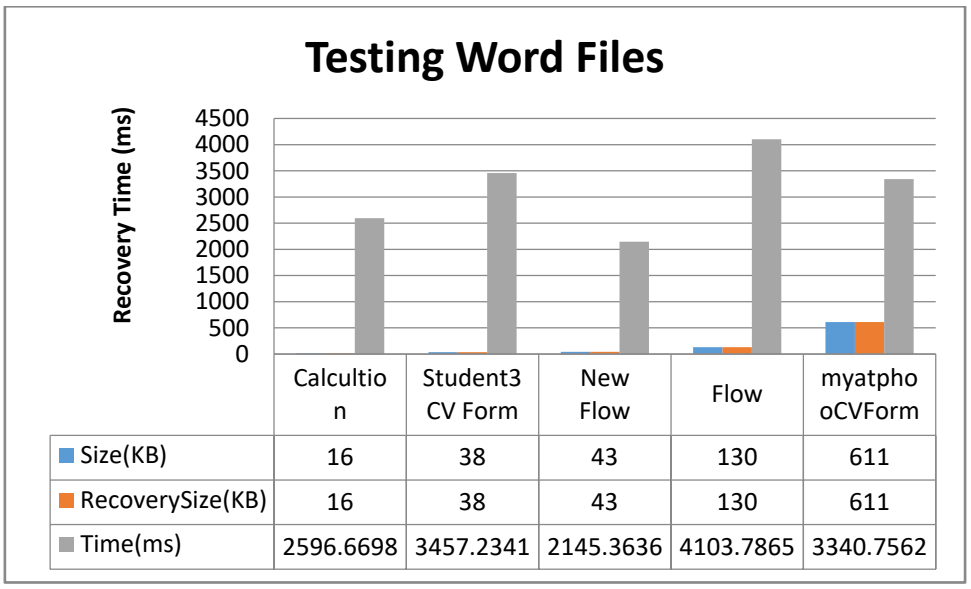
$$d = e^{-1} \pmod{(p-1)*(q-1)}$$

(We require that *e* is selected suitably for it to exist) and obtains $c^d = m^e * d = m \pmod{n}$. The private key consists of n, p, q, e, d (where p and q can be omitted); the public key contains only n and e. The problem for the attacker is that computing the reverse d of e is assumed to be no easier than factorizing n.

The key size should be greater than 1024 bits for a reasonable level of security. Keys of size, say, 2048 bits should allow security for decades.

4.5 Experimental Results

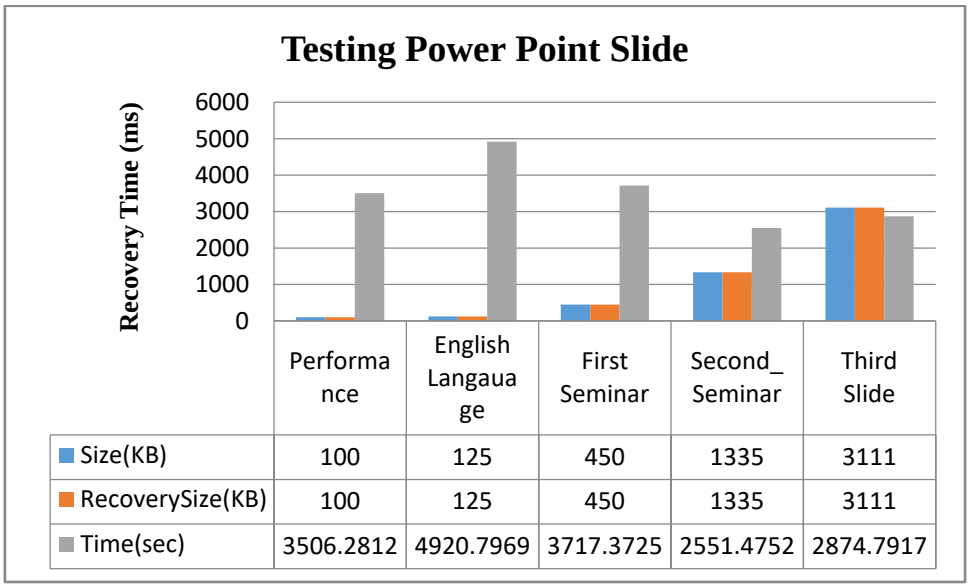
The proposed system is simulated using C# (asp.net) language on Microsoft Visual Studio IDE and Microsoft SQL Server is used as Database Engine of the System. The files are stored in servers until before deleted and this system make uploading and recovering your files quick and easy. Users can store in servers for many different file types and sizes.



	Calcultion	Student3 CV Form	New Flow	Flow	myatphooCVForm
Size(KB)	16	38	43	130	611
RecoverySize(KB)	16	38	43	130	611
Time(ms)	2596.6698	3457.2341	2145.3636	4103.7865	3340.7562

Figure4.2 Testing (Microsoft Office Word with Different File Sizes)

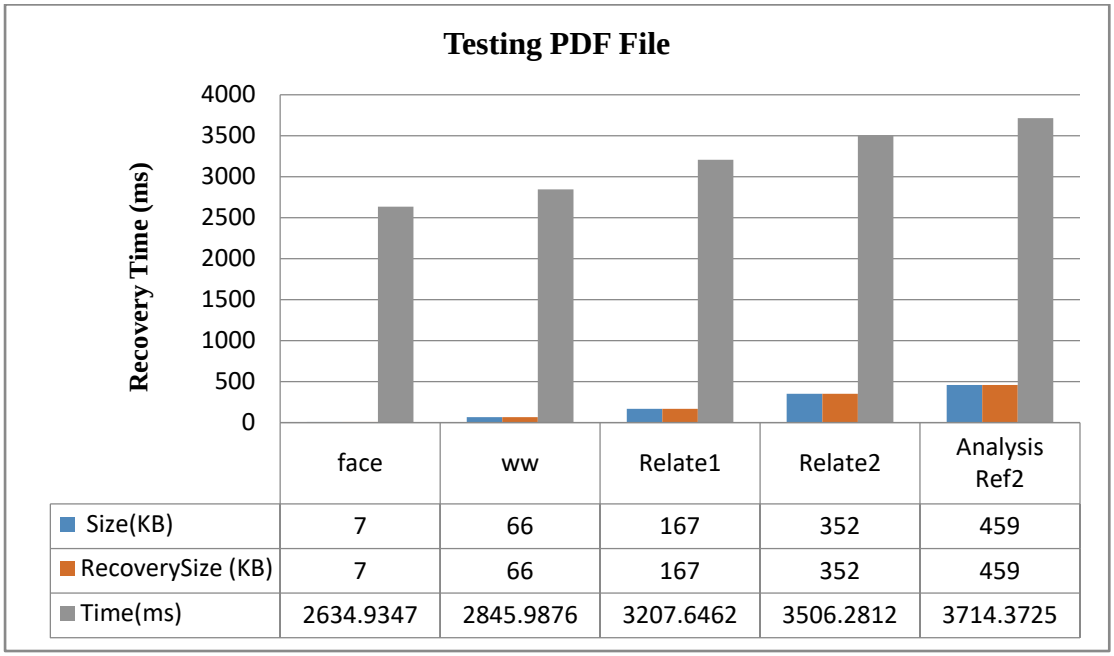
Figure 4.2 shows testing time and file sizes of documents between main server and backup server. In Figure 4.3, the result time and file sizes of Microsoft Office Power Point will be shown.



	Performance	English Langaua ge	First Seminar	Second_ Seminar	Third Slide
Size(KB)	100	125	450	1335	3111
RecoverySize(KB)	100	125	450	1335	3111
Time(sec)	3506.2812	4920.7969	3717.3725	2551.4752	2874.7917

Figure 4.3 Testing (Microsoft Office Power Point with Different File Sizes)

The test of time and main file sizes and recovery files sizes of Portable Document Format and Microsoft Office Excel are shown in Figure 4.4 and Figure 4.5.



	face	ww	Relate1	Relate2	Analysis Ref2
Size(KB)	7	66	167	352	459
RecoverySize (KB)	7	66	167	352	459
Time(ms)	2634.9347	2845.9876	3207.6462	3506.2812	3714.3725

Figure 4.4 Testing (Portable Document Format with Different File Sizes)

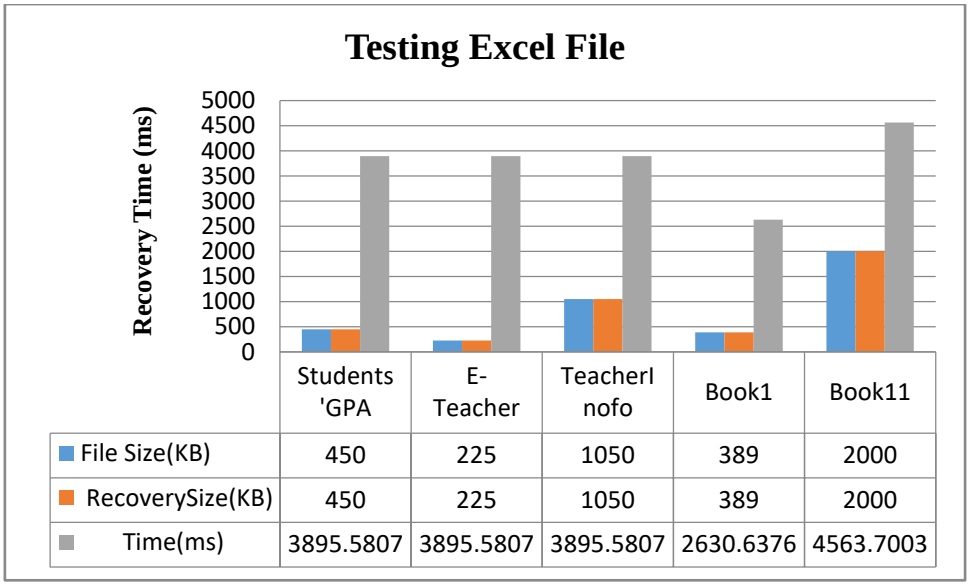


Figure 4.5 Testing (Microsoft Office Excel with different file sizes)

4.6 Implementation of the System

This section describes the user interface of the system implementation. This system implemented on network connection from one PC as Main Server to another PC as Backup Server. So, users should connect two devices firstly. When connection is ready, users can do transaction processes to load files and recover files from their backup files. This system will start with the main page as shown in Figure 4.6.

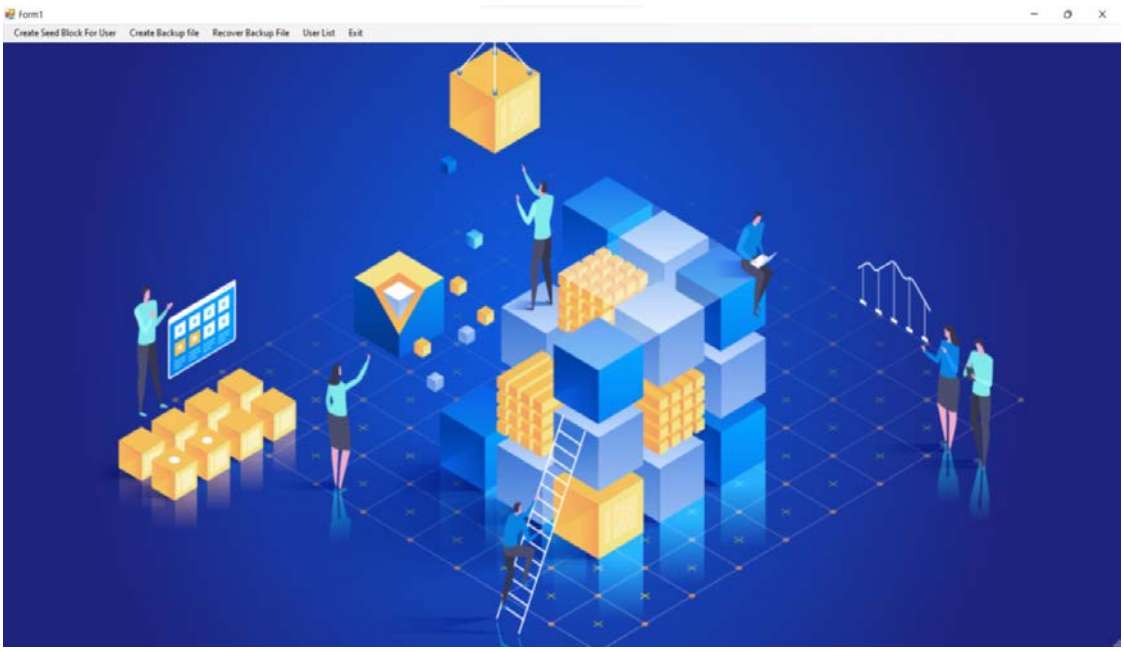


Figure 4.6 Main Page of System

But this system can only be used by registered user to maintain the data security and privacy. So, the new user must be registered / sign up first. Then, the registered user can be entered the system via log in page. The signup page and log in page of the system are shown in Figure 4.7 and Figure 4.8.

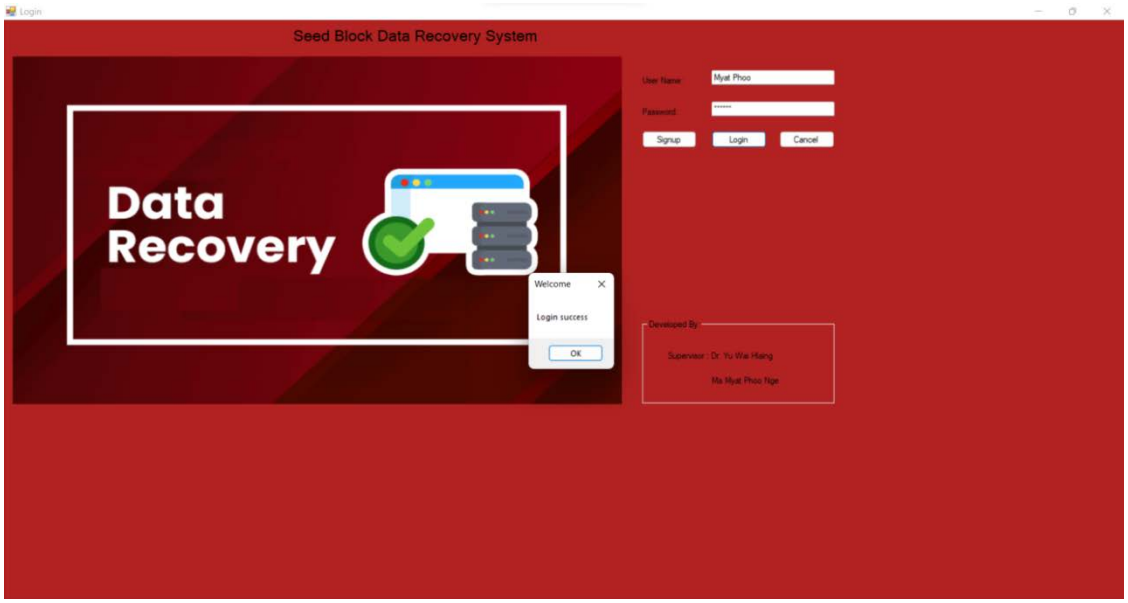


Figure 4.7 The Login Page

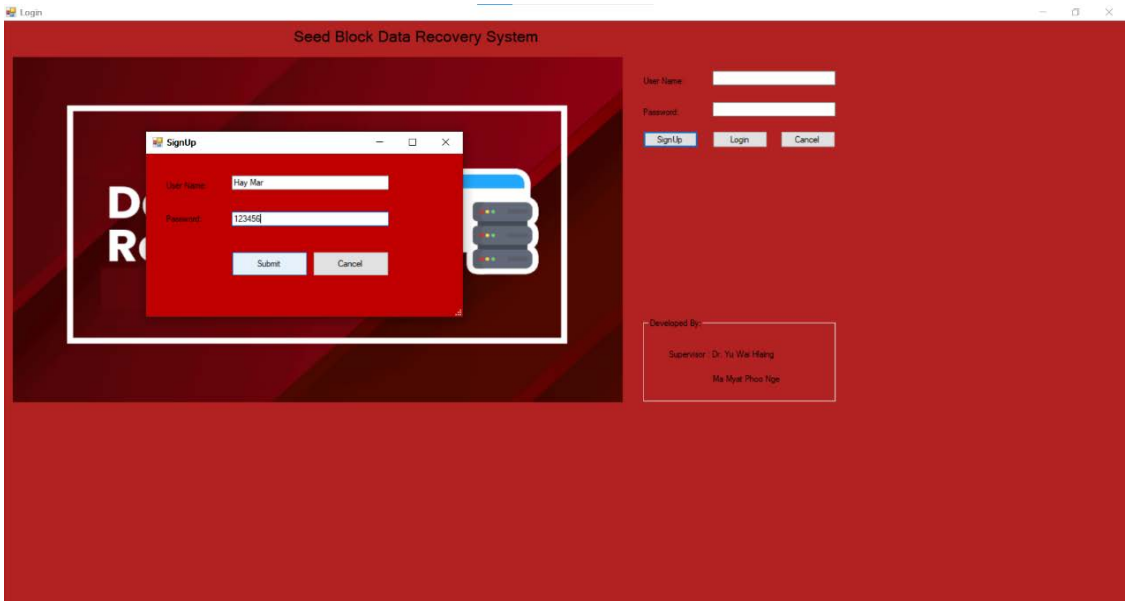


Figure 4.8 The SignUp Page

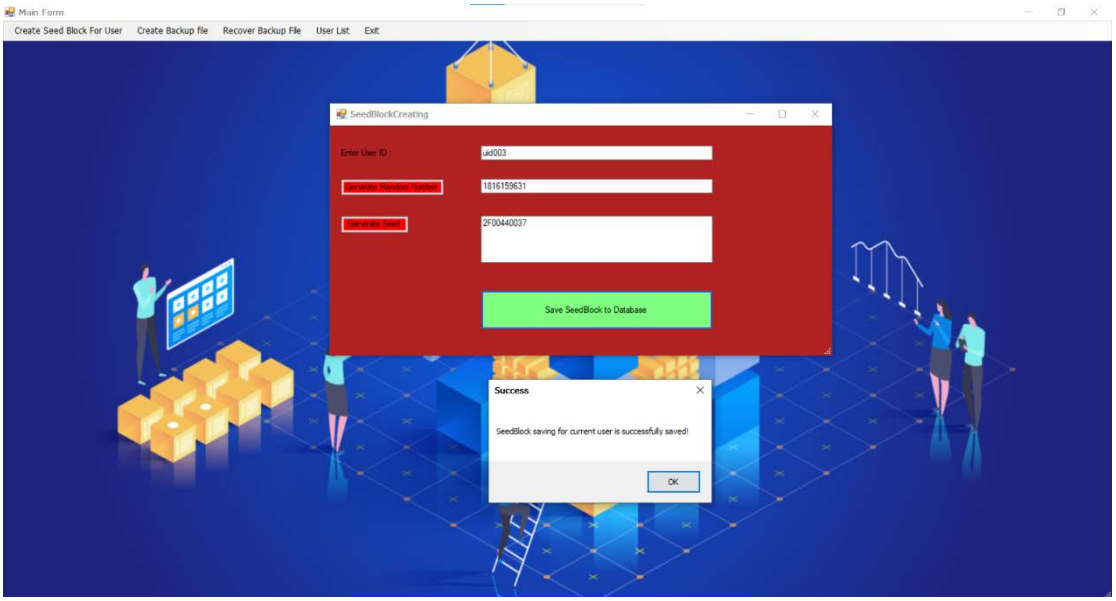


Figure 4.9 Generate Seed Block File

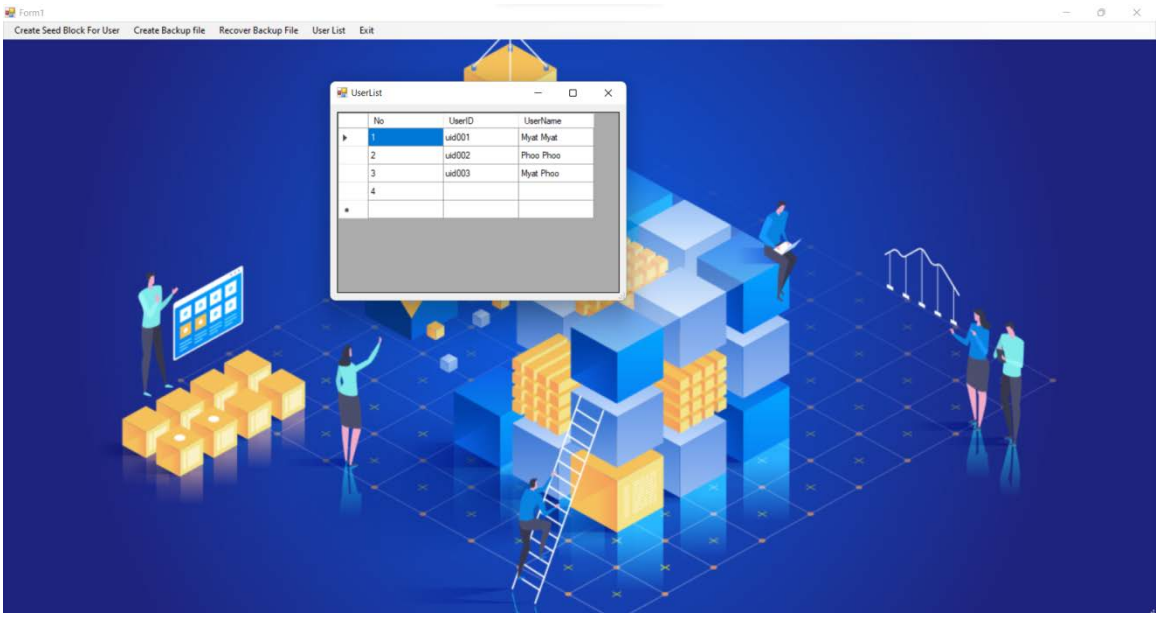


Figure 4.10 'User List' Page

The users or organizations are related with generating seed blocks which are kept in secretly. About developing seed blocks and the list of the users are shown in Figure 4.9 and Figure 4.10.

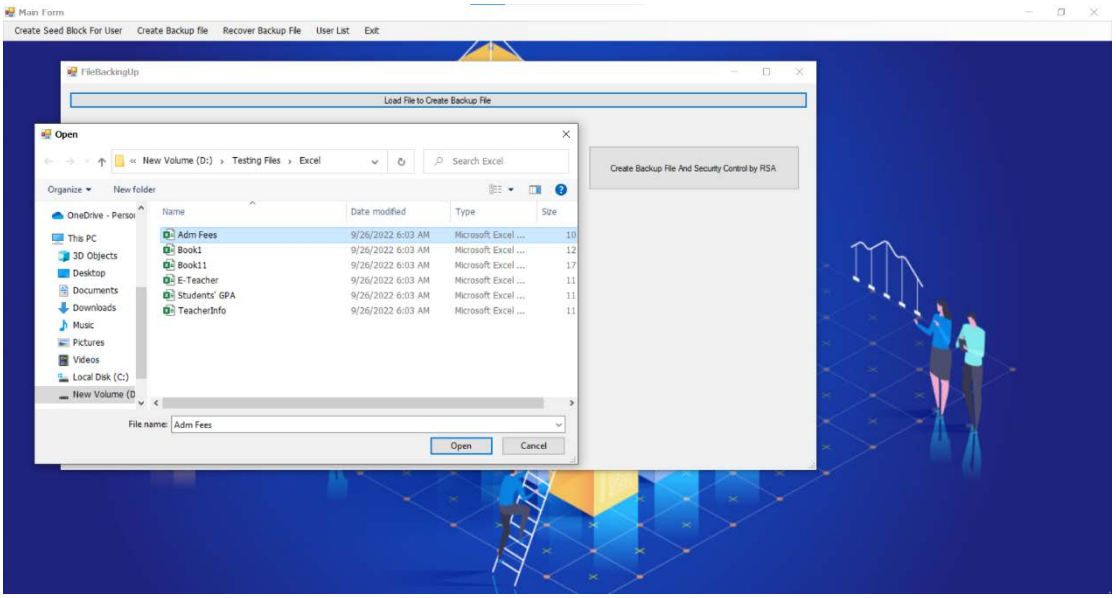


Figure 4.11 The Uploading File

The next step after the login is uploading the file in Figure 4.11. After uploading the file, the system generates the seed automatically for particular file using Seed Block algorithm and stores that file in the system's server. If user wants to retrieve the file, the system makes searches in main server firstly and downloads from the main server. After file upload successful 'acknowledge message' will be shown.

After the end of uploading stage, file will be encrypted by RSA function in Figure 4.12 and in Figure 4.13, the recovery stage of the system will be shown.

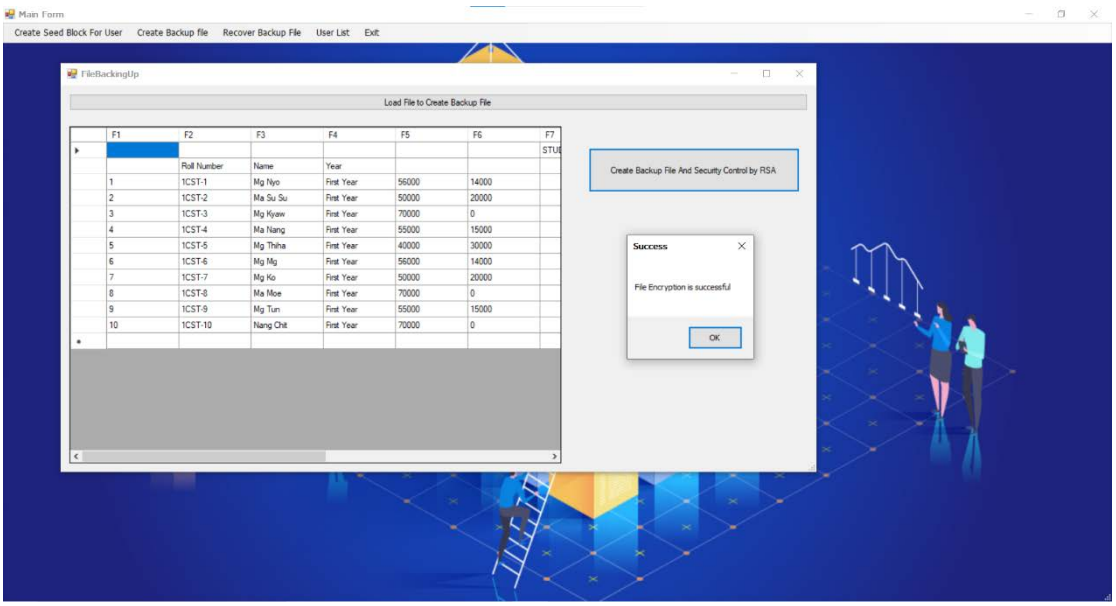


Figure 4.12 Encryption File

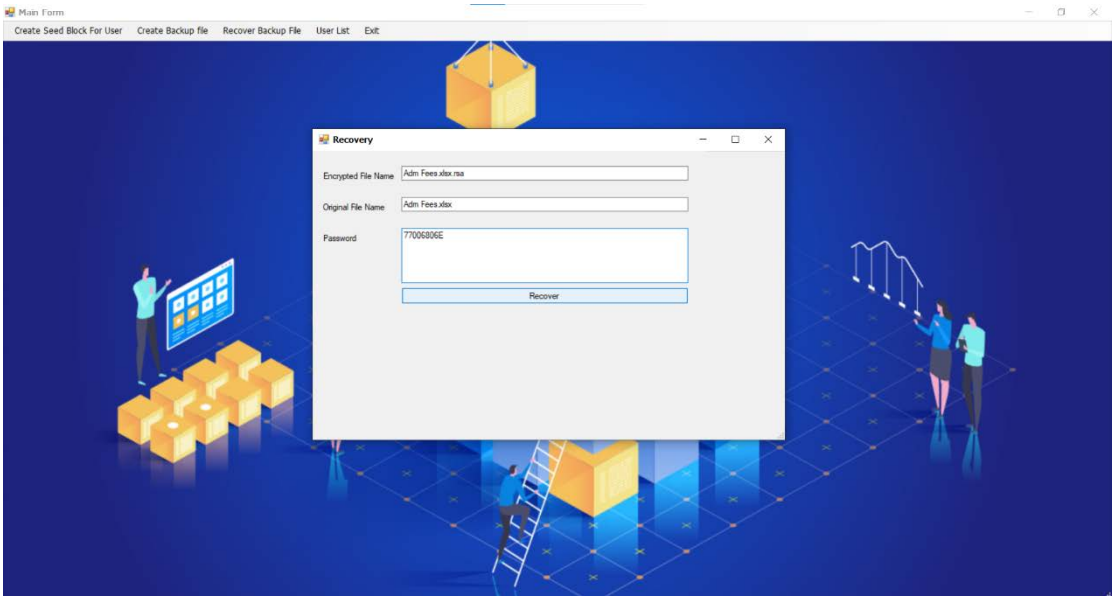


Figure 4.13 Recovery Backup File

During the operation of encryption and decryption, the original files and backup files will be presented in the main server. Although, files get deleted, the backup server can be generated as shown in Figure 4.14 and Figure 4.15.

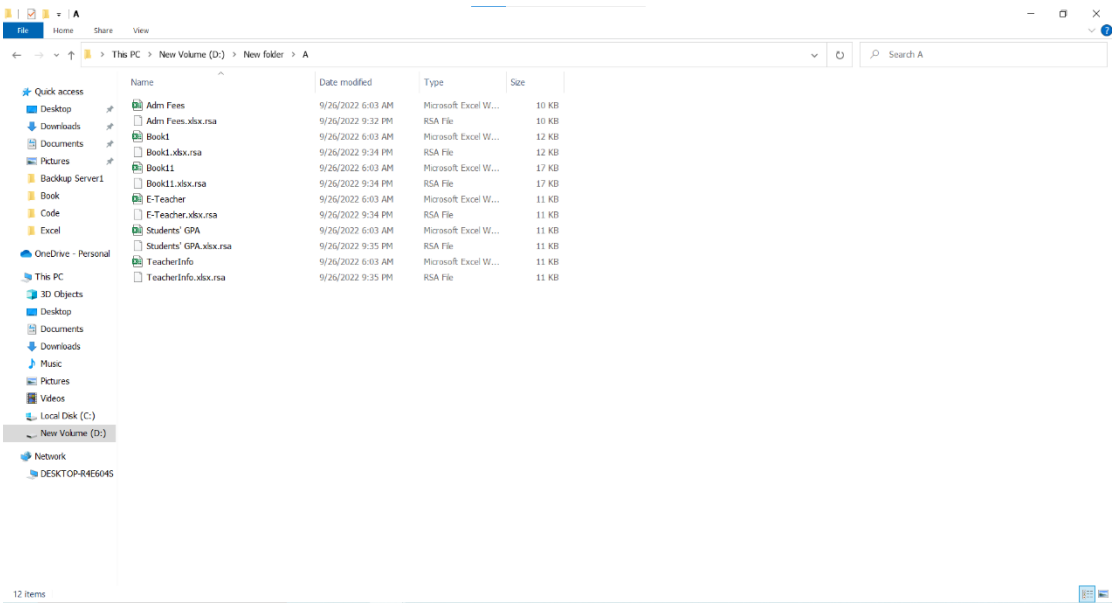


Figure 4.14 Files in Main Server

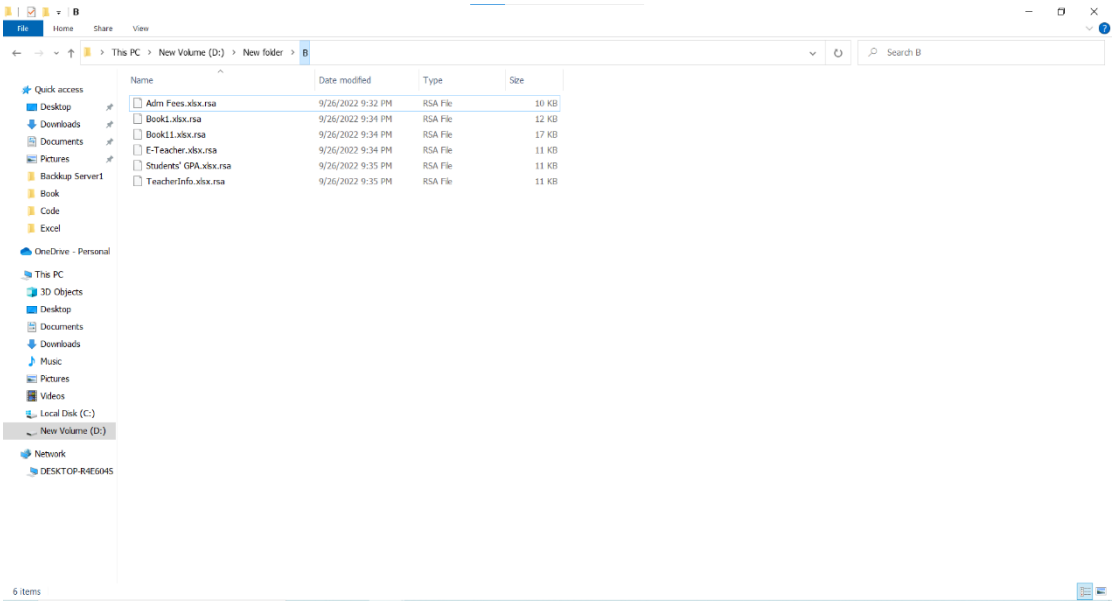


Figure 4.15 Files in Back Up Server

CHAPTER 5

CONCLUSION

Nowadays, a large amount of data is stored in the many ways and becoming very important to all the organization. Because it is the age of technology, almost all organizations are based on transaction processing. So, data backup and recovery are very urgent for data reliability and data availability. Seed Block Algorithm (SBA) is robust in assisting the users to collect information from any remote location in the loss of network connection and if file deletion has been occurred due to any reason, this system can also recover files. The SBA also focuses on the security issue for the back-up files stored at remote server, without using any of the existing encryption techniques. The SBA will take minimum time for processing recovery so that the issue corresponds to time can be solved.

5.1 Benefits of the System

The proposed system consists of the data recovery section which will be proceeded the failure file request transaction to become a successfully completed transaction. Therefore, this system is reliable for the data loss recovery during file request transaction processing; the original database and data backup are parallel stored. The system can also prove zero data loss according to the experiment results and the data can be retrieved when it decrypts with seed block of the particular users. So, this system has privacy and recovery time does not enormously increase as data increase.

5.2 Limitations and Further Extensions of the System

There are some limitations in this system. This system based on distributed storage system. So, it is dependent on having a network connection. If you are on a slow network, you may have issues accessing your storage. In the event you won't be able to access your file. This system can upload and recovery files with the following extensions: .pdf, .docx, .txt, .xlsx and.pptx. You can test recovery on another file types (e.g., image files). This system is not possible to restore files residing on users who are not in network connection. The software discussed above is a great effort to

bring more effectiveness to the whole system of recovery of files on backup server. There is a scope for modification an up gradation in the future.

AUTHOR'S PUBLICATIONS

[1] Myat Phoo Nge, Yu Wai Hlaing, "*University Data Recovery System Using Seed Block Algorithm*", The Proceedings of the Conference on Parallel & Soft Computing (PSC 2022), University of Computer Studies, Yangon, Myanmar, 2022.

REFERENCES

[1] Amman, P., et. al., "Recovery from Malicious Transactions", IEEE Transactions on Knowledge and Data Engineering, Vol. 15, 2015.

[2] B.W.Lampson and H.E.Sturgis., ”Crash recovery in a distributed data storage system”, Technical report, Xerox Palo Alto Research Center, April 2016.

[3] C.U., Orji., J.A. and Solworth. “Doubly distorted mirrors”, ACMSIGMOD internal conference on Management of data, Vol.22, NO.2, 2013.

[4] Engr. FaizullahMahar, “Role of Information Technology in Transaction Processing System”, Department of Electrical Engineering and Computer Science/IT, Balochistan University of Engineering and Technology, Khuzdar, Balochistan, Pakistan, 2015.

[5] Ghazi Alkhatib and Ronny S. Labban, “Transaction Management in Distributed Database Systems: the Case of Oracle’s Two-Phase Commit”, Senior Lecturer of MIS, Qatar College of Technology, Doha, Qatar and Computer & Communications Engineer; Consolidated Contractors International Company Athens, Greece; Alkhatib@qu.edu.sa and r.s.labban@ieee.org.

[6] J.M.Kent., “Performance and Implementation Issues in Database Crash Recovery”, PhD thesis, Princeton University, 2015.

[7] Mahantesh N. Birje, Praveen S. Challagidad, “Remote backup and recovery review: concepts, technology, challenges and security”, International Journal of Cloud Computing, InderScience Publishers, vol. 6, issue 1, 2017.

[8] Malinowski, E. an Chakravarthy, S. (2017), Fragmentation techniques for distributing object-oriented database, in D.W. Embley& R.C Goldstein, eds, ‘Conceptual Modeling – ER ‘97’, Vol . 1331 of lecture notes in computer science, springer, PP, 347-360.

[9] Mr. G. S. Narke, “A smart data backup technique for cloud computing using seed block algorithm strategy”, Comp. Dept. BVCOERI Nasik, India, International Research Journal of Engineering and Technology (IRJET) 2015.

[10] Ruchira. H. Titare, Prof. PravinKulurkar, “Remote Data Back-up and Privacy Preserving Data Distribution: A Review”, International Journal of Computer Science and Mobile Applications, Vol. 2, Issue. 11, November 2014.

[11] R.A.Lorie. “Physical integrity in a large segmented database”, ACM Transactions on Database Systems, Vol.2, NO.1, 2017.

[12] Tripathy, S. and B. Panda, “Post-Intrusion Recovery Using Data Dependency Approach”, Proceedings of the 2021 IEEE Workshop on Information Assurance and Security, pp. 156-160.

[13] Vladimir Zwass,“Foundations of Information System”, Fairleigh Dickinson University, McGraw-Hill Companies, Inc., International Editions 1998.

[14] Yongkun WANG, Kazuo GODA, and Masaru Kitsuregawa, “A Performance Study of Non-In-Place Update Based Transaction**”,** Institute of Industrial Science, the University of Tokyo, 2009.

[15] Vladimir Zwass,“Foundations of Information System”, Fairleigh Dickinson University, McGraw-Hill Companies, Inc., International Editions 1998.