Routing Journal Operations on Disks Using Striping With Parity[1]

Ramzi Haraty and Fadi Yamout
Lebanese American University
P.O. Box 13-5053
Beirut, Lebanon
Email: rharaty@beirut.lau.edu.lb, fadiyam@inco.com.lb

ABSTRACT

Despite technological advances in the reliability of magnetic storage media, the incidence of data loss continues to rise [9]. Data storage remains a fragile science, and data's susceptibility to damage from both natural and human sources remains high. Increasing storage capacities amplify the impact of data loss. As more and more data is stored in smaller and denser areas, mechanical precision becomes crucial. Backup technology and practices have failed to adequately protect data. Most computer users rely on backups and redundant storage technologies as their safety net in the event of data loss. Our research describes a recovery architecture that will achieve recovery with a good performance and establish fault tolerance at low cost.

Keywords: Journal Operations, Disk Mirroring, Disk Stripping with Parity, Check Pointing, and Garbage Collection.

1. Introduction

In this paper, a new architecture will be drawn using distributed journals and files, in such a way that logging to journals used for recovery and the recovery process itself will improve in performance. This improvement will be weighted against the cost needed for establishing fault tolerance on the system. The paper is organized as follows: Section 2 discusses the types of storages that exists, the types of failures in database systems that could affect data, and a feedback on some of the methods and utilities used for recovery. Section 3 presents our proposed architecture. And section 4 presents a conclusion.

2. Existing Techniques

The proposed architecture of the recovery process discussed in this paper, require us to examine some of the issues related to computer hardware and recovery methods.

2.1 Stable Storage and Volatile Storage

Stable Storage is an area of memory that is resistant to processor and operating system failures. It models secondary storage media such as disk and tape on typical computer systems. Files reside on stable storage. Volatile storage is an area of memory that is vulnerable to hardware and operating system failures. It models main memory on typical computer systems. To avoid accessing stable storage to process

[1] Proceedings of the International Conference on Research Trends in Science and Technology (RTST). March 6 – 8, 2000. LAU – Beirut, Lebanon.

every Read and Write, we would like to keep a copy of the database in volatile storage.

2.2 Type of Failures in Database Systems

Generally speaking, there are three main types of failures in database systems. Each one affects the data in a way and needs special mechanisms to recover the data. The first is the **Transaction Failure**. Many of the transaction's failures are due to incorrectly programmed transactions and data entry errors. The second type of failure is the **System Failure**. The system fails when, for example, there is a power failure or when the operating system fails. When this happen, there will be a loss or corruption of the content of the volatile storage. The third type of failure is the **Media Failure** and it is the worst. In a media failure part of the stable storage is destroyed such as a sector of the disk becomes damaged. IN this case, the damaged hardware should be replaced. A **Restart** mechanism is initiated automatically or manually after the system is on and ready to run or after a transaction failure.

2.3 What is a Log?

Whenever a transaction runs, the recovery manager, which is a part of the database system, adds for each of the transaction's operations an entry in a file called the **Log File**. These entries describe the before images of the data items in Stable Storage. This process is called **Logging** (see Figure 1). Therefore, a Log is a representation of the history of execution which is used later by the Recovery Manager to restore the last committed values of data items in the event of failure [1]. The records in the Log are called **"before image"** because they are written to the Log before each activity takes place. Recovery Manager reads the Log backward and stops when either all records in Log have been exhausted or a commit operation is reached. This process is called **Backward Recovery** [9].

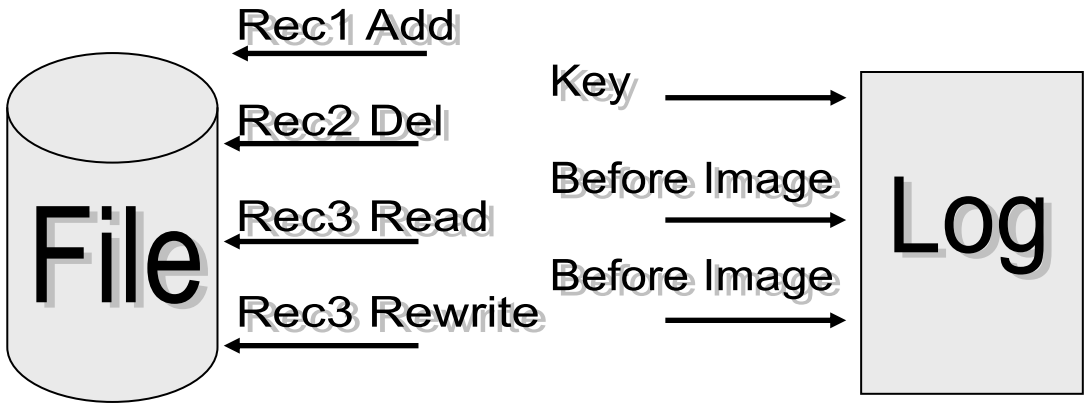

Figure 1. The Logging Process.

2.4 Type of Logs

There are two types of Logs, and each differs in its function. The first type is called **Dynamic Log** and is stored in volatile storage. This Log helps only recover

transaction failures. The reason for that is that when a transaction aborts, the Recovery Manager recovers the files from the journal in volatile storage, but when the system fails, volatile storage is lost and other mechanisms should be considered to recover. The second type is called the **Journal Log** and is stored in stable storage. The Journal Log helps recover transactions, system or media failures. In the Journal Log "before images" stored earlier or a new type of images called **"after images"** that are written in the Log after each operation has taken place, could be stored as well. Therefore, the Recovery Manager might read the Log forward (**Forward Recovery**) or reads it backward (**Backward Recovery**). The reason for doing this is that whenever a media failure occurs and the disk is replaced, data should be restored. To restore new data items written during the day of a failure, a Forward Recovery is needed. Forward Recovery recovers data items written by committed transactions and may recover data items updated by transactions that were active at the time of the failure and did not commit. Therefore, any restart of these transactions leads to a duplication of operations. To avoid this, a Backward Recovery undoes the operations, back to the consistent state of the file [5].

2.5 Log Buffer

Database systems operations are not immediately written to the Log. They are fetched first to a Log Buffer in volatile storage that accumulates many operations before writing them to the Log. The entries are flushed from the Log Buffer to the Log in stable storage when either the buffer fills up totally, fills up *x* percent or when a transaction commits [9]. Many disadvantages arise. If the operations are written to the log each time a transaction commits, that defeats the main reason for using the Log Buffer since the buffer could be filled partially. The commit request could be delayed until the buffer is filled up to a percent and then the system flushes the buffer to the Log. The disadvantage is that many transactions will be halted from committing while waiting for the buffer to fill up.

2.6 Check Pointing

In order to help recovery perform better, two methods are used. One is Check Pointing and the other is Garbage Collection. Check Pointing is an activity that writes information to the Log in order to reduce the amount of work the Restart has to do after a failure. Remember that Restart scans the Log backward undoing the effects that the operations did on the data items. Restart may scan the whole Log. Check Pointing is a state in the database where all transactions committed and where there are no active Transactions. Restart scans the Log backward, beginning from the end, until it reaches the last Check Point marker [2].

The disadvantage of Check Pointing is that users may suffer a long delay waiting for long active transactions to commit.

2.7 Garbage Collection

The other method that helps recovery perform better is Garbage Collection. Garbage Collection restricts the amount of information in the log. The Recovery Manager removes the operations that will never be needed, such as:

- Operations related to aborted transactions, and

- Transaction T_i has committed but some other committed transaction wrote into x after T_i did.

The chief disadvantage of this approach is that a long running transaction prevents the recovery manager from garbage collecting the log.

2.8 Disk Mirroring

Two of the methods used for fault tolerance are disk mirroring and disk striping with parity. Disk mirroring keeps a duplicate copy of each disk. Therefore, each update is written to both disks. When the primary disk fails, operation will be switched to the mirrored one. Using duplexing that takes mirroring one step further by using an additional disk controller could upgrade mirroring. The Log itself could be mirrored [6]. The major disadvantage of disk mirroring is that the size of the data and the write process is doubled.

2.9 Disk Striping

Another method to establish fault tolerance is called disk striping with parity. First disk striping divides the data coming from the files into 64K blocks and spreads it equally in a fixed rate and order among all disks in an array. Therefore, the first 64K of data is written to a stripe on disk 1, the second 64K is written to a stripe on disk 2, and so on. At least two physical drives are needed to apply this method. The good thing about it is the performance because multiple disk controllers result in better performance [6].

2.10 Disk Striping with Parity

If we apply the parity option to the disk striping, the parity will be written across all of the disks in the array. The data and parity information are arranged so that the two are always on different disks. The parity stripe is used to reconstruct data for a failed disk. If a single disk fails, enough information is spread across the remaining disks to allow data to be completely reconstructed [6].

3. Solutions Proposed

How do we make use of all of these? We combine some of these techniques and add new ideas to come up with a new recovery architecture that achieves a better performance with a low cost for establishing fault tolerance.

3.1 Distribute Journals

To start with, we can determine for all applications the relationships between the transactions and the files, partition the applications into **atomic entities** consisting of related transactions and files, define a separate journal for each entity and distribute the journals on different disks as shown in Figure 2.

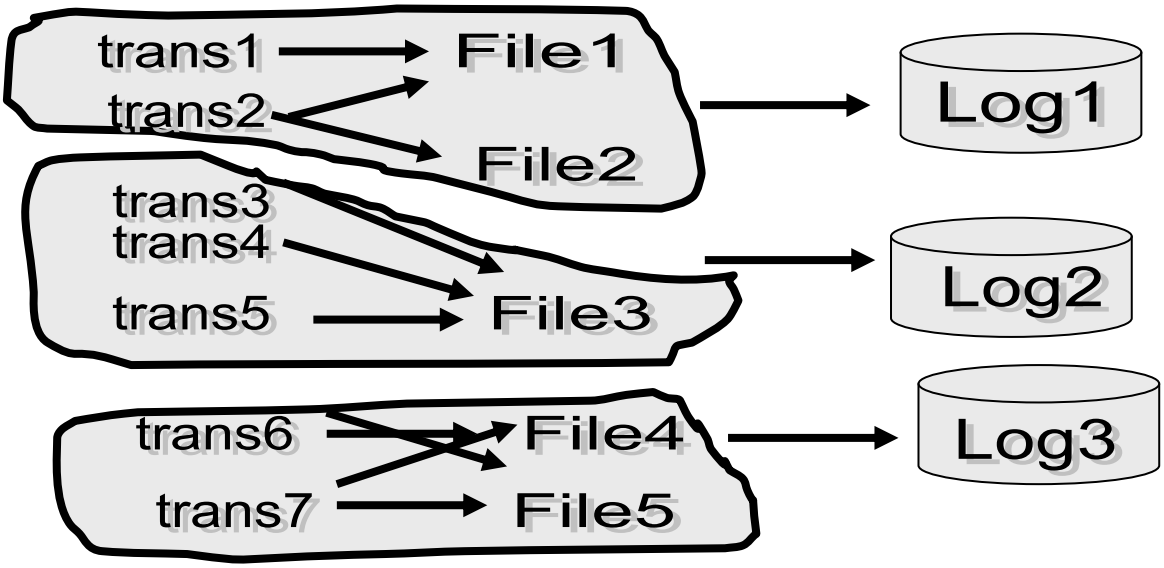

Figure 2. Define Atomic Entities.

We might find that some of the files are still related between two entities. We could have some duplicates of them. Since we are aiming for better performance, we could improve this at the expense of redundancy. Some of the advantages with distributed journals is that multi-volume data sets improve the performance of logging since we are writing on many journals at the same time. Another advantage is that long running transactions affecting an atomic entity will not prevent from activating garbage collection since garbage collection could be activated for each journal alone and at different times. As for the process of check pointing, the system will not wait for all active transactions to commit or abort in order to mark a check point. In addition, aborted transactions will not affect the performance of other transactions in different entities, and the log buffers are smaller and filled by the recovery manager in parallel.

3.2 Files Distributed on Multi-Volume Data Sets

But still we have that whenever a failure occurs, the performance of the recovery process is not improved unless the files are distributed on multi-volume data sets [3]. Another disadvantage that still exists is the possibility of a journal that is defined on a single disk, to run out of space and therefore causes transactions to abort. To achieve better performance when recovering data, files should be distributed on multi-volume data sets as depicted in Figure 3.

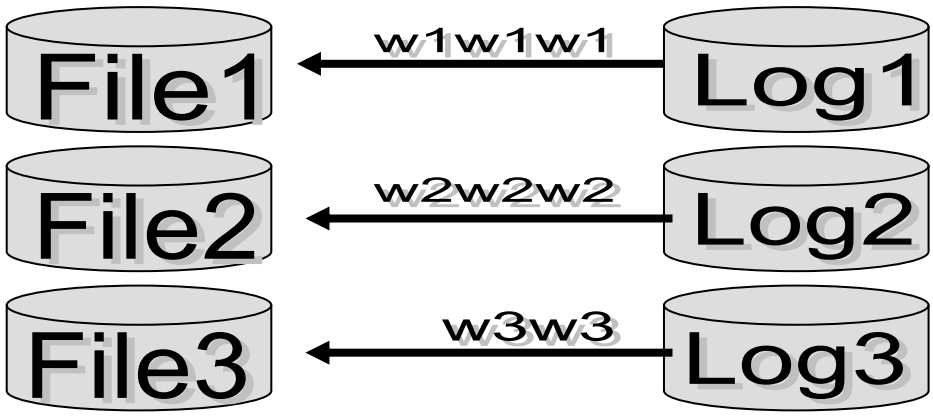

Figure 3. Recovering to Distributed Files.

Multi-volume data sets improve performance since they are writing on many files at the same time. One of the advantages is that there will be separate backup and restore for files. Another advantage is that the process to recover aborted transactions will not affect the performance of other running transactions in different entities.

3.3 Journal Should be Mirrored

Journal should be mirrored for the following reason: when the disk that contains files crashes, data on the disk could be recovered from the archive and the journal as in Figure 4. But if the disk containing the journal crashes, then the process of logging to the journal restarts from the beginning, and a part of the after images are lost. To establish fault tolerance, mirroring requires a large quantity of disks since we have distributed the files and the journals on many of them.

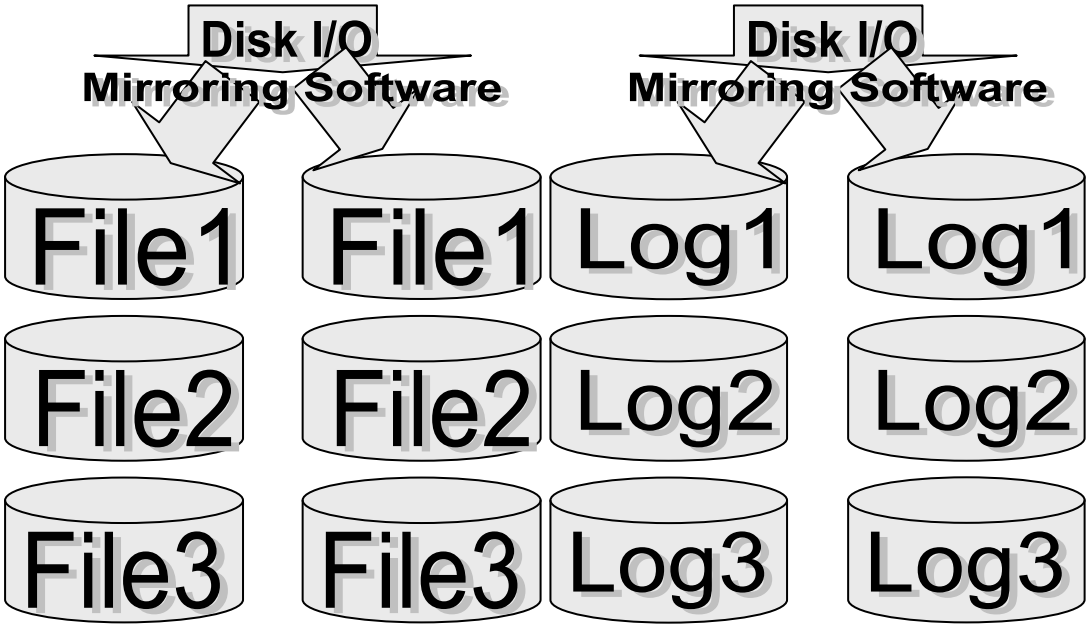

Figure 4. Mirroring Files and Journals.

3.4 Use Disk Striping

We can establish fault tolerance without adding any disks by using disk striping. Disk striping is better option than mirroring since with disk striping there will be no need to add any disk to establish fault tolerance to the system (one additional disk may be needed. The reason is that some space will be allocated for the parity written across all of the disks in the array).

3.5 Advantages of Distributed Journals and Files on Disk Striping

For completeness, we list all the advantages of distributed journals and files with disk striping:

- Multi-volume data sets improve performance (writing on many journals and files at the same time).
- Long running transaction affecting an atomic entity will not prevent from activating Garbage Collection.
- The system will not wait for all active transactions to commit or abort in order to mark a Check Point.
- A journal will not run out of space.
- Aborted transaction will not affect the performance of other transactions in different entities.
- Log buffers are smaller and filled by the Recovery Manager in parallel.
- Fault Tolerance will not cost a lot of disks.
- Separate Backup and Restore for the Files could be performed.

3.6 Applications are Journal Dependent

Many of the application programs may specify the journal identifier on which logging will be done. Application programs may have their own journal records format and choose to have their independent journal. Some disadvantages exist. We may decide that transactions that used to write to some journals will now route their operations to another journal. Therefore, we have to change the journal identifier specified in the applications. Any change in the architecture of the atomic entities would require also the change of the journal identifiers. This means that each program with a change in its journal identifier has to be modified, compiled and tested for running.

3.7 Routing Operations

A solution to the above mentioned problem is to let all the applications write their operations to a single journal identifier along with the transaction identifier to a journal log called, for example, the trigger journal. A **router** loaded in main memory decides the destination journal where to send an operation after consulting a routing table. A **routing table** contains pairs (***T***, J) where ***T*** is the identifier of the source transaction, and J is the identifier of the destination journal. ***T*** could be a file, an operation, or an application as can be seen in Figure 5.

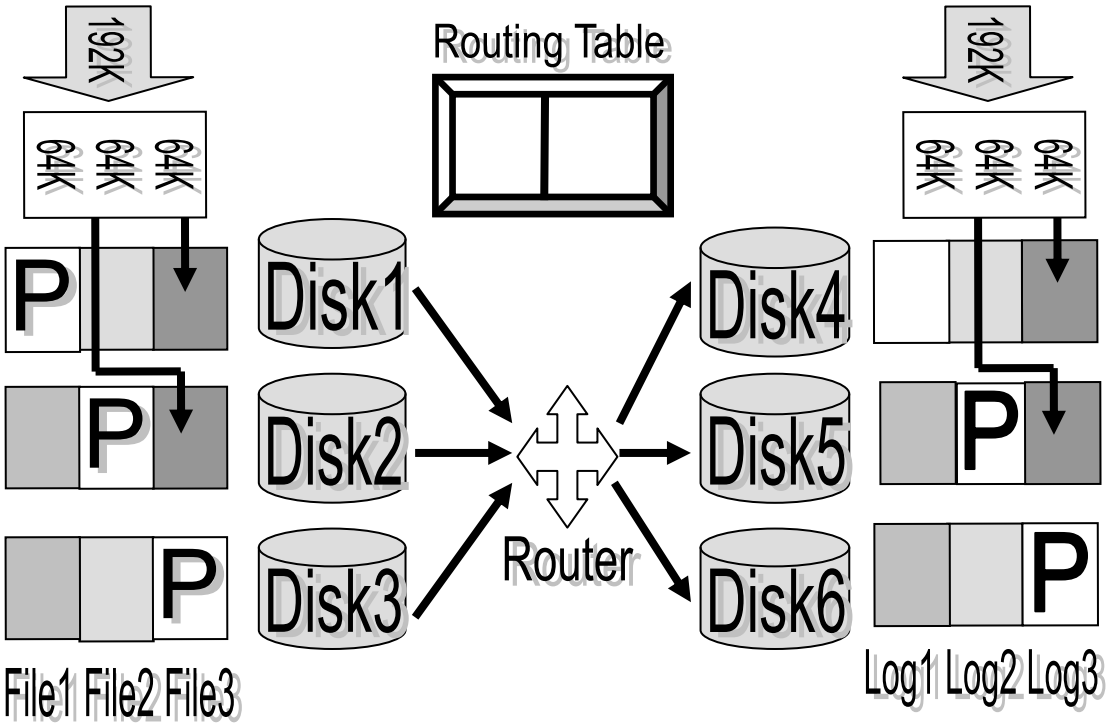

Figure 5. Routing Operations to Distributed Journals.

4. Conclusion

While trying to devise new recovery ideas, we tried to weigh between establishing a fault tolerance to the system and improving the recovery performance. The disk containing the journal should never crash, because the process of logging to the journal restarts from the beginning, and a part of the after images will be lost. Distributed journals on multi-volume data sets improve performance, since data is written on many journals at the same time. To establish fault tolerance, mirroring requires a large quantity of disks, since files and journals are distributed on many of them. Defining distributed journals and files on disks using striping with parity will not only improve performance but will also keep the cost low for establishing fault tolerance to the system. Recovery improvement is not only a better way to recover the data, but also to minimize the cost of applying the new recovery architecture to the system.

References

[1] *Doug Swords,* Storage Management and disaster recovery, **Technical Support.** 1995**.**

[2] *Jeffrey C. Barnard,* An introduction to caching, **Technical Support,** 1995.

[3] *Steve Prior,* Too much of a good thing: Multi volume Data Sets, **Technical Support,** 1996**.**

[4] *Stephan force,* Maintaining system and data availability on a Microsoft Windows NT networks, **Technical Support,** 1994**.**

[5] *Daniel Kitay,* The Oracle DBMS Architecture: A technical introduction, **Technical Support,** 1997**.**

[6] *Laurance Clarke,* Oracle on Windows NT: A fail Safe solution, **Oracle Magazine,** 1998**.**

[7] *Erik Peterson,* Migrating to Oracle8: Quick start on new features, **Oracle Magazine,** 1998**.**

[8] *Microsoft Press,* Network administration and support, **Networking Essentials,** 1997**.**

[9] Centralized Recovery, Distributed recovery, Replicated data, **Recovery and Concurrency Control.**

Biography

Ramzi A. Haraty is an Assistant Professor of Computer Science at the Lebanese American University in Beirut, Lebanon. He received his B.S and M.S degrees in Computer Science from Minnesota State University–Mankato, Minnesota and his Ph.D. in Computer Science from North Dakota State University – Fargo, North Dakota. His research interests include database management systems, artificial intelligence, and multilevel secure systems engineering. He has well over 36 journal and conference paper publications.

Fadi Yamout is currently responsible for the planning and organization unit at the planning and Information Technology department at the Middle East Airlines in Beirut, Lebanon. He received his B.S. and M.S. degree in Computer Science from the Lebanese American University and his B.S. in Civil Engineering from the Christian Brother's College Tennessee, USA. Research Interests include Information Retrieval, Logging, and theoretical sciences.