CMSC 206: Database Management Systems

Recovery

Thomas Laurent

14/11/2020

University of the Philippines Open University

Foreword: SET

I saw that the Student Evaluation of Teachers (SET) survey was available. Please do fill it in, I am always eager to know what I can do better (but also what I did well :)).

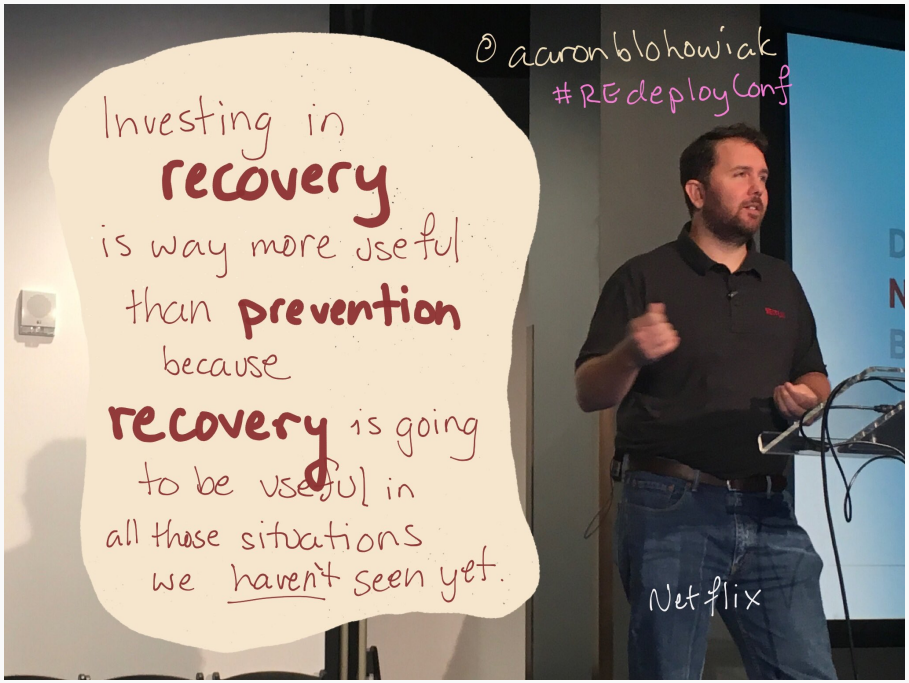
Foreword: Quiz

Typhoons, pandemic, etc are making everyone's life more complicated than they ought to be so let's take a break from the quizzes this week. (No, I am not trying to get you in a good mood before you fill in your SET survey :))

Introduction

Introduction

This week's topic is recovery. Recovery is how the DBMS copes when a failure happens and it is strongly tied with the ACID properties we saw at the beginning of the class.



(from *https://twitter.com/jessitron/status/1030141310039023616*)

Introduction

There are many different ways recovery mechanisms can be implemented in a DBMS, and as usual how things actually work will depends on your system and how it is configured. Your system's documentation should always be your main source of information.

In these slides we will introduce the main concepts linked with DB recovery and give a very high level view of what goes on under the hood.

Crashes

Crashes

Crashes can happen for all kinds of reasons, even with the best thought out and well implemented system, e.g:

- Physical problems: power outage, hardware failure, ...
- Software error: division by 0, buffer overflow, ...
- Concurrency deadlock
- ...

Our system should be able to recover from these crashes. Hopefully rapidly and with no (or minimal) data loss.

Crashes and ACID properties

Durability is the most obviously linked to crashes: once a transaction is committed it should stay committed, no matter what, i.e. we should not lose data even if there is a crash.

Atomicity and consistency are also linked with crashes: if a crash happens in the middle of a transaction we want to make sure the whole transaction is either committed or cancelled and the database is kept in a consistent state after the crash.
In the case of a soft/local crash (e.g. error while executing the transaction) we just rollback, but for hard/system crashes it is more complicated.

Backups and logs

Backups

Backups are a snapshot of the database (or of a system in general) in a consistent state at a past point in time.

Dump and restore tools are available for most DBMSses to create backups and restore the database to the state represented by a backup

Test your backups

Making backups without verifying that you can restore them just leaves you with Schrödinger's backup: maybe you can restore your DB and recover from a crash, maybe not.

Logs

The log journal keeps a trace of all the operations that happen on the DB. It can be used to replay these actions at a later date if we need to perform operations which effect was lost after a crash. e.g:

Start transaction T

Write X

Write Y

Delete Z

Commit T

The system regularly makes synchronisation points where all operations until that point are committed and written to disk and we know the system is in a consistent state.

Recovery

Media (disk) crashes VS other crashes

If the disk was damaged in some way in the crash and we lost data from the database we then need to restore a backup. We can then replay the committed operations in the logs from the time of the backup up to the time of the crash to get back to the situation before the crash.

If no data was lost during the crash then we do not necessarily need to restore a backup and we can recover using the logs.

Replaying the logs

The idea here is we want to replay (redo) committed operations that happened before the crash but were not yet written to disk, or undo operations that were written to disk but were not committed at the time of the crash. There are two main strategies for this:

- NO-UNDO/REDO
- UNDO/REDO

NO-UNDO/REDO

This strategy is used in *deferred update* systems, where changes are written to disk only after they are committed. Then, when a crash happens, we know that all the data on disk corresponds to committed operations, so we do not need to undo any operations. There might be operations that were committed but not yet written to disk at the time of the crash though. In this case we need to redo these operations

UNDO/REDO

This strategy is used in *immediate update* systems where changes can be written to disk before a transaction is committed. In this case we might have changes committed to disk when the crash happens but that were not yet committed, as well as committed changes that were not yet written to disk. We then need to undo the non-committed changes and redo the committed ones.