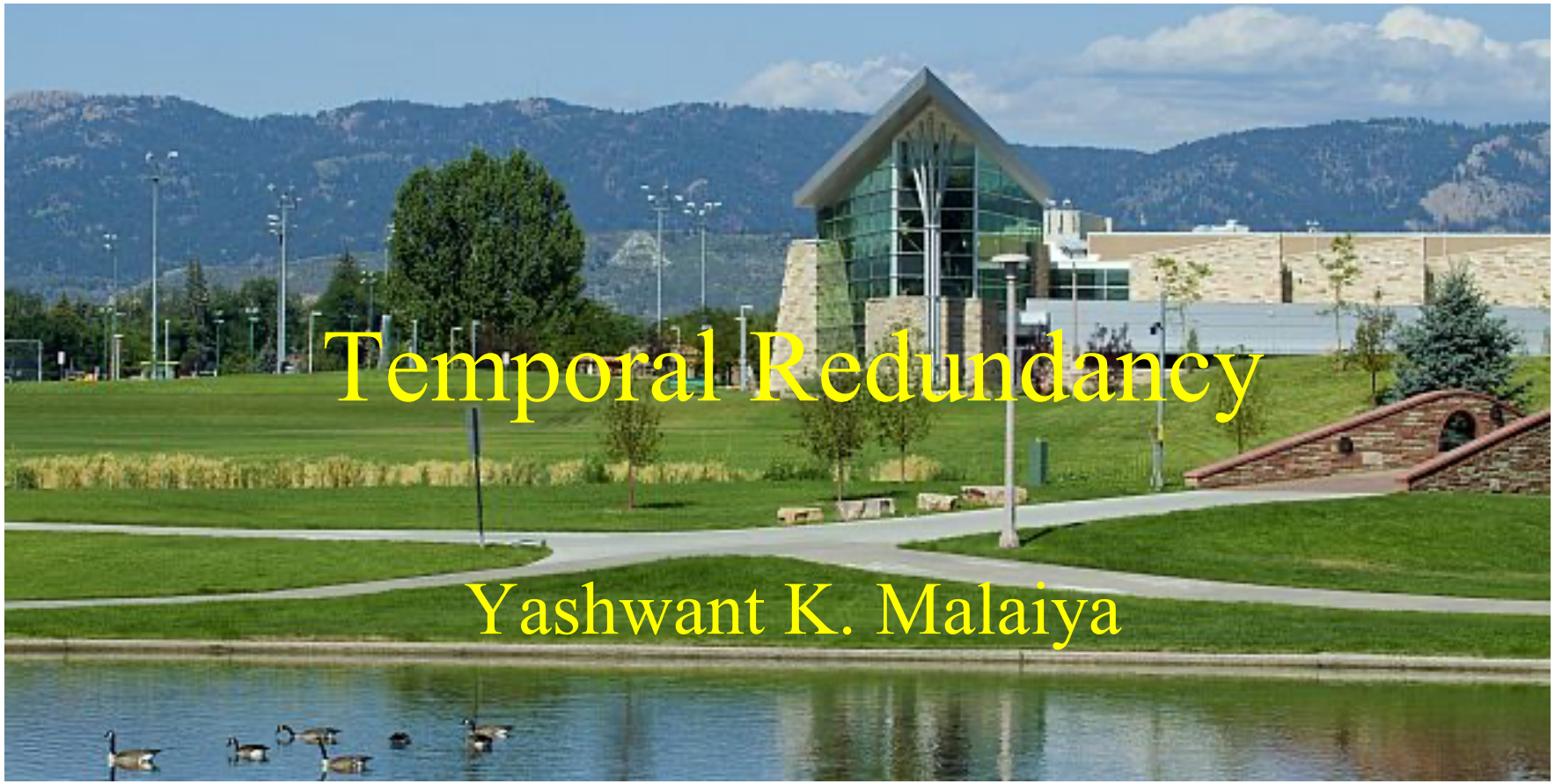

Temporal Redundancy
Yashwant K. Malaiya

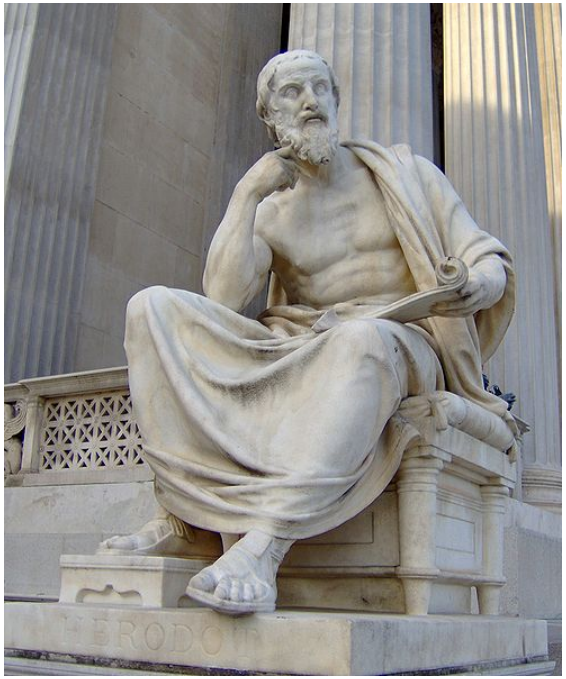
Murphy's Law

- *Anything that can go wrong, will.*
  - (Actually not by Murphy but by Finagle)
- *To every law there is an exception.*
- CS530 laws:
- *Anything that can go wrong, it eventually will*, but
  - It may not go wrong for a while
  - It may not go wrong the next time
  - Only one thing may go wrong at a time

Herodotus 4th cent BC on the Persians

"If an important decision is to be made, they [the Persians] discuss the question when they are drunk, and the following day the master of the house where the discussion was held submits their decision for reconsideration when they are sober. If they still approve it, it is adopted; if not, it is abandoned.

Conversely, any decision they make when they are sober, is reconsidered afterwards when they are drunk."



Temporal Redundancy

Effective for *time limited* faults.

Detection:

- Serial transmission: parity/CRC

Fault tolerance:

- Bus errors: Instruction retry
- Data-bases: check-point & roll-back
- Bad/lost network packets: retransmission

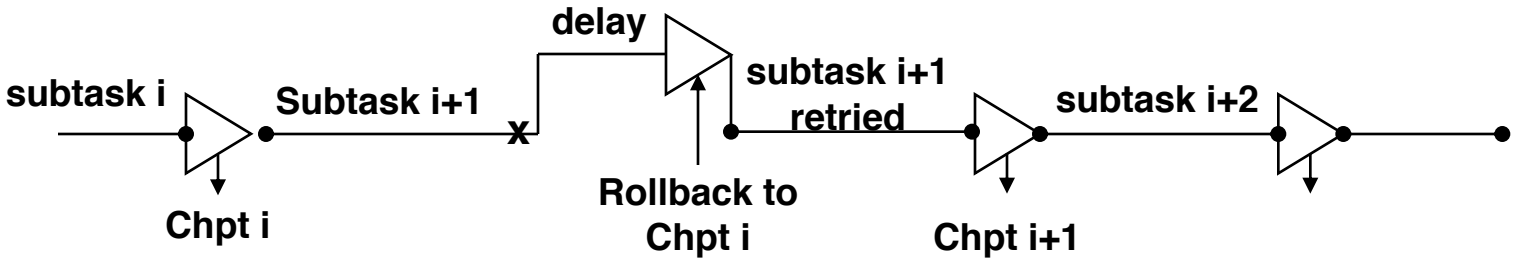
Requirement:

- Save previous state(s) in *stable storage*

Terminology

- Check-pointing: **saving part of the *process state***
  - Registers affected
  - Context
  - Part of the state (registers, memory) affected by next process segment
  - Entire data base etc.
- Rollback: **reestablishing a state of the process**
- Audit Trail: **chronological record (log) of all transactions**
- Retry: **reexecution after rollback (inc. audit-trail reprocessing)**

Strategy



Failed retry?

- Temp fault still active when rollback done. Do another rollback.
- Fault permanent. Reconfigure hardware before rollback.
- Checkpoint i info bad. Rollback to checkpoint i-1.

Analysis of Overhead

- Assumptions :
  - ▷ Fault arrival rate : λ , interchkpt time : T
  - ▷ Additional retry time $\propto$ duration from last chkpt to error
  - ▷ No inputs/errors during chkpt/rollback
- Overhead per T :
  - ▷ $O(T) = F + V(T)$

    where F : fixed time to save/load chkpt info

    V(T) : Average retry time
  - ▷ Average retry time :

    $V(T) = P\{\text{error during T}\}.\text{avg error overhead}$

    $= \lambda T(F + k\frac{T}{2})$

    where k is utilization factor. Note overhead includes time lost due to error and time to rollback.

Justification?

Why T/2?

Analysis of Overhead (2)

- Hence fractional overhead $\rho(T)$:

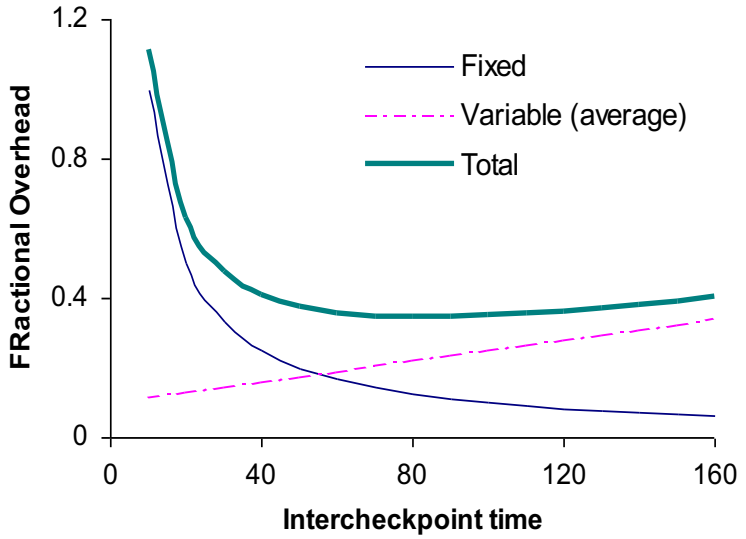
$$\rho(T) = \frac{O(T)}{T} = \frac{F}{T} + \lambda F + \frac{\lambda k}{2} T$$

Minimum occurs at

$$\frac{d\rho}{dT} = -\frac{F}{T^2} + \frac{\lambda k}{2} = 0$$

$$\therefore T_{opt} = \sqrt{\frac{2F}{\lambda k}}$$

$$\text{Note}: k = \frac{\text{transaction arrival rate}}{\text{transaction processing rate}}$$



Ex: λ =0.01, k=0.3, F=10

yields T_{opt}=81.6 (above)

In-lecture exercise

Given λ =0.001/sec, k=0.25, F=100 millisec

Find $T_{opt.}$

Networks: Packet Retransmission

Information is divided into packets

- Each packet contains destination address, packet number, information slice, CRC
- Packet is routed through routers through the internet
- At the destination, each packet is checked.
- Packets are assembled back and delivered to destination.
- Should packets be
  - small or
  - large?

Networks: Packet Retransmission

Stop and Wait protocol

- Packet sent from node A to node B
- Possibilities
  1. Packet delivered correctly: B sends acknowledgment
  2. Packet delivered corrupted: no ack or ack bad
  3. Packet lost: no ack
- Cases 2,3
  - A sends the packet again
  - “ARQ”: automatic repeat request

Stop & Wait Utilization

- When a frame is corrupted, lost (or ACK is lost),
  it is retransmitted.
- Error – free utilization of a link between two nodes (*stop & wait*) :

$$U = \frac{1}{1 + \frac{2t_p}{t_f}},$$

where t_p : propagation time, t_f : frame duration

- If p is P{a frame lost}, then of n frames sent, n.p are lost.

$$U_e = \frac{1 - p}{1 + \frac{2t_p}{t_f}}$$

Stop and Wait utilization(2)

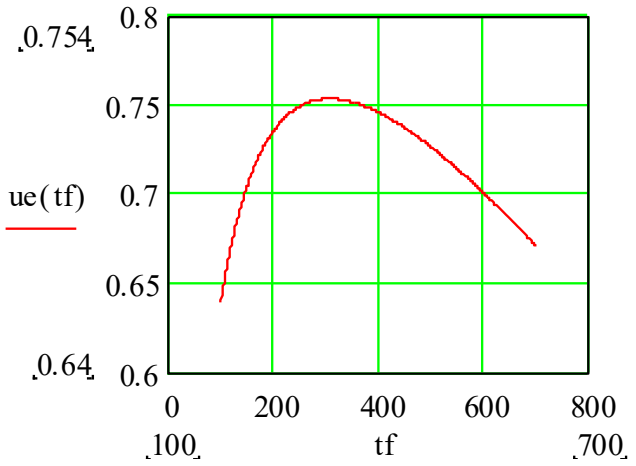
- If $p = c.t_f$ then

$$U_e = \frac{1\text{-}c.t_f}{1 + \frac{2t_p}{t_f}}$$

- Optimal frame size is then

$$t_{fopt} = \frac{-4t_p + \sqrt{16t_p{}^2 + 8\frac{t_p}{c}}}{2}$$

- Higher performance ($U \approx 1$) obtained by sending multiple frames (sliding-windows). A copy of all frames must be saved until acknowledged.



Ex: t_p=25, c=0.0004.

Gives optimal t_f=307

Undo

- Undo recovers from human mistakes (or bad decisions).
- Requires recording a trail and saving any deleted/overwritten information. This may take significant memory.
- Cannot undo some operations.
- Multilevel undo:
  - last-in/first-out
  - Selectable level
- Redo.

References

- A Survey of Analytic Models of Rollback and Recovery Stratergies, Chandy, K.M.; Computer, Volume 8, Issue 5, May 1975 Page(s):40 - 47

Transactions

- A **transaction** is an action, or a series of actions, carried out by a single user or an application program, which reads or updates the contents of a database.
- Database is in stable storage (Disk). Intermediate computations are done in volatile memory.
- A **commit** refers to the saving of data permanently after a set of tentative changes. A commit ends a transaction and allows all other users to see the changes.

Transfer $50 from account A to account B
Read(A)
A = A - 50
Write(A)
Read(B)
B = B+50
Write(B)

Based on N. Alechina

Atomicity and Consistency

- Transactions are the unit of recovery, consistency, and integrity as well
- ACID properties
  - Atomicity: can't be executed partially
  - Consistency
  - Isolation
  - Durability

Atomicity and Consistency

- Transactions are atomic – they don't have parts (conceptually)
  - can't be executed partially; it should not be detectable that they interleave with another transaction
- Consistency: Transactions take the database from one consistent state into another
  - In the middle of a transaction the database might not be consistent

Isolation and Durability

Isolation: The effects of a transaction are not visible to other transactions until it has completed

- From outside the transaction has either happened or not. A consequence of atomicity.

Durability: Once a transaction has completed, its changes are made permanent

- Even if the system crashes, the effects of a transaction must remain in place

Example of transaction

Transfer $50 from account A to account B

Read(A)
A = A - 50
Write(A)
Read(B)
B = B+50
Write(B)

- Atomicity - shouldn't take money from A without giving it to B
- Consistency - money isn't lost or gained
- Isolation - other queries shouldn't see A or B change until completion
- Durability - the money does not go back to A

The Transaction Manager

- The transaction manager enforces the ACID properties. It schedules the operations of transactions
  - COMMIT and ROLLBACK are used to ensure atomicity
  - Locks or timestamps are used to ensure consistency and isolation for concurrent transactions
  - A log is kept to ensure durability in the event of system failure

COMMIT and ROLLBACK

- COMMIT signals the successful end of a transaction
  - Any changes made by the transaction should be saved
  - These changes are now visible to other transactions
- ROLLBACK signals the unsuccessful end of a transaction
  - Any changes made by the transaction should be undone
  - It is now as if the transaction never existed

The Transaction Log

The transaction log records the details of all transactions

- Any changes the transaction makes to the database
- How to undo these changes
- When transactions complete and how
- The log is stored on disk, not in memory
  - If the system crashes it is preserved
- Write ahead log rule
  - The entry in the log must be made before COMMIT processing can complete

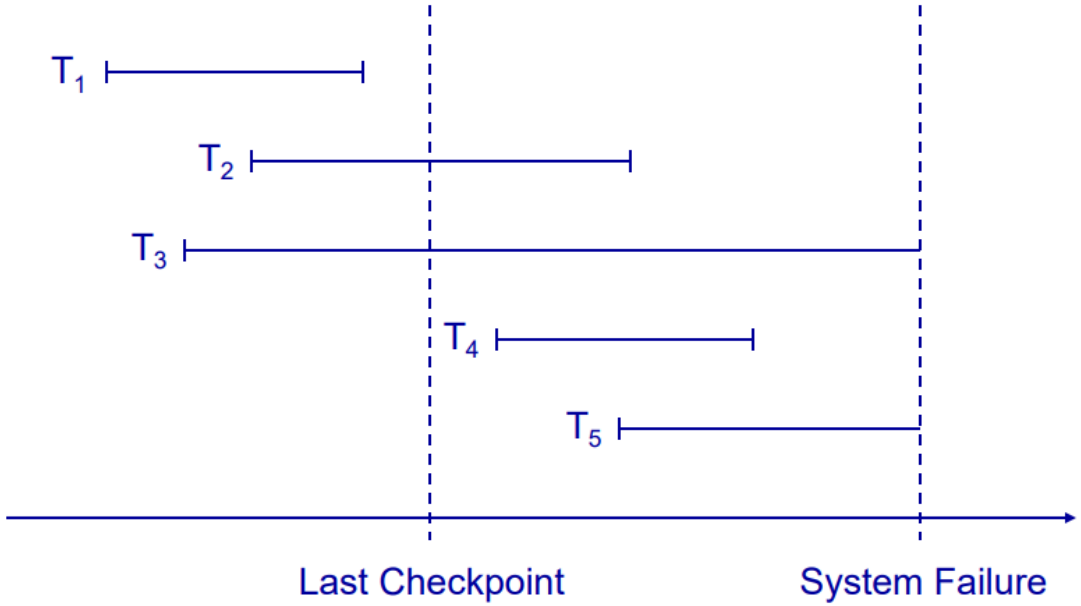
System Failures

- A system failure means all running transactions are affected
  - Software crashes
  - Power failures
  - Assumption: The stable storage (disks) is not damaged
- At various times a DBMS takes a checkpoint
  - All committed transactions are written to disk
  - A record is made (on disk) of the transactions that are currently running

System Recovery

- Any transaction that was running at the time of failure needs to be **undone** and restarted
- Any transactions that committed since the last checkpoint need to be **redone**

System Recovery using Log



- Transactions of type T1 need no recovery
- T3 or T5 need to be **undone** and restarted (disrupted)
- T2 or T4 need to be **redone** (finished/committed)

Recovery using logs

Backwards recovery

- We need to undo some transactions
- Working backwards through the log we undo any operation by a transaction on the UNDO list
- This returns the database to a consistent state

Forwards recovery

- Some transactions need to be redone
- Working forwards through the log we redo any operation by a transaction on the REDO list
- This brings the database up to date

Issues to address

- Recovery from Disc Failure
  - Restore the database from the last backup
  - Use the transaction log to redo any changes made since the last backup
  - Issue: the transaction log is damaged. Keep a backup on a separate device
- Concurrency issues: multiple users
  - Preserve isolation using locks