Logistics Updates

- P1 checkpoint due (11:59EST, Oct 2nd)
  - Part A Due: Friday, Oct 13th
  - Part B Due: Oct 23rd
- HW2 released today (Oct 2nd)
  - Due Friday, Oct 13th
  - (*No Late Days*) => time to prepare for Mid term
- Mid term Monday Oct 16th – 6pm (DH 2210)
  - Will cover everything until the first half of class
  - Midterm Review: Sunday 10/15 @10am, DH2315
- Waitlists have been cleared, email if you think not!
Today's Lecture Outline

• Real Systems (are often unreliable)
  • We ignored failures till now
  • Fault tolerance basic concepts
  • GOAL: Recovering a system back to a consistent state

• Fault Tolerance – Checkpointing

• Fault Tolerance – Logging and Recovery
What is Fault Tolerance?

- Dealing successfully with partial failure within a distributed system
- Fault tolerant -> dependable systems
- Dependability implies the following:
  1. Availability
  2. Reliability
  3. Safety
  4. Maintainability
Dependability Concepts

- **Availability** – the system is ready to be used immediately.

- **Reliability** – the system runs continuously without failure.

- **Safety** – if a system fails, nothing catastrophic will happen. (e.g. process control systems)

- **Maintainability** – when a system fails, it can be repaired easily and quickly (sometimes, without its users noticing the failure). (also called Recovery)
  - What’s a failure? : System that cannot meet its goals => faults
  - Faults can be: Transient, Intermittent, Permanent
## Failure Models

<table>
<thead>
<tr>
<th>Type of failure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crash failure</td>
<td>A server halts, but is working correctly until it halts</td>
</tr>
<tr>
<td>Omission failure</td>
<td>A server fails to respond to incoming requests</td>
</tr>
<tr>
<td>Receive omission</td>
<td>A server fails to receive incoming messages</td>
</tr>
<tr>
<td>Send omission</td>
<td>A server fails to send messages</td>
</tr>
<tr>
<td>Timing failure</td>
<td>A server’s response lies outside the specified time interval</td>
</tr>
<tr>
<td>Response failure</td>
<td>A server’s response is incorrect</td>
</tr>
<tr>
<td>Value failure</td>
<td>The value of the response is wrong</td>
</tr>
<tr>
<td>State transition failure</td>
<td>The server deviates from the correct flow of control</td>
</tr>
<tr>
<td>Arbitrary failure</td>
<td>A server may produce arbitrary responses at arbitrary times</td>
</tr>
</tbody>
</table>
Masking Failures by Redundancy

- **Strategy**: hide the occurrence of failure from other processes using *redundancy*.

1. *Information Redundancy* – add extra bits to allow for error detection/recovery (e.g., Hamming codes and the like).

2. *Time Redundancy* – perform operation and, if needs be, perform it again. Think about how transactions work (BEGIN/END/COMMIT/ABORT).

3. *Physical Redundancy* – add extra (duplicate) hardware and/or software to the system.

Can you think of Physical redundancy in Nature?
Masking Failures by Redundancy

(a)

(b)

Triple modular redundancy in a circuit (b) A, B, C are circuit elements and V* are voters
Today's Lecture Outline

• Real Systems (are often unreliable)
  • We ignored failures till now
  • Fault tolerance basic concepts
  • Goal: Recovering a system back to a consistent state

• Fault Tolerance – Recovery using Checkpointing

• Fault Tolerance – Logging and Recovery
Achieving Fault Tolerance in DS

- Process Resilience (when processes fail) T8.2
  - Have multiple processes (redundancy)
  - Group them (flat, hierarchically), voting
- Reliable RPCs (communication failures) T8.3
  - Several cases to consider (lost reply, client crash, …)
  - Several potential solutions for each case
- Distributed Commit Protocols T8.5
  - Perform operations by all group members, or not at all
  - 2 phase commit, … (last lecture)
- Today: A failure has occurred, can we recover? T8.6
Recovery Strategies

- When a failure occurs, we need to bring the system into an error free state (recovery). This is fundamental to Fault Tolerance.

1. **Backward Recovery**: return the system to some previous correct state (using checkpoints), then continue executing. Example?
   - Packet retransmit in case of lost packet

2. **Forward Recovery**: bring the system into a correct new state, from which it can then continue to execute. Example?
   - Erasure coding, \((n,k)\) where \(k < n \leq 2k\)
Forward and Backward Recovery

- **Major disadvantage of Backward Recovery:**
  - Checkpointing can be very expensive (especially when errors are very rare).
  - [Despite the cost, backward recovery is implemented more often. The “logging” of information can be thought of as a type of checkpointing.]

- **Major disadvantage of Forward Recovery:**
  - In order to work, all potential errors need to be accounted for *up-front*.
  - When an error occurs, the recovery mechanism then knows what to do to bring the system *forward* to a correct state.
Checkpointing

A recovery line to detect the correct distributed snapshot. This becomes challenging if checkpoints are un-coordinated.
Independent Checkpointing

The domino effect – Cascaded rollback
P2 crashes, roll back, but 2 checkpoints inconsistent (P2 shows m received, but P1 does not show m sent)
Coordinated Checkpointing

• Key idea: each process takes a checkpoint after a globally coordinated action. (why is this good?)

• Simple Solution: 2-phase blocking protocol
  • Co-ordinator multicast checkpoint_REQUEST message
  • Participants receive message, takes a checkpoint, stops sending (application) messages and queues them, and sends back checkpoint_ACK
  • Once all participants ACK, coordinator sends checkpoint_DONE to allow blocked processes to go on

• Optimization: consider only processes that depend on the recovery of the coordinator (those it sent a message since last checkpoint)
Recovery – Stable Storage

(a) Stable storage.
(b) Crash after drive 1 is updated.
(c) Bad spot.
Today's Lecture Outline

• Real Systems (are often unreliable)
  • We ignored failures till now
  • Fault Tolerance basic concepts

• Fault Tolerance – Checkpointing

• Fault Tolerance – Logging and Recovery
Goal: Make transactions Reliable

- ...in the presence of failures
  - Machines can crash. Disk Contents (OK), Memory (volatile), Machines don’t misbehave
  - Networks are flaky, packet loss, handle using timeouts

- If we store database state in memory, a crash will cause loss of “Durability”.

- May violate atomicity, i.e. recover such that uncommitted transactions COMMIT or ABORT.

- General idea: store enough information to disk to determine global state (in the form of a LOG)
Challenges:

- Disk performance is poor (vs memory)
  - Cannot save all transactions to disk
  - Memory typically several orders of magnitude faster

- Writing to disk to handle arbitrary crash is hard
  - Several reasons, but HDDs and SSDs have buffers

- Same general idea: store enough data on disk so as to recover to a valid state after a crash:
  - Shadow pages and Write-ahead Logging (WAL)
  - Idea is to provide Atomicity and Durability
• Shadow Pages
  • Provide Atomicity and Durability, “page” = unit of storage
  • Idea: When writing a page, make a “shadow” copy
    • No references from other pages, edit easily!
  • ABORT: discard shadow page
  • COMMIT: Make shadow page “real”. Update pointers to data on this page from other pages (recursive). Can be done atomically
  • Essentially “copy-on-write” to avoid in-place page update
Shadow Paging vs WAL

- **Write-Ahead-Logging**
  - Provide Atomicity and Durability
  - Idea: create a log recording every update to database
  - Updates considered reliable when stored on disk
  - Updated versions are kept in memory (page cache)
  - Logs typically store both REDO and UNDO operations
  - After a crash, recover by replaying log entries to reconstruct correct state

- WAL is more common, fewer disk operations, transactions considered committed once log written.
Write-Ahead Logging

- View as sequence of entries, sequential number
  - Log-Sequence Number (LSN)
  - Database: fixed size PAGES, storage at page level
- Pages on disk, some also in memory (page cache)
  - “Dirty pages”: page in memory differs from one on disk
- Reconstruction of global consistent state
  - Log files + disk contents + (page cache)
- Logs consist of sequence of records
  - Begin LSN, TID #Begin TXN
  - End LSN, TID, PrevLSN #Finish TXN (abort or commit)
  - Update LSN, TID, PrevLSN, pageID, offset, old value, new value
Write-Ahead Logging

- Logs consist of sequence of records
  - To record an update to state
  - Update LSN, TID, PrevLSN, pageID, offset, old value, new value
  - PrevLSN forms a backward chain of operations for each TID
  - Storing “old” and “new” values allow REDO operations to bring a page up to date, or UNDO an update reverting to an earlier version

- Transaction Table (TT): All TXNS not written to disk
  - Including Seq Num of the last log entry they caused

- Dirty Page Table (DPT): all dirty pages in memory
  - Modified pages, but not written back to disk.
  - Includes recoveryLSN: first log entry to make page dirty
Recovery using WAL – 3 passes

- **Analysis Pass**
  - Reconstruct TT and DPT (from start or last checkpoint)
  - Get copies of all pages at the start
- **Recovery Pass (redo pass)**
  - Replay log forward, make updates to all dirty pages
  - Bring everything to a state at the time of the crash
- **Undo Pass**
  - Replay log file backward, revert any changes made by transactions that had not committed (use PrevLSN)
  - For each write Compensation Log Record (CLR)
  - Once you reach BEGIN TXN, write an END TXN entry
### TT: Transaction Table

<table>
<thead>
<tr>
<th>TID</th>
<th>LastLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>567</td>
</tr>
<tr>
<td>2</td>
<td>42</td>
</tr>
<tr>
<td>7</td>
<td>67</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
</tr>
</tbody>
</table>

### DPT: Dirty Page Table

<table>
<thead>
<tr>
<th>pageID</th>
<th>recoveryLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>567</td>
</tr>
<tr>
<td>46</td>
<td>568</td>
</tr>
<tr>
<td>77</td>
<td>34</td>
</tr>
<tr>
<td>3</td>
<td>42</td>
</tr>
</tbody>
</table>

### TID: Transaction ID

LastLSN: LSN of the most recent log record seen for this Transaction. i.e. latest change

### pageID: key/ID of a page

recoveryLSN: LSN of first log record that made page dirty i.e. earliest change to page
ARIES: Log Records in Detail

- All Log Records
  - \([\text{PrevLSN}, \text{TID}, \text{type}]\)

- Update Log Records
  - \([\text{prevLSN}, \text{TID}, \text{“update”}, \text{pageID}, \text{redo}, \text{undo}]\)
  - Redo: info on how to redo the change by this rec
  - Undo: info on how to undo the change by this rec

- Compensation Log Records
  - \([\text{prevLSN}, \text{TID}, \text{“comp”}, \text{redoTheUndo}, \text{undoNextLSN}]\)
  - redoTheUnfo: info on how to redo the undo
  - undoNextLSN: next log record to undo for TXN

Example Ref from: https://www.slideshare.net/ssuser4b58fc/aries-55987958
Example Log File

LSN: [PrevLSN, TID, type]  # All
LSN: [prevLSN, TID, "update", pageID, redo, undo]  # Update

DB Buffer
Page 42
LSN=-
a=77
B=55

Page 46
LSN=-
c=22
Example Log File

LSN: [PrevLSN, TID, type]  # All
LSN: [prevLSN, TID, "update", pageID, redo, undo]  # Update

DB Buffer
Page 42
LSN=1
a=78
B=55

Page 46
LSN=-
c=22

LOG
1: [-,1,"update",42,a+=1, a-=1]
Example Log File

LSN: [PrevLSN, TID, type]  # All
LSN: [prevLSN, TID, "update", pageID, redo, undo]  # Update

DB Buffer
Page 42
LSN=2
a=78
B=58

Page 46
LSN=-
c=22

LOG
1: [-,1,"update",42,a+=1, a-=1
2: [-,2,"update", 42,b+=3, b-=3]

TT
<table>
<thead>
<tr>
<th>TID</th>
<th>LastLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

DPT
<table>
<thead>
<tr>
<th>pageID</th>
<th>recoveryLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>1</td>
</tr>
</tbody>
</table>
Example Log File

LSN: [PrevLSN, TID, type] # All
LSN: [prevLSN, TID, “update”, pageID, redo, undo] # Update

---

<table>
<thead>
<tr>
<th>TT</th>
<th>TID</th>
<th>LastLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>DPT</th>
<th>pageID</th>
<th>recoveryLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>42</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>46</td>
<td>3</td>
</tr>
</tbody>
</table>

---

DB Buffer

Page 42

- LSN=4
- a=78
- B=59

Page 46

- LSN=3
- c=24

---

LOG

1: [-1,"update",42,a+=1, a-=1
2: [-2,"update", 42,b+=3, b-=3]
3: [22,"update",46,c+=2, c-=2]
4: [11,"update",42, b+=1, b-=1]
Example Log File

**DB Buffer**
- Page 42
  - LSN=4
  - a=78
  - B=59
- Page 46
  - LSN=3
  - c=24

**LOG**
1: [-1,"update",42,a+=1, a-=1
2: [-2,"update", 42,b+=3, b-=3]
3: [2,2,"update",46,c+=2, c-=2]
4:[1,1,"update",42, b+=1, b-=1]
5:[3,2,"commit"]

**LSN**
- [PrevLSN, TID, type] # All
- [prevLSN, TID, "update", pageID, redo, undo] # Update

**TT**
<table>
<thead>
<tr>
<th>TID</th>
<th>LastLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

**DPT**
<table>
<thead>
<tr>
<th>pageID</th>
<th>recoveryLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>1</td>
</tr>
<tr>
<td>46</td>
<td>3</td>
</tr>
</tbody>
</table>
Example Log File

LSN: \([\text{PrevLSN, TID, type}]\)
LSN: \([\text{prevLSN, TID, “update”, pageID, redo, undo}]\)
LSN: \([\text{prevLSN, TID, “comp”, redoTheUndo, undoNextLSN}]\) # All  # Update  # Compensation

DB Buffer

Page 42

<table>
<thead>
<tr>
<th>LSN=4</th>
</tr>
</thead>
<tbody>
<tr>
<td>a=78</td>
</tr>
<tr>
<td>B=59</td>
</tr>
</tbody>
</table>

Page 46

<table>
<thead>
<tr>
<th>LSN=3</th>
</tr>
</thead>
<tbody>
<tr>
<td>c=24</td>
</tr>
</tbody>
</table>

LOG

1: \([-1,”update”,42,a+=1, a-=1]\)
2: \([-2,”update”, 42,b+=3, b-=3]\)
3: \([2,2,”update”,46,c+=2, c-=2]\)
4: \([1,1,”update”,42, b+=1, b-=1]\)
5: \([3,2,”commit”]\)

TT

<table>
<thead>
<tr>
<th>TID</th>
<th>LastLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

DPT

<table>
<thead>
<tr>
<th>pageID</th>
<th>recoveryLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>1</td>
</tr>
<tr>
<td>46</td>
<td>3</td>
</tr>
</tbody>
</table>
Example Log File

<table>
<thead>
<tr>
<th>TT</th>
<th>TID</th>
<th>LastLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

Lake

**LOG**

1: [-1,"update",42,a+=1, a=1]
2: [-2,"update", 42,b+=3, b=3]
3: [2,2,"update",46,c+=2, c=2]
4:[1,1,"update",42, b+=1, b=1]
5:[3,2,"commit"]

**DB Buffer**

Page 42

- LSN=4
- a=78
- B=59

Page 46

- LSN=3
- c=24

LSN: [PrevLSN, TID, type]  # All
LSN: [prevLSN, TID, "update", pageID, redo, undo]  # Update
LSN: [prevLSN, TID, "comp", redoTheUndo, undoNextLSN]  #compensation
Example Log File

1. Analysis

1: [-,1,"update",42,a+=1, a-=1
2: [-,2,"update", 42,b+=3, b-=3]  
3: [2,2,"update",46,c+=2, c-=2] 
4:[1,1,"update",42, b+=1, b-=1]  
5:[3,2,"commit"]

1. Analysis to figure our the start of the redo.  
=> start from 1
Example Log File

LSN: [PrevLSN, TID, type]  # All
LSN: [prevLSN, TID, "update", pageID, redo, undo]  # Update
LSN: [prevLSN, TID, "comp", redoTheUndo, undoNextLSN]  # compensation

1. Analysis
2. Redo

1: [-1,"update",42,a+=1, a-=1
2: [-2,"update", 42,b+=3, b-=3]
3: [2,2,"update",46,c+=2, c-=2]
4:[1,1,"update",42, b+=1, b-=1]
5:[3,2,"commit"]
## Example Log File

### LOG

<table>
<thead>
<tr>
<th>LSN: [PrevLSN, TID, type]</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSN: [prevLSN, TID, &quot;update&quot;, pageID, redo, undo]</td>
</tr>
<tr>
<td>LSN: [prevLSN, TID, &quot;comp&quot;, redoTheUndo, undoNextLSN]</td>
</tr>
</tbody>
</table>

1. Analysis
2. Redo
3. Undo

### DB Buffer

- **Page 42**
  - LSN=6
  - a=78
  - B=58

- **Page 46**
  - LSN=3
  - c=24

The log entries are as follows:

1. \([-1, "update", 42, a+=1, a=1\]
2. \([-2, "update", 42, b+=3, b=3]\]
3. \([2, 2, "update", 46, c+=2, c=2]\]
4. \([1, 1, "update", 42, b+=1, b=1]\]
5. \([3, 2, "commit"]\]
6. \([4, 1, "comp", 42, b-=1, 1]\]

### TT Table

<table>
<thead>
<tr>
<th>TID</th>
<th>LastLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>

### DPT Table

<table>
<thead>
<tr>
<th>pageID</th>
<th>recoveryLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>1</td>
</tr>
<tr>
<td>46</td>
<td>3</td>
</tr>
</tbody>
</table>
Example Log File

LSN: [PrevLSN, TID, type]  # All
LSN: [prevLSN, TID, “update”, pageID, redo, undo]  # Update
LSN: [prevLSN, TID, “comp”, redoTheUndo, undoNextLSN]  # Compensation

1. Analysis
2. Redo
3. Undo

1: [-1,”update”,42,a+=1, a-=1
2: [-2,”update”, 42,b+=3, b-=3]
3: [2,2,”update”,46,c+=2, c-=2]
4:[1,1,”update”,42, b+=1, b-=1]
5:[3,2,”commit”]

6: [4,1,”comp”,42,b-=1, 1]
7: [6,1,”comp”, 42, a-=1, 1]
WAL can be integrated with 2PC

• WAL can integrate with 2PC
  • Have additional log entries that capture 2PC operation
  • **Coordinator**: Include list of participants
  • **Participant**: Indicates coordinator
  • Votes to commit or abort
  • Indication from coordinator to Commit/Abort
Optimizing WAL

- As described earlier:
  - Replay operations back to the beginning of time
  - Log file would be kept forever, (entire Database)
- In practice, we can do better with CHECKPOINT
  - Periodically save DPT, TT
  - Store any dirty pages to disk, indicate in LOG file
  - Prune initial portion of log file: All transactions upto checkpoint have been committed or aborted.
Summary

• Real Systems (are often unreliable)
  • Introduced basic concepts for Fault Tolerant Systems including redundancy, process resilience, RPC

• Fault Tolerance – Backward recovery using checkpointing, both Independent and coordinated

• Fault Tolerance – Recovery using Write-Ahead-Loggng, balances the overhead of checkpointing and ability to recover to a consistent state
Characterizing Message-Logging Schemes

Incorrect replay of messages after recovery, leading to an orphan process.
Transactions: ACID Properties

- **Atomicity**: Each transaction completes in its entirety, or is aborted. If aborted, should not have effect on the shared global state.
  - Example: Update account balance on multiple servers

- **Consistency**: Each transaction preserves a set of invariants about global state. (exact nature is system dependent).
  - Example: in a bank system, law of conservation of $$
Transactions: ACID Properties

- **Isolation**: Also means serializability. Each transaction executes as if it were the only one with the ability to RD/WR shared global state.

- **Durability**: Once a transaction has been completed, or “committed” there is no going back. In other words there is no “undo”.

- Transactions can also be nested

- “Atomic Operations” => Atomicity + Isolation