Aidan Rohm
Alex Brooke
CISC-5597

Lab 3: 2 Phase-Commit Protocol

Creating 2 PC Protocol
To create the Two-Phase Commit protocol, we implemented coordinator functions that send prepare, commit, and abort RPCs to all participant nodes. Each participant exposes corresponding prepare, commit, and abort handlers that validate the transaction, vote YES/NO, and apply or discard updates. The coordinator first collects all votes in Phase 1, and only issues a global commit if every participant replies YES, otherwise it sends a global abort. Timeouts and crash flags simulate failures, allowing the coordinator to safely abort when a participant becomes unresponsive. These components ensure the transaction is applied atomically across nodes (or not at all) maintaining correctness under normal execution and fault conditions.

Reliable RPC Coordination Across Nodes
Each node must maintain consistent IPs and stay reachable through the internal network. Static host/port constants in each script and a shared configuration section ensure consistency. Each participant exposes prepare and commit routes. The coordinator retries and aborts on any unreachable participant.

Normal vs Abort logic
There is a need to distinguish between insufficient-funds aborts and system failures. A balance check in participantA.py returns NO vote if A < 100**.** Coordinator collects YES/NO votes commits only if all YES, otherwise issues global ABORT.

Handling timeouts and crashes
The coordinator must behave safely even when a participant becomes unresponsive before or after voting. To simulate these failure cases, the participants include CRASH_BEFORE_VOTE and CRASH_AFTER_VOTE flags in participantB.py, which intentionally stop the node from replying during the prepare or commit phase. On the coordinator side, all RPC calls are wrapped in requests timeouts and try/except blocks so that any lack of response is treated as a failure. If a participant does not reply in time, the coordinator logs a message and immediately triggers a global abort. This ensures that no transaction can be partially committed, preserving atomicity under failures.

Recovery logging
There was a need to have verifiable traces of prepare, vote, and commit. Each node writes to log_node file with timestamps. Logs verify outcomes and support manual inspection (“e.g. cat log_node1_A.txt”)