CORNUS: ATOMIC COMMIT FOR A CLOUD DBMS WITH STORAGE DISAGGREGATION

- Sumedha Joshirao
WHY CLOUD DATABASES?

- Elasticity
- High availability
- Cost Competitiveness
TWO PHASE COMMIT PROTOCOL

• Coordinator log delay: Coordinator must durably log the decision before reply

(a) 2PC with no failure.
TWO PHASE COMMIT PROTOCOL

(b) 2PC with coordinator failure (cooperative termination protocol).
ISSUES WITH 2PC PROTOCOL

Latency Issue:
- Average latency of one network round-trip and two logging operations

Blocking
- Decision postponed till coordinator is restored
ISSUES WITH EXISTING SOLUTIONS THAT TRY TO SOLVE THESE PROBLEMS

- Targeted shared-nothing architecture
- Need to customize storage
- Make strong assumptions about the workload
  - May not be practical for disintegrated storage
  - Mitigate blocking by adding an extra phase -> increase latency
SOLUTION: CORNUS

- Non-blocking
- Low-latency 2PC variant
- Only new storage layer function needed is LogOnce()
  - Implemented using compare and swap
ILLUSTRATION OF CORNUS
CORNUS APIs

• Remote Procedural Calls (RPC) – communication between participants
  A. Log(txn,type): appends log record of certain type to the end of txn’s log
  B. LogOnce(txn,type): guarantees that transaction’s state is written at the most once
ALGORITHM

11 Function Participant::StartCommit(txn)
12   wait for VOTE-REQ from coordinator
13   on timeout $\text{RPC}_{\text{sync}}^{\text{local log}} = \text{Log}(\text{ABORT})$ return
14   if participant votes yes for txn then
15     resp = $\text{RPC}_{\text{sync}}^{\text{local log}} = \text{LogOnce}(\text{VOTE-YES})$
16     if resp is ABORT then
17       # Another participant has logged ABORT for it
18       reply ABORT to coordinator
19     else
20       reply VOTE-YES to coordinator
21       wait for decision from coordinator
22       on timeout decision = TerminationProtocol(txn)
23       $\text{RPC}_{\text{sync}}^{\text{local log}} = \text{Log}(\text{decision})$
24   else
25     $\text{RPC}_{\text{async}}^{\text{local log}} = \text{Log}(\text{ABORT})$
26     reply ABORT to coordinator

1 Function Coordinator::StartCommit(txn)
2   for p in txn.participants do
3     send VOTE-REQ to p asynchronously
4   wait for all responses from participants
5     on receiving ABORT decision ← ABORT
6     on receiving all responses decision ← COMMIT
7     on timeout decision ← TerminationProtocol(txn)
8     reply decision to the txn caller
9   for p in txn.participants do
10     send decision to p asynchronously
11
26 Function TerminationProtocol(txn)
27   for every participant p other than self do
28     $\text{RPC}_{\text{async}}^{p, \text{log}} = \text{LogOnce}(\text{ABORT})$
29   wait for responses
30     on receiving ABORT decision ← ABORT
31     on receiving COMMIT decision ← COMMIT
32     on receiving all responses decision ← COMMIT
33     on timeout retry from the beginning
34   return decision
FAILURE AND RECOVERY

• Coordinator Failure:
  • Case 1: FAILURE BEFORE PROTOCOL STARTS
  • Case 2: FAILURE AFTER SENDING SOME BUT NOT ALL VOTE REQUESTS
  • Case 3: FAILURE AFTER SENDING ALL VOTE REQUESTS BUT BEFORE SENDING DECISION
  • Case 4: FAILURE AFTER SENDING DECISION TO SOME BUT NOT ALL
  • Case 5: FAILURE AFTER SENDING DECISION TO ALL PARTICIPANTS
FAILURE AND RECOVERY

• Participant Failure:
  • Case 1: FAILS BEFORE RECEIVING VOTE REQUEST
  • Case 2: FAILURE BEFORE LOGGING VOTE BUT AFTER RECEIVING VOTE REQUEST
  • Case 3: FAILURE AFTER LOGGING THE VOTE, BEFORE REPLYING TO COORDINATOR
  • Case 4: FAILURE AFTER SENDING VOTE
Figure 4: Cornus under Failures — The behavior of Cornus under two failures scenarios.
EXPERIMENTAL ANALYSIS

**SETUP**

- **Cloud Storage Services**: Microsoft Azure Blob Storage, Microsoft Azure Cache for Redis
- **Workloads**:
  - Yahoo! Cloud Serving Benchmark
  - 10 GB data partitions -> 1 KB Tuples
  - Each transaction -> 16 tuples with 50% reads and 50% writes
- **Parameter Setup**
  - Maximum 8 compute nodes
  - Eight worker threads per node execute transaction logic
  - Eight worker threads per node serve remote requests
SCALABILITY

- As nodes increase latency of both 2PC and Cornus increases linearly
- Speedup of Cornus over 2PC on average latency slightly decreases as the number of nodes increase
- Current version of Azure Blob cannot benefit from Cornus for applications that want separate access control between data and transaction states
PERCENTAGE OF R-ONLY TRANSACTIONS

- Improvements of Cornus increases with decrease in percentage of R-only transactions -> 1.7 times improvement
- Improves latency for RW transactions
- Spends more time in prepare phase
CONTENTION

• Provides less improvement under high contention as abort time dominates the total transaction elapsed time.
TIME TO TERMINATE TRANSACTIONS ON FAILURE

- Always terminates transaction in 4ms up to 8ms on Redis and up to 20 ms on Azure Blob
- Tail latency of Azure Blob increases more than Redis as number of nodes increases
CONCLUSION

• Cornus solves the long latency and blocking problem in 2PC
• Evaluations show a speedup of 1.9x in latency
Questions?
THANK YOU!!!