Formal Verification of COCO Database Framework Using CSP

Peimu Li, Jiaqi Yin, Huibiao Zhu

Shanghai Key Laboratory of Trustworthy Computing,

East China Normal University, Shanghai, China

Email: hbzhu@sei.ecnu.edu.cn

June 15, 2022

Outline

- Introduction
- Overview of COCO Database
- Modeling COCO Database
- Verification
- Conclusion

Outline

Introduction

Background

- Many distributed OLTP databases use a shared-nothing architecture for scale out and data partitioning to achieve the scalability of data storage.
- Lu et al. proposed epoch-based commit and replication, which is an improved protocol based on 2PC, and implemented it in distributed database COCO.
- The COCO database also supports two variants of optimistic concurrency control: physical time and logical time OCC.

Motivation

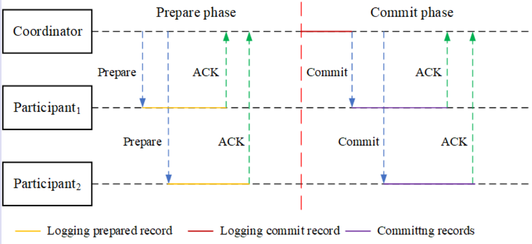
- The design of a distributed database architecture often needs to satisfy many functional properties.
- For the reason that the test workload and benchmarks are artificially set, and the test results are directly affected by the hardware performance, the test results still can be improved.

Outline

- Introduction
- Overview of COCO Database
- Modeling COCO Database
- Verification
- Conclusion

Overview of COCO Database

Overview of Epoch-based Commit Protocol and Replication



- The protocol commits transactions within the epoch synchronously at the end of the epoch.
- Epoch-based commit contains a *prepare* phase and a *commit* phase.
- COCO performs the replication on backup databases asynchronously.

Overview of COCO Database

Pseudo Code of Physical Time OCC

Algorithm 1 PT-OCC

Phase 1: Locking the write set

```
for record in T.WS do
    ok,tid=call_n(lock,record.key)
    if record not in T.RS then
        record.tid=tid
    end if
    if ok==false or tid!=record.tid then
        abort=true
    end if
end for
```

Phase 2: Validating the read set

```
for record in T.RS
T.WS do
    locked,tid=call_n(read_metadata,record.key)
    if locked or tid!=record.tid then
        abort=true
    end if
end for
```

Phase 3: Writing back to the database

```
for record in T.WS do
    call_n(db_write,record.key,record.value,record.tid)
    call_n(unlock, record.key)
    for i in get_replica_nodes(record.key) do
        call_i(db_replicate,record.key,record.value,T.tid)
    end for
end for
```

Overview of COCO Database

Pseudo Code of Logical Time OCC

Algorithm 2 LT-OCC

```
Phase 1: Locking the write set
for record in T.WS do
    ok,{wts,rts}=call_n(lock, record.key)
    if record not in T.RS then
        record.wts=wts
    end if
    if ok==false or wts!=record.wts then
        abort()
    end if
    record.rts=rts
end for
Phase 2: Validating the read set
for record in T.RS
T.WS do
    if record.rts<T.tid then
        locked,{wts,rts}=call_n(read_matadata,record.key)
    end if
    if wts!=record.wts or (rts<T.tid and locked) then
        abort()
    end if
    call_n(write_metadata,record.key,locked,{wts,T.tid})
end for
Phase 3: Writing back to the database
for record in T.WS do
    wts,rts=T.tid,T.tid
    call_n(db_write,record.key,record.value,{wts,rts})
    call_n(unlock, record.key)
    for i in get_replica_nodes(record.key) do
        call_i(db_replicate,record.key,record.value,{wts,rts})
    end for
end for
```

Outline

- Introduction
- Overview of COCO Database
- Modeling COCO Database
- Verification
- Conclusion

Modeling COCO Database

- COCO architecture includes epoch-based commit protocol, replication protocol and two variants of optimistic concurrency control.

Overview the Model

$$Epoch\_commit() =_{df} Coordinator() || (|||i : \{1...N\} @ Participant(i))$$

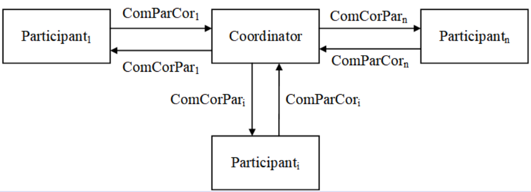
$$Replication(records) =_{df}$$
$$Primary\_replication(records) || (|||i : \{1...N\} @ Replica(i)))$$

$$PT\_OCC() =_{df}$$
$$(|||i : \{1...N\} @ Transaction\_PT(i, read\_set_i, write\_set_i))$$
$$|| Primary() || (|||i : \{1...N\} @ Replica(i))$$

$$LT\_OCC() =_{df}$$
$$(|||i : \{1...N\} @ Transaction\_LT(i, read\_set_i, write\_set_i))$$
$$|| Primary() || (|||i : \{1...N\} @ Replica(i))$$

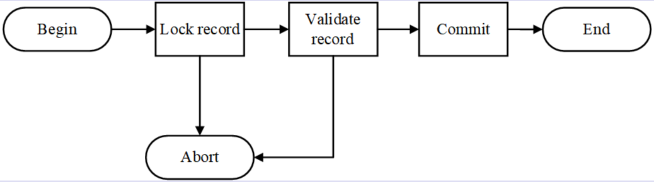
Overview of COCO Database

Overview of Epoch-based Commit Protocol and Replication



- The protocol commits transactions within the epoch synchronously at the end of the epoch.
- Epoch-based commit contains a *prepare* phase and a *commit* phase.
- COCO performs the replication on backup databases asynchronously.

Overview of PT-OCC and LT-OCC



- These algorithms can be divided into three phases: locking, validating and commit.
- In locking phase, transactions lock all data records they need to operate. If transaction can't lock all records it needs, it simply aborts.
- In validating phase, transactions validate records they locked with their read sets. If versions of these records are inconsistent, transaction simply aborts.
- In commit phase, transactions commit their write sets.

Outline

- Motivation
- Overview of COCO Database
- Modeling COCO Database
- Verification
- Conclusion

Verification

- In COCO, we should avoid the situation that two or more clients are waiting the resources which have been occupied by other clients infinitely.
- In the tool PAT, there is a primitive to describe this situation.

Property 1: Deadlock Freedom

$\#$*assert System*() *deadlockfree*;

Verification

- This property asserts during the execution of a transaction, data can only be converted from one consistency state to another consistency state.

Property 2: Consistency

$\#define\ Consistency\ (\wedge i : \{1...N\} record_i == last\_rec$
$ord) \vee (\wedge i : \{1...N\} record_i == cur\_record)$
$\#assert\ Epoch\_commitI() \mid = \ Consistency$

Verification

- It means every request in a distributed system can be responded to.

Property 3: Availability

$\#define\ Availability\ (hasNo == True \land finished == True)$
$\#assert\ Epoch\_commit() \mid= \ Availability$

- It means that when a node or network partition in a distributed system fails, the entire system can still provide external services that satisfy consistency and availability.

Property 4: Partition Tolerance

$\#define\ PartitionTolerance\ finished == True$
$\#assert\ Epoch\_commit() \mid= PartitionTolerance$

Verification

- This property means that when some requests failure or unpredictable failures occur in the system, the system can still guarantee the normal execution of most transactions.

Property 5: Basically Availability

$\#define\ BasicallyAvailability\ (existCrash == True) \wedge$
$(available == True)$
$\#assert\ PT\_OCC()\ |=\ BasicallyAvailability$
$\#assert\ LT\_OCC()\ |=\ BasicallyAvailability$

Verification

- It refers to the fact that all data copies in the system can finally reach a consistent state after a period of synchronization without the guarantee of strong consistency of system data.

Property 6: Eventually Consistency

$\#define\ EventuallyConsistency\ EG((\wedge i:\{1...N\}recor$
$d_i == last\_record) \vee (\wedge i:\{1...N\}record_i == cur\_record))$
$\#assert\ PT\_OCC() \mid= \ BasicallyAvailability$
$\#assert\ LT\_OCC() \mid= \ BasicallyAvailability$

Verification

- This property refers to allowing the data in the system to have an intermediate state, and this state does not affect the overall availability of the system.

Property 7: Soft State

$\#define\ SoftState\ (\vee i : \{1...N\} record_i! = last\_rec$
$ord) \wedge (\vee i : \{1...N\} record_i! = cur\_record)$
$\#assert\ PT\_OCC() \mid = \ SoftState$
$\#assert\ LT\_OCC() \mid = \ SoftState$

Verification

Evaluation Result

We use the model checker PAT to verify the main frameworks of COCO distributed database such as epoch-based commit and replication, PT-OCC and LT-OCC. The verification results are shown below.

✓	1	CommitProtocol() deadlockfree
✓	2	CommitProtocol() |= Consistency
✓	3	CommitProtocol() reaches Availability
✗	4	CommitProtocol() reaches PartitionTolerance
✓	5	PT_OCC() deadlockfree
✓	6	PT_OCC_BA() reaches BasicallyAvailability
✓	7	PT_OCC_EC() reaches SoftState
✓	8	PT_OCC_EC() |= F G EventuallyConsistency
✓	9	LT_OCC() deadlockfree
✓	10	LT_OCC_BA() reaches BasicallyAvailability
✓	11	LT_OCC_EC() reaches SoftState
✓	12	LT_OCC_EC() |= F G EventuallyConsistency

Outline

Conclusion

- Conclusion
  - The CAP and BASE theories put forward the properties that the distributed system architecture needs to satisfy, and we verified the properties of COCO in an epoch cycle.
  - It has been verified that (1) epoch-based commit and replication satisfy consistency and availability but not partition tolerance, and (2) PT-OCC and LT-OCC satisfy basic availability, soft state, and eventually consistency.
  - This shows that COCO can guarantee high availability during an epoch cycle, and can also guarantee consistency at the end of the epoch.
- Future Work
  - In the future, we will verify the isolation of COCO and sequential consistency of concurrency control.

Thank you