Consistency in Transactional Distributed Databases: Protocols and Testing

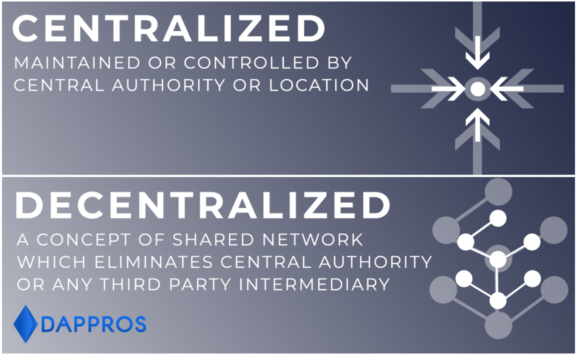
Hengfeng Wei (魏恒峰)

hfwei@nju.edu.cn

Nov. 04, 2022

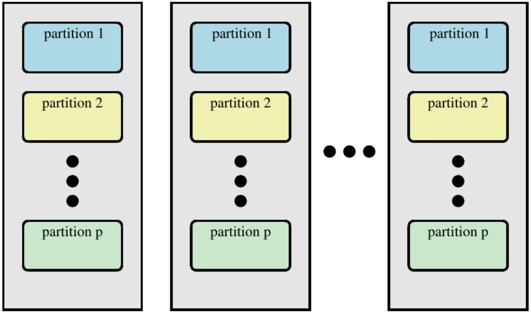
Background

Centralized Databases *vs.* Distributed Databases



Data Consistency Problem

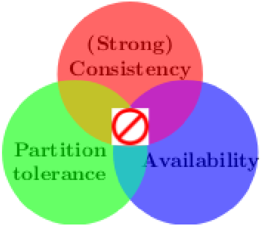
"Data Partition + Data Replication"



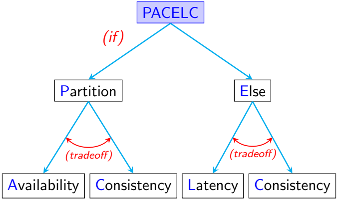
Data Consistency Problem

Data Consistency Problem

(Strong) Consistency, Availability, Latency、Patition tolerance



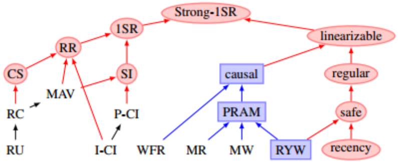
CAP Theorem

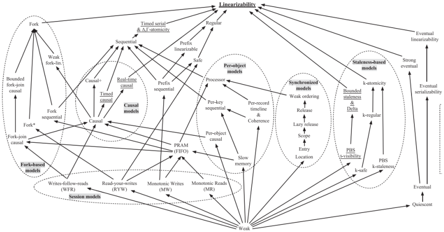


PACELC Tradeoff

Consistency Models

Use consistency models to capture these tradeoffs





My Researches

Researches on data consistency around consistency models:

Computability: What is possible or impossible?

Protocol: How to design fast, scalable, and fault-tolerant protocols?

Testing: What is the complexity?

How to design efficient testing algorihthms?

My Researches

Researches on data consistency around consistency models:

Computability: What is possible or impossible?

Protocol: How to design fast, scalable, and fault-tolerant protocols?

Testing: What is the complexity?

How to design efficient testing algorihthms?

Classic problems with the ever-changing requirements

My Researches

Research I ($\geq$ 2012): Read/Write Register[a]

Distributed NoSQL Key-Value Stores (TODO: 重新画图)

TODO: +research outcomes

[a] 读写寄存器，也就是读写变量，虽然最初与计算机系统中的“寄存器”概念相关，但已慢慢解耦。

My Researches

Research II ($\geq$ 2017): Replicated Data Types[b]

(a) Google Docs

(b) Apache Wave

(c) Wikipedia

(d) LaTeX Editor

TODO: +research outcomes

[b] 复制数据类型，是经典数据类型的分布式版本，如列表，集合，队列等。

My Researches

Research III ($\geq$ 2020): Distributed Transactions[c]

TODO: +research outcomes TODO: +logos

[c] 分布式事务。每个事务由一组操作构成，这些操作要么全成功，要么全不成功。

Overview of the Work on UniStore

UniStore: A fault-tolerant marriage of causal and strong consistency

Manuel Bravo Alexey Gotsman Borja de Régil
IMDEA Software Institute

Hengfeng Wei *
Nanjing University

ATC'2021 (CCF A)

UniStore is the first fault-tolerant and scalable transactional data store that combines causal and strong consistency.

Overview of the Work on UniStore

Partial Order-Restrictions Consistency (PoR consistency)

$$\text{CC} < \text{PoR} < \text{SER}$$

CC: CausalConsistency; SER: Serializability

Key Challenges (I): Ensure liveness in presence of faults

Key Challenges (II): Provide rigorous correctness proof

Overview of the Work on UniStore

UniStore: A fault-tolerant marriage of causal and strong consistency

Manuel Bravo Alexey Gotsman Borja de Régil
IMDEA Software Institute

Hengfeng Wei *
Nanjing University

ATC'2021 (CCF A)

I am fully responsible for the rigorous correctness proof:

- Finished a proof of 20 pages contained in the arXiv version
- Identified several nontrivial bugs in the early versions of the protocol[d]

[d]One of these bugs also exists in the well-known Granola protocol proposed by James Cowling and Barbara Liskov, something that had gone unnoticed for 10 years.

What is UniStore?

UniStore is a fast, scalable, and fault-tolerant

transactional distributed key-value store

that supports a combination of weak and strong consistency.

What is UniStore?

UniStore is a fast, scalable, and fault-tolerant

transactional distributed key-value store

that supports a combination of weak and strong consistency.

Weak consistency: CausalConsistency

Strong consistency: Serializability

CausalConsistency and Serializability

Why Uni-?

Weak consistency: low latency, high availability

Strong consistency: easy to preserve critical application invariants

Why UNI-?

DEPOSIT WITHDRAW QUERY INTEREST

Invariant: $\mathsf{balance} \geq 0$

Why UNI-?

DEPOSIT WITHDRAW QUERY INTEREST

Invariant: $\mathsf{balance} \geq 0$

Causal consistency allows two concurrent WITHDRAW to execute without knowing each other.

Why UNI-?

DEPOSIT WITHDRAW QUERY INTEREST

Invariant: $\mathsf{balance} \geq 0$

Causal consistency allows two concurrent WITHDRAW to execute without knowing each other.

Only WITHDRAW needs to use strong consistency.

Consistency Model of UniStore

UniStore implements a transactional variant of
Partial Order-Restrictions (PoR) consistency [Li@ACT'2018]

(I) transactional causal consistency by default

(II) to specify conflicting transactions under strong consistency

Consistency Model of UniStore

Definition (Session Order)

A transaction t_1 precedes a transaction t_2 in the session order, denoted $t_1 \xrightarrow{so} t_2$, if they are executed by the same client and t_1 is executed before t_2.

Definition (Conflict Relation)

The conflict relation, denoted $\bowtie$, between transactions is a symmetric relation.

$$t_1 \bowtie t_2 \iff t_2 \bowtie t_1.$$

Consistency Model of UniStore

Definition (PoR)

A set of transactions $T \triangleq T_{causal} \uplus T_{strong}$ committed by UniStore satisfies PoR if there exists a causal order $\prec$ on T such that

Consistency Model of UniStore

Definition (PoR)

A set of transactions $T \triangleq T_{causal} \uplus T_{strong}$ committed by UniStore satisfies PoR if there exists a causal order $\prec$ on T such that

Causality: '$\prec$' is a partial order and $so \subseteq \prec$.

Consistency Model of UniStore

Definition (PoR)

A set of transactions $T \triangleq T_{causal} \uplus T_{strong}$ committed by UniStore satisfies PoR if there exists a causal order $\prec$ on T such that

Causality: '$\prec$' is a partial order and $so \subseteq \prec$.

ConflictOrdering: $\forall t_1, t_2 \in T_{strong}.\ t_1 \bowtie t_2 \implies t_1 \prec t_2 \lor t_2 \prec t_1$.

Consistency Model of UniStore

Definition (PoR)

A set of transactions $T \triangleq T_{causal} \uplus T_{strong}$ committed by UniStore satisfies PoR if there exists a causal order $\prec$ on T such that

- Causality: '$\prec$' is a partial order and $so \subseteq \prec$.
- ConflictOrdering: $\forall t_1, t_2 \in T_{strong}.\ t_1 \bowtie t_2 \implies t_1 \prec t_2 \lor t_2 \prec t_1$.
- EventualVisibility: A transaction $t \in T$ that is either strong or originates at a correct data center eventually become visible at all correct data centers: from some point on, t precedes in $\prec$ all transactions issued at correct data centers.

Consistency Model of UniStore

Definition (PoR)

A set of transactions $T \triangleq T_{causal} \uplus T_{strong}$ committed by UniStore satisfies PoR if there exists a causal order $\prec$ on T such that

Causality: '$\prec$' is a partial order and $so \subseteq \prec$.

ConflictOrdering: $\forall t_1, t_2 \in T_{strong}.\ t_1 \bowtie t_2 \implies t_1 \prec t_2 \lor t_2 \prec t_1$.

EventualVisibility: A transaction $t \in T$ that is either strong or originates at a correct data center eventually become visible at all correct data centers: from some point on, t precedes in $\prec$ all transactions issued at correct data centers.

RetVal: IntRetVal $\land$ ExtRetVal

Consistency Model of UniStore

Consider a read r from key k in a transaction t.

$\textsc{IntRetVal}$: read from the latest update on k preceding r in t

$$\textsc{RetVal} = \textsc{IntRetVal} \land \textsc{ExtRetVal}$$

$\textsc{ExtRetVal}$: read from the last update on k
of the latest transaction (in an order consistent with $\prec$) preceding t

Consistency Model of UniStore

DEPOSIT WITHDRAW QUERY INTEREST

Invariant: $\textsf{balance} \geq 0$

Declaring that strong transactions
including WITHDRAW on the same account conflict.

Design Challenges of UNISTORE

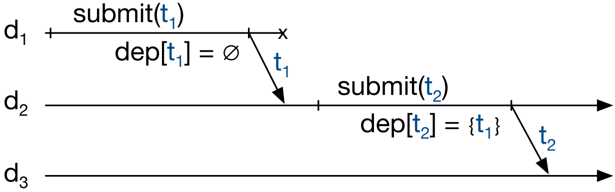
How to satisfy liveness (EVENTUALVISIBILITY) despite failures?

A transaction $t \in T$ that is either strong or originates at a correct data center eventually become visible at all correct data centers.

Design Challenge of UniStore (I)

Data center d_1 crashes

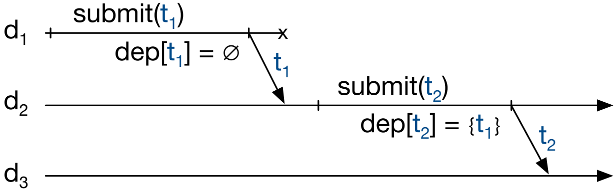
before t_1 is replicated to correct data center d_3.



Design Challenge of UniStore (I)

Data center d_1 crashes

before t_1 is replicated to correct data center d_3.



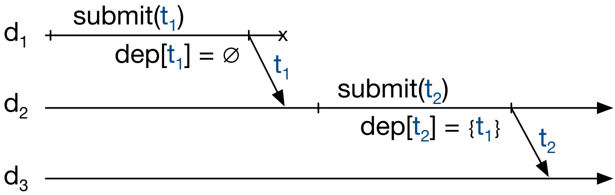
Transaction t_2 (at correct data center d_2)

may never become visible at correct data center d_3.

Fault-tolerance of UniStore (I)

Data center d_1 crashes

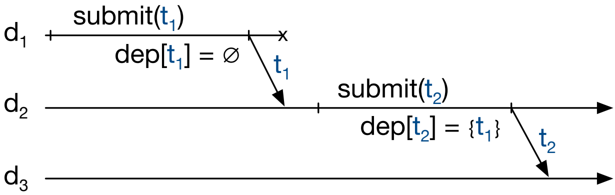
before t_1 is replicated to correct data center d_3.



Fault-tolerance of UniStore (I)

Data center d_1 crashes

before t_1 is replicated to correct data center d_3.



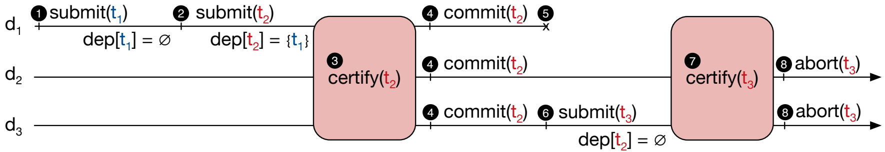
Data center d_2 need to forward causal transactions

to other data centers.

Design Challenge of UniStore (II)

Data center d_1 crashes

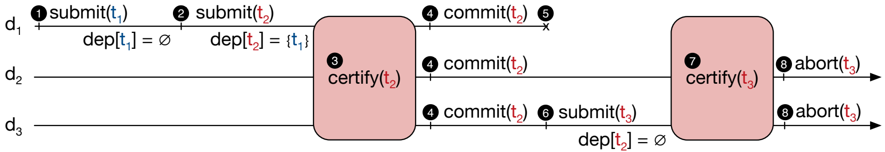
before t_1 is replicated to correct data center d_3.



Design Challenge of UniStore (II)

Data center d_1 crashes

before t_1 is replicated to correct data center d_3.

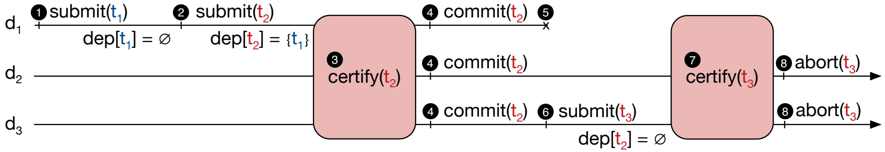


Transaction t_2 will never be visible at d_3.

No transaction t_3 conflicting with t_2 can commit (by ConflictOrdering).

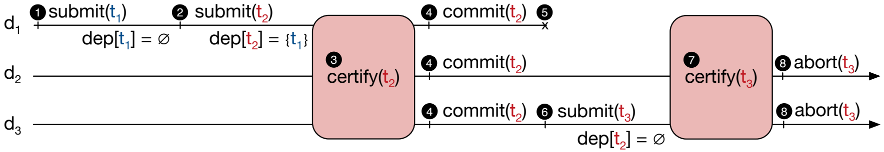
Fault-tolerance of UniStore (II)

UniStore ensures that before a strong transaction commits,
all its causal dependencies are uniform,
i.e., will eventually become visible at all correct data centers.



Fault-tolerance of UniStore (II)

UniStore ensures that before a strong transaction commits,
all its causal dependencies are uniform,
i.e., will eventually become visible at all correct data centers.



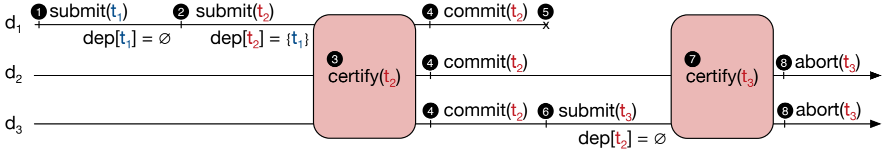
Transaction t_1 will eventually be visible at d_3.

Transaction t_2 will eventually be visible at d_3.

Transaction t_3 may be committed at d_3.

Performance of UniStore

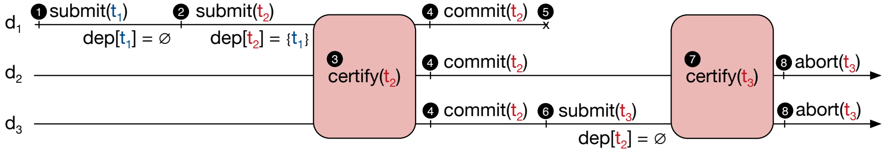
Causal transactions remain highly-available, i.e., committed locally.



A strong transaction may have to wait for some of its dependencies to become uniform before committing.

Performance of UniStore

Causal transactions remain highly-available, i.e., committed locally.



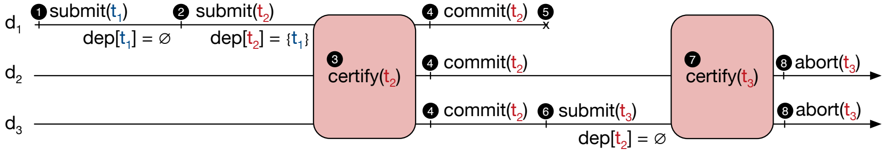
A strong transaction may have to wait for some of its dependencies to become uniform before committing.

However, this may cost too much.

Performance of UniStore

UniStore makes a remote causal transaction visible to clients only after it is uniform.

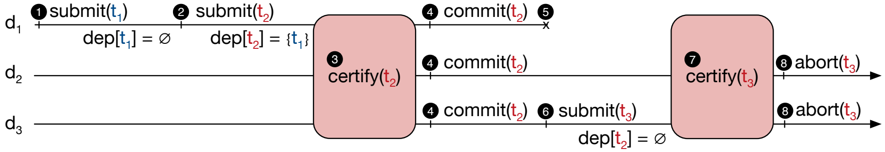
Causal transactions are executed on an (almost) uniform snapshot that may be slightly in the past.



Performance of UniStore

UniStore makes a remote causal transaction visible to clients only after it is uniform.

Causal transactions are executed on an (almost) uniform snapshot that may be slightly in the past.



A strong transaction only needs to wait for causal transactions originating at the local data center to become uniform.

Scalability of UniStore

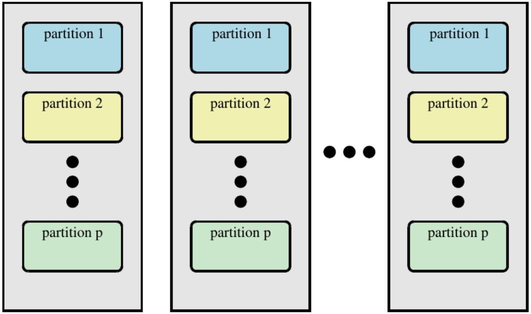
UniStore scales horizontally,

i.e., with the number of machines (partitions) in each data center.

System Model

$$\mathcal{D} = \{1, \ldots, D\} : \text{the set of data centers}$$

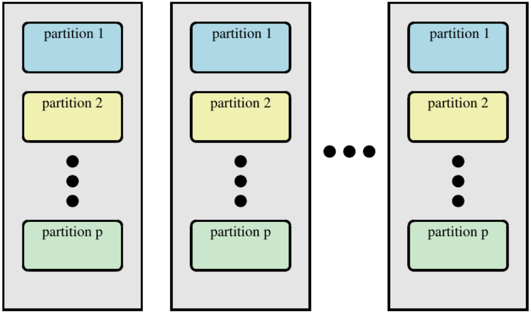
$$\mathcal{P} = \{1, \ldots, N\} : \text{the set of (logical) partitions}$$



p_d^m : the replica of partition m at data center d

System Model

$D = 2f + 1$ and $\leq f$ data centers may fail



Any two replicas are connected by a reliable FIFO channel.

Messages between correct data centers will eventually be delivered.

System Model

Replicas have loosely synchronized physical clocks.

The correctness of UniStore does *not* depend on the precision of clock synchronization.

Fault-tolerant Causal Consistency Protocol

Requirement: Tracking Uniformity

UniStore makes a remote causal transaction visible to clients only after it is uniform.

Requirement: Tracking Uniformity

UniStore makes a remote causal transaction visible to clients only after it is uniform.

Definition (Uniform)

A transaction is uniform if both the transaction and its causal dependencies are guaranteed to be eventually replicated at all correct data centers.

Requirement: Tracking Uniformity

UniStore makes a remote causal transaction visible to clients only after it is uniform.

Definition (Uniform)

A transaction is uniform if both the transaction and its causal dependencies are guaranteed to be eventually replicated at all correct data centers.

A transaction is considered uniform once it is visible at $f+1$ data centers.

Metadata for Causal Transactions

Each transaction is tagged with a commit vector $\mathit{commitVec}$.

$$\mathit{commitVec} \in [\mathcal{D} \to \mathbb{N}]$$

For a transaction originating at data center d,
we call $\mathit{commitVec}[d]$ its local timestamp.

Metadata for Causal Transactions

Each transaction is tagged with a commit vector $commitVec$.

$$commitVec \in [\mathcal{D} \to \mathbb{N}]$$

For a transaction originating at data center d,
we call $commitVec[d]$ its local timestamp.

Commit vectors are sent to sibling replicas
via replication and forwarding.

Metadata for Causal Transactions

Each replica p_d^m maintains the following three vectors:

$$\mathsf{knownVec} \in [\mathcal{D} \to \mathbb{N}]$$

$$\mathsf{stableVec} \in [\mathcal{D} \to \mathbb{N}]$$

$$\mathsf{uniformVec} \in [\mathcal{D} \to \mathbb{N}]$$

Metadata for Causal Transactions

$$\mathsf{knownVec} \in [\mathcal{D} \to \mathbb{N}]$$

Property (Property of knownVec)

For each data center i,
the replica p_d^m stores the updates to partition m
by transactions originating at i with local timestamps $\leq \mathsf{knownVec}[i]$.

Metadata for Causal Transactions

$$\mathsf{stableVec} \in [\mathcal{D} \to \mathbb{N}]$$

Property (Property of $\mathsf{stableVec}$)

For each data center i,
the data center d stores the updates
by transactions originating at i with local timestamps $\leq \mathsf{stableVec}[i]$.

Metadata for Causal Transactions

function BROADCAST_VECS()
  send $\boxed{\text{KNOWNVEC\_LOCAL}}(m, \mathsf{knownVec})$ **to** p_d^l, $l \in \mathscr{P}$
  send STABLEVEC$(d, \mathsf{stableVec})$ **to** p_i^m, $i \in \mathscr{D}$
  send KNOWNVEC_GLOBAL$(d, \mathsf{knownVec})$ **to** p_i^m, $i \in \mathscr{D}$

$$\mathsf{stableVec} \in [\mathcal{D} \to \mathbb{N}]$$

when received KNOWNVEC_LOCAL$(l, knownVec)$
  $\mathsf{localMatrix}[l] \gets knownVec$
  for $i \in \mathscr{D}$ **do**
    $\boxed{\mathsf{stableVec}[i] \gets \min\{\mathsf{localMatrix}[n][i] \mid n \in \mathscr{P}\}}$
  $\mathsf{stableVec}[strong] \gets \min\{\mathsf{localMatrix}[n][strong] \mid n \in \mathscr{P}\}$

Metadata for Causal Transactions

$$\mathsf{uniformVec} \in [\mathcal{D} \to \mathbb{N}]$$

Property (Property of uniformVec)

All update transactions originating at i

with local timestamps $\leq \mathsf{uniformVec}[i]$

are replicated at $f + 1$ data centers including d.

Metadata for Causal Transactions

function BROADCAST_VECS()
  send KNOWNVEC_LOCAL$(m, \mathsf{knownVec})$ **to** p_d^l, $l \in \mathscr{P}$
  send $\boxed{\text{STABLEVEC}}(d, \mathsf{stableVec})$ **to** p_i^m, $i \in \mathscr{D}$
  send KNOWNVEC_GLOBAL$(d, \mathsf{knownVec})$ **to** p_i^m, $i \in \mathscr{D}$

$$\mathsf{uniformVec} \in [\mathcal{D} \to \mathbb{N}]$$

when received STABLEVEC$(i, \mathit{stableVec})$
  $\mathsf{stableMatrix}[i] \leftarrow \mathit{stableVec}$
  $G \leftarrow$ all groups with $\boxed{f+1}$ replicas that include p_d^m
  for $j \in \mathscr{D}$ **do**
    var $\mathit{ts} \leftarrow \max\{\boxed{\min\{\mathsf{stableMatrix}[h][j] \mid h \in g\}} \mid g \in G\}$
    $\mathsf{uniformVec}[j] \leftarrow \max\{\mathsf{uniformVec}[j], \mathit{ts}\}$

Metadata for Causal Transactions

$$\mathsf{uniformVec} \in [\mathcal{D} \to \mathbb{N}]$$

Lemma

All update transactions
with commit vectors $\leq$ $\mathsf{uniformVec}$ *are uniform.*

Metadata for Causal Transactions

$$\mathsf{uniformVec} \in [\mathcal{D} \to \mathbb{N}]$$

Lemma

All update transactions
with commit vectors $\leq \mathsf{uniformVec}$ *are uniform.*

UniStore makes a remote causal transaction visible to clients only after it is uniform.

Causal Consistency Protocol: Start

pastVec : causal past of client

function $\boxed{\text{START()}}$
    p $\leftarrow$ a random partition in data center d
    $\langle \mathsf{tid}, \mathit{snapVec} \rangle \leftarrow$ **send** START_TX(pastVec) **to** p
    pastVec $\leftarrow \mathit{snapVec}$
    return tid

$\forall i \in \mathcal{D} \setminus \{d\}$, all transactions originating at i with local timestamps $\leq \mathsf{pastVec}[i]$ are already uniform.

Causal Consistency Protocol: Start

Causal transactions are executed on an (almost) uniform snapshot.

```
function START_TX(V)
    for i ∈ 𝒟 \ {d} do
        uniformVec[i] ← max{ V[i], uniformVec[i] }
    var tid ← generate_tid()
    snapVec[tid] ← uniformVec
    snapVec[tid][d] ← max{ V[d], uniformVec[d] }
    snapVec[tid][strong] ← max{ V[strong], stableVec[strong] }
    return ⟨tid, snapVec[tid]⟩
```

snapVec$[tid][d]$ ensures "read-your-writes".

Causal Consistency Protocol: Update

```
function UPDATE(k, v)
    _ ← send DO_UPDATE(tid, k, v) to p
    return ok
```

```
function DO_UPDATE(tid, k, v)
    var l ← partition(k)
    wbuff[tid][l][k] ← v
    rset[tid][l] ← rset[tid][l] ∪ {k}
    return ok
```

wbuff$[tid][l]$: buffer for the latest local update on each key

Causal Consistency Protocol: Read

function READ(k)
  $\langle v, c \rangle \leftarrow$ **send** DO_READ(tid, k, lc) **to** p
  if $c \neq \perp$ **then**
    lc $\leftarrow$ max{lc, c}
  return v

function DO_READ(tid, k, c)
  lc $\leftarrow$ max{lc, c}
  var $l \leftarrow$ partition(k)
  if wbuff$[tid][l][k] \neq \perp$ **then**
    return $\langle$wbuff$[tid][l][k], \perp\rangle$
  $\langle v, c \rangle \leftarrow$ **send** READ_KEY(snapVec$[tid]$, k) **to** p_d^l
  rset$[tid][l] \leftarrow$ rset$[tid][l] \cup \{k\}$
  return $\langle v, c \rangle$

Causal Consistency Protocol: Read

Causal transactions are executed on an (almost) uniform snapshot.

when received READ_KEY($snapVec, k$) **from** p
  for $i \in \mathscr{D} \setminus \{d\}$ **do**
    $\text{uniformVec}[i] \leftarrow \max\{snapVec[i], \text{uniformVec}[i]\}$
  wait until $\text{knownVec}[d] \geq snapVec[d] \land \text{knownVec}[strong] \geq snapVec[strong]$
  $\langle v, commitVec, c\rangle \leftarrow \text{snapshot}(\text{opLog}[k], snapVec)$ ▷ returns the latest $commitVec$ (in terms of Lamport clock order in Definition 50) such that $commitVec \leq snapVec$
  send $\langle v, c\rangle$ **to** p

wait : ensure that it is as up-to-date as required by the snapshot

Causal Consistency Protocol: Commit

function $\boxed{\text{COMMIT\_CAUSAL\_TX()}}$

$\langle vc, c \rangle \leftarrow$ **send** COMMIT_CAUSAL(tid, lc) **to** p

pastVec $\leftarrow vc$

lc $\leftarrow c$

return ok

Causal Consistency Protocol: Commit

Read-only transactions returns immediately.

function $\boxed{\textsc{commit\_causal}(tid, c)}$

  $\mathsf{lc} \leftarrow \max\{\mathsf{lc}, c\} + 1$

  if $\forall l \in \mathscr{P}.\ \mathsf{wbuff}[tid][l] = \emptyset$ **then**

    return $\langle \mathsf{snapVec}[tid], \mathsf{lc} \rangle$

  var $commitVec \leftarrow \mathsf{snapVec}[tid]$

  send $\boxed{\textsc{prepare}}(tid, \mathsf{wbuff}[tid][l], \mathsf{snapVec}[tid])$ **to** p_d^l, $l \in \mathscr{P}$

  for all $l \in \mathscr{P}$ **do**

    $\boxed{\textbf{wait receive } \textsc{prepare\_ack}}(tid, ts)$ **from** p_d^l

    $commitVec[d] \leftarrow \max\{commitVec[d], ts\}$

  send $\boxed{\textsc{commit}}(tid, commitVec, \mathsf{lc})$ **to** p_d^l, $l \in \mathscr{P}$

  return $\langle commitVec, \mathsf{lc} \rangle$

2PC protocol for update transactions

Causal Consistency Protocol: Commit

ts : prepare timestamp from its local clock

when received PREPARE$(tid, wbuff, snapVec)$ **from** p
    for $i \in \mathscr{D} \setminus \{d\}$ **do**
        $\mathsf{uniformVec}[i] \leftarrow \max\{snapVec[i], \mathsf{uniformVec}[i]\}$
    var $\boxed{ts \leftarrow \mathsf{clock}}$
    $\mathsf{preparedCausal} \leftarrow \mathsf{preparedCausal} \cup \{\langle tid, wbuff, ts \rangle\}$
    send PREPARE_ACK(tid, ts) **to** p

Causal Consistency Protocol: Commit

wait : ensure that its local clock is up-to-date

when received COMMIT$(tid, commitVec, c)$

  $\boxed{\textbf{wait until } \mathsf{clock} \geq commitVec[d]}$

  $\langle tid, wbuff, \_\rangle \leftarrow \mathsf{find}(tid, \mathsf{preparedCausal})$

  $\boxed{\mathsf{preparedCausal}} \leftarrow \mathsf{preparedCausal} \setminus \{\langle tid, \_, \_\rangle\}$

  for all $\langle k, v\rangle \in wbuff$ **do**

    $\mathsf{opLog}[k] \leftarrow \mathsf{opLog}[k] \cdot \langle v, commitVec, c\rangle$

  $\boxed{\mathsf{committedCausal}[d]} \leftarrow \mathsf{committedCausal}[d] \cup \{\langle tid, wbuff, commitVec, c\rangle\}$

$\mathsf{committedCausal}[d]$: for replication

Causal Consistency Protocol: Replication

Property (Property of knownVec)

For each data center i,

the replica p_d^m stores the updates to partition m

by transactions originating at i with local timestamps $\leq \mathsf{knownVec}[i]$.

function PROPAGATE_LOCAL_TXS()

  if $\mathsf{preparedCausal} = \emptyset$ **then**

    $\boxed{\mathsf{knownVec}[d] \leftarrow \mathsf{clock}}$

  else

    $\boxed{\mathsf{knownVec}[d] \leftarrow \min\{ts \mid \langle \_, \_, ts \rangle \in \mathsf{preparedCausal}\} - 1}$

  var $txs \leftarrow \{\langle \_, \_, commitVec, c \rangle \in \mathsf{committedCausal}[d] \mid commitVec[d] \leq \mathsf{knownVec}[d]\}$

  if $txs \neq \emptyset$ **then**

    send $\boxed{\text{REPLICATE}(d, txs)}$ **to** $p_i^m, i \in \mathscr{D} \setminus \{d\}$

    $\mathsf{committedCausal}[d] \leftarrow \mathsf{committedCausal}[d] \setminus txs$

  else

    send $\boxed{\text{HEARTBEAT}(d, \mathsf{knownVec}[d])}$ **to** $p_i^m, i \in \mathscr{D} \setminus \{d\}$

HEARTBEAT : for liveness

Adding Strong Transactions

Requirement: ConflictOrdering

$$\forall t_1, t_2 \in T_{strong}.\ t_1 \bowtie t_2 \implies t_1 \prec t_2 \lor t_2 \prec t_1.$$

Each strong transaction is assigned a scalar strong timestamp.

$$commitVec \in [\mathcal{D} \cup \{strong\} \to \mathbb{N}]$$

Metadata for Strong Transactions

$$\mathsf{knownVec} \in [\mathcal{D} \cup \{strong\} \to \mathbb{N}]$$

Property (Property of $\mathsf{knownVec}[strong]$)

Replica p_d^m stores the updates to m by all strong transactions with $commitVec[strong] \le \mathsf{knownVec}[strong]$.

Metadata for Strong Transactions

$$\mathsf{stableVec} \in [\mathcal{D} \cup \{strong\} \to \mathbb{N}]$$

when received KNOWNVEC_LOCAL($l, knownVec$)
  $\mathsf{localMatrix}[l] \gets knownVec$
  for $i \in \mathscr{D}$ **do**
    $\mathsf{stableVec}[i] \gets \min\{\mathsf{localMatrix}[n][i] \mid n \in \mathscr{P}\}$
  $\mathsf{stableVec}[strong] \gets \min\{\mathsf{localMatrix}[n][strong] \mid n \in \mathscr{P}\}$

Property (Property of $\mathsf{stableVec}[strong]$)

Data center d stores the updates by all strong transactions with $commitVec[strong] \leq \mathsf{knownVec}[strong]$.

Metadata for Strong Transactions

$$\mathsf{uniformVec} \in [\mathcal{D} \to \mathbb{N}]$$

```
when received STABLEVEC(i, stableVec)
    stableMatrix[i] ← stableVec
    G ← all groups with f + 1 replicas that include p_d^m
    for j ∈ 𝒟 do
        var ts ← max{min{stableMatrix[h][j] | h ∈ g} | g ∈ G}
        uniformVec[j] ← max{uniformVec[j], ts}
```

The commit protocol for strong transactions guarantees their uniformity.

Strong Consistency Protocol: Commit

function COMMIT_STRONG(tid, c)
    UNIFORM_BARRIER(snapVec[tid])
    $\langle d, vc, c \rangle \leftarrow$ CERTIFY(tid, wbuff[tid], rset[tid], snapVec[tid], c)
    lc $\leftarrow$ max{lc, c} $+ 1$
    return $\langle d, vc,$ lc$\rangle$

A strong transaction only needs to wait for causal transactions originating at the local data center to become uniform.

function UNIFORM_BARRIER(V, c)
    lc $\leftarrow$ max{lc, c} $+ 1$
    wait until uniformVec[d] $\geq V[d]$
    return lc

Strong Consistency Protocol: Commit

function $\textsc{Commit\_Strong}(tid, c)$

  $\boxed{\textsc{Uniform\_Barrier}(\mathsf{snapVec}[tid])}$

  $\langle d, vc, c \rangle \leftarrow \boxed{\textsc{Certify}}(tid, \mathsf{wbuff}[tid], \mathsf{rset}[tid], \mathsf{snapVec}[tid], c)$

  $\mathsf{lc} \leftarrow \max\{\mathsf{lc}, c\} + 1$

  return $\langle d, vc, \mathsf{lc} \rangle$

$$\langle d \in \{\textsc{commit}, \textsc{abort}\}, vc \rangle \leftarrow \textsc{certify}(t)$$

Strong Consistency Protocol: Commit

function COMMIT_STRONG(tid, c)
  $\boxed{\text{UNIFORM\_BARRIER}(\mathsf{snapVec}[tid])}$
  $\langle d, vc, c \rangle \leftarrow \boxed{\text{CERTIFY}}(tid, \mathsf{wbuff}[tid], \mathsf{rset}[tid], \mathsf{snapVec}[tid], c)$
  $\mathsf{lc} \leftarrow \max\{\mathsf{lc}, c\} + 1$
  return $\langle d, vc, \mathsf{lc} \rangle$

$$\langle d \in \{\text{COMMIT}, \text{ABORT}\}, vc \rangle \leftarrow \text{CERTIFY}(t)$$

Multi-Shot Distributed Transaction Commit

Gregory Chockler
Royal Holloway, University of London, UK

Alexey Gotsman[1]
IMDEA Software Institute, Madrid, Spain

White-Box Atomic Multicast

Alexey Gotsman, IMDEA Software Institute
Anatole Lefort, Télécom SudParis
Gregory Chockler, Royal Holloway, University of London

2PC across partitions + Paxos among replicas of each partition

uses white-box optimizations that minimize the commit latency

Strong Consistency Protocol: Deliver

upon DELIVER_UPDATES(W)

  for $\langle k, v, commitVec, c \rangle \in W$ in $\boxed{commitVec[strong] \text{ order}}$ **do**

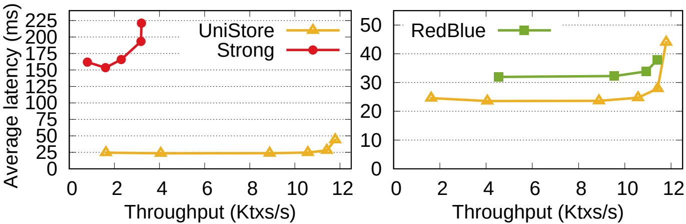
    $\mathsf{opLog}[k] \leftarrow \mathsf{opLog}[k] \cdot \langle v, commitVec, c \rangle$

    $\mathsf{knownVec}[strong] \leftarrow commitVec[strong]$

Evaluation

Performance of UniStore

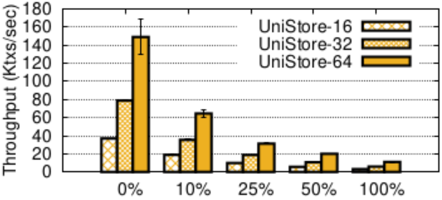
Throughput: 5% and 259% higher than RedBlue and Strong



RUBiS benchmark: throughput vs. average latency.

Latency: 24ms vs. 32ms of RedBlue and 162ms of Strong

Scalability of UniStore



Scalability when varying the ratio of strong transactions.

UniStore is able to scales almost linearly.

Evaluation

For more evaluations, please refer to the paper.

Conclusion

UNISTORE is a fast, scalable, and fault-tolerant

transactional distributed key-value store

that supports a combination of weak and strong consistency.

Conclusion

UniStore is a fast, scalable, and fault-tolerant

transactional distributed key-value store

that supports a combination of weak and strong consistency.

"We expect the key ideas in UniStore to pave the way
for practical systems that combine causal and strong consistency."

总结

魏恒峰 (hfwei@nju.edu.cn)

聘期合同要求	工作情况
教学: 承担一门课程	问题求解课程 五个学期; 共 164 学时 (2019 级本科生 “我心目中的好课程”)
科研: 4-6 篇高水平论文	发表 3 篇 (含 1 篇短文) 在审 4 篇 (2017 年 CCF 优秀博士学位论文奖)
人才培养	负责或协助指导学生 9 人次 (学术积累: 组织 TLA$^+$ 与 Coq 讨论班)
主持/参与 多个基金项目	主持 1 项; 参与 1 项 个人可支配总经费 75 万元

Hengfeng Wei (hfwei@nju.edu.cn)