分布式事务一致性：协议设计与系统测试

魏恒峰

hfwei@nju.edu.cn

2022 年 11 月 04 日

数据一致性问题

理论基础:

协议设计:

协议验证:

系统测试:

经典、传统与现代

技术发展趋势

读写寄存器 (Read/Write Register)

(Since 2012)

分布式 NoSQL Key-Value 数据库

复制数据类型 (Replicated Data Types)

(Since 2017)

分布式事务 (Distributed Transactions)

(Since 2020)
+ research outcomes

What is ?

is a fast, scalable, and fault-tolerant

transactional distributed key-value store

that supports a combination of weak and strong consistency.

What is ?

is a fast, scalable, and fault-tolerant

transactional distributed key-value store

that supports a combination of weak and strong consistency.

Weak consistency:

Strong consistency:

Why UNI-?

Weak consistency: low latency, high availability

Strong consistency: easy to preserve critical application invariants

Why UNI-?

DEPOSIT WITHDRAW QUERY INTEREST

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

Why UNI-?

DEPOSIT WITHDRAW QUERY INTEREST

Invariant: balance ≥ 0

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

Why UNI-?

DEPOSIT WITHDRAW QUERY INTEREST

Invariant: balance ≥ 0

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

Only WITHDRAW needs to use strong consistency.

Consistency Model of

implements a transactional variant of

Partial Order-Restrictions () consistency [Li@ACT'2018]

(I) transactional causal consistency by default

(II) to specify conflicting transactions under strong consistency

Consistency Model of

Definition (Session Order)

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

Definition (Conflict Relation)

The conflict relation, denoted , between transactions is a symmetric relation.

$$t_1\,t_2 \iff t_2\,t_1.$$

Consistency Model of

Definition ()

A set of transactions $\triangleq \uplus$ committed by satisfies if there exists a causal order $\prec$ on T such that

Consistency Model of

Definition ()

A set of transactions $\triangleq \uplus$ committed by satisfies if there exists a causal order $\prec$ on T such that

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

Consistency Model of

Definition ()

A set of transactions $\triangleq \uplus$ committed by satisfies if there exists a causal order $\prec$ on T such that

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

: $\forall t_1, t_2 \in . \; t_1 t_2 \implies t_1 \prec t_2 \lor t_2 \prec t_1$.

Consistency Model of

Definition ()

A set of transactions $\triangleq \uplus$ committed by satisfies if there exists a causal order $\prec$ on T such that

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

: $\forall t_1, t_2 \in . \ t_1 t_2 \implies t_1 \prec t_2 \lor t_2 \prec t_1$.

: 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

Definition ()

A set of transactions $\triangleq \uplus$ committed by satisfies if there exists a causal order $\prec$ on T such that

- : '$\prec$' is a partial order and $\subseteq \prec$.
- : $\forall t_1, t_2 \in . \; t_1 t_2 \implies t_1 \prec t_2 \lor t_2 \prec t_1$.
- : 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.
- : $\land$

Consistency Model of

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

: read from the latest update on k preceding r in t

$$= \wedge$$

: read from the last update on k

of the latest transaction (in an order consistent with $\prec$) preceding t

Consistency Model of

DEPOSIT WITHDRAW QUERY INTEREST

Invariant: balance ≥ 0

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

Design Challenge of

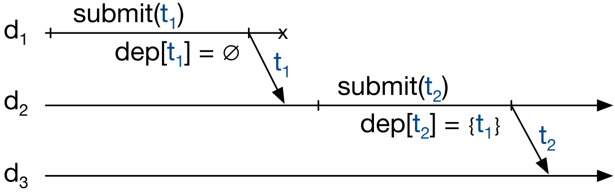
To satisfy liveness () despite failures

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 (I)

Data center d_1 crashes

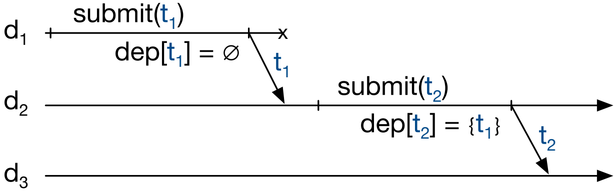
before t_1 is replicated to correct data center d_3.



Design Challenge of (I)

Data center d_1 crashes

before t_1 is replicated to correct data center d_3.



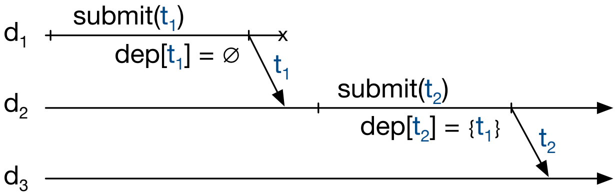
Transaction t_2 (at correct data center d_2)

may never become visible at correct data center d_3.

Fault-tolerance of (I)

Data center d_1 crashes

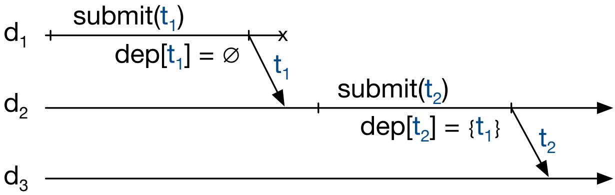
before t_1 is replicated to correct data center d_3.



Fault-tolerance of (I)

Data center d_1 crashes

before t_1 is replicated to correct data center d_3.



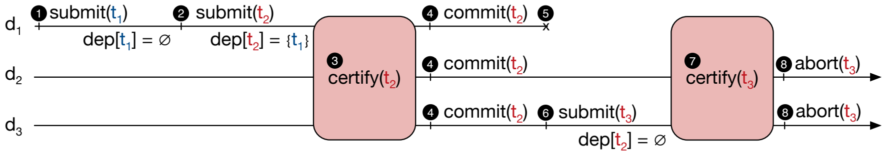
Data center d_2 need to forward causal transactions

to other data centers.

Design Challenge of (II)

Data center d_1 crashes

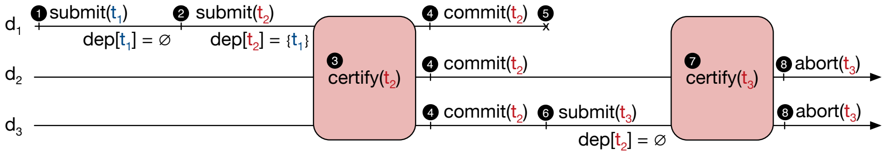
before t_1 is replicated to correct data center d_3.



Design Challenge of (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

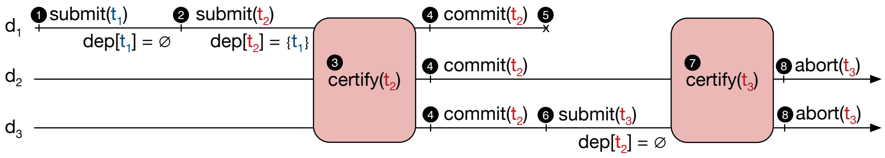
(by).

Fault-tolerance of (II)

ensures that before a strong transaction commits,

all its causal dependencies are uniform,

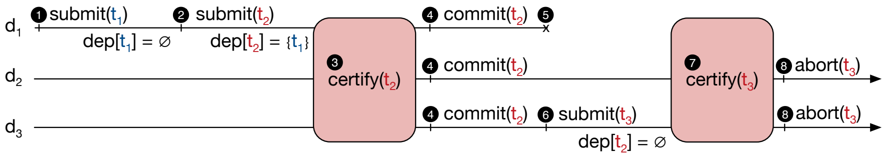
i.e., will eventually become visible at all correct data centers.



ensures that before a strong transaction commits,

all its causal dependencies are uniform,

i.e., will eventually become visible at all correct data centers.



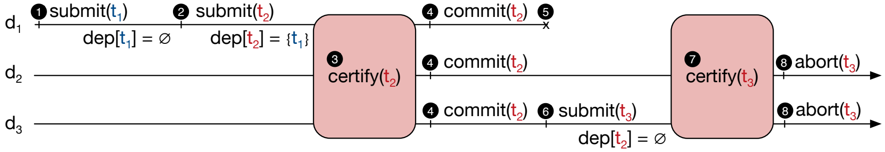
Transaction t_1 will eventually be visible at d_3.

Transaction t_2 will eventually be visible at d_3.

Transaction t_3 may be committed at d_3.

Performance of

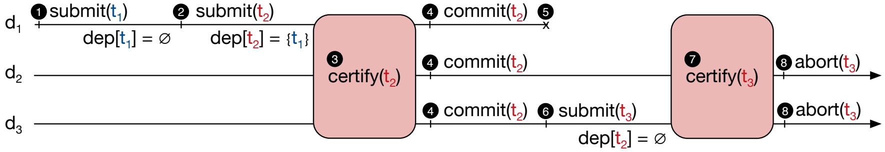
Causal transactions remain highly-available, i.e., committed locally.



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

Performance of

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



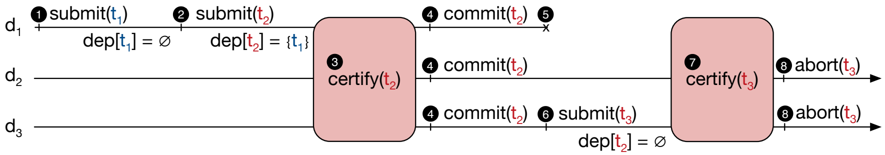
A strong transaction may have to wait for some of its dependencies to become uniform before committing.

However, this may cost too much.

Performance of

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

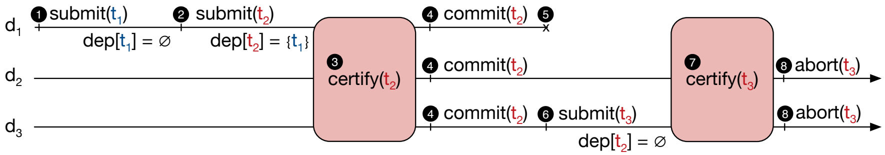
Causal transactions are executed on an (almost) uniform snapshot that may be slightly in the past.



Performance of

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

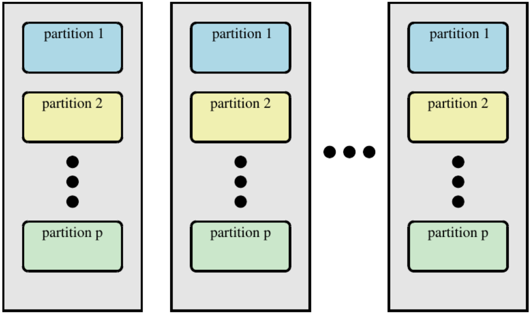
scales horizontally,

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

System Model

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

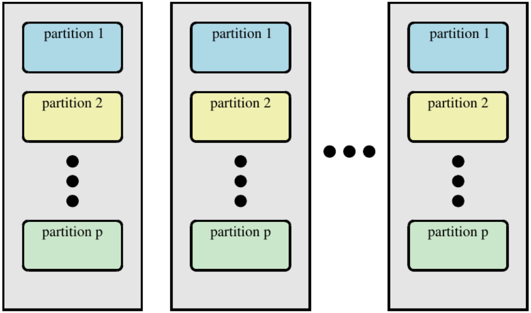
$$\P = 1, \ldots, N : \text{the set of (logical) partitions}$$



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 does not
depend on the precision of clock synchronization.

Fault-tolerant Causal Consistency Protocol

Requirement: Tracking Uniformity

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

Requirement: Tracking Uniformity

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

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 .

$$\in [\to]$$

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

Metadata for Causal Transactions

Each transaction is tagged with a commit vector .

$$\in [\rightarrow]$$

For a transaction originating at data center d,
we call $[d]$ 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:

$$\in [\rightarrow]$$

$$\in [\rightarrow]$$

$$\in [\rightarrow]$$

Metadata for Causal Transactions

$$\in [\rightarrow]$$

Property (Property of)

For each data center i,

the replica p_d^m stores the updates to partition m

by transactions originating at i with local timestamps $\leq [i]$.

Metadata for Causal Transactions

$$\in [\rightarrow]$$

Property (Property of)

For each data center i,

the data center d stores the updates

by transactions originating at i with local timestamps $\leq [i]$.

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** $p_i^m,\ i \in \mathscr{D}$

$$\in [\rightarrow]$$

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

Metadata for Causal Transactions

$$\in [\rightarrow]$$

Property (Property of)

All update transactions originating at i

with local timestamps $\leq [i]$

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

Metadata for Causal Transactions

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

$$\in [\rightarrow]$$

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

Metadata for Causal Transactions

$$\in [\rightarrow]$$

Lemma

All update transactions
with *commit vectors* $\leq$ *are uniform.*

Metadata for Causal Transactions

$\in [\rightarrow]$

Lemma

All update transactions

with commit vectors $\leq$ are uniform.

makes a remote causal transaction visible to clients

only after it is uniform.

Causal Consistency Protocol: Start

: causal past of client

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

$\forall i \in \setminus d$, all transactions originating at i

with local timestamps $\leq [i]$ 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]⟩
```

$[tid][d]$ 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
```

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

Causal Consistency Protocol: Read

```
function READ(k)
    ⟨v, c⟩ ← send DO_READ(tid, k, lc) to p
    if c ≠ ⊥ then
        lc ← max{lc, c}
    return v
```

```
function DO_READ(tid, k, c)
    lc ← max{lc, c}
    var l ← partition(k)
    if wbuff[tid][l][k] ≠ ⊥ then
        return ⟨wbuff[tid][l][k], ⊥⟩
    ⟨v, c⟩ ← send READ_KEY(snapVec[tid], k) to p_d^l
    rset[tid][l] ← rset[tid][l] ∪ {k}
    return ⟨v, c⟩
```

Causal Consistency Protocol: Read

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

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

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

Causal Consistency Protocol: Commit

```
function COMMIT_CAUSAL_TX()
    ⟨vc, c⟩ ← send COMMIT_CAUSAL(tid, lc) to p
    pastVec ← vc
    lc ← c
    return ok
```

Causal Consistency Protocol: Commit

Read-only transactions returns immediately.

function $\boxed{\text{COMMIT\_CAUSAL}(tid, c)}$

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

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

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

    $\boxed{\textbf{wait receive}\ \text{PREPARE\_ACK}}(tid, ts)$ **from** p_d^l

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

  send $\boxed{\text{COMMIT}}(tid, commitVec, \mathsf{lc})$ **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**
    $\text{uniformVec}[i] \leftarrow \max\{snapVec[i], \text{uniformVec}[i]\}$
  var $\boxed{ts \leftarrow \text{clock}}$
  $\text{preparedCausal} \leftarrow \text{preparedCausal} \cup \{\langle tid, wbuff, ts\rangle\}$
  send PREPARE_ACK(tid, ts) **to** p

Causal Consistency Protocol: Commit

: 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**
        $\mathsf{opLog}[k] \leftarrow \mathsf{opLog}[k] \cdot \langle v, commitVec, c \rangle$
    $\boxed{\mathsf{committedCausal}[d]} \leftarrow \mathsf{committedCausal}[d] \cup \{\langle tid, wbuff, commitVec, c \rangle\}$

$[d]$: for replication

Causal Consistency Protocol: Replication

Property (Property of)

For each data center i,

the replica p_d^m stores the updates to partition m

by transactions originating at i with local timestamps $\leq [i]$.

function PROPAGATE_LOCAL_TXS()
  if preparedCausal $= \emptyset$ **then**
    $\boxed{\text{knownVec}[d] \leftarrow \text{clock}}$
  else
    $\boxed{\text{knownVec}[d] \leftarrow \min\{ts \mid \langle \_, \_, ts \rangle \in \text{preparedCausal}\} - 1}$
  var $txs \leftarrow \{\langle \_, \_, commitVec, c \rangle \in \text{committedCausal}[d] \mid commitVec[d] \leq \text{knownVec}[d]\}$
  if $txs \neq \emptyset$ **then**
    send $\boxed{\text{REPLICATE}(d, txs)}$ **to** p_i^m, $i \in \mathcal{D} \setminus \{d\}$
    $\text{committedCausal}[d] \leftarrow \text{committedCausal}[d] \setminus txs$
  else
    send $\boxed{\text{HEARTBEAT}(d, \text{knownVec}[d])}$ **to** p_i^m, $i \in \mathcal{D} \setminus \{d\}$

: for liveness

Adding Strong Transactions

Requirement:

$$\forall t_1, t_2 \in . \; t_1 t_2 \implies t_1 \prec t_2 \vee t_2 \prec t_1.$$

Each strong transaction is assigned a scalar strong timestamp.

$$\in [\cup \to]$$

Metadata for Strong Transactions

$$\in [\cup \to]$$

Property (Property of [])

Replica p_d^m stores the updates to m by all strong transactions with $[] \leq []$.

Metadata for Strong Transactions

$$\in [\cup \to]$$

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

Property (Property of [])

Data center d stores the updates by all strong transactions with $[] \leq []$.

Metadata for Strong Transactions

$$\in [\rightarrow]$$

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

The commit protocol for strong transactions guarantees their uniformity.

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

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

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

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

$$\langle d \in ,, vc \rangle \leftarrow (t)$$

Strong Consistency Protocol: Commit

```
function COMMIT_STRONG(tid, c)
    UNIFORM_BARRIER(snapVec[tid])
    ⟨d, vc, c⟩ ← CERTIFY(tid, wbuff[tid], rset[tid], snapVec[tid], c)
    lc ← max{lc, c} + 1
    return ⟨d, vc, lc⟩
```

$$\langle d \in ,, vc \rangle \leftarrow (t)$$

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**

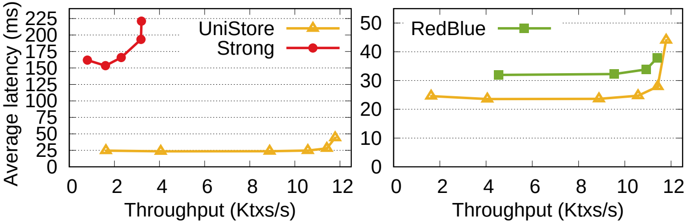
    $\mathsf{opLog}[k] \leftarrow \mathsf{opLog}[k] \cdot \langle v, commitVec, c \rangle$

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

Evaluation

Performance of

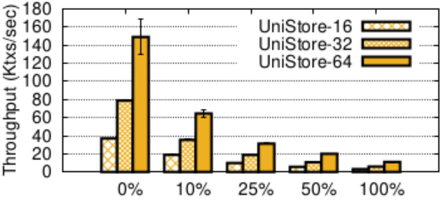
Throughput: 5% and 259% higher than RedBlue and Strong



RUBiS benchmark: throughput vs. average latency.

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

Scalability of



Scalability when varying the ratio of strong transactions.

is able to scales almost linearly.

Evaluation

For more evaluations, please refer to the paper.

Conclusion

is a fast, scalable, and fault-tolerant

transactional distributed key-value store

that supports a combination of weak and strong consistency.

Conclusion

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 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)