Consistency in Transactional Distributed Databases: Protocols and Testing

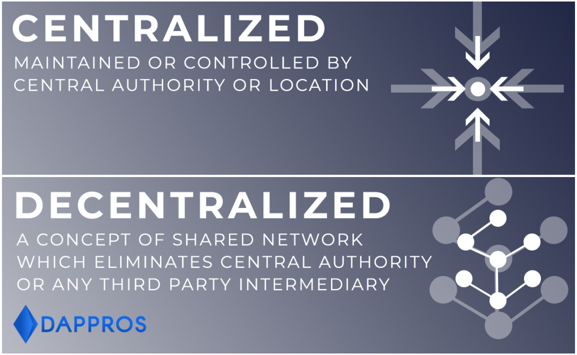
Hengfeng Wei (魏恒峰)

hfwei@nju.edu.cn

Nov. 04, 2022

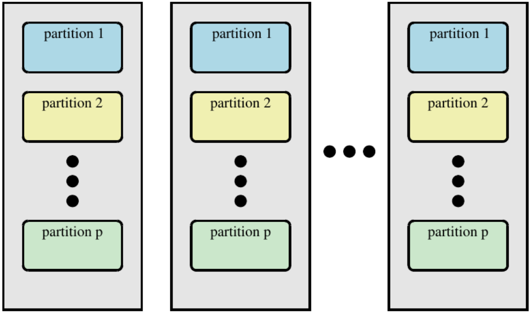
Background

Centralized Databases *vs.* Distributed Databases



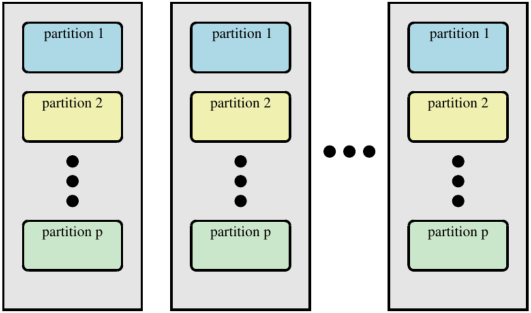
Data Consistency Problem

The Classic "Data Partition + Data Replication" Architecture



Data Consistency Problem

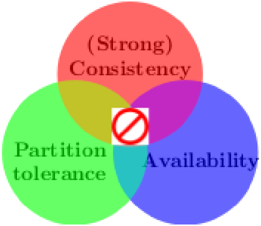
The Classic “Data Partition + Data Replication” Architecture



(Distributed) Data Consistency Problem

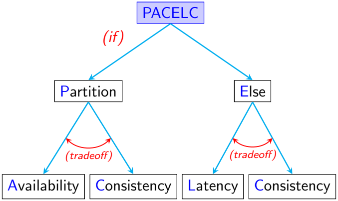
Data Consistency Problem

(Strong) Consistency, Availability, Latency、Patition tolerance



CAP Theorem

(Brewer@PODC2000)



PACELC Tradeoff

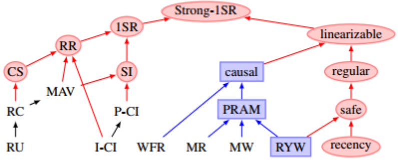
(Abadi@Computer2012)

Data Consistency Problem

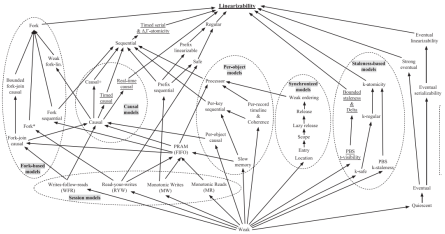
没有一劳永逸的解决方案

Consistency Models

Use various consistency models to capture these tradeoffs



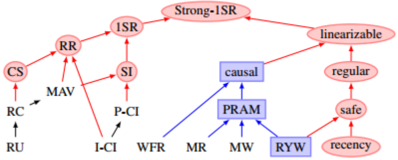
(Bailis@VLDB2012)



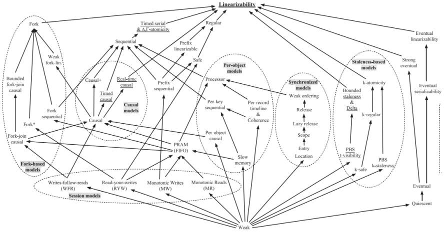
(Viotti@CSUR2016)

Consistency Models

Use various consistency models to capture these tradeoffs



(Bailis@VLDB2012)



(Viotti@CSUR2016)

My Researches

On the data consistency theory around consistency models:

My Researches

On the data consistency theory around consistency models:

Computability: What is possible or impossible?

My Researches

On the data consistency theory around consistency models:

Computability: What is possible or impossible?

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

My Researches

On the data consistency theory 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 algorithms?

My Researches

On the data consistency theory 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 algorithms?

My Researches

On the data consistency theory 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 algorithms?

Classic problems with the ever-changing requirements

My Researches (I)

On read/write registers[a] ($\geq$ 2014)

cassandra

Project Voldemort
A distributed database.
zoie · bobo · cleo · decomposer · norbert

amazon DynamoDB

etcd

Distributed Non-transactional Key-Value Stores

[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 (I)

Hengfeng Wei, Marzio De Biasi, Yu Huang, Jiannong Cao, and Jian Lu. "Verifying Pipelined-RAM consistency over read/write traces of data replicas". In: *IEEE Trans. Parallel Distrib. Syst. (**TPDS'2016**)* 27.5 (May 2016), pp. 1511–1523. DOI: 10.1109/TPDS.2015.2453985

Hengfeng Wei, Yu Huang, and Jian Lu. "Probabilistically-Atomic 2-Atomicity: enabling almost strong consistency in distributed storage systems". In: *IEEE Trans. Comput. (**TC'2017**)* 66.3 (Mar. 2017), pp. 502–514. DOI: 10.1109/TC.2016.2601322

Kaile Huang, Yu Huang, and Hengfeng Wei. "Fine-Grained Analysis on Fast Implementations of Distributed Multi-Writer Atomic Registers". In: *Proceedings of the 39th Symposium on Principles of Distributed Computing (**PODC'2020**)*. Association for Computing Machinery, 2020, pp. 200–209. DOI: 10.1145/3382734.3405698

My Researches (II)

On replicated data types[b] ($\geq$ 2017)

(a) Google Docs

(b) Apache Wave

(c) Wikipedia

(d) LaTeX Editor

[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 (II)

Hengfeng Wei, Yu Huang, and Jian Lu. "Brief Announcement: Specification and Implementation of Replicated List: The Jupiter Protocol Revisited". In: *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*. PODC '18. ACM, 2018, pp. 81–83

Hengfeng Wei, Yu Huang, and Jian Lu. "Specification and Implementation of Replicated List: The Jupiter Protocol Revisited". In: *22nd International Conference on Principles of Distributed Systems, OPODIS 2018*. 2018, 12:1–12:16

Xue Jiang, Hengfeng Wei*, and Yu Huang. "A Generic Specification Framework for Weakly Consistent Replicated Data Types". In: *International Symposium on Reliable Distributed Systems (**SRDS'2020**)*. 2020, pp. 143–154. DOI: 10.1109/SRDS51746.2020.00022

Ye Ji, Hengfeng Wei*, Yu Huang, and Jian Lu. "Specifying and verifying CRDT protocols using TLA^{+}". In: *Journal of Software (**软件学报 JOS'2020**)* 31.5 (2020), pp. 1332–1352. DOI: 10.13328/j.cnki.jos.005956

My Researches (III)

On distributed transactions[c] ($\geq$ 2020)

[c]分布式事务。每个事务由一组操作构成，这些操作要么都成功，要么都失败。

My Researches (III)

Manuel Bravo, Alexey Gotsman, Borja de Régil, and Hengfeng Wei. "UniStore: A fault-tolerant marriage of causal and strong consistency". In: *2021 USENIX Annual Technical Conference (**USENIX ATC'2021**)*. USENIX Association, July 2021, pp. 923–937

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 weak and strong consistency.

Overview of the Work on UniStore

Partial Order-Restrictions (PoR) Consistency

CC < PoR < SER

CC: CausalConsistency; SER: Serializability

Overview of the Work on UniStore

Partial Order-Restrictions (PoR) Consistency

CC < PoR < SER

CC: CausalConsistency; SER: Serializability

Challenges (I): To ensure liveness of the system in the presence of data center failures

Overview of the Work on UniStore

Partial Order-Restrictions (PoR) Consistency

CC < PoR < SER

CC: CausalConsistency; SER: Serializability

Challenges (I): To ensure liveness of the system in the presence of data center failures

Challenges (II): To provide a rigorous correctness proof of the protocol which is rather complicated

Overview of the Work on UniStore

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

Manuel Bravo Alexey Gotsman Borja de Régil Hengfeng Wei *
IMDEA Software Institute *Nanjing University*

ATC'2021 (CCF A)

I am fully responsible for developing a rigorous correctness proof:

[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.

Overview of the Work on UniStore

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

Manuel Bravo Alexey Gotsman Borja de Régil Hengfeng Wei *
IMDEA Software Institute *Nanjing University*

ATC'2021 (CCF A)

I am fully responsible for developing a rigorous correctness proof:

- Finished a proof of 20 pages contained in the arXiv version

[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.

Overview of the Work on UniStore

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

Manuel Bravo Alexey Gotsman Borja de Régil Hengfeng Wei *
IMDEA Software Institute *Nanjing University*

ATC'2021 (CCF A)

I am fully responsible for developing a 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], and proposed solutions to fix them

[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.

Why Uni-?

Weak Consistency: low latency, high availability, but unable to preserve critical application invariants

Strong Consistency: easy to preserve critical application invariants, but require global synchronization among data centers

The UNI- Approach

- Multiple consistency levels coexist in a store

The Uni- Approach

- Multiple consistency levels coexist in a store
- Take weak consistency as the default and the baseline

The Uni- Approach

- Multiple consistency levels coexist in a store
- Take weak consistency as the default and the baseline
- Programmers can choose the transactions that should be executed under strong consistency

The Uni- Approach

- Multiple consistency levels coexist in a store
- Take weak consistency as the default and the baseline
- Programmers can choose the transactions that should be executed under strong consistency
  - e.g., if the execution of a set of transactions may violate the application violations

The UNI- Approach

Partial Order-Restrictions (PoR) Consistency

(Li@OSDI'2012, Li@ACT'2018)

The Uni- Approach

Partial Order-Restrictions (PoR) Consistency

(Li@OSDI'2012, Li@ACT'2018)

- PoR allows programmers to classify transactions as either causal or strong.

The UNI- Approach

Partial Order-Restrictions (PoR) Consistency

(Li@OSDI'2012, Li@ACT'2018)

- PoR allows programmers to classify transactions as either causal or strong.
- Causal transactions satisfy CAUSALCONSISTENCY:
  - Clients see updates in an order that respects the potential causality between them.
  - Causally independent transactions can be executed concurrently.

The UNI- Approach

Partial Order-Restrictions (PoR) Consistency

(Li@OSDI'2012, Li@ACT'2018)

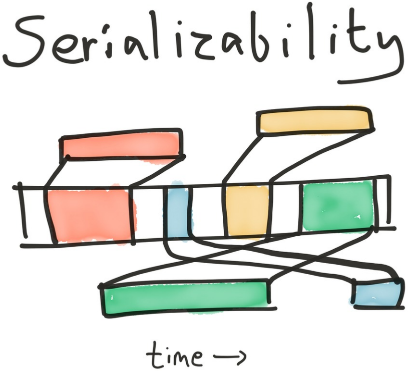
- PoR allows programmers to classify transactions as either causal or strong.
- Causal transactions satisfy CAUSALCONSISTENCY:
  - Clients see updates in an order that respects the potential causality between them.
  - Causally independent transactions can be executed concurrently.
- Programmers use strong transactions to enforce orders between some causally independent transactions.

CausalConsistency

TODO: +fig Clients see updates in an order that respects the potential causality between them.

Causally independent transactions can be executed concurrently.

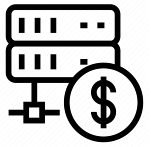
Serializability

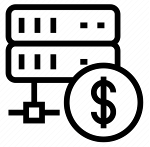


All transactions seem to be executed in some sequential order.

A Banking Application

DEPOSIT WITHDRAW QUERY INTEREST

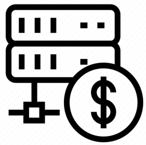


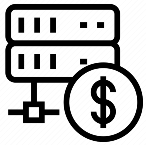


Invariant: $\mathsf{balance} \geq 0$

A Banking Application

DEPOSIT WITHDRAW QUERY INTEREST



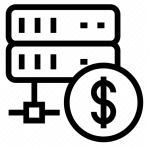


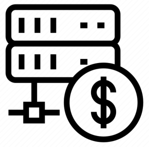
Invariant: $\mathsf{balance} \geq 0$

DEPOSIT operations can be executed under CAUSALCONSISTENCY.

A Banking Application

DEPOSIT WITHDRAW QUERY INTEREST





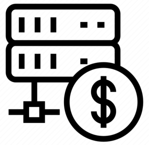
Invariant: $\mathsf{balance} \geq 0$

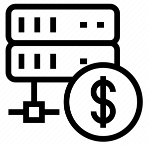
DEPOSIT operations can be executed under CAUSALCONSISTENCY.

$\text{DEPOSIT}(50)$

A Banking Application

DEPOSIT WITHDRAW QUERY INTEREST

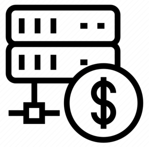


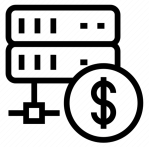


Invariant: $\mathsf{balance} \geq 0$

A Banking Application

DEPOSIT WITHDRAW QUERY INTEREST



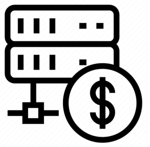


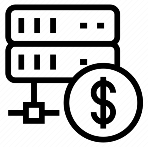
Invariant: $\textsf{balance} \geq 0$

However, CAUSALCONSISTENCY also allows two causally independent WITHDRAW to execute concurrently, without knowing each other.

A Banking Application

DEPOSIT WITHDRAW QUERY INTEREST





Invariant: balance ≥ 0

However, CausalConsistency also allows two causally independent WITHDRAW to execute concurrently, without knowing each other.

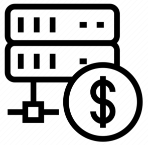
WITHDRAW(60)

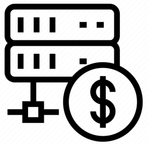
The PoR Approach

- The programmer provides a symmetric conflict relation $\bowtie$ on transactions.
- Any transaction involved in the conflict relation is marked strong.
- PoR ensures that conflicting transactions are executed serially.

A Banking Application

DEPOSIT WITHDRAW QUERY INTEREST





Invariant: balance ≥ 0

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

Related Work

non-transactional

Hagit Attiya and Roy Friedman. "A Correctness Condition for High-Performance Multiprocessors". In: *SIAM Journal on Computing* 27.6 (1998), pp. 1637–1670

Hagit Attiya, Soma Chaudhuri, Roy Friedman, and Jennifer L. Welch. "Shared Memory Consistency Conditions for Nonsequential Execution: Definitions and Programming Strategies". In: *SIAM Journal on Computing* 27.1 (1998), pp. 65–89

Related Work

not fault-tolerant

Valter Balegas, Sérgio Duarte, Carla Ferreira, Rodrigo Rodrigues, Nuno Preguiça, Mahsa Najafzadeh, and Marc Shapiro. "Putting Consistency Back into Eventual Consistency". In: *Proceedings of the Tenth European Conference on Computer Systems*. EuroSys '15. 2015

Cheng Li, Daniel Porto, Allen Clement, Johannes Gehrke, Nuno Preguiça, and Rodrigo Rodrigues. "Making Geo-replicated Systems Fast As Possible, Consistent when Necessary". In: *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*. OSDI '12. 2012, pp. 265–278

Cheng Li, Nuno Preguiça, and Rodrigo Rodrigues. "Fine-Grained Consistency for Geo-Replicated Systems". In: *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference*. USENIX ATC '18. 2018, pp. 359–371

Liveness of UNISTORE

Liveness of UniStore

How to ensure liveness despite data center failures?

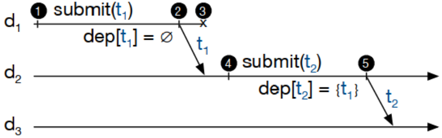
Liveness of UniStore

How to ensure liveness despite data center 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.

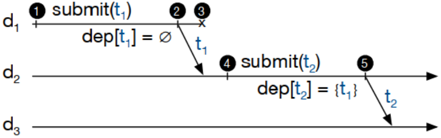
Liveness of Causal Transactions

d_1 crashes before t_1 is replicated to d_3.



Liveness of Causal Transactions

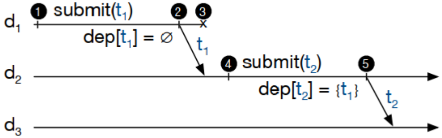
d_1 crashes before t_1 is replicated to d_3.



Transaction t_2 may never become visible at d_3.

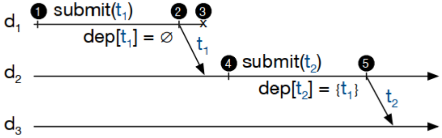
Liveness of Causal Transactions

d_1 crashes before t_1 is replicated to d_3.



Liveness of Causal Transactions

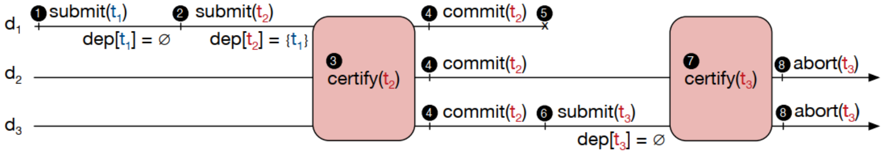
d_1 crashes before t_1 is replicated to d_3.



d_2 need to forward causal transactions to other data centers.

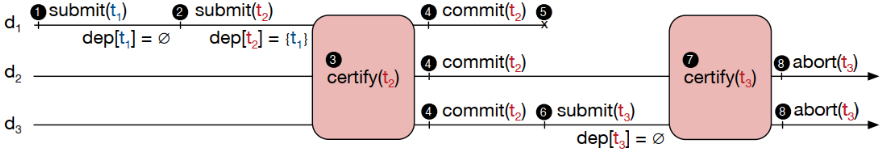
Liveness of Strong Transactions

d_1 crashes before t_1 is replicated to d_3.



Liveness of Strong Transactions

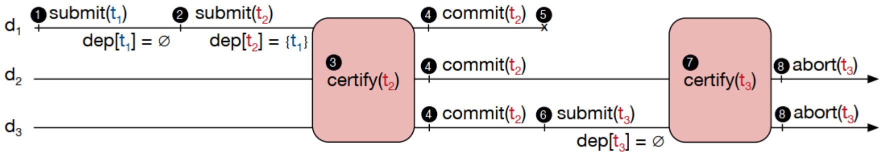
d_1 crashes before t_1 is replicated to d_3.



t_2 will never be visible at d_3.

Liveness of Strong Transactions

d_1 crashes before t_1 is replicated to d_3.



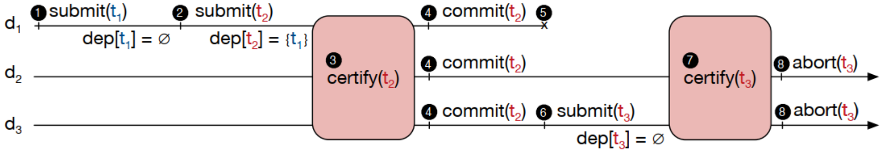
t_2 will never be visible at d_3.

No transaction t_3 conflicting with t_2 can commit.

Liveness of Strong Transactions

A strong transaction should wait for all its causal dependencies to become uniform before committing.

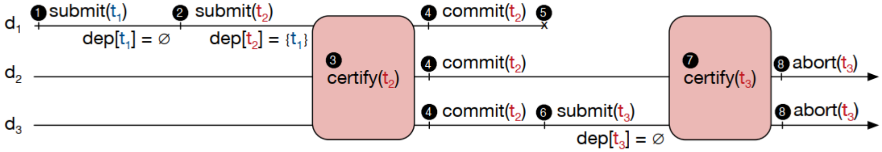
(eventually become visible at all correct data centers)



Liveness of Strong Transactions

A strong transaction should wait for all its causal dependencies to become uniform before committing.

(eventually become visible at all correct data centers)

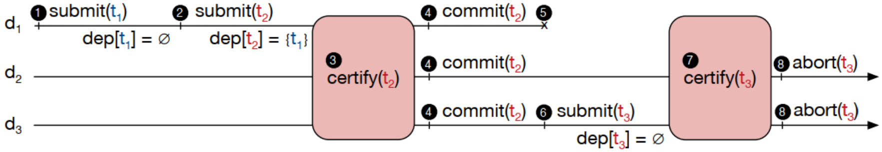


t_1 will eventually be visible at d_3.

Liveness of Strong Transactions

A strong transaction should wait for all its causal dependencies to become uniform before committing.

(eventually become visible at all correct data centers)



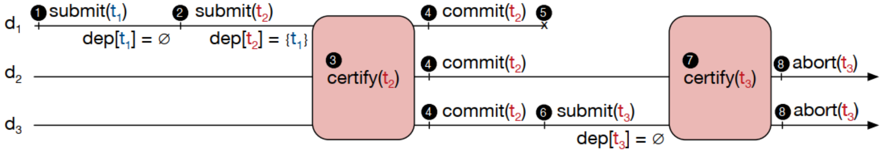
t_1 will eventually be visible at d_3.

t_2 will eventually be visible at d_3.

Liveness of Strong Transactions

A strong transaction should wait for all its causal dependencies to become uniform before committing.

(eventually become visible at all correct data centers)



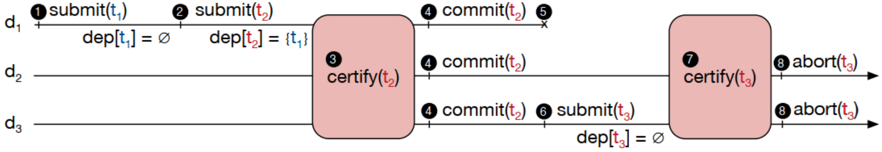
t_1 will eventually be visible at d_3.

t_2 will eventually be visible at d_3.

t_3 may be committed at d_3.

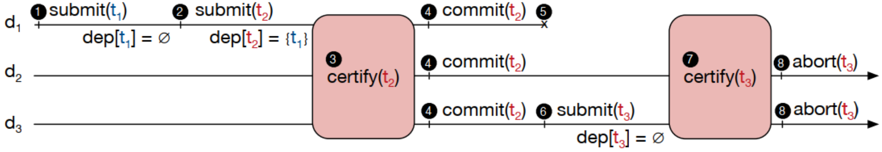
Minimizing the Latency of Strong Transactions

A strong transaction should wait for all its causal dependencies to become uniform before committing.



Minimizing the Latency of Strong Transactions

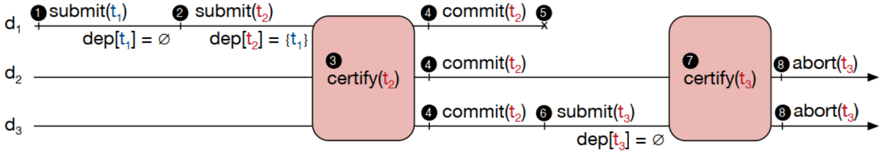
A strong transaction should wait for all its causal dependencies to become uniform before committing.



t_2 waits for local t_1 to become uniform before committing.

Minimizing the Latency of Strong Transactions

A strong transaction should wait for all its causal dependencies to become uniform before committing.

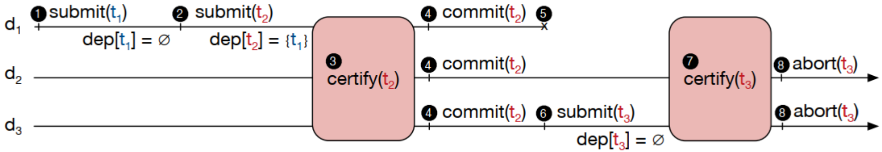


t_2 waits for local t_1 to become uniform before committing.

However, it may cost too much to wait for remote causal dependencies to become uniform.

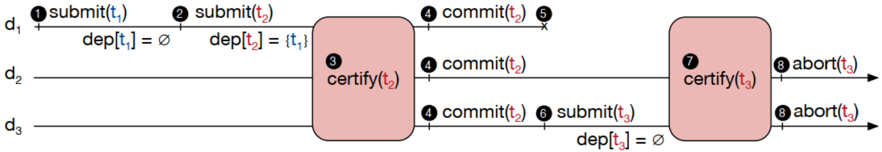
Minimizing the Latency of Strong Transactions

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



Minimizing the Latency of Strong Transactions

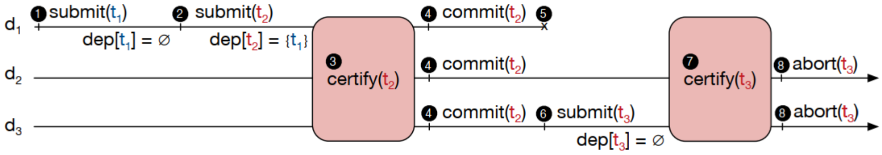
Let causal transactions be executed on an (almost) uniform snapshot that may be slightly in the past.



Make a remote causal transaction visible only after it is uniform.

Minimizing the Latency of Strong Transactions

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



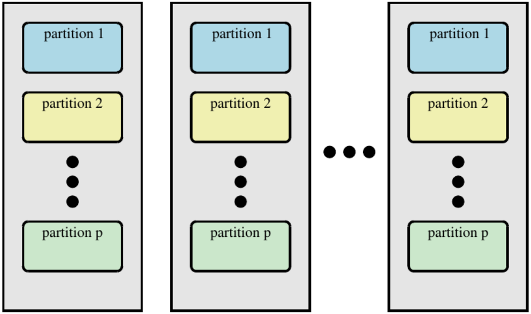
Make a remote causal transaction visible only after it is uniform.

A strong transaction does *not* need to wait for remote causal dependencies to become uniform before committing.

System Model and Notations

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

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



Fault-tolerant Causal Consistency Protocol

UNISTORE relies on various vector clocks $\in [\mathcal{D} \to \mathbb{N}]$ for

- Tracking causality ($commitVec$)
- Representing causally consistent snapshots ($snapshotVec$)
- Tracking what is replicated where

Tracking What is Replicated Where

Make a remote causal transaction visible only after it is uniform.

Tracking What is Replicated Where

Make a remote causal transaction visible 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.

Tracking What is Replicated Where

Make a remote causal transaction visible 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.

Tracking What is Replicated Where

Each replica maintains three vectors:

knownVec: track the set of transactions replicated at this replica

Tracking What is Replicated Where

Each replica maintains three vectors:

- knownVec: track the set of transactions replicated at this replica
- stableVec: track the set of transactions replicated at its local data center

Tracking What is Replicated Where

Each replica maintains three vectors:

- knownVec: track the set of transactions replicated at this replica
- stableVec: track the set of transactions replicated at its local data center
- uniformVec: track the set of transactions replicated at $f + 1$ data centers

Fault-tolerant Causal Consistency Protocol

- Causal transactions are executed on a uniform snapshot.
- Read-only transactions returns immediately.
- Update transactions commit using a variant of 2PC protocol.

Adding Strong Transactions

PoR ensures that conflicting transactions are executed serially.

Adding Strong Transactions

PoR ensures that conflicting transactions are executed serially.

Assigning to each strong transaction a scalar *strong timestamp*.

Adding Strong Transactions

PoR ensures that conflicting transactions are executed serially.

Assigning to each strong transaction a scalar *strong timestamp*.

$$[\mathcal{D} \to \mathbb{N}] \quad vs. \quad [\mathcal{D} \cup \{strong\} \to \mathbb{N}]$$

Adding Strong Transactions

PoR ensures that conflicting transactions are executed serially.

Assigning to each strong transaction a scalar *strong timestamp*.

$$[\mathcal{D} \to \mathbb{N}] \quad \textit{vs.} \quad [\mathcal{D} \cup \{\textit{strong}\} \to \mathbb{N}]$$

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

Adding Strong Transactions

Each replica maintains three vectors:

knownVec[*strong*]: track the set of strong transactions replicated at this replica

Adding Strong Transactions

Each replica maintains three vectors:

knownVec[*strong*]: track the set of strong transactions replicated at this replica

stableVec[*strong*]: track the set of strong transactions replicated at its local data center d

Adding Strong Transactions

Each replica maintains three vectors:

knownVec[*strong*]: track the set of strong transactions replicated at this replica

stableVec[*strong*]: track the set of strong transactions replicated at its local data center d

Adding Strong Transactions

Each replica maintains three vectors:

knownVec[*strong*]: track the set of strong transactions replicated at this replica

stableVec[*strong*]: track the set of strong transactions replicated at its local data center d

We do not extend uniformVec, since the commit protocol for strong transactions automatically guarantees their uniformity.

Adding Strong Transactions

- A strong transaction waits for local causal dependencies to become uniform before committing.

Adding Strong Transactions

- A strong transaction waits for local causal dependencies to become uniform before committing.
- Uses optimistic concurrency control protocol

Adding Strong Transactions

- A strong transaction waits for local causal dependencies to become uniform before committing.
- Uses optimistic concurrency control protocol
  - First *speculatively* execute strong transactions locally

Adding Strong Transactions

- A strong transaction waits for local causal dependencies to become uniform before committing.
- Uses optimistic concurrency control protocol
  - First *speculatively* execute strong transactions locally
  - Then certify them globally using a *transaction certification service* (TCS)

Strong Consistency Protocol: Commit

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

Proof of Correctness

We use the $(\mathit{vis}, \mathit{ar})$ formal specification framework.

A Framework for Transactional Consistency Models with Atomic Visibility

Andrea Cerone, Giovanni Bernardi, and Alexey Gotsman

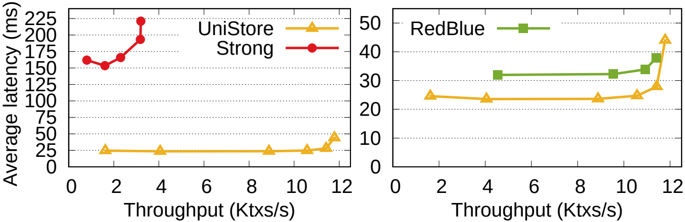
IMDEA Software Institute, Madrid, Spain

Proof of Correctness

The key is to construct the appropriate “visibility” (*vis*) relation and the “arbitration” (*ar*) relation between transactions such that

- The desired properties of various vector clocks hold.
- The liveness property holds (hard).

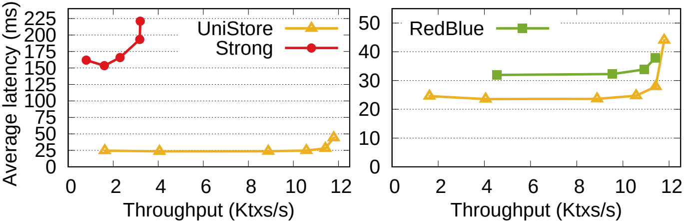
Q_1 : Does UniStore Combine Causal and Strong Consistency Effectively?



RUBiS benchmark: throughput vs. average latency.

Q_1 : Does UniStore Combine Causal and Strong Consistency Effectively?

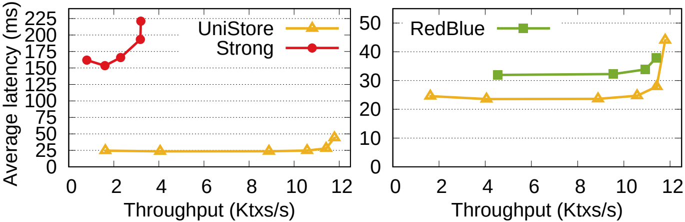
Throughput: 72% and 183% higher than RedBlue and Strong, 45% lower than Causal



RUBiS benchmark: throughput vs. average latency.

Q_1 : Does UniStore Combine Causal and Strong Consistency Effectively?

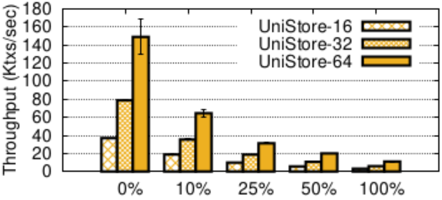
Throughput: 72% and 183% higher than RedBlue and Strong, 45% lower than Causal



RUBiS benchmark: throughput vs. average latency.

Average latency: 16.5ms of UniStore *vs.* 80.4ms of Strong, comparable to that of RedBlue

Q_2 : How does UniStore Scale with the Number of Machines?

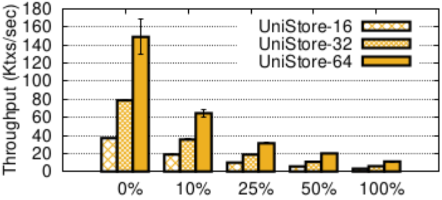


Scalability when varying the ratio of strong transactions with uniform data access (top) and under contention (bottom).

Q_2 : How does UniStore Scale with the Number of Machines?

Scales almost linearly (top):

9.76% throughput drop compared to the optimal scalability.

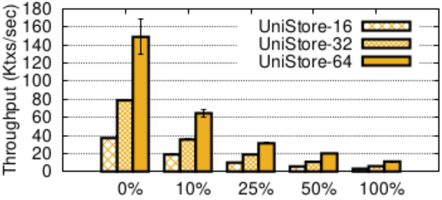


Scalability when varying the ratio of strong transactions with uniform data access (top) and under contention (bottom).

Q_2 : How does UniStore Scale with the Number of Machines?

Scales almost linearly (top):

9.76% throughput drop compared to the optimal scalability.



Scalability when varying the ratio of strong transactions with uniform data access (top) and under contention (bottom).

Under contention (bottom): 17.15% throughput drop

Q_3 : What is the Cost of Uniformity?

Throughput penalty of tracking uniformity

7.97% throughput drop on average

Q_3 : What is the Cost of Uniformity?

Throughput penalty of tracking uniformity

7.97% throughput drop on average

Update visibility delay (ms)

5ms $\sim$ 92ms at the 90^{th} percentile

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."

Hengfeng Wei (hfwei@nju.edu.cn)

- Attiya, Hagit, Soma Chaudhuri, Roy Friedman, and Jennifer L. Welch. "Shared Memory Consistency Conditions for Nonsequential Execution: Definitions and Programming Strategies". In: *SIAM Journal on Computing* 27.1 (1998), pp. 65–89.
- Attiya, Hagit and Roy Friedman. "A Correctness Condition for High-Performance Multiprocessors". In: *SIAM Journal on Computing* 27.6 (1998), pp. 1637–1670.
- Balegas, Valter, Sérgio Duarte, Carla Ferreira, Rodrigo Rodrigues, Nuno Preguiça, Mahsa Najafzadeh, and Marc Shapiro. "Putting Consistency Back into Eventual Consistency". In: *Proceedings of the Tenth European Conference on Computer Systems*. EuroSys '15. 2015.
- Bravo, Manuel, Alexey Gotsman, Borja de Régil, and Hengfeng Wei. "UniStore: A fault-tolerant marriage of causal and strong consistency". In: *2021 USENIX Annual Technical Conference (**USENIX ATC'2021**)*. USENIX Association, July 2021, pp. 923–937.
- Huang, Kaile, Yu Huang, and Hengfeng Wei. "Fine-Grained Analysis on Fast Implementations of Distributed Multi-Writer Atomic Registers". In: *Proceedings of the 39th Symposium on Principles of Distributed Computing (**PODC'2020**)*. Association for Computing Machinery, 2020, pp. 200–209. DOI: 10.1145/3382734.3405698.

Ji, Ye, Hengfeng Wei*, Yu Huang, and Jian Lu. "Specifying and verifying CRDT protocols using TLA$^+$". In: *Journal of Software (软件学报 **JOS'2020**)* 31.5 (2020), pp. 1332–1352. DOI: 10.13328/j.cnki.jos.005956.

Jiang, Xue, Hengfeng Wei*, and Yu Huang. "A Generic Specification Framework for Weakly Consistent Replicated Data Types". In: *International Symposium on Reliable Distributed Systems (**SRDS'2020**)*. 2020, pp. 143–154. DOI: 10.1109/SRDS51746.2020.00022.

Li, Cheng, Daniel Porto, Allen Clement, Johannes Gehrke, Nuno Preguiça, and Rodrigo Rodrigues. "Making Geo-replicated Systems Fast As Possible, Consistent when Necessary". In: *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*. OSDI '12. 2012, pp. 265–278.

Li, Cheng, Nuno Preguiça, and Rodrigo Rodrigues. "Fine-Grained Consistency for Geo-Replicated Systems". In: *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference*. USENIX ATC '18. 2018, pp. 359–371.

Wei, Hengfeng, Marzio De Biasi, Yu Huang, Jiannong Cao, and Jian Lu. "Verifying Pipelined-RAM consistency over read/write traces of data replicas". In: *IEEE Trans. Parallel Distrib. Syst. (**TPDS'2016**)* 27.5 (May 2016), pp. 1511–1523. DOI: 10.1109/TPDS.2015.2453985.

Wei, Hengfeng, Yu Huang, and Jian Lu. "Brief Announcement: Specification and Implementation of Replicated List: The Jupiter Protocol Revisited". In: *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*. PODC '18. ACM, 2018, pp. 81–83.

— ."Probabilistically-Atomic 2-Atomicity: enabling almost strong consistency in distributed storage systems". In: *IEEE Trans. Comput. (**TC'2017**)* 66.3 (Mar. 2017), pp. 502–514. DOI: 10.1109/TC.2016.2601322.

— ."Specification and Implementation of Replicated List: The Jupiter Protocol Revisited". In: *22nd International Conference on Principles of Distributed Systems, OPODIS 2018*. 2018, 12:1–12:16.