Parameterized and Runtime-tunable Snapshot Isolation in Distributed Transactional Key-value Stores

Hengfeng Wei, Yu Huang, Jian Lu

Nanjing University, China

September 19, 2017

RVSI: Relaxed Version Snapshot Isolation

1. Motivation for RVSI
2. Definition of RVSI
3. CHAMELEON Prototype and RVSI Protocol
4. Experimental Evaluation

RVSI: Relaxed Version Snapshot Isolation

Figure: Distributed key-value stores.

`put(K key, V val)`　　`get(K key)`

Transactional semantics

existential consistency atomic visibility example

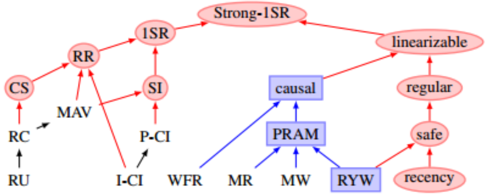


Figure: Transactional consistency models (from [Bailis@VLDB'14]).

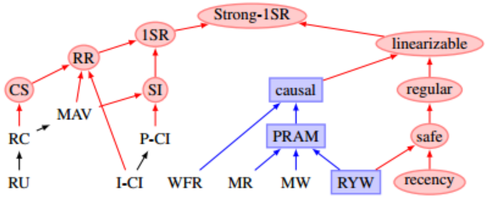


Figure: Transactional consistency models (from [Bailis@VLDB'14]).

Snapshot isolation (SI [Berenson@SIGMOD'95], [Adya@Thesis'99]):

- Read from the "latest" snapshot as of the time the transaction started
- No write-conflicting concurrent transactions

Reading the "latest" in a distributed setting
often requires intensive coordinations.

[1] GSI: Generalized Snapshot Isolation [Elnikety@SRDS'05]
[2] NMSI: Non-Monotonic Snapshot Isolation [Ardekani@SRDS'13]
[3] PL-FCV: Forward Consistent View [Aday@Thesis'99]
[4] PSI: Parallel Snapshot Isolation [Sovran@SOSP'11]

Reading the "latest" in a distributed setting
often requires intensive coordinations.

Relaxed variants of (distributed) SI:

- GSI [1]: allows to read from "older" snapshots
- NMSI [2]: allows to observe non-monotonically ordered snapshots
- PL-FCV [3]: allows a transaction to observe the updates of transactions that commit after it started
- PSI [4]: causal ordering of transactions across sites

[1] GSI: Generalized Snapshot Isolation [Elnikety@SRDS'05]
[2] NMSI: Non-Monotonic Snapshot Isolation [Ardekani@SRDS'13]
[3] PL-FCV: Forward Consistent View [Aday@Thesis'99]
[4] PSI: Parallel Snapshot Isolation [Sovran@SOSP'11]

Two possible drawbacks:

1. Unbounded inconsistency
    - no specification of the severity of the anomalies w.r.t SI

Two possible drawbacks:

1. Unbounded inconsistency
   - no specification of the severity of the anomalies w.r.t SI
2. Untunable at runtime
   - determined at the system design phase
   - remain unchanged once the system is deployed

An online bookstore application for motivating
"bounded inconsistency" and "runtime-tuable":

Title	Authors	Publisher	Sales	Inventory	Ratings	Reviews	$\cdots$

An online bookstore application for motivating
"bounded inconsistency" and "runtime-tuable":

Title	Authors	Publisher	Sales	Inventory	Ratings	Reviews	$\cdots$

Customer (T_1): Obtaining the basic info. about a book

- *out-of-date* reviews

An online bookstore application for motivating
"bounded inconsistency" and "runtime-tuable":

Title	Authors	Publisher	Sales	Inventory	Ratings	Reviews	$\cdots$

Customer (T_1): Obtaining the basic info. about a book

- *out-of-date* reviews

Bookstore Clerk (T_2): Checking the inventory of a book

- inventory updated by concurrent transactions committed *after* T_2 starts

An online bookstore application for motivating "bounded inconsistency" and "runtime-tuable":

Title	Authors	Publisher	Sales	Inventory	Ratings	Reviews	$\cdots$

Customer (T_1): Obtaining the basic info. about a book

- *out-of-date* reviews

Bookstore Clerk (T_2): Checking the inventory of a book

- inventory updated by concurrent transactions committed *after* T_2 starts

Sales Analyst (T_3): Studying sales vs. ratings of a book

- sales and ratings from *separate snapshots*

The idea of "parameterized and runtime-tunable snapshot isolation".

- RVSI: Relaxed Version Snapshot Isolation

  k_1-BV: k_1-version bounded *backward* view
  k_2-FV: k_2-version bounded *forward* view
  k_3-SV: k_3-version bounded *snapshot* view

[5] http://www.aliyun.com/

The idea of "parameterized and runtime-tunable snapshot isolation".

- RVSI: Relaxed Version Snapshot Isolation
  - k_1-BV: k_1-version bounded *backward* view
  - k_2-FV: k_2-version bounded *forward* view
  - k_3-SV: k_3-version bounded *snapshot* view
- CHAMELEON prototype: distributed transactional key-value store
  - achieves RVSI
  - allows each transaction to tune its consistency level at runtime

[5] `http://www.aliyun.com/`

The idea of "parameterized and runtime-tunable snapshot isolation".

- RVSI: Relaxed Version Snapshot Isolation

  k_1-BV: k_1-version bounded *backward* view
  k_2-FV: k_2-version bounded *forward* view
  k_3-SV: k_3-version bounded *snapshot* view

- CHAMELEON prototype: distributed transactional key-value store
  - achieves RVSI
  - allows each transaction to tune its consistency level at runtime
  - deployed on Aliyun [5]
  - explore the impacts of RVSI on the transaction abort rates

[5] http://www.aliyun.com/

RVSI: Relaxed Version Snapshot Isolation

Transaction T_i : $\quad s_i \quad (r_i/w_i)^+ \quad c_i/a_i$

s_i: start operation

r_i/w_i: read/write operation

c_i/a_i: commit/abort operation

Transaction T_i : $s_i \quad (r_i/w_i)^+ \quad c_i/a_i$

s_i: `start operation`

r_i/w_i: `read/write operation`

c_i/a_i: `commit/abort operation`

x_i: version i of data item x written by T_i

$r_i(x_j)$: transaction T_i reading x_j

$w_i(x_i)$: transaction T_i writing x_i

History: modelling an execution of a transactional key-value store

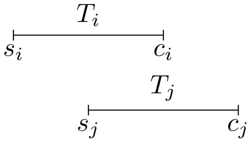
- *time-precedes partial order* $\prec_h$ over operations

History: modelling an execution of a transactional key-value store

- *time-precedes partial order* $\prec_h$ over operations

Two transactions are *concurrent* if

$$s_i \prec_h c_j \wedge s_j \prec_h c_i$$



A history h is in *snapshot isolation* iff it satisfies [Adya@Thesis'99]

Snapshot Read: All reads of transaction T_i occur at T_i's start time.

Snapshot Write: No concurrent committed transactions may write the same data item. (WCF: write-conflict freedom)

A history h is in *snapshot isolation* iff it satisfies [Adya@Thesis'99]

Snapshot Read: All reads of transaction T_i occur at T_i's start time.

$$\forall r_i(x_{j \neq i}), w_{k \neq j}(x_k), c_k \in h :$$
$$(c_j \in h \land c_j \prec_h s_i) \land (s_i \prec_h c_k \lor c_k \prec_h c_j).$$

Snapshot Write: No concurrent committed transactions may write the same data item. (WCF: write-conflict freedom)

$$\forall w_i(x_i), w_{j \neq i}(x_j) \in h \implies (c_i \prec_h s_j \lor c_j \prec_h s_i).$$

Principles of RVSI:

- Using parameters (k_1, k_2, k_3) to control the severity of the anomalies w.r.t SI

[6] RC: Read Committed isolation.

Principles of RVSI:

- Using parameters (k_1, k_2, k_3) to control the severity of the anomalies w.r.t SI
- RC [6] $\supset$ RVSI$(k_1, k_2, k_3) \supset$ SI
- RVSI$(\infty, \infty, \infty) =$ RC RVSI$(1, 0, *) =$ SI

[6] RC: Read Committed isolation.

Each transaction reads data from the "latest" snapshot as of the time the transaction started.

– The "Snapshot Read" property of SI

RVSI relaxes "Snapshot Read" in three ways:

Each transaction reads data from the "latest" snapshot as of the time the transaction started.

– The "Snapshot Read" property of SI

RVSI relaxes "Snapshot Read" in three ways:

k_1-BV (Backward View): "stale" data versions staleness $\leq k_1$

Each transaction reads data from the "latest" snapshot as of the time the transaction started.

– The "Snapshot Read" property of SI

RVSI relaxes "Snapshot Read" in three ways:

k_1-BV (Backward View): "stale" data versions — staleness $\leq k_1$

k_2-FV (Forward View): "concurrent" data versions — forward level $\leq k_2$

> *Each transaction reads data from the "latest" snapshot as of the time the transaction started.*
>
> *– The "Snapshot Read" property of SI*

RVSI relaxes "Snapshot Read" in three ways:

k_1-BV (Backward View): "stale" data versions — staleness $\leq k_1$

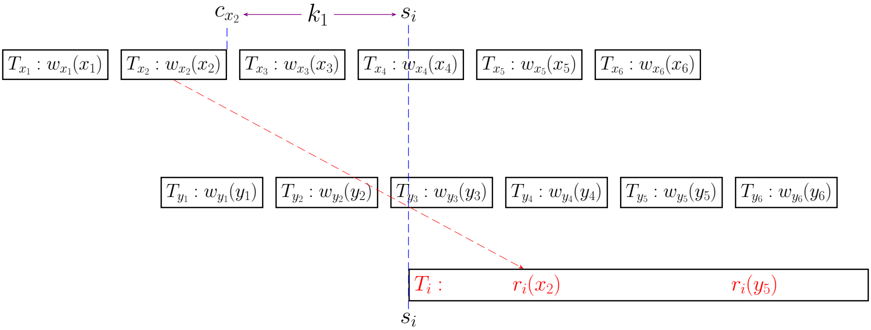
k_2-FV (Forward View): "concurrent" data versions — forward level $\leq k_2$

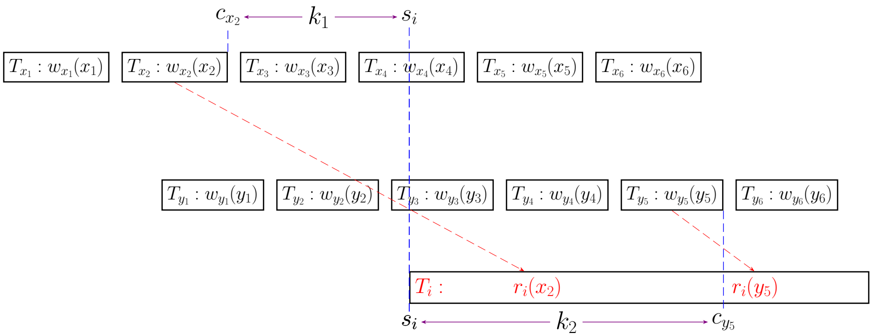
k_3-SV (Snapshot View): "non-snapshot" data versions — distance $\leq k_3$

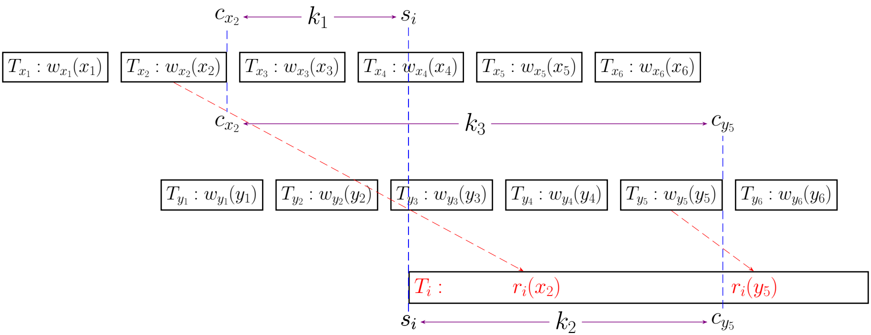
$\boxed{T_{x_1} : w_{x_1}(x_1)}$ $\boxed{T_{x_2} : w_{x_2}(x_2)}$ $\boxed{T_{x_3} : w_{x_3}(x_3)}$ $\boxed{T_{x_4} : w_{x_4}(x_4)}$ $\boxed{T_{x_5} : w_{x_5}(x_5)}$ $\boxed{T_{x_6} : w_{x_6}(x_6)}$

$\boxed{T_{y_1} : w_{y_1}(y_1)}$ $\boxed{T_{y_2} : w_{y_2}(y_2)}$ $\boxed{T_{y_3} : w_{y_3}(y_3)}$ $\boxed{T_{y_4} : w_{y_4}(y_4)}$ $\boxed{T_{y_5} : w_{y_5}(y_5)}$ $\boxed{T_{y_6} : w_{y_6}(y_6)}$

$\boxed{T_i : \quad r_i(x_2) \quad r_i(y_5)}$


c_{x_2}
k_1
s_i
$T_{x_1} : w_{x_1}(x_1)$
$T_{x_2} : w_{x_2}(x_2)$
$T_{x_3} : w_{x_3}(x_3)$
$T_{x_4} : w_{x_4}(x_4)$
$T_{x_5} : w_{x_5}(x_5)$
$T_{x_6} : w_{x_6}(x_6)$
$T_{y_1} : w_{y_1}(y_1)$
$T_{y_2} : w_{y_2}(y_2)$
$T_{y_3} : w_{y_3}(y_3)$
$T_{y_4} : w_{y_4}(y_4)$
$T_{y_5} : w_{y_5}(y_5)$
$T_{y_6} : w_{y_6}(y_6)$
$T_i :$
$r_i(x_2)$
$r_i(y_5)$
s_i


c_{x_2}
k_1
s_i
$T_{x_1} : w_{x_1}(x_1)$
$T_{x_2} : w_{x_2}(x_2)$
$T_{x_3} : w_{x_3}(x_3)$
$T_{x_4} : w_{x_4}(x_4)$
$T_{x_5} : w_{x_5}(x_5)$
$T_{x_6} : w_{x_6}(x_6)$
$T_{y_1} : w_{y_1}(y_1)$
$T_{y_2} : w_{y_2}(y_2)$
$T_{y_3} : w_{y_3}(y_3)$
$T_{y_4} : w_{y_4}(y_4)$
$T_{y_5} : w_{y_5}(y_5)$
$T_{y_6} : w_{y_6}(y_6)$
$T_i :$
$r_i(x_2)$
$r_i(y_5)$
s_i
k_2
c_{y_5}


c_{x_2} k_1 s_i
$T_{x_1} : w_{x_1}(x_1)$
$T_{x_2} : w_{x_2}(x_2)$
$T_{x_3} : w_{x_3}(x_3)$
$T_{x_4} : w_{x_4}(x_4)$
$T_{x_5} : w_{x_5}(x_5)$
$T_{x_6} : w_{x_6}(x_6)$
c_{x_2} k_3 c_{y_5}
$T_{y_1} : w_{y_1}(y_1)$
$T_{y_2} : w_{y_2}(y_2)$
$T_{y_3} : w_{y_3}(y_3)$
$T_{y_4} : w_{y_4}(y_4)$
$T_{y_5} : w_{y_5}(y_5)$
$T_{y_6} : w_{y_6}(y_6)$
$T_i :$ $r_i(x_2)$ $r_i(y_5)$
s_i k_2 c_{y_5}

(k_1-BV)

$$\forall r_i(x_j), w_k(x_k), c_k \in h : \left(c_j \in h \wedge \bigwedge_{k=1}^{m} (c_j \prec_h c_k \prec_h s_i) \right) \Rightarrow m < k_1$$

(k_2-FV)

$$\forall r_i(x_j), w_k(x_k), c_k \in h : \left(c_j \in h \wedge \bigwedge_{k=1}^{m} (s_i \prec_h c_k \prec_h c_j) \right) \Rightarrow m \leq k_2$$

(k_3-SV)

$$\forall r_i(x_j), r_i(y_l), w_k(x_k), c_k \in h : \left(\bigwedge_{k=1}^{m} (c_j \prec_h c_k \prec_h c_l) \right) \Rightarrow m \leq k_3.$$

$$h \in \mathsf{RVSI} \iff h \in k_1\text{-}\mathsf{BV} \cap k_2\text{-}\mathsf{FV} \cap k_3\text{-}\mathsf{SV} \cap \mathsf{WCF}$$

$$\mathsf{RVSI}(\infty, \infty, \infty) = \mathsf{RC} \qquad \mathsf{RVSI}(1, 0, *) = \mathsf{SI}$$

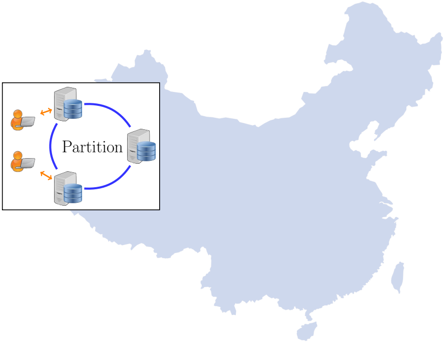
RVSI: Relaxed Version Snapshot Isolation

CHAMELEON prototype:

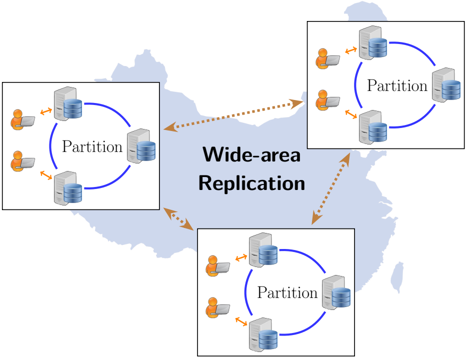
A prototype **partitioned replicated** distributed transactional **key-value** store

Classic **key-value** data model

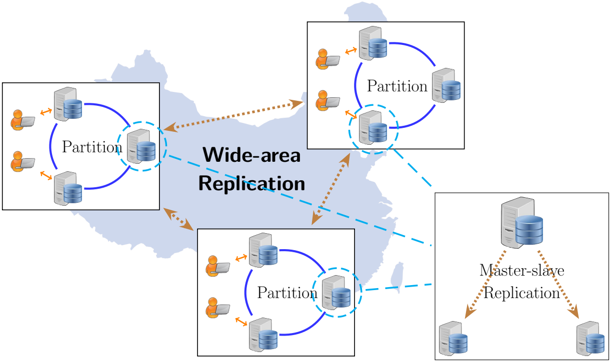
Key: (row key, column key)



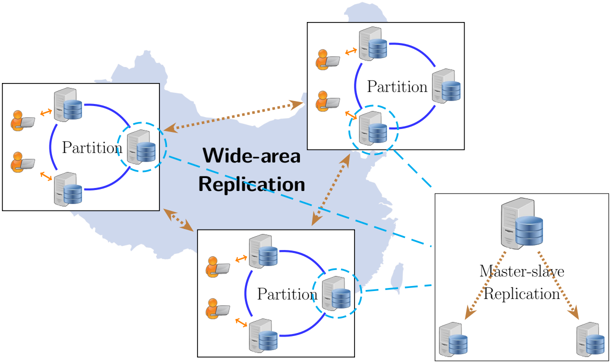
Keys are **partitioned** within a single datacenter.



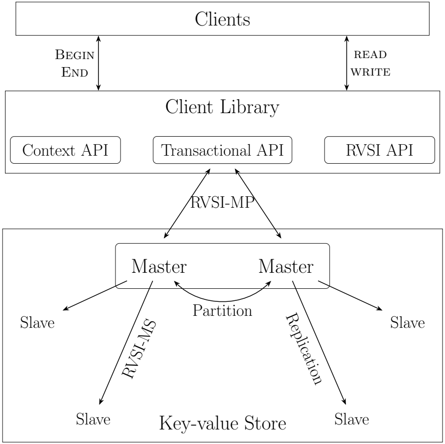
Each key is **replicated** across datacenters

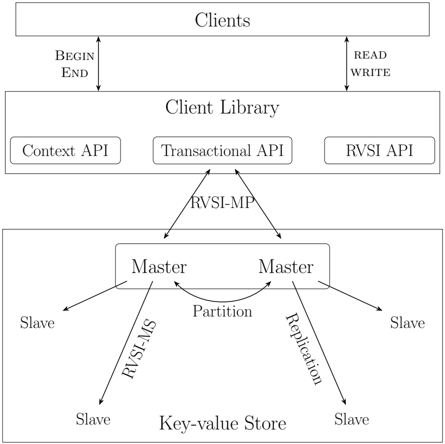


Each key is **replicated** across datacenters in a **master-slave** manner.

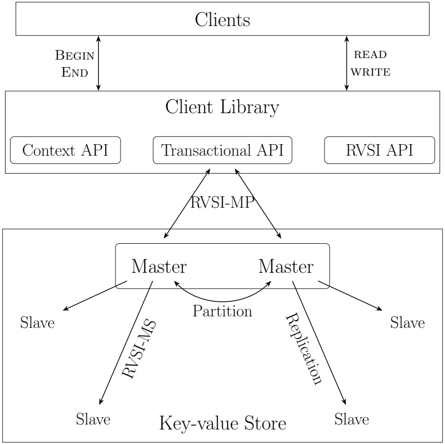


Transactions are first executed and committed on the **masters**, and are then asynchronously propagated to **slaves**.


Clients
Begin
End
read
write
Client Library
Context API
Transactional API
RVSI API
RVSI-MP
Master
Master
Partition
Slave
Slave
RVSI-MS
Replication
Slave
Slave
Key-value Store



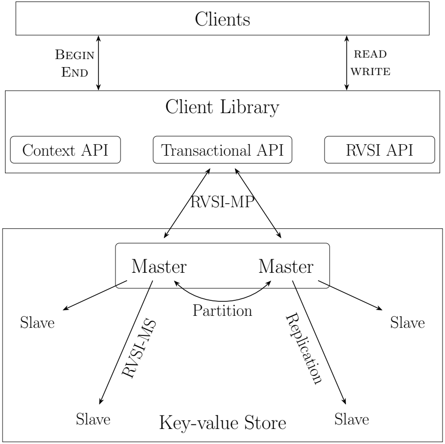
1. Partitioned replicated transactional key-value store



2. Client library

Code snippet for writing RVSI transactions:

```
// Initialize keys (ck1 and ck2) here

ITx tx = new RVSITx(/** context **/);

tx.begin();

// Read and write here

// Specify RVSI specs. (e.g., SVSpec)
RVSISpec sv = new SVSpec();
sv.addSpec({ck1, ck2}, 2);
tx.collectRVSISpec(sv);

boolean committed = tx.end();
```



3. RVSI protocol: RVSI-MS + RVSI-MP

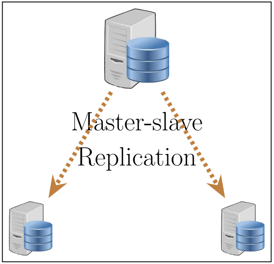
RVSI-MS: RVSI protocol for master-slave replication

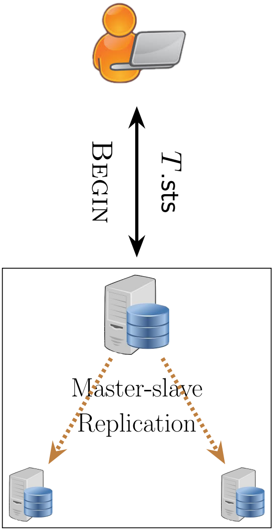
In terms of *event* generation and handling:

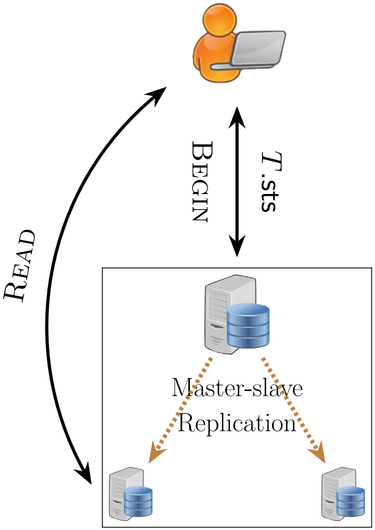
Clients: BEGIN, READ, WRITE, END

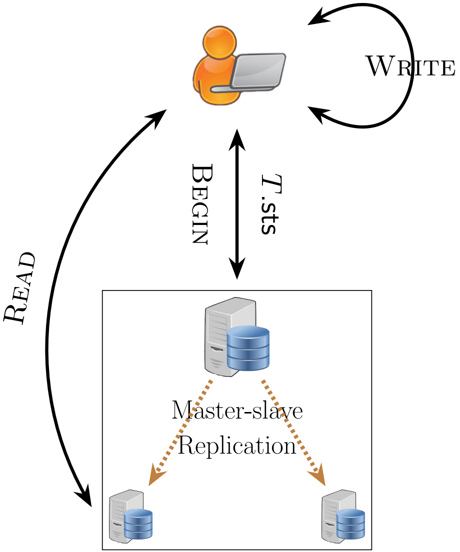
Master: START, COMMIT, SEND

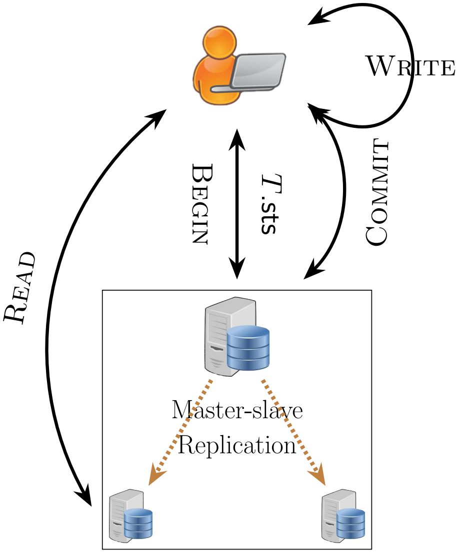
Slaves: RECEIVE

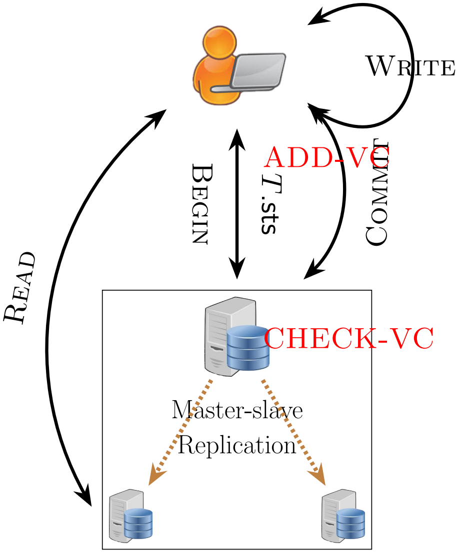

Master-slave
Replication


BEGIN
T.sts
Master-slave
Replication


BEGIN
T.sts
READ
Master-slave
Replication


WRITE
BEGIN
T.sts
READ
Master-slave
Replication


WRITE
BEGIN
T.sts
COMMIT
READ
Master-slave
Replication


WRITE
ADD-VC
BEGIN
T.sts
COMMIT
READ
CHECK-VC
Master-slave
Replication

Calculating version constraints for RVSI:

$$\mathcal{O}_x(t) = \# \text{ of versions of } x \text{ before time } t$$

$$r_i(x_j) \in T_i$$

k_1-BV:

$$\mathcal{O}_x(T_i.\textit{sts}) - \mathcal{O}_x(T_j.\textit{cts}) < k_1$$

k_2-FV:

$$\mathcal{O}_x(T_j.\textit{cts}) - \mathcal{O}_x(T_i.\textit{sts}) \leq k_2$$

Calculating version constraints for RVSI:

$$\mathcal{O}_x(t) = \#\text{ of versions of } x \text{ before time } t$$

$$r_i(x_j) \in T_i$$

k_1-BV:

$$\mathcal{O}_x(T_i.\textit{sts}) - \mathcal{O}_x(T_j.\textit{cts}) < k_1$$

k_2-FV:

$$\mathcal{O}_x(T_j.\textit{cts}) - \mathcal{O}_x(T_i.\textit{sts}) \leq k_2$$

k_3-SV:

$$r_i(x_j), r_i(y_l) \in T_i$$

$$\mathcal{O}_x(T_l.\textit{cts}) - \mathcal{O}_x(T_j.\textit{cts}) \leq k_3$$

Algorithm 1 RVSI-MS Protocol for Executing Transaction T (Client).

```
procedure BEGIN()
    T.sts ← rpc-call START() at master 𝓜
procedure READ(x)
    x.ver ← rpc-call READ(x) at any site
procedure WRITE(x, v)
    add (x, v) to T.writes
procedure END(T)
    T.vc ← ADD-VC()
    c/a ← rpc-call COMMIT(T.writes, T.vc) at 𝓜
```

Algorithm 1 RVSI-MS Protocol for Executing Transaction T (Master).

$\mathcal{M}.ts$: for start-timestamps and commit-timestamps
$\{x.\mathit{ver} = (x.\mathit{ts}, x.\mathit{ord}, x.\mathit{val})\}$: set of versions of x

procedure START()
    return $++\mathcal{M}.ts$
procedure READ(x)
    return the latest $x.\mathit{ver}$ installed
procedure COMMIT($T.\mathit{writes}, T.\mathit{vc}$)
    if CHECK-VC($T.\mathit{vc}$) && write-conflict freedom **then**
        $T.\mathit{cts} \leftarrow ++\mathcal{M}.ts$
        ▷ *apply $T.\mathit{writes}$ locally and propagate it*
        $T.\mathit{upvers} = \emptyset$ ▷ *collect updated versions*
        for $(x, v) \in T.\mathit{writes}$ **do**
            $x.\mathit{new\text{-}ver} \leftarrow (T.\mathit{cts}, ++x.\mathit{ord}, v)$
            add $x.\mathit{new\text{-}ver}$ to $\{x.\mathit{ver}\}$ and $T.\mathit{upvers}$
        broadcast $\langle$PROP$, T.\mathit{upvers}\rangle$ to slaves
        return c denoting "committed"
    return a denoting "aborted"

Algorithm 1 RVSI-MS Protocol for Executing Transaction T (Slave).

$x.\mathit{ver} = (x.\mathit{ts}, x.\mathit{ord}, x.\mathit{val})$: the latest version of x

```
procedure READ(x)
    return x.ver
upon RECEIVED(⟨PROP, T.upvers⟩)
    for (x.ver' = (x.ts', x.ord', x.val')) ∈ T.upvers do
        if x.ord' > x.ord then
            x.ver ← x.ver'
```

Distributed transactions spanning multiple masters need to be committed atomically.

Distributed transactions spanning multiple masters
need to be committed atomically.

Using the two-phase commit (2PC) protocol [Bernstein@Book'87].

Distributed transactions spanning multiple masters need to be committed atomically.

Using the two-phase commit (2PC) protocol [Bernstein@Book'87].

Two issues to address

Assumes a timestamp oracle [Peng@OSDI'10]:

Client: asks for the start-timestamp in BEGIN

Coordinator: asks for the commit-timestamp in COMMIT

Split the RVSI version constraints according to partitions:

$$r_i(x_j) \in T_i$$

k_1-BV:

$$\mathcal{O}_x(T_i.sts) - \mathcal{O}_x(T_j.cts) < k_1$$

k_2-FV:

$$\mathcal{O}_x(T_j.cts) - \mathcal{O}_x(T_i.sts) \leq k_2$$

k_3-SV:

$$r_i(x_j), r_i(y_l) \in T_i$$

$$\mathcal{O}_x(T_l.cts) - \mathcal{O}_x(T_j.cts) \leq k_3$$

All version constraints involve only one data item.

Algorithm 2 RVSI-MP for Executing Transaction T (Client).

1: **procedure** BEGIN()
2: **return rpc-call** GETTS() at $\mathcal{T}$
3: **procedure** END()
4: $T.vc \leftarrow$ ADD-VC()
5: $c/a \leftarrow$ **rpc-call** C-COMMIT($T.writes, T.vc$) at $\mathcal{C}$

Algorithm 2 RVSI-MP for Executing Transaction T (Timestamp Oracle).

```
𝒯.ts: for start-timestamps and commit-timestamps
procedure GETTS()
    return ++𝒯.ts
```

Algorithm 2 RVSI-MP for Executing Transaction T (Coordinator).

procedure C-COMMIT($T.writes, T.vc$)
    split $T.writes$ and $T.vc$ with the data partitioning strategy
    ▷ *the prepare phase:*
    rpc-call PREPARE($T.writes, T.vc$) at each $\mathcal{M}$
    ▷ *the commit phase:*
    if all PREPARE($T.writes, T.vc$) return *true* **then**
        $T.cts \leftarrow$ **rpc-call** GETTS() at $\mathcal{T}$
        rpc-call COMMIT($T.cts, T.writes$) at each $\mathcal{M}$
    else
        rpc-call ABORT() at each $\mathcal{M}$
        return a denoting “aborted”
    if all COMMIT($T.cts, T.writes$) return *true* **then**
        return c denoting “committed”
    else
        return a denoting “aborted”

Algorithm 2 RVSI-MP for Executing Transaction T (Master).

procedure PREPARE(T.*writes*, T.*vc*)
    return CHECK-VC(T.*vc*) && write-conflict freedom
procedure COMMIT(T.*cts*, T.*writes*)
    ▷ *apply* T.*writes locally and propagate it*
procedure ABORT()
    ▷ *abort*

RVSI: Relaxed Version Snapshot Isolation

Impacts of RVSI specification

on the *transaction abort rates* in various scenarios

Impacts of RVSI specification

on the *transaction abort rates* in various scenarios

Performance?

- Not done yet in this work
- CHAMELEON prototype is . . .

Transactions abort for two reasons:

- "vc-aborted": RVSI version constraints violated
- "wcf-aborted": the WCF property violated

Transactions abort for two reasons:

- "vc-aborted": RVSI version constraints violated
- "wcf-aborted": the WCF property violated

Transaction abort rates due to "vc-aborted" are *sensitive* to different values of k_1, k_2, or k_3,

Transactions abort for two reasons:

- "vc-aborted": RVSI version constraints violated
- "wcf-aborted": the WCF property violated

Transaction abort rates due to "vc-aborted" are *sensitive* to different values of k_1, k_2, or k_3, but those due to "wcf-aborted" are not.

Transactions abort for two reasons:

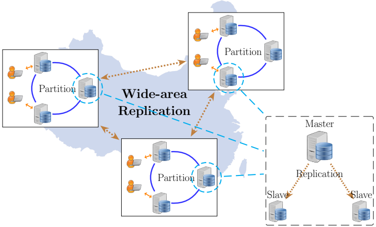
- "vc-aborted": RVSI version constraints violated
- "wcf-aborted": the WCF property violated

Transaction abort rates due to "vc-aborted" are *sensitive* to different values of k_1, k_2, or k_3, but those due to "wcf-aborted" are not.

$$h \in \text{RVSI} \iff h \in k_1\text{-BV} \cap k_2\text{-FV} \cap k_3\text{-SV} \cap \text{WCF}.$$

CHAMELEON prototype on Aliyun:

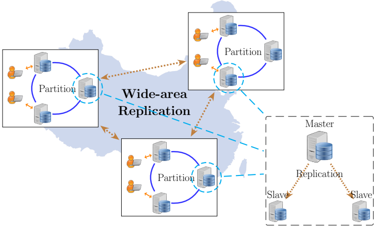
- 3 datacenters [1]
- 3 nodes in each datacenter
- Partition & Replication
- Clients in our lab [2]



[1]Located in East China, North China, and South China, respectively.
[2]Located in East China.
[3]https://github.com/hengxin/aliyun-ping-traces

CHAMELEON prototype on Aliyun:

- 3 datacenters [1]
- 3 nodes in each datacenter
- Partition & Replication
- Clients in our lab [2]



(One-way) delays among nodes [3]:

Within datacenter: $1 \sim 2$ms

Across datacenters: $15 \sim 25$ms

Clients to nodes: $15 \sim 20$ms

[1] Located in East China, North China, and South China, respectively.

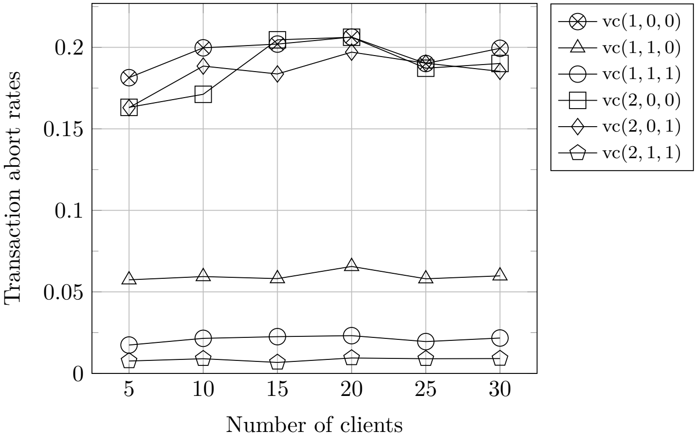
[2] Located in East China.

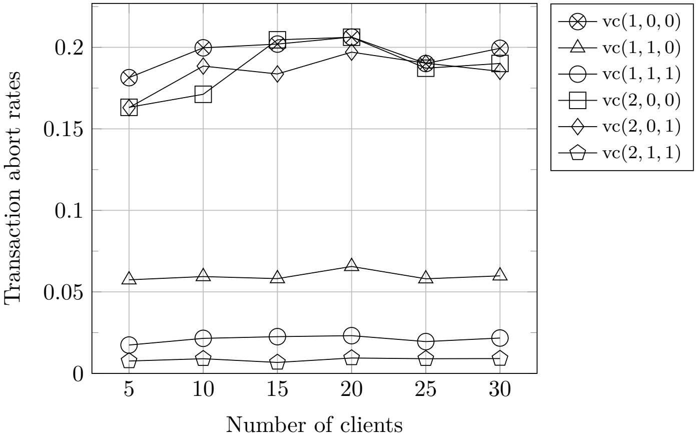
[3] https://github.com/hengxin/aliyun-ping-traces

Table: Three categories of workload parameters for experiments on Aliyun.

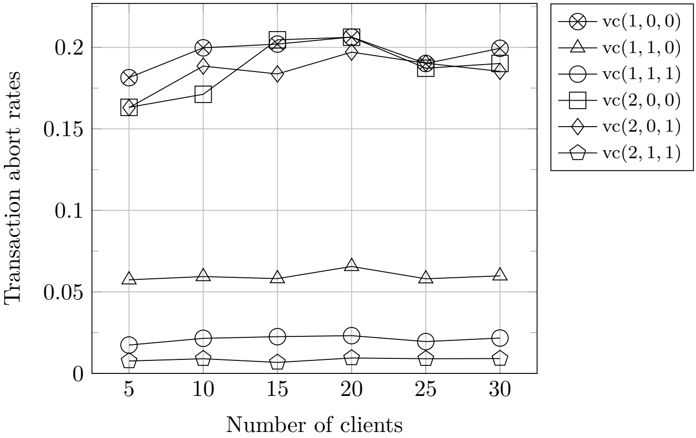
Parameter		**Value**	
Transaction-related	#keys	$25 = 5$ (rows) $\times$ 5 (columns)	
	#clients	5, 10, 15, 20, 25, 30	
	#txs/client	1000	
	#ops/tx	$\sim$ Binomial(20, 0.5)	n
	rwRatio	1:2, 1:1, 4:1	
	zipfExponent	1	
Execution-related	minInterval	0ms	
	maxInterval	10ms	
	meanInterval	5ms	
RVSI-related	(k_1, k_2, k_3)	(1,0,0) (1,1,0) (1,1,1) (2,0,0) (2,0,1) (2,1,1)	

under read-frequent workloads


Transaction abort rates
0.2
0.15
0.1
0.05
0
5
10
15
20
25
30
Number of clients
vc(1, 0, 0)
vc(1, 1, 0)
vc(1, 1, 1)
vc(2, 0, 0)
vc(2, 0, 1)
vc(2, 1, 1)

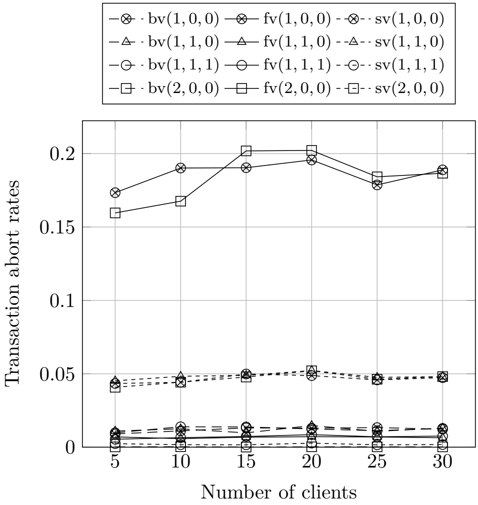


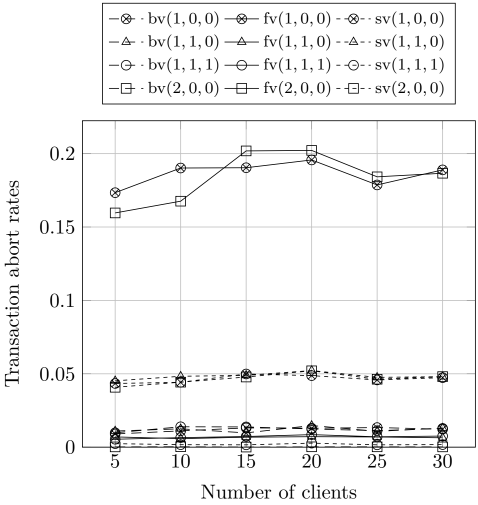
The transaction abort rates due to “vc-aborted”



The transaction abort rates due to "vc-aborted" can be greatly reduced by slightly increasing the values of k_1, k_2, or k_3:

$$vc(1,0,0) = 0.1994 \implies vc(2,1,1) = 0.0091 \quad (\text{\#clients} = 30)$$


bv(1, 0, 0)
fv(1, 0, 0)
sv(1, 0, 0)
bv(1, 1, 0)
fv(1, 1, 0)
sv(1, 1, 0)
bv(1, 1, 1)
fv(1, 1, 1)
sv(1, 1, 1)
bv(2, 0, 0)
fv(2, 0, 0)
sv(2, 0, 0)
Transaction abort rates
0
0.05
0.1
0.15
0.2
5
10
15
20
25
30
Number of clients



Most "vc-aborted" transactions abort because of violating k_2-FV.

$$fv(1,0,0) = 0.1889 \implies fv(2,0,0) = 0.1866 \implies fv(1,1,0) = 0.0064$$

Question: when does k_1 for k_1-BV take effect?

It seems that k_1-BV has *little* impact on the transaction abort rates.

Question: when does k_1 for k_1-BV take effect?

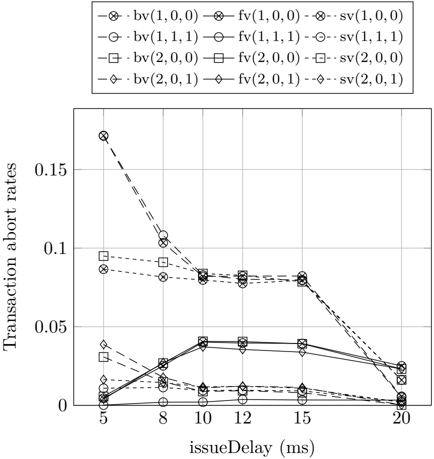
It seems that k_1-BV has *little* impact on the transaction abort rates.

It may be the case in the Aliyun scenarios.

What about other scenarios?

Three types of delays for controlled experiments on local hosts.

Types	Values (ms)	Explanation
issueDelay	5, 8, 10, 12, 15, 20	delays between clients and replicas
replDelay	5, 10, 15, 20, 30	delays between masters and slaves
2pcDelay	10, 20, 30, 40, 50	delays among masters



When the "issueDelay" gets shorter,
the impacts of k_2-FV go weaker,
and the impacts of k_1-BV have begun to emerge.

issueDelay $=$ 20ms: $bv(1,0,0) = 0.0057$ $\quad fv(1,0,0) = 0.0251$

issueDelay $=$ 15ms: $bv(1,0,0) = 0.08225$ $\quad fv(1,0,0) = 0.0393$

issueDelay $=$ 5ms: $bv(1,0,0) = 0.1716$ $\quad fv(1,0,0) = 0.0045$

issueDelay $=$ 20ms: $bv(1,0,0) = 0.0057$ $\quad fv(1,0,0) = 0.0251$

issueDelay $=$ 15ms: $bv(1,0,0) = 0.08225$ $\quad fv(1,0,0) = 0.0393$

issueDelay $=$ 5ms: $bv(1,0,0) = 0.1716$ $\quad fv(1,0,0) = 0.0045$

larger issueDelay $\Longrightarrow$ longer transaction

more concurrent transactions

more likely to obtain data versions updated by concurrent transactions

more sensitive to k_2-FV

Generally, RVSI helps to reduce the transaction abort rates when applications are willing to tolerate certain anomalies.

Generally, RVSI helps to reduce the transaction abort rates when applications are willing to tolerate certain anomalies.

k_2-FV: In the Aliyun scenarios, most transactions have been aborted because of violating k_2-FV.

Generally, RVSI helps to reduce the transaction abort rates when applications are willing to tolerate certain anomalies.

k_2-FV: In the Aliyun scenarios, most transactions have been aborted because of violating k_2-FV.

k_1-BV: In controlled experiments, the impacts of k_1-BV emerge when the issueDelay gets shorter.

Generally, RVSI helps to reduce the transaction abort rates when applications are willing to tolerate certain anomalies.

- k_2-FV: In the Aliyun scenarios, most transactions have been aborted because of violating k_2-FV.
- k_1-BV: In controlled experiments, the impacts of k_1-BV emerge when the issueDelay gets shorter.
- k_3-SV: Complex and challenging (involving multiple data items)