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

Hengfeng Wei, Yu Huang, Jian Lu

Nanjing University, China

September 16, 2017

RVSI: Relaxed Version Snapshot Isolation

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

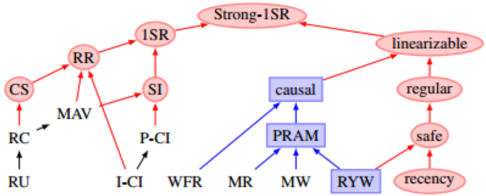


Figure: Transactional consistency models (from [Bailis@VLDB'14]).

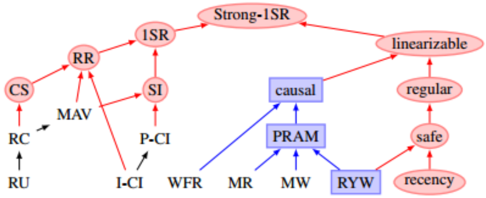


Figure: Transactional consistency models (from [Bailis@VLDB'14]).

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	···

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

Principle of RVSI

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

[6] RC: Read Committed

Principle of RVSI

- 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

Principle of RVSI

. . .

– *"Snapshot Read" property of SI*

1. "stale" data versions

Principle of RVSI

. . .

– *"Snapshot Read" property of SI*

1. "stale" data versions
2. "concurrent" data versions

Principle of RVSI

. . .

– "Snapshot Read" property of SI

1. "stale" data versions
2. "concurrent" data versions
3. "non-snapshot" data versions

Principle of RVSI

. . .

– *"Snapshot Read" property of SI*

1. "stale" data versions bounded staleness
2. "concurrent" data versions
3. "non-snapshot" data versions

Principle of RVSI

. . .

– *"Snapshot Read" property of SI*

1. "stale" data versions — bounded staleness
2. "concurrent" data versions — bounded concurrency level
3. "non-snapshot" data versions

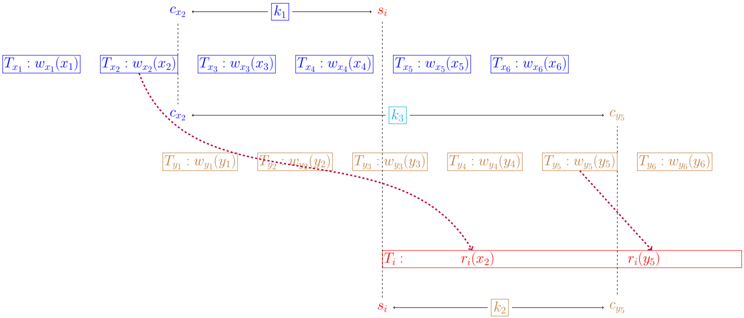
Principle of RVSI

. . .

– *"Snapshot Read" property of SI*

1. "stale" data versions — bounded staleness
2. "concurrent" data versions — bounded concurrency level
3. "non-snapshot" data versions — bounded distance

Illustration of RVSI



Definition of RVSI

(k_1-BV)

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

(k_2-FV)

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

(k_3-SV)

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

Definition of RVSI

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

Theorem

$$\text{RVSI}(1, 0, *) = \text{SI}.$$

RVSI: Relaxed Version Snapshot Isolation

Chameleon Protocol

Chameleon:

A prototype **partitioned replicated**

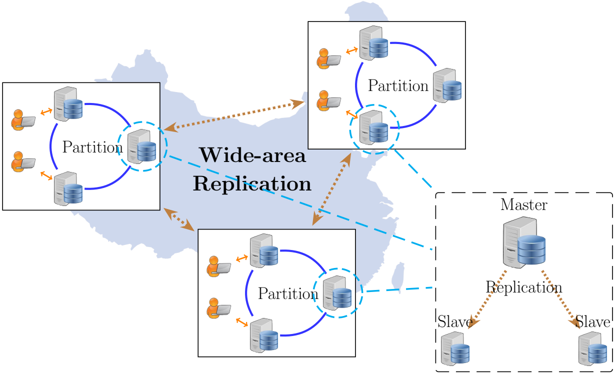
distributed transactional **key-value** store

Chameleon Prototype

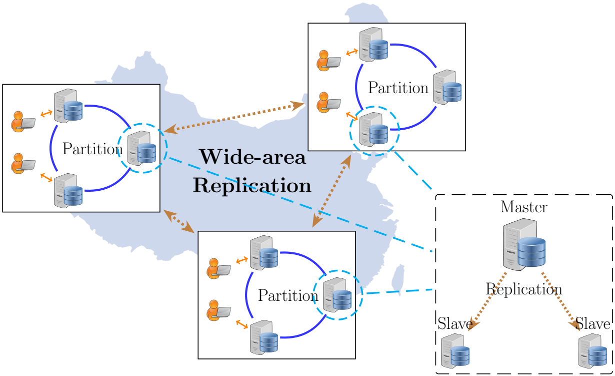
Classic **key-value** data model

Key: (row key, column key)

Chameleon Prototype

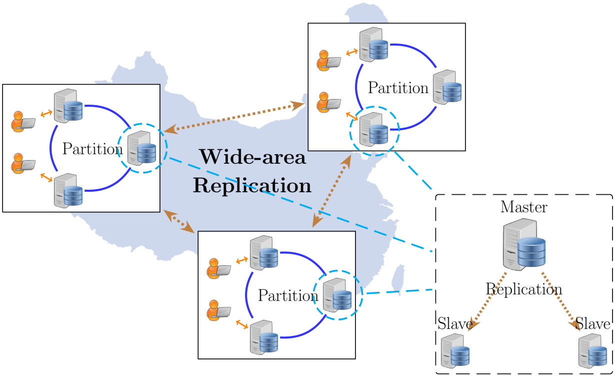


Chameleon Prototype



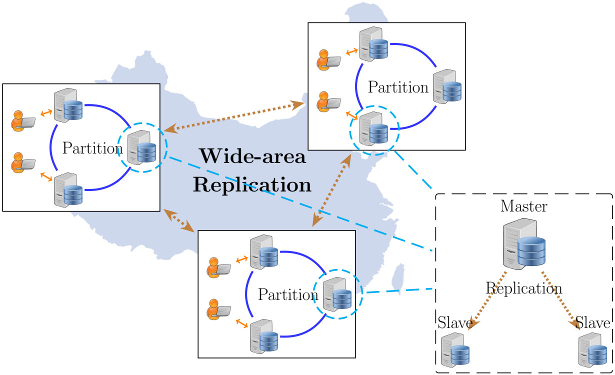
Keys are **partitioned** within a single datacenter.

Chameleon Prototype



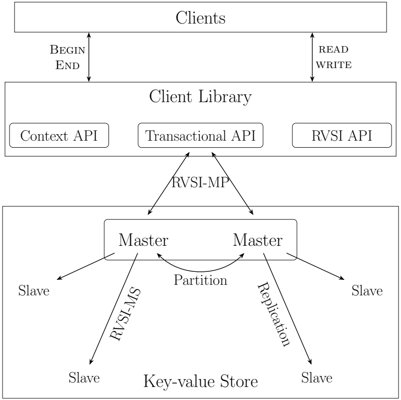
Each key is **replicated** in a master-slave manner across datacenters.

CHAMELEON Prototype

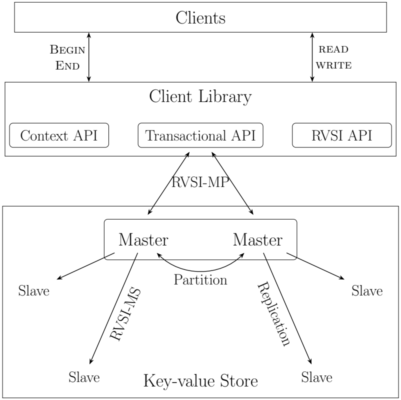


Transactions are first executed and committed on the masters, and are then asynchronously propagated to slaves.

Chameleon Protocol

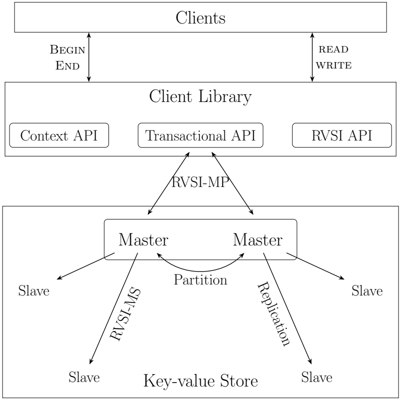


Chameleon Protocol



Partitioned replicated transactional key-value store

Chameleon Protocol



Client library

Chameleon Protocol

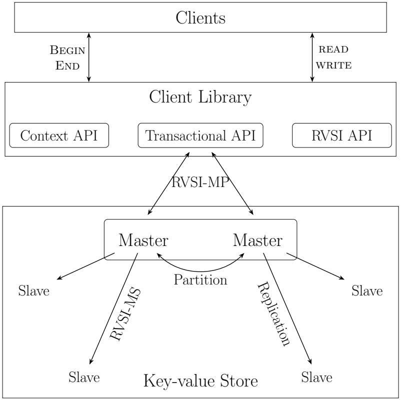
```
// Initialize keys (ck, ck1, and ck2) here
ITx tx = new RVSITx(/** context **/);

tx.begin();

// Read and write
ITsCell tsCell = tx.read(ck);
ITsCell tsCell1 = tx.read(ck1);
tx.write(ck1, new Cell("R1C1"));
ITsCell tsCell2 = tx.read(ck2);

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

CHAMELEON Protocol



RVSI protocol: RVSI-MS + RVSI-MP

RVSI-MS Protocol

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

RVSI-MS Protocol

TikZ overlay for RVSI-MS

RVSI-MS Protocol

Calculating version constraints:

RVSI-MP Protocol

Distributed transactions spanning multiple masters need to be committed atomically.

RVSI-MP Protocol

Distributed transactions spanning multiple masters need to be committed atomically.

Using the two-phase commit (2PC) protocol.

RVSI-MP Protocol

Assumes a timestamp oracle:

Clients:

Masters:

RVSI-MP Protocol

RVSI version constraints in 2PC protocol:

k_1-BV:

k_2-FV:

k_3-SV:

RVSI-MS Protocol

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

Client-side methods:

procedure BEGIN()
    T.*sts* ← **rpc-call** START() at master $\mathcal{M}$
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 $\mathcal{M}$

RVSI-MP Protocol

Algorithm 2 RVSI-MP: RVSI Protocol for Partition (for Executing Transaction T).

Client-side methods:

procedure $\textsc{Begin}()$
    return rpc-call $\textsc{getTS}()$ at $\mathcal{T}$
procedure $\textsc{End}()$
    $T.vc \leftarrow \textsc{add-vc}()$
    $c/a \leftarrow$ **rpc-call** $\textsc{c-commit}(T.writes, T.vc)$ at $\mathcal{C}$

Timestamp oracle methods:

$\mathcal{T}.ts$: for start-timestamps and commit-timestamps
procedure $\textsc{getTS}()$
    return $++\mathcal{T}.ts$

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:

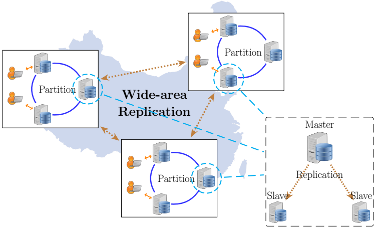
- "vc-aborted": RVSI version constraints violated
- "wcf-aborted": the WCF property violated

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:

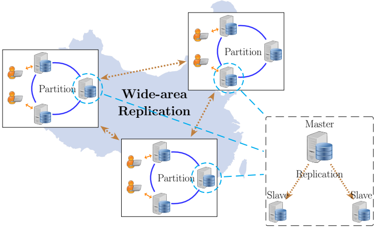
- 3 datacenters [1]
- 3 nodes in each datacenter
- Partition & Replication
- Clients in our lab [2]



[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.

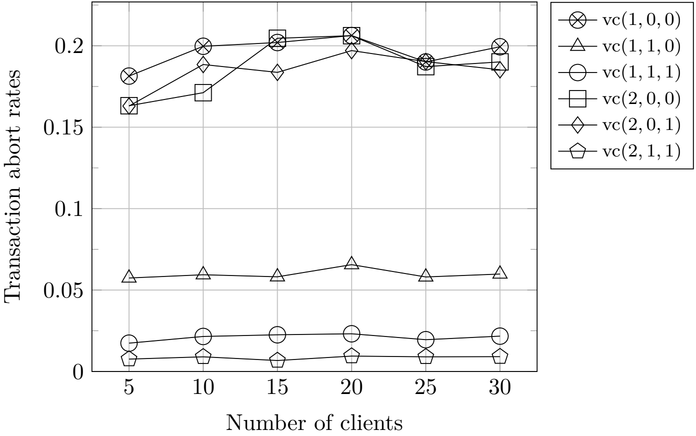
[2] Located in East China.

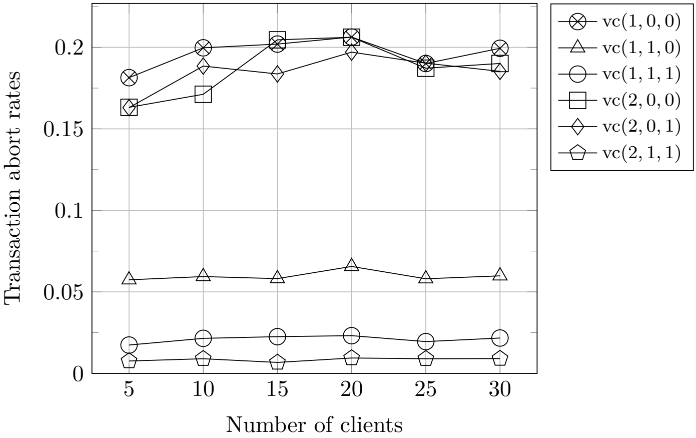
[3] https://github.com/hengxin/aliyun-ping-traces

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

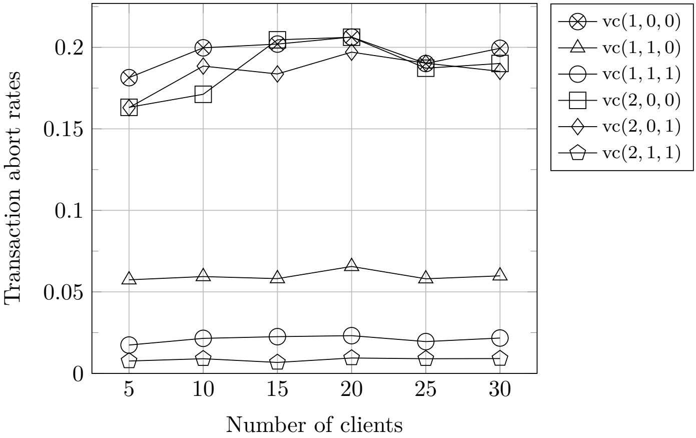
Parameter		**Value**	
Transaction-related	#keys	$25 = 5$ (rows) $\times$ 5 (columns)	
	#clients	5, 10, 15, 20, 25, 30	
	#txs/client	1000	
	#ops/tx	$\sim$ Binomial(20, 0.5)	n
	rwRatio	1:2, 1:1, 4:1	
	zipfExponent	1	
Execution-related	minInterval	0ms	
	maxInterval	10ms	
	meanInterval	5ms	
RVSI-related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

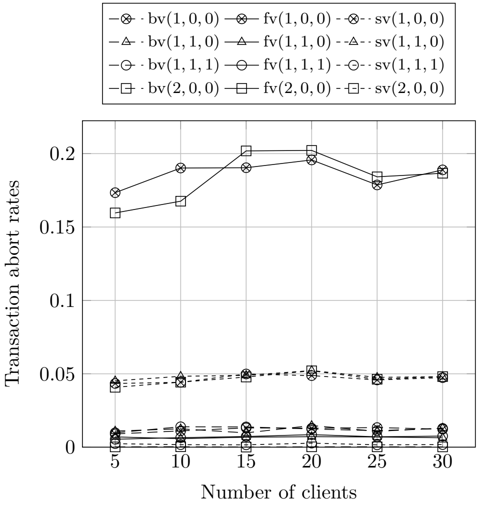


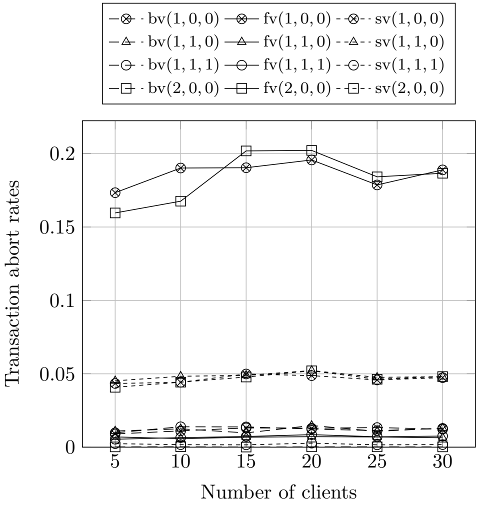
The transaction abort rates due to “vc-aborted”



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
0.2
0.15
0.1
0.05
0
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?

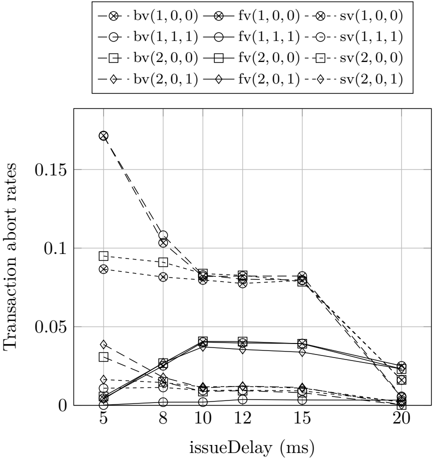
It seems that k_1-BV has *little* impact on the transaction abort rates.

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

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

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

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

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

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

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)