

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

**Hengfeng Wei**, Yu Huang, Jian Lu

Nanjing University, China

September 16, 2017



# Parameterized and Runtime-tunable Snapshot Isolation

## RVSI: Relaxed Version Snapshot Isolation

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

# Parameterized and Runtime-tunable Snapshot Isolation

## RVSI: Relaxed Version Snapshot Isolation

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

# Distributed key-value stores

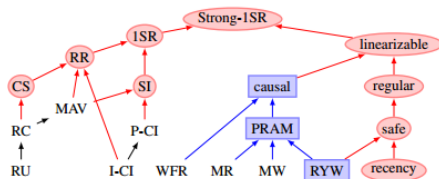


`put(K key, V val)`      `get(K key)`

# Transactional semantics

existential consistency atomic visibility example

# Transactional consistency models



Snapshot isolation (SI):

- ▶ **Read** from the latest consistent snapshot of all data items
- ▶ No **write** conflicts among concurrent transactions

# Distributed SI

Costs of SI

Relaxed variants of SI:

GSI <sup>1</sup>

NMSI <sup>2</sup>

PL-FCV <sup>3</sup>

PSI <sup>4</sup>

---

<sup>1</sup>GSI: Generalized Snapshot Isolation

<sup>2</sup>NMSI: Non-Monotonic Snapshot Isolation

<sup>3</sup>PL-FCV: Forward Consistent View

<sup>4</sup>PSI: Parallel Snapshot Isolation

# Two 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



# A Motivating Example

The *Books* table.

Title	Authors	Publisher	Sales	Inventory	Ratings	Reviews	...
-------	---------	-----------	-------	-----------	---------	---------	-----

Customer:

Bookstore Clerk:

Sales Analyst:

# Contributions



# Parameterized and Runtime-tunable Snapshot Isolation

## RVSI: Relaxed Version Snapshot Isolation

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

# Principle of RVSI

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

# Principle of RVSI

- ▶ Parameters  $(k_1, k_2, k_3)$  to control the severity of the anomalies w.r.t SI
- ▶  $RC^5 \supset RVSI(k_1, k_2, k_3) \supset SI$
- ▶  $RVSI(\infty, \infty, \infty) = RC$        $RVSI(1, 0, *) = SI$

---

<sup>5</sup>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
2. “concurrent” data versions
3. “non-snapshot” data versions

bounded staleness

# 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

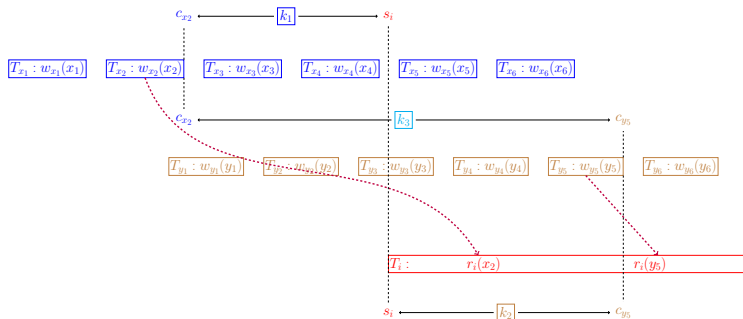
# Principle of RVSI

...

– “*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

$(k_1\text{-BV})$

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

$(k_2\text{-FV})$

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

$(k_3\text{-SV})$

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

# 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

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







# Parameterized and Runtime-tunable Snapshot Isolation

## RVSI: Relaxed Version Snapshot Isolation

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

# CHAMELEON Protocol

CHAMELEON:

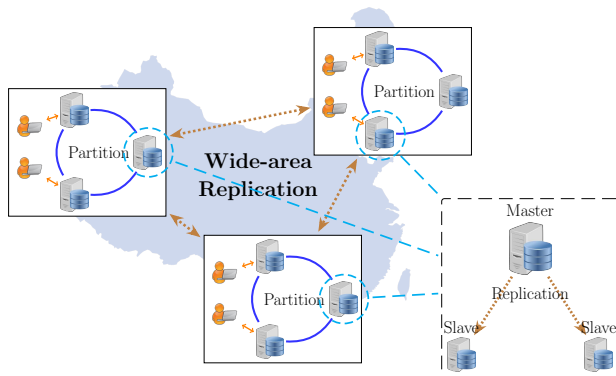
A prototype **partitioned replicated**  
distributed transactional **key-value** store

# CHAMELEON Prototype

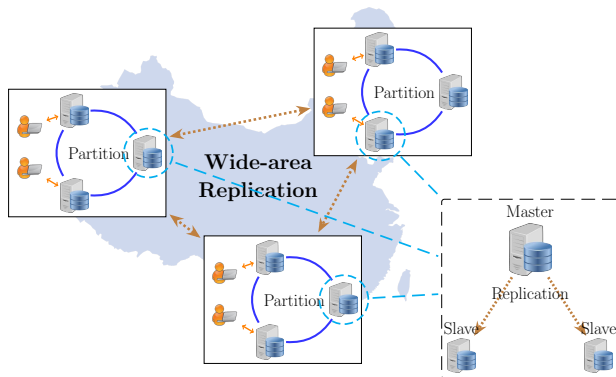
Classic **key-value** data model

Key: (row key, column key)

# CHAMELEON Prototype

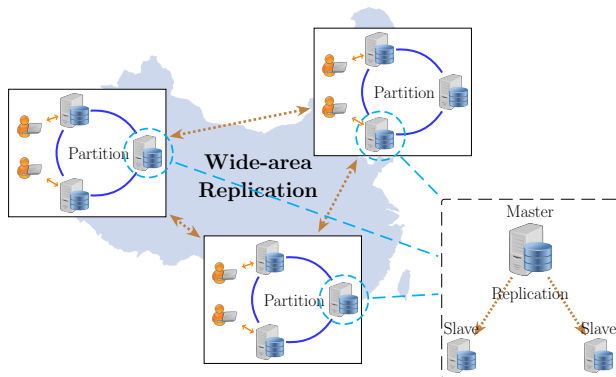


# CHAMELEON Prototype



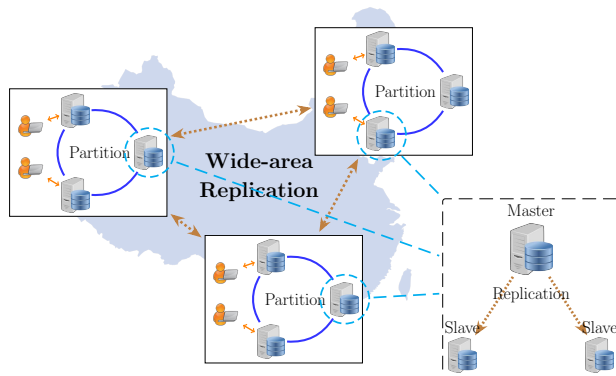
Keys are **partitioned** within a single datacenter.

# CHAMELEON Prototype



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

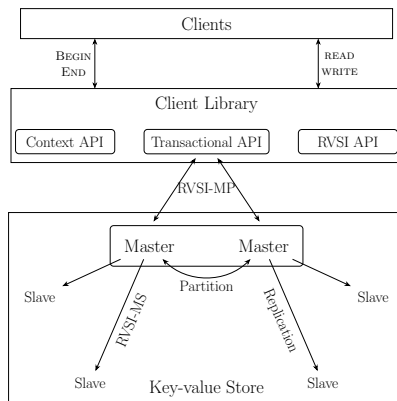
# CHAMELEON Prototype



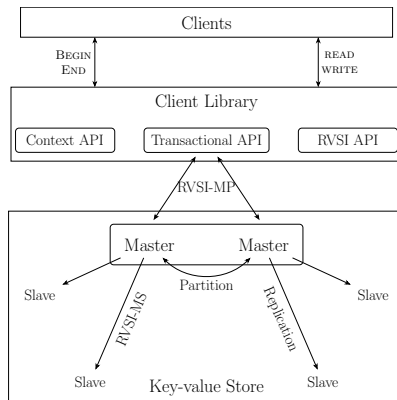
Transactions are first executed and committed on the masters, and are then asynchronously propagated to slaves.



# CHAMELEON Protocol

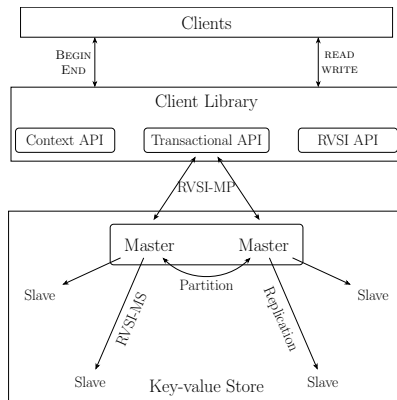


# CHAMELEON Protocol



Partitioned replicated transactional key-value store

# CHAMELEON Protocol



Client library

# CHAMELEON Protocol

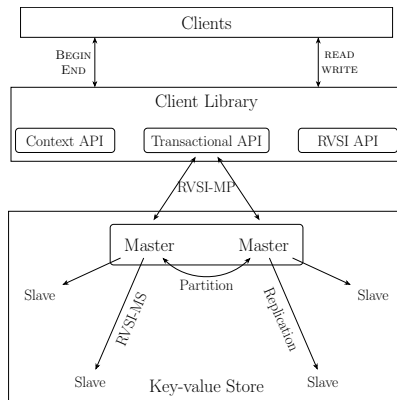
```
// Initialize keys (ck, ck1, and ck2) here
ITx tx = new RVSI Tx(** context **);

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)
RVSI Spec sv = new SVSpec();
sv.addSpec({ck, ck1, ck2}, 2);
tx.collectRVSI Spec(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:***

- 1: **procedure** BEGIN()
- 2:      $T.sts \leftarrow$  **rpc-call** START() at master  $\mathcal{M}$
- 3: **procedure** READ( $x$ )
- 4:      $x.ver \leftarrow$  **rpc-call** READ( $x$ ) at any site
- 5: **procedure** WRITE( $x, v$ )
- 6:     add  $(x, v)$  to  $T.writes$
- 7: **procedure** END( $T$ )
- 8:      $T.vc \leftarrow$  ADD-VC()
- 9:      $c/a \leftarrow$  **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:***

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

---

***Timestamp oracle methods:***

$\mathcal{T}.ts$ : for start-timestamps and commit-timestamps

- 6: **procedure** GETTS()
- 7:     **return**  $++\mathcal{T}.ts$



# Parameterized and Runtime-tunable Snapshot Isolation

## RVSI: Relaxed Version Snapshot Isolation

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



## 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:

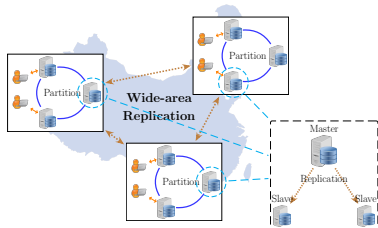
- ▶ “vc-aborted”: RVSI version constraints violated
- ▶ “wcf-aborted”: the WCF property violated

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 <sup>1</sup>
- ▶ 3 nodes in each datacenter
- ▶ Partition & Replication
- ▶ Clients in our lab <sup>2</sup>



<sup>1</sup> Located in East China, North China, and South China, respectively.

<sup>2</sup> Located in East China.

<sup>3</sup> <https://github.com/hengxin/aliyun-ping-traces>

## CHAMELEON prototype on Aliyun:

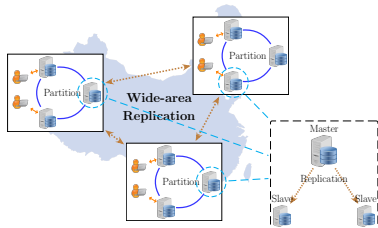
- ▶ 3 datacenters <sup>1</sup>
- ▶ 3 nodes in each datacenter
- ▶ Partition & Replication
- ▶ Clients in our lab <sup>2</sup>

(One-way) delays among nodes <sup>3</sup>:

Within datacenter: 1 ~ 2ms

Across datacenters: 15 ~ 25ms

Clients to nodes: 15 ~ 20ms



<sup>1</sup>Located in East China, North China, and South China, respectively.

<sup>2</sup>Located in East China.

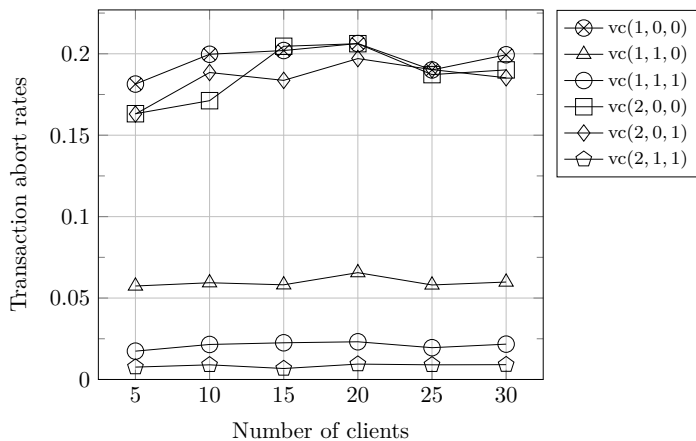
<sup>3</sup><https://github.com/hengxin/aliyun-ping-traces>

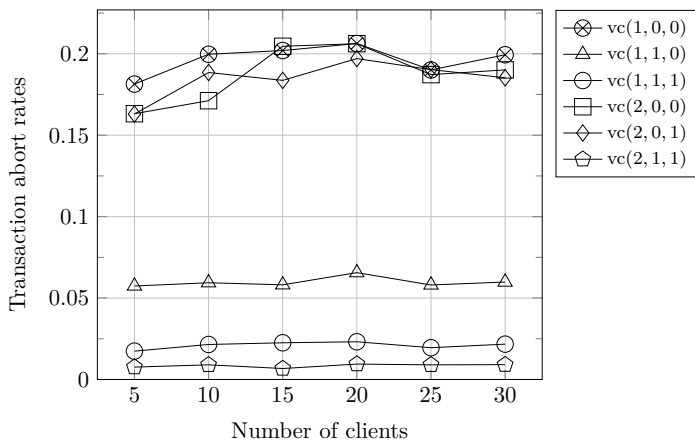


Table: Three categories of workload parameters for experiments on Aliyun.

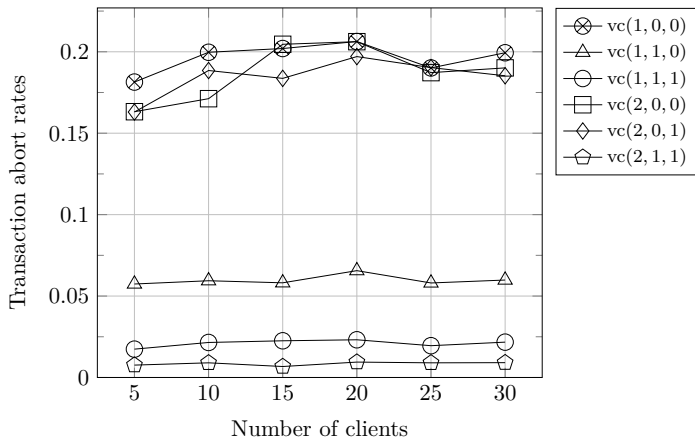
Parameter		Value	
Transaction-related	#keys	$25 = 5 \text{ (rows)} \times 5 \text{ (columns)}$	
	#clients	5, 10, 15, 20, 25, 30	
	#txs/client	1000	
	#ops/tx	$\sim \text{Binomial}(20, 0.5)$	n
	rwRatio	1:2, 1:1, 4:1	
	zipfExponent	1	
Execution-related	minInterval	0ms	
	maxInterval	10ms	
	meanInterval	5ms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



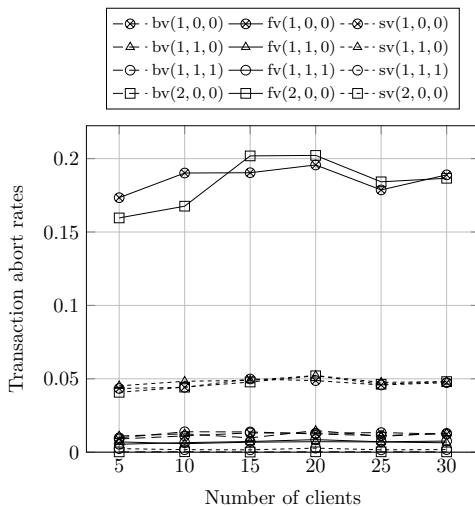


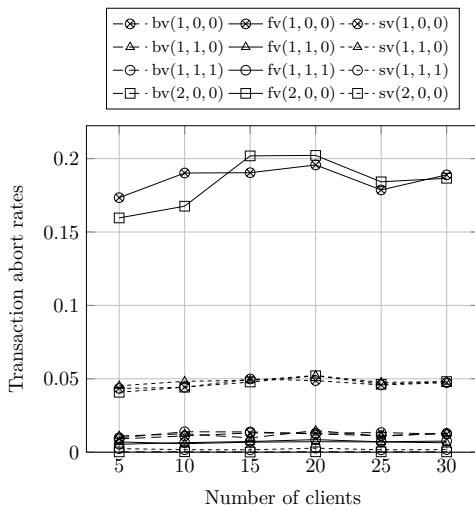
The transaction abort rates due to “vc-aborted”



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

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





Most “vc-aborted” transactions abort because of violating  $k_2\text{-FV}$ .

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

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?

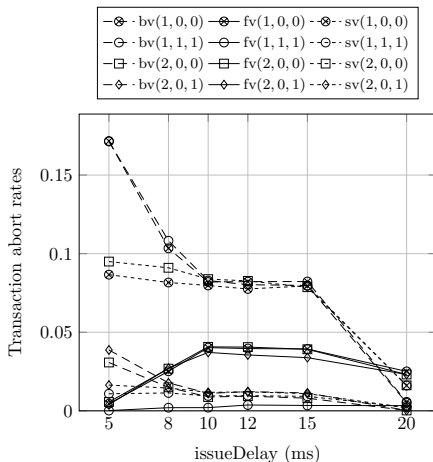
It seems that  $k_1$ -BV has *little* impact on the transaction abort rates.

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
<b>issueDelay</b>	5, 8, 10, 12, 15, 20	delays between clients and replicas
<b>replDelay</b>	5, 10, 15, 20, 30	delays between masters and slaves
<b>2pcDelay</b>	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:  $bv(1, 0, 0) = 0.0057$      $fv(1, 0, 0) = 0.0251$

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

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

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  $\implies$  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)