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. CHAMELEON Prototype and RVSI Protocol

RVSI: Relaxed Version Snapshot Isolation

1. CHAMELEON Prototype and RVSI Protocol

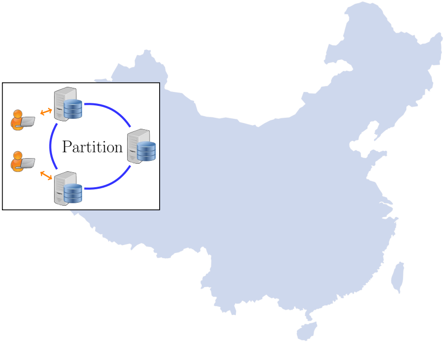
CHAMELEON prototype:

A prototype **partitioned replicated**

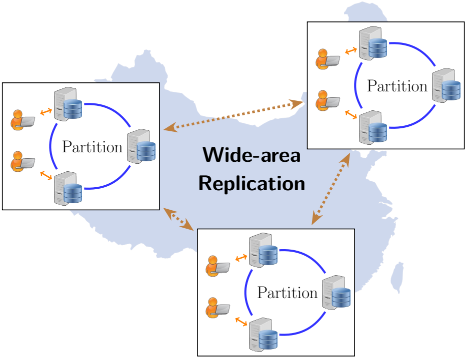
distributed transactional **key-value** store

Classic **key-value** data model

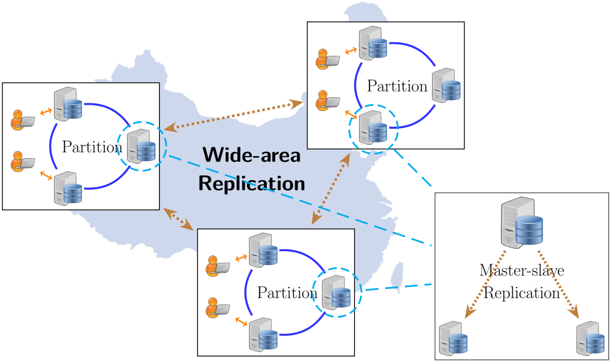
Key: (row key, column key)



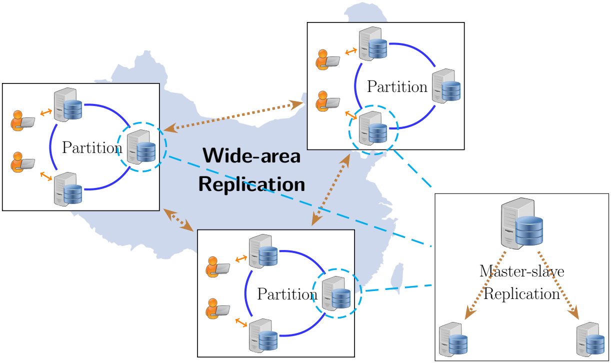
Keys are **partitioned** within a single datacenter.



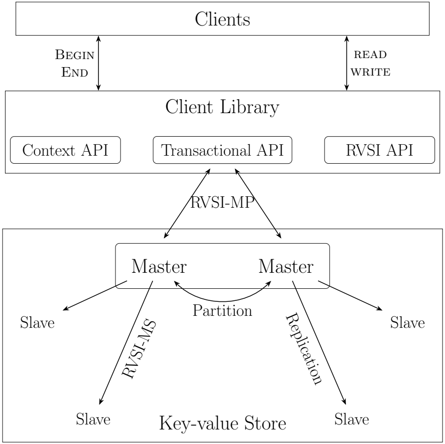
Each key is **replicated** across datacenters

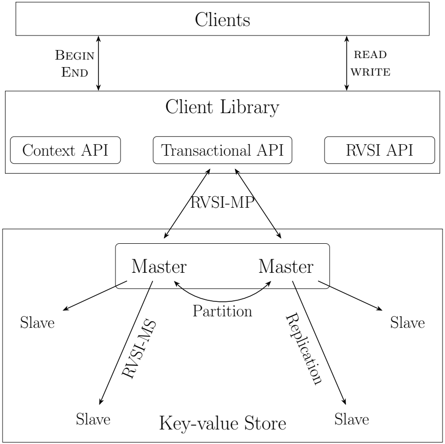


Each key is **replicated** across datacenters in a **master-slave** manner.

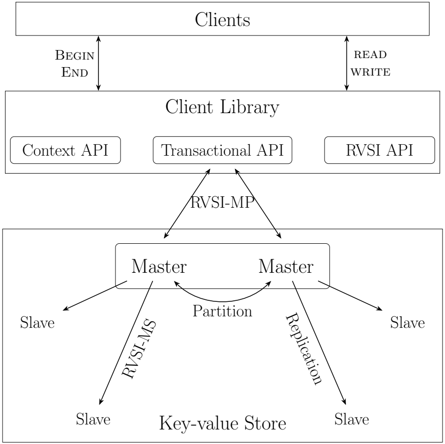


Transactions are first executed and committed on the **masters**, and are then asynchronously propagated to **slaves**.


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



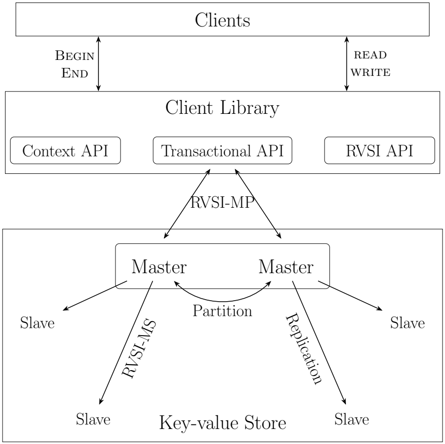
1. Partitioned replicated transactional key-value store



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

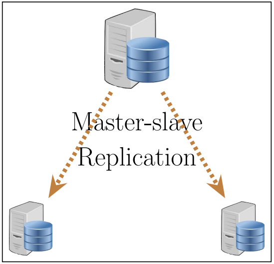
RVSI-MS: RVSI protocol for master-slave replication

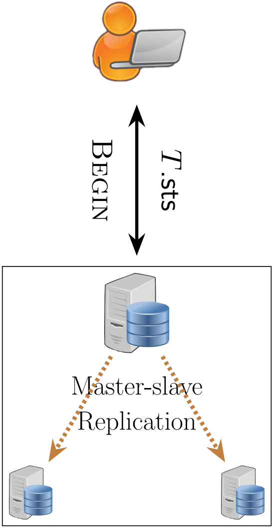
In terms of *event* generation and handling:

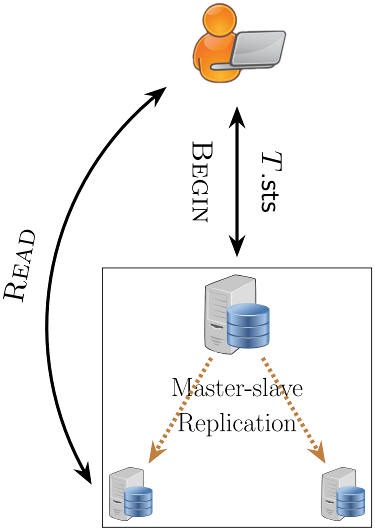
Clients: Begin, Read, Write, End

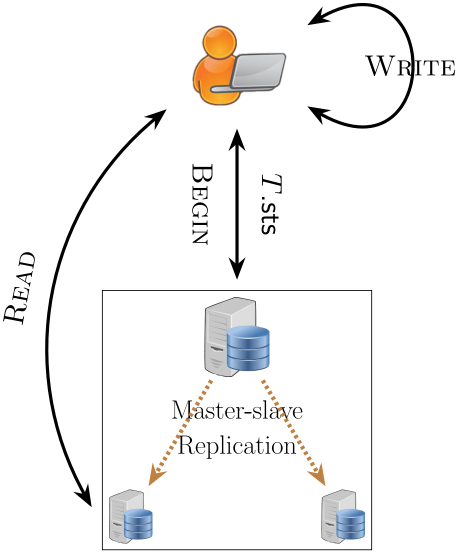
Master: Start, Commit, Send

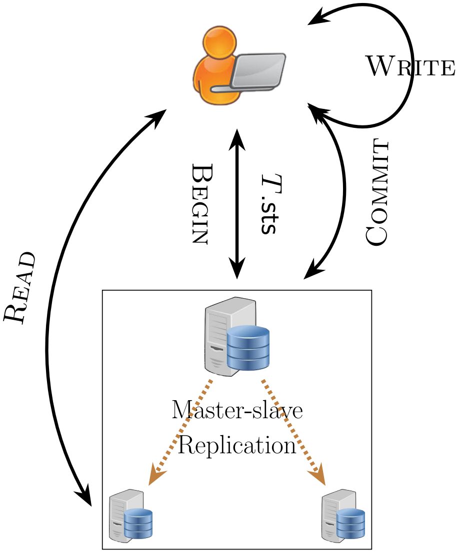
Slaves: Receive

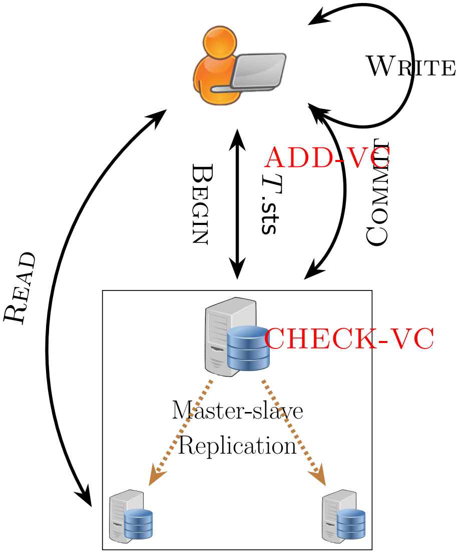

Master-slave
Replication


Begin
T.sts
Master-slave
Replication


Read
Begin
T.sts
Master-slave
Replication


WRITE
BEGIN
T.sts
READ
Master-slave
Replication


WRITE
BEGIN
T.sts
COMMIT
READ
Master-slave
Replication


Write
Begin
T.sts
ADD-VC
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:

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

k_2-FV:

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

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:

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

k_2-FV:

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

k_3-SV:

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

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

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
$\{x.ver = (x.ts, x.ord, x.val)\}$: set of versions of x

procedure START()
  return $++\mathcal{M}.ts$
procedure READ(x)
  return the latest $x.ver$ installed
procedure COMMIT($T.writes, T.vc$)
  if CHECK-VC($T.vc$) && write-conflict freedom **then**
    $T.cts \leftarrow ++\mathcal{M}.ts$
    ▷ *apply $T.writes$ locally and propagate it*
    $T.upvers = \emptyset$ ▷ *collect updated versions*
    for $(x, v) \in T.writes$ **do**
      $x.new\text{-}ver \leftarrow (T.cts, ++x.ord, v)$
      add $x.new\text{-}ver$ to $\{x.ver\}$ and $T.upvers$
    broadcast $\langle \text{PROP}, T.upvers \rangle$ 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 $\textsc{Read}(x)$
    return $x.\mathit{ver}$
upon $\textsc{Received}(\langle \text{PROP}, T.\mathit{upvers} \rangle)$
    for $(x.\mathit{ver}' = (x.\mathit{ts}', x.\mathit{ord}', x.\mathit{val}')) \in T.\mathit{upvers}$ **do**
        if $x.\mathit{ord}' > x.\mathit{ord}$ **then**
            $x.\mathit{ver} \leftarrow x.\mathit{ver}'$

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.

Distributed transactions spanning multiple masters need to be committed atomically.

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

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).

```
procedure BEGIN()
    return rpc-call GETTS() at 𝒯
procedure END()
    T.vc ← ADD-VC()
    c/a ← rpc-call C-COMMIT(T.writes, T.vc) at 𝒞
```

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

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

procedure GETTS()

  return $++\mathcal{T}.ts$

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*