TARDiS: A branch and merge approach to weak consistency

By: Natacha Crooks, Youer Pu, Nancy Estrada, Trinabh Gupta, Lorenzo Alvis, Allen Clement

Presented by: Samodya Abeysiriwardane

TARDiS

Transactional key-value store for weakly consistent systems

Weakly consistent systems

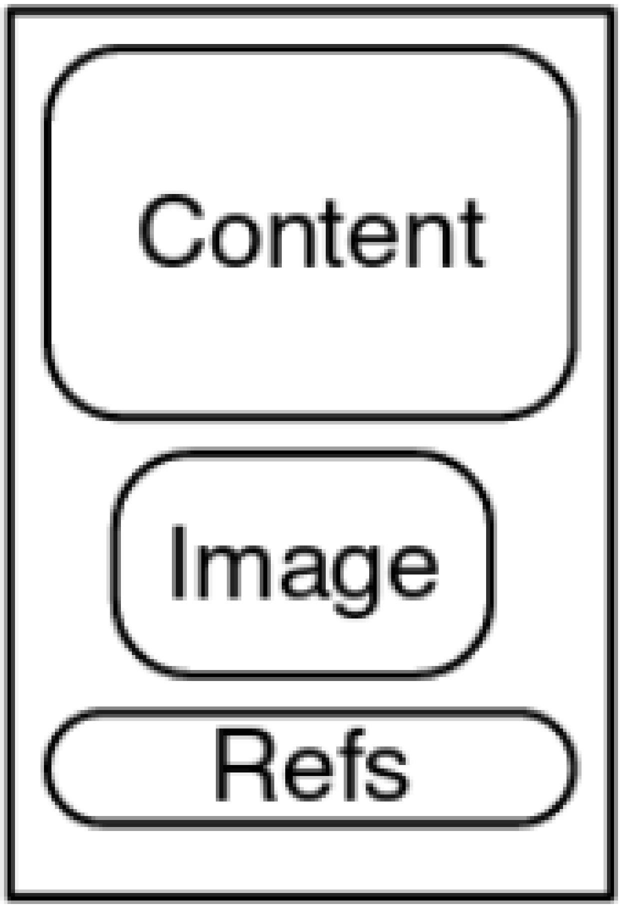
ALPS (Available, low Latency, Partition tolerance, high Scalability)

Conflicting operations cause replicas to diverge

Current solutions: Deterministic Writer Wins, per object eventual convergence (object as unit of merging)

Current solutions are not sufficient

Motivation



A wiki page with three objects

Edited at two georeplicated replicas

Motivation

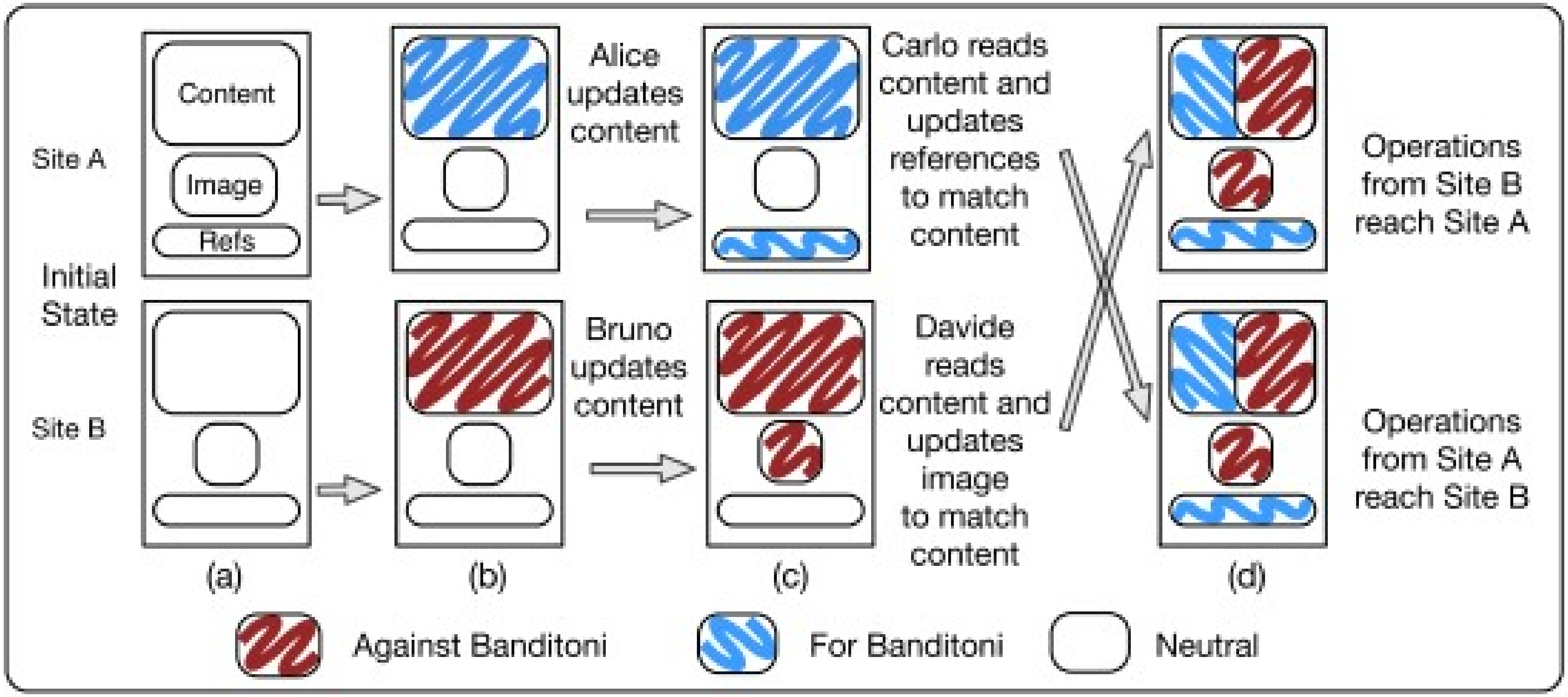


Figure 1: Weakly-consistent Wikipedia

Main goal

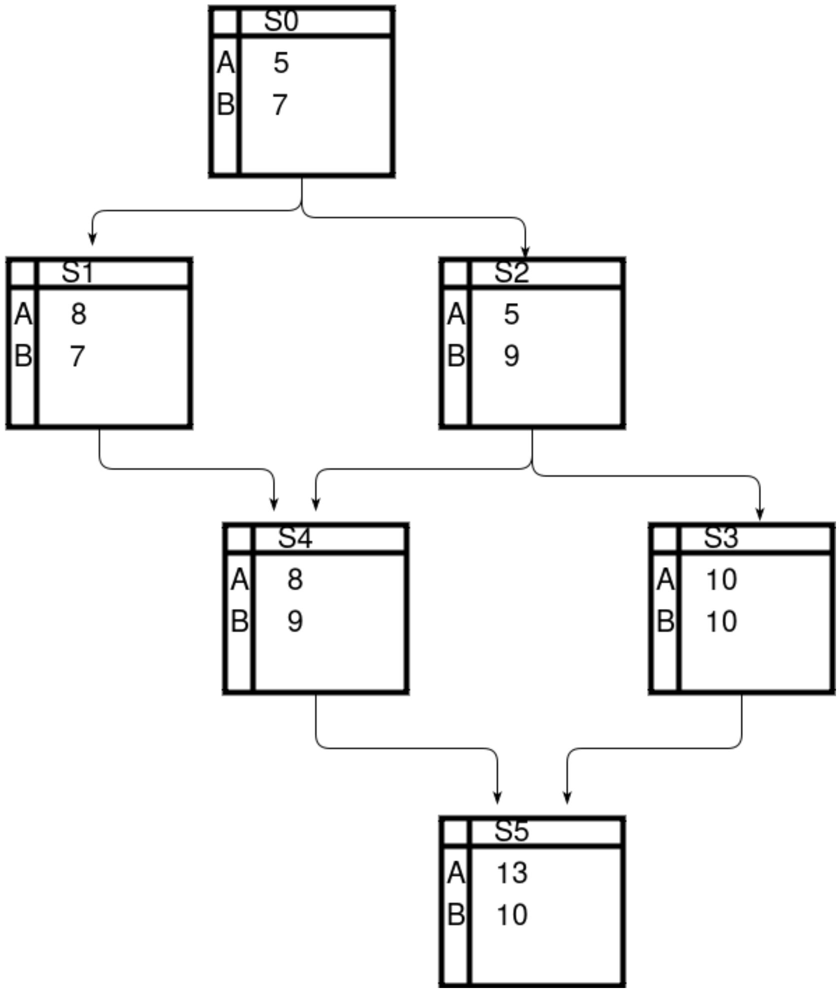
Give applications access to context that is essential for reasoning about concurrent updates

Proposed solution

Expose branches as a unit of merging

- **branch on conflict**
- **branch isolation**
- **application driven merges**

Simple Example with Counters



Key value store
of Counters

Merge

Need to define a merge function for the application

Merging two counters A and B

For counters 2-way merge

```
fn merge (lca, a, b) = lca + (a-lca) + (b-lca)
```

For counters n-way merge

```
fn merge {
lca = find_fork_point
val = lca
for v in conflicting_values:
  val += (a – lca) + (b – lca)
}
```

Simple Example with Counter (Code)

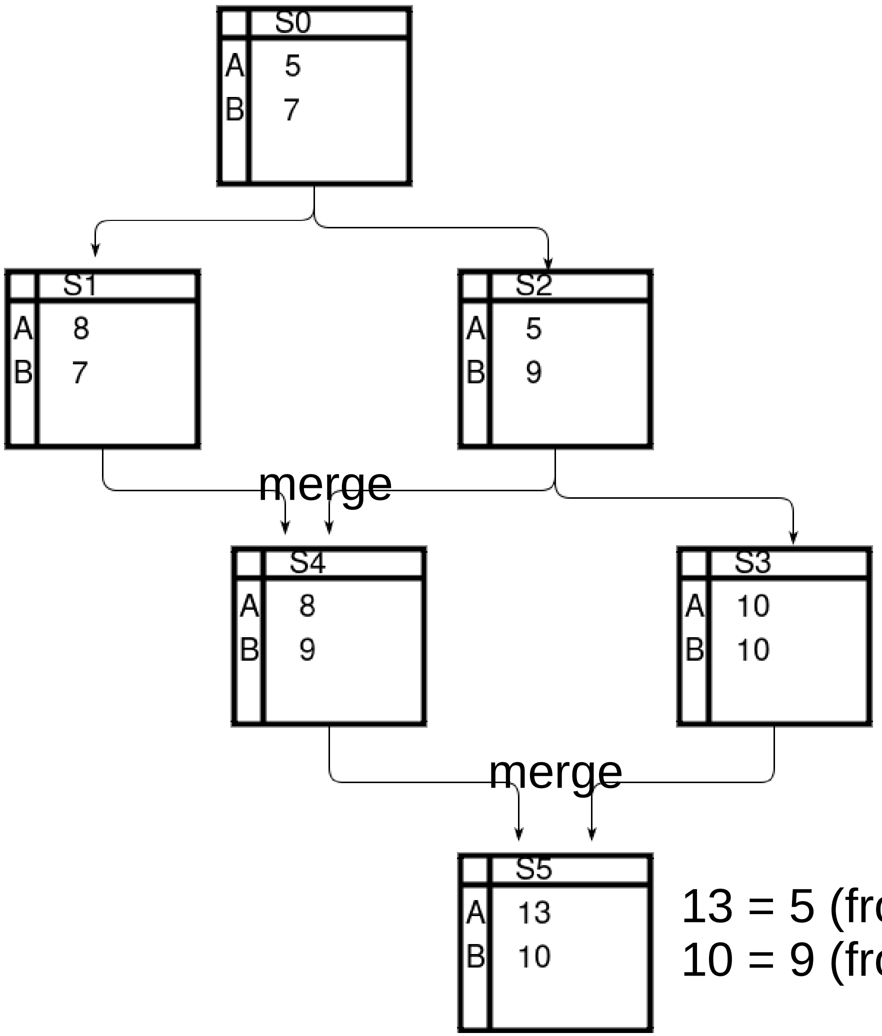
```
func increment(counter)
 Tx t = begin(AncestorConstraint)
 int value = t.get(counter)
 t.put(counter, value + 1)
 t.commit(SerializabilityConstraint)

func decrement(counter)
 Tx t = begin(AncestorConstraint)
 int value = t.get(counter)
 t.put(counter, value - 1)
 t.commit(SerializabilityConstraint)

func merge()
  Tx t = beginMerge(AnyConstraint)
  forkPoint forkPt =
          t.findForkPoints(t.parents).first
  int forkVal = t.getForID(counter,forkPt)
  list<int> currentVals =
          t.getForID(counter, t.parents)
  int result = forkVal
  foreach c in currentVals
    result += (c - forkVal)
  t.put(counter,result)
  t.commit(SerializabilityConstraint)
```

Figure 3: TARDiS' counter implementation

Simple Example with Counter (Code)



13 = 5 (from S2) + (8-5)+(10-5)
10 = 9 (from S2) + (9-9)+(10-9)

<u>Client1</u>
T1:
inc(A, 3)

Tm:
merge

<u>Client2</u>
T2:
inc(B,2)

T3:
inc(A,5)
inc(B,1)

Tm:
merge

Example

Impose an application invariant of

- if A > 8: B should max at 10

- the merge function can be changed to reflect that

Highlights the need for cross object merging semantics vs per object merging

Therefore branches as a unit of merging

Another example: Inventory

XYZ_stock: 1

ABC_stock: 3

Alice buys XYZ

XYZ_stock: 0

Bob buys XYZ and ABC

XYZ_stock: 0

ABC_stock: 2

Merge

Bob get XYZ, and exp

Alice gets error

XYZ_stock: 0

exp_stock: 2

Invariant: stock cannot be < 0

Other advantages

No locking required

Branching as a fundamental abstraction for modeling conflicts end to end – replicas as well the local site can be viewed as branches

TARDIS API

S	M	return type	method
√		transaction	begin(*beginConstraint*)
	√	transaction	beginMerge(*beginConstraint*)
√	√	void	put(*key, value*)
√		value	get(*key*)
	√	value	getForID(*key, StateID[ ]*)
	√	key[]	findConflictWrites(*StateID[ ]*)
	√	forkPoints[]	findForkPoints(*StateID[ ]*)
√	√	abort|commit	commit(*endConstraint*)

Table 2: TARDiS API - S:single mode, M:merge mode

TARDiS architecture

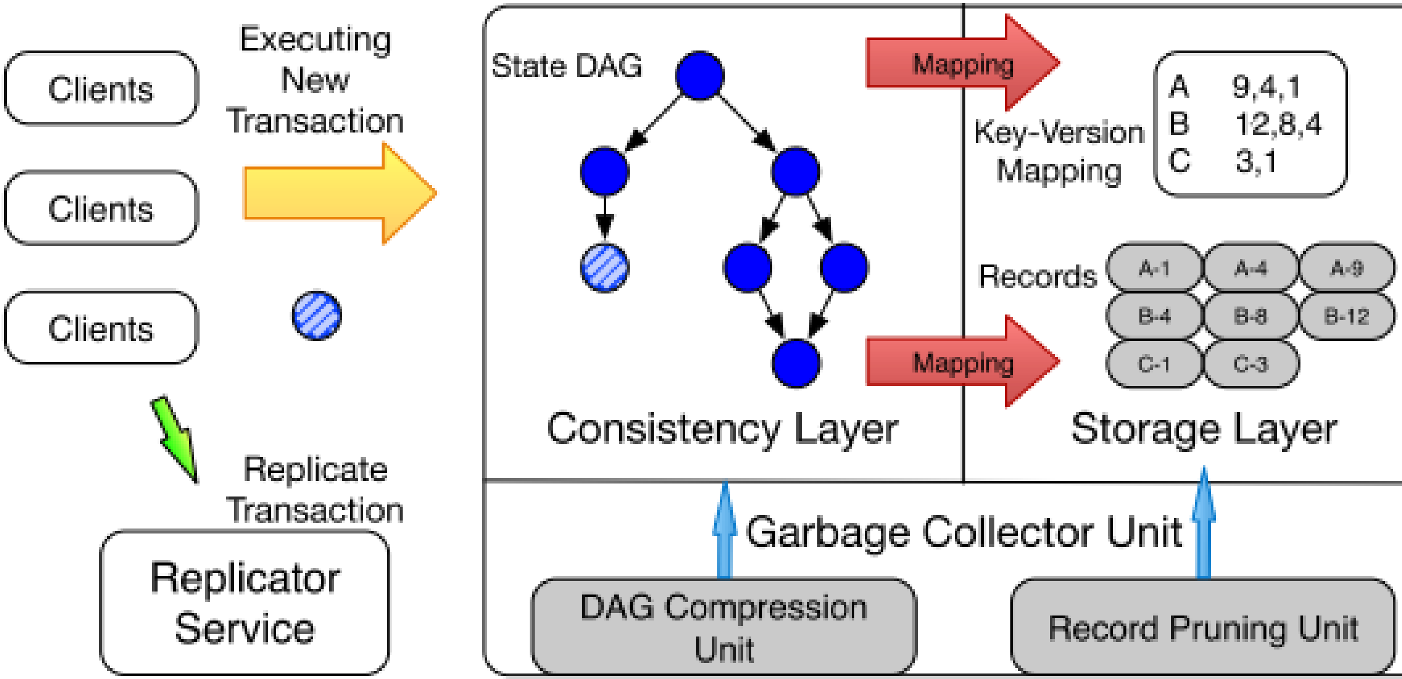


Figure 2: TARDiS architecture

TARDiS architecture

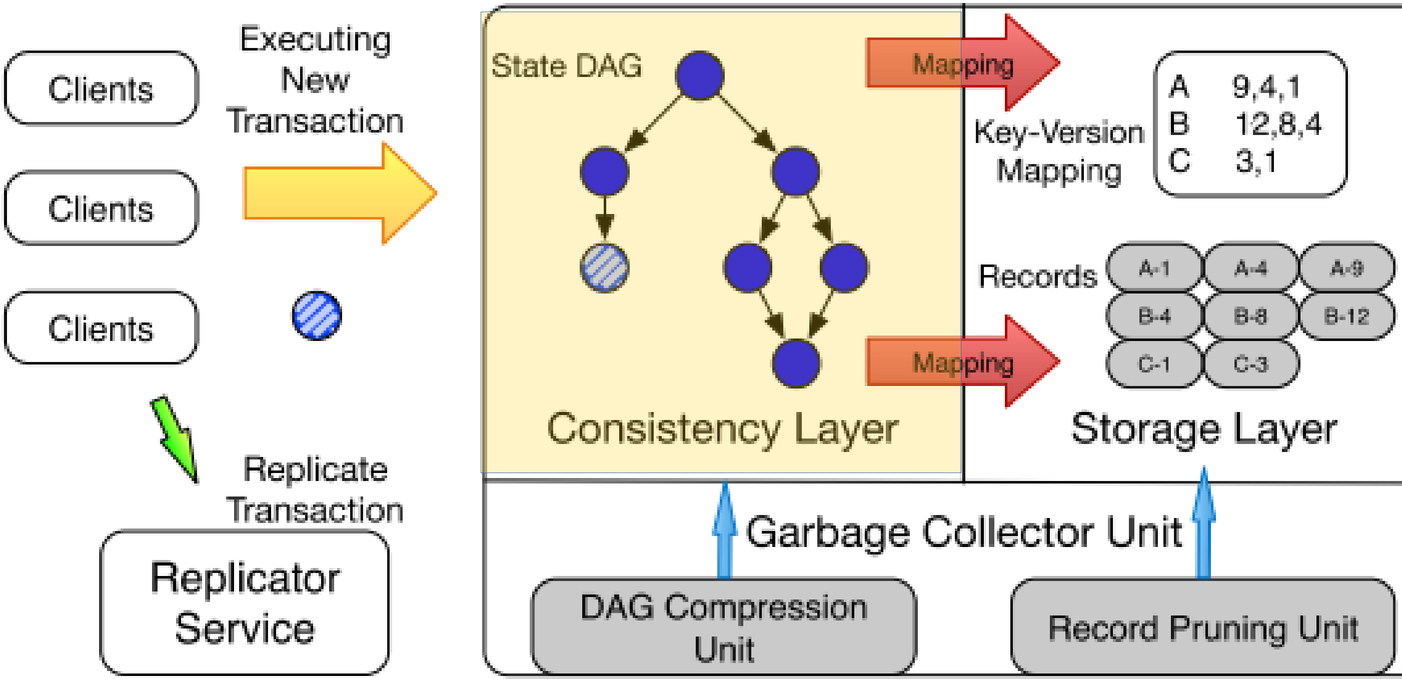


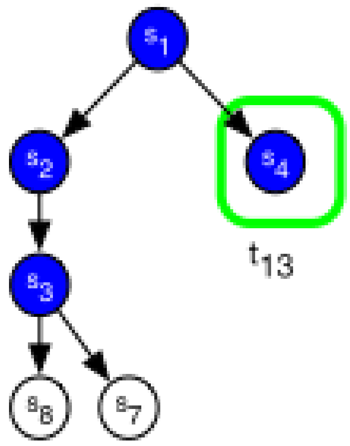
Figure 2: TARDiS architecture

Consistency layer

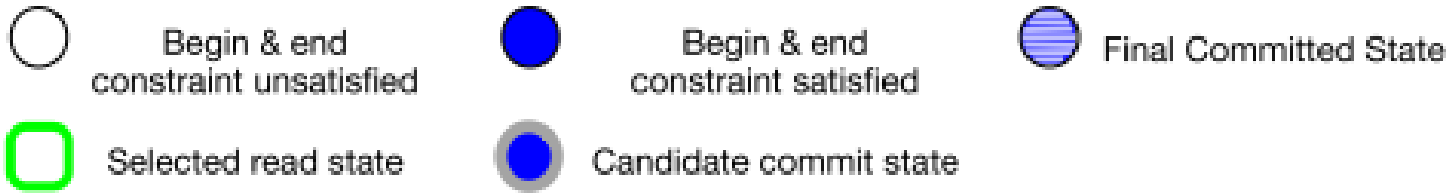
Constraint	B	E	Description
Any	√	√	Always Satisfies
Serializability		√	Guarantees Serializability
Snapshot Iso		√	Guarantees Snapshot Isolation
Read Committed		√	Guarantees Read Committed
No Branching		√	State has no children
K-Branching		√	State has fewer than k-1 children
Parent	√		State where client last committed
Ancestor	√		Child of client's last committed state
State Identifier	√		State ID matches the specified ID

Table 1: Begin (B) and end (E) constraints supported by TARDiS

Consistency layer



begin(AncestorConstraint)

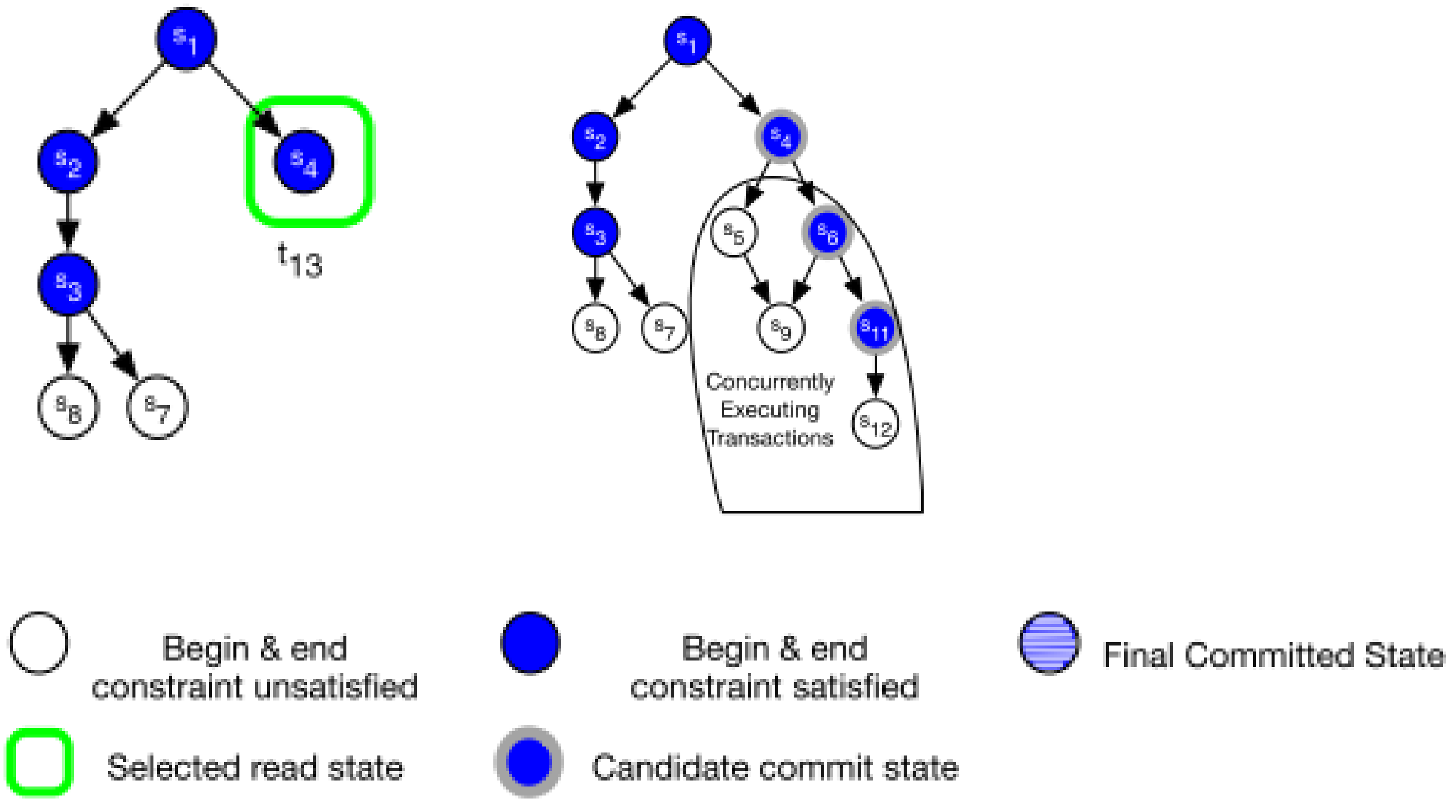


Consistency layer

Constraint	B	E	Description
Any	√	√	Always Satisfies
Serializability		√	Guarantees Serializability
Snapshot Iso		√	Guarantees Snapshot Isolation
Read Committed		√	Guarantees Read Committed
No Branching		√	State has no children
K-Branching		√	State has fewer than k-1 children
Parent	√		State where client last committed
Ancestor	√		Child of client's last committed state
State Identifier	√		State ID matches the specified ID

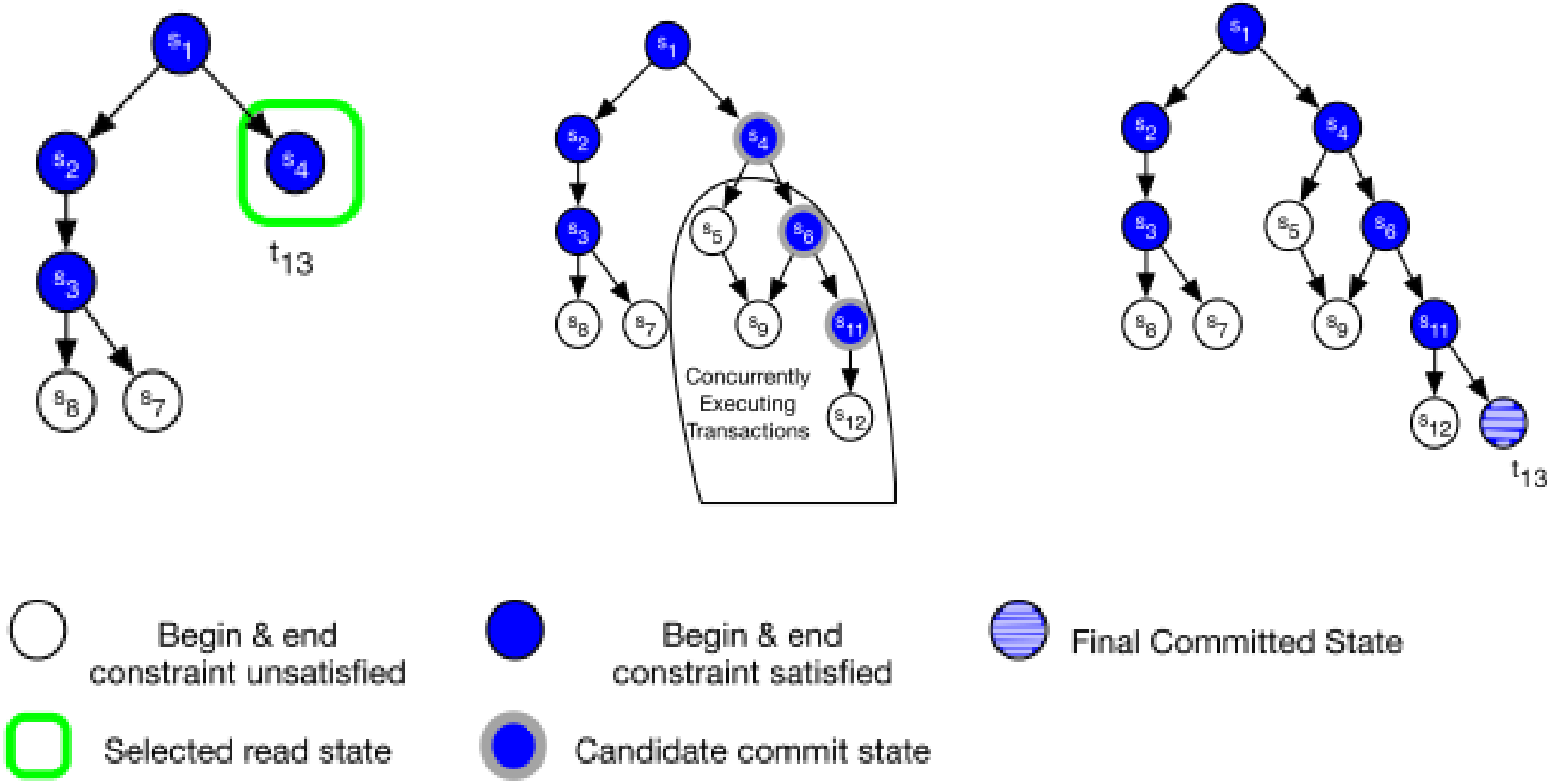
Table 1: Begin (B) and end (E) constraints supported by TARDiS

Consistency layer



Consistency layer

commit(SerializabilityConstraint)



Consistency layer

Constraint	B	E	Description
Any	√	√	Always Satisfies
Serializability		√	Guarantees Serializability
Snapshot Iso		√	Guarantees Snapshot Isolation
Read Committed		√	Guarantees Read Committed
No Branching		√	State has no children
K-Branching		√	State has fewer than k-1 children
Parent	√		State where client last committed
Ancestor	√		Child of client's last committed state
State Identifier	√		State ID matches the specified ID

Table 1: Begin (B) and end (E) constraints supported by TARDiS

TARDiS architecture

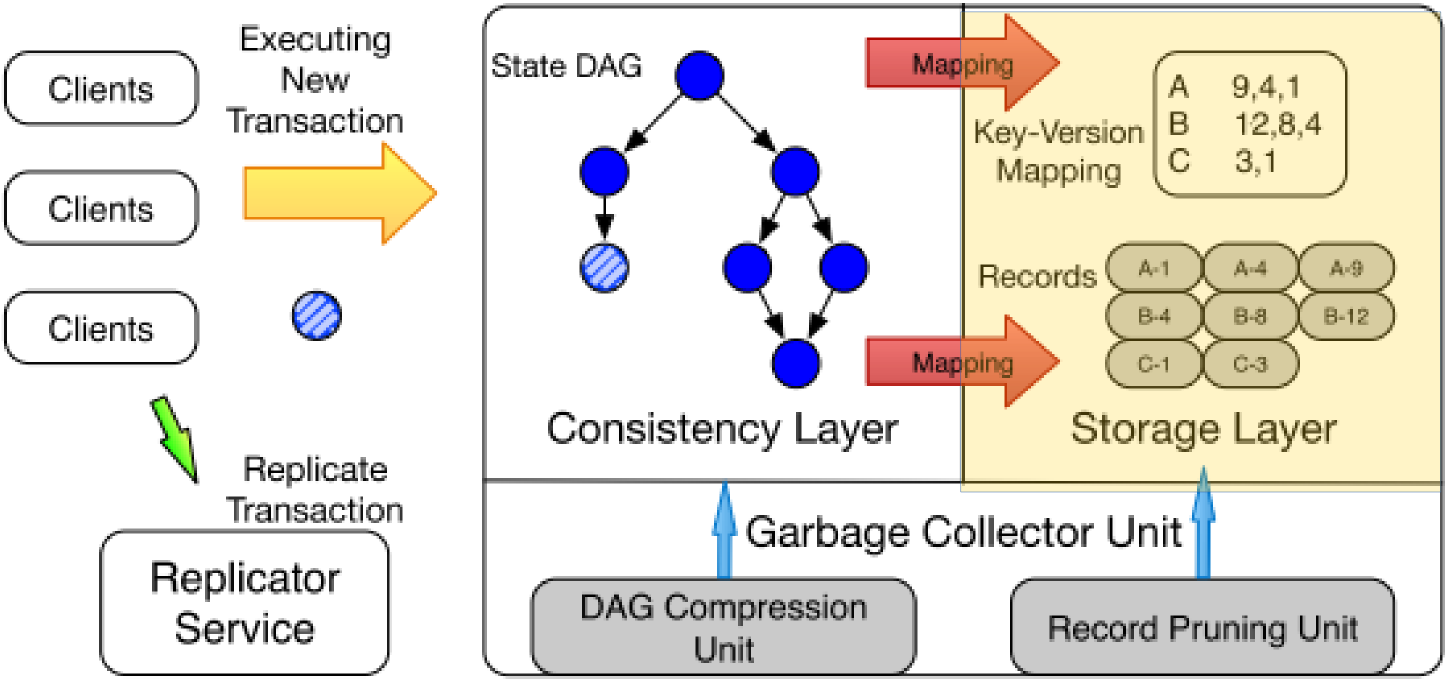
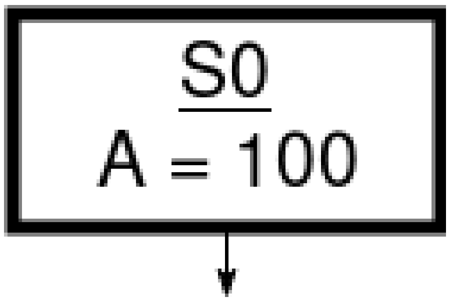


Figure 2: TARDiS architecture

Data structures

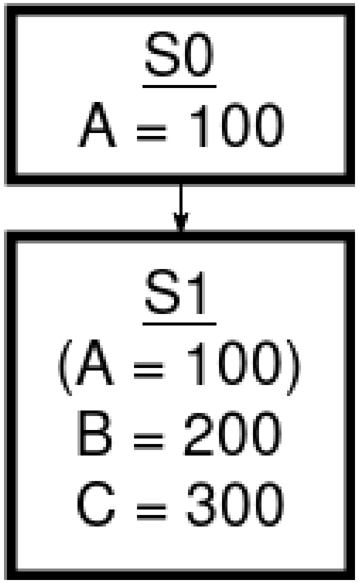


Key version mapping
A | S0

Record B-tree
A | S0

Fork paths:
The set of fork points
S0: {}

Data structures

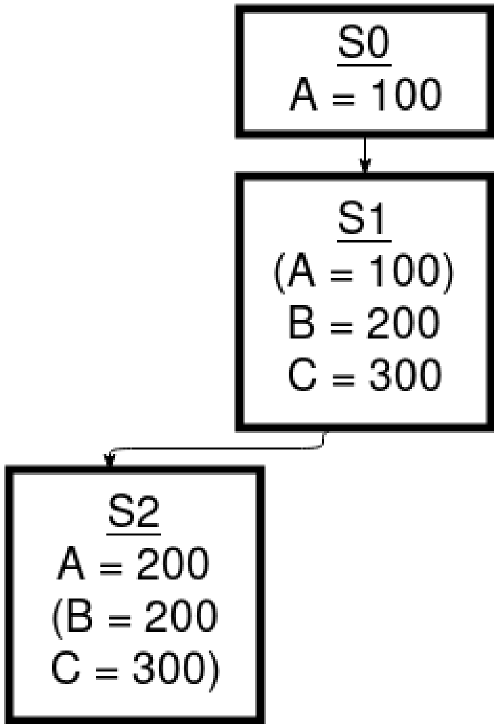


Key version mapping
A | S0
B | S1
C | S1

Record B-tree
A | S0
B | S1
C | S1

Fork paths:
S0: {}
S1: {}

Data structures

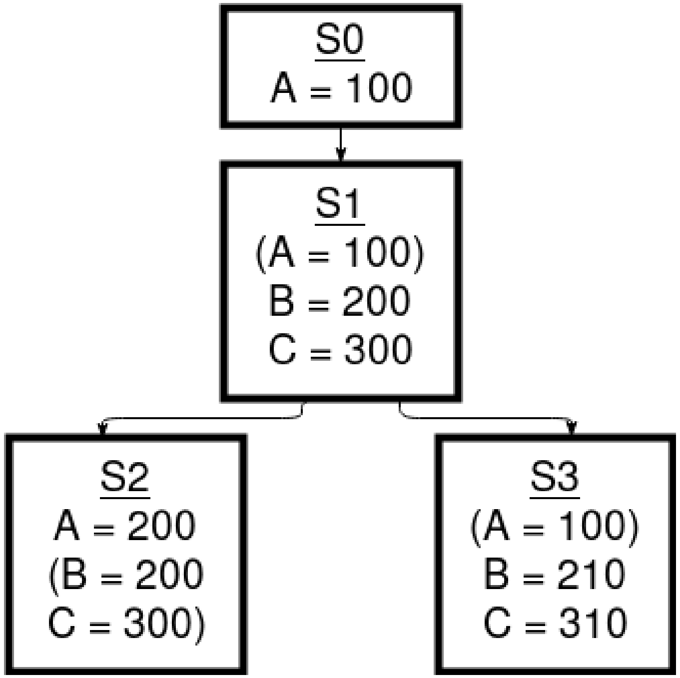


Key version mapping
A | S2, S0
B | S1
C | S1

Record B-tree
A | S0 → S2
B | S1
C | S1

Fork paths: (set of tuples i,b where current state is bth child of state i)
S0: {}
S1: {}
S2: { (1,1) }

Data structures



Key version mapping
A | S2, S0
B | S3, S1
C | S3, S1

Record B-tree
A | S0 → S2
B | S1 → S3
C | S1 → S3

Fork paths: (set of tuples i,b where current state is bth child of state i)
S0,S1: {}
S2: { (1,1) }
S3: { (1,2) }

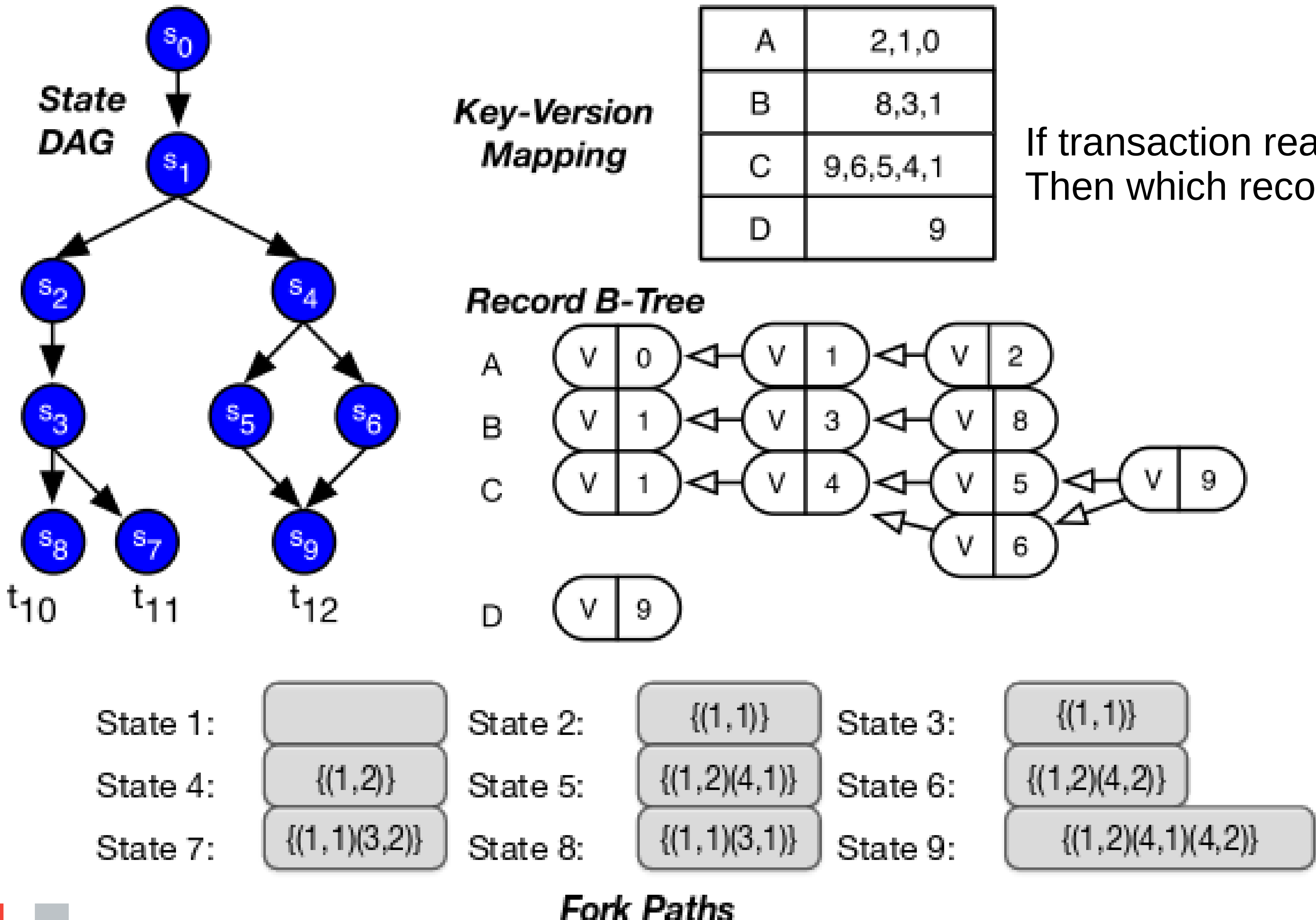
Data structures

```
descendantCheck(x, y):
  if x.id = y.id then return true
  else if x.id > y.id then return false
  else if x.path ⊈ y.path then return false
  else return true
```

Figure 7: Check if state y can see records associated with state x

A record version belongs to the selected branch if the fork path associated with this record version is a subset of the fork path of the transaction’s read state

Data structures



If transaction read state is S3
Then which record version of C?

TARDiS architecture

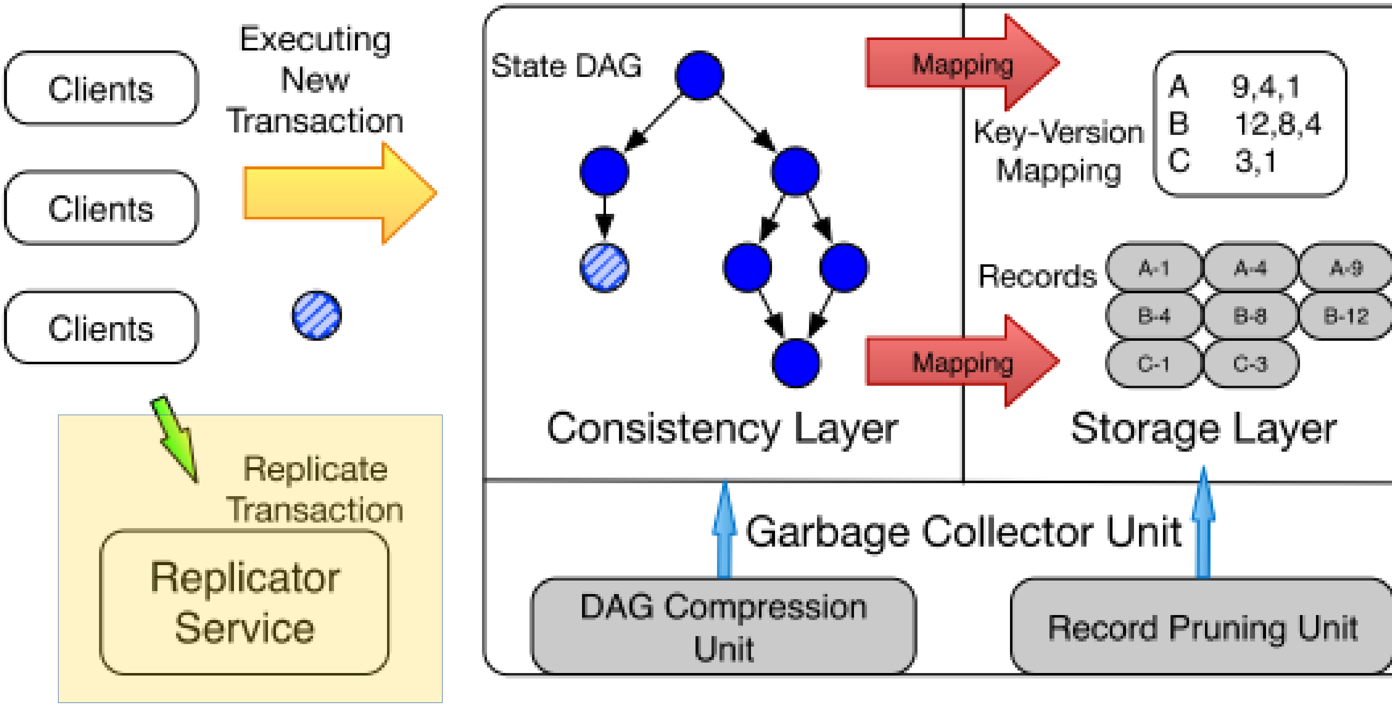


Figure 2: TARDiS architecture

Evaluation setup

Shared local cluster

2.67 GHz Intel Xeon CPU X5650

48GB memory

2Gbps network

3 dedicated server machines

3 dedicated replicators

Equally spread clients

For comparison

Databases

Berkley DB (BDB) – ACID datastore

An implementation that does not require read write transactions to be verified against read-only transactions (OCC)

Operation composition

Read heavy (75R/25W)

Write heavy (0R/100W)

Baseline TARDiS

Selecting constraints so that execution is serializable, and there is no branching

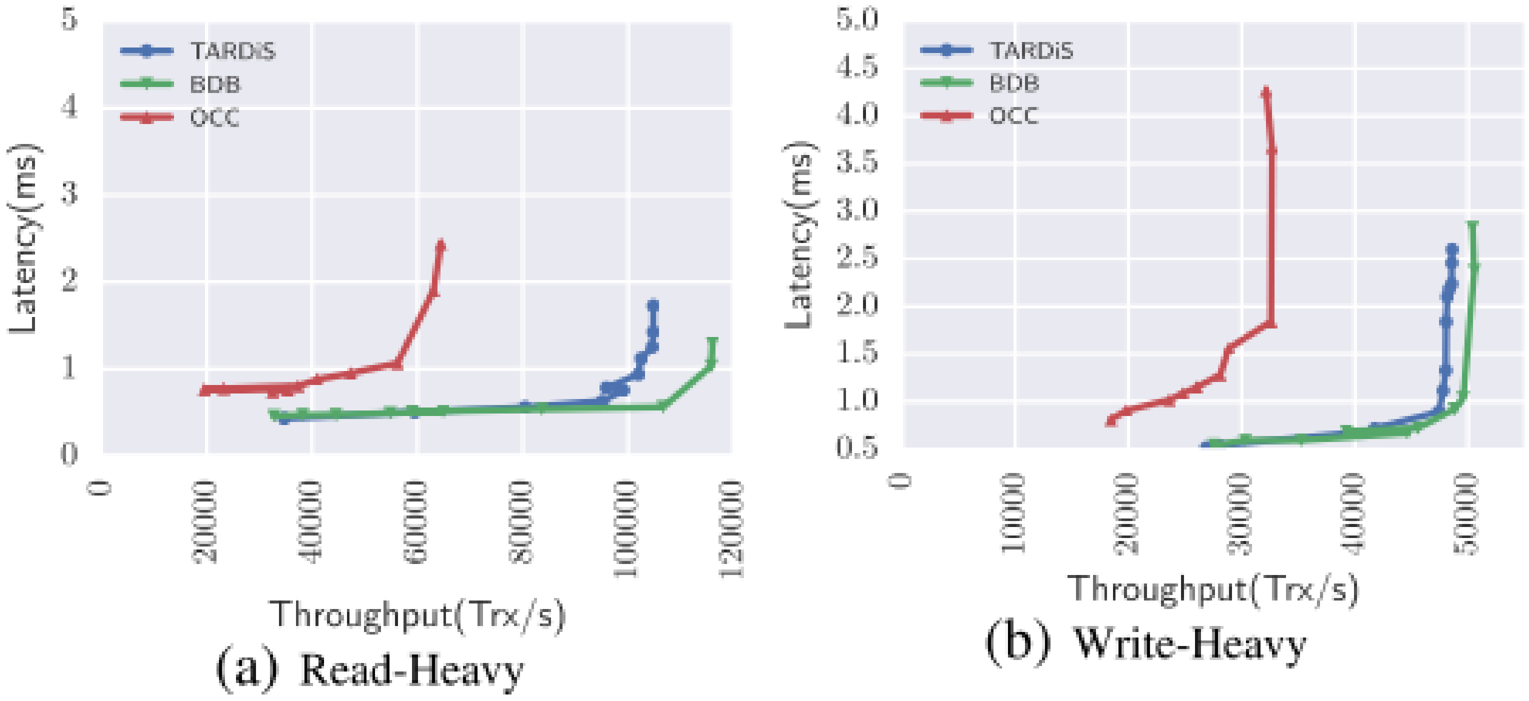
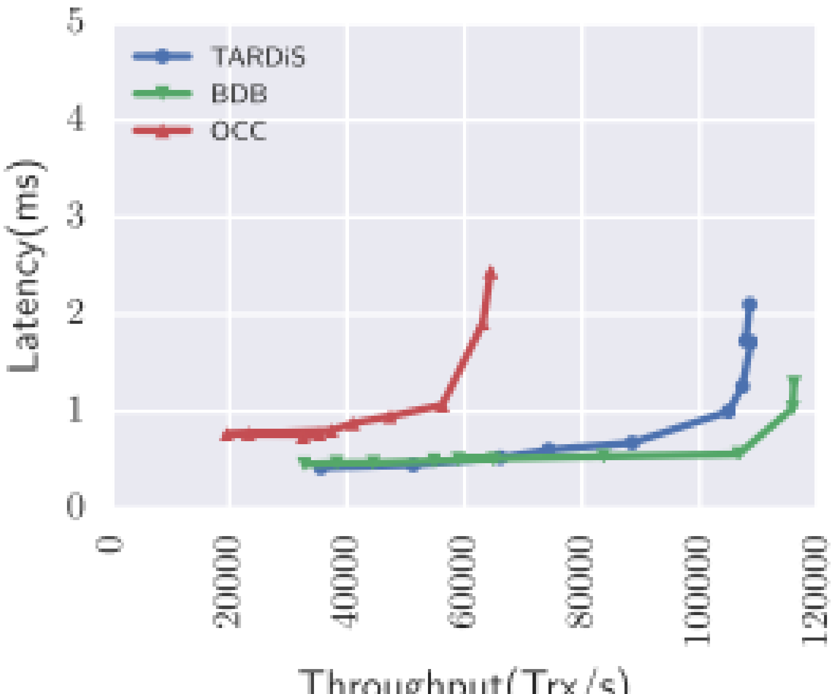
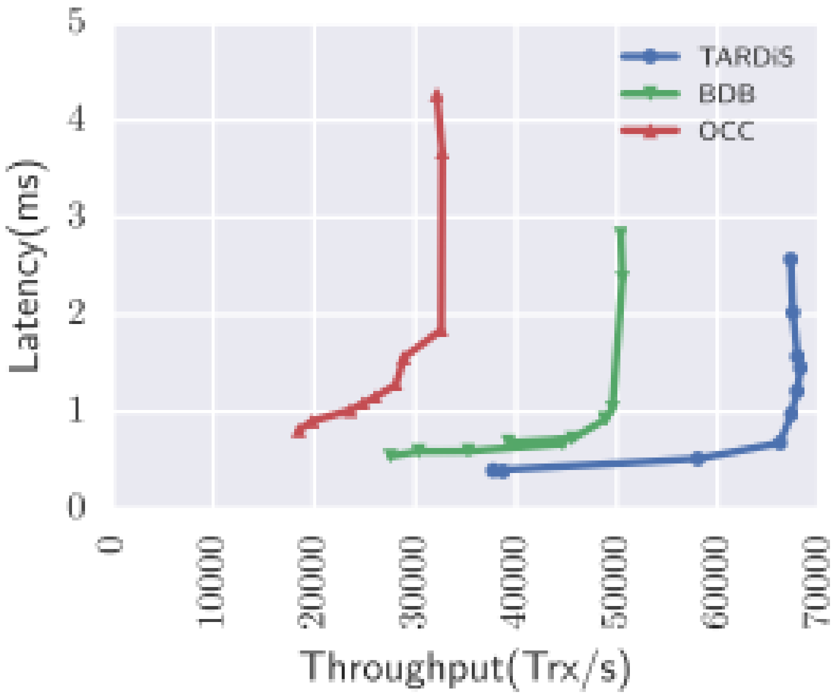


Figure 9: TARDiS-BDB vs BerkeleyDB vs OCC

With branching

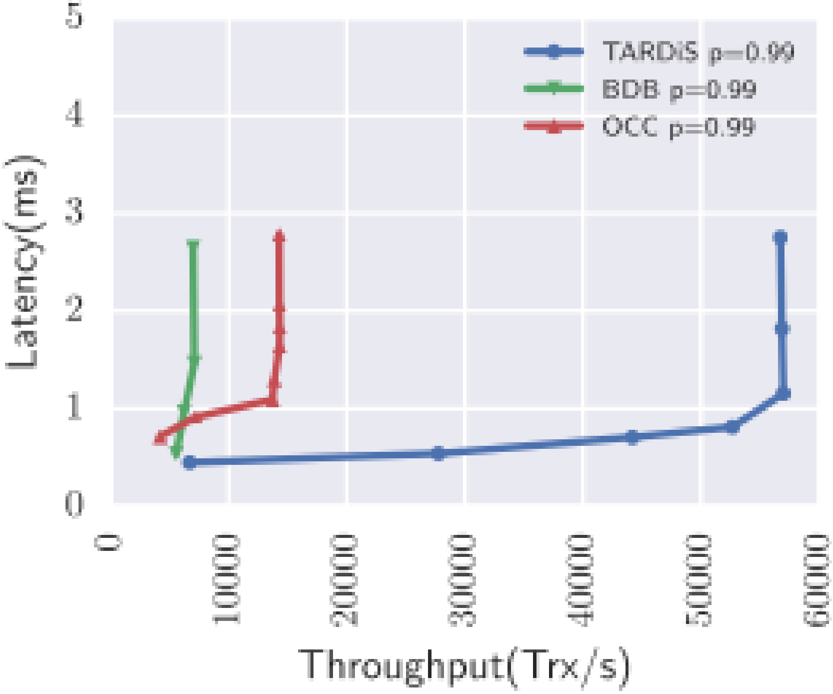


(a) Uniform Read-Heavy

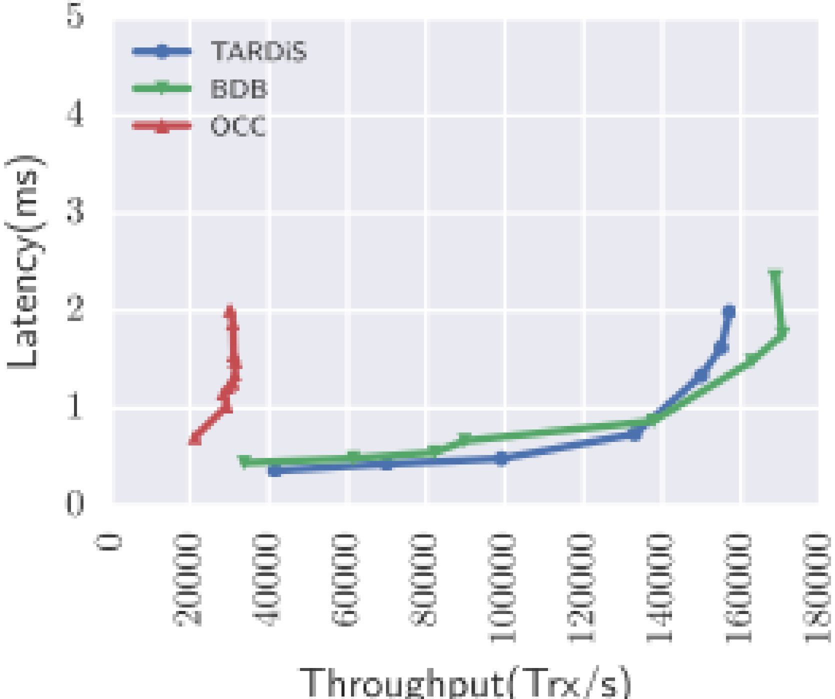


(b) Uniform Write-Heavy

With branching



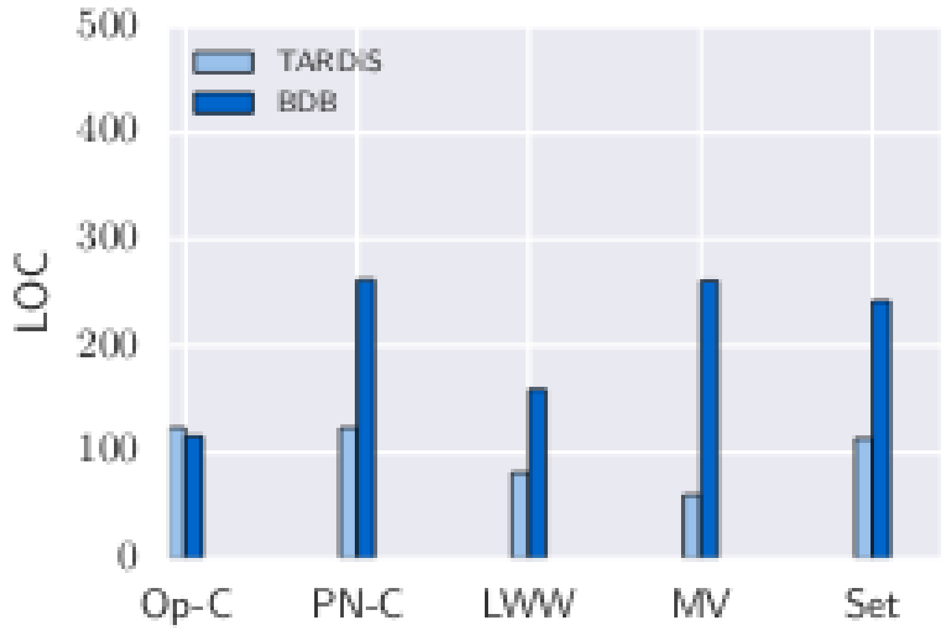
(c) Zipfian Write-Heavy



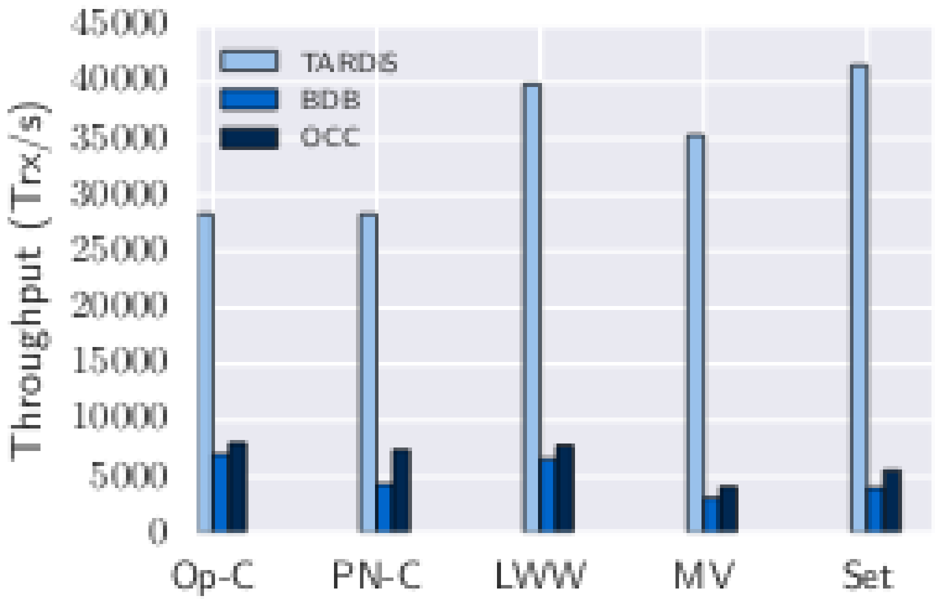
(d) Uniform Blind Writes

Figure 10: Benefit of branching as a function of workload

CRDT implementations



(a) CRDT Lines of Code



(b) CRDT Throughput

Op-C:Operation Based Counter, PN-C: State Based Counter, LWW: Last-Writer-Wins Register, MV: Multivalued Register, Set: Or-Set

Insight

Branching as a means to provide an abstraction that lifts WW conflicts to the application level so that application developer can determine the intended outcome of conflicts in a weakly consistent application

Next

Hard for programmer to reason about the whole application state in merge function. Therefore have the ability to compose a merge function from multiple merge functions

Having the ability to push and pull from other states so that synchronization can happen asynchronosly and by on request