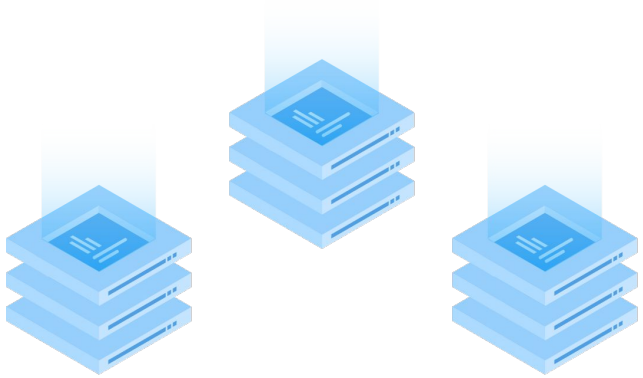
Head First
Distributed Transaction in TiDB

Presented by wuxuelian

PingCAP TiDB



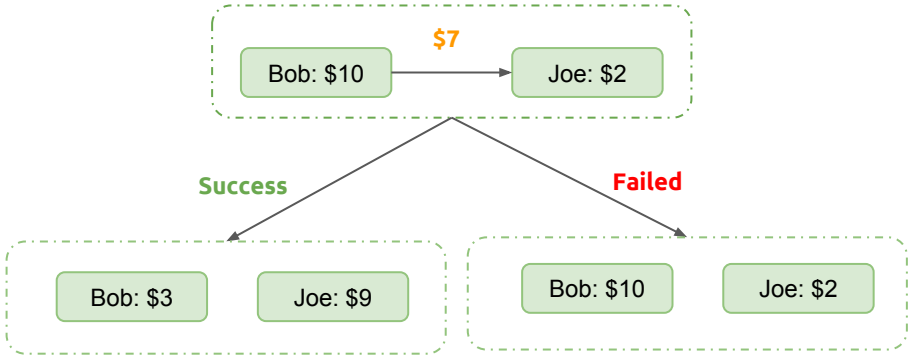
Agenda

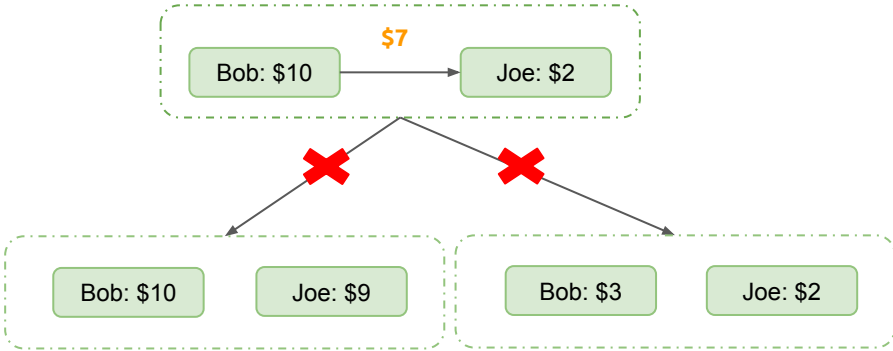
- ACID
- ISOLATION LEVEL
- Percolator
- Transaction in TiDB

PingCAP

Part I - ACID

ACID


$7
Bob: $10
Joe: $2
Success
Failed
Bob: $3
Joe: $9
Bob: $10
Joe: $2


$7
Bob: $10
Joe: $2
Bob: $10
Joe: $9
Bob: $3
Joe: $2

ACID

- **Atomicity**
  - Each transaction is treated as a single "unit", which *either succeeds completely, or fails completely*
- **Consistency**
  - Any data written to the database must be valid according to *all defined rules*.
- **Isolation**
  - Isolation ensures that *concurrent execution of transactions leaves the database in the same state* that would have been obtained if the transactions were executed sequentially
- **Durability**
  - Once a transaction has been committed, it will *remain committed even in the case of a system failure*

PingCAP

Part II - Isolation Levels

Read uncommitted

Session A	*Session B*
begin; select account from account where id = 1 // will get 1000	
	begin; update account set account=account+500 where id = 1 // not commit here
select account from account where id = 1 // will get 1500 (**Dirty read**)	
	rollback;

Read committed

Session A	*Session B*
begin; select account from account where id=1; // get 1000	
	begin; update account set account = account+500 where id=1; **commit**;
select account from account where id = 1; // get 1500 (**Non-repeatable reads**) **commit**;	

Repeatable read

Session A	*Session B*
begin; select account from account where id=1; // get 1000	
	begin; update account set account = account+500 where id=1; **commit**;
select account from account where id = 1; // get 1000 **commit**;	

Repeatable read

Session A	*Session B*
begin; select id from account; // get id(1), id(2)	
	begin; insert into account values(3,"Dada",5000); **commit**;
select id from account; // get id(1), id(2)	
insert into account values(3,"Dada",5000); // ERROR 1062 (23000): Duplicate entry '3' for key 'PRIMARY' (**Phantom reads**)	

Serializable

Session: A

```
mysql> set session transaction isolation level serializable;
Query OK, 0 rows affected (0.00 sec)

mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from account;
+----+-------+---------+
| id | name  | account |
+----+-------+---------+
|  1 | Alice |    1000 |
|  2 | Bob   |    1000 |
+----+-------+---------+
2 rows in set (0.00 sec)
```

Transaction A

Session: B

```
mysql> insert into account values(3,"Dada",5000);
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
mysql>
```

PingCAP

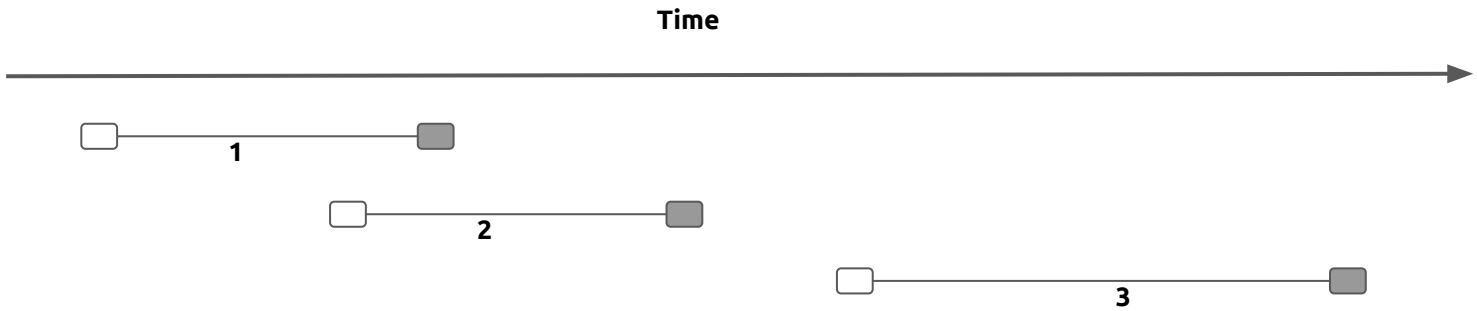
Summary

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
Read uncommitted	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible
Repeatable read	Not possible	Not possible	Not possible in TiDB
Serializable	Not possible	Not possible	Not possible

PingCAP

Part III - Percolator

Snapshot Isolation



- Read: read from a stable snapshot at some timestamp
- Write: protects against write-write conflicts.

2 Phase Commit

Bob have $10, Joe have $2, Bob will give Joe $7.

key	data	lock	write
Bob	5: $10		
			6: data @5
Joe	5: $2		
			6: data @5

Phase#1 : Prewrite

key	data	lock	write
Bob	5: $10		
			6: data @5
	7:$3	7:I'm primary	
Joe	5: $2		
			6: data @5
	7:$9	7:primary @ Bob	

Phase#2: Primary Commit (Sync)

Bob have **$3**, Joe have **$9** now.

key	data	lock	write
Bob	5: $10		
			6: data @5
	7:$3	~~7: I'm primary~~	
			8: data @7
Joe	5: $2		
			6: data @5
	7:$9	7: primary @ Bob	

PingCAP

Phase#2: Secondary Commit (Async)

Bob have **$3**, Joe have **$9** now.

key	data	lock	write
Bob	5: $10		
			6: data @5
	7:$3	~~7: I'm primary~~	
			8: data @7
Joe	5: $2		
			6: data @5
	7:$9	~~7: primary @ Bob~~	
			8: data @7

PingCAP

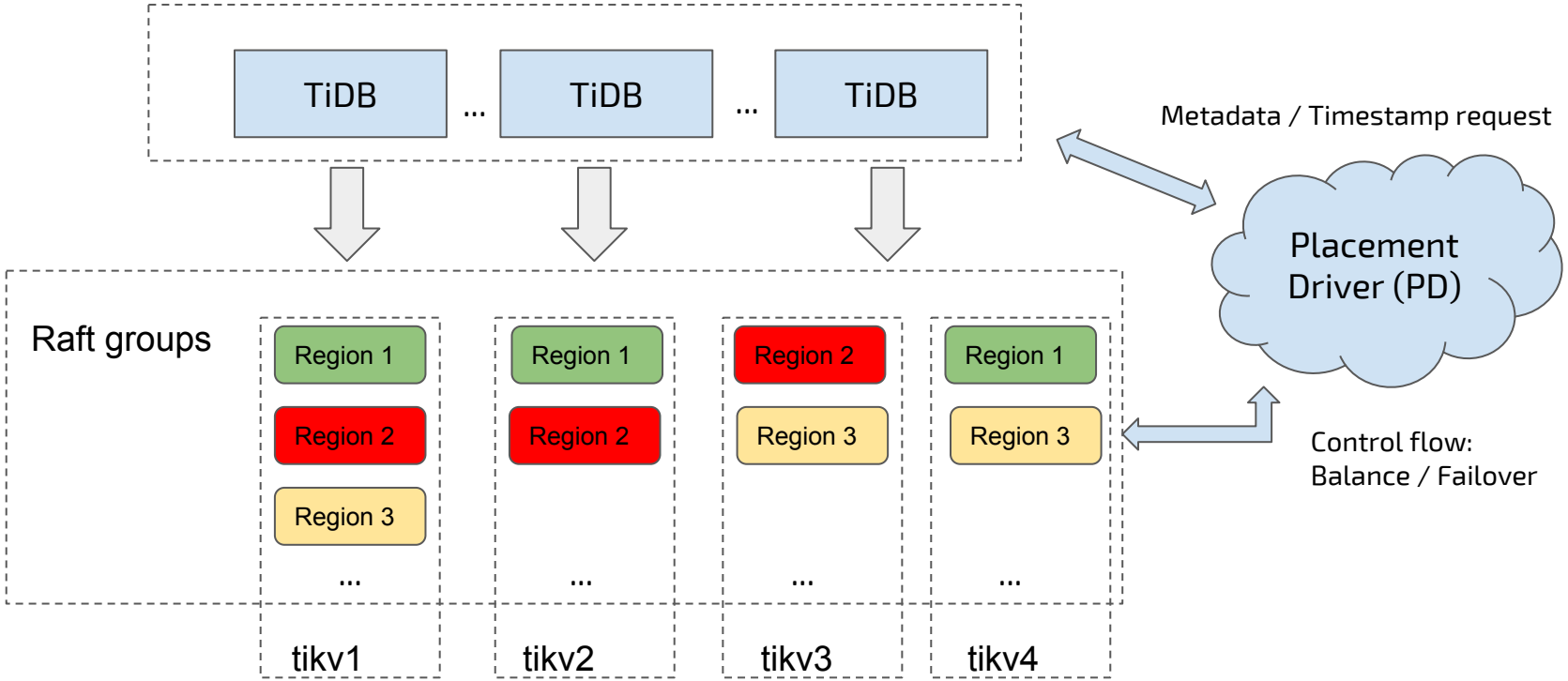
Summary

- **Advantage**
  - Simple
  - Implement *cross-row transaction* based on *single-row transaction* (BigTable)
  - Decentralized lock management
- **Disadvantage**
  - Centralized timestamp oracle.
  - More RPC

PingCAP

Part IV - Transaction in TiDB

Architecture



How to convert from SQL to Key-Value

id (primary)	*name(unique)*	*age(non-unique)*	*score*
1	Bob	12	99

SQL Model

index_type	*key*	*value*
primary_index	1	(Bob, 12, 99)
name(unique)	Bob	1
age(non-unique)	(12,1)	null

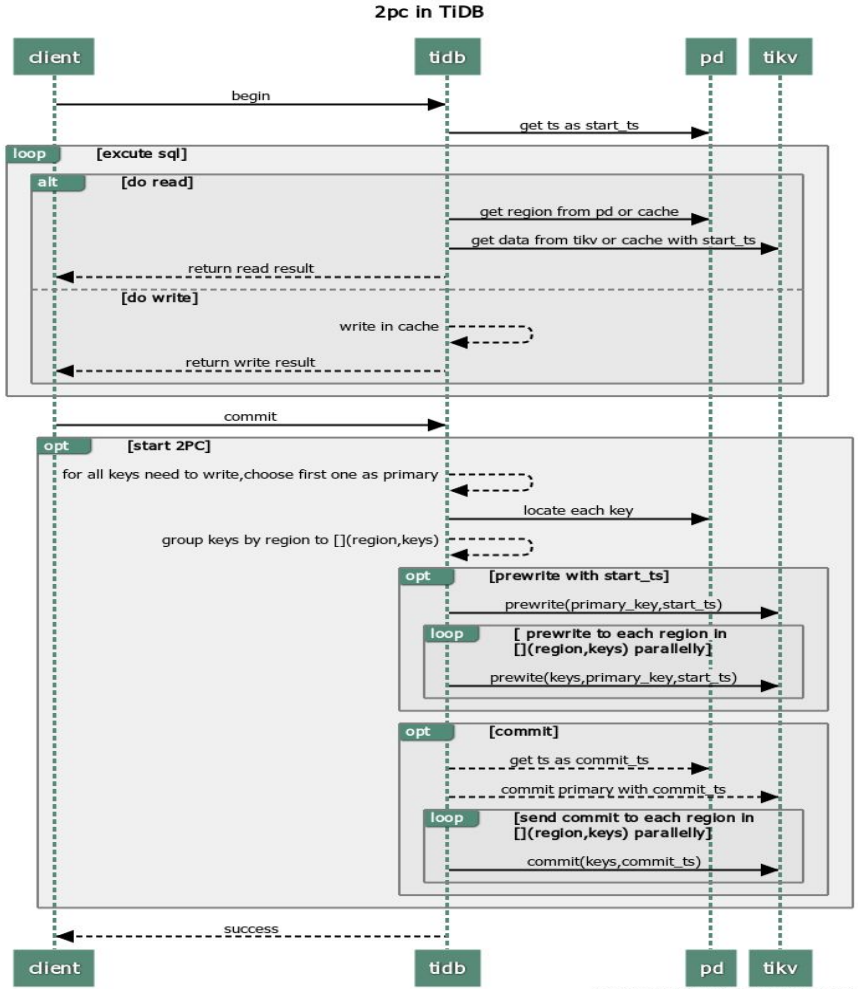
Key-Value Model

Column Families in RocksDB

Column Family	Key	Value
Data	key, start_ts	value
Lock	key	start_ts, primary_key, ttl
Write	key, commit_ts	start_ts [, short_value]

- Start_ts: timestamp when the transaction begins
- Commit_ts: timestamp get after prewrite, use in commit.
- Primary_key: key used to store the status of transaction.
- Short_value: value which is short.(with length<64 byte)

2 PC in TiDB

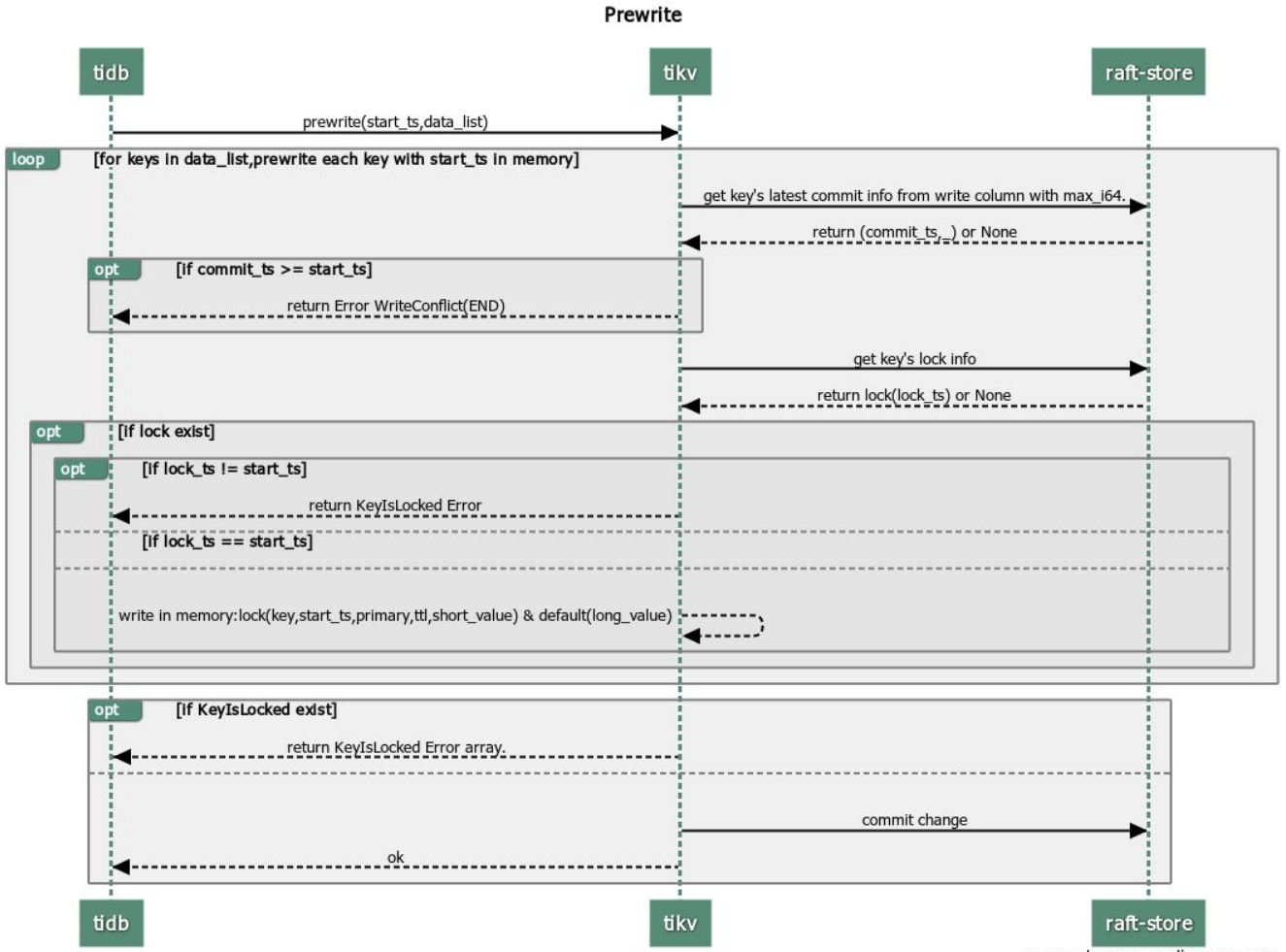


PingCAP

Prewrite

Errors:

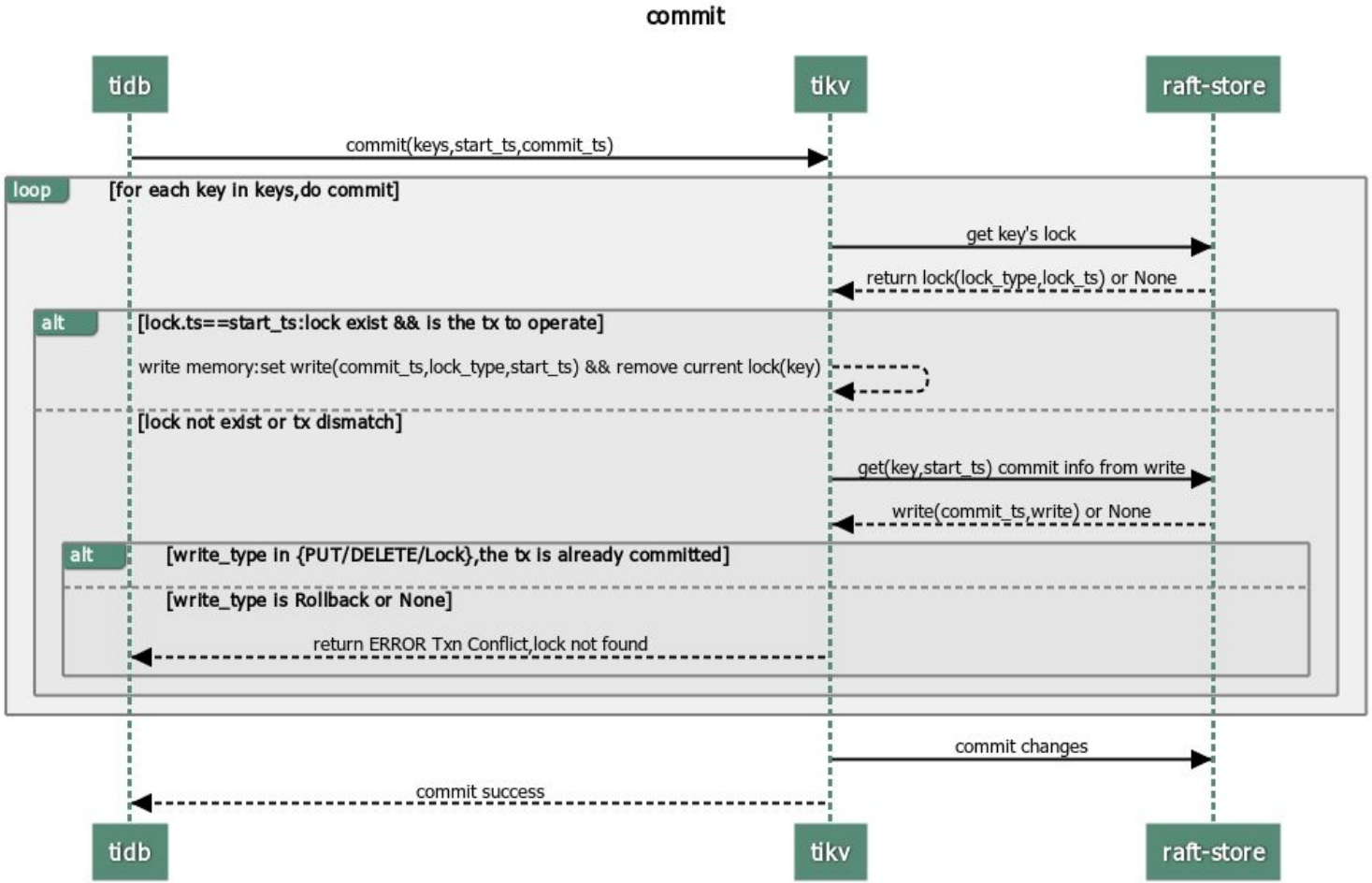
- *WriteConflict (newer version exist)*
- *KeyIsLocked*



Commit

Errors:

- *Lock Not Found*



Attentions for Using Optimistic Lock

session 1	***session 2***
begin;	**begin**;
select balance from T where id = 1; *// use the result of select* if balance > 100 { update T set balance = balance + 100 where id = 2; }	update T set balance=balance - 100 where id =1; update T set balance=balance - 100 where id = 2;
commit; *// auto retry*	**commit**;

Set @@global.tidb_disable_txn_auto_retry = 1

Attentions for large transaction

Due to the distributed, 2-phase commit requirement of TiDB, large transactions that modify data can be particularly problematic:

- Long duration
- More conflicts
- And so on ...

TiDB intentionally sets some limits on transaction sizes to reduce this impact:

- Each Key-Value entry is no more than 6MB
- The total number of Key-Value entries is no more than 300,000
- The total size of Key-Value entries is no more than 100MB

Attentions for small transaction

```
# original version with auto_commit
UPDATE my_table SET a='new_value' WHERE id = 1;
UPDATE my_table SET a='newer_value' WHERE id = 2;
UPDATE my_table SET a='newest_value' WHERE id = 3;

# improved version
START TRANSACTION;
UPDATE my_table SET a='new_value' WHERE id = 1;
UPDATE my_table SET a='newer_value' WHERE id = 2;
UPDATE my_table SET a='newest_value' WHERE id = 3;
COMMIT;
```

PingCAP

Thank You!

Any Questions ?

关注 PingCAP 官方微信

了解更多技术干货