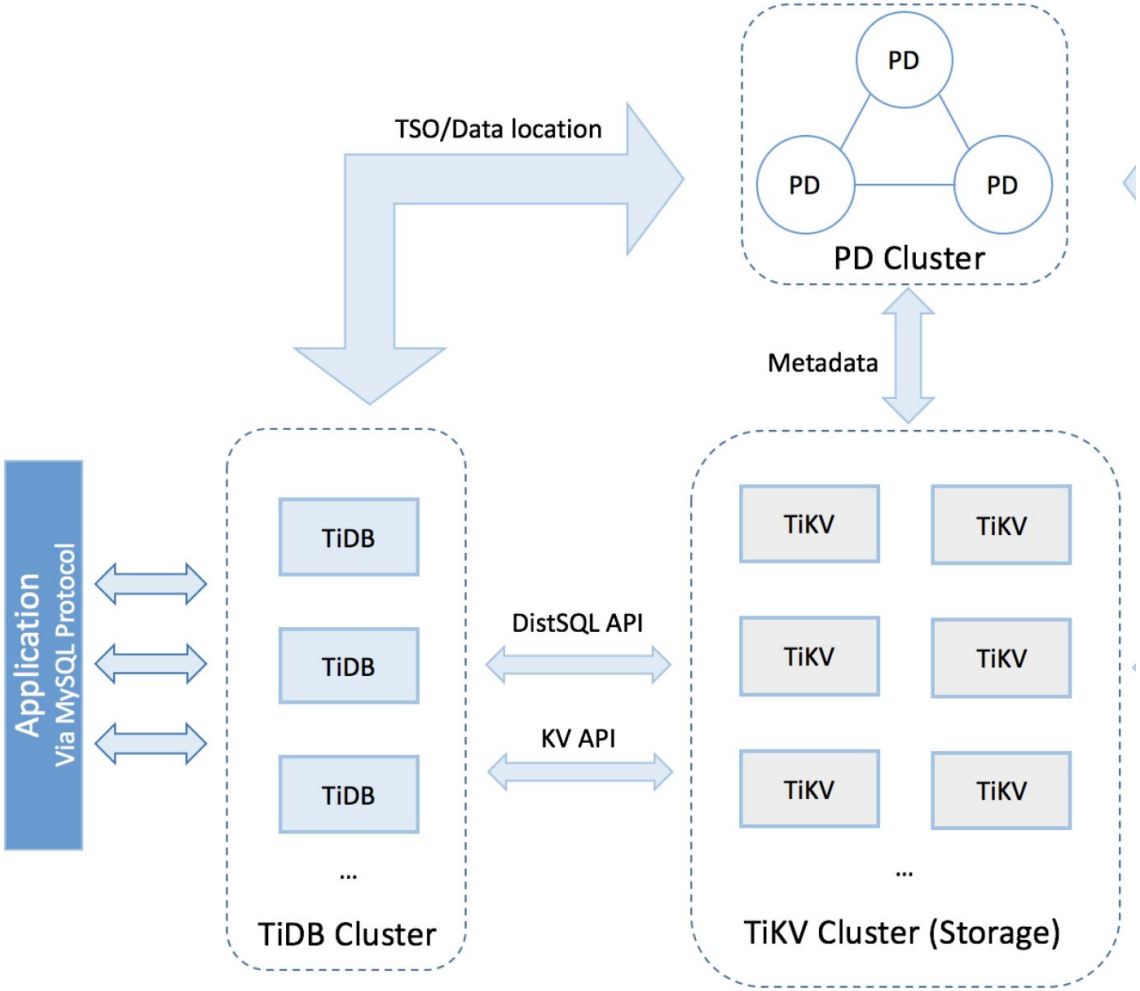
TiDB跨数据中心~~解决方案~~优化方案

林豪翔
庄天翼
朱贺天
屈鹏

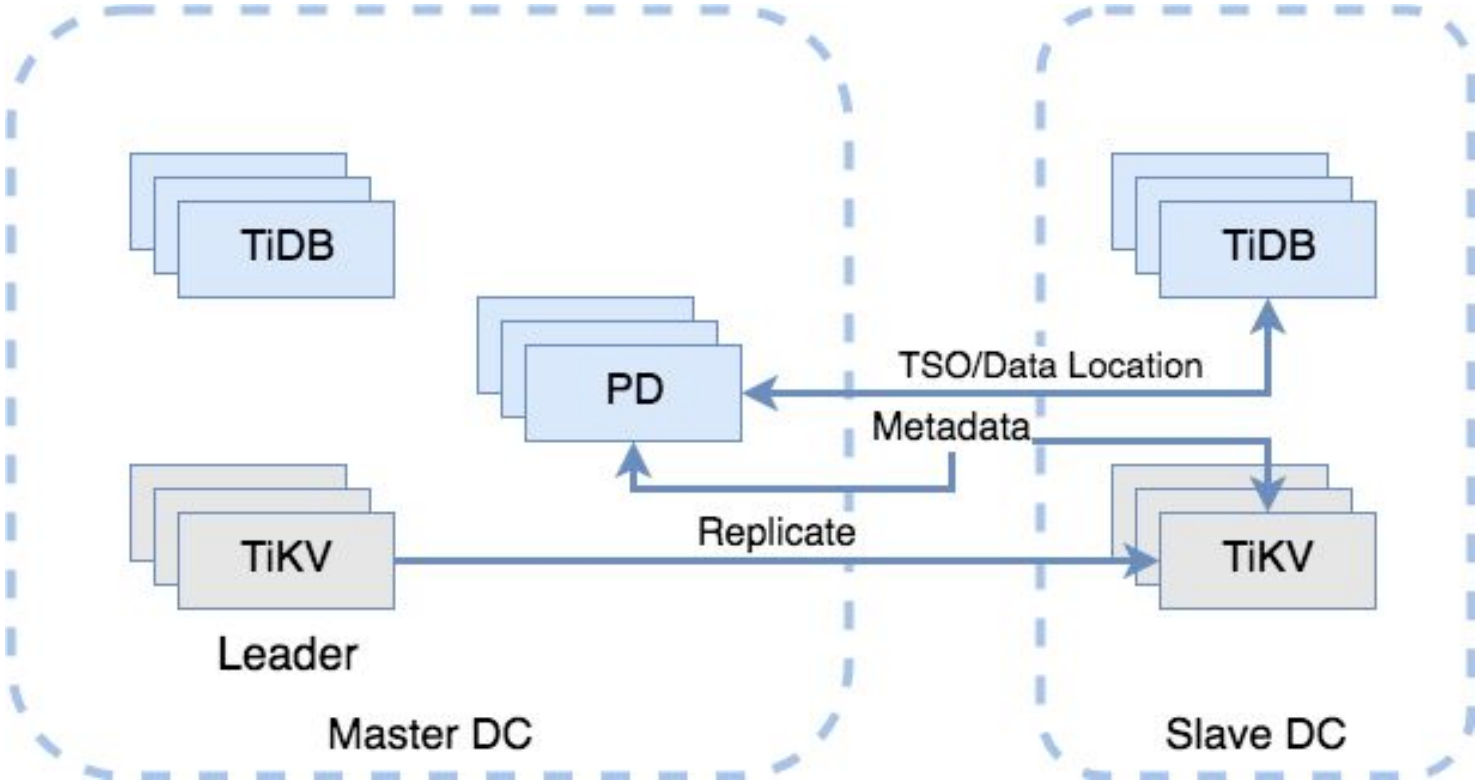
Background

- 两地三中心拓扑在行业内十分常见
- 业务上可能有多个地区的用户需要读取数据
- 跨地区的 RTT 较高
- 跨 DC 带宽昂贵

TiDB


PD
PD
PD
PD Cluster
TSO/Data location
Metadata
Application
Via MySQL Protocol
TiDB
TiDB
TiDB
...
TiDB Cluster
DistSQL API
KV API
TiKV
TiKV
TiKV
TiKV
TiKV
TiKV
...
TiKV Cluster (Storage)

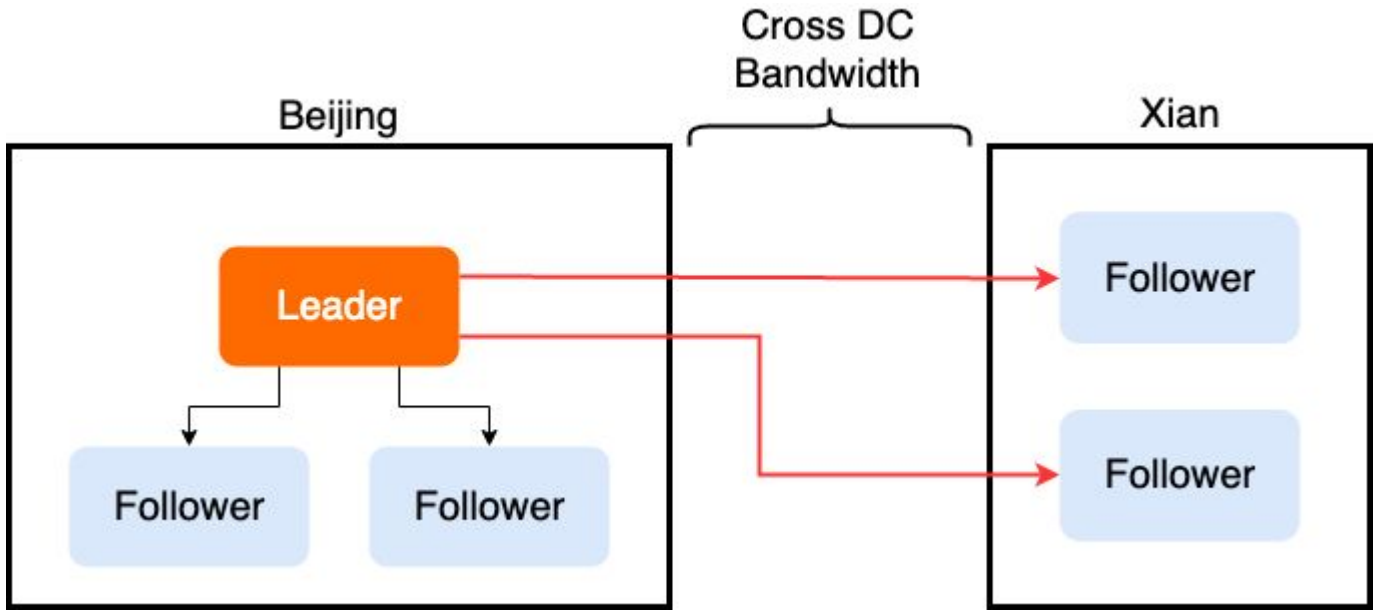
TiDB Cross DC



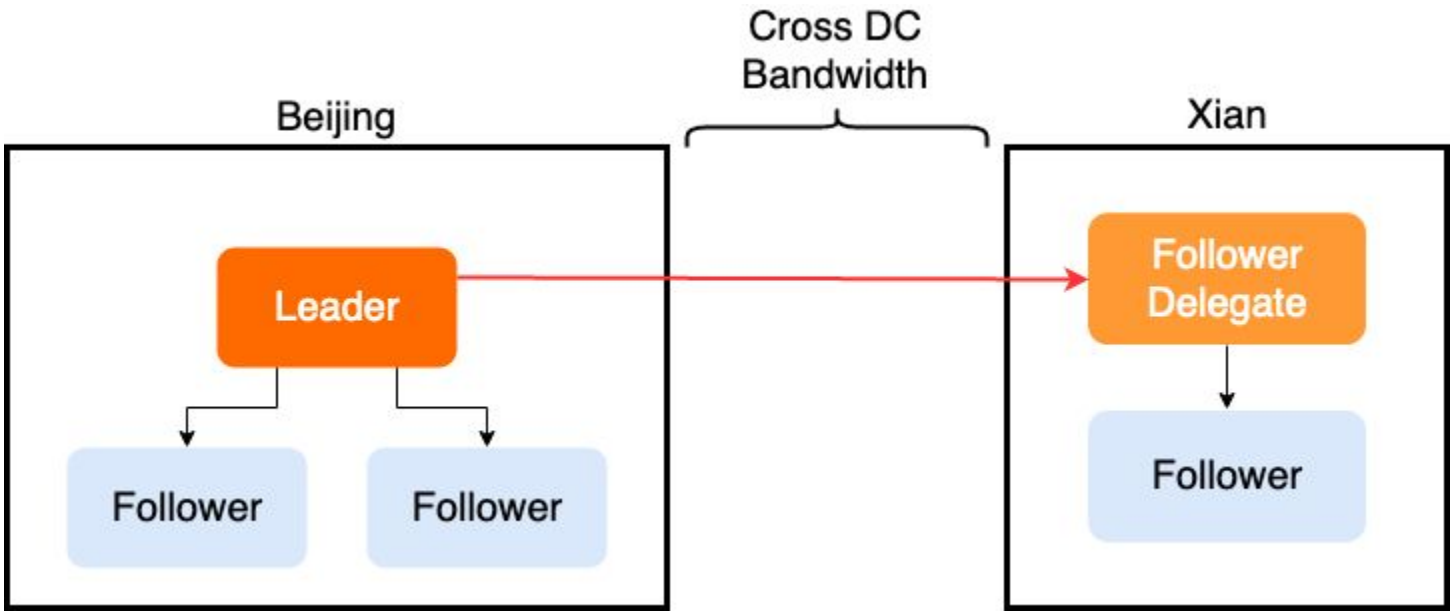
Improve

- **Follower Replication**
- Follower Read improvement

Current

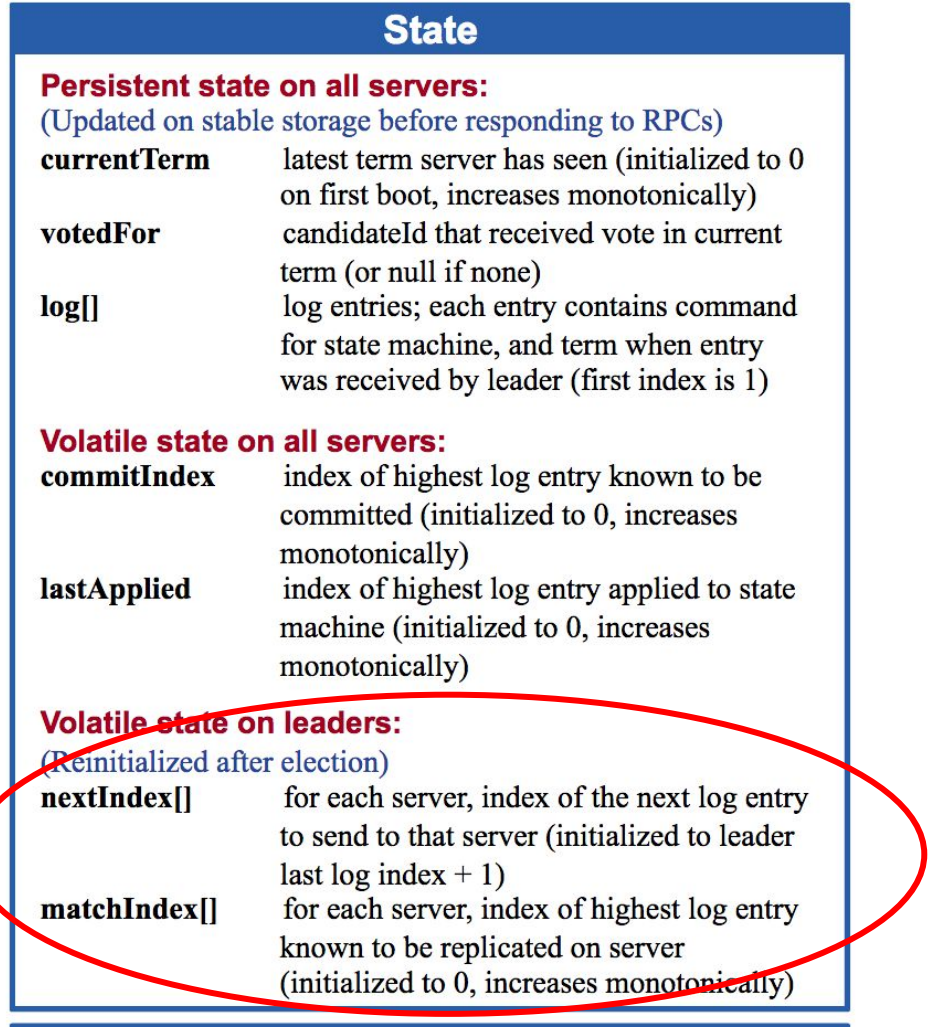

Cross DC
Bandwidth
Beijing
Xian
Leader
Follower
Follower
Follower
Follower

After our improvement



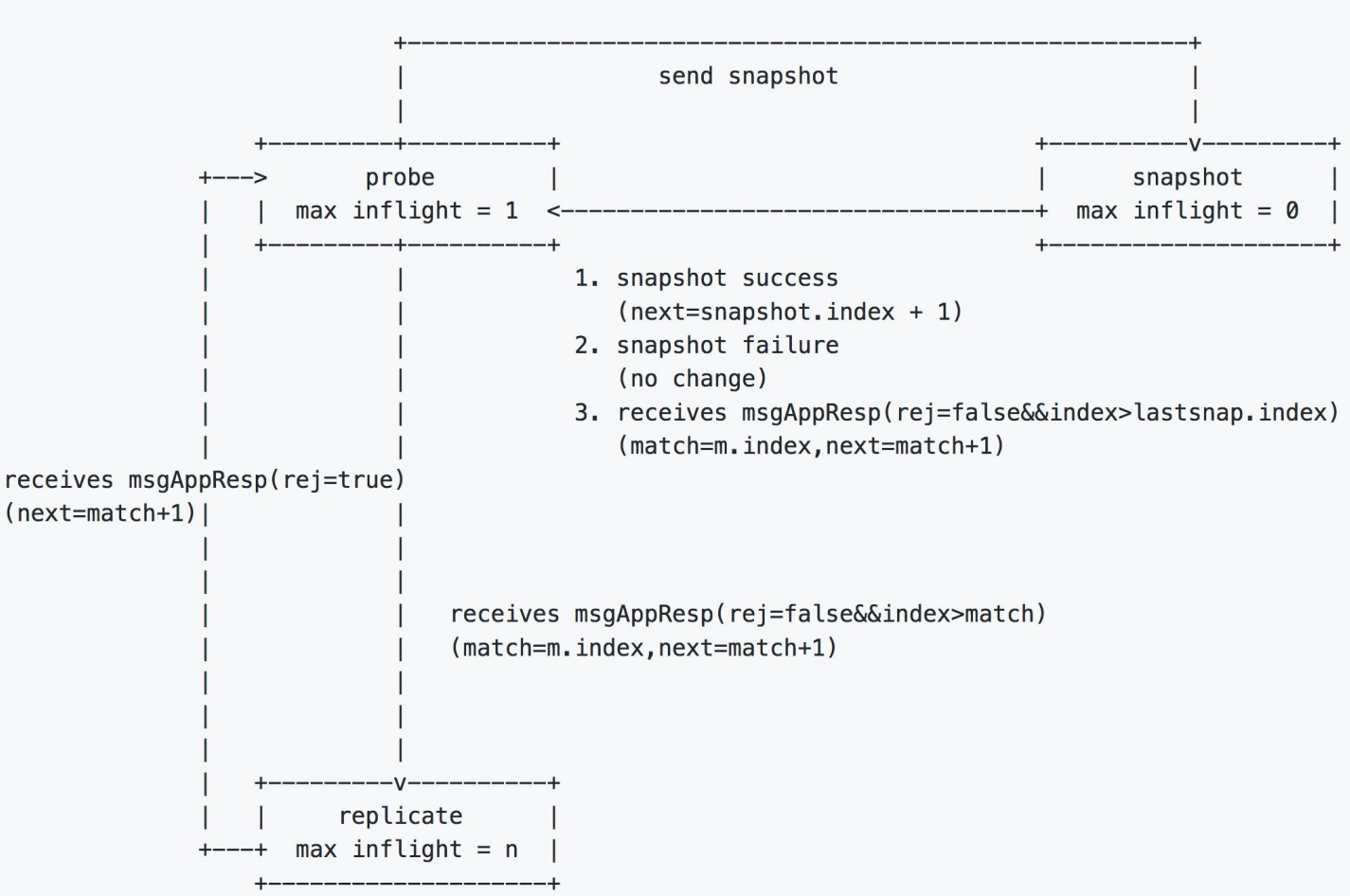
raft

Leader tracks progress of all followers.



etcd/raft

A progress is in one of the three state: `probe`, `replicate`, `snapshot`.



Solution 1: Maintain progress in per DC.

- Solution
  - Elect one delegate (or group leader) in per DC.
  - Maintain progress of followers/learners per DC on delegate.
  - Merge progress of different DC.
  - https://docs.google.com/document/d/1Sp9Tnc_nk_i0feTOgLGPT3jZYVFjfP7C6pwA-wuVHko
- Adventage
  - Leader only need to replicate messages to DC delegates.
  - DC delegate replicate messages to other peers.
- Drawback
  - The progress merge is complicated.

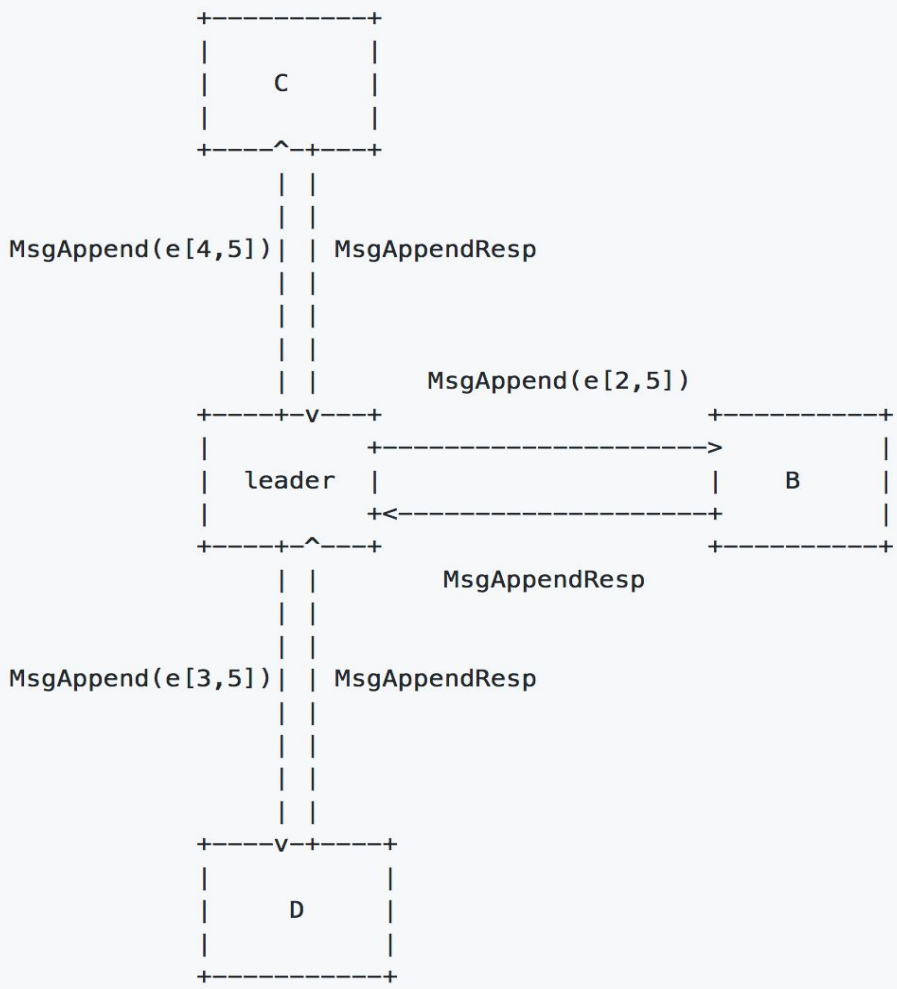
Solution 2: Introduce MsgBroadcast instead of MsgAppend

- Solution
  - Leader choose one DC delegate.
  - Leader send MsgBroadcast to DC delegate, with commission.

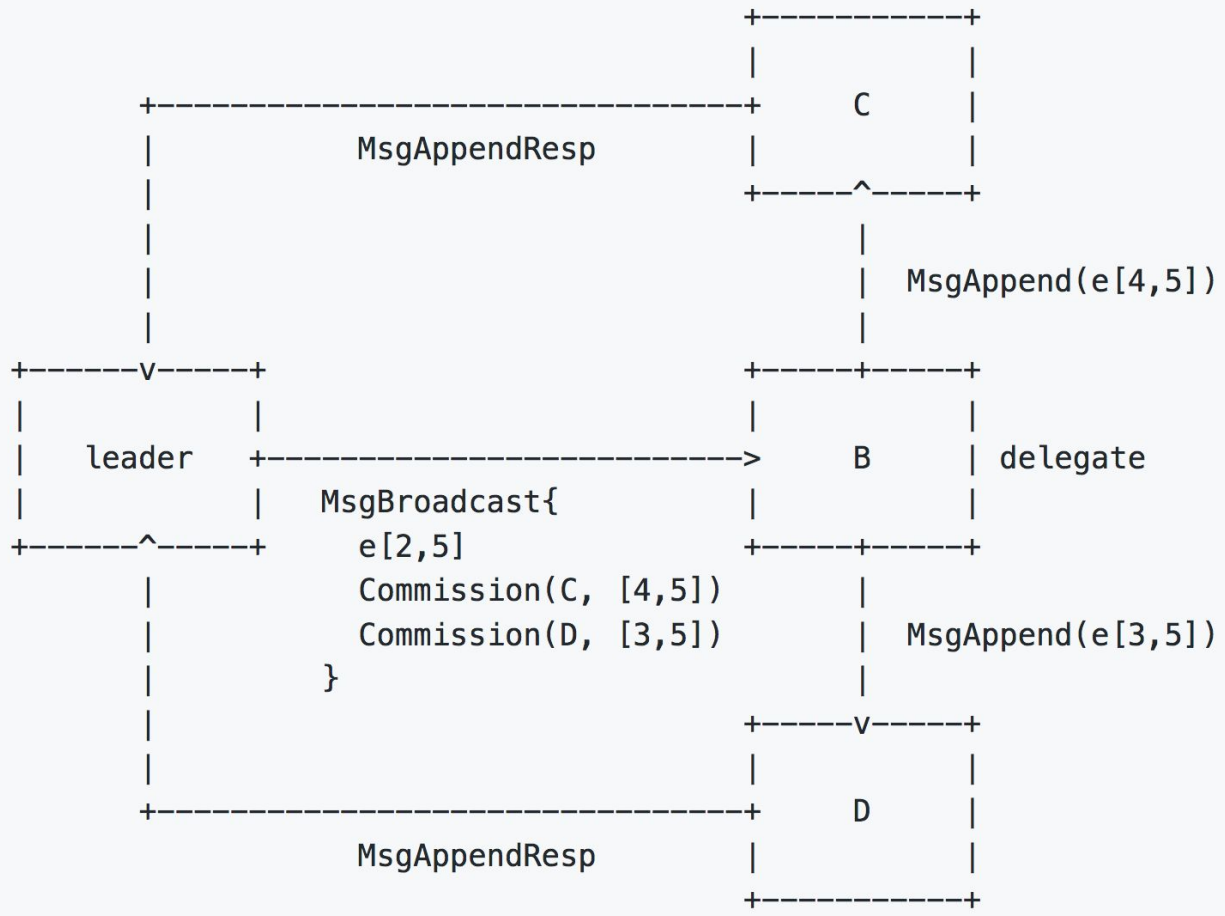
For example, in a 2DC based raft cluster with 5 nodes ABCDE where A, B, and C are at DC1 (master) and DE are at DC2.

Leader need to send:

- e[4, 5] to C
- e[2, 5] to B
- e[3, 5] to D



- e[x, y] stands for all the entries within an index range [x, y] (both inclusive)
- Commission(target, [x, y]) stands for a job that the delegate should send e[x, y] to the target



Handler of MsgBroadcast

1. Apply the original message (Always be Append or Snapshot), if failed, return failed.
2. Extract commisions from message, and extract contents data by last_index, then forward commisions to specific followers.
3. If any request failed due to stale progress info from leader, return reject.

```
message Commission {
    MessageType msg_type = 1;
    uint64 to = 2;
    // Same as `index` in `Message`
    uint64 last_index = 3;
    uint64 log_term = 4;
}
```

```
message Message {
    // Original message
    repeated Commission commissions = 16;
}
```

Choose the delegate

1. If all the members are requiring snapshots, choose the delegate randomly.
2. Among the members who are requiring new entries, choose the node satisfies conditions below :
   a. Must be recent_active
   b. The progress state should be Replicate but not paused
   c. The progress has the smallest match_index
3. Fallback to MsgAppend.

Interface modification

Easy to use on application layer (TiKV), just add a GroupConfig while initilizing raft.

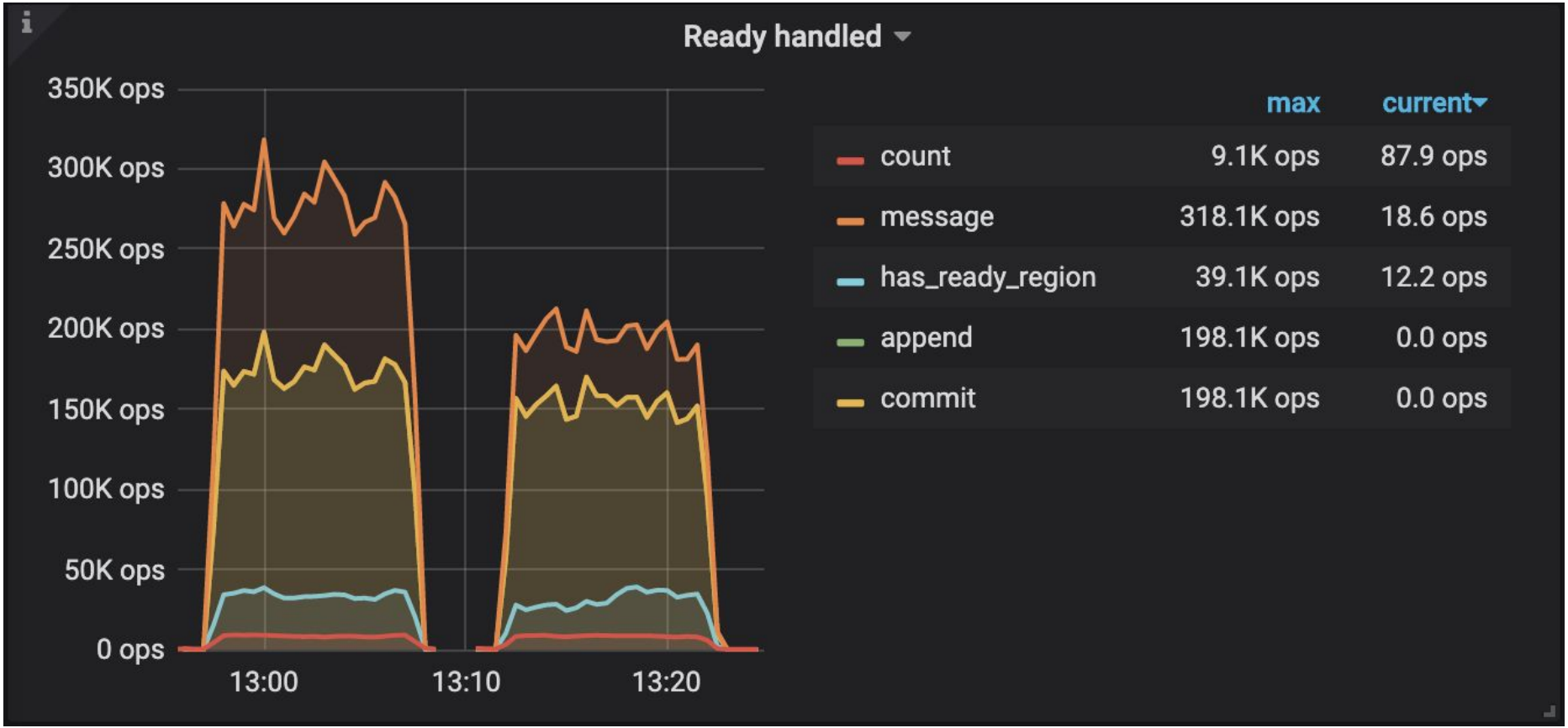
```rust
/// Configuration for distribution of raft nodes in groups.
/// For the inner hashmap, the key is group ID and value is the group members.
#[derive(Clone, Debug)]
pub struct GroupsConfig {
    strategy: ProxyStrategy,
    inner: HashMap<u64, Vec<u64>>,
}
```

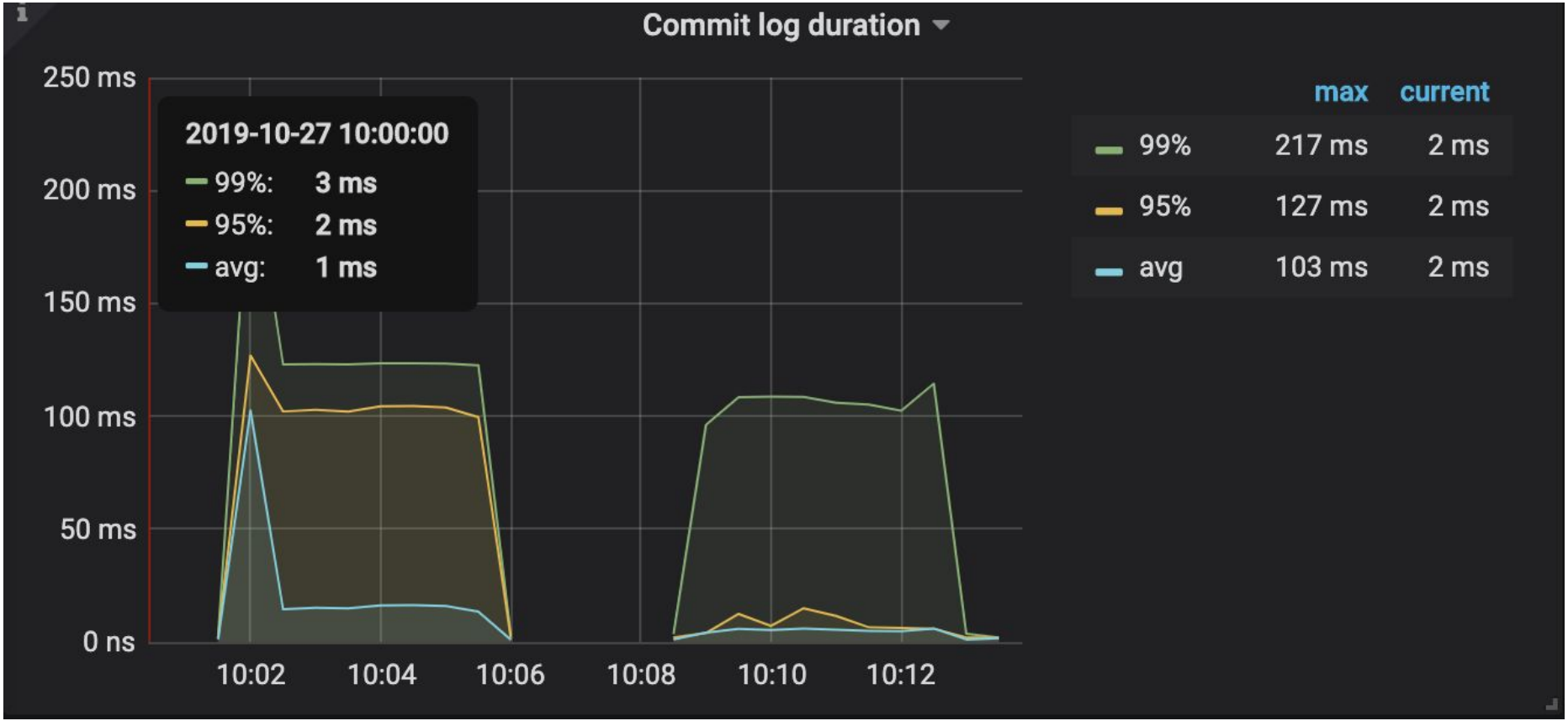
Introduce feature to TiKV

- Concept
  - Region: storage of a range of key value, the unit of raft.
  - Peer: raft peer of a region.
  - Store: A physical TiKV node.
  - **Location: DC Identity.**
- **Location Info**
  - **A global hashmap from store id to location.**
  - **Persistent and export in PD.**

Benchmark

- 2DC with 5 TiDB and 5 TiKVs (3+2, no learner).
- Message Size: 1MB


Ready handled
350K ops
300K ops
250K ops
200K ops
150K ops
100K ops
50K ops
0 ops
13:00
13:10
13:20
max
current
count 9.1K ops 87.9 ops
message 318.1K ops 18.6 ops
has_ready_region 39.1K ops 12.2 ops
append 198.1K ops 0.0 ops
commit 198.1K ops 0.0 ops


Commit log duration
250 ms
200 ms
150 ms
100 ms
50 ms
0 ns
10:02
10:04
10:06
10:08
10:10
10:12
2019-10-27 10:00:00
99%: 3 ms
95%: 2 ms
avg: 1 ms
max
current
99% 217 ms 2 ms
95% 127 ms 2 ms
avg 103 ms 2 ms

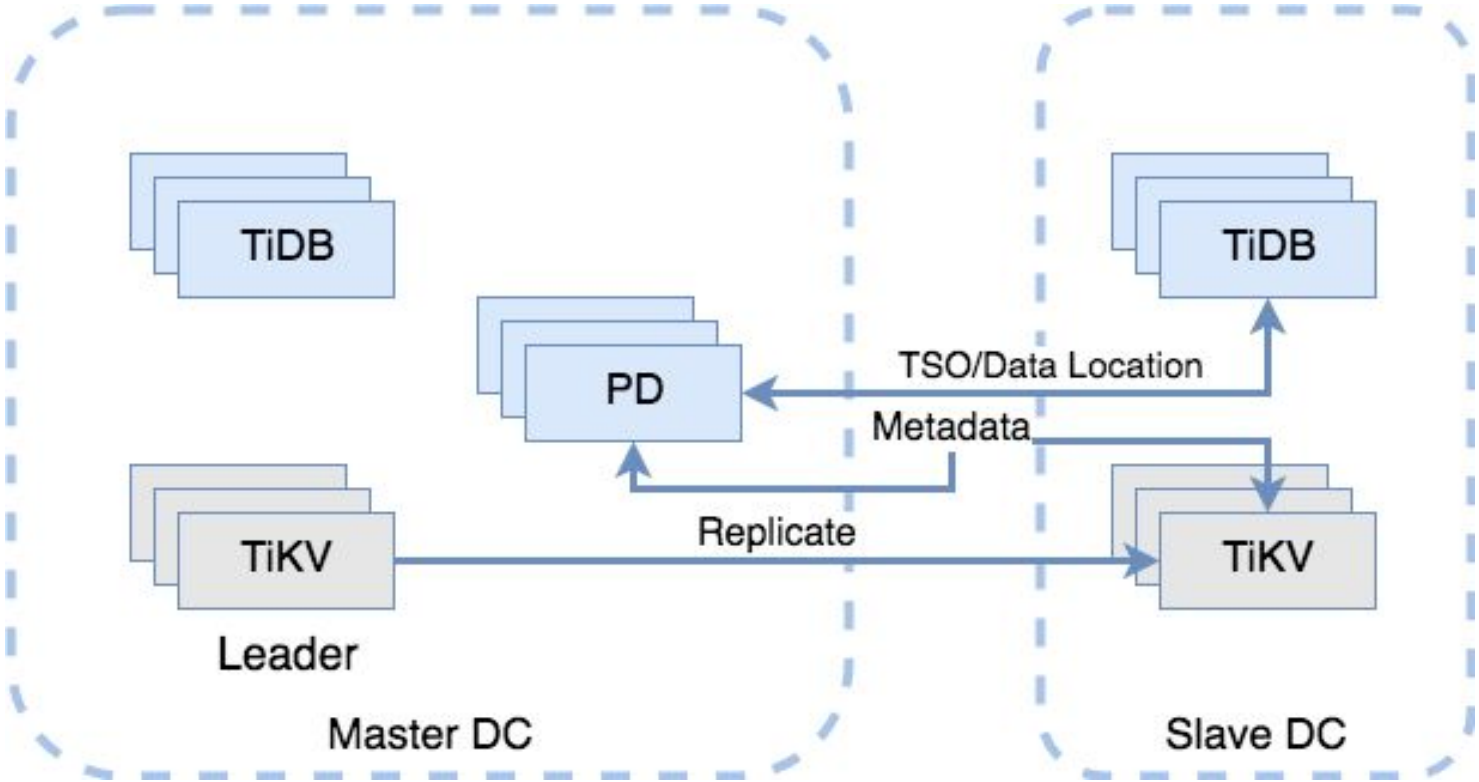
Drawback

1. Only two layer replication chain is implemented.
2. Replication latency may increase.

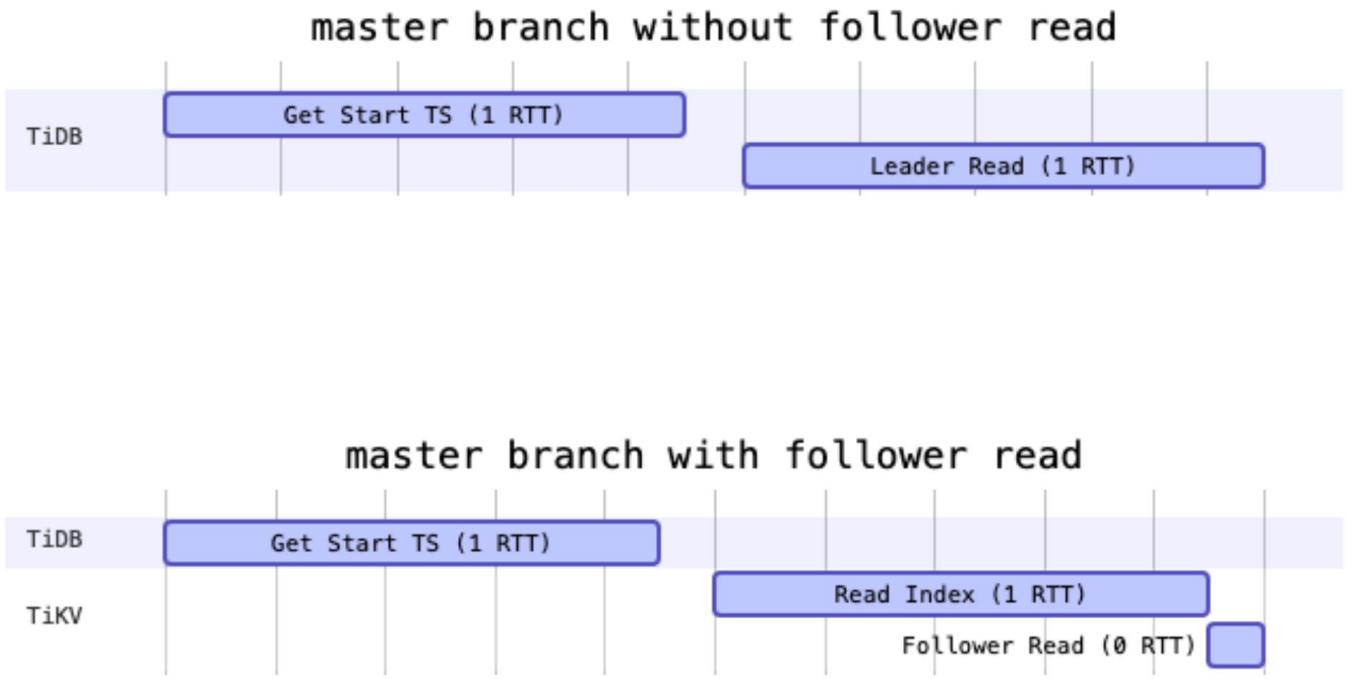
Improve

- Follower Replication
- **Follower Read Improvement**

TiDB Cross DC



Motivation



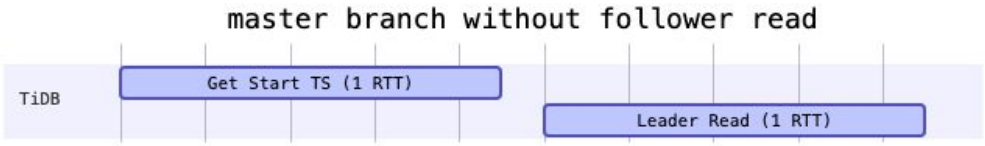
Reduce the latency

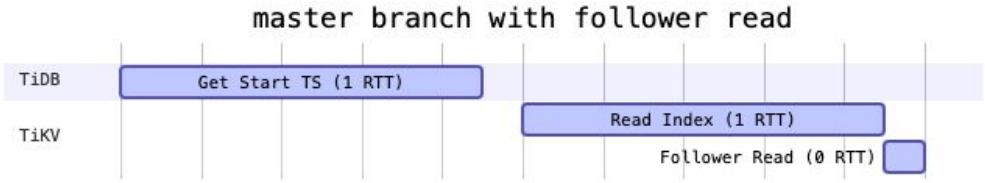
Is read_index required?

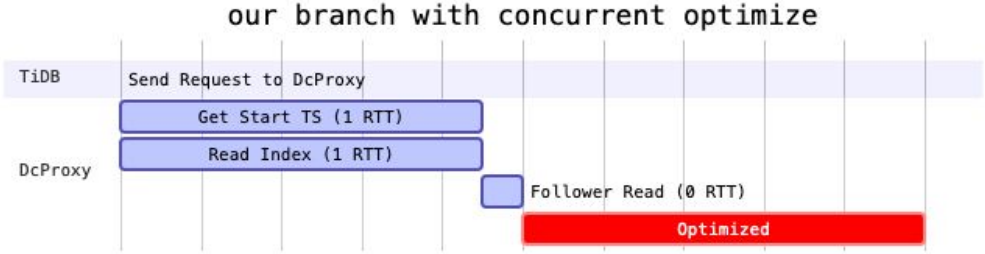
- In TiKV, when we receive a read request with a start timestamp, we cannot determine whether all messages under the timestamp is applied.
- Can we maintain an applied_timestamp in every region peer in TiKV?
- No, because a message with higher raft log_index may have lower commit timestamp due to network delay.
- ~~BTW: for spanner, the answer is Yes :(~~

Is read_index required to happen after get_tso?

- No, read_index only required to happen after user invoke the read request.
- **We can concurrent read_index and get_tso**


master branch without follower read
TiDB
Get Start TS (1 RTT)
Leader Read (1 RTT)


master branch with follower read
TiDB
TiKV
Get Start TS (1 RTT)
Read Index (1 RTT)
Follower Read (0 RTT)


our branch with concurrent optimize
TiDB
DcProxy
Send Request to DcProxy
Get Start TS (1 RTT)
Read Index (1 RTT)
Follower Read (0 RTT)
Optimized

Hackathon Implementation

- Introduce a new service DCProxy in tikv.
  - As a stateless service.
  - Only invoke PD and tikv public API.
- Add GetTSAndCommitIndexes API on DCProxy
  - Call get_tso and multiple read_index concurrent
- Modify TiDB coprocessor client
  - Call GetTSAndCommittedIndexes directly instead of get ts then read local tikv

A better implementation

- Call get_tso and pre read_index concurrently in TiDB, and cache the read_index.
- Pass the read_index in Context struct in follower read.
- If TiKV found valid read_index in request context, it will directly read its local storage.

Benchmark

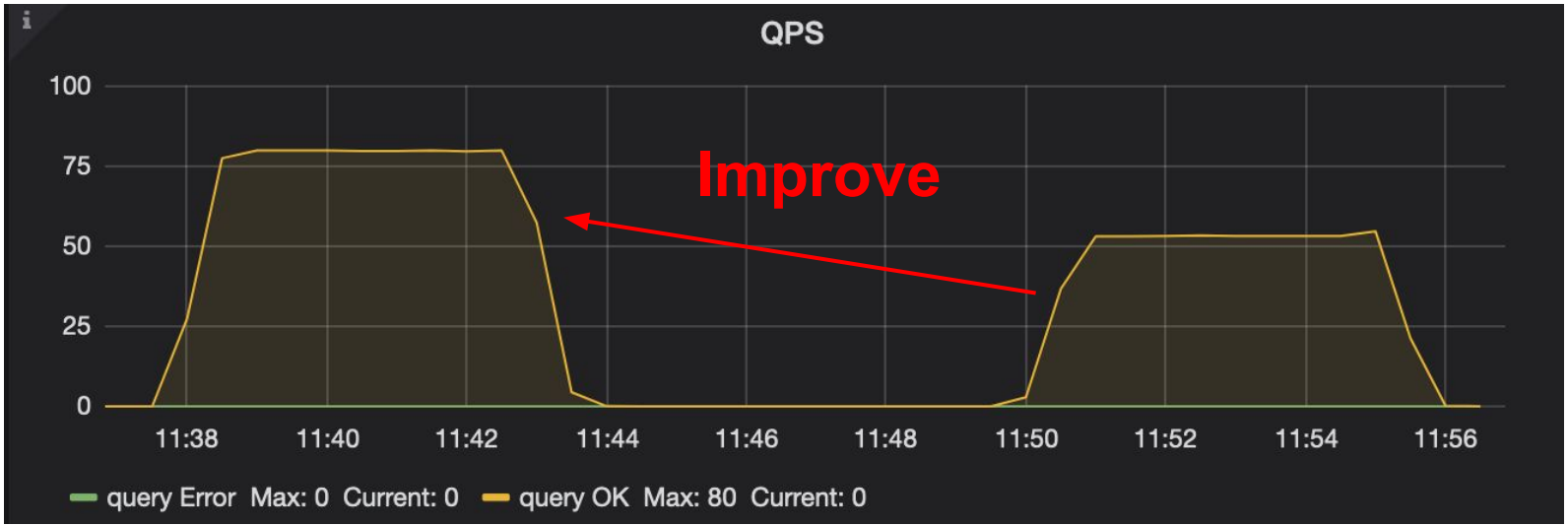
- 2 DC with 5 TiDB and 5 TiKV (3+2)
- Exactly 100ms latency between DC (mock by network tool)
- sysbench run on slave DC.

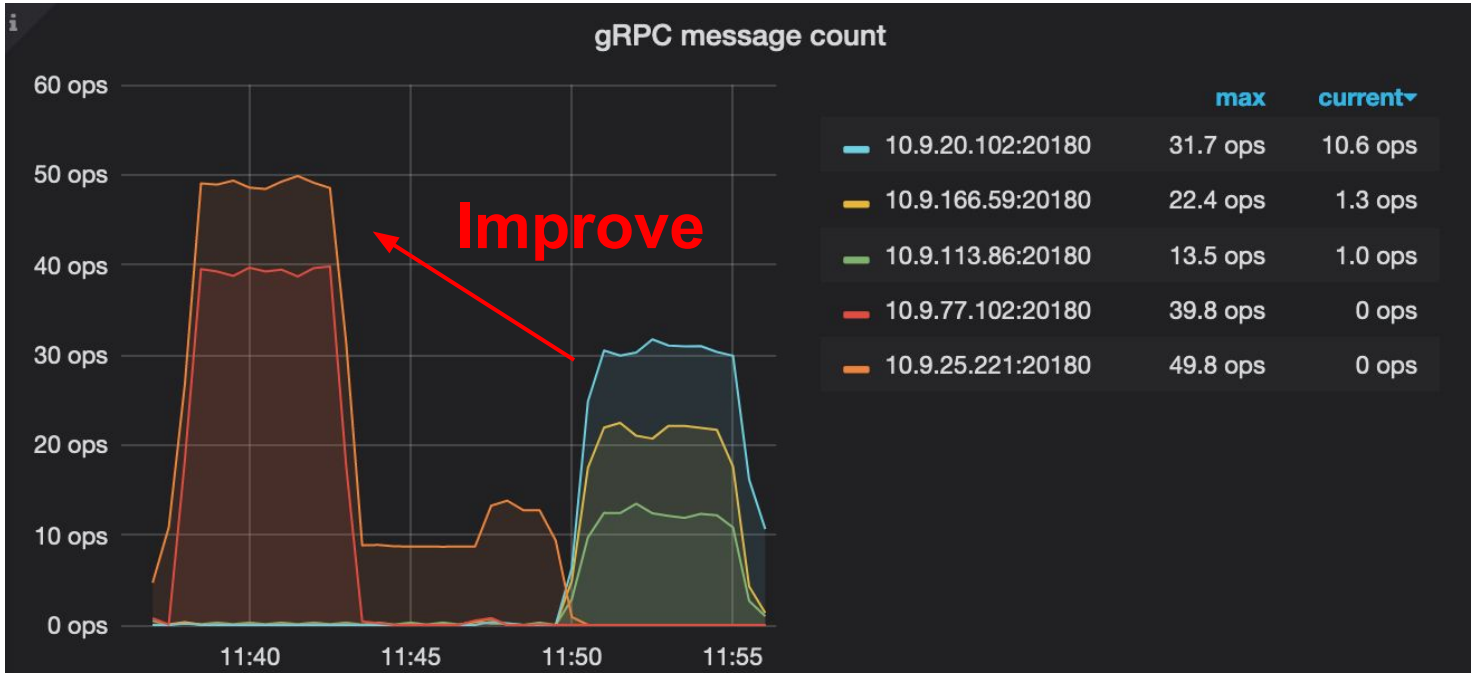
```
MySQL [sbtest]> select count(*) from sbtest1 where id = 2;
+----------+
| count(*) |
+----------+
|        1 |
+----------+
1 row in set (0.20 sec)
```


Improve

```
MySQL [sbtest]> select count(*) from sbtest1 where id = 2;
+----------+
| count(*) |
+----------+
|        1 |
+----------+
1 row in set (0.10 sec)
```


Duration
Improve
300 ms
200 ms
100 ms
0 ns
11:38
11:40
11:42
11:44
11:46
11:48
11:50
11:52
11:54
11:56
80
avg


QPS
100
75
50
25
0
Improve
11:38
11:40
11:42
11:44
11:46
11:48
11:50
11:52
11:54
11:56
query Error Max: 0 Current: 0
query OK Max: 80 Current: 0


gRPC message count
60 ops
50 ops
40 ops
30 ops
20 ops
10 ops
0 ops
11:40
11:45
11:50
11:55
Improve
max
current
10.9.20.102:20180 31.7 ops 10.6 ops
10.9.166.59:20180 22.4 ops 1.3 ops
10.9.113.86:20180 13.5 ops 1.0 ops
10.9.77.102:20180 39.8 ops 0 ops
10.9.25.221:20180 49.8 ops 0 ops

Drawback

- For interactive readonly transaction, our implementation can only optimize the latency from n+1 RTT to n RTT
  - Can be optimized to 1 RTT by maintain an safe_timestamp and sync periodically.
  - If timestamp order is consistent with raft commit_index order, then safe_timestamp is applied_timestamp.

Further Read: Spanner 0 RTT?

- True Time
  - Implement by atomic clock.
  - Get Timestamp from local DC.

Stale Read

Pre get_tso periodly, and use proper timestamp by specific stale threashold.

Reference

Follower Replication RFC: https://github.com/tikv/rfcs/pull/33/

Follower Replication Implementation: https://github.com/tikv/raft-rs/pull/249

TiKV Hackathon Branch:
https://github.com/TennyZhuang/tikv/tree/hackathon-2019

TiDB Hackathon Branch:

https://github.com/TennyZhuang/tidb/tree/hackathon-2019

Q & A