Cursor Implementation in Partitioned Transaction FoundationDB

Xiaoge Su

Apple Inc.

March 14, 2022

Table of Content

Introduction

- FoundationDB is a key-value database with ACID support.
- Internally, the journal is stored in `TLog`, and a `cursor` is used by the `Storage Server` to extract the commits from `TLogs`.
- When *storage team* is introduced, the `cursor` needs to be redesigned to support this new feature.

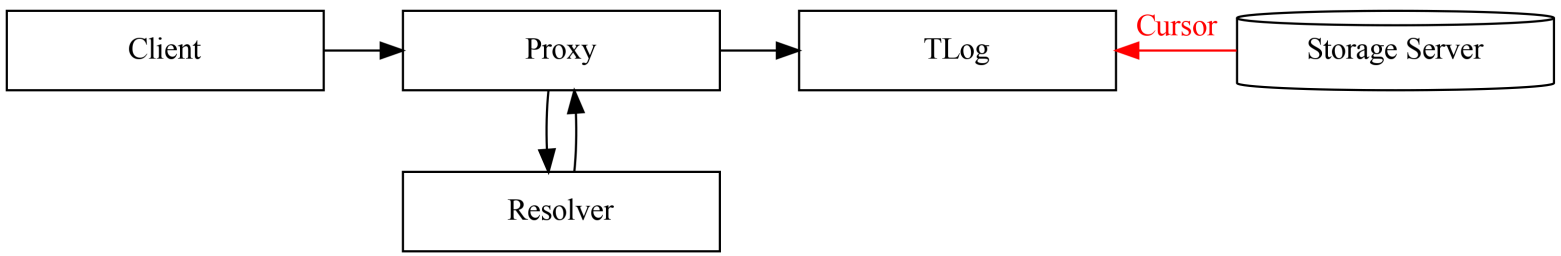
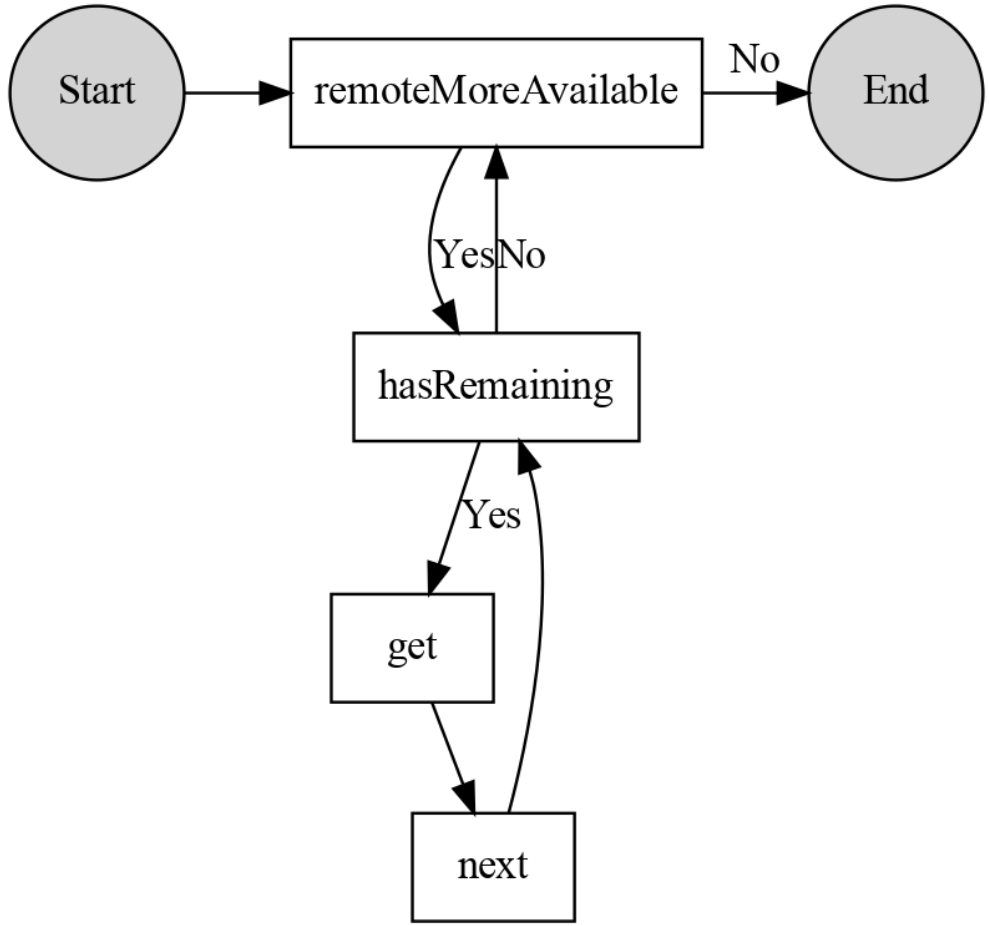


Figure: Overall design



Base Class Implementation

A base class `ptxn::PeekCursorBase`[1] is implemented with methods:

- `remoteMoreAvailable`
- `hasRemaining`
- `get`
- `next`
- `reset`

An iterator interface is also implemented for ranged `for` loop.

[1] `fdbserver/ptxn/TLogPeekCursor.actor.h`

Requirements: Single Storage Team

- For a given storage team, pulls serialized data from the corresponding `TLog` to local `Storage Server`.
- Allows the `Storage Server` iteratively access received data as `MutationRefs`.
- The data should be re-iterable.

Implementation: Single Storage Team

To support storage team peek cursor, `ptxn::StorageTeamPeekCursor`[1] is introduced.

- Inherited from `ptxn::PeekCursorBase`
- Requires
  - Start version
  - Storage team ID
  - `TLogInterface`

[1] `fdbserver/ptxn/TLogPeekCursor.actor.h`

Requirements: Multiple Storage Teams

Commit Version		1	2	3	4	5	6	7	8	...
Storage Teams	A	*		*	*		*		*	...
	B	*		*		*	*	*		...
	C		*	*	*			*	*	...

* means the storage team has `MutationRefs` for the specific commit.

- Each commit has its commit version.
- One or more storage team(s) will be involved in the commit.
- Each storage team has its own storage team version.

Broadcasting Model

- A commit might **not** impact **all** storage teams.
- In the broadcasting model, all storage teams will be informed when a new commit is coming in.
  - All storage teams are sharing the same storage team version.
  - All storage servers are sharing the same commit version/storage team version.
  - For each commit, the storage server will peek from **all** storage teams it is subscribing.
- This is done by sending the storage version (without `MutationRef`) to the uninvolved teams.

Broadcasting Model II

Commit Version		1	2	3	4	5	6	7	8	...
Storage Teams	A	*	.	*	*	.	*	.	*	...
	B	*	.	*	.	*	*	*	.	...
	C	.	*	*	*	.	.	*	*	...

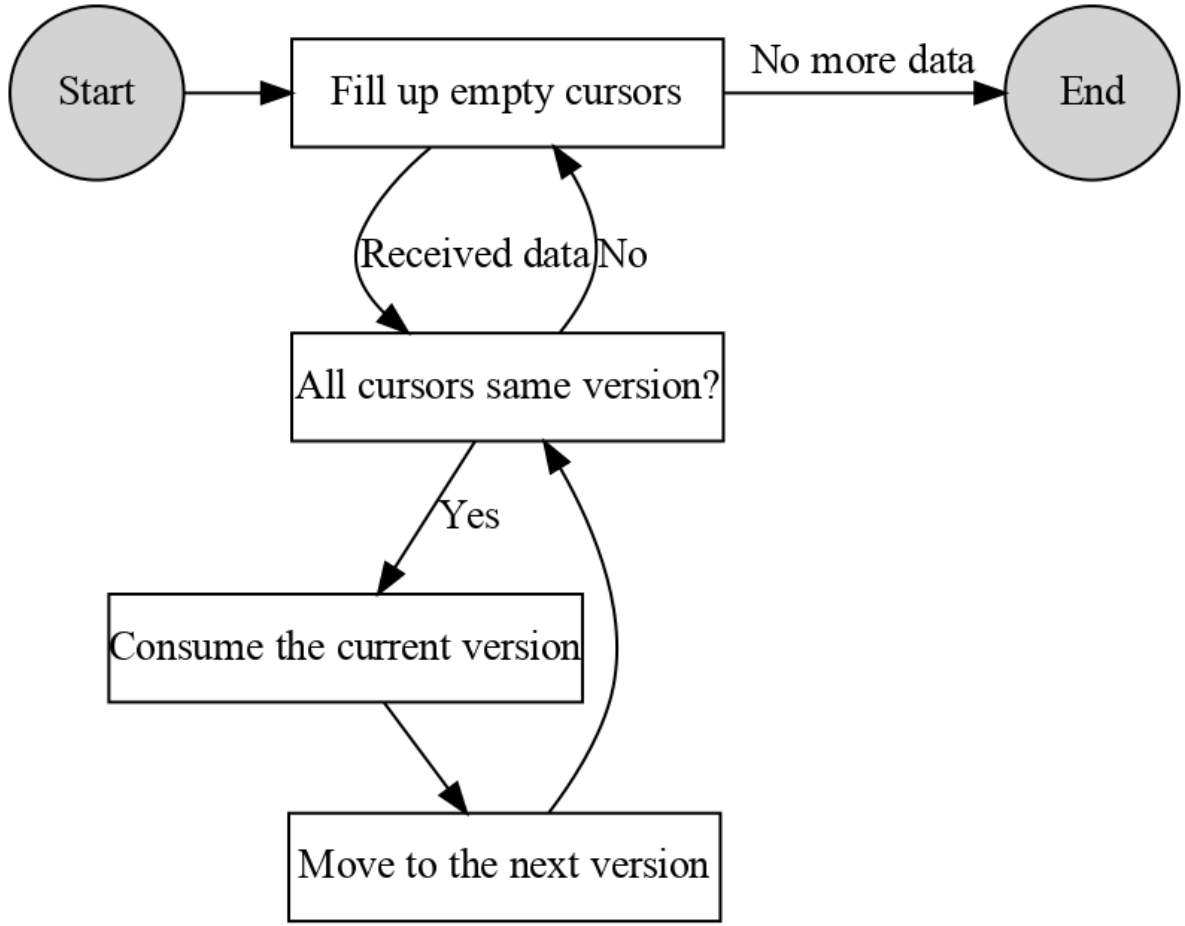
* means the storage team has `MutationRefs` for the specific commit.

. means the storage team receives an `EmptyVersionMessage`.

Requirements: Multiple Storage Teams (Broadcasting)

- The `Storage Server` should be able to subscribe from multiple storage teams.
- The received `MutationRefs` should be ordered by the version and subsequence when iterating.

Figure: Multiple Storage Teams (Broadcasting)



Note in the storage server context, commit version is always used.

Implementation: Multiple Storage Teams (Broadcasting)

```
while !empty(container) do
    cursor ← pop(container);
    mutationRef ← get_mutationref(cursor);
    yield(mutationRef);
    if get_version(cursor) == version then
        push(container, cursor);
    else
        nextVersion(cursor);
        if !hasRemaining(cursor) then
            push(emptyCursors, cursor);
        end
    end
end
```

Result: `MutationRefs` in the commit

Algorithm 1: Consume the `MutationRefs` in a commit

Implementation: Multiple Storage Teams (Broadcasting)

In namespace `ptxn::details`

`StorageTeamIDCursorMapper` [1]

Stores the storage team IDs and its corresponding `StorageTeamPeekCursor`.

`CursorContainerBase` [1]

For the *current* version, stores the cursors those still have `MutationRefs` to be consumed.

`OrderedCursorContainer`

Yields `MutationRefs` in subsequence order.

`UnorderedCursorContainer`

Yields `MutationRefs` in storage team ID order.

[1] `fdbserver/ptxn/TLogPeekCursor.actor.h`

Implementation: Multiple Storage Teams (Broadcasting)

In namespace `ptxn::details`

`BroadcastedStorageTeamPeekCursorBase` [1]

Base class supports iterating over multiple storage teams.

`BroadcastedStorageTeamPeekCursor_Ordered`

Uses `OrderedCursorContainer` as container.

`BroadcastedStorageTeamPeekCursor_Unordered`

Uses `UnorderedCursorContainer` as container.

[1] `fdbserver/ptxn/TLogPeekCursor.actor.h`

Requirements: Mutable Multiple Storage Teams (Broadcasting)

- The `Storage Server` should be able to subscribe/unsubscribe the storage teams *on the fly*.
- The storage team information is provided by a special storage team, *private mutation team*.
- The format of data in the private mutation team:

  Key Prefix + Storage Server ID
  Value Private Mutation Team ID, {Storage Team ID, ...}
- Every commit related to storage team change will only contain `MutationRef` in private mutation team.

Mutable Multiple Storage Teams Model (Broadcasting)

The cursor of the storage server will monitor the private mutation team for relevant Prefix + Storage Server ID key.

- New storage team added
  - Create corresponding `StorageTeamPeekCursor` object
  - Set its version next to the cursor version.
  - Mark it as empty.
- Existing storage team removed
  - Remove the storage team ID from the list along with the cursor.

Implementation: Mutable Multiple Storage Teams (Broadcasting)

In namespace `ptxn::details`

`MutableTeamPeekCursor` [1]

- Implement the mutable team peek cursor base on ordered/unordered boradcasted storage team peek cursor.

[1] `fdbserver/ptxn/MutableTeamPeekCursor.actor.h`

Requirements: Re-iterable cursor

Re-iterable cursor: before `remoteMoreAvailable` is called, the content of the cursor can be iterated repeatly.

- Single storage team cursor: only uses a deserializer, which is re-iterable already.
- Broadcasted storage team cursor: after the `remoteMoreAvailable`, record the initial state of the cursor container.
- Mutable storage team cursor: Also need to temporarily store the deleted cursors.

The state of the cursor is able to be `reset` base on previous stored state.

Further Developments

- Mutable team peek cursor without version broadcasting.
- Decouple the RPC part and the iterate part.