Next Generation of FoundationDB Serialization

Xiaoge Su

Apple Inc.

February 25, 2022

Table of Content

1. Introduction

2. Motivation

3. Design

4. Implementation

Introduction

- FoundationDB is a key-value database with ACID support.
- Internally, the key-value pairs, or `MutationRef`s, are serialized before transported between components.

$$\left.\begin{array}{c}\texttt{MutationRef}\\ \texttt{MutationRef}\\ \dots\\ \texttt{MutationRef}\end{array}\right\}\xrightarrow{\text{Serialization}}\text{Binary data}$$

$$\text{Proxy}\xrightarrow{\text{Serialized data}}\text{TLog}\xrightarrow{\text{Serialized data}}\text{Storage Server}$$

- Recently, storage teams are introduced to FoundationDB, demanding new requirements for the serialization module.

Motivation

Requirement for the serializer that supports storage teams:

- Serializes `MutationRefs` in a storage team
  - Storage team version
  - `MutationRefs`, together with subsequence number
- Proxy: Supports serializing data among multiple storage teams simultaneously
- TLog: Batches multiple versions of serialized data of the same storage team

Motivation

The issues on the current serializer (`LogPushData`):

- Major modifications are needed to support the storage team.
- The serializer does not have a well-defined deserializer.
- Tightly bound to `flow` and `LogSystem`.
- Weakly typed.
- No tests included.

A re-implementation of serializer should address these issues.

Two-level serialization

- Proxy requires packing multiple `MutationRefs` in *one* version.
- TLog requires packing multiple versions.

The requirements imply a two-level serialization model.

Two-level serialization - Level 1

Within the storage team version:

Head	Item	Item	...	Item

The head contains the version information, whereas the items are serialized `MutationRefs`.

Two-level serialization - Level 2

Batched storage team versions:

Head	VersionedItem	VersionedItem	...	VersionedItem

The head contains storage team information, whereas each versioned item is the pack of `MutationRefs` with version information, shown in the previous slide.

Two-level serialization Implementation

- Abstract header: `MultipleItemHeaderBase`[1]
  - `numItems`
  - `length`
- Level 1 serialization: `HeaderedItemsSerializer`[1]
- Level 2 serialization: `TwoLevelHeaderedItemsSerializer`[1]
- Level 2 deserialization: `TwoLevelHeaderedItemsDeserializer`[1]

The dependency of `flow` is restricted to this part of code.

[1] `fdbserver/ptxn/Serializer.h`

Items

- MutationRefs are serialized as Messages[1]:
  - MutationRef
  - SpanContextMessage[2]
  - LogProtocolMessage[3]
  - EmptyVersionMessage[1]
- Each Message has a corresponding subsequence number
- The pair (Subsequence, Message) defines an item

[1] fdbserver/ptxn/MessageTypes.h

[2] fdbserver/SpanContextMessage.h

[3] fdbserver/LogProtocolMessage.h

Headers

- For each storage team version, `SubsequencedItemsHeader`[1] is prefixed:
  - `version`
  - `lastSubsequence`
- For each batch of versions, `MessageHeader`[1] is prefixed:
  - `storageTeamID`
  - `firstVersion`
  - `lastVersion`

[1] `fdbserver/ptxn/MessageSerializer.h`

Basic serializer

The basic serializer, `SubsequencedMessageSerializer`[1] supports the following operations:

`startVersionWriting`	Starts a new storage team version.
`write`	Appends a new message.
`completeVersionWriting`	Ends the current version.
`completeMessageWriting`	Ends the current batch.
`getSerialized`	Gets the serialized data.

[1] `fdbserver/ptxn/MessageSerializer.h`

Basic serializer

```cpp
SubsequencedMessageSerializer serializer(storageTeamID);
Subsequence subsequence = 1;

for (const auto& version: versions) {
    for (const auto& message: getMessageFromVersion(version)) {
        serializer.write(subsequence++, message);
    }
    serializer.completeVersionWriting();
}
serializer.completeMessageWriting();
auto serialized = serializer.getSerialized();
```

Serializer for Proxy

- In proxy, only *one* version in one serialization step.
- Messages are distributed over multiple storage teams.
- `ProxySubsequencedMessageSerializer`[1] is implemented.

`write`	Writes a new message to a given storage team.
`broadcastSpanContext`	Broadcasts a `SpanContext` to all teams.
`getAllSerialized`	Gets the serialized data.

[1] `fdbserver/ptxn/MessageSerializer.h`

Serializer for Proxy

```
ProxySubsequencedMessageSerializer serializer(version);

serializer.write(mutationInTeam1, storageTeamID1);
serializer.write(mutationInTeam2, storageTeamID2);

auto serialized = serializer.getAllSerialized();
```

Serializer for TLog

- In TLog, multiple versions of messages in *one* storage team will be batched.
- The data is previously serialized by proxy.
- `TLogSubsequencedMessageSerializer`[1] is implemented.

`writeSerializedVersionSection`	Writes a serialized version of messages.
`getSerialized`	Gets the serialized data

[1] `fdbserver/ptxn/MessageSerializer.h`

Serializer for TLog

```
TLogSubsequencedMessageSerializer serializer(storageTeamID);

for(const auto& version : versions) {
    serializer.writeSerializedVersionSection(getDataForVersion(version));
}

auto serialized = serializer.getSerialized();
```

Deserializer Interface

Deserializer is implemented as `SubsequencedMessageDeserializer`[1].

- Deserialized data can be accessed via iterators.
- Dereferencing the iterator will yield a `VersionSubsequenceMessage`[2] object with fields:
  - `version`
  - `subsequence`
  - `message`
- The `VersionSubsequenceMessage` object is sequencable.
- The deserializer can be reset to accept new serialized data.

[1] `fdbserver/ptxn/MessageSerializer.h`

[2] `fdbserver/ptxn/MessageTypes.h`

Deserializer Interface

```
ptxn::SubsequencedMessageDeserializer deserializer(serializedData);
for (const auto& vsm : deserializer) {
    processMutationRef(std::get<MutationRef>(vsm.message));
}

deserializer.reset(anotherSerializedData);
for (const auto& vsm : deserializer) {
    processMutationRef(std::get<MutationRef>(vsm.message));
}
```

Tests

For all components, tests are included:[1]

- Serializer
- Deserializer

The tests can also be used as examples.

[1] `fdbserver/ptxn/tests/TestSerialization.cpp`