Tao Lin (04/21/2022)

GetMappedRange is an experimental feature introduced in FDB 7.1. It is intended to improve the client throughput and reduce latency for a commonly used traffic pattern. An experiment with Record Layer shows that this optimization can get 4x client throughput on a certain workload.

This document describes what this feature is for, how to use it, and the known issues (or future work).

Background

Layers (like Record Layer) are developed on top of FDB to build relational databases. And index lookup is the bread and butter of relational databases (especially OLTP databases).

A common way to model secondary indexes with a key-value database like FDB is to have an additional key-value pair for each record as an index entry (See more details in FDB Data Modeling or FDB Record Layer paper). In such a case, **querying records by scanning an index** in relational databases can be translated to **a GetRange request on the index entries followed up by multiple GetValue requests for the record entries** in FDB.

This means a relational database query is translated to multiple FDB requests. If we can teach FDB how to "follow up" a GetRange request with GetValue requests, this can happen in **one request** without additional back and forth. Considering the overhead of each request, this saves time and resources on serialization, deserialization, and network.

A new one-request API is needed to do this work. We name it GetMappedRange. A GetMappedRange request needs to:

1. Do a range query (same as a GetRange request) and get the result. We call it the **primary query**.
2. For each key-value pair in the primary query result, translate it to a GetValue (`get`) or GetRange (`getRange`) query and get the result. We call them **secondary queries**.
3. Put all results in a nested structure and return them.

Note: Do not confuse the name "primary" and "secondary" with the concept of primary index and secondary index in relational databases. There is no secondary index abstraction in FDB. And, interestingly, considering the original use case of GetMappedRange described above, we use the primary GetRange to scan secondary index entries and then use secondary GetValue to fetch records from corresponding primary keys.

APIs

The APIs are provided at different layers of the FDB client:

- `Transaction` (`NativeAPI.actor.h`): `getMappedRange`.
- `ReadYourWritesTransaction` (`ReadYourWrites.h`): `getMappedRange`. It wraps around `Transaction` and provides read-your-writes and serializable semantics.
- C Binding (`fdb_c.h`): `fdb_transaction_get_mapped_range`. It wraps around `ReadYourWritesTransaction`.
- Java Binding `ReadTransaction` (`ReadTransaction.java`): `getMappedRange`. It wraps around the C Binding.

Input

The arguments are the same as the ordinary `getRange`/`fdb_transaction_get_range`, except that there is a new argument called Mapper.

The **Mapper** is a byte array that defines how to map **a key-value pair** (one of the key-value pairs you get from the first range query) to **a GetRange (or GetValue) request**.

As an example, `Tuple.pack("prod", "sales", "{K[5]}", "{V[6]}", "{...}")` is a `mapper` that will request getting a range with the prefix `("prod", "sales", the 5th element of the key, the 6th element of the value)`.

The detailed specs are described below. You can skip to the "Output" section if not interested.

To define Mapper, we first define the concept of **Element Selector**. It is a string that maps **a key-value pair** to **a tuple element**. The tuple is a data model defined but optional in FDB, but it is necessary to use GetMappedRange today (There is an idea around how to make the mapper more general and eventually become something like a lambda function or stored procedure. But we are not there yet).

The Element Selector is written as a string that is enclosed by `{}` with format: `{keyOrValue(startByteIndex:stopByteIndex)[elementIndex]}`

- `keyOrValue` is either `K` (means using the key) or `V` (means using the value).
- This is not supported yet. `(startByteIndex:stopByteIndex)` gives access to a specified range of the key (or value) as a byte array.
  - `startByteIndex` is the starting index (0-indexed, invlusive) of the slice (optional, defaults to 0).
  - `stopByteIndex` is the last index (0-indexed, noninclusive) of the slic (optional, defaults to len(byte array)).
  - If all bytes in the key (or value) are considered in the tuple, you can use `(:)`, or simply omit this part.
- `[elementIndex]` specifies the index (0-indexed) of the element in the tuple (parsed from the byte slice).

For example, `{K(8:-2)[3]}` is an Element Selector which means: Remove the first 8 bytes and the last 2 bytes from the key, then parse it as a tuple, and get the 4th element of it.

Now let's see how to use Element Selectors to define a Mapper.

The Mapper is a tuple that is serialized as a byte array. The basic idea is that this tuple will be used as the key to issue a secondary GetValue/GetRange request, after replacing the Element Selector elements with the actual element given the key-value pair.

- If an element of the tuple is a valid Element Selector, replace it with the actual element given the key-value pair.
- When (literally) `"{...}"` is used as the last element of the tuple, serialize the tuple (except this element) and use it as a prefix to issue the GetRange request. Otherwise, serialize the tuple and use it as a key to issue GetValue request.
- literal `"{"` or `"}"` are escaped as `"{{"` and `"}}"`.

Output

We use the Java Binding API as an example. Considering it derives from the other layers, everything described here is available in the APIs at the other layers.

The output is an `AsyncIterable<MappedKeyValue>`, it can be considered as a list of `MappedKeyValue`.

Each `MappedKeyValue` is corresponding to one key-value in the primary range query and its secondary range query (Java binding today only supports GetRange but not GetValue as the secondary query). It includes:

- A parent `KeyValue` object, which is the key-value pair in the primary range query.
- A byte array `rangeBegin` and a byte array `rangeEnd`, which describes the corresponding secondary range query.
- A `List` of `KeyValue`s, which is the result of the secondary range query.

Options

Serializable

Only serializable isolation is supported now.

Technically, supporting snapshot isolation based on it is trivial. But work needs to be done to add it and, more difficultly, test it.

Read Your Writes

The transaction should have read-your-write (RYW) enabled. But it will only succeed when the transaction does not read anything that is written. Otherwise, it throws `get_mapped_range_reads_your_writes` error. This is a way to make the RYW semantic correct (for succeeded transactions) without actually implementing it.

In the future, it's possible to fully support RYW, but it needs a lot of work because all write information is kept in the client. Multiple back-and-forths between the client and the servers will be needed. If it is not done performantly, it may diminish the performance gain of GetMappedRange.

Technically, what we support today is the same as RYW being disabled. We should provide the option to let RYW be disabled. But work needs to be done to add it and test it.

Fallback when missing local shard

If the data needed by the secondary queries are not in the same storage server (SS) as the primary query, the SS that handles the primary query cannot do the work by itself. We can is a "miss". It should fall back at a certain layer to issue the secondary queries to other SSes that hold the data.

We have two FDB knobs (`QUICK_GET_VALUE_FALLBACK` for secondary GetValue, `QUICK_GET_KEY_VALUES_FALLBACK` for secondary GetRange) to control the fallback behaviors.

When they are turned on (default), SSes will issue requests to other SSes for the missed secondary queries. If a GetMappedRange involves multiple secondary queries that miss, the queries are sent out one by one (instead of sending them in parallel).

When they are turned off, GetMappedRange with any miss secondary query will return `quick_get_value_miss` or `quick_get_key_values_miss` error, so that the user can catch it and handle fallback manually.

The fallback mechanism can be improved in the future:

Send requests from FDB client instead of SS

We want to do a fallback in the FDB clients to protect overwhelming SSes, Because FDB clients are stateless and more scalable than SSes.

In SS's response, it returns a special token for the keys or ranges that could not be fetched locally. The client issues the queries and fills the results in.

Parallelized the missed secondary queries

The user's request should also provide a max parallelism value, similar to the pipeline size concept in Record Layer. This can be considered no matter in

the context of handling misses on clients or in SSes.

GetRange vs GetValue as secondary query

In the Background section, we mentioned "GetValue for the record entries" as the secondary query, because a record entry is considered a key-value pair. But considering FDB imposes a size limit on the value, records are sometimes too large to fit in one key- pair. So a record can be split into multiple contiguous key-value pairs (See SplitHelper of FDB Record Layer). In this case, a record lookup is a GetRange instead of GetValue.

In the Mapper definition level, we have provided support for both by specifying in the Mapper. See the "APIs/Input" section.

Both types of secondary queries are supported in native `Transaction` or `ReadYourWritesTransaction` APIs.

But in C binding and Java binding, only GetRange is supported as the secondary query for now. In C binding, there is code supporting it but not tested. The main difficulty is that the response in native C++ code is `std::variant<GetValueReqAndResultRef, GetRangeReqAndResultRef>`, whose memory structure is hard to be mapped to a C struct in the C binding, compared to a vanilla C `union`.

Handle limits and continuations

At the primary range query level, the continuation mechanism is the same as regular GetRange queries.

At the secondary range query level, the continuation becomes tricky:

In the native `Transaction` or `ReadYourWritesTransaction`, the result of each secondary query contains a boolean `more`. It is `true` if the secondary query is not complete.

In the C binding, there is a similar `more` field but it is not verified whether it works correctly. In the Java binding, there is no such field.

We are considering changing it to make the sematic simpler: GetMappedRange only returns completed secondary queries. When a secondary query has a continuation, instead of setting `more` to `true` and working on the next secondary query, we stop the entire GetMappedRange and only return previous secondary queries. (Return early if a secondary query has more)

The byte limits and key-value limits of GetMapRange queries are supported but not well defined or tested. More work needs to be done to use it seriously.

Knobs

- `QUICK_GET_VALUE_FALLBACK`

- QUICK_GET_KEY_VALUES_FALLBACK
- QUICK_GET_KEY_VALUES_LIMIT
- QUICK_GET_KEY_VALUES_LIMIT_BYTES

Errors

- mapper_bad_index
- mapper_no_such_key
- mapper_bad_range_decriptor
- quick_get_key_values_has_more
- quick_get_value_miss
- quick_get_key_values_miss
- get_mapped_key_values_has_more
- get_mapped_range_reads_your_writes
- key_not_tuple
- value_not_tuple
- mapper_not_tuple

Code Information

Version History

This feature was initially named `GetRangeAndFaltMap` and landed in 7.0. But because of a lot of incompatible changes, we no longer support it in 7.0.

We re-introduced it to 7.1 as `GetMappedRange`.

7.1 is the earliest version that supports this feature.

Examples and Tests

To get a better understanding of how to use GetMappedRange, you can checkout out the related tests:

- Tests for native APIs: foundationdb/GetMappedRange.actor.cpp at main · apple/foundationdb · GitHub
- Tests for C binding: foundationdb/unit_tests.cpp at main · apple/foundationdb · GitHub
- Tests for Java binding: foundationdb/MappedRangeQueryIntegrationTest.java at main · apple/foundationdb · GitHub

They are mostly the same. But tests for native APIs also cover the continuation when hitting the limit.

Pull Requests

All merged PRs have been merged in master and release 7.1 (except one).

Most of the code changes are at the highlighted PRs. Check them out to see what is changed to support this feature.

Merged

1. **Introduce GetRangeAndFlatMap to push computations down to FDB**
2. Change SSI endpoints order to be consistent with 7.0
3. GetKeyValuesAndFlatMap should return error if not retriable
4. Restricted getRangeAndFlatMap to snapshot
5. Tests for "Restricted getRangeAndFlatMap to snapshot #5978"
6. Return error when getRangeAndFlatMap has more & Improve simulation tests
7. **GetMappedRange support serializable & check RYW & continuation**
8. Fixes for when getMappedRange cannot parse as tuple
9. Fixes #6793: Upgrade from 7.0.0 to 7.1.0 fails because of missing function fdb_transaction_get_mapped_range
10. Fix GetMappedRange test when rangeResult has more (This one hasn't been cherry-picked to release 7.1 yet)

Non essentail ones:

- (#5945 to 7.0) Introduce GetRangeAndFlatMap to push computations down to FDB
- Revert "Introduce GetRangeAndFlatMap to push computations down to FDB"
- Introduce GetRangeAndFlatMap to push computations down to FDB

Pending

- Option to omit non-boundary KVs
- Change Hop Info to more structured
- Return early if a secondary query has more

If you have any questions, please feel free to reply to this post. It's the best way to reach out to me and future developers of this feature.