Full Chunk-based Computing

TiKV

Presented by Chi Zhang (@skyzh)

Contents

- Motivation
- Developer Interface Changes
- Internal Changes
- Benchmark
- Future works

TiKV

What is Chunk-based Computing

- Using TiDB Chunk format during computation in coprocessor framework
- Previously: Vec<Option<T>>

TiKV

Motivation

- Many SQL functions handle “NULL” in the same way
- Memory allocation overhead of “Bytes” and “Json”
- Overhead in decoding and encoding data of TiDB
- Solution: use TiDB Chunk format in TiKV coprocessor framework

TiKV

Changes in Developer Interface

For primitive types (Int, Decimal, etc.)

- add “nullable” to “rpn_fn” macro attributes
- “&Option<T>” to “Option<&T>”

```
// From
#[rpn_fn]
pub fn logical_and(lhs: &Option<Int>, rhs: &Option<Int>) -> Result<Option<Int>>;

// To
#[rpn_fn(nullable)]
pub fn logical_and(lhs: Option<&Int>, rhs: Option<&Int>) -> Result<Option<Int>>;
```

TiKV

Changes in Developer Interface

For “Bytes” and “Json”

- add “nullable” to “rpn_fn” macro attributes
- “&Option<Bytes>” to “Option<BytesRef>”
- “&Option<Json>” to “Option<JsonRef>”

```
// From
#[rpn_fn]
pub fn length(arg: &Option<Bytes>) -> Result<Option<Int>> {
    Ok(arg.map(|bytes| bytes.len() as Int))
}

// To
#[rpn_fn(nullable)]
pub fn length(arg: Option<BytesRef>) -> Result<Option<Int>> {
    Ok(arg.map(|bytes| bytes.len() as Int))
}
```

TiKV

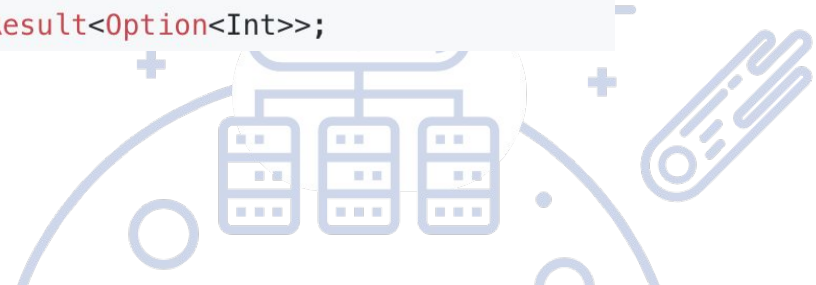
Changes in Developer Interface

For functions that return “NULL” if any of the argument is “NULL”

- no attributes for “rpn_fn” macro
- remove “Option” in parameters

```
// A nullable example
#[rpn_fn(nullable)]
pub fn logical_and(lhs: Option<&Int>, rhs: Option<&Int>) -> Result<Option<Int>>;

// A non-nullable example
#[rpn_fn]
pub fn logical_and(lhs: &Int, rhs: &Int) -> Result<Option<Int>>;
```



TiKV

Changes in Developer Interface

For functions that may return large amounts of data

- Use writer-guard pattern to avoid allocation
- partial writer: begin ➡️ partial_write ➡️ finish

```rust
// using partial writer
#[rpn_fn(writer)]
pub fn repeat(input: BytesRef, cnt: &Int, writer: BytesWriter) -> Result<BytesGuard> {
    let mut writer = writer.begin();
    for _i in 0..*cnt {
        writer.partial_write(input);
    }
    Ok(writer.finish())
}
```


TiKV

Changes in Developer Interface

For functions that may return large amounts of data

- Use writer-guard pattern to avoid allocation
- direct write: write(Some(Bytes)), write_ref(Some(BytesRef)), write(None)

```
// using direct write
#[rpn_fn(writer)]
fn json_type(arg: JsonRef, writer: BytesWriter) -> Result<BytesGuard> {
    Ok(writer.write(Some(Bytes::from(arg.json_type()))))
}
```

TiKV

Changes in Developer Interface

- using writer + Bytes / Json is recommended
- using “rpn_fn” without “nullable” is recommended

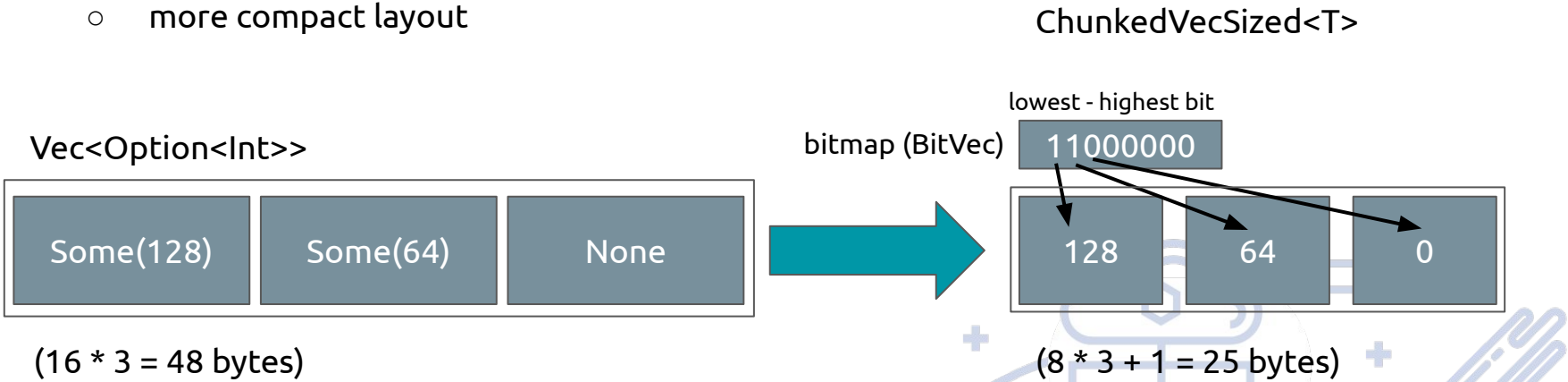
```rust
#[rpn_fn(writer)]
#[inline]
pub fn repeat(input: BytesRef, cnt: &Int, writer: BytesWriter) -> Result<BytesGuard> {
    let cnt = if *cnt > std::i32::MAX.into() {
        std::i32::MAX.into()
    } else {
        *cnt
    };
    let mut writer = writer.begin();
    for _i in 0..cnt {
        writer.partial_write(input);
    }
    Ok(writer.finish())
}
```

```rust
#[rpn_fn(capture = [ctx])]
#[inline]
pub fn arithmetic_with_ctx<A: ArithmeticOpWithCtx>(
    ctx: &mut EvalContext,
    lhs: &A::T,
    rhs: &A::T,
) -> Result<Option<A::T>> {
    A::calc(ctx, lhs, rhs)
}
```

TiKV

Internal Changes

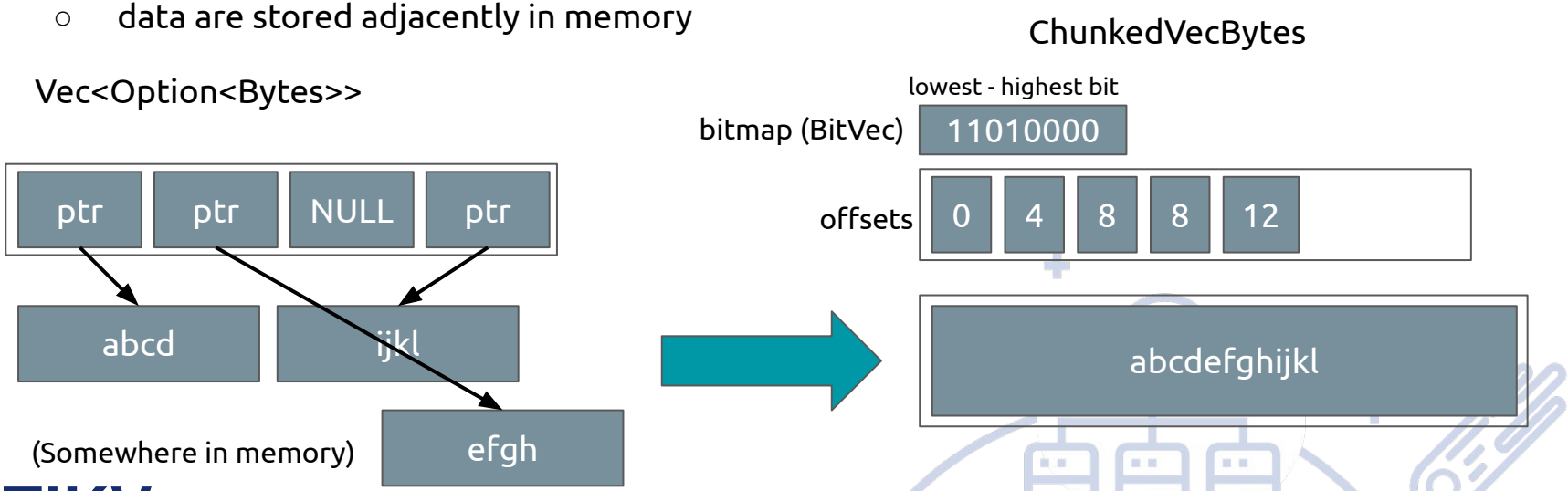
- “Vec<Option<T>>” to “ChunkedVecSized<T>”
  - append-only
  - can only get reference to an element (&T)
  - use “bitmap” to represent if cell is null or not
  - more compact layout



TiKV

Internal Changes

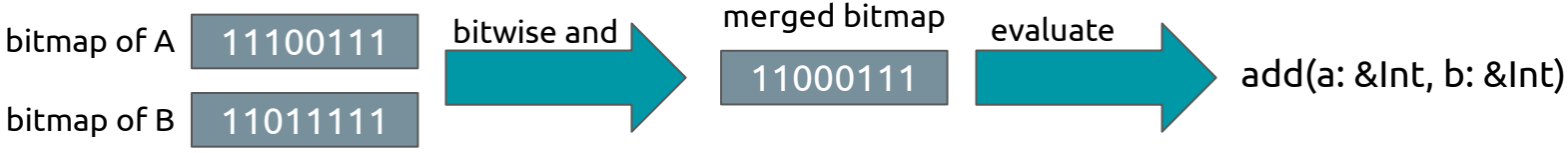
- “Vec<Option<Bytes>>” to “ChunkedVecBytes”, “Vec<Option<Json>>” to “ChunkedVecJson”
  - append-only
  - can only get “BytesRef” or “JsonRef”
  - data are stored adjacently in memory



TiKV

Internal Changes

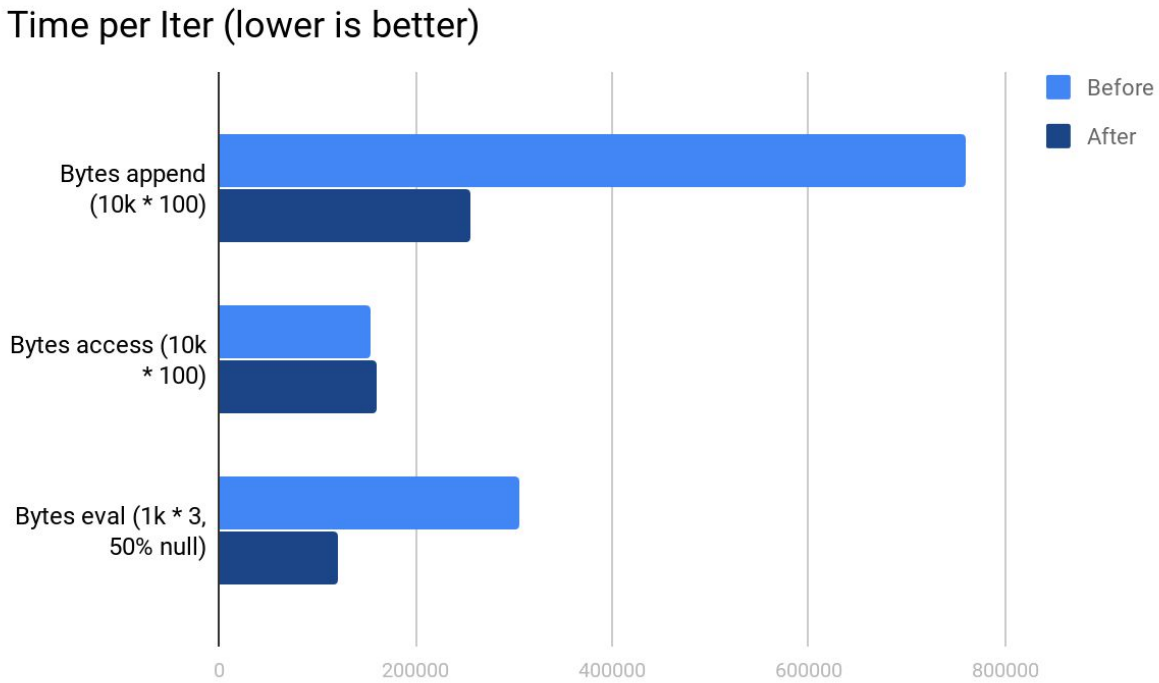
- “NULL”s are merged before evaluating for functions without “nullable”
  - Previously: “add” receives Option<Int> and handles NULL inside RPN function



TiKV

Micro Benchmarks

- Significant improvement in processing Bytes data



TiKV

Future works

- Speedup encode and decode speed
  - efficiently tell if “logical_rows” are identical
  - construct “ChunkedVec” directly from TiDB Chunk format
  - construct TiDB Chunk “Column” directly from “ChunkedVec”
- Support “nullable” and NULL merging in variable argument RPN function #[rpn_fn(vargs)]
- Refactor existing vectorized functions to use new interface

TiKV

Thank You !

TiKV