RUST CHINA CONF 2020

首届中国 Rust 开发者大会

2020.12.26–27 深圳

Rust China Conf 2020

Shenzhen, China

2020conf.rustcc.cn

Use Rust to Build Novel High Performance Open-Source Data Warehouse

金明剑

Rust China Conf 2020 – Shenzhen, China

Introduction

- J in Mingjian, Ph.D. (jin.phd at gmail.com)
- Bigdata engineering, high performance infrastructure, language and system
  - Cassandra, Spark, Impala, Kudu, ClickHouse...
  - Eclipse pioneer in China, Scala pioneer in China, Landz Java 8 foundation(faster than Netty), First Swift Linux SDE developer…
  - Contributing official PRs to Flink, LLVM, MLIR, Scala, Eclipse…

For this Talk

- **Works and practices** involving in my recent open-source project **TensorBase**(tensorbase.io)
- For **such a grand DB topic**, 30 minutes is so short
  - Skip tech details: consult one recent blog material "***Architect a High-performance SQL Query Engine in Rust***" (for RustFest Global 2020) in site(tensorbase.io)
- KISS
  - System: world outlook of Tensorbase
  - Engineering Rust: Rust's help on this

Outline

- Introduction
- System
- Evaluation
- Engineering Rust
- Near Future

System

- Why grand
  - How to implement a **top performance** and **industry strength** JIT engine with Rust? (Done in TensorBase)
  - How to hit **10 million HTTP request/second** in a single socket in TechEmpower Framework Benchmarks Plaintext? (Done in TensorBase)
  - How to **sum 1.5 billion rows in 80 millisecond**? (Done in TensorBase)
  - How to **saturate modern NVME** storage bandwidth in data ingestion?
  - How to efficiently coordinate and manage concurrency and parallelism? (Partially done in TensorBase)
  - logging, security, new hardware…


How high is the sky

System

Near Goal

Data Warehouse for Modern World

Data Warehouse

- Massive data in seconds
- Concerns on large collective infos rather than small
- Strong consistency is not that important (in that massive data ingesting in real time) (YMMV)
- Reliable Storage (most OLAP does not guarantee this)

Modern World

- Massive data in seconds
- Core counts scaling while core frequency died
- Memory wall is growing fat
- NVME SSD is high speed
- NVME SSD is cheap enough

Modern World

Not Well Known

- Tens of Billion calculations done in one second for a single modern core, trillions/s for single package
- Scalability does not mean high performance, and possibly be low performance and thus cost expensive

Modern World

Not Well Known

- Petabytes(PB) level Data can be hold in a single modern node
- Current OLAPs are far from taking use of the full capabilities of modern CPU

Modern World

Not Well Known

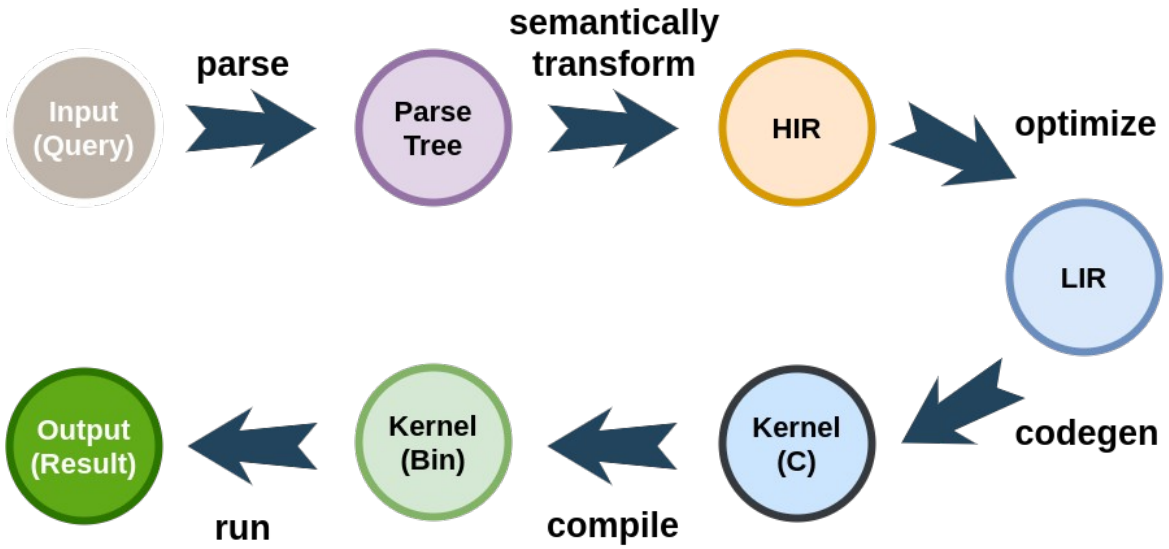
- Speed of IO is on par with or **faster than** that of memory (in throughput)
- In memory OLAP is economically cost prohibitive for massive data analysis

System

- Base Core
  - First principle
  - Shared Something
  - Architectural Performance
  - Almost from scratch

System

- Base Arch



System

- Components
  - Ansi-SQL Parser (in Rust)
  - Layered IR (in Rust)
  - JIT Compiler (SQL to execution)
  - Storage (in Rust)
  - Server (in Rust)
    - Also have the client and also in Rust

System

- Unique Components （First appear in the “verifiable” world）
  - Unified RA(Relational Algebra)
    - All-in-Four **Math Simplicity**: ->(map), +(union/agg), *(join), <>(sort/top)
  - JIT based whole-system optimization
    - Industry level strength C based JIT compiler
    - Iterative optimization
  - Decentralized, self-scheduling
    - Self adaptive execution and storage
    - Computing and data granularity is elastic rather than fixed small (morsels)
    - v.s. Morsel-driven system, e.g. HyPer…

System

- Unique Components
  - Rust almost
    - All things rather than “Read” (e.g. Write/Storage) are done in Rust
    - “Read” sides are in C(/C++) + Rust mixing
  - ClickHouse native protocol compatible
    - A new in-Rust server for fastest-in-world data warehouse ingestion
    - current several Rust ClickHouse client drivers have flaws which has been fixed in TensorBase

Evaluation

Talk is cheap.

Show me the code,

and result…

Evaluation

All Clients which support ClickHouse native TCP protocol can connect

```
➥  clickhouse-client
ClickHouse client version 20.11.1.4897 (official build).
Connecting to localhost:9000 as user default.
Connected to TensorBase server version 2020.12.0 revision 54

ClickHouse client version is older than ClickHouse server. I
or new features.

TensorBase :) insert into trips_lite_n10 values (7, 10), (9,

INSERT INTO trips_lite_n10 VALUES

Query id: 8e3c1561-2ad5-4d23-addc-80e0408b2930

Ok.

2 rows in set. Elapsed: 0.009 sec.

TensorBase :)
```

Evaluation

Phase	Language (TensorBase M0)	Time	Ref Time (from public)	Language (Ref system)
Parsing (TPC-DS)	Rust	130 us*	~50 us*	C++
Parsing/IR/ codegen (one column sum)	Rust	130 us*	~1 ms*	C++
C Kernel JIT Compilation	C/C++	**13 ms** (boost) - 20 ms (no boost) (Q#1 like, -O2)*	59 ms* (TPC-H Q#1, opt.)	C++

Evaluation

Phase	SQL Sample	TensorBase frontier	ClickHouse	Throughput Comparsion
End-to-end Simple Sum Query (1.47B row NYC taxi dataset)	select sum(trip_id-100)*123 from nyc_taxi	**~60 ms** (compilation cached)- ~90 ms(compilation uncached)*	**642 ms*** In ClickHouse 20.05 (compilation cached)	TB: ~95GB/s CH: ~20GB/s Node: 100GB/s
End-to-end Query Time (one 1B row dataset from online data analysis vendor)**	select activity_name,activity_id,count(1) from event by group by activity_name,activity_id	**~120 ms** (compilation cached) (in ClickHouse compatible data storage mode)	**~1.1s (690ms)** In ClickHouse 20.11 (compilation cached)	TB: ~50GB/s CH: ~10GB/s Node: 100GB/s

Sky-high: can not be faster any more for GB+ level dataset

Engineering Rust

- **High Performance** is to orthogonal to the language
  - But a right language **makes high performance easier - Rust**
- **Engineering** is to orthogonal to the language
  - But a right language **makes you and your users happier - Rust**
- **Programming style** is to orthogonal to the language
  - But is “**async**” necessary for your next favorite in Rust?

Engineering Rust

- Engineering
  - TensorBase benefits from
    - Crates ecosystem, cargo, tooling/IDE, modules, macros, language(memory safe), language(zero-overhead C interop), language(ADT)
    - **one-person project** in several months

Engineering Rust

- Cargo
  - Highly extensible
  - Useful commands (except that most commonly used):
    - ***add***: for add latest version of deps
    - ***expand***: for proc macro debugging
    - ***tree***: for transitive deps checking
- Tooling: VS Code + RA

Engineering Rust

- Proc macro
  - Learning curve is high
  - Too many out-of-date materials but less practices
  - Tooling is in primary stage
  - Hard to debug
  - Enable feature nightly ***proc_macro_diagnostic***, ***proc_macro_span*** for better debugging output
  - Proc_macro2 should improve the testability

Engineering Rust

- C interoperability
  - Zero overhead
    - MaybeUninit, transmute, std::ptr
  - Resource management
    - Objects in C is managed manually but not in safe Rust
  - Error handling
  - Too many “unsafe”s and “as”
    - Nice watch PR: [RFC - Safer Transmute](/rfcs/pull/2981)

Engineering Rust

- Concurrency
  - Nice for share-nothing thread safety
  - Awkward when memory sharing needed
    - Memory sharing - cornerstone of modern multicores
    - High frequency copy is a performance disaster
    - “Channel” is behind on sharing
  - Lack memory model like in Java/C++

Engineering Rust

- Lifetime
  - Engineering excellence but make codes complex
    - Always recommends: to dance with, rather than to evade
    - Compiler enforces more than that needed when not smart
  - Alternatives
    - Arena allocator: TensorBase IR
    - Unsafe into C: TensorBase kernel algorithms in C

Engineering Rust

- For tiny example
  - Why not allow implicitly cast i32 to i64?
    - Assumed you can accept explicitly cast u64 as u32
    - write_to_wire(some_i64_var)
    - v.s. write_to_wire(some_i32_var as i64)
    - **Silent failure** when you have generic write_to_wire which deals with i8/16/32/64 in one round
    - Because 8 byte is 2x larger with 4 byte in wire/memory, no matter you can 123456789 is as i32 or i64

Engineering Rust

- For example
  - Function evolvement
  - **Floor#1**: Top function
    - write_varint(buf: &mut BytesMuts)
  - **Floor#2**: Trait to abstract common behaviors
    - Impl ByteMuts for CHMsgAware { … write_varint … }
    - let bs: &mut ByteMuts = …
    - bs.write_varint(123);
    - let bs: &mut &[u8] if you impl &[u8] from CHMsgAware
  - **Floor#3?**: #[derive(MyBuf)]

Engineering Rust

- For bigger example

```
let mut bs =  ...
let skip_len =  ...

 ...
let ptr: &[u8] = bs.get_slice_at( ... )
bs.advance_mut_unsafe(skip_len);
bs.ensure_enough_bytes_to_write(max_len);
 ... (many other mutations)
```

code snippet with many unsafe wrappings

Wath's the catch?

Engineering Rust

- For bigger example
  - Memory safety is a must-have if you can have, especially for high performance engineering

```
let mut bs = ...
let skip_len = ...

...
let ptr: &[u8] = bs.get_slice_at( ... )
bs.advance_mut_unsafe(skip_len);
bs.ensure_enough_bytes_to_write(max_len);
... (many other mutations)
```

code snippet with many unsafe wrappings

Great catch!

Near Future

- TensorBase Frontier Edition
  - Will be released under a new license to explore kernel-like development model, but still free for any commercial uses
  - Enterprises are welcome to join the frontier ed development
- TensorBase Base Edition
  - Continue to as an APLv2 licensed baseline
  - Personals are mentored under the base ed by me
- Interesting modules could be published as standalone projects to Rust ecosystem

Thanks

RUST CHINA CONF 2020

首届中国 Rust 开发者大会

2020.12.26–27 深圳