StratoQL and Local Type Inference

Runhang Li (rli@twitter.com)
03/12/2018

Outlines

1

Design of Stratostore

10%

2

StratoQL and Local Type Inference

75%

3

Demo

15%

Background Of Twitter's Core Services

- Core Services: Tweet services, User services, Social graph…
- First monolithic Ruby-on-Rails application, then refactored to microservices
- No unified data model. Existing services become inflexible. Hard to launch new service.

Death star


Zipkin
Investigate system behavior
Find a trace
Aggregates

Stratostore

A federated database system through which data at Twitter may be uniformly accessed, modified, and composed

(Think Stratostore as SQL database, where tables are backed by Thrift services)

Stratostore

- A catalog of *tables* described by *schemas*
  Or in Strato world, a catalog of *columns* described by *types*

- A language called *StratoQL* to construct columns

Stratostore

- A catalog of *tables* described by *schemas*
  Or in Strato world, a catalog of *columns* described by *types*

- A language called *StratoQL* to construct columns

Data in Strato

Spaces

All data sharing a key (e.g. Tweet ID)

Columns

Dataset defined by types of its key, value, and operations it supports (e.g. CRUD)

A column is associated with the space of its key

Data in Strato

Spaces

All data sharing a key (e.g. Tweet ID)

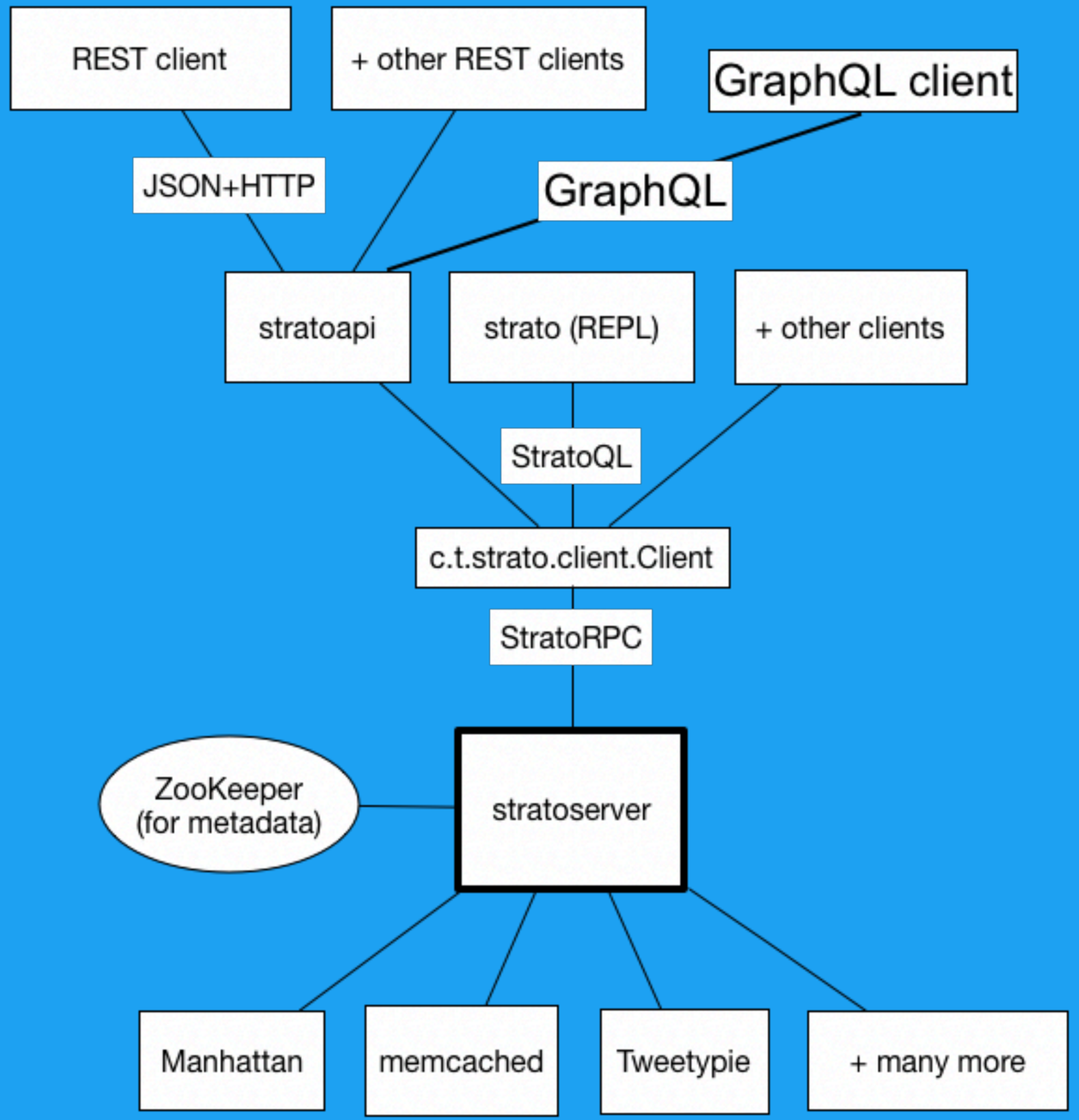
Columns

Dataset defined by types of its key, value, and operations it supports (e.g. CRUD)

A column is associated with the space of its key

Stratostore

- Column is constructed using StratoQL
- StratoQL is compiled into composed RPC calls which will fetch data from other Strato columns or existing external datasets


REST client
+ other REST clients
GraphQL client
JSON+HTTP
GraphQL
stratoapi
strato (REPL)
+ other clients
StratoQL
c.t.strato.client.Client
StratoRPC
ZooKeeper
(for metadata)
stratoserver
Manhattan
memcached
Tweetypie
+ many more

Entering StratoQL

Simple Scala-like statically typed language with type inference support

Design Choices

- Scala-like syntax since Twitter uses Scala
- Static type system
- Type inference

Benefits Of Type Inference

- Type inference allows users to omit type annotations. Config written in StratoQL looks concise and expressive

Benefits Of Static Type System: Service Robustness

- Type error can be caught before service deployment to avoid any disruption after service starts.
- Safe interaction with other columns and services
- Exhaustiveness checking in pattern matching can make sure all possible cases have been considered.

Benefits Of Static Type System: Schema Derivation

- GraphQL is a query language developed by Facebook and it has its own static type system to ensure runtime safety
- Deriving a sound GraphQL schema from a Strato column means a mapping from StratoQL type system to GraphQL type system. It is impossible to derive a typed schema from a dynamically typed language.

Syntax

Basic types and primitives	`Int, Double, Unit, String, 1.0, 2.hour, 20L, 1S, ()`
Variant, record, function types	`<Red|Green|Blue>, {a: Int, b:Double}, Int => Double`
declaration	`val x = 2.0`
tuples, set, seq, map	`(1, “A”), Set(1,2,3), Seq(1, 2, 3), Map(1->2, 3->4)`
Record	`{ left = 1.0, right = 2.0, upper = Some(20)}`
variants	`Finished(2.0), Error(404, “BadAccess”)`
patter matching	`x match {case Foo => true | Bar => false}`
ascription	`val x : Int = …`
column value	`#Tweet(20L)`

Polymorphism

- No support of imperative polymorphism

```
(val x : [T] T => (T, T) = { x => (x, x)}
```

- Support predefined polymorphic data structure and operations on them

```
Set(1, 2, 3).map({ case x => x + 1})
```

Subtyping

Primitive	`Short<:Int, Int<:Double`
Variants (width and depth)	`<Foo(Short)> <: <Foo(Int)>, <A> <: <A|B>`
Records (width and depth)	`{a:Short} <: <a:Int>, {a:Short} <: {a:Short, b:Int}`
Tuples, Set, Seq	`(Int, Short) <: (Int, Int), Set[Short] <: Set[Int],`
Functions	`Int=>Short <: Short=>Int`
Map	`Map[Int, Int] <: Map[Int, Double]`

Subtyping

- Width and depth subtyping for variants and records
- Support predefined polymorphic data structure and operations on them

```
Set(1, 2, 3).map({ case x => x + 1})
```

Algorithm W

- Works on language with Hindley-Milner type system (lambda calculus with parametric polymorphism)
- Cannot work on language with subtyping or imperative polymorphism
- In fact, complete type inference for imperative polymorphism is already unknown (Wells, 1994)

Local Type Inference

Partial type inference for language with subtyping and imperative polymorphism

Local Type Inference

Three categories of annotations

A. Type arguments in application of polymorphic functions
B. Annotations on bound variables in anonymous function abstractions
C. Annotations on local variable bindings

Category A:
Type arguments in application of polymorphic functions

```
val f : [T] T => (T, T) = { x => (x, x)}

f(42)
```

Type argument `Int` is omitted:

```
f[Int](42)
```

Category B: Annotations on bound variables in anonymous function abstractions

```
val f : (Int => Int) => Int = { g => g(42)}

f({x => x + 1})
```

Type annotation `Int` is omitted:

```
f({x: Int => x + 1})
```

Category C: Annotations on local variable bindings

```
val x = 1
```

Type argument `Int` is omitted:

```
val x: Int = 1
```

Category A:
Type arguments in application of polymorphic functions

```
val f : [T] T => (T, T) = { x => (x, x)}

f(42)
```

- More than one choices for type argument here!
  `f[Long](42)` or `f[Int](42)`?

Category A:
Type arguments in application of polymorphic functions

```
val f : [T] T => (T, T) = { x => (x, x)}

f(42)
```

Rule:
always choose type arguments such that result type is the most informative

Category A:
Type arguments in application of polymorphic functions

```
val f : [T] T => (T, T) = { x => (x, x)}

f(42)
```

Rule:
always choose type arguments such that result type is the most informative

In this case, we choose `Int` since `Int <: Long`

Category A:
Type arguments in application of polymorphic functions

What if we cannot decide the most informative type?

```
val f : [T] T => (T => T) = {x => (x => x)}
```

`Int => Int` is not a subtype of `Long => Long`, vice versa!

Just announce failure in such case:
we cannot synthesize a type here!

Some formal definitions

$$\Gamma \vdash e \in \mathbb{T} \Rightarrow e'$$

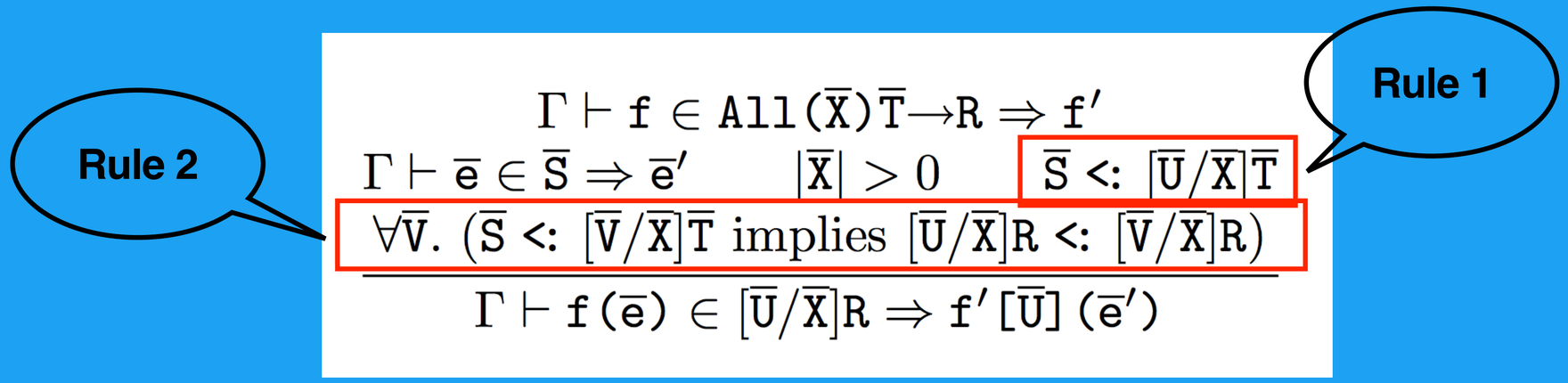
In context Γ, type annotations can be added to the external language term (no annotation) `e` to yield the internal language (with annotation) term `e'`, which has type `𝕋`.

Category A:
Type arguments in application of polymorphic functions

Technique 1: Local Type Argument Synthesis

$$\frac{\begin{array}{c}\Gamma \vdash \mathtt{f} \in \mathtt{All}(\overline{\mathtt{X}})\,\overline{\mathtt{T}} \to \mathtt{R} \Rightarrow \mathtt{f}' \\ \Gamma \vdash \overline{\mathtt{e}} \in \overline{\mathtt{S}} \Rightarrow \overline{\mathtt{e}}' \qquad |\overline{\mathtt{X}}| > 0 \qquad \overline{\mathtt{S}} <: [\overline{\mathtt{U}}/\overline{\mathtt{X}}]\overline{\mathtt{T}} \\ \forall \overline{\mathtt{V}}.\ (\overline{\mathtt{S}} <: [\overline{\mathtt{V}}/\overline{\mathtt{X}}]\overline{\mathtt{T}} \text{ implies } [\overline{\mathtt{U}}/\overline{\mathtt{X}}]\mathtt{R} <: [\overline{\mathtt{V}}/\overline{\mathtt{X}}]\mathtt{R})\end{array}}{\Gamma \vdash \mathtt{f}(\overline{\mathtt{e}}) \in [\overline{\mathtt{U}}/\overline{\mathtt{X}}]\mathtt{R} \Rightarrow \mathtt{f}'[\overline{\mathtt{U}}](\overline{\mathtt{e}}')}$$

Technique 1: Local Type Argument Synthesis



Two conditions:

1. The types of the value parameters `S` must be subtypes of the function's parameter types `[U/X]T`
2. The arguments `U` must be chosen in such a way that any other choice of arguments `V` satisfying the previous condition will yield a less informative result type, i.e., a supertype of `[U/X]R`.

Technique 1: Local Type Argument Synthesis

```
val f : [T] T => (T, T) = { x => x)}

f(42)
```

Rule:
always choose type arguments such that result type is the most informative

In this case, we choose `Int` since `Int <: Long`

Technique 2: Bidirectional typechecking

Typechecker operates in two distinct modes:

- *Synthesis mode*

  Typing information is propagated upward from subexpressions. Used when we do not know anything about the expected type of an expression

- *Checking mode*

  Typing information is propagated downward from enclosing expressions. Used when the surrounding context determines the type of the expression and we only need to check that it does have that type.

Technique 2: Bidirectional typechecking

Used on category B and C

Category B: Annotations on bound variables in anonymous function abstractions

```
val f = { g:(Int=>Int) => g(42)}
f({x => x + 1})
```

Category C: Annotations on local variable bindings

```
val x = 1
```

Technique 2: Bidirectional typechecking

A simple example

```
val f = { g:(Int=>Int) => g(42)}

f({x => x + 1})
```

Switching back and forth between two modes:

- `42` is *checked* to have type `Int`
- `f` is *synthesized* to have type `(Int => Int) => Int`
- `{x => x + 1}` is *checked* against `(Int => Int)`

Basic formal definitions

$$\Gamma \vdash e \rightarrow\in T \Rightarrow e' \quad \textbf{synthesis}$$

$$\Gamma \vdash e \leftarrow\in T \Rightarrow e' \quad \textbf{checking}$$

Formal definitions: variables

Synthesis

$$\Gamma \vdash \mathbf{x} \overrightarrow{\in} \Gamma(\mathbf{x})$$

Checking

$$\frac{\Gamma \vdash \Gamma(\mathbf{x}) <: \mathtt{T}}{\Gamma \vdash \mathbf{x} \overleftarrow{\in} \mathtt{T}}$$

Formal definitions: abstractions

Synthesis

$$\frac{\Gamma, \overline{X}, \overline{x}{:}\overline{S} \vdash e \overrightarrow{\in} T}{\Gamma \vdash \mathtt{fun}[\overline{X}](\overline{x}{:}\overline{S})\, e \overrightarrow{\in} \mathtt{All}(\overline{X})\overline{S} \to T}$$

No synthesis rule for unannotated abstractions since we cannot determine from context

Checking (unannotated)

$$\frac{\Gamma, \overline{X}, \overline{x}{:}\overline{S} \vdash e \overleftarrow{\in} T}{\Gamma \vdash \mathtt{fun}[\overline{X}](\overline{x})\, e \overleftarrow{\in} \mathtt{All}(\overline{X})\overline{S} \to T}$$

Able to determine annotation in checking mode

Checking (annotated)

$$\frac{\Gamma, \overline{X} \vdash \overline{T} <: \overline{S} \qquad \Gamma, \overline{X}, \overline{x}{:}\overline{S} \vdash e \overleftarrow{\in} R}{\Gamma \vdash \mathtt{fun}[\overline{X}](\overline{x}{:}\overline{S})\, e \overleftarrow{\in} \mathtt{All}(\overline{X})\overline{T} \to R}$$

Check if existing annotations is consistent with what we expect

Formal definitions: applications

Synthesis

$$\frac{\Gamma \vdash \mathtt{f} \overrightarrow{\in} \mathtt{All}(\overline{\mathtt{X}})\overline{\mathtt{T}}{\to}\mathtt{R} \qquad \Gamma \vdash \overline{\mathtt{e}} \overrightarrow{\in} \overline{\mathtt{S}} \qquad |\overline{\mathtt{X}}| > 0 \qquad \Gamma \vdash \overline{\mathtt{S}} <: [\overline{\mathtt{U}}/\overline{\mathtt{X}}]\overline{\mathtt{T}} \qquad \forall \overline{\mathtt{V}}.\ (\Gamma \vdash \overline{\mathtt{S}} <: [\overline{\mathtt{V}}/\overline{\mathtt{X}}]\overline{\mathtt{T}} \text{ implies } \Gamma \vdash [\overline{\mathtt{U}}/\overline{\mathtt{X}}]\mathtt{R} <: [\overline{\mathtt{V}}/\overline{\mathtt{X}}]\mathtt{R})}{\Gamma \vdash \mathtt{f}(\overline{\mathtt{e}}) \overrightarrow{\in} [\overline{\mathtt{U}}/\overline{\mathtt{X}}]\mathtt{R}}$$

Basically the same as type argument synthesis

Checking

$$\frac{\Gamma \vdash \mathtt{f} \overrightarrow{\in} \mathtt{All}(\overline{\mathtt{X}})\overline{\mathtt{T}}{\to}\mathtt{R} \qquad \Gamma \vdash \overline{\mathtt{e}} \overrightarrow{\in} \overline{\mathtt{S}} \qquad |\overline{\mathtt{X}}| > 0 \qquad \Gamma \vdash \overline{\mathtt{S}} <: [\overline{\mathtt{U}}/\overline{\mathtt{X}}]\overline{\mathtt{T}} \qquad \Gamma \vdash [\overline{\mathtt{U}}/\overline{\mathtt{X}}]\mathtt{R} <: \mathtt{V}}{\Gamma \vdash \mathtt{f}(\overline{\mathtt{e}}) \overleftarrow{\in} \mathtt{V}}$$

Check on types of arguments and application result

Formal definitions: Local variable bindings

Synthesis

$$\frac{\Gamma \vdash e \overrightarrow{\in} S \qquad \Gamma, x{:}S \vdash b \overrightarrow{\in} T}{\Gamma \vdash \texttt{let}\ x = e\ \texttt{in}\ b \overrightarrow{\in} T}$$

Checking

$$\frac{\Gamma \vdash e \overrightarrow{\in} S \qquad \Gamma, x{:}S \vdash b \overleftarrow{\in} T}{\Gamma \vdash \texttt{let}\ x = e\ \texttt{in}\ b \overleftarrow{\in} T}$$

Demos