Scala Days

Gallia

A schema-aware Scala library for generic data transformation



Anthony Cros

Gallia Project

What: Gallia in a nutshell



Trivial example:

```
"""{"first": "anthony", "last": "cros", "age": 39, "fit": true}""
   .read() // lazily + infers schema
        .nest("first", "last").under("name")
        .increment("age")
        .remove ("fit")
   .display() // trigger meta then data transformation
```

```
prints data:
    { "name": {
          "first": "anthony",
          "last": "cros" },
          "age": 40 }
```

Gallia is "schema aware"



- Before processing data (potential schema inference notwithstanding)
- For any level of nesting/multiplicity
- For any operation where such checks can be realized (won't catch IOOB errors for instance)

How do I get started?



```
mkdir /tmp/sd23 && cd /tmp/sd23
echo 'libraryDependencies +=
  "io.github.galliaproject" %% "gallia-core" % "0.4.0"' \
    > build.sbt # 2.12 or 2.13 only
sbt console
scala> import gallia._
scala> """{"report": "TPS", "submitted": false}"""
             .read().flip("submitted").display()
```

Main goals



- 1. Practicality
- 2. Readability
- 3. Scalability (optionally)

When should I use it?



Python pandas: small dataframes

/

Gallia: small and big data

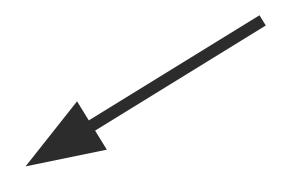
not necessarily "framed" data (think nesting)

Apache Spark: big dataframes

⇒ One-stop shop paradigm for data transformations*

Who's the team behind it?





(hopefully I'm standing somewhere down there)

FOR NOW!

Scala Days

Why not Case Classes + Collections?

- 1. May require **lots** of *case classes* (see <u>DbNSFP</u> example on github)
 - Input, output, and intermediate case classes
 - Lots of boilerplate code
 - Lots of naming: naming things judiciously is hard
- Transformation code only commits to schema elements used
 - o For instance: val f: HeadS => HeadS = _.remove("name").flip("fit")
 - Only "commits" to there being:
 - "name" field (whose type is irrelevant)
 - "fit" field (boolean, because we flip it)
 - Great for schema evolution or schema unification
- Data model may still be changing, or you may just be poking around

I/O Supported



Currently:

- {T,C}SV files: *plain, GZIP*, and *BZ2*
- JSON/jsonl: plain, GZIP, and BZ2
- Apache Avro and Parquet
- RDBMS: via JDBC
- Mongodb: read-only (for now)

In the future:

- Excel (yes, lots of research and clinical labs still rely on it)
- XML
- GraphQL, Sparql, Cypher (NEO4J), ...





Choosing what to act on: its own meta-querying language

```
"f1" )
remove
                    "parent" |> "child" )
remove
                    _.firstKev) }
remove
                     _.allBut ("f2", "f3") }
remove
                     _.findKey(_.endsWith("1")) }
remove
                     _.fullCustomKey { schema => [...] } }
remove
transform { _.string("f1") } .using([...])
transform { _.string(_.firstKey) }.using([...])
transform { _.string(_.allkeys ) }.using([...])
```

Live coding session



Let's write some code!





(When Spark isn't available)

"Poor man's" scaling leveraging *mergesort* algorithm, think:



Scaling: Poor man's version

Invoke via the use of .iteratorMode:

```
"/my/big/file/input.jsonl.gz2"
    .stream(
        _.iteratorMode.schema([...]))
        .decrement("fanciness")
        .groupBy([...])
        [...]
```

CAVEAT: only works if GNU sort is available on your system (never tested with Windows/Mac)

Scaling with Gallia in practice



See example of real world usage: GeneMania dataset processing

- **Poor man**/mergesort: <u>GeneMania.scala</u>

- Gallia+Spark equivalent: GeneManiaSparkDriver.scala

They share **the exact same code** for the transformations, and differ only in their I/O

More features



See this <u>Towards Data Science</u> article which introduce:

- Union Types: minimal support
- Metaschema:
 - As a result of supporting union types
 - Could dogfood schema transformations
- Cotransformations:
 - Via intermediate case classes representing a subset of schema
 - Great for encapsulation
 - For *M-to-N* field transformations where *M>1* and *N>1*
 - Example: gist

So, why should I use Gallia?



- 1. Most common/useful data operations
- 2. Readable DSL
- 3. Scaling is not an afterthought
- 4. Meta-awareness
- 5. Can process individual entities
- 6. Can process deeply nested entities of any multiplicity
- 7. Provides flexible target selection
- 8. Execution DAG can be optimized easily thanks to Gallia's abstractions

Conclusion



- Seeking feedback to decide on future directions
- Welcome comparisons with existing tools
 - What would the complex biomedical <u>data</u> <u>transformations</u> (e.g. *dbNSFP*) look like with *pandas*, *Spark SQL*, etc?
 - Keeping in mind Gallia's priorities!
- Welcome help in general:
 - Terminology ("deserialize1b"...)
 - Documentation ("deserialize1b"???!!!)
 - Optimization
 - Semantic information?

Scala Days 2023

Thanks!

