GIO: Generating Efficient Matrix and Frame Readers for Custom Data Formats by Example

Saeed Fathollahzadeh\textsuperscript{1,2}  
Matthias Boehm\textsuperscript{3}

\textsuperscript{1}TU Graz  \hspace{1cm} \textsuperscript{2}KNOW Center  \hspace{1cm} \textsuperscript{3}TU Berlin
Motivation

- Data formats vary in structure, syntax, semantics, and compression
  Examples: CSV, JSON, XML, Parquet, ORC, HDF5, FITS, etc.

- Existing systems have limited support for custom formats

- Custom readers require low-level programming and system knowledge

- Custom readers are not portable across systems and languages

➡️ Having state-of-the-art readers isn’t enough:
Is there an efficient and automatic way to get around?
Problem of Custom Data Formats

Custom data formats originate from systems and machines, whose data representation was not designed for data exchange and interoperability

- Such data formats include flat or nested structure
- Optional key or positional attributes
- Multiple custom delimiters or prefixes
- Potentially multi-line records
- Undocumented semantics of attribute sequences
- Co-appearances and repeating groups of attributes

Examples: semiconductor manufacturing, smart grid data management, paper production, and waste recycling
Is there an automatic way to get around?

Today:

Read { Raw Data }

- CSV Reader?
- JSON Reader?
- XML Reader?
- HL7 Reader?

Reader Exists?
Is there an automatic way to get around?

Today:

Read:

```
Raw Data
```

- CSV Reader?
- JSON Reader?
- XML Reader?
- HL7 Reader?

Reader Exists?

GIO: Generating Efficient Readers

```
Raw Data
```

Create Examples

Identify Format

Generate Reader,

Read:

```
GIO: Generating Efficient Matrix and Frame Readers for Custom Data Formats by Example
Matthias Boehm
```
SystemDS I/O Gen API

APIs

Compiler

Control Program

Buffer Pool

Mem/FS I/O  GIO  DFS I/O

https://systemds.apache.org

LibSVM, AMiner, MatrixMarket, ...
GIO’s Parameters

Given a custom text-based dataset $D$, and user-provided examples of raw and target data:

- **Sample Raw ($S$) Input:** Let $S = \{S_1, S_2, \ldots, S_l\}$ be a list of input strings (i.e., selected rows of the input dataset $D$).

- **Sample Matrix/Frame ($F$) Input:** Let $F = \{F_1, F_2, \ldots, F_n\}$ be a sample matrix or frame, corresponding to $S$ with $n$ records.

⇒ our goal is to generate a matrix or frame reader for reading the full dataset and other data of this format.
As data collections become larger and larger, data loading evolves to a major bottleneck. ... unlimited set of references

Storage and memory systems for modern data analytics are heavily layered, managing shared ...

Exploratory big data applications often run on raw unstructured or semi-structured data ...

Sample Raw (S)

Sample Frame (F)

F =
\[
\begin{array}{cccc}
#col_1 & #col_2 & #col_3 & #col_4 \\
2015101 & Ioannis Alagiannis & Renata Borovica & Anastasia Ailamaki \\
2019102 & Jia Zou & Arun Iyengar & Chris Jermaine \\
2018103 & Shoumik Palkar & Firas Abuzaid & Matei Zaharia \\
\end{array}
\]
GIO Overview

Step 1
- Identification
- Mapping
- Shape Inference

Step 2
- Code Generation
  - Modify Reader Src
  - Runnable Reader (R)

Output Matrix/Frame

Example (S)
Dataset (D)
Matrix/Frame
(R(D))
Step 1: Mapping Identification

Sample Raw (S)

Cell Value Mapping

Sample Frame (F)
Step 1: Mapping Identification

Mapping Rules:
1. Shape Inference Functions
2. Cell Value Mapping Functions

Sample Raw (S)

Sample Frame (F)

<table>
<thead>
<tr>
<th>Prefixes</th>
<th>Suffixes</th>
</tr>
</thead>
<tbody>
<tr>
<td>#col₁</td>
<td>#col₄</td>
</tr>
<tr>
<td>#col₂</td>
<td>#col₃</td>
</tr>
<tr>
<td>#col₃</td>
<td>#col₄</td>
</tr>
</tbody>
</table>

Note: ■ The value has already been selected by another column(s) and is now empty.
Step 1: Mapping Identification

Mapping Rules:
1. Shape Inference Functions
2. Cell Value Mapping Functions

Pattern Creation
\[ \forall \text{col} \in \{1, 2, 3, 4\}, P_{\text{col}} = \text{LCS}(l_1, l_2, l_3) \]

Prefixes

<table>
<thead>
<tr>
<th>#col1</th>
<th>#col2</th>
<th>#col3</th>
<th>#col4</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_1</td>
<td>#index</td>
<td>#@</td>
<td>#@</td>
</tr>
<tr>
<td>l_2</td>
<td>#index</td>
<td>#@</td>
<td>#@</td>
</tr>
<tr>
<td>l_3</td>
<td>#index</td>
<td>#@</td>
<td>#@</td>
</tr>
</tbody>
</table>

Note: ■ The value has already been selected by another column(s) and is now empty.

Suffixes

<table>
<thead>
<tr>
<th>#col1</th>
<th>#col2</th>
<th>#col3</th>
<th>#col4</th>
</tr>
</thead>
<tbody>
<tr>
<td>\n</td>
<td>;</td>
<td>;</td>
<td>\n</td>
</tr>
<tr>
<td>\n</td>
<td>;</td>
<td>;</td>
<td>\n</td>
</tr>
<tr>
<td>\n</td>
<td>;</td>
<td>;</td>
<td>\n</td>
</tr>
<tr>
<td>\n</td>
<td>;</td>
<td>;</td>
<td>\n</td>
</tr>
</tbody>
</table>

GIO: Generating Efficient Matrix and Frame Readers for Custom Data Formats by Example

Matthias Boehm 9 / 18
Step 2: Code Generation

- Composed according to the passed mapping rules.
- Template-based Code Generation: parsing primitives, conditions, path expressions, and value indexing:
  - Code Templates:
    - pre-pass for obtaining additional metadata from data
    - inferring the dimensions
    - iterate over records of the raw dataset
  - Indexing Code:
    - Row Indexing Code: determine the number of rows
    - Column Index and Value Code: code gets column info and values
  - Cell Value Code:
    - Cell Value by Nested Conditions
    - Cell Value by Sequential String Matching
    - Cell Value by Regular Expressions
Overview of GIO Code Templates and Code Generation

GenericTemplate(\( R, C, V \))
1: [srcInitRow, srcRow] = GenerateRowCode(\( R, C \), \( V \));
2: [srcInitCol, srcCol] = GenerateColCode(\( R, C \), \( V \));
3: src = "InitFile(); " + FrameBlock pro = Estimation(br, \( R, C \), \( V \)); return for _text = r.Substring(" + pos + "); +_key = matcher.Group(1); " +_end = r.IndexOf(" +_newPos = _index+len(" +child
4: _rStr += getRecordStr(r, r.index, _rIndexes[rowIndex]); " +else { " +if(_rIndexes[rowIndex].key <= r.index
5: _val = Parse(_text, map.GetValType(cellIndex)); " +}, _val); 
6: fb.set(rowIndex, " +node.cellIndex + "}); } } return src;

GenerateRowCode(\( R \))
1: srcInit = ""; src = "";
2: if (Rstructure == Identity)
3: srcInit = ""; 
4: "rowindex = ParseInt(_text); " +
5: "for (key k ∈ \( R \)) +_index = 0;" +
6: "_index = r.IndexOf(k, _index);" +
7: "_end = r.IndexOf(\( \text{delimiter} \), _index);" +
8: "text = r.Substring(_index, _end);" +
9: "rowindex = ParseInt(_text); " +
10: return src;

GenerateColCode(\( V, T \))
1: srcInit = ""; src = "";
2: trie = InitTrie( Root );
3: for ( column v ∈ \( V \))
4: Node node = new Node( \( V \)_key, \( V \)_colIndex, \( V \)_valueType );
5: trie.insert( node ); // node’s key is a list of strings
6: if T > |trie.Root.GetChild|
7: if trie.GetHeight() == |trie.GetNodes|
8: src = GenerateCodeRegular( trie )
9: else
10: src = GenerateCodeRegex( trie.Root, src, " ”);
11: else
12: regexes = EmptySet(); map = EmptyMap();
13: for ( column v ∈ \( V \))
14: regex = BuildRegex( \( V \)_key );
15: regexes.Add( regex );
16: map.Put( \( V \)_key, \( V \)_colIndex );
17: srcInit = MapToString( map ); // convert map to string src
18: src = GenerateCodeRegex( regexes );
19: return srcInit, src;

GenerateCodeTrie( Node node, String pos )
19: if ( node ∈ EndOfColPattern )
20: src = "" + 
21: _end = FindEndPos(" + node.colPattern[0];" +
22: _text = r.Substring(" + pos + "._end);" +
23: _val = Parse(_text, " + node.valType + " );" +
24: fb.set(rowindex, " + node.cellIndex + "._val);" +
25: if ( node.GetChild() > 0 )
26: for ( child in node.GetChild() )
27: src = "" + 
28: _index = r.IndexOf(" + child.key + "._val+ " );" +
29: GenerateCodeTrie(child, newPos);

Details are in the paper!
Reader Generation Example by Nested Conditions

```java
1: _index = r.indexOf("#index", 0);
2: if(_index != -1) {
3:   _index += 6;
4:   _end = r.length();
5:   _text = r.substring(_index, _end);
6:   fb.set(rowIndex, 1, Parse(_text, INT64));
7: }
8: _index = r.indexOf("@", 0);
9: if(_index != -1) {
10:   _index += 2;
11:   _end = r.indexOf(";", _index);
12:   _text = r.substring(_index, _end);
13:   fb.set(rowIndex, 2, _text);
14:   _index = r.indexOf(";", _index);
15:   if(_index != -1) {
16:     _index += 1;
17:     _end = r.indexOf(";", _index);
18:     _text = r.substring(_index, _end);
19:     fb.set(rowIndex, 3, _text);
20:     _index = r.indexOf(";", _index);
21:     if(_index != -1) {
22:       _index += 1;
23:       _end = r.length();
24:       _text = r.substring(_index, _end);
25:       fb.set(rowIndex, 3, _text);
26:     }
27:   }
28: }
```

GIO: Generating Efficient Matrix and Frame Readers for Custom Data Formats by Example
Matthias Boehm
Experiments – Experimental Setting

- **HW:** AMD EPYC 7302 CPU @ 3.0-3.3 GHz (32 cores) 128 GB DDR4 RAM
- **SW:** Ubuntu 20.04.1, OpenJDK 11, Python 3.8, and Clang++10

### Datasets:

<table>
<thead>
<tr>
<th>Dataset</th>
<th>n (nrow)</th>
<th>m (ncol)</th>
<th>o (objects)</th>
<th>Size [GB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMiner-Author (JSON)</td>
<td>1,712,432</td>
<td>Nested</td>
<td>1</td>
<td>0.62</td>
</tr>
<tr>
<td>AMiner-Paper (JSON)</td>
<td>2,092,355</td>
<td>Nested</td>
<td>2</td>
<td>3.7</td>
</tr>
<tr>
<td>Yelp (JSON)</td>
<td>8,635,403</td>
<td>Nested</td>
<td>7</td>
<td>19</td>
</tr>
<tr>
<td>AMiner-Author (Custom)</td>
<td>1,712,432</td>
<td>N/A</td>
<td>N/A</td>
<td>0.5</td>
</tr>
<tr>
<td>AMiner-Paper (Custom)</td>
<td>2,092,355</td>
<td>N/A</td>
<td>N/A</td>
<td>2.1</td>
</tr>
<tr>
<td>HL7 (Custom)</td>
<td>10,240,000</td>
<td>100</td>
<td>N/A</td>
<td>7.5</td>
</tr>
<tr>
<td>Yelp-Review (CSV)</td>
<td>8,635,403</td>
<td>9</td>
<td>Flat</td>
<td>6.5</td>
</tr>
<tr>
<td>Mnist8m (LibSVM)</td>
<td>8,100,000</td>
<td>784</td>
<td>Flat</td>
<td>12</td>
</tr>
<tr>
<td>Susy (LibSVM)</td>
<td>5,000,000</td>
<td>18</td>
<td>Flat</td>
<td>2.4</td>
</tr>
<tr>
<td>Higgs (CSV)</td>
<td>11,000,000</td>
<td>28</td>
<td>Flat</td>
<td>7.5</td>
</tr>
<tr>
<td>Queen (MM)</td>
<td>4,147,110</td>
<td>4,147,110</td>
<td>Flat</td>
<td>4.5</td>
</tr>
<tr>
<td>ReWaste F (CSV)</td>
<td>1,953,434</td>
<td>313</td>
<td>Flat</td>
<td>1.2</td>
</tr>
<tr>
<td>ADF (XML)</td>
<td>10,000,000</td>
<td>146</td>
<td>20</td>
<td>41</td>
</tr>
</tbody>
</table>

- **Baselines:**
  - Apache SystemDS
  - RapidJSON
  - pandas
  - { Gson }
  - HL7
  - <Hapi/>
  - JSON4J

GIO: Generating Efficient Matrix and Frame Readers for Custom Data Formats by Example

Matthias Boehm
Reader Performance on Full Custom Datasets

(a) Single-threaded Readers

(b) Multi-threaded Readers
### Micro-Benchmark Use Cases with Various Data/Query Characteristics

<table>
<thead>
<tr>
<th>Q#</th>
<th>Dataset</th>
<th>Format</th>
<th>(Projection) Query</th>
<th>Nesting &amp; Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>AMiner-Author</td>
<td>JSON</td>
<td><code>index</code></td>
<td>L1</td>
</tr>
<tr>
<td>Q2</td>
<td>AMiner-Author</td>
<td>JSON</td>
<td><code>name, paper_count</code></td>
<td>L1</td>
</tr>
<tr>
<td>Q3</td>
<td>AMiner-Author</td>
<td>JSON</td>
<td><code>index, name, paper_count, citation_number, hIndex</code></td>
<td>L1</td>
</tr>
<tr>
<td>Q4</td>
<td>AMiner-Author</td>
<td>JSON</td>
<td><code>name, affiliations[1, 2, 3, 4]</code></td>
<td>L1, L1 Array</td>
</tr>
<tr>
<td>Q5</td>
<td>AMiner-Paper</td>
<td>JSON</td>
<td><code>index</code></td>
<td>L1</td>
</tr>
<tr>
<td>Q6</td>
<td>AMiner-Paper</td>
<td>JSON</td>
<td><code>title, year</code></td>
<td>L1</td>
</tr>
<tr>
<td>Q7</td>
<td>AMiner-Paper</td>
<td>JSON</td>
<td><code>index, title, year, publication_venue, abstract</code></td>
<td>L1</td>
</tr>
<tr>
<td>Q8</td>
<td>AMiner-Paper</td>
<td>JSON</td>
<td><code>index, references[1, 2, 3, 4]</code></td>
<td>L1, L1 Array</td>
</tr>
<tr>
<td>Q9</td>
<td>Yelp</td>
<td>JSON</td>
<td><code>id</code></td>
<td>L1</td>
</tr>
<tr>
<td>Q10</td>
<td>Yelp</td>
<td>JSON</td>
<td><code>id, text</code></td>
<td>L1</td>
</tr>
<tr>
<td>Q11</td>
<td>Yelp</td>
<td>JSON</td>
<td><code>id, text, business.id, user.id, business.postal_code</code></td>
<td>L1, L2</td>
</tr>
<tr>
<td>Q12</td>
<td>Yelp</td>
<td>JSON</td>
<td><code>id, text, business.id, user.id, business.checkin.date, business.attribute.wifi</code></td>
<td>L1, L2, L3</td>
</tr>
<tr>
<td>Q13</td>
<td>Yelp</td>
<td>JSON</td>
<td><code>business.checkin.date, business.hours.monthday, business.attribute.HhashTV</code></td>
<td>L3</td>
</tr>
<tr>
<td>Q14</td>
<td>AMiner-Author</td>
<td>Custom</td>
<td><code>index</code></td>
<td>N/A</td>
</tr>
<tr>
<td>Q15</td>
<td>AMiner-Author</td>
<td>Custom</td>
<td><code>name, paper_count</code></td>
<td>N/A</td>
</tr>
<tr>
<td>Q16</td>
<td>AMiner-Author</td>
<td>Custom</td>
<td><code>index, name, paper_count, citation_number, hIndex</code></td>
<td>N/A</td>
</tr>
<tr>
<td>Q17</td>
<td>AMiner-Author</td>
<td>Custom</td>
<td><code>name, affiliations[1, 2, 3, 4]</code></td>
<td>N/A</td>
</tr>
<tr>
<td>Q18</td>
<td>AMiner-Paper</td>
<td>Custom</td>
<td><code>index</code></td>
<td>N/A</td>
</tr>
<tr>
<td>Q19</td>
<td>AMiner-Paper</td>
<td>Custom</td>
<td><code>title, year</code></td>
<td>N/A</td>
</tr>
<tr>
<td>Q20</td>
<td>AMiner-Paper</td>
<td>Custom</td>
<td><code>index, title, year, publication_venue, abstract</code></td>
<td>N/A</td>
</tr>
<tr>
<td>Q21</td>
<td>AMiner-Paper</td>
<td>Custom</td>
<td><code>index, references[1, 2, 3, 4]</code></td>
<td>N/A</td>
</tr>
<tr>
<td>Q22</td>
<td>Yelp-Review</td>
<td>CSV</td>
<td><code>id</code></td>
<td>FLAT</td>
</tr>
<tr>
<td>Q23</td>
<td>Yelp-Review</td>
<td>CSV</td>
<td><code>id, text, stars</code></td>
<td>FLAT</td>
</tr>
<tr>
<td>Q24</td>
<td>HL7</td>
<td>Custom</td>
<td><code>evn_code, datetime, reason_code, operator_id</code></td>
<td>N/A</td>
</tr>
<tr>
<td>Q25</td>
<td>HL7</td>
<td>Custom</td>
<td><code>patient_name, birth_day, address, phone_number, account_number</code></td>
<td>N/A</td>
</tr>
</tbody>
</table>
Micro Benchmark - Reader Runtime Performance

(a) Yelp (JSON)

(b) AMiner-Paper (Custom)

(c) HL7 (Custom)
Comparison with Varying Number of Attributes

- **Yelp (JSON)**
- **AMiner Paper (Custom)**
- **Higgs (CSV)**
- **HL7 (Custom)**
- **Susy (LibSVM)**

- RapidJSON is single-threaded and slow
- Projections are not exploited by SystemDS
- Python is doing projection, but it’s still slow
- Python reads sparse formats slowly
- GIO linear scales and robust performance
Conclusions

Summary

• GIO (generated I/O) reader framework for custom text data formats
• GIO automatically identifies position/value mapping rules by giving samples
• Efficiently generates code for efficient, multi-threaded readers for datasets

Conclusion

• GIO is capable of correctly identifying the mapping rules
• Generated readers yield competitive performance
• GIO makes data analysis and modeling easier with custom data formats
• Users can adjust mapping rules and readers manually