Modern Stored Procedures Using GraalVM

Oracle Labs

Matthias Brantner <matthias.brantner@oracle.com>
Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.
Stored Procedures & UDFs

Applications

neworder(...);
neworder(...);
neworder(...);

Database

Stored Procedure

neworder(...) {
    // SQL Query
    // DML Statement
    // SQL Query
}
Problems

- Often vendor specific languages (e.g., PL/SQL, Transact-SQL)
- Hard to find developers ($$$)
- Relatively small ecosystems & marketplaces
- Lacking behind with tool support
- Hard to manage within VCSs
JavaScript UDF Demo
Features

• High-performance JavaScript / TypeScript
• MySQL and Oracle Database
• Driver for executing SQL (also ORM support)
• Support for querying JSON tables
• User-defined JavaScript functions
  – Scalar UDFs
  – Aggregation
  – Table functions
• Deployment tool (allows for integration with JS CIs)
Proliferation of Languages

Programming Language Market Share (TIOBE 8/17)
Embedding New Stored Procedure Languages is Hard

• Choose or implement

• Integrate
  – Integrate with query runtime for UDFs
  – Access to data (no copy, data conversion)
  – Manage & secure system resources
  – Provide driver for executing SQL
  – Provide tooling

• Maintain

Repeat for each new language
Prototype a new language
Parser and language work to build syntax tree (AST), AST Interpreter

Write a “real” VM
In C/C++, still using AST interpreter, spend a lot of time implementing runtime system, GC, …

People complain about performance
Define a bytecode format and write bytecode interpreter

Performance is still bad
Write a JIT compiler Improve the garbage collector
Truffle & Graal

Prototype a new language
- Parser and language work to build syntax tree (AST), AST Interpreter

Truffle
- Optimize AST via profiling and node rewriting

Graal
- Just-in-time compile using partial evaluation (first Futamura projection)
Profiling, Node Rewriting and Compilation

AST Interpreter
Uninitialized Nodes

Node Transitions
Node Rewriting for Profiling Feedback

Uninitialized Integer
Integer

Uninitialized Double
Double

Uninitialized Generic
Generic

String

Compilation using Partial Evaluation

Compiled Code

AST Interpreter Rewritten Nodes

Truffle

Graal
Deoptimization, Node Rewriting and Recompilation
Embedding GraalVM
JavaScript Performance (Octane 1.0 benchmark suite)

Speed-up normalized vs Nashorn JDK9, higher is better

- Box2D
- Crypto
- Deltablue
- EarleyBoyer
- Gameboy
- NavierStokes
- RayTrace
- Richards
- Splay
- Composite

- Nashorn JDK9
- Graal.js Basic
- Graal.js Enterprise
- V8
R Performance

Speedup over latest GNU R version on simple benchmarks

b25

<table>
<thead>
<tr>
<th></th>
<th>gnur</th>
<th>basic</th>
<th>enterprise</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1.73</td>
<td>1.96</td>
</tr>
</tbody>
</table>

shootout

<table>
<thead>
<tr>
<th></th>
<th>gnur</th>
<th>basic</th>
<th>enterprise</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>13.27</td>
<td>14.2</td>
</tr>
</tbody>
</table>
Specialization of Data Conversion (Oracle Number => IEEE 754 double)

Profile length and exponent

<table>
<thead>
<tr>
<th>Length</th>
<th>Exponent</th>
<th>Mantissa</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>193</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>193</td>
<td>46</td>
</tr>
<tr>
<td>3</td>
<td>194</td>
<td>3 21</td>
</tr>
<tr>
<td>2</td>
<td>193</td>
<td>97</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>194</td>
<td>3 25 48</td>
</tr>
</tbody>
</table>

Self-rewriting conversion code

```c
specialize(length, exp, mantissaBytes);

if (length == 2 && exp == 0xc1) { result = mantissaBytes[0] - 1;}
else specialize(length, exp, mantissaBytes);

if (length == 2 && exp == 0xc1) { result = mantissaBytes[0] - 1;}
else if (length == 3 && exp == 0xc2) {
    result = (mantissaBytes[0] - 1) * 100 + (mantissaBytes[1] - 1);});
else specialize(length, exp, mantissaBytes);

... if (length == 2 && exp == 0xc1) { result = mantissaBytes[0] - 1;}
else if (length == 3 && exp == 0xc2) {
    result = (mantissaBytes[0] - 1) * 100 + (mantissaBytes[1] - 1);});
else genericConversion(length, exp, mantissaBytes);
```
Conclusion

• Stored procedures and their challenges
• Demo of JavaScript UDF development and deployment
• Proliferation of languages
• GraalVM to simplify implementation of languages
• GraalVM embedded in various data processing platforms
• Speculative optimization for data conversion
Thank You!
Questions?

matthias.brantner@oracle.com
Integrated Cloud
Applications & Platform Services