Executing Code in the Past: Efficient In-Memory Object Graph Versioning

Pluquet Frédéric
Stefan Langerman
Roel Wuyts

GRASCOMP, ULB, November 2011
In-Memory Object Versioning

«If I want revisit these 3 versions, how can I do it?»

⇒ An ad hoc solution must be built
But...

- It is a recurring problem!
- Eiffel-like Checked Post-Conditions
- Stateful Tracer, Debugger, Google Wave
- More than 20 applications in the theoretical algorithmic domain (computational geometry,...)
- ...

Frédéric Pluquet - histoory@fpluquet.be
Our Solution

In-Memory Object Versioning
for any Object-Oriented Language

• What are the challenges?
  • Be generally applicable!
  • Be expressive!
  • Be efficient in time and in space!
Intuitive approaches

- Deep copy of the system after each change
  ➡ The memory fills up very quickly
- Deep copy of the system at each given interval of time
  ➡ Some useful values can be not saved
- Use database
  ➡ Not the same goal
What we really want

• Only keep
  • interesting old values
  • of interesting fields
  • of interesting objects

Snapshots

Field Selection
Example
Implementation

History

• Early version in Java
• Stable version in Smalltalk
  • A standard library
  • Can be loaded in any application
  • Existing code does not need modification (bytecode instrumentation)
Smalltalk integration

Demo
Kinds Of Versioning

- Linear
- Backtracking
- Branching
## Complexities

### Linear

- Take a snapshot: $O(1)$
- Access a non selected field: $O(1)$
- Store in a selected field: $O(1)$
- Read last value of a selected field: $O(1)$
- Read old value of a selected field: $O(\log \#\text{values})$
## Complexities

### Backtracking

<table>
<thead>
<tr>
<th>Action</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Take a snapshot</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Backtrack</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Access a non selected field</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Store in a selected field</td>
<td>$O(1)*$</td>
</tr>
<tr>
<td>Read last value of a selected field</td>
<td>$O(1)*$</td>
</tr>
<tr>
<td>Read old value of a selected field</td>
<td>$O(\log #\text{values})$</td>
</tr>
</tbody>
</table>

* amortized time
# Complexities

## Branching

<table>
<thead>
<tr>
<th>Action</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Take a snapshot</td>
<td>$O(1)^*$</td>
</tr>
<tr>
<td>Create a branch</td>
<td>$O(1)^*$</td>
</tr>
<tr>
<td>Access a non selected field</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Store in a selected field</td>
<td>$O(\log #\text{values})$</td>
</tr>
<tr>
<td>Read last value of a selected field</td>
<td>$O(\log #\text{values})$</td>
</tr>
<tr>
<td>Read old value of a selected field</td>
<td>$O(\log #\text{values})$</td>
</tr>
</tbody>
</table>

* amortized time
# For all kinds of versioning

<table>
<thead>
<tr>
<th>Access a non selected field</th>
<th>O(1)</th>
</tr>
</thead>
</table>

If we do not select any field, your system keeps same performance.
Performance Store

- **Average Time (ms) to save one state**
- **Slowdown factor**
  - Branching at end
  - Branching at root
  - Backtracking
  - Linear
  - Instrumented, not selected
  - Ephemeral

- **# states saved in the versioned field**
  - 0
  - 0.005
  - 0.01
  - 0.015
  - 0.02
  - 0.025

- **Frédéric Pluquet - histoory@fpluquet.be**

ULB
**Performance**

**Read**

- **Slowdown factor**
  - Synthetic worst case update: 7
  - Real cases update: 1.3 to 2

**Graph:**

- **Average time (ms) to retrieve one state**
  - Y-axis values: 0.0002 to 0.008

- **Slowdown factor**
  - X-axis values: 1 to 100000

- Graph lines represent:
  - Branching at root
  - Branching at end
  - Backtracking - any state
  - Backtracking - last state
  - Linear - any state
  - Linear - last state
  - Instrumented, Not selected
  - Ephemeral

- **# states saved in a versioned field**
  - X-axis values range from 1 to 100000

---

Frédéric Pluquet - histoory@fpluquet.be
Random Treap Store

- Branching
- Backtracking
- Linear - all states
- Linear - consistent states
- Linear - no states
- Instrumented, not selected
- Ephemeral

Average time (ms) per insertion vs. # insertions in a random treap.

- 5x speedup
- 3x speedup
Conclusion

• A general and efficient model for in-memory object versioning for object-oriented languages
• Theoretical algorithms with implementation challenges
• We implemented it in Smalltalk
  • 3 kinds of versioning
  • Non-intrusive library
  • Efficient
Thank you!

- histoory@fpluquet.be

- **History** on Google
