A Study of Erlang ETS Table Implementations and Performance

Or: Judy Arrays Are Amazing Data Structures

Scott Lystig Fritchie

<slfritchie@snookles.com>

Snookles Music Consulting
Overview

- ETS table data structures
- Judy arrays
- “Contiguous Key Problem”
- Solving the “Contiguous Key Problem”
- Performance results
Audience

- Erlang community
  - Using ETS directly
  - Using ETS indirectly via Mnesia and other OTP applications
- C/C++ developers using hash tables and balanced trees
  - Performance gains by using “Judy arrays” can be impressive
  - Consider using Judy arrays in your applications
ETS Table Implementations

- Types included in Erlang/OTP:
  - AVL balanced binary tree: ordered_set
  -Resizable linear hash table: set, bag, duplicate_bag

- New research types:
  - In-memory B-tree: btree
  - Judy arrays (based on tries): judysl, judyesl, judyeh
Judy Arrays

- Invented by Doug Baskins, implemented by Hewlett-Packard.
- Named after Baskins's sister.
- Source code now available under GNU LGPL license.
- Source & docs at http://judy.sourceforge.net/
Judy Arrays (continued)

- Judy arrays are dynamic arrays
  - Index = 1 word, 32- or 64-bit
  - Value = 1 bit or 1 word
- Handles small & large populations, sparse & dense populations, *no tuning parameters!*
- Implemented as a logical 256-ary trie
Data Structures Review: The Trie

*ABC ... Z Root Node

*ABCDFGHJKLMNOPQRSTUVWXYZ

*ABC ... QRST ... Z

*ABC ... JKL ... Z

A AIR AISLE

ASK ASKED ASKING

A Study of Erlang ETS Table Implementations and Performance – p.7/19
Data Structures Review: The Trie

Root Node

*ABC … Z

*ABCDEFGHIJKLMNOPQRSTUVWXYZ

*ABC … QRST … Z

*ABC … JKL … Z

A

AIR

AISLE

ASK

ASKED

ASKING
Data Structures Review: The Trie

Root Node

*ABC ... Z

*ABCDE

*ABCDEF

*ABCDEFGHIJKLMNOPQRSTUVWXYZ

A

AIR

AISLE

ASK

ASKED

ASKING

A Study of Erlang ETS Table Implementations and Performance – p.7/19
Data Structures Review: The Trie

Root Node

*ABC...Z

*ABCDEFGHIJKLMNOPQRSTUVWXYZ

*ABC...QRST...Z

*ABC...JKL...Z

*ABCDEFGHIJ...Z

A  AIR  AISLE

ASK ASKED ASKING
Data Structures Review: The Trie

Root Node:

*ABC ... Z

*ABCDE

*ABC ... QRST ... Z

*ABC ... JKL ... Z

*ABCDEF

ASK

ASKED

ASKING
JudySL: A Trie of JudyL Arrays

JudyL

BE\0 BEAR BEEH

Short-cut leaf

HOUS I VE\0

Short-cut leaf

JudyL

E-PAINT\0

Words: BE, BEAR, BEEHOUSE–PAINT, BEEHIVE
JudyESL: A Variation of JudySL

Words: BE, BEAR, BEEHIVE
The Contiguous Key Problem

Example tuple

```
{scott, "scott", 
  "To", 
  "scott"}
```

Tuple, size = 4:
- element 0 = atom #A
- element 1
- element 2
- element 3

Atom Table

<table>
<thead>
<tr>
<th>number</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>A − 1</td>
<td>foo</td>
</tr>
<tr>
<td>A</td>
<td>scott</td>
</tr>
<tr>
<td>A + 1</td>
<td>bar</td>
</tr>
</tbody>
</table>

Ref-counted binary:
- refcount = 2
- size = 8
- data = 84,111,58,115,99,111,116,116

Tuple, size = 4:
- element 0 = atom #A
- element 1
- element 2
- element 3
Judy-Based Tables

- judysl table type
  - Serialized key = 
    ```
    encode_NUL_bytes(term_to_binary(Key))
    ```

- JudyESL table type
  - The JudyESL library uses explicit string length, not NUL termination.
  - Serialized key = `term_to_binary(Key)`

**NOTE**: JudySL and JudyESL preserve lexicographic sort order of serialized keys, *not of original Erlang key terms.*
Judy-Based Tables (continued)

- Judyeh table type
- JudyL array for hash table: $2^{32}$ hash buckets!
- No serialization, unlike judysl and judyesl
- No meta-trie: search one JudyL array, not several
- Hash collision rate $< 0.2\%$ for 7 million items
Experiment Design

- Intentionally maximize time executing ETS-related code.
- Show table differences as much as possible.
- Benchmark time in ETS-related code: 35-70%
- SCCT time in ETS-related code: 18%
- All other parts of VM unchanged.

Benchmark result graphs
- Overall, set is fastest “old” table type.
- All run times normalized against set’s time.
- Run time $< 1.0$ → better

CPU cache size reflected between $10^4$ and $10^5$ keys.
Sequential Insertion Into Empty Table

![Graph showing time relative to 'set' type vs number of keys for different table implementations: set, judyeh, ordered_set, judysl, judyesl, btree. The graph indicates that as the number of keys increases, the time relative to 'set' type decreases for all implementations. The set implementation consistently shows the lowest time relative to 'set' type.](image-url)
Sequential Insertion, Per 1K Keys

![Graph showing the performance of different data structures for sequential insertion. The x-axis represents the number of keys, and the y-axis represents the seconds per thousand keys. The graph compares set, judyeh, judysl, judyesl, ordered_set, and btree implementations. The trend shows an increase in time as the number of keys increases.]
Random Lookup in Full Table

![Graph showing random lookup times for different set types]

- **Time relative to 'set' type**
- **Number of keys**
- **Labels:**
  - set
  - judyeh
  - judysl
  - judyesl
  - ordered_set
  - btree

A Study of Erlang ETS Table Implementations and Performance – p.16/19
Forward Traversal of Full Table
### Memory Utilization

<table>
<thead>
<tr>
<th>Table type</th>
<th>Memory used by 70K keys</th>
<th>Memory used by 21M keys</th>
<th>Difference from set</th>
</tr>
</thead>
<tbody>
<tr>
<td>btree</td>
<td>10.4MB</td>
<td>1,055MB</td>
<td>7.7%</td>
</tr>
<tr>
<td>judyeh</td>
<td>10.4MB</td>
<td>1,036MB</td>
<td>5.7%</td>
</tr>
<tr>
<td>judysl</td>
<td>10.4MB</td>
<td>1,033MB</td>
<td>5.4%</td>
</tr>
<tr>
<td>judyesl</td>
<td>11.3MB</td>
<td>1,324MB</td>
<td>35%</td>
</tr>
<tr>
<td>ordered_set</td>
<td>10.7MB</td>
<td>1,129MB</td>
<td>15%</td>
</tr>
<tr>
<td>set</td>
<td>10.2MB</td>
<td>980MB</td>
<td>—</td>
</tr>
</tbody>
</table>
Conclusion

- Judy array-based ETS tables perform very well for ETS table sizes that exceed CPU cache size.
- Table traversal performance is probably fixable.
- Performance gain of Judy-based tables far exceeds extra memory consumption.
- JudySL- or JudyESL-based technique could perform better than set and still preserve key sort order.
- Using Judy arrays in a “real world” application can improve performance. Your application can probably benefit, too.