More Scalable Ordered Set for ETS
Using Adaptation

Kostis Sagonas
Kjell Winblad

Department of Information Technology
Uppsala University, Sweden

5 Sept 2014
What is ETS?

- Erlang Term Storage
- Key-value store
- In-memory database
- Shared memory
ETS Table Types and their Options

- Creates a new shared **hash based** table with **write concurrency** enabled

```
TID1 = ets:new(tab, [set, public,
  {write_concurrency, true}])
```

- **write concurrency** enables fine grained locking

- Other hash based table types are **bag** and **duplicate bag**
ETS Table Types and their Options

- Creates a new **search tree based** table with **read concurrency** enabled

  ```erl
  TID2 = ets:new(tab, [ordered_set, public,
                         {read_concurrency, true}])
  ```

- Current ordered set does not support fine grained locking
- read concurrency enables frequent-read-optimized readers-writer locks
  - One cache line per scheduler
  - Read only operations do not interfere each other
Use of ETS

- Insert an Erlang tuple into a table

```
ets:insert(TID, {42, "A value"}),
```

Many other functions like `delete`, `match`, `foldl`, etc.
Use of ETS

- Insert an Erlang tuple into a table
  ```erlang
  ets:insert(TID, {42, "A value"}),
  ```

- Lookup a value
  ```erlang
  ets:lookup(TID, 42),
  ```
Use of ETS

- Insert an Erlang tuple into a table
  
  ```erlang
  ets:insert(TID, {42, "A value"}),
  ```

- Lookup a value
  
  ```erlang
  ets:lookup(TID, 42),
  ```

- Find first element
  
  ```erlang
  FirstElem = ets:first(TID),
  ```

- Find next element
  
  ```erlang
  NextElem = ets:next(TID, FirstElem).
  ```
Use of ETS

- Insert an Erlang tuple into a table
  
  ```erlang```
  ets:insert(TID, {42, "A value"}),
  ```

- Lookup a value
  
  ```erlang```
  ets:lookup(TID, 42),
  ```

- Find first element
  
  ```erlang```
  FirstElem = ets:first(TID),
  ```

- Find next element
  
  ```erlang```
  NextElem = ets:next(TID, FirstElem).
  ```

- Many other functions like delete, match, foldl, etc.
Current ETS Scalability Benchmark

Benchmark
- ets_bench from bencherl
- key range \([1, 2^{21}]\), \(2^{21} \approx 2 \times 10^6\)
- Three phases:
  1. Insert phase: inserts \(2^{20}\) random keys, \(2^{20} \approx 10^6\)
  2. Update and read phase, parameter for percentage update
  3. Delete phase: deletes \(10^6\) random keys

Machine
- Four Intel(R) Xeon(R) CPU E5-4650 CPUs (2.70GHz), eight cores each
  - total of 32 physical cores, each with hyperthreading
- The machine has 128GB of RAM and is running Debian Linux 3.10.17-amd64 and Erlang/OTP release 17.0
100% Updates

Number of Threads

Operations / Microsecond

ordset
ordset,r
set
set,r
100% Updates

More Scalable Ordered Set for ETS Using Adaptation
http://www.it.uu.se/research/group/languages/software/ca_tree
100% Updates

![Graph showing performance metrics for different sets and operations with varying number of threads.](http://www.it.uu.se/research/group/languages/software/ca_tree)
100% Lookups

![Graph showing operations per microsecond vs number of threads.]
100% Lookups

![Graph showing operations per microsecond for ordset and ordset,r against number of threads.](http://www.it.uu.se/research/group/languages/software/ca_tree)
100% Lookups

---

Motivation

CA Trees

Performance

Future Work

Conclusion

More Scalable Ordered Set for ETS Using Adaptation
http://www.it.uu.se/research/group/languages/software/ca_tree
100% Lookups

![Graph showing performance of different operations under varying number of threads]

- ordset
- ordset,r
- set
- set,r

Operations / Microsecond vs. Number of Threads
100% Lookups

![Graph showing performance metrics for different operations across various thread counts. The graph compares ordset, ordset,r, set, set,r, and set,w operations. Each data point is accompanied by error bars to indicate variability. The x-axis represents the number of threads, ranging from 0 to 70, while the y-axis represents operations per microsecond, ranging from 0 to 40.]
100% Lookups

Operations / Microsecond

Number of Threads

Operations / Microsecond

Number of Threads

ordset
ordset,r
set
set,r
set,w
set,r,w
50% Lookups, 50% Updates
50% Lookups, 50% Updates

More Scalable Ordered Set for ETS Using Adaptation
http://www.it.uu.se/research/group/languages/software/ca_tree
99% Lookups, 1% Updates
99% Lookups, 1% Updates

![Graph showing operations per microsecond vs number of threads for different operations: ordset, ordset,r, set, set,r, set,w, set,r,w.](http://www.it.uu.se/research/group/languages/software/ca_tree)
Summary of Current ETS Scalability

- **Something needs to be done with ordered set to make it scale when there are parallel writes!**
  - Huge slow down even with 99% reads
We Want

Wish list for ordered set

- Good scalability
- Reuse code from the current ETS implementation
- Low overhead in sequential case
  - Current algorithms for concurrent ordered sets sacrifice sequential performance and memory consumption for scalability
Contention Adapting Binary Search Trees (CA trees)

Key Ideas
- Start with sequential binary search tree protected by a lock
- Collect statistics from the lock
- Adapt the tree according to the statistics
CA Tree Components

Statistics Collecting Lock

- The lock has an associated counter C
- C += 250 if needed to wait to acquire the lock
- C -= 1 if not needed to wait to acquire the lock
- Adapt when C reach thresholds
  - E.g. -1000 and 1000
CA Tree Components

Sequential Ordered Set Data Structure

- Requires support for split and join
  - The **split** operation splits a tree into two so that all keys in one tree are smaller than the keys in the other
  - The **join** operation merges two trees given that all keys in one tree are smaller than the keys in the other
  - $O(\log(N))$ implementations for **AVL trees**, **Red-Black trees**, **Treaps**, etc.
CA Tree Structure
CA Tree Animation
CA Tree Animation
CA Tree Animation
CA Tree Animation
CA Tree Animation

CA Trees
Performance
Future Work
Conclusion

More Scalable Ordered Set for ETS Using Adaptation
http://www.it.uu.se/research/group/languages/software/ca_tree
CA Tree Animation
Integration into ETS

- Reuse code from current ordered_set
Integration into ETS

- Reuse code from current ordered_set
- Routing layer needs special memory management
  - Currently quiescent state based reclamation
  - Better to reuse memory reclamation system for lock free data structures that is already integrated into the Erlang runtime system
The same benchmark again
100% Updates

![Graph showing operations per microsecond for different data structures with varying number of threads. The graph compares ordset, AVL-CA tree, and set.]

- **Context**: Provides background information.
- **Motivation**: Explains why the research is important.
- **CA Trees**: Discusses the specifics of using CA Trees.
- **Performance**: Analyzes the performance of different data structures.
- **Future Work**: Suggests areas for further research.
- **Conclusion**: Summarizes the findings and implications.
100% Lookups

Graph showing the performance of different data structures (ordset, AVL-CA tree, set) in terms of number of operations per microsecond against the number of threads. The graph indicates that AVL-CA tree performs better than ordset and set as the number of threads increases.
50% Update 50% Lookup Scalability

![Graph showing operations per microsecond for different thread counts and data structures: ordset, AVL-CA tree, and set. The x-axis represents the number of threads, and the y-axis represents operations per microsecond. The graph indicates performance trends and scalability for these data structures.]
20% Update 80% Lookup Scalability

![Graph showing comparison of ordset, AVL-CA tree, and set operations per microsecond across different numbers of threads.](http://www.it.uu.se/research/group/languages/software/ca_tree)
1% Update 99% Lookup Scalability

![Graph showing performance metrics for different data structures.](http://www.it.uu.se/research/group/languages/software/ca_tree)
Summary of Scalability Improvements

- The CA trees does not suffer from large slow down
- Scales reasonably well on one chip
- Update heavy scenarios scale far from perfect on NUMA
  - Centralized statistics counter in memory management
  - set has even more problems on NUMA
Sequential Performance, 80% reads

![Graph showing performance of ordset, AVL-CA tree, and set with set size vs. time in seconds for different set sizes (2^3 to 2^27). The ordset has the lowest time for smaller set sizes, AVL-CA tree has a consistent time across all set sizes, and set has a higher time for larger set sizes.]
Future Work

To integrate into ETS

- Implement the whole ETS interface
  - Code can be reused from current implementation
- Decide how to integrate it into ETS
  - Always activate on public tables
    - Read only case might suffer
  - Only activate when write_concurrency is specified
Work Already Done

- Compare to other concurrent ordered set data structures
- Investigate optimization for read heavy scenarios
  - RW-locks, Sequence Lock, Hardware Lock Elision (HLE)
- Discuss algorithm in detail

More in a Technical Report

http://www.it.uu.se/research/group/languages/software/ca_tree
Concluding Remarks

- Performance of concurrent writes on ordered set can be substantially improved in ETS
- CA tree based table scales even better than the current hash based implementation on a NUMA system
- good scalability with low sequential overhead
Questions?