Vector Clocks in Coq
An Experience Report

Christopher Meiklejohn

Basho Technologies, Inc.
Cambridge, MA 02139

cmeiklejohn@basho.com

March 6, 2014
Outline of the talk

▶ Introduction
▶ Background
▶ Implementation
▶ Evaluation
▶ Future Work
Introduction

- Goals of the project
- Goals of the talk
- Contributions
- Out of scope
Goals of the project

- Distributed data structures (RICON West, 2012)
- Explore applicability of code extraction from Coq
- Attempt to provide an alternative to rigorous testing
- Prevent flaws in building QuickCheck models
Goals of the talk

- Introduction to Coq
- Introduction to Core Erlang
- Introduction of vector clocks
- Overall experience report of implementation
Contributions

- Coq model providing vector clock implementation
- Extracted Erlang model from the Coq proof assistant
- Erlang glue-code support module
- Detailed experience report
- Rebar extension
Out of scope

- Verification of the actual model
- Proofs, theorems, lemmas, axioms, etc...
- Efficiency
Background

- Coq
- Core Erlang
- verlang
- Vector clocks
Coq

- Interactive theorem prover
- Dependently typed programming language
- Code extraction; Scheme, Haskell, OCaml, Core Erlang
Example Coq Inductive Data Type

Inductive nat : Type :=
| 0 : nat
| S : nat nat.
Fixpoint ble nat (n m : nat) {struct n} : bool :=
    match n with
    | 0 => true
    | S n =>
        match m with
        | 0 => false
        | S m => ble nat n m end
    end.
Core Erlang

- Intermediate representation of Erlang
- Designed for programatic manipulation
- Simple grammar
- $c(module\_name, [to\_core])$, $c(module\_name, [from\_core])$. 
Example Core Erlang Function

`ble_nat'/2 = fun (_n, _m) ->
    case _n of
        '0' when 'true' ->
            'True'
        {'S', _n@} when 'true' ->
            case _m of
                '0' when 'true' ->
                    'False'
                {'S', _m@} when 'true' ->
                    call 'vvclock':'ble_nat'
                    ( _n@
                      , _m@
                    )
            end
    end
end
verlang

- Experimental extraction module for Coq
- Extracts to Core Erlang from MiniML
- Number of caveats
verlang caveats

- Lack of module nesting
- No currying
- Intra- vs. inter-module calls
- receive

receive
Vector clocks

- Method for reasoning about events in a distributed system.
- Identifying causal vs. concurrent events.
- List of pairs; made of up actors and operation counts.
- Structurally the same as version vectors; different semantics.
Implementation

- Vector clocks in Coq
- Code extraction to Core Erlang
- Adapter layer
Vector clocks in Coq

- Provide compatible API for use with Riak Core
- fresh, increment, equal, descends, merge, get_counter, get_timestamp, all_nodes, prune
Definition actor := nat.
Definition count := nat.
Definition timestamp := nat.

Definition clock := prod actor (prod count timestamp).

Definition vclock := (list clock)%type.
Vector clocks in Coq: \textit{increment}

\begin{verbatim}
Definition increment (actor : actor) :
    (vclock : vclock) :=
match find (fun clock => match clock with
    | pair x _ => beq_nat actor x
    end) vclock with
| None =>
    cons (pair actor (pair init_count init_timestamp))
    vclock
| Some (pair x (pair count timestamp)) =>
    cons (pair x (pair (incr_count count)
        (incr_timestamp timestamp)))
    (filter (fun clock => match clock with
        | pair x _ =>
            negb (beq_nat actor x)
        end) vclock)
\end{verbatim}
Vector clocks in Coq: \textit{merge}

Definition max' (vclock : vclock) (clock : clock) :=
  match clock with
  | pair actor (pair count timestamp) =>
    match find (fun clock => match clock with
    | pair x _ => beq_nat actor x
    end) vclock with
    | None =>
      cons (pair actor (pair count timestamp)) vclock
    | Some (pair _ (pair y z)) =>
      cons (pair actor (pair (max count y) (max timestamp z)))(filter (fun clock =>
        match clock with
        | pair x _ => negb (beq_nat actor x)
        end) vclock)
    end
  end.

Definition merge (vc1 vc2 : vclock) := fold_left max' vc1 vc2.
Vector clocks in Coq: *prune*

Fixpoint prune’
  (vclock : vclock)
  (small large : nat)
  (young old : timestamp) :=
match vclock with
  | nil =>
    vclock
  | pair actor (pair count timestamp) :: clocks =>
    match (ble_nat (length vclock) small) with
    | true =>
      vclock
    | false =>
      match (ble_nat timestamp young) with
      | true =>
        vclock
Definition descends (vc1 vc2 : vclock) :=
    match fold_left descends’ vc2 (pair true vc1) with
    | pair false _ =>
      false
    | pair true _ =>
      true
    end.
Code extraction to Core Erlang

- Missing data constructors
- Incorrectly qualified calls
- Lack of currying
Missing data constructors

'fresh'/0 = fun () ->
  []
Incorrectly qualified calls

call 'vvclock.VVClock': 'ble_nat'
   ( _actor
     , _a
   )
'$\text{descends}'/2 = \text{fun} \ (\_\text{vc1}, \_\text{vc2}) \rightarrow$
\begin{align*}
\text{case} \ \text{call} \ '\text{Coq.Lists.List}':'\text{fold_left}' \\
\quad (\ '\text{descends@}' \\
\quad \quad , \_\text{vc2} \\
\quad \quad , \{ \ '\text{Pair}' \\
\quad \quad \quad , \ '\text{True}' \\
\quad \quad \quad , \_\text{vc1} \\
\quad \quad } \\
\quad \text{of} \\
\end{align*}
Lack of currying

Definition find’’ (actor : actor) :=
    fun clock : clock => match clock with
        | pair x _ => negb (beq_nat actor x)
    end.

'find@'/2 = fun (_actor, _clock) ->
    case _clock of
        { 'Pair'
            , _c
            , _x
        } when 'true' ->
            call 'Coq.Arith.EqNat':'beq_nat'
                ( _actor
                    , _c
                )
        end
Adapter layer

- Type conversions
- Timestamps; model as Peano numbers
- Actors; model as Peano numbers or Strings
- Environment variables
- API normalization
- Circular dependencies
Type conversions

natural_to_peano(0) -> '0';
natural_to_peano(Natural) -> {'S', natural_to_peano(Natural - 1)}.

peano_to_natural('0') -> 0;
peano_to_natural({'S', Peano}) -> 1 + Peano.
equal(VClock1, VClock2) ->
    case vvclock:equal(VClock1, VClock2) of
        'True' ->
            true;
        'False' ->
            false
    end.

descends(VClock1, VClock2) ->
    case vvclock:descends(VClock1, VClock2) of
        'True' ->
            true;
        'False' ->
            false
    end.
timestamps

```erlang
timestamp() ->
    calendar:datetime_to_gregorian_seconds(erlang:universaltime()).

peano_timestamp() ->
    term_to_peano(timestamp()).
```
Actors

Inductive string : Set :=
| EmptyString : string
| String : ascii → string → string.

Definition zero := Ascii false
    false
    false
    false
    false
    false
    false
    false.
    false.
prune(VClock, _Timestamp, BProps) ->
    Old = term_to_peano(get_property(old_vclock, BProps)),
    Young = term_to_peano(get_property(young_vclock, BProps)),
    Large = term_to_peano(get_property(large_vclock, BProps)),
    Small = term_to_peano(get_property(small_vclock, BProps)),
    vvclock:prune(VClock, Small, Large, Young, Old).
API normalization

merge([VClock1, VClock2|VClocks]) ->
    merge([vvclock:merge(VClock1, VClock2)|VClocks]);
merge([VClock]) ->
    VClock;
merge([]) ->
    [].

increment(Actor, VClock) ->
    vvclock:increment(term_to_peano(Actor), VClock).
Circular dependencies

%%% Call into vvclock.core from vclock.erl
increment(Actor, VClock) ->
    vvclock:increment(term_to_peano(Actor), VClock).

%%% Calls back out to vclock for Riak/Erlang specifics
'init_timestamp'/0 = fun () ->
    call 'vclock': 'peano_timestamp' ()
Evaluation

- Passing test suite
- Performance problems
  - Inefficient implementations
  - Use of naturals, strings or other inductive types
- Testability; type conversion to/from
Future Work

- Fixing bugs in verlang
- Explore other applications; CRDTs
- Adapter layer; performance, testability
- QuickCheck or PropEr integration
Thanks!

▸ Questions?