ORACLE®

Design and Implementation of HAMM

Haskell Authenticated Modular Maps

Victor Miraldo, Harold Carr, Alex Kogan, Mark Moir, Maurice Herlihy
Oracle Labs
September 21, 2018

Motivation

From a blockchain participant’s perspective:

- To start verifying, it needs a state.
- Traditionally, download and verify all transactions since forever. This data grows large.
- Instead, transfer just the necessary part of the state first.
- Fetch the other parts of the state as need arises.

Motivation

From a blockchain participant's perspective:

- To start verifying, it needs a state.
- Traditionally, download and verify all transactions since forever. This data grows large.
- Instead, transfer just the necessary part of the state first.
- Fetch the other parts of the state as need arises.

ORACLE

Motivation

From a blockchain participant's perspective:

- To start verifying, it needs a state.
- Traditionally, download and verify all transactions since forever. This data grows large.
- Instead, transfer just the necessary part of the state first.
- Fetch the other parts of the state as need arises.

Motivation

From a blockchain participant's perspective:

- To start verifying, it needs a state.
- Traditionally, download and verify all transactions since forever. This data grows large.
- Instead, transfer just the necessary part of the state first.
- Fetch the other parts of the state as need arises.

Motivation

From a blockchain participant's perspective:

- To start verifying, it needs a state.
- Traditionally, download and verify all transactions since forever. This data grows large.
- Instead, transfer just the necessary part of the state first.
- Fetch the other parts of the state as need arises.

The Problem

1. To be able to start participation in a blockchain-like system with partial state.
2. To verify this *summary* against some obtained hash.
3. To fetch and verify missing pieces over time, amortizing the cost of starting to verify.

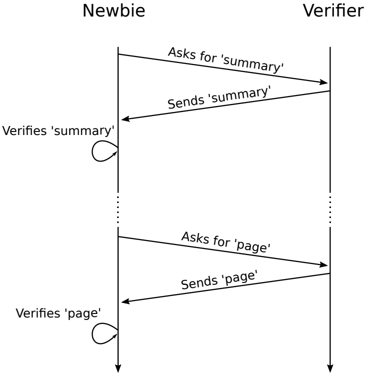
The Problem

1. To be able to start participation in a blockchain-like system with partial state.
2. To verify this *summary* against some obtained hash.
3. To fetch and verify missing pieces over time, amortizing the cost of starting to verify.

The Problem

1. To be able to start participation in a blockchain-like system with partial state.
2. To verify this *summary* against some obtained hash.
3. To fetch and verify missing pieces over time, amortizing the cost of starting to verify.

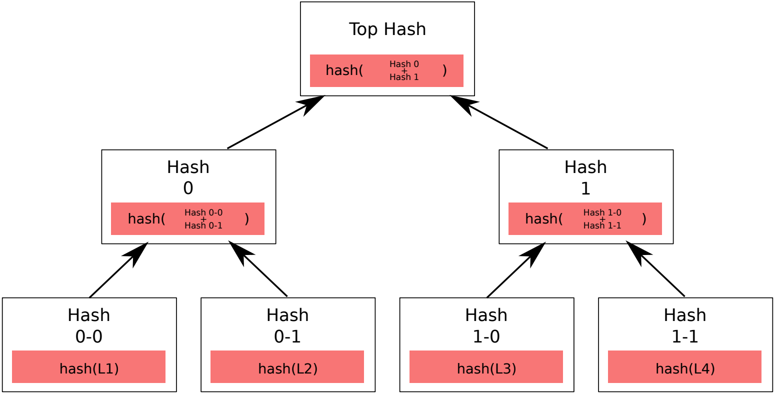
Verifier Comming Online



Merkle Trees
[Merkle,1979]

Verification of the state is not novel. One could use *Merkle Trees* to construct proofs of membership or compare roots.

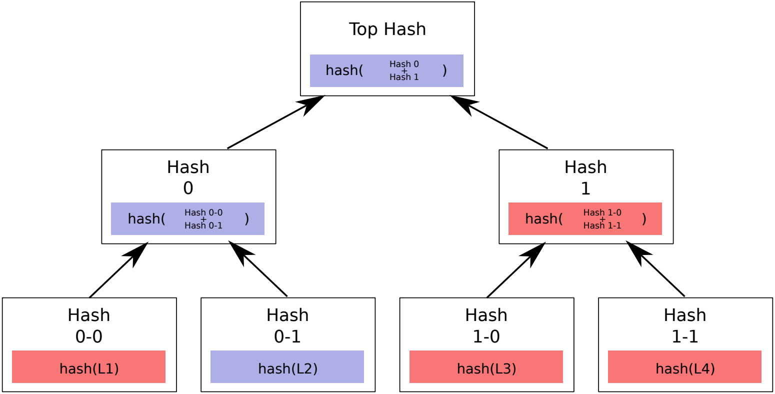
Merkle Trees
[Merkle,1979]



Merkle Root is the *Top Hash*.

Merkle Trees

[Merkle,1979]



Prove L_2 is member: give L_2, $hash\ 0{-}0$ and $hash\ 1$.

HAMM 101

DSL for combining key-value store components.

Quickly study different map architectures, eg:

Different add-ons alter the behavior of base maps.

HAMM 101

DSL for combining key-value store components.

Quickly study different map architectures, eg:

```
myMap : BoundedCacheOf b
            (BloomOf h m
               (PagesOf [3, 4, 4]
                  (PartialOf RB)))
          String Int
myMap = fromList [("A", 0), ("X", 10)]
```

Different add-ons alter the behavior of base maps.

HAMM 101

DSL for combining key-value store components.

Quickly study different map architectures, eg:

$$
\begin{array}{l}
\mathit{myMap} : \mathit{BoundedCacheOf}\ b \\
\qquad\qquad\qquad (\mathit{BloomOf}\ h\ m \\
\qquad\qquad\qquad\quad (\mathit{PagesOf}\ [3,4,4] \\
\qquad\qquad\qquad\qquad (\mathit{PartialOf}\ \mathit{RB}))) \\
\qquad\qquad \mathit{String}\ \mathit{Int} \\
\mathit{myMap} = \mathit{fromList}\ [(\texttt{"A"},0),(\texttt{"X"},10)]
\end{array}
$$

Different add-ons alter the behavior of base maps.

HAMM 101

DSL for combining key-value store components.

Quickly study different map architectures, eg:

```
myMap : BoundedCacheOf b
               (BloomOf h m
                  (PagesOf [3, 4, 4]
                     (PartialOf RB)))
        String Int
myMap = fromList [("A", 0), ("X", 10)]
```

Different add-ons alter the behavior of base maps.

Interlude: This presentation

- `hamm` is under active development.
- Some of the content is novel from paper.
- Some code is slightly different.

Interlude: This presentation

- `hamm` is under active development.
- Some of the content is novel from paper.
- Some code is slightly different.

Interlude: This presentation

- `hamm` is under active development.
- Some of the content is novel from paper.
- Some code is slightly different.

Cooking HAMM from *Data.Map*

Take *Data.Map.lookup* as an example:

$$\begin{aligned} \mathit{lookup} &:: (\mathit{Ord}\ k) \\ &\Rightarrow k \to \mathit{Map}\ k\ v \to \mathit{Maybe}\ v \end{aligned}$$

Cooking HAMM from *Data*.*Map*

Abstract away *Map* for a type variable $c :: * \to * \to *$

$$\begin{aligned} &lookup :: (Ord\ k) \\ &\qquad \Rightarrow k \to c\ k\ v \to Maybe\ v \end{aligned}$$

Cooking HAMM from *Data.Map*

Abstract away *Ord* by a type family

$$\begin{aligned} &\mathit{lookup} :: (\mathit{IsMapCnstr}\ c\ k\ v) \\ &\qquad \Rightarrow k \rightarrow c\ k\ v \rightarrow \mathit{Maybe}\ v \end{aligned}$$

Cooking HAMM from *Data.Map*

Allow for arbitrary errors

$$\begin{aligned} &\mathit{lookup} :: (\mathit{IsMapCnstr}\ c\ k\ v) \\ &\qquad \Rightarrow k \to c\ k\ v \to \mathit{Except}\ (\mathit{Err}\ c)\ (\mathit{Maybe}\ v) \end{aligned}$$

Cooking HAMM from *Data.Map*

Parametrize everything with a Monad

$$\begin{aligned} &\mathit{lookup} :: (\mathit{IsMapCnstr}\ m\ c\ k\ v, \mathit{Monad}\ m) \\ &\qquad \Rightarrow k \to c\ k\ v \to \mathit{ExceptT}\ (\mathit{Err}\ c)\ m\ (\mathit{Maybe}\ v) \end{aligned}$$

Cooking HAMM from *Data.Map*

Wrap it in a typeclass

```
class IsMap (c :: * → * → *) where
  type Err          c        :: *
  type IsMapCnstr m c k v :: Constraint
  lookup :: (Monad m, IsMapCnstr m c k v)
         ⇒ k → c k v → ExceptT (Err c) m (Maybe v)
  ...
```

Add-ons

- Have kind $(* \to * \to *) \to * \to * \to *$

- *add-ons* $\approx$ *monad transformer*

- Alter the implementation under same API

```
instance (IsMap c) ⇒ IsMap (BloomOf h m c) where
  lookup k (BloomOf blf c)
      | bloomMember k blf = lookup k c
      | otherwise         = return Nothing
```

Add-ons

- Have kind $(* \to * \to *) \to * \to * \to *$

- *add-ons* $\approx$ *monad transformer*

- Alter the implementation under same API

$$
\begin{array}{l}
\textbf{instance}\ (\mathit{IsMap}\ c) \Rightarrow \mathit{IsMap}\ (\mathit{BloomOf}\ h\ m\ c)\ \textbf{where} \\
\quad \mathit{lookup}\ k\ (\mathit{BloomOf}\ \mathit{blf}\ c) \\
\qquad \mid \mathit{bloomMember}\ k\ \mathit{blf} = \mathit{lookup}\ k\ c \\
\qquad \mid \mathit{otherwise} \qquad\qquad\quad = \mathit{return}\ \mathit{Nothing}
\end{array}
$$

ORACLE

Add-ons

- Have kind $(* \to * \to *) \to * \to * \to *$

- *add-ons* $\approx$ *monad transformer*

- Alter the implementation under same API

```
instance (IsMap c) ⇒ IsMap (BloomOf h m c) where
  lookup k (BloomOf blf c)
      | bloomMember k blf = lookup k c
      | otherwise         = return Nothing
```

Add-ons

- Have kind $(* \to * \to *) \to * \to * \to *$

- *add-ons* $\approx$ *monad transformer*

- Alter the implementation under same API

```
instance (IsMap c) ⇒ IsMap (BloomOf h m c) where
  lookup k (BloomOf blf c)
     | bloomMember k blf = lookup k c
     | otherwise         = return Nothing
```

Meet the Add-ons

BloomOf h m Adds a bloom-filter with *h* hash functions and *m* machine words.

PartialOf Allows for the argument map to be absent.

PagesOf l Replicates the argument map on a tree structure following *l*.

CacheOf c p Adds a cache *c* with eviction policy *p*.

BoundedCacheOf b c p Forces the size of the cache to never exceed *b*.

Meet the Add-ons

BloomOf h m Adds a bloom-filter with *h* hash functions and *m* machine words.

PartialOf Allows for the argument map to be absent.

PagesOf l Replicates the argument map on a tree structure following *l*.

CacheOf c p Adds a cache *c* with eviction policy *p*.

BoundedCacheOf b c p Forces the size of the cache to never exceed *b*.

Meet the Add-ons

BloomOf h m Adds a bloom-filter with *h* hash functions and *m* machine words.

PartialOf Allows for the argument map to be absent.

PagesOf l Replicates the argument map on a tree structure following *l*.

CacheOf c p Adds a cache *c* with eviction policy *p*.

BoundedCacheOf b c p Forces the size of the cache to never exceed *b*.

Meet the Add-ons

BloomOf h m Adds a bloom-filter with *h* hash functions and *m* machine words.

PartialOf Allows for the argument map to be absent.

PagesOf l Replicates the argument map on a tree structure following *l*.

CacheOf c p Adds a cache *c* with eviction policy *p*.

BoundedCacheOf b c p Forces the size of the cache to never exceed *b*.

Meet the Add-ons

BloomOf h m Adds a bloom-filter with *h* hash functions and *m* machine words.

PartialOf Allows for the argument map to be absent.

PagesOf l Replicates the argument map on a tree structure following *l*.

CacheOf c p Adds a cache *c* with eviction policy *p*.

BoundedCacheOf b c p Forces the size of the cache to never exceed *b*.

Properties

- Type-classes provide access to properties satsified by certain combinations of add-ons.

- Similar to how `mtl` implements *MonadReader*, *MonadError*, etc.

Properties

- Type-classes provide access to properties satsified by certain combinations of add-ons.

- Similar to how `mtl` implements *MonadReader*, *MonadError*, etc.

Properties: Partitioned

Combination of add-ons containing *PagesOf*: supports notion of “page” or “partition”.

```
class (IsMap c) ⇒ Partitioned c where
  type Partition c :: ∗ → ∗ → ∗
  getPartition :: (IsMapCnstr m c k v)
               ⇒ Int
               → c k v
               → ErrM m c (Maybe (Partition c))
```

Properties: Partitioned

Combination of add-ons containing *PagesOf*: supports notion of “page” or “partition”.

```
class (IsMap c) ⇒ Partitioned c where
  type Partition c :: ∗ → ∗ → ∗
  getPartition :: (IsMapCnstr m c k v)
                ⇒ Int
                → c k v
                → ErrM m c (Maybe (Partition c))
```

Properties: Cached

Combination of add-ons containing *CacheOf*: supports a lookup that alters the structure of the map, maintaining the eviction policy.

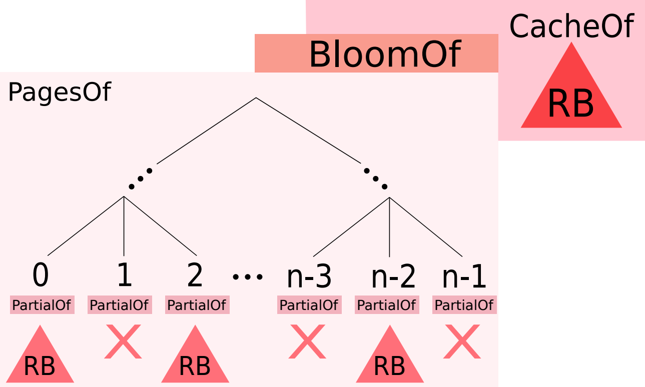
```
class (IsMap c) ⇒ Cached c where
  lookup :: (IsMapCnstr m c k v)
           ⇒ k
           → c k v
           → ErrM m c (Maybe (v, c k v))
```

Properties: Cached

Combination of add-ons containing *CacheOf*: supports a lookup that alters the structure of the map, maintaining the eviction policy.

```
class (IsMap c) ⇒ Cached c where
  lookup :: (IsMapCnstr m c k v)
         ⇒ k
         → c k v
         → ErrM m c (Maybe (v, c k v))
```

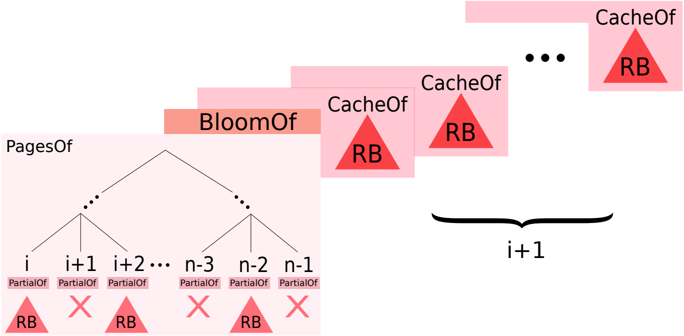
Examples
PWB_1BC



A *authenticated* tree, with possibly absent individual pages, and a cache acting as a *summary*.

Examples

PWB_Cascache



A fixed-point-like construction of just the *summary* of the previous state. Different eviction policies might show interesting differences.

The Authenticated Interface

- `hamm` supports proofs-of-membership

```
class (IsMap c) ⇒ IsAuthMap c where
   type Ev c :: ∗

   vlookup :: (IsMapCnstr m c k v)
           ⇒ k → c k v → ExceptT (Err c) m
                                  (Maybe (v, Ev c))

   rebuild  :: (IsMapCnstr m c k v)
           ⇒ Proxy c → Ev c → k → v → Digest
```

Upon seing a successufl *vlookup*, we can *rebuild* a digest and check it matches the merkle root of the map.

The Authenticated Interface

- `hamm` supports proofs-of-membership

```
class (IsMap c) ⇒ IsAuthMap c where
  type Ev c :: ∗
  vlookup :: (IsMapCnstr m c k v)
          ⇒ k → c k v → ExceptT (Err c) m
                                (Maybe (v, Ev c))
  rebuild :: (IsMapCnstr m c k v)
          ⇒ Proxy c → Ev c → k → v → Digest
```

Upon seing a successufl *vlookup*, we can *rebuild* a digest and check it matches the merkle root of the map.

ORACLE

The Authenticated Interface

- `hamm` supports proofs-of-membership

```
class (IsMap c) ⇒ IsAuthMap c where
  type Ev c :: ∗
  vlookup :: (IsMapCnstr m c k v)
          ⇒ k → c k v → ExceptT (Err c) m
                                 (Maybe (v, Ev c))
  rebuild :: (IsMapCnstr m c k v)
          ⇒ Proxy c → Ev c → k → v → Digest
```

Upon seing a successufl *vlookup*, we can *rebuild* a digest and check it matches the merkle root of the map.

The Authenticated Interface

Example Instance

```
data PartialOf c k v = Missing Digest
                     | Present   (c k v)

instance IsAuthMap c ⇒ IsAuthMap (PartialOf c) where
  type Ev (PartialOf c) = Ev c

  vlookup k (Missing _) = throwError ErrOnMissing
  vlookup k (Present c) = withExceptT ErrOnPresent
    $ vlookup k c

  rebuild _ = rebuild (Proxy :: Proxy c)
```

Experiment: Routine

- Insert 200000 keys in a map.

- Simulate transfering the “summary”.

- Perform *n createOrUpdate* operations, serving and counting page misses as they arise.
  - Draw keys according to uniform and geometric distributions (99.99%)

- We assume 14ms latency and 16Mbps transfer speed.

Experiment: Routine

- Insert 200000 keys in a map.

- Simulate transfering the “summary”.

- Perform *n createOrUpdate* operations, serving and counting page misses as they arise.
    - Draw keys according to uniform and geometric distributions (99.99%)

- We assume 14ms latency and 16Mbps transfer speed.

ORACLE

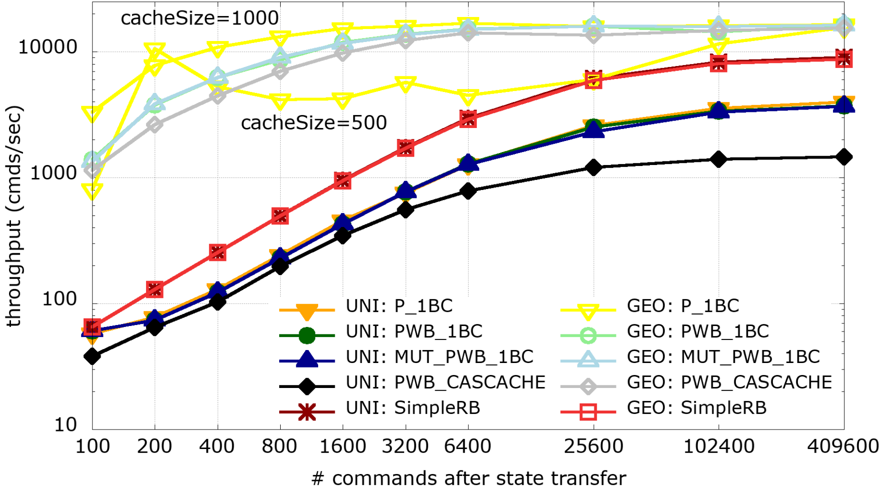
Experiment: Routine

- Insert 200000 keys in a map.
- Simulate transfering the “summary”.
- Perform *n createOrUpdate* operations, serving and counting page misses as they arise.
  - Draw keys according to uniform and geometric distributions (99.99%)
- We assume 14ms latency and 16Mbps transfer speed.

Experiment: Routine

- Insert 200000 keys in a map.
- Simulate transfering the “summary”.
- Perform *n createOrUpdate* operations, serving and counting page misses as they arise.
    - Draw keys according to uniform and geometric distributions (99.99%)
- We assume 14ms latency and 16Mbps transfer speed.

Experiment: Results



Discussion
Bloom Filter

- Seemed like a great idea! Prevent unecessary page fetches.
- Turns out its not so great:
  - Either too big to be transfered efficiently
  - Or fills up quite fast and produces false positives.
- `hamm` allowed us to easily identify that! Removing the bloom filter is as simple as changing one type.
- Even with mutable state, the main performance gain was on insertions only.

Discussion

Bloom Filter

- Seemed like a great idea! Prevent unecessary page fetches.
- Turns out its not so great:
  - Either too big to be transfered efficiently
  - Or fills up quite fast and produces false positives.
- `hamm` allowed us to easily identify that! Removing the bloom filter is as simple as changing one type.
- Even with mutable state, the main performance gain was on insertions only.

Discussion

Bloom Filter

- Seemed like a great idea! Prevent unecessary page fetches.
- Turns out its not so great:
  - Either too big to be transfered efficiently
  - Or fills up quite fast and produces false positives.
- `hamm` allowed us to easily identify that! Removing the bloom filter is as simple as changing one type.
- Even with mutable state, the main performance gain was on insertions only.

Discussion
Bloom Filter

- Seemed like a great idea! Prevent unecessary page fetches.
- Turns out its not so great:
  - Either too big to be transfered efficiently
  - Or fills up quite fast and produces false positives.
- `hamm` allowed us to easily identify that! Removing the bloom filter is as simple as changing one type.
- Even with mutable state, the main performance gain was on insertions only.

Discussion
Bloom Filter

- Seemed like a great idea! Prevent unecessary page fetches.
- Turns out its not so great:
  - Either too big to be transfered efficiently
  - Or fills up quite fast and produces false positives.
- `hamm` allowed us to easily identify that! Removing the bloom filter is as simple as changing one type.
- Even with mutable state, the main performance gain was on insertions only.

Conclusions and Future Work

- The flexibility provided by `hamm` is invaluable for our research on blockchain consensus.
- We are on our way to open-sourcing `hamm`.
- There are still a bunch of add-ons we'd like to look into such as (true) hash tables.
- There are many optimization opportunities in the library.

Conclusions and Future Work

- The flexibility provided by `hamm` is invaluable for our research on blockchain consensus.
- We are on our way to open-sourcing `hamm`.
- There are still a bunch of add-ons we'd like to look into such as (true) hash tables.
- There are many optimization opportunities in the library.

Conclusions and Future Work

- The flexibility provided by `hamm` is invaluable for our research on blockchain consensus.

- We are on our way to open-sourcing `hamm`.

- There are still a bunch of add-ons we'd like to look into such as (true) hash tables.

- There are many optimization opportunities in the library.

Conclusions and Future Work

- The flexibility provided by `hamm` is invaluable for our research on blockchain consensus.

- We are on our way to open-sourcing `hamm`.

- There are still a bunch of add-ons we'd like to look into such as (true) hash tables.

- There are many optimization opportunities in the library.

Design and Implementation of HAMM

Haskell Authenticated Modular Maps

Victor Miraldo, Harold Carr, Alex Kogan, Mark Moir, Maurice Herlihy
Oracle Labs
September 21, 2018

ORACLE®