Merkle Search Trees: Efficient State-based CRDTs in Open Networks

Alex Auvolat, François Taïani

Team WIDE
Univ. Rennes, Inria, CNRS, IRISA
alex.auvolat@inria.fr

October 3, 2019
SRDS 2019, Lyon, France
We consider the problem of **efficient state reconciliation**

- Applicable to CRDTs that implement **eventual consistency**
Example
Example
Example
Our Setting

- We study the simple case of storing and broadcasting events (a distributed event store)
- It is a simple CRDT: a grow-only set
- The principles of this paper can be applied to other CRDTs
We consider a large network with many nodes.

Nodes may join, leave and re-join at any time.

Typically the case of large-scale geo-replicated systems.
Nodes communicate with random other nodes using a gossip protocol.
Gossip in Very Large Networks
Gossip in Very Large Networks
Nodes communicate with **random other nodes** using a **gossip protocol**

Nodes do not keep track of other nodes’ states (too expensive)
The Problem: How to Minimize Cost of Exchange?

Nodes must be able to exchange missing events with zero a-priori knowledge on each other’s state, with **minimal data exchange**.

This is known as **anti-entropy**.
Approach 1: vector clocks

- Known as Scuttlebutt anti-entropy [1]

- The clock grows linearly with the number of nodes
  → inefficient in large networks
  → metadata for disconnected nodes cannot be discarded
  → fundamentally uns suited to networks with churn
Approach 2: **Merkle trees**

- Encode the set of events in a **Merkle tree**

- Enables efficient comparison between two remote sites: equal subtrees have equal hashes and can be skipped entirely

- Only **paths to changed nodes** need to be exchanged
What Is a Merkle Tree Anyways?

root hash = hash of this block

new or modified hashes

root hash
What Is a Merkle Tree Anyways?

root hash

of this block

new or modified hashes

root hash

= hash
What Is a Merkle Tree Anyways?

- root hash
- hash of this block
- new or modified hashes

Diagram:

- Node labeled "root hash"
- Branching structure with hashed values
- Left branch with a purple diamond
- Right branch with a green circle
What Is a Merkle Tree Anyways?

Root hash = hash of this block

New or modified hashes

Diagram of a Merkle tree with nodes and hashes.
Merkle Tree Remote Comparison

root hash

root hash

Alex Auvolat
Merkle Search Trees October 3, 2019 @ SRDS, Lyon, France
Merkle Tree Remote Comparison

![Diagram of Merkle Tree Remote Comparison](image_url)
Kinds of Merkle Trees: State of the Art

- **Standard Merkle trees:** balanced binary tree
  keys = integers from 0 to $n$
  → too restrictive for our use case

- **Merkle hash prefix trees:**
  Keys can be any values in any space with a hash function
  Algorithm: hash the keys and build a prefix tree
  → uses randomization to generate a balanced tree
Our Contribution: Merkle Search Trees

- **Issue:** the new events end up in *different branches*, lots of data needs to be exchanged for intermediate nodes.
Our Contribution: Merkle Search Trees

Randomized order
Merkle hash prefix tree

new events
Our Contribution: Merkle Search Trees

- **Issue**: the new events end up in *different branches*, lots of data needs to be exchanged for intermediate nodes

- **Our contribution**: a tree structure that *preserves order*

- We order events by their creation timestamp
  → new events end up close together
Our Contribution: Merkle Search Trees

SotA:
Randomized order

Our Solution:
Preserved order
Our Contribution: Merkle Search Trees

Use randomization to generate a **balanced tree structure**, but preserve the **order of items**

$h_B = \text{hash written in base } B$

X = any digit except 0

$x_1 < x_2 < x_3 < x_4 < x_5 < x_6 < x_7 < x_8 < x_9 < x_{10} < x_{11}$
### Theoretical comparison

<table>
<thead>
<tr>
<th>Anti-entropy algorithm</th>
<th>Dissemination time</th>
<th>Traffic per anti-entropy round</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scuttlebutt (vector clocks)</td>
<td>$2\lambda \log m$</td>
<td>$O(p + d)$</td>
</tr>
<tr>
<td>Merkle Search Trees</td>
<td>$2\lambda \log m \log_B n$</td>
<td>$O(d \log_B n)$</td>
</tr>
</tbody>
</table>

$p$ number of nodes, past and present  
$m$ number of nodes currently connected  
$n$ number of past events  
$d$ number of new events in anti-entropy round  
$\lambda$ network latency
Experimental Evaluation

- We evaluate Merkle Search Trees against:
  - SB: Scuttlebutt (vector clocks)
  - MPT: Merkle hash prefix trees (do not preserve order)

- By adjusting the interval between gossip events, we can adapt for faster delivery or lower bandwidth use
We test our experiment in a simulation:

- **1000 nodes** that generate events at random
- First experiment: 1 event every 10 simulation rounds (in the whole network)
Experimental Results: Low Event Rate

### Bandwidth usage (bytes per round)

- **Scuttlebutt**
- **MPT**
- **MST (ours)**

### Entropy of round (average)

- **Scuttlebutt**
- **MPT**
- **MST (ours)**

<table>
<thead>
<tr>
<th>Method</th>
<th>Bandwidth use(^a)</th>
<th>Entropy(^b)</th>
<th>99% delivery delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scuttlebutt</td>
<td>1.3 Mo</td>
<td>1.61</td>
<td>64 rounds</td>
</tr>
<tr>
<td>MPT</td>
<td>0.51 Mo</td>
<td>1.44</td>
<td>56 rounds</td>
</tr>
<tr>
<td>MST (ours)</td>
<td>0.44 Mo</td>
<td>1.06</td>
<td>44 rounds</td>
</tr>
</tbody>
</table>

- **Gain vs. SB**: -66%
- **Gain vs. MPT**: -13%

\(^a\) per round, on average  
\(^b\) at each round, on average
Experimental Results: High Event Rate

Second experiment: **10 times higher event rate**

1000 nodes

<table>
<thead>
<tr>
<th>Method</th>
<th>Bandwidth use</th>
<th>Entropy</th>
<th>99% delivery delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scuttlebutt</td>
<td>2.1 Mo</td>
<td>15.4</td>
<td>50 rounds</td>
</tr>
<tr>
<td>MST (ours)</td>
<td>2.2 Mo</td>
<td>17.5</td>
<td>74 rounds</td>
</tr>
</tbody>
</table>

2000 nodes

<table>
<thead>
<tr>
<th>Method</th>
<th>Bandwidth use</th>
<th>Entropy</th>
<th>99% delivery delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scuttlebutt</td>
<td>7.6 Mo</td>
<td>14.9</td>
<td>54 rounds</td>
</tr>
<tr>
<td>MST (ours)</td>
<td>4.2 Mo</td>
<td>21.0</td>
<td>88 rounds</td>
</tr>
</tbody>
</table>
Merkle Search Trees are competitive for low event rates

Merkle Search Trees scale better when many nodes participate
Merkle Search Trees are an efficient way of implementing CRDTs in open networks.

Merkle Search Trees could have many other applications (examples: distributed databases).

See our paper for more details:
Alex Auvolat and François Taïani, *Merkle Search Trees: Efficient State-based CRDTs in Open Networks*, SRDS 2019