AIP-006: State Version Control

lia@, seo@ 2020-06-08

Goals

- Introduce an efficient state version control for block versions of Streamlet Consensus algorithm

Problem Definition

- With Streamlet consensus protocol, a blockchain can have multiple unfinalized chains, or forks, at a given time (see Streamlet: Textbook Streamlined Blockchains)
- Each fork as well as each block within a fork have different intermediate states, modified by different sets of transactions
- A block may theoretically be appended to any of the unfinalized blocks, and blockchain nodes have to verify the transactions in the block by executing them on the snapshot of the database state that reflects the chain the new block is appended to

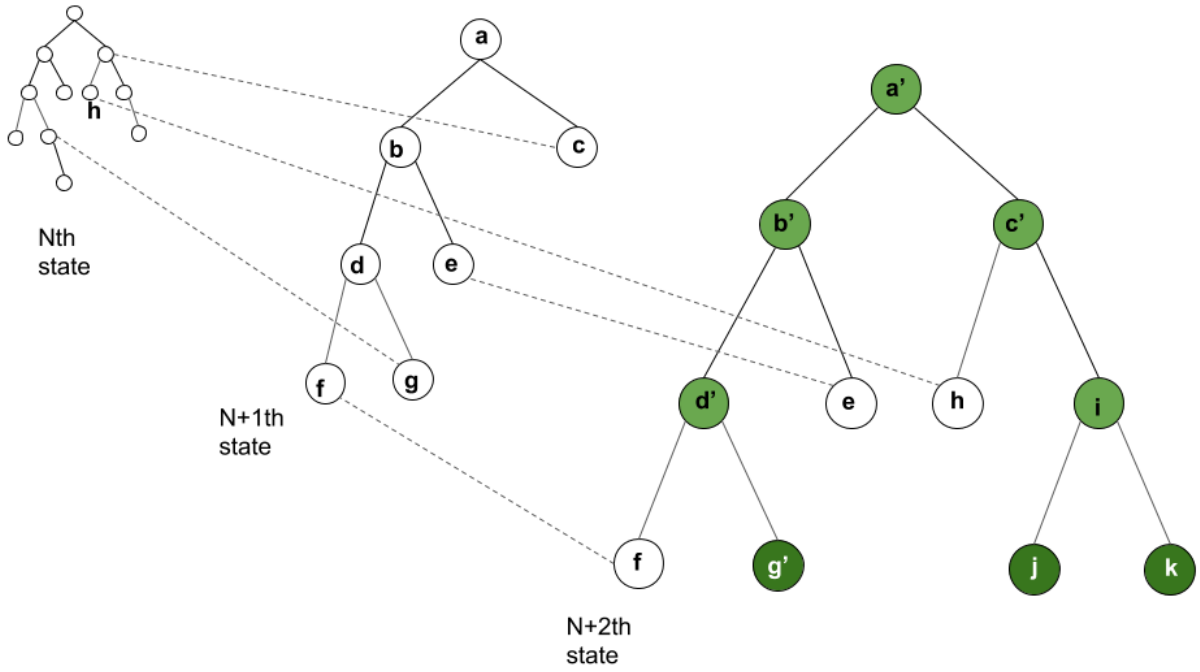
Requirements

- Handle state version branching and merging for multi-version block receive and block finalization, respectively
- Support version control operations (create, delete, finalize) in O(d) time where d is the size of the delta between the state versions

Proposed Design

State Referencing

- To minimize data copying, use references to previous blocks' states if that particular part of the state tree wasn't changed by the new block
- Will always hold a full copy of the latest finalized state and there may be several unfinalized states with partial copies and references



Node Abstraction

- In JavaScript, primitive values are always copied instead of being referenced.
- Since on AI Network blockchain, we accept primitive types such as strings, numbers, etc., we need node abstraction to pass them as references.

Node Class

Fields

Name	Type	Description
childMap	HashMap<string, Node>	Mapping between child nodes' labels and child nodes
versions	Set<number> | List<number>	Set of versions (epochs of the blocks) that are referencing the node. Implementation could differ depending on the programming language, but a set or a list would suffice.
proofHash	string	Merkle hash of the node. See AIP-007 for more information
key	string	Key of the node. Appending the keys of the parent

		nodes will give the path to the node
value	undefined | any	Value at the node. Should always be undefined if the node is not a leaf, and should never be undefined if the node is a leaf

Functions

Name	Arguments	Return	Description
getChild()	string	Node | null	Returns a child node with the input key
getChildLabels()	-	string[]	Returns an array of child node's labels
getChildNodes()	-	Node[]	Returns an array of child nodes
getNumChild()	-	number	Returns the number of child nodes
setChild()	label, Node	boolean	Set a child Node
hasChild()	label	boolean	Returns whether the node has a child node with the given label
deleteChild()	string	boolean	Deletes a child node with the given label
getVersions()	-	Set<number> | List<number>	Returns all versions
addVersion()	number	boolean	Adds a version to the versions
hasVersion()	number	boolean	Returns if a version is in the versions
deleteVersion()	number	boolean	Deletes a version from the versions
getProofHash()	-	string	Returns proofHash
setProofHash()	string	boolean	Sets proofHash
getValue()	-	undefined | any	Returns value
setValue()	any	boolean	Sets value

Example

Database State Tree (**Block #10 version**)

```
/a: {
  b: {
    c: {
```

```
      d: val_1,
      e: val_2
    }
  },
  f: {
    g: {
      h: {
        i: val_3
      }
    },
    j: {
      k: {
        l: val_4
       }
      },
      m: {
        n: val_5
      }
    }
  }
}
```

Database State Tree (**Block #11 version**)

- Changes
    - /a/b/c/d: val_6
    - /a/f/j/k/l: val_7

```
/a: {
  b: {
    c: {
      d: val_6,
      e: Ref<#10:/a/b/c/e>
    }
  },
  f: {
    g: Ref<#10:/a/f/g>,
    j: {
      k: {
        l: val_7
      },
      m: Ref<#10:/a/f/j/m>
    }
  }
}
```

State Versioning Operations

Key Idea: we only need to update versions at the earliest point where the state diverges from the previous state

Creating state branches

Function

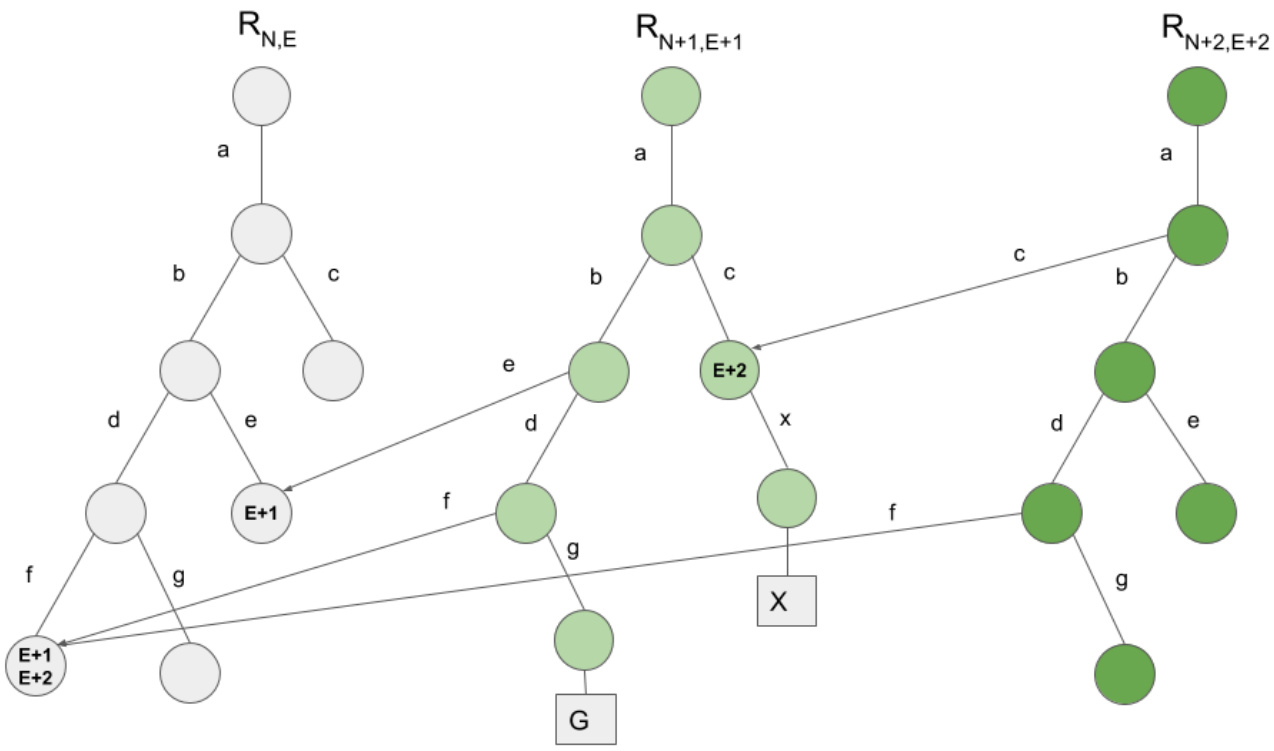
Name	Arguments	Return	Description
createStateBranch()	Node, Block	Node	Returns the root to a new state branch

Example

- Block of number N+1 and epoch E+1 (B<N+1, E+1>)
    - appends B<N, E>
    - contains following transactions:
        - SET_VALUE at ref “/a/c” => ref: /a/c, value: { x: X }
        - SET_VALUE at ref “/a/b/d/g” => ref: /a/b/d/g, value: G
- Execution flow
    1. Create a new Node for the root R<N+1, E+1>
    2. For the first transaction (setting value { x: X } at ref “/a/c”)
        a. Since there’s no child node with a label “a”, create a new node N’<a> by making a copy of N<a> in tree R<N, E>, and add it as a child to the R<N+1, E+1>
        b. Create another node N’<c>, which is a copy of N<c>, to add as a child of N’<a>, and set its value as { x: X } (represented with another node)
        c. Sees N<b>, a sibling of N<c>, and
            i. Add a reference to N<b> as a child of N’<a>
            ii. Add a version “E+1” to N<b>
    3. For the second transaction (setting value G at ref “/a/b/d/g”)
        a. Create a new node N’<b>, a copy of N<b>, and
            i. Replace it with the N’<a>’s current child, which is a reference to N<b>
            ii. Remove version “E+1” from N<b>
        b. Create a new node N’<d>, a copy of N<d>, and add it as a child to N’<b>
        c. Create a new node N’<g>, a copy of N<g>, and
            i. Set its value as G
            ii. Add it as a child to the N’<d>
        d. Sees N<f>, a sibling of N<g>, and
            i. Add a reference to N<f> as a child of N’<d>

ii. Add a version “E+1” to N<f>

e. Sees N<e>, a sibling of N<d>, and
   i. Add a reference to N<e> as a child of N<’b>
   ii. Add a version “E+1” to N<e>



Analysis

The time / space complexity of createStateBranch() is O(d) where d is the size of the delta between the two state trees.

Deleting state branches

We assume that when we’re deleting a state branch for block {N, E}, states for blocks of {N’, E’} where N’ < N and E’ < E must have already been deleted.

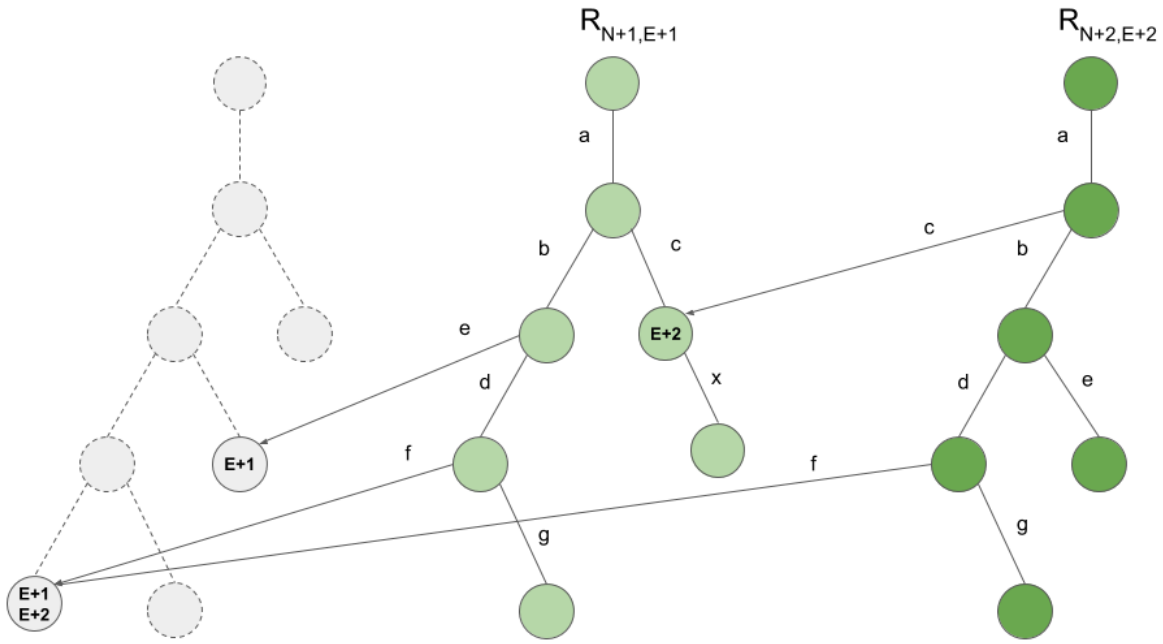
1. Traverse the state tree in a depth-first manner
2. If we see a node with non-empty version where there are versions other than the version to delete
   a. Stop going down the path
   b. Delete the already traversed nodes (nodes without a version or nodes with only the version to delete)

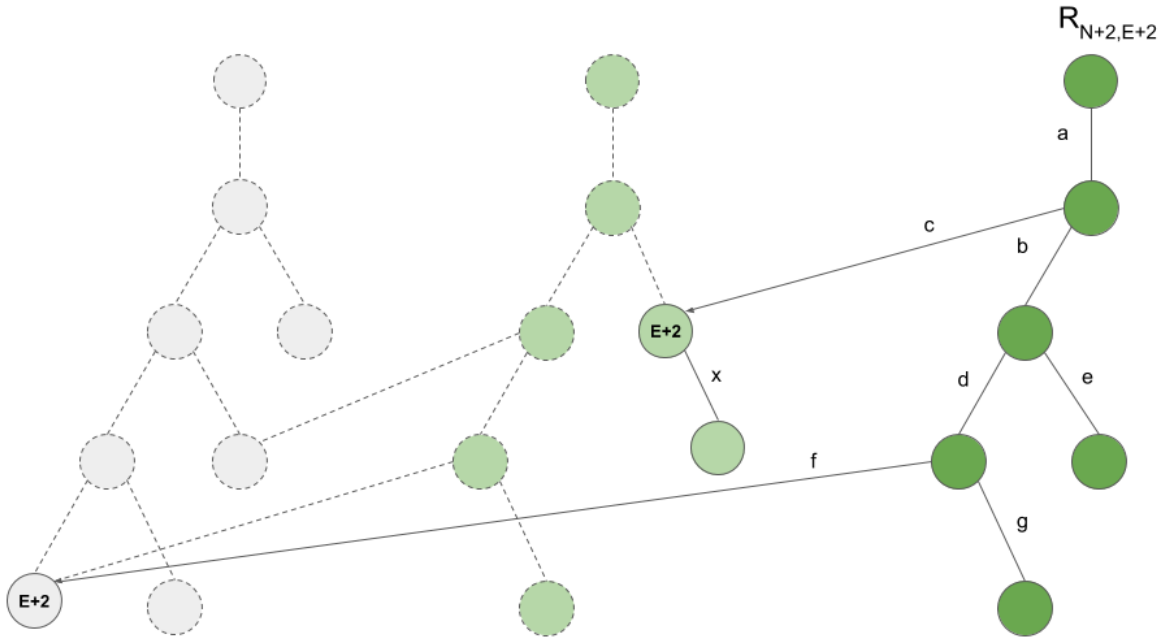
Function

Name	Arguments	Return	Description
deleteStateBranch()	Node, number	boolean	Removes the state branch corresponding to the given stateRoot (Node) and epoch (number)

Example

- Block B<n+1, e+1> becomes finalized and state tree of B<n, e> is no longer needed
- Execution flow
    1. Starting from R<n, e>, traverse to N<a>
    2. Since N<a> has two child nodes, go down to N<b>
    3. Since N<b> has two child nodes, go down to N<d>
    4. Since N<d> has two child nodes, go down to N<f>
    5. Notice that N<f> is a leaf node and has versions greater than e (> e)
        a. Do nothing
    6. Move on to N<g>, which is a leaf node without versions > e
    7. Delete N<g> from memory
    8. Move back up to N<d>, which doesn't have versions > e, and delete N<d>
    9. N<e>, a sibling of N<d>, has a version "e+1", so keep it
    10. Move up to N<b> and delete it
    11. Delete N<c>
    12. Delete R<n, e>





Analysis

The time complexity of deleteStateBrach() is O(d) where d is the size of the delta between the two state trees.

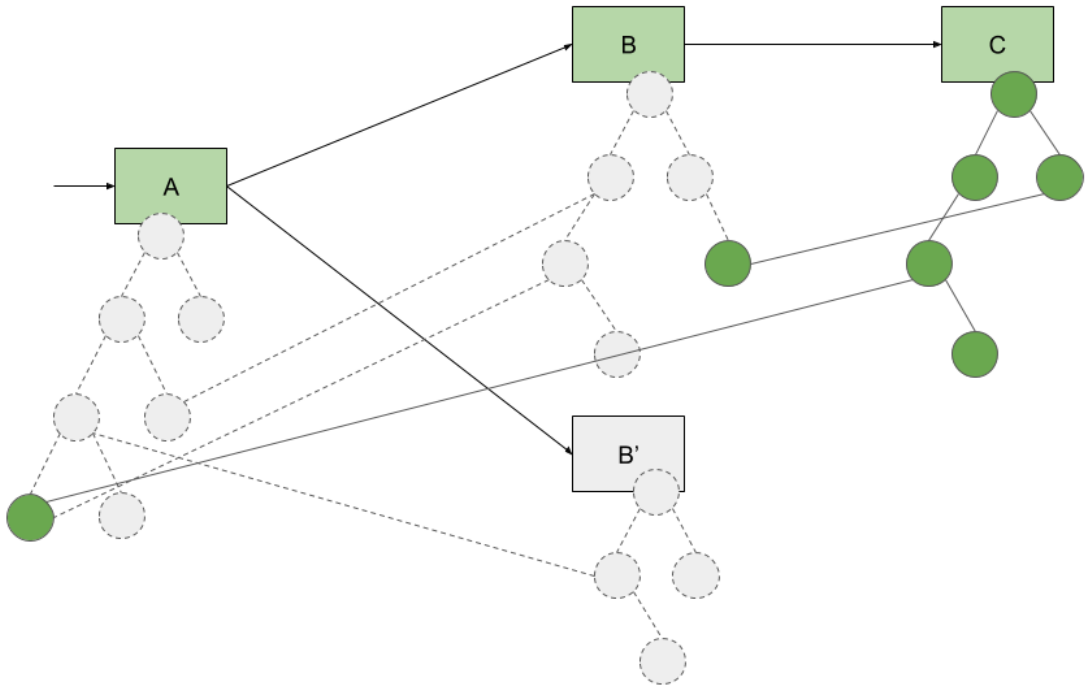
Finalizing state branches

Function

Name	Arguments	Return	Description
finalizeStateBranch()	number	boolean	Removes unfinalized state branches that are no longer relevant (e.g. states of another fork and intermediary states between the last finalized state and the new finalized state)

Example

- Two unfinalized chains A-B-C and A-B' exist and the A-B-C branch becomes finalized
- Execution flow
    1. From the earliest block (block A), perform state deletion
    2. Perform state deletion of blocks B, C that append block A
    3. Last finalized state, which is the state modified by block C, remains



Issues

- Deletion (SET_VALUE operation with the value of null)

Links

- Streamlet: Textbook Streamlined Blockchains (pdf)
- AIP-007 Provable Blockchain States (link)

Document History

Date	Who	Change	Notes
2020-06-24	lia@	Initial draft	
2020-06-30	lia@, seo@	Internal review	
2021-05-07	seo@	Published	