Encoded Vector Clock: Using Primes to Characterize Causality in Distributed Systems

Ajay D. Kshemkalyani  Ashfaq Khokhar  Min Shen

University of Illinois at Chicago

ajay@uic.edu
Overview

1. Introduction

2. Encoded Vector Clock (EVC)
   - Operations on the EVC
   - Complexity of EVC

3. Operations on Cuts Using EVC
   - Timestamping a Cut
   - Common Past of Events on a Cut
   - Union and Intersection
   - Comparison of Cuts

4. Scalability of EVCs

5. Discussion and Conclusions
Introduction

- Scalar clocks: $e \rightarrow f \Rightarrow C(e) < C(f)$
- Vector clocks: $e \rightarrow f \iff V(e) < V(f)$
  - Fundamental tool to characterize causality
  - To capture the partial order $(E, \rightarrow)$, size of vector clock is the dimension of the partial order, bounded by the size of the system, $n$
  - Not scalable!

Contribution

propose encoding of vector clocks using prime numbers to use a single number to represent vector time
Vector Clock Operation at a Process $P_i$

1. Initialize $V$ to the 0-vector.
2. Before an internal event happens at process $P_i$, $V[i] = V[i] + 1$ (local tick).
3. Before process $P_i$ sends a message, it first executes $V[i] = V[i] + 1$ (local tick), then it sends the message piggybacked with $V$.
4. When process $P_i$ receives a message piggybacked with timestamp $U$, it executes
   \[ \forall k \in [1 \ldots n], V[k] = \max(V[k], U[k]) \] (merge);
   \[ V[i] = V[i] + 1 \] (local tick)
   before delivering the message.
A vector clock $V = \langle v_1, v_2, \cdots, v_n \rangle$ can be encoded by $n$ distinct prime numbers, $p_1, p_2, \cdots, p_n$ as:

$$Enc(V) = p_1^{v_1} \times p_2^{v_2} \times \cdots \times p_n^{v_n}$$

EVC operations: Tick, Merge, Compare

**Tick** at $P_i$: $Enc(V) = Enc(V) \times p_i$
EVC Operations (contd.)

- **Merge:** For \( V_1 = \langle v_1, v_2, \cdots, v_n \rangle \) and \( V_2 = \langle v'_1, v'_2, \cdots, v'_n \rangle \), merging yields:

\[
U = \langle u_1, u_2, \cdots, u_n \rangle, \text{ where } u_i = \max(v_i, v'_i)
\]

The encodings of \( V_1 \), \( V_2 \), and \( U \) are:

\[
\begin{align*}
Enc(V_1) &= p_1^{v_1} \ast p_2^{v_2} \ast \cdots \ast p_n^{v_n} \\
Enc(V_2) &= p_1^{v'_1} \ast p_2^{v'_2} \ast \cdots \ast p_n^{v'_n} \\
Enc(U) &= \prod_{i=1}^{n} p_i^{\max(v_i,v'_i)}
\end{align*}
\]

However, we show

\[
Enc(U) = LCM(Enc(V_1), Enc(V_2)) = \frac{Enc(V_1) \ast Enc(V_2)}{GCD(Enc(V_1), Enc(V_2))}
\]
EVC Operations (contd.)

**Compare:**

1. \( \text{Enc}(V_1) \prec \text{Enc}(V_2) \) if \( \text{Enc}(V_1) < \text{Enc}(V_2) \) and \( \text{Enc}(V_2) \mod \text{Enc}(V_1) = 0 \)

2. \( \text{Enc}(V_1) \parallel \text{Enc}(V_2) \) if \( \text{Enc}(V_1) \not\prec \text{Enc}(V_2) \) and \( \text{Enc}(V_2) \not\prec \text{Enc}(V_1) \)

Thus, to manipulate the EVC,

- Each process needs to know only its own prime
- Merging EVCs requires computing LCM
  - Use Euclid’s algorithm for GCD, which does not require factorization
### Table: Correspondence between vector clocks and EVC.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Vector Clock</th>
<th>Encoded Vector Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Representing clock</td>
<td>$V = \langle v_1, v_2, \ldots, v_n \rangle$</td>
<td>$Enc(V) = p_1^{v_1} \ast p_2^{v_2} \ast \cdots \ast p_n^{v_n}$</td>
</tr>
<tr>
<td>Local Tick (at process $P_i$)</td>
<td>$V[i] = V[i] + 1$</td>
<td>$Enc(V) = Enc(V) \ast p_i$</td>
</tr>
<tr>
<td>Merge</td>
<td>Merge $V_1$ and $V_2$ yields $V$ where $V[j] = \max(V_1[j], V_2[j])$</td>
<td>Merge $Enc(V_1)$ and $Enc(V_2)$ yields $Enc(V) = \text{LCM}(Enc(V_1), Enc(V_2))$</td>
</tr>
<tr>
<td>Compare</td>
<td>$V_1 &lt; V_2$: $\forall j \in [1, n], V_1[j] \leq V_2[j]$, and $\exists j, V_1[j] &lt; V_2[j]$</td>
<td>$Enc(V_1) &lt; Enc(V_2)$: $Enc(V_1) &lt; Enc(V_2)$, and $Enc(V_2) \mod Enc(V_1) = 0$</td>
</tr>
</tbody>
</table>
**Operation of the Encoded Vector Clock**

1. Initialize $t_i = 1$.
2. Before an internal event happens at process $P_i$, $t_i = t_i \times p_i$ (local tick).
3. Before process $P_i$ sends a message, it first executes $t_i = t_i \times p_i$ (local tick), then it sends the message piggybacked with $t_i$.
4. When process $P_i$ receives a message piggybacked with timestamp $s$, it executes
   
   $t_i = \text{LCM}(s, t_i)$ (merge);
   $t_i = t_i \times p_i$ (local tick)

   before delivering the message.

**Figure**: Operation of EVC $t_i$ at process $P_i$. 

Kshemkalyani, Khokhar, Shen (UIC)  
Encoded Vector Clock

9 / 24
Illustration of Using EVC

Figure: The vector timestamps and EVC timestamps are shown above and below each timeline, respectively. In real scenarios, only the EVC is stored and transmitted.
Complexity of Vector Clock and EVC

- $h$: number of bits or digits in EVC value $H$
- $n$: number of processes in the system

**Table:** Comparison of the time complexity of the three basic operations and the space complexity, for vector clock and EVC.

<table>
<thead>
<tr>
<th></th>
<th>Vector Clock (bounded storage) (uniform cost model)</th>
<th>Encoded Vector Clock (unbounded storage) (logarithmic cost model)</th>
<th>Encoded Vector Clock (bounded storage) (uniform cost model)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Local Tick</strong></td>
<td>$O(1)$</td>
<td>$O(h)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>Merge</strong></td>
<td>$O(n)$</td>
<td>$O(h \log^2 h (\log \log h))$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>Compare</strong></td>
<td>$O(n)$</td>
<td>$O(h \log h (\log \log h))$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>Storage</strong></td>
<td>$O(n)$</td>
<td>$O(h)$</td>
<td>$O(1) + O(d)$ (with resetting)</td>
</tr>
</tbody>
</table>
EVC Timestamps of Cuts

- **Cut**: is an execution prefix
- **State after the events of a cut** represents a *global state*
- \[ \downarrow e = \{ f \mid f \rightarrow e \land f \in E \} \cup \{ e \} \] (causal history of \( e \))
- \( S(cut) \): set that contains the last event of \( cut \) at each process
- \( \hat{cut} \): smallest consistent cut larger than or equal to \( cut \)
EVC Timestamp of a Cut

- Timestamp of a cut, \( cut \):
  \[
  \forall k \in [1, n], V(cut)[k] = V(e_k)[k], \text{ for } e_k \in S(\widehat{cut})
  \]
  \[
  = \max_{e_i \in S(cut)} V(e_i)[k]
  \]

- For \( e_i \in S(cut) \), let \( V(e_i) = \langle v_{i1}, v_{i2}, \cdots, v_{in} \rangle \).
- For \( \widehat{e}_i \in \widehat{cut} \), let \( V(\widehat{e}_i) = \langle \widehat{v}_{i1}, \widehat{v}_{i2}, \cdots, \widehat{v}_{in} \rangle \).
- EVC of a cut, \( cut \):
  \[
  Enc(V(cut)) = \prod_{i=1}^{n} p_i^{\widehat{v}_i}
  \]
  \[
  = \prod_{i=1}^{n} p_i^{\max(v_{i1}, v_{i2}, \cdots, v_{in})}
  \]

- However, we show that
  \[
  Enc(V(cut)) = LCM(Enc(V(e_1)), Enc(V(e_2)), \cdots, Enc(V(e_n))).
  \]
Figure: The vector timestamps and EVC timestamps are shown above and below each timeline, respectively. In real scenarios, only the EVC is stored and transmitted.

For events $e_i \in S(\text{CutA})$:

- We have $\text{Enc}(V(e_1)) = 20$, $\text{Enc}(V(e_2)) = 54$, and $\text{Enc}(V(e_3)) = 5$.
- $\text{Enc}(V(\text{CutA})) = \text{LCM}(\text{Enc}(V(e_1)), \text{Enc}(V(e_2)), \text{Enc}(V(e_3))) = \text{LCM}(20, 54, 5) = 540$.
EVC Timestamp of Common Past

- Common Past $CP(cut) = \bigcap_{e_i \in S(cut)} \downarrow e_i$ is the execution prefix in the causal history of each event in $S(cut)$
- Vector timestamp of common past of $cut$:
  \[
  \forall k \in [1, n], V(CP(cut))[k] = \min_{e_i \in S(cut)} V(e_i)[k]
  \]
- For $e_i \in S(cut)$, $V(e_i) = \langle v^i_1, v^i_2, \cdots v^i_n \rangle$.
- We observe that\[
  Enc(V(CP(cut))) = \prod_{i=1}^{n} p_i^{\min(v^i_1, v^i_2, \cdots, v^i_n)}
  \]
- We show that\[
  Enc(V(CP(cut))) = GCD(Enc(V(e_1)), Enc(V(e_2)), \cdots, Enc(V(e_n))).
  \]
**Example: EVC Timestamp of Common Past**

![Diagram showing vector timestamps and EVC timestamps]

**Figure:** The vector timestamps and EVC timestamps are shown above and below each timeline, respectively. In real scenarios, only the EVC is stored and transmitted.

- For events $e_i \in S(CutB)$:
  - We have $Enc(V(e_1)) = 40$, $Enc(V(e_2)) = 3240$, and $Enc(V(e_3)) = 1350$.
  - $Enc(V(CP(CutB))) = GCD(Enc(V(e_1)), Enc(V(e_2)), Enc(V(e_3))) = GCD(40, 3240, 1350) = 10$. 
EVC Timestamp of Union and Intersection Cuts

- Let $V(\text{cut1}) = \langle v_1, v_2, \ldots, v_n \rangle$ and $V(\text{cut2}) = \langle v'_1, v'_2, \ldots, v'_n \rangle$
- We have that
  
  $V(\text{cut1} \cap \text{cut2}) = \langle u_1, u_2, \ldots, u_n \rangle$, where $u_i = \min(v_i, v'_i)$
  
  $V(\text{cut1} \cup \text{cut2}) = \langle u_1, u_2, \ldots, u_n \rangle$, where $u_i = \max(v_i, v'_i)$

- The encodings of $V(\text{cut1}), V(\text{cut2}), V(\text{cut1} \cap \text{cut2}), V(\text{cut1} \cup \text{cut2})$ are:
  
  $Enc(V(\text{cut1})) = p_1^{v_1} * p_2^{v_2} * \cdots * p_n^{v_n}$;
  
  $Enc(V(\text{cut2})) = p_1^{v'_1} * p_2^{v'_2} * \cdots * p_n^{v'_n}$

  $Enc(V(\text{cut1} \cap \text{cut2})) = \prod_{i=1}^{n} p_i^{\min(v_i, v'_i)}$

  $Enc(V(\text{cut1} \cup \text{cut2})) = \prod_{i=1}^{n} p_i^{\max(v_i, v'_i)}$

- We show that
  
  $Enc(V(\text{cut1} \cap \text{cut2})) = \text{GCD}(Enc(V(\text{cut1})), Enc(V(\text{cut2})))$

  $Enc(V(\text{cut1} \cup \text{cut2})) = \text{LCM}(Enc(V(\text{cut1})), Enc(V(\text{cut2})))$
Example: EVC Timestamp of Union and Intersection Cuts

Figure: The vector timestamps and EVC timestamps are shown above and below each timeline, respectively. In real scenarios, only the EVC is stored and transmitted.

- $\text{Enc}(V(\text{CutA})) = \text{LCM}(20, 54, 5) = 540$ and $\text{Enc}(V(\text{CutC})) = \text{LCM}(2, 54, 1350) = 1350$.
- $\text{Enc}(V(\text{CutA} \cap \text{CutC})) = \text{GCD}(\text{Enc}(V(\text{CutA})), \text{Enc}(V(\text{CutC}))) = \text{GCD}(540, 1350) = 270$.
- $\text{Enc}(V(\text{CutA} \cup \text{CutC})) = \text{LCM}(\text{Enc}(V(\text{CutA})), \text{Enc}(V(\text{CutC}))) = \text{LCM}(540, 1350) = 2700$. 
Comparison of Cuts

- Comparing \textit{cut}1 and \textit{cut}2:
  1. \textit{cut}1 \subset \textit{cut}2 (or symmetrically, \textit{cut}2 \subset \textit{cut}1),
  2. or ii) \textit{cut}1 \not\subset \textit{cut}2 and \textit{cut}2 \not\subset \textit{cut}1, i.e., \textit{cut}1 \parallel \textit{cut}2.

- We show:
  
  \begin{itemize}
  \item \textit{i) } \text{Enc}(\text{V(\textit{cut}1)}) \prec \text{Enc}(\text{V(\textit{cut}2)}) \text{ if } \text{Enc}(\text{V(\textit{cut}1)}) < \text{Enc}(\text{V(\textit{cut}2)}) \text{ and } \\
  \hspace{2cm} \text{Enc}(\text{V(\textit{cut}2)}) \text{ mod } \text{Enc}(\text{V(\textit{cut}1)}) = 0
  \item \textit{ii) } \text{Enc}(\text{V(\textit{cut}1)}) \parallel \text{Enc}(\text{V(\textit{cut}2)}) \text{ if } \text{Enc}(\text{V(\textit{cut}1)}) \not\prec \text{Enc}(\text{V(\textit{cut}2)}) \text{ and } \\
  \hspace{2cm} \text{Enc}(\text{V(\textit{cut}2)}) \not\prec \text{Enc}(\text{V(\textit{cut}1)})
  \end{itemize}
Correspondence between Operations on Cuts

**Table:** Correspondence between operations on cuts using vector clocks and EVC.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Vector Clock</th>
<th>Encoded Vector Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut</td>
<td>$\forall k \in [1, n], V(\text{cut})[k] = \max_{e_i \in S(\text{cut})} V(e_i)[k]$ (cut may not be consistent)</td>
<td>$\text{Enc}(V(\text{cut})) = \text{LCM}(\text{Enc}(V(e_1)), \ldots, \text{Enc}(V(e_n)))$, where $e_i \in S(\text{cut})$</td>
</tr>
<tr>
<td>Common past</td>
<td>$\forall k \in [1, n], V(\text{CP}(\text{cut}))[k] = \min_{e_i \in S(\text{cut})} V(e_i)[k]$ (cut is consistent)</td>
<td>$\text{Enc}(V(\text{cut})) = \text{GCD}(\text{Enc}(V(e_1)), \ldots, \text{Enc}(V(e_n)))$, where $e_i \in S(\text{cut})$</td>
</tr>
<tr>
<td>Intersection</td>
<td>If $V(\text{cut}_1)[j] = v_j$ and $V(\text{cut}_2)[j] = v_j'$, $V(\text{cut}_1 \cap \text{cut}_2)[j] = \min(v_j, v_j')$, $V(\text{cut}_1 \cup \text{cut}_2)[j] = \max(v_j, v_j')$</td>
<td>Merge $\text{Enc}(V(\text{cut}_1))$ and $\text{Enc}(V(\text{cut}_2))$ yields $\text{Enc}(V) = \text{GCD}(\text{Enc}(V(\text{cut}_1)), \text{Enc}(V(\text{cut}_2)))$</td>
</tr>
<tr>
<td>Union</td>
<td>$V(\text{cut}_1 \cup \text{cut}_2)[j] = \max(v_j, v_j')$</td>
<td>$\text{Enc}(V) = \text{LCM}(\text{Enc}(V(\text{cut}_1)), \text{Enc}(V(\text{cut}_2)))$</td>
</tr>
<tr>
<td>Compare</td>
<td>$V(\text{cut}_1) &lt; V(\text{cut}_2)$; $\forall j \in [1, n], V(\text{cut}_1)[j] \leq V(\text{cut}_2)[j]$, and $\exists j, V(\text{cut}_1)[j] &lt; V(\text{cut}_2)[j]$</td>
<td>$\text{Enc}(V(\text{cut}_1)) &lt; \text{Enc}(V(\text{cut}_2))$; $\text{Enc}(V(\text{cut}_1)) &lt; \text{Enc}(V(\text{cut}_2))$, and $\text{Enc}(V(\text{cut}_2)) \mod \text{Enc}(V(\text{cut}_1)) = 0$</td>
</tr>
</tbody>
</table>
## Complexity of Operations on Cuts

**Table:** Comparison of the time complexity of the operations on cuts using vector clocks and EVC.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Vector Clock (bounded storage) (uniform cost model)</th>
<th>Encoded Vector Clock (unbounded storage) (logarithmic cost model)</th>
<th>Encoded Vector Clock (bounded storage) (uniform cost model)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computing</td>
<td>$O(n^2)$ (<em>cut</em> may not be consistent)</td>
<td>$O(n h (\log^2 h)(\log \log h))$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>timestamp</td>
<td>$O(n)$ (<em>cut</em> is consistent)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Computing</td>
<td>$O(n^2)$</td>
<td>$O(n h (\log^2 h)(\log \log h))$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>common past</td>
<td>$O(n)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intersection</td>
<td>$O(n)$</td>
<td>$O(h (\log^2 h)(\log \log h))$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>and union</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compare</td>
<td>$O(n)$</td>
<td>$O(h (\log h)(\log \log h))$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
Scalability of EVCs

EVC timestamps grow very fast. To alleviate this problem:

1. Tick only at relevant events, e.g., when the variables alter the truth value of a predicate
   - On social platforms, e.g., Twitter and Facebook, max length of any chain of messages is usually small

2. Application requiring a vector clock is confined to a subset of processes

3. Reset the EVC at a strongly consistent (i.e., transitless) global state

4. Use logarithms to store and transmit EVCs
   - Local tick: single addition
   - Merge and Compare: Take anti-logs and then logs,
     - complexity is subsumed by that of GCD computation
     - extra space is only scratch space
Conclusions

- Proposed the encoding of vector clocks using prime numbers, to use a single number to represent vector time.
- To manipulate the EVC:
  - each process needs to know only its own prime
  - Merging EVCs can be done by finding LCM; does not require factorization!
- EVC provides savings in space over vector clocks.
- Time complexity of EVC operations performed using two models:
  - Bounded storage (uniform cost model): better than vector clocks
  - Unbounded storage (logarithmic cost model)
- EVCs grow very fast
  - Proposed several solutions to deal with this problem.
Thank You!