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

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.

Encoded Vector Clock (EVC) and Operations

- 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:

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

- EVC operations: Tick, Merge, Compare
- **Tick** at P_i: $Enc(V) = Enc(V) * p_i$

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:

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

However, we show

$$Enc(U) = LCM(Enc(V_1), Enc(V_2)) = \frac{Enc(V_1) * Enc(V_2)}{GCD(Enc(V_1), Enc(V_2))}$$

EVC Operations (contd.)

- **Compare:**

$$
\begin{aligned}
i)\ Enc(V_1) \prec Enc(V_2) \text{ if } Enc(V_1) < Enc(V_2) \text{ and}\\
Enc(V_2) \bmod Enc(V_1) = 0\\
ii)\ Enc(V_1) \| Enc(V_2) \text{ if } Enc(V_1) \not\prec Enc(V_2) \text{ and}\\
Enc(V_2) \not\prec Enc(V_1)
\end{aligned}
$$

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

Correspondence of Operations

Table: Correspondence between vector clocks and EVC.

Operation	Vector Clock	Encoded Vector Clock
Representing clock	$V = \langle v_1, v_2, \cdots, v_n \rangle$	$Enc(V) = p_1^{v1} * p_2^{v2} * \cdots * p_n^{v_n}$
Local Tick (at process P_i)	$V[i] = V[i] + 1$	$Enc(V) = Enc(V) * p_i$
Merge	Merge V_1 and V_2 yields V where $V[j] = \max(V_1[j], V_2[j])$	Merge $Enc(V_1)$ and $Enc(V_2)$ yields $Enc(V) = LCM(Enc(V_1), Enc(V_2))$
Compare	$V_1 < V_2$: $\forall j \in [1, n]$, $V_1[j] \leq V_2[j]$, and $\exists j$, $V_1[j] < V_2[j]$	$Enc(V_1) \prec Enc(V_2)$: $Enc(V_1) < Enc(V_2)$, and $Enc(V_2) \mod Enc(V_1) = 0$

Operation of the Encoded Vector Clock

1. Initialize $t_i = 1$.
2. Before an internal event happens at process P_i,
   $t_i = t_i * p_i$ (local tick).
3. Before process P_i sends a message, it first executes $t_i = t_i * p_i$ (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
   $t_i = LCM(s, t_i)$ (merge);
   $t_i = t_i * p_i$ (local tick)
   before delivering the message.

Figure: Operation of EVC t_i at process P_i.

Illustration of Using EVC

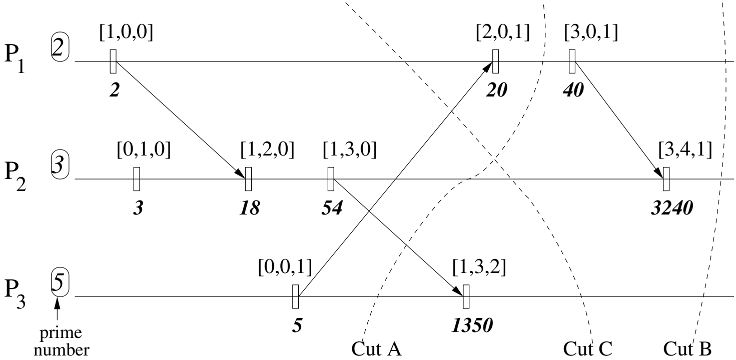

Figure: The vector timestamps and EVC timestamps are shown above and below each timeline, respectively. In real scenarios, only the EVC is stored and transmitted.

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.

	Vector Clock (bounded storage) (uniform cost model)	Encoded Vector Clock (unbounded storage) (logarithmic cost model)	Encoded Vector Clock (bounded storage) (uniform cost model)
Local Tick	$O(1)$	$O(h)$	$O(1)$
Merge	$O(n)$	$O(h(\log^2 h)(\log\log h))$	$O(1)$
Compare	$O(n)$	$O(h(\log h)(\log\log h))$	$O(1)$
Storage	$O(n)$	$O(h)$	$O(1) + O(d)$ (with resetting)

EVC Timestamps of Cuts

- Cut: is an execution prefix
- State after the events of a cut represents a *global state*
- $\downarrow e = \{f \mid f \rightarrow e \wedge f \in E\} \bigcup \{e\}$ (causal history of e)
- $S(cut)$: set that contains the last event of cut at each process
- $\widehat{cut}$: 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 $V(e_i) = \langle v_1^i, v_2^i, \cdots v_n^i \rangle$.
- For $\hat{e}_i \in \widehat{cut}$, let $V(\hat{e}_i) = \langle \hat{v}_1^i, \hat{v}_2^i, \cdots \hat{v}_n^i \rangle$.
- EVC of a cut, *cut*:

$$Enc(V(cut)) = \prod_{i=1}^{n} p_i^{\hat{v}_i^i}$$
$$= \prod_{i=1}^{n} p_i^{\max(v_i^1, v_i^2, \cdots, v_i^n)}$$

- However, we show that

$$Enc(V(cut)) = LCM(Enc(V(e_1)), Enc(V(e_2)), \cdots, Enc(V(e_n))).$$

Example: EVC Timestamp of a Cut

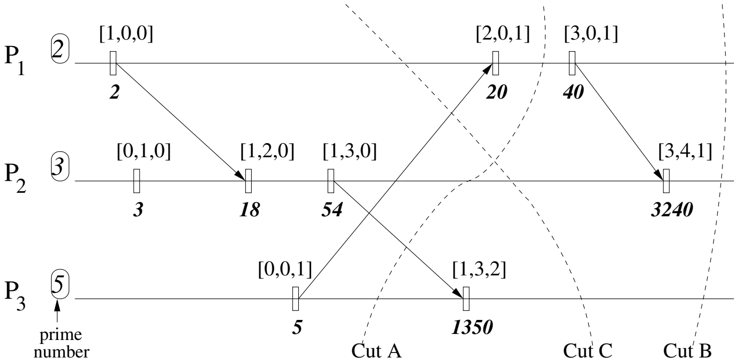

Figure: The vector timestamps and EVC timestamps are shown above and below each timeline, respectively. In real scenarios, only the EVC is stored and transmitted.

- For events $e_i \in S(CutA)$:
  - We have $Enc(V(e_1)) = 20$, $Enc(V(e_2)) = 54$, and $Enc(V(e_3)) = 5$.
  - $Enc(V(CutA)) = LCM(Enc(V(e_1)), Enc(V(e_2)), Enc(V(e_3))) = LCM(20, 54, 5) = 540$.

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 $e_i \in S(cut)$, $V(e_i) = \langle v_1^i, v_2^i, \cdots v_n^i \rangle$.
- 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

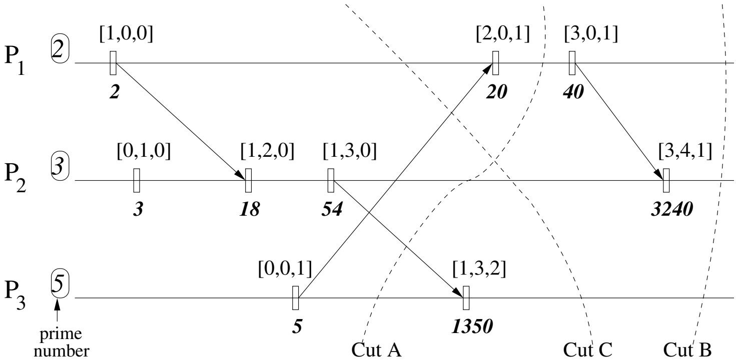

Figure: The vector timestamps and EVC timestamps are shown above and below each timeline, respectively. In real scenarios, only the EVC is stored and transmitted.

- 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 $V(cut1) = \langle v_1, v_2, \cdots, v_n \rangle$ and $V(cut2) = \langle v'_1, v'_2, \cdots, v'_n \rangle$
- We have that

$$V(cut1 \bigcap cut2) = \langle u_1, u_2, \cdots, u_n \rangle, \text{ where } u_i = \min(v_i, v'_i)$$

$$V(cut1 \bigcup cut2) = \langle u_1, u_2, \cdots, u_n \rangle, \text{ where } u_i = \max(v_i, v'_i)$$

- The encodings of $V(cut1)$, $V(cut2)$, $V(cut1 \bigcap cut2)$, $V(cut1 \bigcup cut2)$ are:

$$\begin{aligned} Enc(V(cut1)) &= p_1^{v_1} * p_2^{v_2} * \cdots * p_n^{v_n}; \\ Enc(V(cut2)) &= p_1^{v'_1} * p_2^{v'_2} * \cdots * p_n^{v'_n} \\ Enc(V(cut1 \bigcap cut2)) &= \prod_{i=1}^{n} p_i^{\min(v_i, v'_i)} \\ Enc(V(cut1 \bigcup cut2)) &= \prod_{i=1}^{n} p_i^{\max(v_i, v'_i)} \end{aligned}$$

- We show that

$$\begin{aligned} Enc(V(cut1 \bigcap cut2)) &= GCD(Enc(V(cut1)), Enc(V(cut2))) \\ Enc(V(cut1 \bigcup cut2)) &= LCM(Enc(V(cut1)), Enc(V(cut2))) \end{aligned}$$

Example: EVC Timestamp of Union and Intersection Cuts

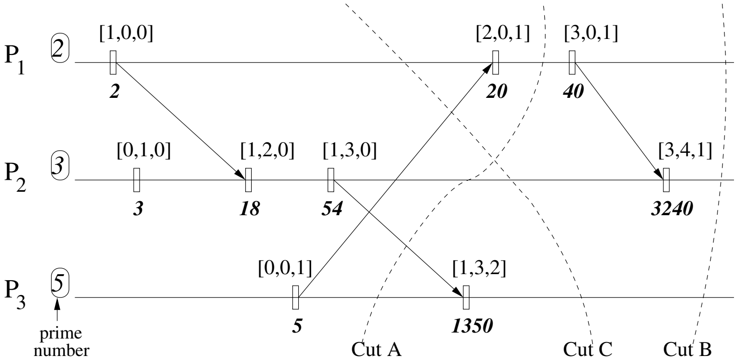

Figure: The vector timestamps and EVC timestamps are shown above and below each timeline, respectively. In real scenarios, only the EVC is stored and transmitted.

- $Enc(V(CutA)) = LCM(20, 54, 5) = 540$ and $Enc(V(CutC)) = LCM(2, 54, 1350) = 1350$.
- $Enc(V(CutA \bigcap CutC)) = GCD(Enc(V(CutA)), Enc(V(CutC))) = GCD(540, 1350) = 270$.
- $Enc(V(CutA \bigcup CutC)) = LCM(Enc(V(CutA)), Enc(V(CutC)))$
  $= LCM(540, 1350) = 2700$.

Comparison of Cuts

- Comparing $cut1$ and $cut2$:
  i) $cut1 \subset cut2$ (or symmetrically, $cut2 \subset cut1$),
  or ii) $cut1 \not\subset cut2$ and $cut2 \not\subset cut1$, i.e., $cut1 \| cut2$.
- We show:

$$i)\ Enc(V(cut1)) \prec Enc(V(cut2)) \text{ if } Enc(V(cut1)) < Enc(V(cut2)) \text{ and } Enc(V(cut2)) \bmod Enc(V(cut1)) = 0$$

$$ii)\ Enc(V(cut1)) \| Enc(V(cut2)) \text{ if } Enc(V(cut1)) \not\prec Enc(V(cut2)) \text{ and } Enc(V(cut2)) \not\prec Enc(V(cut1))$$

Correspondence between Operations on Cuts

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

Operation	Vector Clock	Encoded Vector Clock
Cut	$\forall k \in [1, n], V(cut)[k] = \max_{e_i \in S(cut)} V(e_i)[k]$ (*cut* may not be consistent) $\forall k \in [1, n], V(cut)[k] = V(e_k)[k]$ for $e_k \in S(cut)$ (*cut* is consistent)	$Enc(V(cut)) = LCM(Enc(V(e_1)), \cdots, Enc(V(e_n)))$, where $e_i \in S(cut)$
Common past	$\forall k \in [1, n], V(CP(cut))[k] = \min_{e_i \in S(cut)} V(e_i)[k]$	$Enc(V(cut)) = GCD(Enc(V(e_1)), \cdots, Enc(V(e_n)))$, where $e_i \in S(cut)$
	If $V(cut1)[j] = v_j$ and $V(cut2)[j] = v'_j$,	Merge $Enc(V(cut1))$ and $Enc(V(cut2))$ yields
Intersection	$V(cut1 \bigcap cut2)[j] = \min(v_j, v'_j)$	$Enc(V) = GCD(Enc(V(cut1)), Enc(V(cut2)))$
Union	$V(cut1 \bigcup cut2)[j] = \max(v_j, v'_j)$	$Enc(V) = LCM(Enc(V(cut1)), Enc(V(cut2)))$
Compare	$V(cut1) < V(cut2)$: $\forall j \in [1, n], V(cut1)[j] \leq V(cut2)[j]$, and $\exists j, V(cut1)[j] < V(cut2)[j]$	$Enc(V(cut1)) \prec Enc(V(cut2))$: $Enc(V(cut1)) < Enc(V(cut2))$, and $Enc(V(cut2)) \mod Enc(V(cut1)) = 0$

Complexity of Operations on Cuts

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

	Vector Clock (bounded storage) (uniform cost model)	Encoded Vector Clock (unbounded storage) (logarithmic cost model)	Encoded Vector Clock (bounded storage) (uniform cost model)
Computing timestamp	$O(n^2)$ (*cut* may not be consistent) $O(n)$ (*cut* is consistent)	$O(nh(\log^2 h)(\log\log h))$	$O(n)$
Computing common past	$O(n^2)$	$O(nh(\log^2 h)(\log\log h))$	$O(n)$
Intersection and union	$O(n)$	$O(h(\log^2 h)(\log\log h))$	$O(1)$
Compare	$O(n)$	$O(h(\log h)(\log\log h))$	$O(1)$

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!