Naiad: A Timely Dataflow System

Derek G Murray et al.

24^{th} ACM Symposium on Operating System Principles, 2013

Presentation by:
SB Ramalingam Santhanakrishnan
K Kleeberger
B Jain

Agenda

- Introduction
- Timely Data Flow

- Distributed Implementation
- Programming Model

- Performance Evaluation
- Real World Applications

Introduction - The Problem

Batch

- Synchronous iteration
- Good consistency
- Poor latency

Stream

- Low latency
- Weak consistency
- Difficult to iterate

Graph

- Supports iteration
- Poor latency

TUDelft

Introduction - The Problem

Batch

- Synchronous iteration
- Good consistency
- Poor latency

Stream

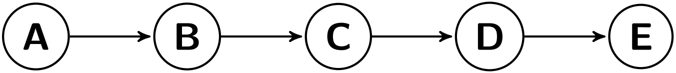
- Low latency
- Weak consistency
- Difficult to iterate

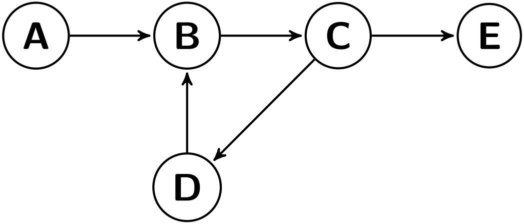
Graph

- Supports iteration
- Poor latency

What if we want all of this?

Dataflow programming





Least common denominator is *Dataflow*

Timely Dataflow

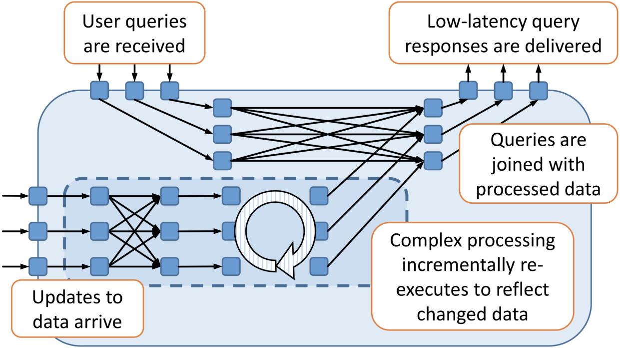
A general system to support high-level constructs using Dataflow

- Structured Loops
- Stateful vertices
- Notification for vertices on iteration completion

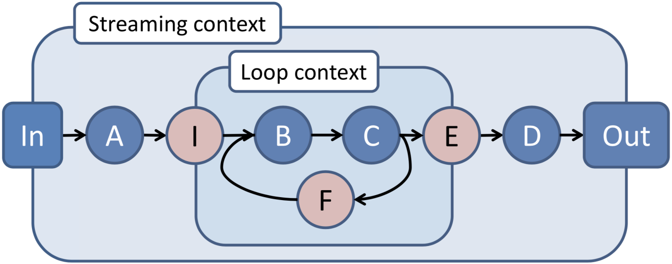
Timely Dataflow

A general system to support high-level constructs using Dataflow

- Structured Loops
- Stateful vertices
- Notification for vertices on iteration completion

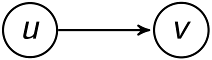


Timely Dataflow - Graph Structure



- Time epoch on every input
- Streaming Context - Process data and pass
- Loop Context
  - Loop - Ingress (I) $\Rightarrow$ Feedback (F) $\Rightarrow$ Egress (E)
  - Monitors progress

Timely Dataflow - Concurrency Primitives



Vertices register callbacks

v.OnRecv(e: Edge, m: Message, t: Timestamp)
v.OnNotify(e: Edge, m: Message, t: Timestamp)

Vertices can notify others (coordination)

this.SendBy(e: Edge, m: Message, t: Timestamp)
this.NotifyAt(t: Timestamp)

Timely Dataflow - Concurrency Primitives contd.

$$u \longrightarrow v$$

- $\textsc{OnRecv}$ and $\textsc{OnNotify}$ are queued, no strict ordering
- Guarantee:
  $v.\textsc{OnNotify}(t)$ is invoked only after no further invocations of $v.\textsc{OnRecv}(e, m, t')$, for $t' \le t$
- Constraint:
  $v.\textsc{SendBy}(t')$ and $v.\textsc{NotifyAt}(t')$
  such that $t' \ge t$

Timely Dataflow - Timestamp

$$(e \in \mathbb{N}, \langle c_1, \ldots, c_k \rangle \in \mathbb{N}^k)$$

Example: (epoch, counter) - $(1, \langle 0, 1, 2 \rangle)$

Vertex Behavior:

- Ingress - $\langle c_1, \ldots, c_k \rangle \Rightarrow \langle c_1, \ldots, c_k, 0 \rangle$
- Egress - $\langle c_1, \ldots, c_k, c_{k+1} \rangle \Rightarrow \langle c_1, \ldots, c_k \rangle$
- Feedback - $\langle c_1, \ldots, c_k \rangle \Rightarrow \langle c_1, \ldots, c_k, c_{k+1} \rangle$

Ordering:
$t_1 = (e_1, \vec{c_1}), t_2 = (e_2, \vec{c_2})$
$t_1 < t_2 \iff e_1 < e_2$ and $\vec{c_1} < \vec{c_2}$

Timely Dataflow - Progress Tracking

Future timestamps constrained by,

- Unprocessed *events* (SendBy and NotifyAt)
- Graph structure

Timely Dataflow - Progress Tracking

Future timestamps constrained by,

- Unprocessed *events* ($\textsc{SendBy}$ and $\textsc{NotifyAt}$)
- Graph structure

Pointstamp: $(t \in \textbf{Timestamp}, \; l \in \textbf{Edge} \cup \textbf{Vertex}\;)$

TUDelft

Timely Dataflow - Progress Tracking

Future timestamps constrained by,

- Unprocessed *events* ($\textsc{SendBy}$ and $\textsc{NotifyAt}$)
- Graph structure

Pointstamp: $(t \in$ **Timestamp,** $l \in$ **Edge** $\cup$ **Vertex** $)$

for, $v.\textsc{SendBy}(e, m, t)$, pointstamp$(m) = (t, e)$

for, $v.\textsc{NotifyAt}(t)$, pointstamp$(m) = (t, v)$

TUDelft

Timely Dataflow - Progress Tracking

Future timestamps constrained by,

- Unprocessed *events* ($\textsc{SendBy}$ and $\textsc{NotifyAt}$)
- Graph structure

Pointstamp: $(t \in$ **Timestamp,** $l \in$ **Edge** $\cup$ **Vertex** $)$

for, $v.\textsc{SendBy}(e, m, t)$, pointstamp$(m) = (t, e)$

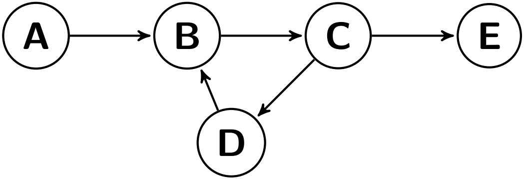
for, $v.\textsc{NotifyAt}(t)$, pointstamp$(m) = (t, v)$

Structure constraint induces ordering:

(t_1, l_1) *could-result-in* $(t_2, l_2) \iff \exists$ path $\psi = \langle l_1, \ldots, l_2 \rangle$

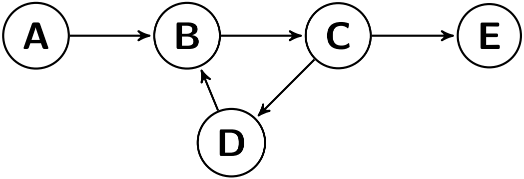
such that t_1 is adjusted by each I, E or F, satisfies $\psi(t_1) \leq t_2$.

Timely Dataflow - could-result-in



Is there a path between D and E?

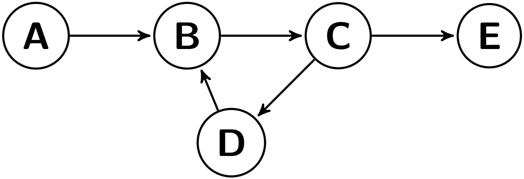
Timely Dataflow - could-result-in



Is there a path between D and E?

$(1, A)$ *could-result-in* $(1, E)$?

Timely Dataflow - could-result-in

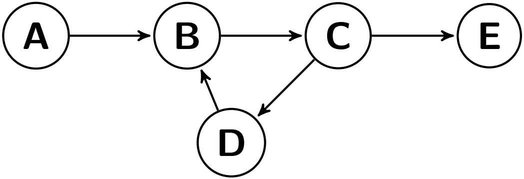


Is there a path between D and E?

$(1, A)$ *could-result-in* $(1, E)$?

$((1,2), D)$ *could-result-in* $((1,4), C)$?

Timely Dataflow - could-result-in



Is there a path between D and E?

$(1, A)$ *could-result-in* $(1, E)$?

$((1, 2), D)$ *could-result-in* $((1, 4), C)$?

$((3, 4), D)$ *could-result-in* $(2, E)$?

Timely Dataflow - Single-Threaded Scheduler

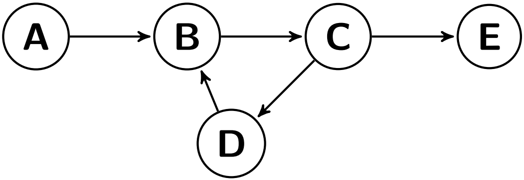
- A set of *active pointstamps* (at least 1 unprocessed event).
- For each active poinstamp, maintains,
  - *occurrence count* - outstanding events.
  - *precursor count* - how many active poinstamps precede.
- Update *occurrence count* for each event.

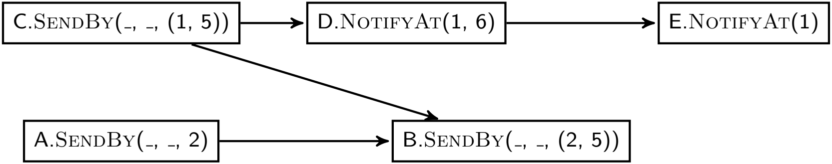
Timely Dataflow - Single-Threaded Scheduler

- A set of *active pointstamps* (at least 1 unprocessed event).
- For each active poinstamp, maintains,
  - *occurrence count* - outstanding events.
  - *precursor count* - how many active poinstamps precede.
- Update *occurrence count* for each event.

Scheduler is a simple message sorting function,
to deliver notifications

Visualizing the scheduler





Thank you

SB Ramalingam Santhanakrishnan

K Kleeberger

B Jain