

Naiad: A Timely Dataflow System

Derek G Murray et al.

24th 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

Introduction - The Problem

Batch

- Synchronous iteration
- Good consistency
- Poor latency

Stream

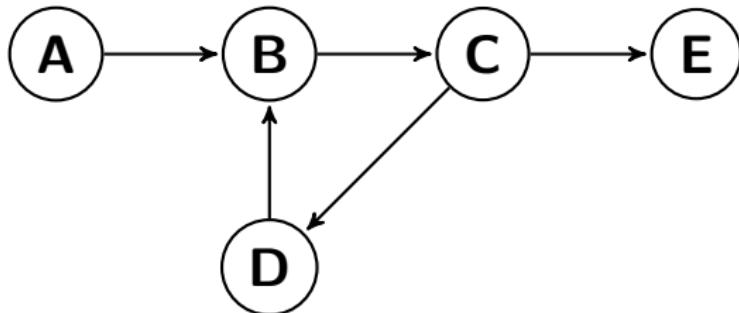
- Low latency
- Weak consistency
- Difficult to iterate

Graph

- Supports iteration
- Poor latency

What if we want
all of this?

Dataflow programming



Least common denominator is *Dataflow*

Timely Dataflow

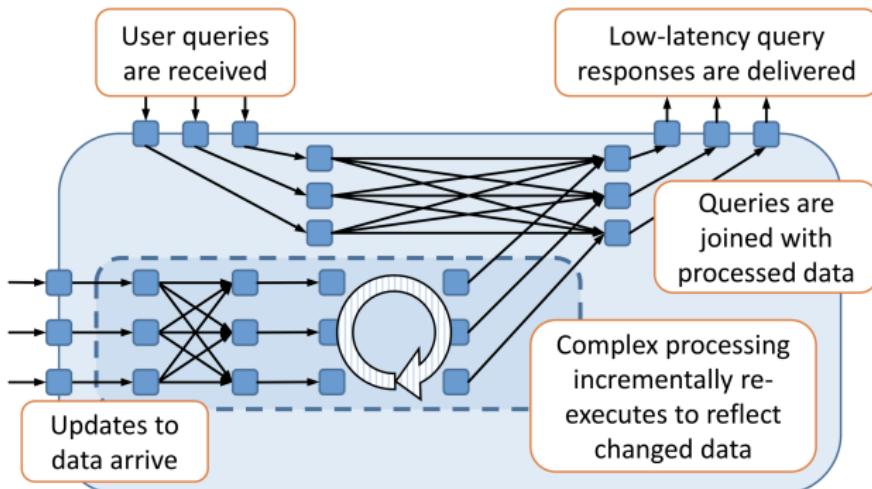
A general system to support high-level constructs using Dataflow

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

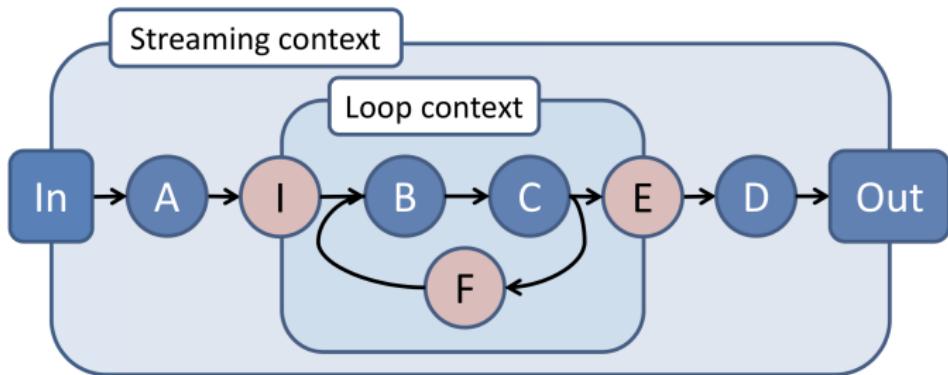
Timely Dataflow

A general system to support high-level constructs using Dataflow

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

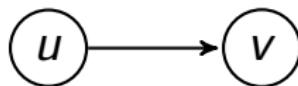


Timely Dataflow - Graph Structure



- 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

- $v.\text{ONRECV}(e: \text{Edge}, m: \text{Message}, t: \text{Timestamp})$
- $v.\text{ONNOTIFY}(e: \text{Edge}, m: \text{Message}, t: \text{Timestamp})$

Vertices can notify others (coordination)

- $this.\text{SENDBY}(e: \text{Edge}, m: \text{Message}, t: \text{Timestamp})$
- $this.\text{NOTIFYAT}(t: \text{Timestamp})$

Timely Dataflow - Concurrency Primitives contd.



- ONRECV and ONNOTIFY are queued, no strict ordering
- Guarantee:
 $v.\text{ONNOTIFY}(t)$ is invoked only after no further invocations of $v.\text{ONRECV}(e, m, t')$, for $t' \leq t$
- Constraint:
 $v.\text{SENDBY}(t')$ and $v.\text{NOTIFYAT}(t')$
such that $t' \geq t$

Timely Dataflow - Timestamp

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

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

Vertex Behavior:

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

Ordering:

$$t_1 = (e_1, \vec{c}_1), t_2 = (e_2, \vec{c}_2)$$

$$t_1 < t_2 \iff e_1 < e_2 \text{ and } \vec{c}_1 < \vec{c}_2$$

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 (SENDBY and NOTIFYAT)
- Graph structure

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

Timely Dataflow - Progress Tracking

Future timestamps constrained by,

- Unprocessed events (SENDBY and NOTIFYAT)
- Graph structure

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

for, $v.\text{SENDBY}(e, m, t)$, $\text{pointstamp}(m) = (t, e)$

for, $v.\text{NOTIFYAT}(t)$, $\text{pointstamp}(m) = (t, v)$

Timely Dataflow - Progress Tracking

Future timestamps constrained by,

- Unprocessed events (SENDBY and NOTIFYAT)
- Graph structure

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

for, $v.\text{SENDBY}(e, m, t)$, $\text{pointstamp}(m) = (t, e)$

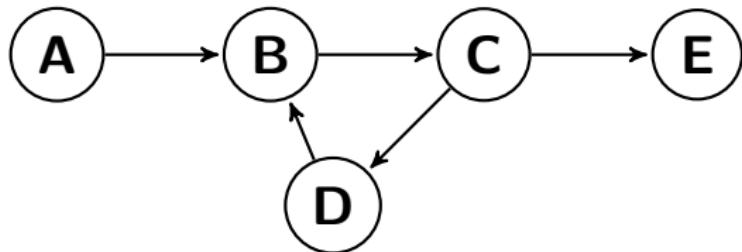
for, $v.\text{NOTIFYAT}(t)$, $\text{pointstamp}(m) = (t, v)$

Structure constraint induces ordering:

(t_1, l_1) could-result-in $(t_2, l_2) \iff \exists \text{ path } \psi = \langle l_1, \dots, l_2 \rangle$

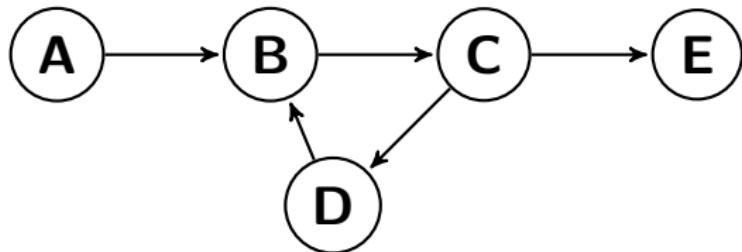
such that t_1 is adjusted by each I, E or F, satisfies $\psi(t_1) \leq t_2$.

Timely Dataflow - could-result-in



Is there a path between D and E?

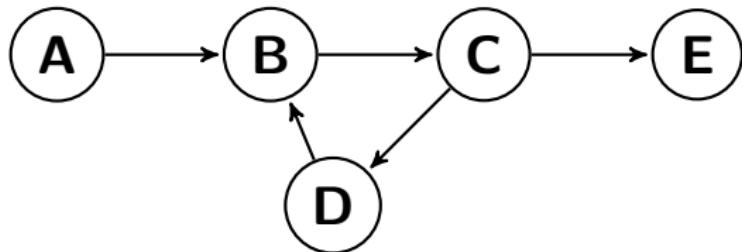
Timely Dataflow - could-result-in



Is there a path between D and E?

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

Timely Dataflow - could-result-in

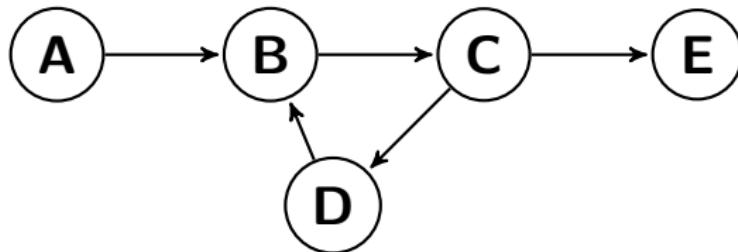


Is there a path between D and E?

$(1, A)$ 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

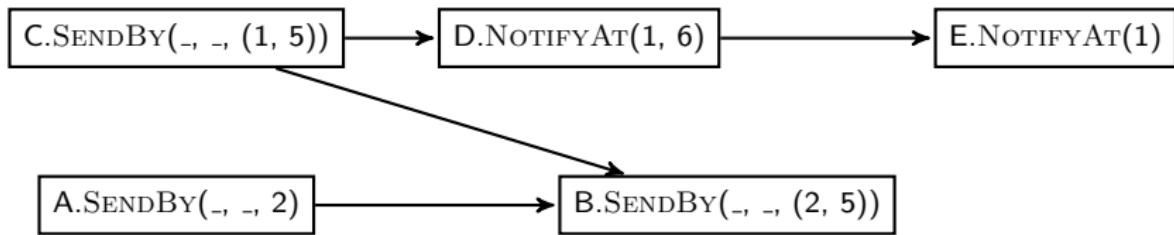
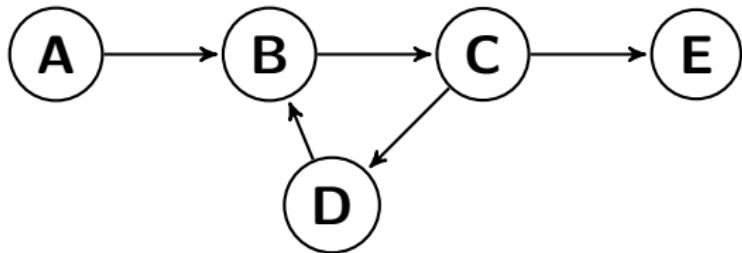
- A set of *active pointstamps* (at least 1 unprocessed event).
- For each active pointstamp, maintains,
 - *occurrence count* - outstanding events.
 - *precursor count* - how many active pointstamps precede.
- Update *occurrence count* for each event.

Timely Dataflow - Single-Threaded Scheduler

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

Scheduler is a simple message sorting function,
to deliver notifications

Visualizing the scheduler



Hi, this is Konrad Kleeberger!



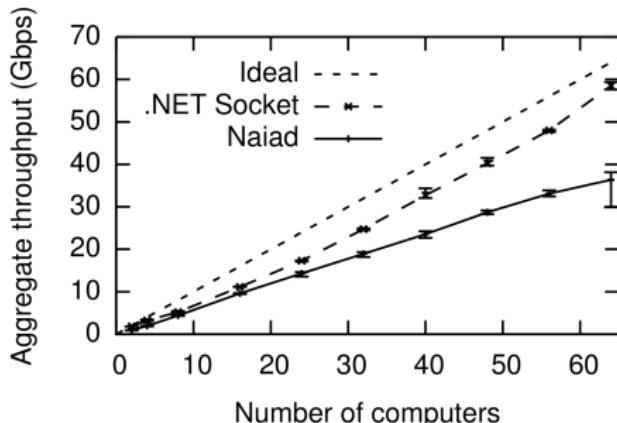
Hi, this is Konrad Kleeberger again!



Performance Evaluation

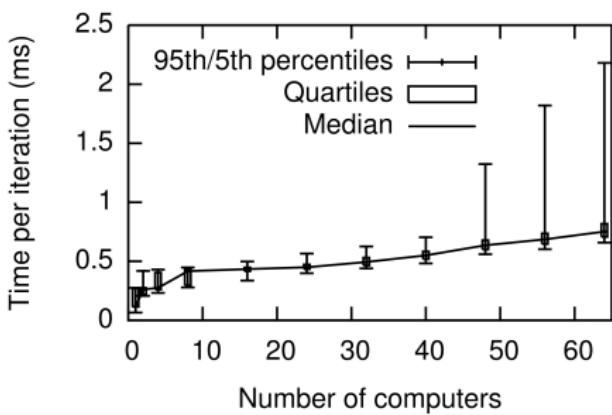
Performance Evaluation - Throughput

- Cyclic dataflow that repeatedly performs all-to-all data exchange of a fixed number of records.
- Ideal aggregate throughput based on Ethernet Bandwidth.
- .NET Socket demonstrates achievable throughput given network topology, TCP overheads and .NET API costs.
- Naiad throughput scales linearly.



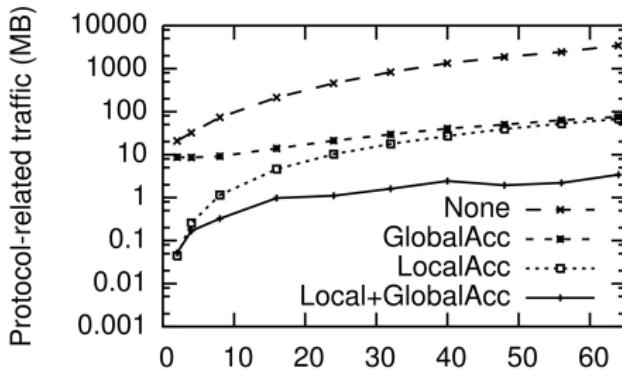
Performance Evaluation - Latency

- Minimal time required for Global Coordination.
- Request and receive completeness notifications.
- Median time per iteration small at $753\mu\text{s}$ for 64 computers.
- 95th percentile results - adverse impact of micro-stragglers.



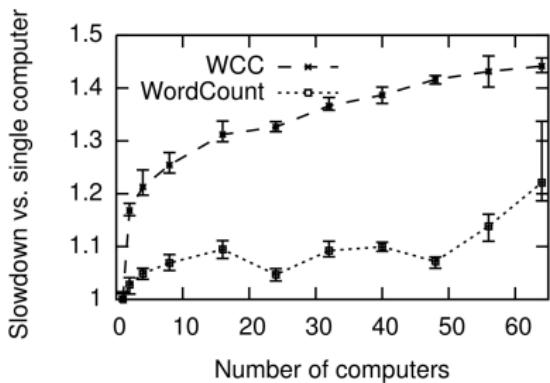
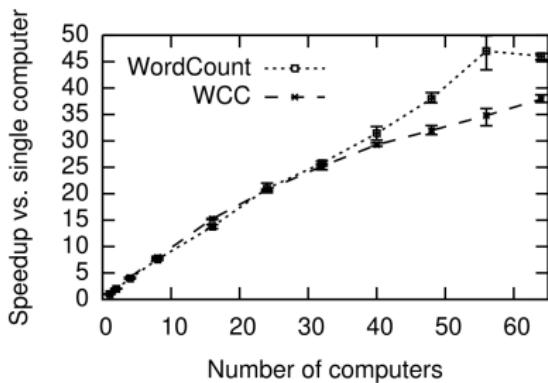
Performance Evaluation - Protocol Optimizations

- Weakly Connected Components Computation (WCC) on random graph of 300 edges (2.2GB of raw input).
- Reduction in volume of protocol traffic depending on level of accumulation.
- In practice, reduction in messages from local accumulation is sufficient.



Performance Evaluation - Scaling

- Strong scaling Adding compute resources with fix input size.
- Weak scaling Adding compute resources and increasing input.
- Word Count 128GB uncompressed corpus.
- WCC Random Graph of 200M edges.



Real World Applications

Real World Applications - Batch Iterative Graph Computation

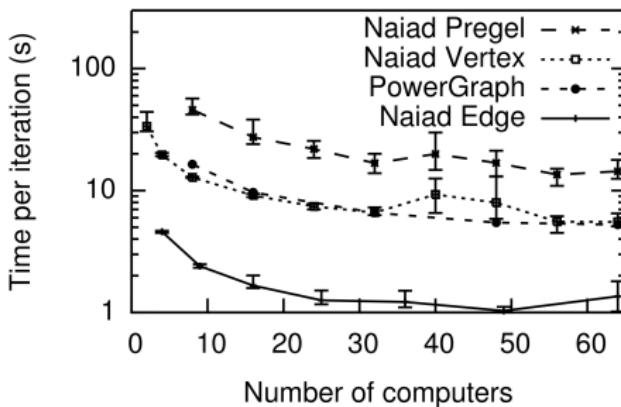
- Graph Computation on Large scale real world datasets.
- PDW – Distributed Database
- DryadLINQ – General Purpose Batch Processor
- SHS – Purpose Built Distributed Graph Store
- Speedups demonstrate the power of being able to maintain application-specific state in memory between iterations.

Algorithm	PDW	DryadLINQ	SHS	Naiad
PageRank	156,982	68,791	836,455	4,656
SCC	7,306	6,294	15,903	729
WCC	214,479	160,168	26,210	268
ASP	671,142	749,016	2,381,278	1,131

Running time in seconds

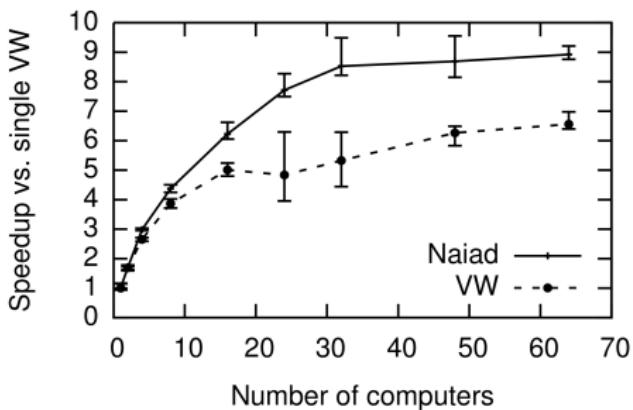
Batch Iterative Graph Computation

- Several systems have adopted the computation of PageRank on a Twitter follower graph as a standard benchmark.
- Naiad Vertex – Partitions edges by source vertex.
- Naiad Edge – Partitions edges using a space-filling curve.



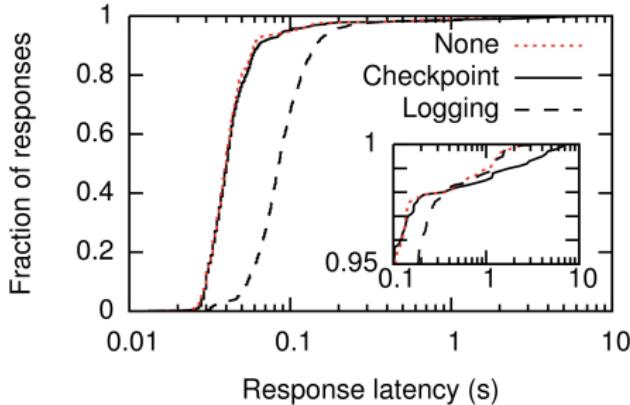
Batch Iterative Machine Learning

- Vowpal Wabbit (VW) – open-source distributed machine learning library
- Modified VW –
first and second phases run inside a Naiad vertex,
third phase uses Naiad implementation of the AllReduce.



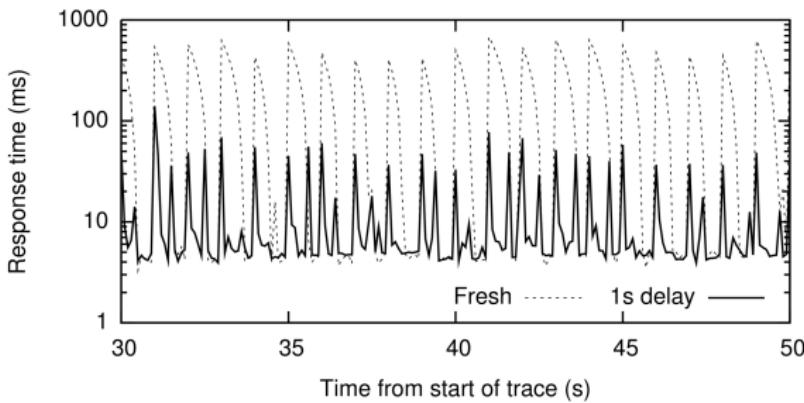
Streaming Acyclic Computation

- Kineograph ingests arriving graph data, takes regular snapshots for data parallel computation, and produces consistent results.
- The system is partitioned into ingest nodes and compute nodes.
- k-exposure metric for identifying controversial topics on Twitter.



Streaming Iterative Graph Analytics

- There are two input stages: stream of tweets and requests, specified by a user name and query identifier.
- "Fresh" – queries being delayed behind tweet processing
- "1s delay" – the benefit of querying stale but consistent data.
- Naiads support for overlapped computation by trading off responsiveness for staleness.



Thank you

SB Ramalingam Santhanakrishnan

K Kleeberger

B Jain