Naiad: A Timely Dataflow System

Phil Gibbons

15-712 F15

Lecture 13

Today's Reminders

- **Discuss Project Ideas with Phil & Kevin**
  - Office Hours: Kevin on Tues, Phil on Wed
  - Sign up for a slot: 11-12:30 or 3-4:20 this Friday

Naiad: A Timely Dataflow System
[SOSP'13 best paper]

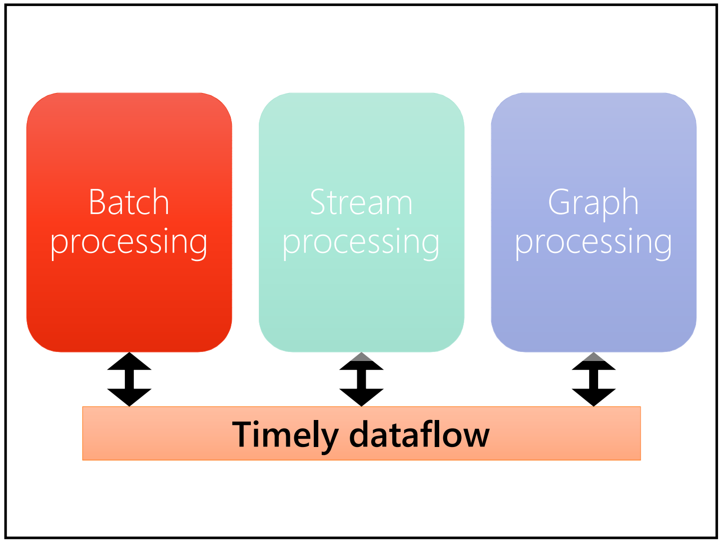
- **Derek Murray** (Google)
- **Frank McSherry** (Free agent*)
- **Rebecca Isaacs** (Google)
- **Michael Isard** (Free agent)
- **Paul Barham** (Google)
- **Martin Abadi** (Google)

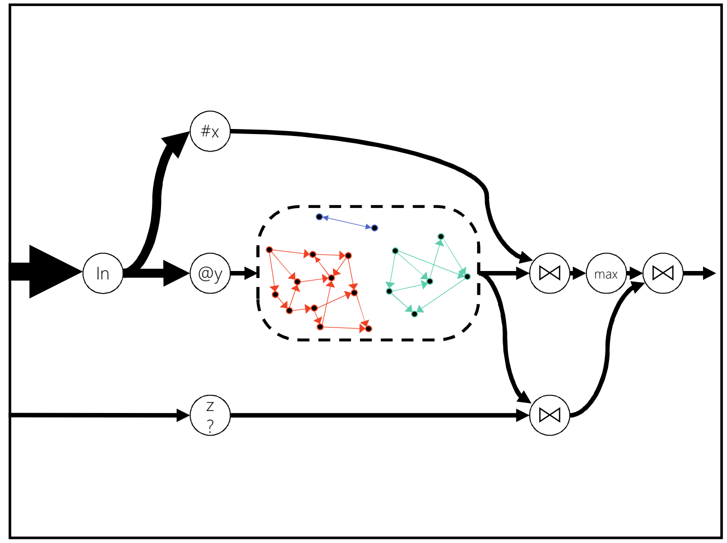
* "Doing less of what I used to do, and more of what I'd rather be doing."

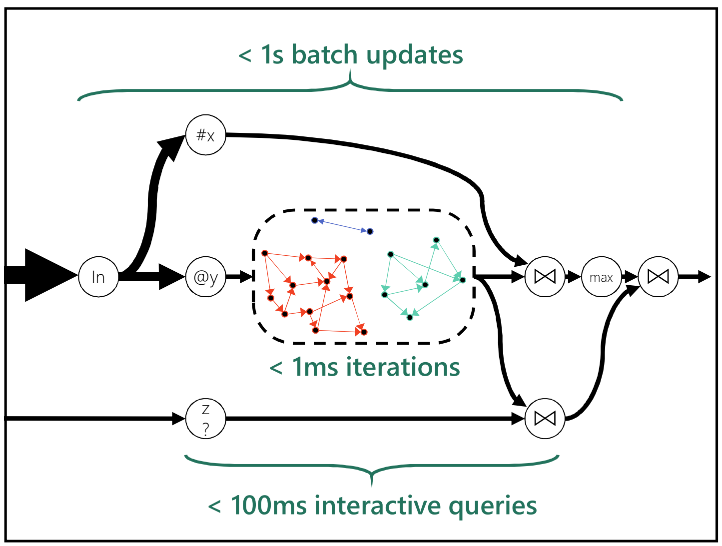
[Slides from SOSP'13 talk]

Naiad: A Timely Dataflow System

Derek G. Murray Frank McSherry Rebecca Isaacs
Michael Isard Paul Barham Martín Abadi
Microsoft Research







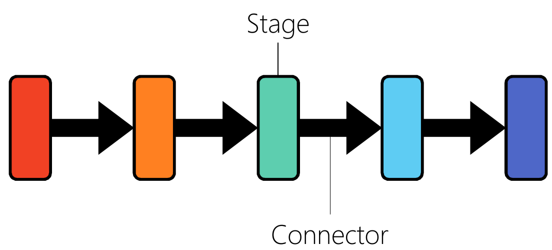
Outline

Revisiting dataflow

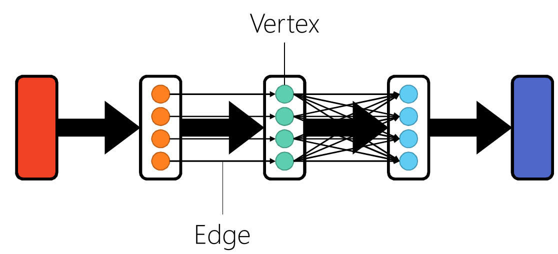
How to achieve low latency

Evaluation

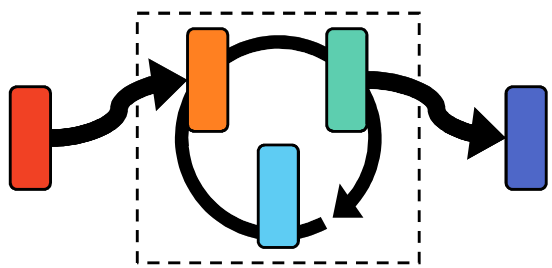
Dataflow



Dataflow: parallelism

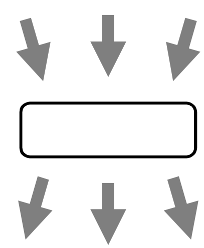


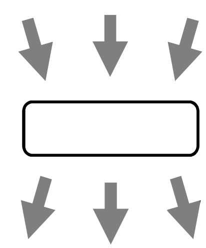
Dataflow: iteration



Batching vs. Streaming

(synchronous) | (asynchronous)

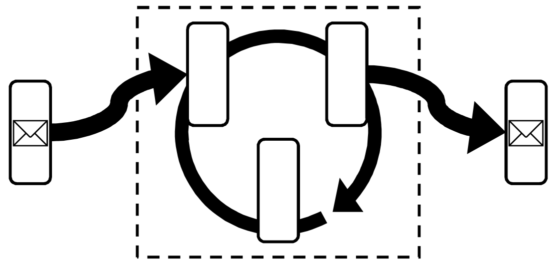




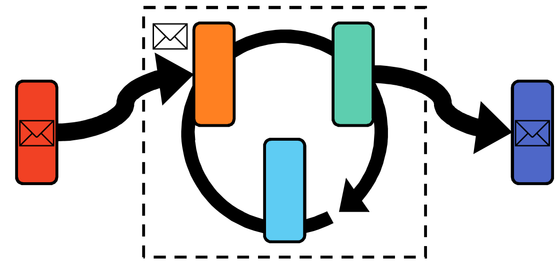
- × Requires coordination
- ✓ Supports aggregation

- ✓ No coordination needed
- × Aggregation is difficult

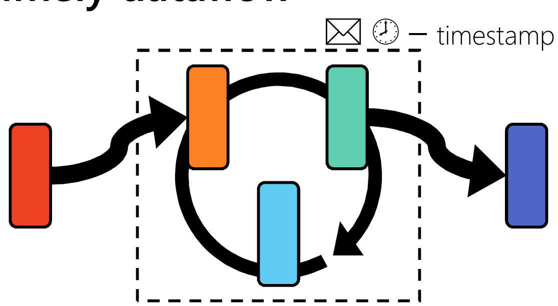
Batch iteration



Streaming iteration



Timely dataflow



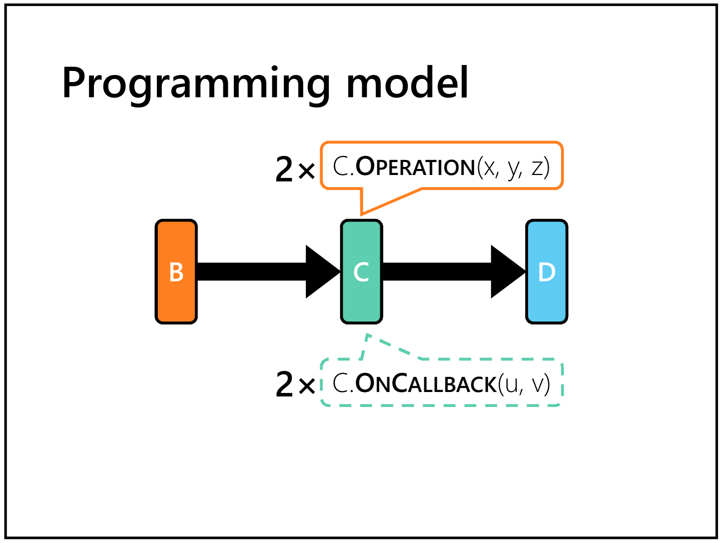
Supports **asynchronous** and **fine-grained synchronous** execution

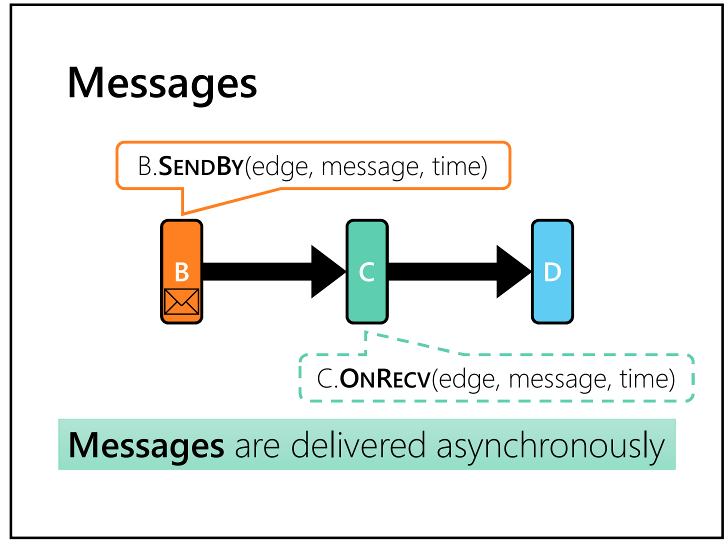
How to achieve low latency

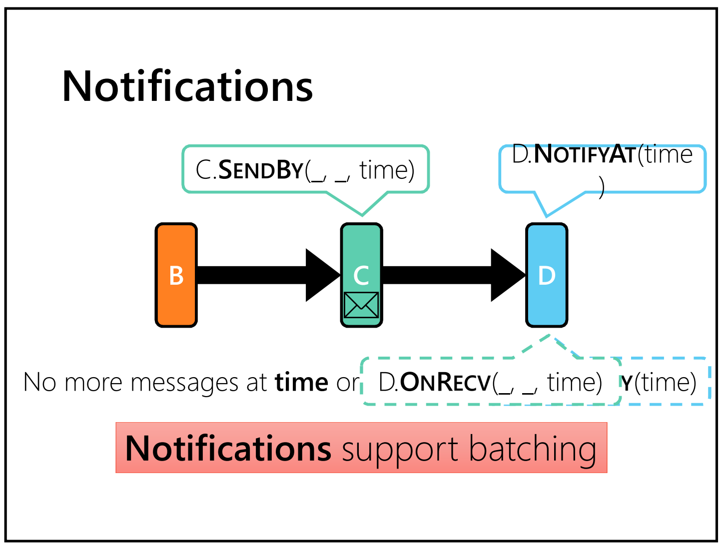
Programming model

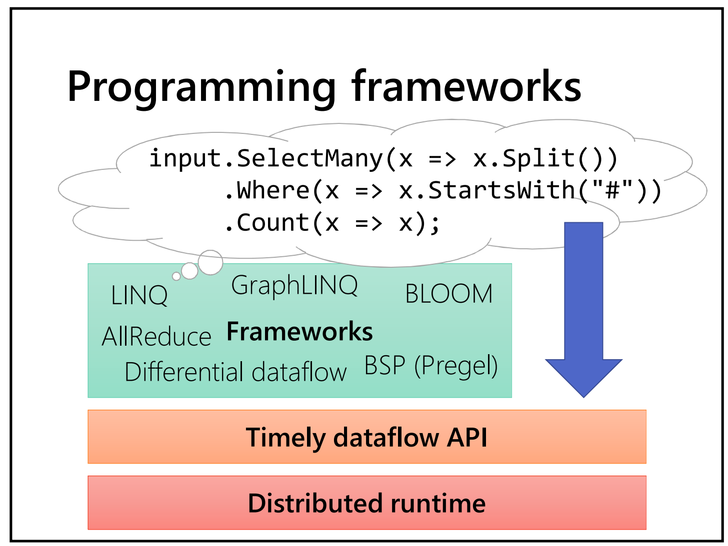
Distributed progress tracking protocol

System performance engineering


Programming model
2 ×
C.OPERATION(x, y, z)
B
C
D
2 ×
C.ONCALLBACK(u, v)


Messages
B.SENDBY(edge, message, time)
B
C
D
C.ONRECV(edge, message, time)
Messages are delivered asynchronously


Notifications
C.SENDBY(_, _, time)
D.NOTIFYAT(time)
B
C
D
No more messages at time or
D.ONRECV(_, _, time)
Y(time)
Notifications support batching


Programming frameworks
input.SelectMany(x => x.Split())
.Where(x => x.StartsWith("#"))
.Count(x => x);
LINQ
GraphLINQ
BLOOM
AllReduce
Frameworks
Differential dataflow
BSP (Pregel)
Timely dataflow API
Distributed runtime

How to achieve low latency

Programming model

Asynchronous and fine-grained synchronous execution

Distributed progress tracking protocol

System performance engineering

How to achieve low latency

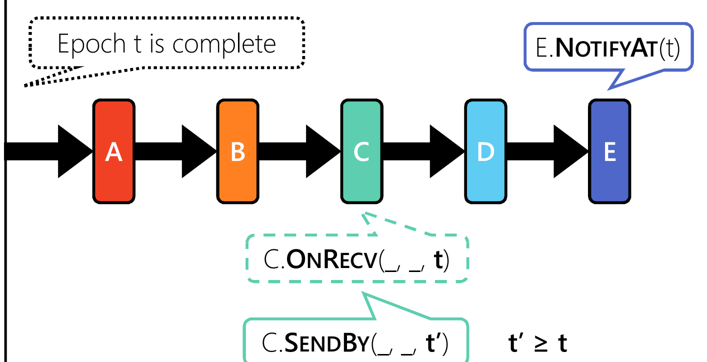
Programming model

Asynchronous and fine-grained synchronous execution

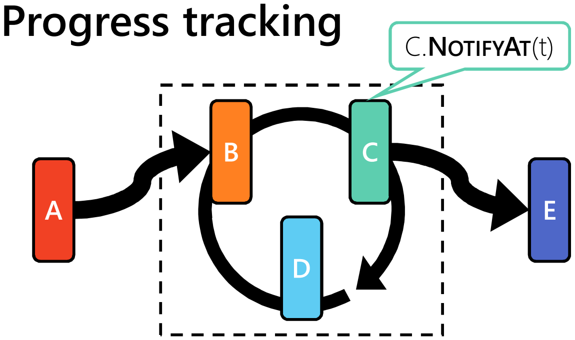
Distributed progress tracking protocol

System performance engineering

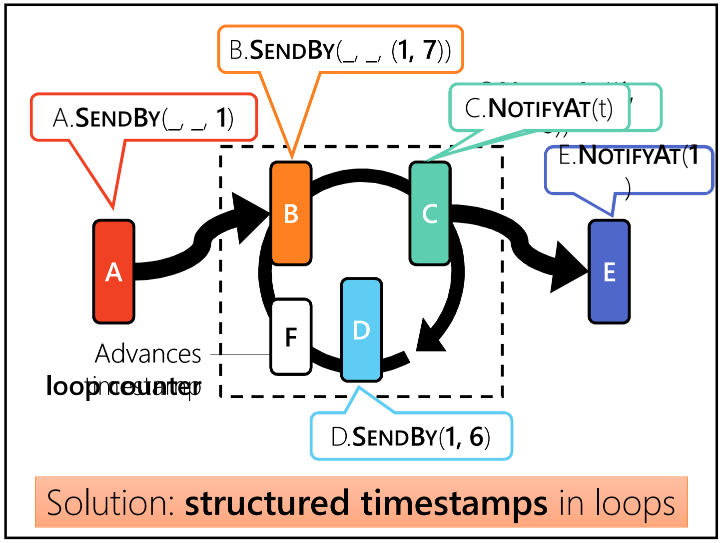
Progress tracking



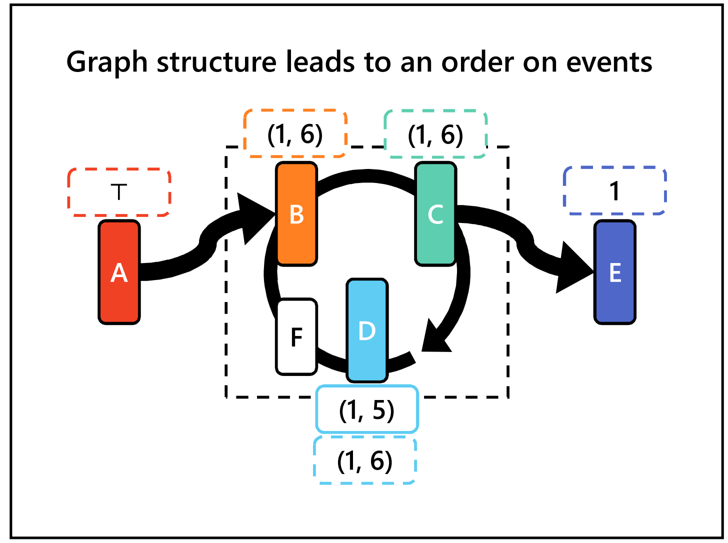
Progress tracking



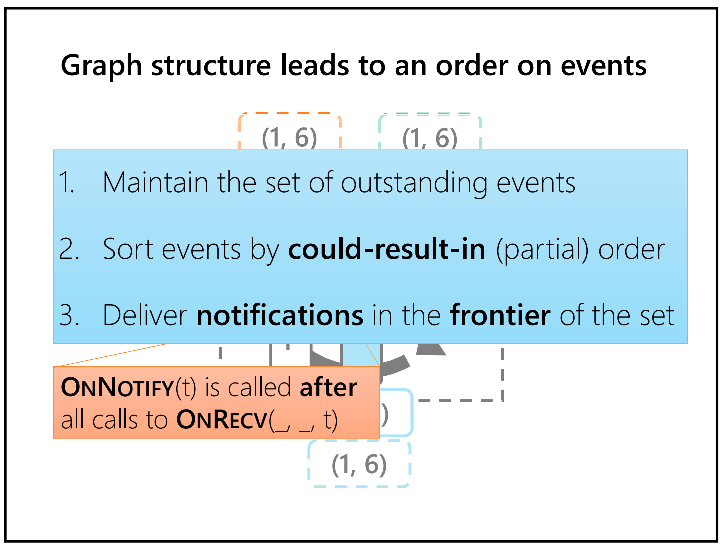
Problem: **C** depends on its own output



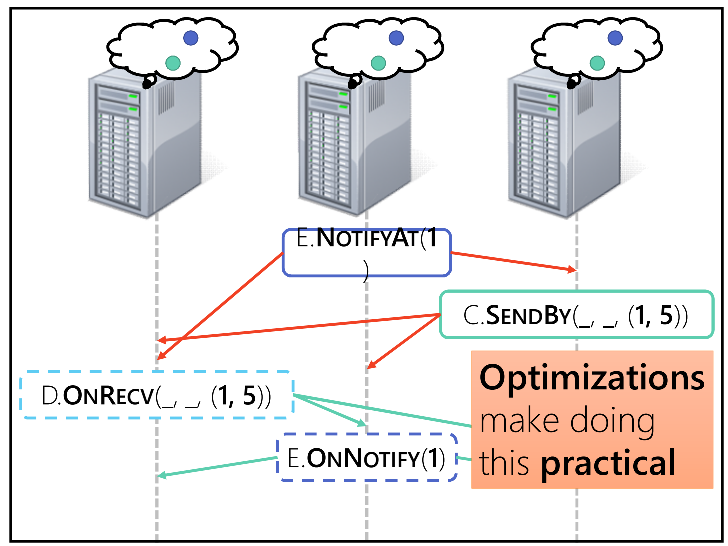
B.SendBy(_, _, (1, 7))
A.SendBy(_, _, 1)
C.NotifyAt(t)
E.NotifyAt(1)
A
B
C
D
E
F
Advances
loop counter
D.SendBy(1, 6)
Solution: **structured timestamps** in loops



Graph structure leads to an order on events
(1, 6)
(1, 6)
T
1
A
B
C
D
E
F
(1, 5)
(1, 6)



Graph structure leads to an order on events
(1, 6)
(1, 6)
1. Maintain the set of outstanding events
2. Sort events by **could-result-in** (partial) order
3. Deliver **notifications** in the **frontier** of the set
OnNotify(t) is called **after**
all calls to OnRecv(_, _, t)
(1, 6)



E.NotifyAt(1)
C.SendBy(_, _, (1, 5))
D.OnRecv(_, _, (1, 5))
E.OnNotify(1)
Optimizations
make doing
this **practical**

How to achieve low latency

Programming model

Asynchronous and fine-grained synchronous execution

Distributed progress tracking protocol

Enables processes to deliver notifications promptly

System performance engineering

How to achieve low latency

Programming model

Asynchronous and fine-grained synchronous execution

Distributed progress tracking protocol

Enables processes to deliver notifications promptly

System performance engineering

Performance engineering

Microstragglers are the primary challenge

Garbage collection	O(1–10 s)
TCP timeouts	O(10–100 ms)
Data structure contention	O(1 ms)

For detail on how we handled these, see paper (Sec. 3)

How to achieve low latency

Programming model

Asynchronous and fine-grained synchronous execution

Distributed progress tracking protocol

Enables processes to deliver notifications promptly

System performance engineering

Mitigates the effect of microstragglers

Outline

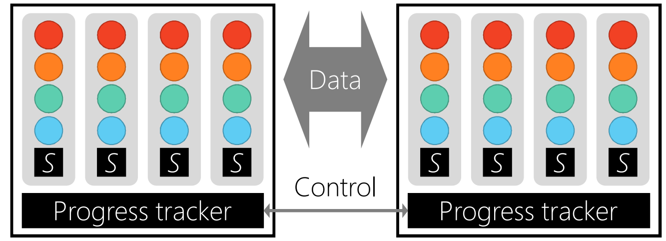
Revisiting dataflow

How to achieve low latency

Evaluation

System design

64 × 8-core 2.1 GHz AMD Opteron
16 GB RAM per server
Gigabit Ethernet

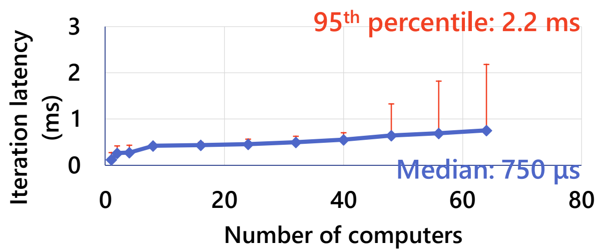


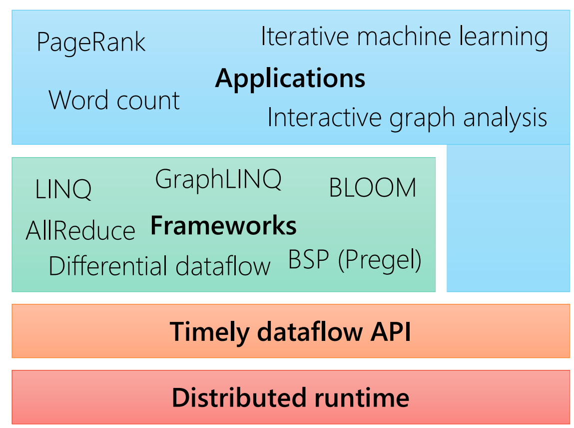
Limitation:

Fault tolerance via checkpointing/logging (see paper)

Iteration latency

64 × 8-core 2.1 GHz AMD Opteron
16 GB RAM per server
Gigabit Ethernet

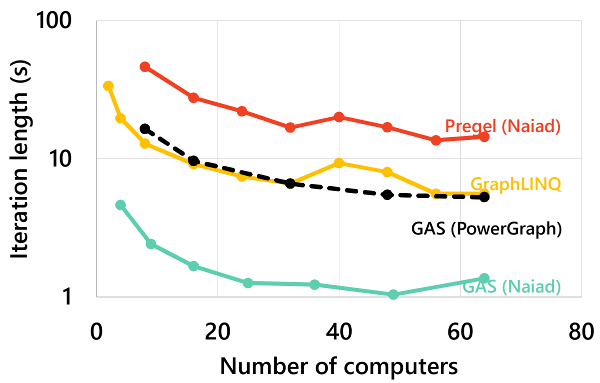




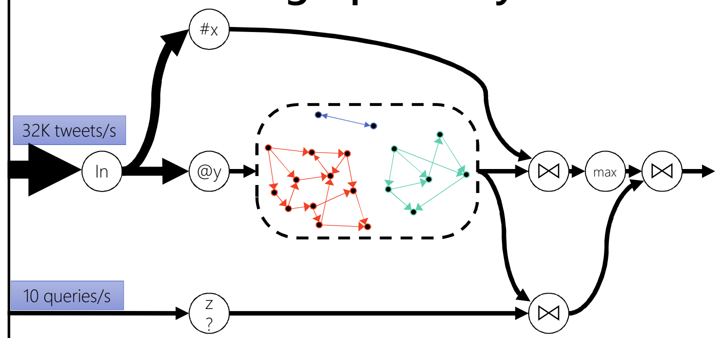
PageRank

Twitter graph
42 million nodes
1.5 billion edges

64 × 8-core 2.1 GHz AMD Opteron
16 GB RAM per server
Gigabit Ethernet

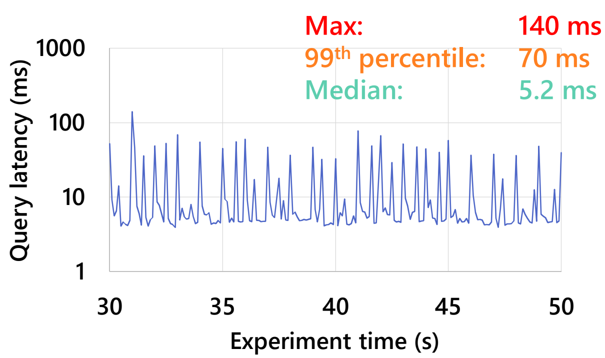


Interactive graph analysis



Query latency

32 × 8-core 2.1 GHz AMD Opteron
16 GB RAM per server
Gigabit Ethernet



Conclusions

Low-latency distributed computation enables Naiad to:

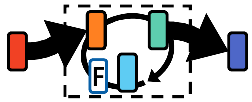
- achieve the **performance** of specialized frameworks
- provide the **flexibility** of a generic framework

The **timely dataflow** API enables **parallel innovation**

Now available for download:
http://github.com/MicrosoftResearchSVC/naiad/

[End of slides from SOSP'13 talk]

Could-Result-In



- **Timestamp**
  - (epoch, sequence of loop counters)
- **Pointstamp**
  - (timestamp, location), location is either edge or vertex
- **Path Summary for l_1 to l_2**
  - Loop coordinates that its vertices remove, add, increment
  - $\Psi[l_1, l_2]$ transforms a timestamp at l_1 to a timestamp at l_2
- **Pointstamp (t_1, l_1) could-result-in pointstamp (t_2, l_2)**
  - $\Psi[l_1, l_2](t_1) \leq t_2$
  - OK to notify pointstamp p iff no p' could-result-in p

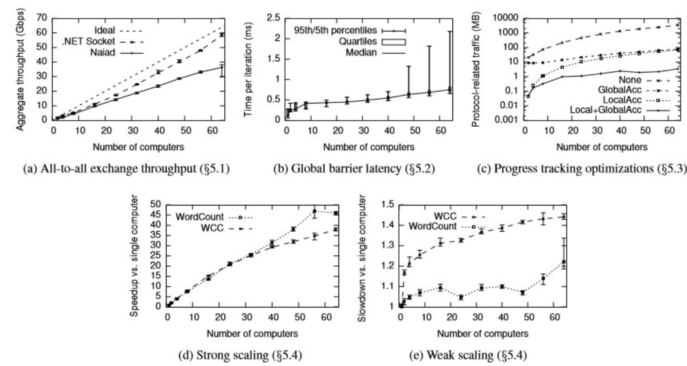
Distributed Implementation

- **Workers communicate using shared queues**
  - Have no other shared state
  - Single thread of control within a vertex
- **Optimizing broadcast updates**
  - Track progress on per stage/connector-basis not vertex/edge
  - Accumulate updates in buffer by summing their deltas
  - Accumulate at process level & cluster level
  - Optimistic UDP update packup
  - Wake workers in parallel

Micro-straggler Mitigation

- **Networking**
  - Disable Nagle's algorithm; Reduce delayed ack timeout to 10ms; Reduce min transmission time from 300ms to 20ms
  - Future: Try Datacenter TCP & RDMA over InfiniBand
- **Data structure contention (in .NET concurrent queues)**
  - Reduce clock granularity to 1 ms
- **Garbage collection**
  - Trigger GC less frequently; Avoid object allocation; Use value types (single pointer)

Microbenchmark Performance



(a) All-to-all exchange throughput (§5.1) (b) Global barrier latency (§5.2) (c) Progress tracking optimizations (§5.3)

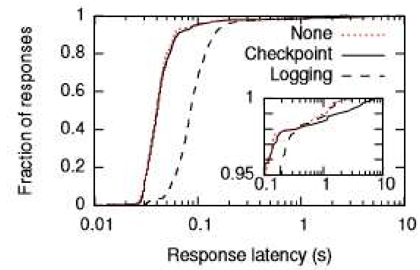
(d) Strong scaling (§5.4) (e) Weak scaling (§5.4)

Graph Algorithms

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**

Table 1: Running times in seconds of several graph algorithms on the Category A web graph. Non-Naiad measurements are due to Najork *et al.* [34].

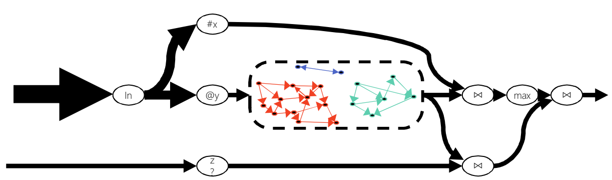
Streaming Acyclic Computation

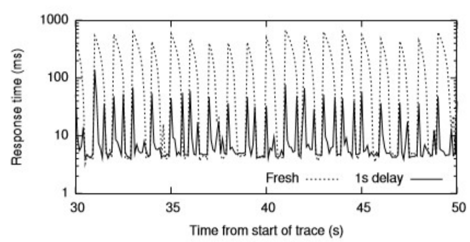


(c) k-Exposure response time (§6.3)

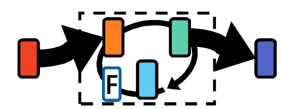
Naiad: 482K tweets/s (None); 273K (Logging)
Kineograph: 185K tweets/s in 90 s avg latency; reduced ingestion rate in 10 s avg latency

Streaming Iterative Graph Analytics



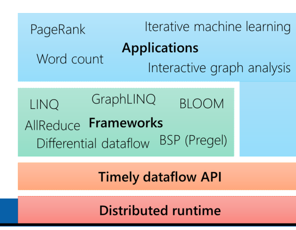


Naiad



- **Timely Dataflow**
  - Structured loops
  - Stateful vertices consume/produce records w/o coordination
  - Notifications for vertices when input or iteration is done

"We believe that separating systems design into a common platform component and a family of libraries or domain-specific languages is good for both users and researchers."



Wednesday's Paper

Spanner: Google's Globally-Distributed Database
James Corbett, Jeffrey Dean, Michael Epstein, et al.

OSDI'12 best paper